TU WIEN Informatics

Decision Procedures for Separation Logic: Beyond Symbolic Heaps

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktor der technischen Wissenschaften

by

Jens Pagel
Registration Number 01528298

to the Faculty of Informatics
at the TU Wien

Advisor: Prof. Florian Zuleger
Second Advisor: Prof. Georg Weissenbacher

The dissertation has been reviewed by:

Radu Iosif

Thomas Wies

Vienna, August 3, 2020

Jens Pagel

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

TU Bibliothek WIEN Your knowledge hub
Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.

Jens Pagel: *Decision Procedures for Separation Logic: Beyond Symbolic Heaps*

DECLARATION OF AUTHORSHIP

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Vienna, August 3, 2020

Jens Pagel

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.

ABSTRACT

Separation logic is a formalism for the verification of programs that make extensive use of dynamic resources, such as heap-allocated memory. Separation logic enables modular program analyses and correctness proofs through its key innovation, the separating conjunction. This has sparked the development of mature, scalable static analysis and verification tools in both academia and industry.

The degree of automation and the precision that such tools can achieve depend on the availability of solvers for checking the satisfiability and entailment problems for separation-logic formulas. As these problems are undecidable in general, various decidable fragments of separation logic have been proposed. This leads to a trade-off between expressiveness and tractability: On the one hand, we need solvers for expressive fragments of separation logic to be able to automatically prove interesting properties or discover subtle bugs. On the other hand, increasing the expressiveness of a logical formalism generally comes at a computational cost.

In this thesis, I focus on two variants of separation logic that attempt to strike the balance between expressiveness and tractability.

1. I introduce a new separation logic, *strong-separation logic with lists, trees, and data*, $\mathbf{SSL}_{\mathsf{data}}$. This logic combines a nonstandard semantics of the separating conjunction with a novel approach for constraining the data stored within data structures. This enables automated reasoning about the functional correctness of programs that use lists and trees of unbounded size. $\mathbf{SSL}_{\mathsf{data}}$ also is the first decidable separation logic that combines unbounded data structures with the so-called *magic wand* operator. This makes it possible, for example, to implement weakest-precondition calculi using $\mathbf{SSL}_{\mathsf{data}}$.

   I present a PSpace decision procedure for the logic with magic wand but without constraints on data and an SMT-based, coNP decision procedure for the entailment problem of a fragment of $\mathbf{SSL}_{\mathsf{data}}$ that subsumes the widely-used *symbolic-heap fragment*.

2. I design a novel decision procedure for a variant of *separation logic with inductive definitions* (SLID). SLID supports the definition of custom predicates by recursive equation systems. This makes it possible to reason about intricate overlaid data structures, such as trees with linked leaves (used to implement sorted-set data structures).

   While the entailment problem for SLID is undecidable in general, a large fragment was shown to be decidable by Iosif et al. [IRS13].

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

The original decidability proof relied on a reduction that caused a blow-up of the problem size by several exponentials. In contrast, the decision procedure I present in this thesis is asymptotically optimal and operates directly on SLID formulas.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

KURZFASSUNG

Separation Logic ist ein Formalismus für die Verifikation von Programmen, die in großem Ausmaß dynamische Ressourcen, wie etwa dynamisch allokierten Speicher, verwenden. Der Erfolg von Separation Logic fußt auf der namensgebenden *Separating Conjunction* ("trennende Konjunktion"), die modulare Programmanalysen und Korrektheitsbeweise ermöglicht. Diese Modularität ist der Grund dafür, dass sich inzwischen eine Reihe von ausgereiften akademischen und industriellen Tools für statische Analyse und Programmverifikation Separation Logic zunutze machen.

Der Grad der Automatisierung sowie die Präzision, die solche Tools erreichen können, hängen von der Verfügbarkeit von Solvern ab, die automatisch die Erfüllbarkeit von Separation-Logic-Formeln sowie die Gültigkeit von Implikationen zwischen solchen Formeln ueberprüfen können. Weil die entsprechenden Entscheidungsprobleme im Allgemeinen unentscheidbar sind, wurden eine Vielzahl entscheidbarer Fragmente von Separation Logic entwickelt. Die Auswahl von Fragmenten geht stets mit einem Kompromiss zwischen Ausdrucksstärke und Komplexität einher: Einerseits kann man mit Separation Logic interessante Programmeigenschaften nur dann automatisiert zeigen und subtile Bugs nur dann automatisiert finden, wenn Solver für ausdrucksstarke Fragmente der Logik zur Verfügung stehen. Andererseits führt eine hohe Ausdrucksstärke eines logischen Formalismus oft dazu, dass Entscheidungsprozeduren eine impraktiabel hohe Komplexität haben.

In dieser Dissertation konzentriere ich mich auf zwei Varianten von Separation Logic, die auf einen Kompromiss zwischen Ausdrucksstärke und Komplexität abzielen.

1. Ich entwickle eine neue Separation Logic, *Strong-Separation Logic mit Listen, Bäumen und Daten*, $\mathbf{SSL}_{\mathsf{data}}$. Diese Logik kombiniert eine nicht-standard Semantik der Separating Conjunction mit einem neuartigen Ansatz, Aussagen über die in Datenstrukturen gespeicherten Daten zu treffen. Dies ermöglicht automatisierte Schlussfolgerungen über die funktionale Korrektheit von Programmen, die Listen und Bäume unbeschränkter Größe verwenden. $\mathbf{SSL}_{\mathsf{data}}$ ist außerdem die erste entscheidbare Separation Logic, die sowohl Datenstrukturen unbeschränkter Größe als auch den sogenannten *Magic Wand* ("Zauberstab") unterstützt. Der Magic Wand kann beispielsweise genutzt werden, um einen Weakest-Precondition-Kalkül ("schwächste Voerbedingungen") mittels $\mathbf{SSL}_{\mathsf{data}}$ zu implementieren.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Ich präsentiere eine PSpace-Entscheidungsprozedur für die Logik mit Magic Wands, aber ohne Aussagen über gespeicherte Daten; und eine SMT-basierte, coNP-Entscheidungsprozedur für Implikationen in einem Fragment, das das weitverbreitete *Symbolic-Heap-Fragment* umfasst und Aussagen über gespeicherte Daten erlaubt.

2. Ich entwerfe eine neuartige Entscheidungsprozedur für *Separation Logic mit induktiven Definitionen* (SLID). In SLID kann man rekursive Gleichungssysteme verwenden, um eigene Prädikate zu definieren. Somit können die in einem Programm verwendeten Datenstrukturen in der Separation Logic abgebildet werden. SLID ist ausdrucksstark genug, um komplizierte, überlagerte Datenstrukturen zu formalisieren. Beispielsweise kann man Bäume mit verketteten Blättern axiomatisieren, die in der Implementierung von Sorted-Set-Datenstrukturen Verwendung finden.

   Das Problem, Implikationen zwischen SLID-Formeln zu überprüfen, ist im Allgemeinen unentscheidbar, aber Iosif et al. [IRS13] konnten die Entscheidbarkeit eines großen Fragments der Logik zeigen. Der ursprüngliche Entscheidbarkeitsbeweis verwendete eine Reduktion auf einen anderen Formalismus, was zu einer mehrfach exponentiellen Komplexität führte. Im Gegensatz dazu arbeitet die Entscheidungsprozedur, die ich in dieser Dissertation präsentiere, direkt auf SLID-Formeln und erreicht eine asymptotisch optimale Komplexität.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

PUBLICATIONS

[Jan+17] Christina Jansen, Jens Katelaan, Christoph Matheja, Thomas Noll, and Florian Zuleger. "Unified Reasoning About Robustness Properties of Symbolic-Heap Separation Logic." In: *Programming Languages and Systems - 26th European Symposium on Programming, ESOP 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings*. 2017, pp. 611–638.

[KJW18] Jens Katelaan, Dejan Jovanović, and Georg Weissenbacher. "A Separation Logic with Data: Small Models and Automation." In: *Automated Reasoning - 9th International Joint Conference, IJCAR 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings*. 2018, pp. 455–471.

[Kat+18] Jens Katelaan, Christoph Matheja, Thomas Noll, and Florian Zuleger. "Harrsh: A Tool for Unified Reasoning about Symbolic-Heap Separation Logic." In: *Proceedings of the 13th International Workshop on the Implementation of Logics (IWIL)*. 2018.

[KMZ19] Jens Katelaan, Christoph Matheja, and Florian Zuleger. "Effective Entailment Checking for Separation Logic with Inductive Definitions." In: *Tools and Algorithms for the Construction and Analysis of Systems - 25th International Conference, TACAS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings, Part II*. 2019, pp. 319–336.

[PMZ20] Jens Pagel, Christoph Matheja, and Florian Zuleger. "Complete Entailment Checking for Separation Logic with Inductive Definitions." In: *To be published. Preprint available at CoRR* abs/2002.01202 (2020). arXiv: 2002.01202.

[PZ20a] Jens Pagel and Florian Zuleger. "Beyond Symbolic Heaps: Deciding Separation Logic With Inductive Definitions." In: *LPAR 2020: 23rd International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Alicante, Spain, May 22-27, 2020*. Ed. by Elvira Albert and Laura Kovács. Vol. 73. EPiC Series in Computing. EasyChair, 2020, pp. 390–408.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[PZ20b] Jens Pagel and Florian Zuleger. “Strong-Separation Logic.” In: *Submitted. Preprint available at CoRR* abs/2001.06235 (2020). arXiv: 2001.06235.

[Sig+19] Mihaela Sighireanu et al. “SL-COMP: Competition of Solvers for Separation Logic.” In: *Tools and Algorithms for the Construction and Analysis of Systems - 25 Years of TACAS: TOOLympics, Held as Part of ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings, Part III*. 2019, pp. 116–132.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

ACKNOWLEDGMENTS

First and foremost, I would like to thank my supervisors. Without Florian Zuleger's guidance, his attention to details, his ideas and insights, his insistence to improve my scientific writing, and his ability to turn espresso into proofs at ungodly hours, this thesis would likely never have been. (Or at the very least, it would be even more incromprehensible than it is now.) I could not have wished for a better supervisor. Without Georg Weissenbacher's support throughout my PhD, and in particular in the year following Helmut Veith's untimely death, I may very well have quit the PhD program years ago. I could not have wished for a better co-supervisor.

Unfortunately, I will never have a chance to thank Helmut Veith in person. Helmut inspired me to pursue my PhD in Vienna. He managed to leave a lasting impact on me over the few short months that I worked with him.

I would also like to thank Radu Iosif and Thomas Wies for all the discussions we had over the years, but especially for taking on the far from easy task to review my thesis.

I am extremely grateful to Christoph Matheja, who often felt more like a supervisor than a fellow PhD student to me. Our long-lasting collaboration has led to the main results of my thesis.

I am just as grateful to Dejan Jovanović—for hosting me at SRI New York—and for teaching me much more about doing science than should have been possible in twelve short weeks.

My PhD would not have been half as much fun without my fellow students of the doctoral college LogiCS. I could really list everyone of you here. I loved sharing my office with Katalin Fazekas, Benjamin Kiesl and Nadia Labai. Neha Lodha and her WG made sure that I had a roof over my head, Indian food in my belly and lots of fun during my visits to Vienna in the final year of my PhD. Thank you!

There's a long list of people have played a role, large or small, in the success of my PhD. I would especially like to thank Mnacho Echenim, Nikos Gorogiannis, Lukáš Holík, Christina Jansen, Tomer Kotek, Siddharth Krishna, Thomas Noll, Nicolas Peltier, Ruzica Piskac, Adam Rogalewicz, Mihaela Sighireanu, Veronika Šoková and Tomáš Vojnar for the many fruitful discussions (over caffeinated or alcoholic beverages), for our collaborations, for your hospitality, and for the extremely useful feedback you provided on my work. Each of you will know which points on the list apply to you.

I am also grateful to Juliane Auerböck, Beatrix Buhl, Eva Nedoma and Toni Pisjak for always making sure that I could focus on my research rather than paperwork and IT infrastructure.

TU WIEN Bibliothek Your knowledge hub
Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.

Finally, I've always been able to count on the support of my friends and family. One person in particular has been so important to me over the last seven years that I now share her last name. Thank you, Mélanie.

TU Bibliothek Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
WIEN Your knowledge hub The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.

CONTENTS

List of Figures xvi
List of Tables xvii
List of Symbols xviii

I INTRODUCTION & OVERVIEW
1 INTRODUCTION 3
1.1 A Short Guide to This Thesis 5
2 AN INTRODUCTION TO SEPARATION LOGIC 7
2.1 Notation 7
2.2 Why Separation Logic? 8
2.3 A Basic First-Order Separation Logic 13
2.3.1 The Syntax of Separation Logic 13
2.3.2 The Semantics of Separation Logic 14
2.3.3 Automated Reasoning about Separation Logic 17
2.3.4 Further Reading 18
2.4 Reasoning about Stack–Heap Models 19
3 CONTRIBUTIONS & OVERVIEW 21
3.1 Strong-Separation Logic 22
3.2 Data Predicates 24
3.3 Inductive Definitions 25
3.4 Abstraction-Based Satisfiability Checking 27
4 HEAPS AS DIRECTED GRAPHS 31

II DECIDING STRONG-SEPARATION LOGIC WITH TREES AND DATA
5 STRONG-SEPARATION LOGIC 39
5.1 Strong-Separation Logic without Data Predicates 40
5.1.1 Data Structures 40
5.1.2 Syntax of Strong-Separation Logic 41
5.1.3 Semantics of Strong-Separation Logic 43
5.1.4 Strong-Separation Logic vs Weak-Separation Logic 50
5.2 Adding Data Predicates to SSL 53
5.2.1 A Semi-Formal Introduction to SSL with Data Predicates 53
5.2.2 Formal Syntax and Semantics of SSL with Data 55
6 DECIDING SSL WITHOUT DATA PREDICATES 57
6.1 Abstract Memory States 58
6.2 The Refinement Theorem for SSL 65
6.2.1 Abstract Memory States of Atomic Formulas 65
6.2.2 Composing Abstract Memory States 67
6.2.3 Proving the Refinement Theorem 68
6.3 Computing Abstract Memory States 70

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

6.3.1 Abstracting Non-Atomic SSL Formulas 70
6.3.2 Refining the Refinement Theorem: Bounding Garbage 72
6.4 Deciding SSL by AMS Computation 75
6.5 Complexity of the SSL Satisfiability Problem 79
7 DECIDING POSITIVE SSL 83
7.1 The Small-Model Property of SSL 83
7.2 Encoding Positive Formulas into SMT 91
7.2.1 Correctness of the SMT Encoding of Lists and Trees 99
7.2.2 Correctness of the SMT Encoding of Arbitrary Positive Formulas 103

III DECIDING SEPARATION LOGIC WITH INDUCTIVE DEFINITIONS
8 SEPARATION LOGIC WITH INDUCTIVE DEFINITIONS 109
8.1 Syntax of Separation Logic with Inductive Definitions 110
8.2 Semantics of Separation Logic with Inductive Definitions 113
8.3 SIDs with Models of Bounded Treewidth 115
8.4 Guarded Models and Dangling Pointers 117
9 TOWARDS A COMPOSITIONAL ABSTRACTION FOR SLID 121
9.1 First Attempt: Abstracting Models by Symbolic Heaps 121
9.2 Second Attempt: Unfolding Predicates into Forests 122
9.3 Third Attempt: Forest Projections 126
9.4 Beyond the Third Attempt 127
10 FORESTS AND THEIR PROJECTIONS 129
10.1 Forests 129
10.1.1 Composing Forests 133
10.2 Forest Projections 135
10.2.1 Interlude: Guarded Quantifiers 135
10.2.2 Forest Projections without Stacks 137
10.2.3 Stack–Forest Projection 142
10.3 Composing Projections 146
10.3.1 Motivation 146
10.3.2 Formal Definition of Projection Composition 148
10.3.3 Relating Forest Composition and Projection Composition 150
11 THE TYPE ABSTRACTION 157
11.1 Delimited Unfolded Symbolic Heaps 158
11.1.1 Interfaces 158
11.1.2 Delimited Forests and Their Projections 159
11.1.3 Finiteness of the DUSH Fragment 162
11.2 Defining the Type Abstraction 163
11.2.1 From Stacks to Stack-Aliasing Constraints 165
11.2.2 Composing Types 166

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

11.2.3 Instantiating Variables in Φ-Types 169
11.2.4 Forgetting Variables in Φ-Types 170
11.2.5 Size of the Type Domain 171
12 USING TYPES TO DECIDE GUARDED SEPARATION LOGIC 173
12.1 The Refinement Theorem for Guarded Formulas 173
12.2 Computing the Types of Predicate Calls 179
12.2.1 Computing on Sets of Types 179
12.2.2 Interlude: Consequences of Establishment 180
12.2.3 A Fixed-Point Algorithm for Computing the Types of Predicates 184
12.2.4 Soundness of the Type Computation 186
12.2.5 Completeness of the Type Computation 189
12.2.6 Correctness and Complexity of the Fixed-Point Computation 192
12.3 Computing the Types of Guarded Formulas 195
13 BEYOND GUARDED SEPARATION LOGIC: UNDECIDABILITY PROOFS 201
13.1 Undecidability of Unguarded SLID 201
13.2 Discussion 207

IV CONCLUSION
14 RELATED WORK 211
14.1 Logics for Reasoning about the Shape of the Heap 211
14.2 Logics for Reasoning about Shape and Content 214
14.3 Tool Support 215
15 CONCLUSION 217
Index 221

Bibliography 227

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

LIST OF FIGURES

Figure 2.1 Syntax of $\mathbf{SL}_{\mathbf{base}}$ 13
Figure 2.2 Semantics of $\mathbf{SL}_{\mathbf{base}}$ 15
Figure 3.1 Example: Model decomposition with strong and weak separation 24
Figure 3.2 Example: SID defining trees with linked leaves 27
Figure 5.1 Syntax of **SSL**, $\mathbf{SSL}^{+}$ 42
Figure 5.2 Derived **SSL** formulas 42
Figure 5.3 Semantics of **SSL** 45
Figure 5.4 Example: **SSL** vs. **WSL** semantics 51
Figure 5.5 Semantics of $\mathbf{SSL}_{\mathsf{data}}$ 56
Figure 6.1 Example: Parameterization of **SSL** satisfiability by stack variables 58
Figure 6.2 Example: Graphical representation of chunks and AMS 59
Figure 6.3 Types of negative chunks 61
Figure 6.4 Algorithm: Computing the AMS of an **SSL** formula 76
Figure 6.5 Reduction: Translating QBF to **SSL** 80
Figure 7.1 Example: Witnesses of existential data predicates 87
Figure 7.2 Example: Removal of locations without changing the induced AMS 88
Figure 7.3 SMT encoding of $\mathbf{SSL}^{+}$ without inductive predicates 94
Figure 8.1 Syntax of SL with inductive definitions 110
Figure 8.2 Semantics of SL with inductive definitions 114
Figure 9.1 Example: A model and its Φ-tree 123
Figure 9.2 Example: Φ-trees with holes 125
Figure 9.3 Example: A Φ-forest for trees with parent pointers 125
Figure 10.1 Example: A Φ-forest whose projection contains an existential 132
Figure 10.2 Rewriting rules for $\mathbf{SLID}^{\exists\forall}_{\mathsf{btw}}$ 138
Figure 10.3 Example: Induced models of Φ-forests 144
Figure 10.4 Proof: Soundness of stack–forest projection 145
Figure 12.1 Algorithm: Computing the Φ-types of SID rules 185
Figure 12.2 Algorithm: Computing the Φ-types of $\mathbf{SLID}^{\mathsf{g}}_{\mathsf{btw}}$ formulas 195
Figure 13.1 An SID encoding of CFG derivations 202
Figure 13.2 Example: Encoding a CFG derivation as stack–heap model 203

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

LIST OF TABLES

Table 3.1 Key definitions and results in Parts ii and iii 29

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Symbol	Description
$\bot$	undefined; $f(x) = \bot$ iff $x \notin \mathrm{dom}(f)$ 7
$\cdot$	sequence concatenation 7
$\circ$	function composition 7
$\rightharpoonup$	partial function 7
$\bullet_{\mathbf{F}}$	composition of Φ-forests 135
$\bullet_{\mathbf{P}}$	composition of unfolded symbolic heaps 149
$\bullet$	composition of AMS or sets of AMS 67
$-\!\bullet$	abstract magic wand in AMS computation 72
$\uplus^{\mathfrak{s}}$	strong disjoint union of heaps w.r.t. stack $\mathfrak{s}$ 43
$+$	disjoint union of functions 43
$\overset{\mathsf{mp}}{\Longrightarrow}$	implies by generalized modus ponens 138
$\Rightarrow$	one-step derivability relation between words induced by CFG 201
$\Rightarrow^{+}$	transitive closure of $\Rightarrow$ 201
$\blacktriangleright$	one-step derivability relation on Φ-forests 134
$\blacktriangleright^{*}$	reflexive–transitive closure of $\blacktriangleright$ 134
$\triangleright$	one-step derivability relation on unfolded symbolic heaps 147
$\triangleright^{*}$	reflexive–transitive closure of $\triangleright$ 149
$\equiv_{\alpha}$	α-equivalence on Φ-trees and Φ-forests 151
$\cong$	isomorphism 19
$\equiv$	rewrite equivalence on formulas with guarded quantifiers 137
$\xrightarrow{\exists\mathbf{e}/\forall\mathbf{u}}$	partial instantiation of universals with existentials from $\mathbf{e}$ or universals from $\mathbf{u}$ 148
$\models$	model relationship entailment 9
$\models_{\mathbf{Data}}$	model relationship of the data theory $\mathcal{T}_{\mathbf{Data}}$ 40
$\models_{\Phi}$	model relationship or entailment w.r.t. SID Φ 114

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

$\models_{\mathbf{x}}$	entailment on models with $\mathrm{dom}(\mathfrak{s}) = \mathbf{x}$ 49
$\mapsto$	points-to assertion 13, 110
$\mapsto_{\mathsf{l,r}}$	points-to assertion allocating a tree node (no data field specified) 41
$\mapsto_{\mathsf{ls}}$	points-to assertion allocating a list node 55
$\mapsto_{\mathsf{n}}$	points-to assertion allocating a list node (no data field specified) 41
$\mapsto_{\mathsf{tree}}$	points-to assertion allocating a tree node 55
$\prec$	access path ordering on locations in directed graph 33
$\approx$	equality between variables 13, 42, 110
$\not\approx$	dis-equality between variables 13, 42, 110
$\sqsubseteq$	sub-function relation; $f \sqsubseteq g$ holds iff for all $x \in \mathrm{dom}(f)$, $f(x) \subseteq g(x)$ 8, 186
$\star$	separating conjunction 11, 13, 42
$\bigstar$	iterated separating conjunction 43, 111
$\star^{\mathbf{w}}$	weak separating conjunction 50
$\bar{\star}$	separating conjunction with re-scoping (pushing existentials out) 148
$-\!\star$	magic wand / separating implication 13, 42, 111
$-\!\circledast$	septraction / existential wand 13, 42, 111
$\lceil\phi\rceil$	the chunk size of formula ϕ 73
$[x]^{\mathfrak{s}}_{=}$	equivalence class of x w.r.t. stack $\mathfrak{s}$ 20
$\langle x_1, \ldots, x_k \rangle$	ordered sequence consisting of the elements $x_1, \ldots, x_k$ 7
$[F]^{\exists}$	existential unary data predicate 53
$[\mathsf{f}\colon F]^{\exists}$	existential binary data predicate for field f 53
$[F]^{\forall}$	universal unary data predicate 53
$[\mathsf{f}\colon F]^{\forall}$	universal binary data predicate for field f 53
$\{\phi\}\, c\, \{\psi\}$	Hoare triple with precondition ϕ, program c, and postcondition ψ 8
$\forall\!\!\!\forall$	guarded universal quantifier 135
$\exists\!\!\exists$	guarded existential quantifier 135
α	first free variable of a unary or binary data predicate 53

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Wien Bibliothek Your knowledge hub

β	second free variable of a binary data predicate 53
ε	the empty sequence 7
λ	notation for defining anonymous functions 8
Π	a pure constraint 111
Σ	a stack-aliasing constraint 20
$\Sigma\|_{\mathbf{y}}$	restriction of stack-aliasing constraint Σ to $\mathbf{y}$ 20
$\Sigma[\mathbf{x}/\mathbf{y}]$	instantiate variables $\mathbf{x}$ with $\mathbf{y}$ in stack-aliasing constraint Σ 184
Φ	system of inductive definitions 112
$\phi(\mathbf{z})$	replace free variables of ϕ with $\mathbf{z}$, equivalent to $\phi[\mathsf{fvars}(\phi)/\mathbf{z}]$ 113
$\phi[\mathbf{y}/\mathbf{z}]$	instantiate (free or bound) variables $\mathbf{y}$ with $\mathbf{z}$ in ϕ 113
$\mathcal{A}$	an abstract memory state (AMS) 61
$\mathsf{abst}_{\mathfrak{s}}(\phi)$	algorithm for computing the set $\mathsf{ams}_{\mathfrak{s}}(\phi) \cap \mathbf{AMS}_{k,\mathfrak{s}}$, for $k = \lceil\phi\rceil$ 75
$\mathbf{AbstLists}(\mathfrak{s}, x, \mathbf{y})$	set of abstract lists w.r.t. $\mathfrak{s}$ with head x and holes $\mathbf{y}$ 66
$\mathbf{AbstLists}_{\geq 2}(\mathfrak{s}, x, \mathbf{y})$	set of abstract lists of size at least 2 w.r.t. $\mathfrak{s}$ with head x and holes $\mathbf{y}$ 66
$\mathbf{AbstTrees}(\mathfrak{s}, x, \mathbf{y})$	set of abstract trees w.r.t. $\mathfrak{s}$ with root x and holes $\mathbf{y}$ 66
$\mathbf{AbstTrees}_{\geq 2}(\mathfrak{s}, x, \mathbf{y})$	set of abstract trees of size at least 2 w.r.t. $\mathfrak{s}$ with root x and holes $\mathbf{y}$ 66
$\mathbf{AC}^{\mathbf{x}}$	the set of all stack-aliasing constraints over variables $\mathbf{x}$ 20
$\mathsf{aliasing}(\mathfrak{s})$	the stack-aliasing constraint of stack $\mathfrak{s}$ 20
$\mathsf{allholepreds}(\mathfrak{t})$	set of all hole predicates of Φ-tree $\mathfrak{t}$ 130
$\mathsf{allholes}(\mathfrak{f})$	set of all holes across all trees of Φ-forest $\mathfrak{f}$ 132
$\mathsf{allholes}(\mathfrak{t})$	set of all holes of Φ-tree $\mathfrak{t}$ 130
$\mathbf{alloc}(\mathcal{A})$	the allocated variables of an AMS 62
$\mathsf{alloc}(x)$	derived SSL formula expressing that x is allocated 42
$\mathsf{alloced}(\mathfrak{s}, \mathfrak{h})$	set of allocated variables of the model 19
$\mathsf{alloc}^{-}(\mathfrak{s}, \mathfrak{h})$	the sets of variables allocated in negative chunks of $(\mathfrak{s}, \mathfrak{h})$ 63
$\mathbf{AMS}$	the set of all AMS 62

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Symbol	Description
$\mathsf{ams}(\mathfrak{s}, \mathfrak{h})$	the induced AMS of $(\mathfrak{s}, \mathfrak{h})$ 63
$\mathbf{AMS}_{k,\mathfrak{s}}$	the set of all AMS with nodes $\mathsf{classes}(\mathfrak{s})$ and garbage-chunk count at most k 75
$\mathbf{AMS}_{\mathfrak{s}}$	the set of all AMS with nodes $\mathsf{classes}(\mathfrak{s})$ 62
$\mathsf{ams}_{\mathfrak{s}}(\phi)$	the abstraction of **SSL** formula ϕ by the set of all AMS of models with stack $\mathfrak{s}$ that satisfy ϕ 64
$\mathsf{ams}_{\mathbf{x}}(\phi)$	the abstraction of **SSL** formula ϕ by the set of all AMS of models with $\mathrm{dom}(\mathfrak{s}) = \mathbf{x}$ that satisfy ϕ 64
$\mathsf{ar}(\mathsf{pred})$	arity of predicate identifier pred 110
$\mathsf{array}\ \mathsf{s}_1\ \mathsf{s}_2$	the sort of arrays with indices of sort s_1 and elements of sort s_2 91
Bool	Boolean sort in SMT 91
$\mathsf{bound}(\phi)$	upper bound on the size of the minimal model of $\phi \in \mathbf{SSL}^{+}_{\mathsf{data}}$ 87
$\mathsf{calls}_{\mathfrak{t}}(l)$	the recursive calls at location l in Φ-tree $\mathfrak{t}$ 130
CFG	the set of all context-free grammars 201
$\mathsf{chunks}(\mathfrak{s}, \mathfrak{h})$	all chunks of the model $(\mathfrak{s}, \mathfrak{h})$ 58
$\mathsf{chunks}^{-}(\mathfrak{s}, \mathfrak{h})$	the negative chunks of the model $(\mathfrak{s}, \mathfrak{h})$ 60
$\mathsf{chunks}^{+}(\mathfrak{s}, \mathfrak{h})$	the positive chunks of the model $(\mathfrak{s}, \mathfrak{h})$ 60
$\mathsf{classes}(\mathfrak{s})$	the equivalence classes of $\mathsf{aliasing}(\mathfrak{s})$ 20
$\mathcal{D}$	the set of built-in data structures of **SSL** 40
$\mathsf{d}(\ell)$	data stored at location ℓ 41
$\mathsf{dangling}(\mathfrak{h})$	set of dangling locations of the heap $\mathfrak{h}$ 19
Data	the set of data values that can be stored in the heap 39
$\mathrm{dom}(\Sigma)$	domain of stack-aliasing constraint Σ 20
$\mathrm{dom}(f)$	domain of (partial) function f 7
$\mathrm{dom}(\mathfrak{f})$	the domain of the Φ-forest $\mathfrak{f}$ 131
$\mathsf{dropdata}(\phi)$	drop all data predicates, data fields, and data formula from ϕ 87
ds	an arbitrary data structure from $\mathcal{D}$ 40
$\mathbf{DUSH}_{\Phi}$	set of all DUSHs over SID Φ 162
$\mathsf{dushroots}_{\mathfrak{s}}(\phi)$	roots (i.e., root variables of the predicates on the right-hand side of magic wands) of DUSH ϕ 174
$\mathbf{DUSH}^{\mathbf{x}}_{\Phi}$	set of all DUSHs ϕ over SID Φ with $\mathrm{fvars}(\phi) \subseteq \mathbf{x}$ 162

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Symbol	Description
$\mathsf{edges}_{\mathfrak{s}}(\mathfrak{h}_c)$	the induced hyperedges of chunk $\mathfrak{h}_c$ w.r.t. stack $\mathfrak{s}$ 62
emp	empty-heap predicate 13, 42, 110
even	singly-linked lists of even length 112
f	false 42
$\mathfrak{f}$	a Φ-forest 131
$\mathcal{F}_{\mathbf{Data}}$	the set of quantifier-free $\mathcal{T}_{\mathbf{Data}}$ formulas 39
$\mathsf{forests}_{\Phi}(\mathfrak{h})$	set of Φ-forests $\mathfrak{f}$ with $\mathsf{heap}(\mathfrak{f}) = \mathfrak{h}$ 157
$\mathsf{forget}_{\Sigma,y}(\phi)$	forget variable y in ϕ, assuming stack-aliasing constraint Σ 170
$\mathsf{forget}_{\Sigma,y}(\mathcal{T})$	forget variable y in Φ-type $\mathcal{T}$, assuming stack-aliasing constraint Σ 170
$\mathsf{fvars}(\phi)$	the set of free variables of formula ϕ 43
$\mathsf{fvars}(\mathsf{pred})$	the set of parameters of predicate pred 112
$\mathcal{G}$	a directed (indexed) graph 31
$\mathbf{G}$	a context-free grammar 201
$\mathsf{graph}(\mathfrak{h})$	induced graph of heap $\mathfrak{h}$ 31
$\mathsf{graph}(\mathfrak{f})$	induced graph of Φ-forest $\mathfrak{f}$ 131
$\mathsf{graph}(\mathfrak{t})$	induced graph of Φ-tree $\mathfrak{t}$ 130
$\mathfrak{h}$	a heap 14
$\mathsf{head}_{\mathfrak{t}}(l)$	the head predicate at location l in Φ-tree $\mathfrak{t}$ 130
Heaps	the set of all heaps 15
$\mathsf{heap}(\mathfrak{t})$	the induced heap of Φ-tree $\mathfrak{t}$ 130
$\mathsf{heap}_{\mathfrak{t}}(l)$	the induced heap of location l in Φ-tree $\mathfrak{t}$ 130
$\mathsf{height}(\mathfrak{t})$	the height of Φ-tree $\mathfrak{t}$ 130
$\mathsf{holepreds}_{\mathfrak{t}}(l)$	the hole predicates of location l in Φ-tree $\mathfrak{t}$ 130
$\mathsf{holes}_{\mathfrak{t}}(l)$	the holes of location l in Φ-tree $\mathfrak{t}$ 130
$\mathbf{ID}_{\mathsf{btw}}$	SIDs that satisfy progress, connectivity, and establishment 26, 116
$\mathsf{igraph}(\mathfrak{h})$	induced indexed graph of heap $\mathfrak{h}$ 31
$\mathrm{img}(f)$	image of (partial) function f 7
$\mathsf{interface}(\mathfrak{f})$	the interface, i.e., the roots and holes, of the Φ-forest $\mathfrak{f}$ 158
$\mathbf{K}$	constant combinator of array theory 91
$\mathsf{l}(\ell)$	left child of tree location ℓ 41
$\mathcal{L}(\mathbf{G})$	the language of context-free grammar $\mathbf{G}$ 201

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

$\mathsf{lalloc}(\phi)$	variables allocated locally, i.e., not considering recursive calls 116
$\mathsf{lift}_{m \nearrow n}(\mathcal{A})$	bound-lifting of AMS $\mathcal{A}$ from $m \in \mathbb{N}$ to $n \in \mathbb{N}$ 75
Loc	the set of memory locations 14
$\mathsf{loc}_{\mathsf{ls}}(\mathfrak{h})$	allocated list locations of heap $\mathfrak{h}$ 40
$\mathsf{loc}_{\mathsf{tree}}(\mathfrak{h})$	allocated tree locations of heap $\mathfrak{h}$ 40
$\mathsf{locs}(\mathfrak{h})$	the locations $\mathrm{dom}(\mathfrak{h}) \cup \bigcup \mathrm{img}(\mathfrak{h})$ for heap $\mathfrak{h}$ 19
$\mathsf{locs}(\phi)$	locations that occur as terms in formula ϕ 112
$\mathsf{lref}(\phi)$	variables referenced locally, i.e., not considering recursive calls 116
ls	singly-linked list predicate in **SSL** 41
	user-defined singly-linked list predicate in **SLID** 112
$\mathsf{ls}_{\geq 2}$	restriction of ls to lists of length at least 2 41
lseg	user-defined singly-linked list segment in **SLID** 26, 112
map	map combinator of array theory 91
max	returns the maximal variable among a set or sequence of variables 20
$\mathbf{Models}_{\Phi}^{\mathrm{g}}$	the set of guarded models w.r.t. SID Φ 117
$\mathsf{n}(\ell)$	successor of list location ℓ 41
nil	a variable representing the null pointer 14
odd	singly-linked lists of odd length 112
predroot	root parameter of a predicate call 116
Preds	set of predicate identifiers 110
$\mathbf{Preds}(\Phi)$	the set of predicate identifiers that occur in SID Φ 112
$\mathsf{project}(\mathfrak{s}, \mathfrak{f})$	stack–forest projection 142
$\mathsf{project}^{\mathbf{Loc}}(\mathbf{v}, \mathfrak{f})$	forest projection of Φ-forest $\mathfrak{f}$ w.r.t. locations $\mathbf{v}$ 140
$\mathsf{project}^{\mathbf{Loc}}(\mathbf{v}, \mathfrak{t})$	tree projection of Φ-tree $\mathfrak{t}$ w.r.t. locations $\mathbf{v}$ 138
$\mathsf{ptrlocs}(\mathfrak{t})$	Set of all locations that occur in a points-to assertion in Φ-tree $\mathfrak{t}$ 130
ptr_k	a predicate defined by the single rule $\mathsf{ptr}_k(\mathbf{x}) \Leftarrow x_1 \mapsto \langle x_2, \ldots, x_{k+1} \rangle$ 117

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

$\mathsf{ptypes}^{\mathbf{x}}_{p}(\phi, \Sigma)$	the **x**-types of ϕ for aliasing-constraint Σ that can be computed by looking up the types of predicate calls using the function p 184
$\mathsf{r}(\ell)$	right child of tree location ℓ 41
$\mathsf{refed}(\mathfrak{s}, \mathfrak{h})$	set of referenced variables of the model 19
$\mathsf{root}(\mathfrak{t})$	the root location of Φ-tree $\mathfrak{t}$ 131
$\mathsf{roots}(\mathfrak{f})$	the set of all roots of the Φ-forest $\mathfrak{f}$ 131
$\mathsf{rsize}(\mathcal{A})$	the realizability size of AMS $\mathcal{A}$ 84
$\mathsf{rsize}(\phi)$	the realizability size of formula $\phi \in \mathbf{SSL}^{+}_{\mathsf{data}}$ 86
$\mathsf{rule}_{\mathfrak{f}}(l)$	the rule instance at location l in one of the Φ-trees in Φ-forest $\mathfrak{f}$ 132
$\mathsf{rule}_{\mathfrak{t}}(l)$	the rule instance at location l in Φ-tree $\mathfrak{t}$ 130
$\mathfrak{s}$	a stack 14
$\mathfrak{s}^{-1}$	the inverse of stack $\mathfrak{s}$ 20
$\mathfrak{s}^{-1}_{\mathsf{max}}$	the stack-choice function of stack $\mathfrak{s}$ 20
$\mathbf{SH}^{\exists}$	set of existentially-quantified symbolic heaps 110
$\mathsf{sig}(\mathsf{ds})$	node signature of data structure ds 40
$\mathsf{sinkseq}(\mathfrak{h})$	ordered (w.r.t. $\prec$) sink sequence of directed tree $\mathfrak{h}$ 34
$\mathsf{sinkvars}_{\mathfrak{s}}(\mathfrak{h})$	the stack-equivalence classes w.r.t. $\mathfrak{s}$ corresponding to the sinks of $\mathfrak{h}$, in the same order as the sink sequence 34
$\mathbf{SL}_{\mathbf{base}}$	a basic first-order separation logic 13
$\mathbf{SLID}^{\mathbb{V}}_{\mathsf{btw}}$	SLID formulas with guarded universals 135
$\mathbf{SLID}^{\mathbb{E}\mathbb{V}}_{\mathsf{btw}}$	SLID formulas with guarded existentials and guarded universals 135
$\mathbf{SLID}^{\mathsf{g}}$	guarded quantifier-free separation logic 110
$\mathbf{SLID}^{\mathsf{g}}_{\mathsf{btw}}$	guarded quantifier-free separation logic 117
$\mathbf{SLID}^{\mathsf{qf}}$	quantifier-free separation logic 110
$\mathbf{SLID}^{\mathsf{qf}}_{\mathsf{btw}}$	quantifier-free separation logic 117
$\mathbf{SLID}_{\mathsf{btw}}(\cdot_1, \ldots, \cdot_k)$	the SLID variant with additional atoms and operators $\cdot_1, \ldots, \cdot_k$. 201
$\mathsf{sort}(\mathsf{f})$	sort associated with the field f 41

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

$\mathsf{sourcevars}_{\mathfrak{s}}(\mathfrak{h})$ — the stack-equivalence classes w.r.t. $\mathfrak{s}$ corresponding to the sources of $\mathfrak{h}$ 34

$\mathsf{split}(\mathfrak{f}, \mathbf{l})$ — the unique $\mathbf{l}$-split of Φ-forest $\mathfrak{f}$ 134

$\mathbf{SSL}$ — strong-separation logic 39, 41

$\mathbf{SSL}^+$ — positive strong-separation logic 41

$\mathbf{SSL}_{\mathsf{data}}$ — strong-separation logic with data predicates 39, 55

$\mathbf{SSL}^+_{\mathsf{data}}$ — positive strong-separation logic with data predicates 55

$\mathbf{Stacks}$ — the set of all stacks 15

$\mathsf{succ}_{\mathfrak{t}}(l)$ — successors of location l in Φ-tree $\mathfrak{t}$ 130

t — true 42, 111, 201

$\mathfrak{t}$ — a Φ-tree 129

$\mathcal{T}_{\mathsf{array}}$ — array theory 91

$\mathcal{T}_{\mathbf{Data}}$ — background data theory 39

tll — user-defined predicate for trees with linked leaves 27, 119

tree — binary tree predicate in $\mathbf{SSL}$ 41, 112

$\mathsf{tree}_{\geq 2}$ — restriction of tree to trees of depth at least 2 41

$\mathsf{type}_{\Phi}(\mathfrak{s}, \mathfrak{h})$ — Φ-type of $(\mathfrak{s}, \mathfrak{h})$ 163

$\mathbf{Types}(\Phi)$ — all Φ-types over SID Φ 164

$\mathsf{types}(\phi, \Sigma)$ — algorithm that computes the Φ-types of $\mathbf{SLID}^{\mathsf{g}}_{\mathsf{btw}}$ formula ϕ for fixed stack-aliasing constraint Σ 195

$\mathbf{Types}^{\mathfrak{s}}(\Phi)$ — restriction of $\mathbf{Types}(\Phi)$ to types of models with stack $\mathfrak{s}$ 164

$\mathbf{Types}^{\Sigma}_{\Phi}(\phi)$ — all Φ-types of the models of ϕ with stack-aliasing constraint Σ 165

$\mathbf{Types}^{\mathfrak{s}}_{\Phi}(\phi)$ — all Φ-types of the models of ϕ with stack $\mathfrak{s}$ 164

$\mathbf{Types}^{\mathbf{x}}_{\Phi}(\phi)$ — all Φ-types of the models of ϕ with $\mathrm{dom}(\mathfrak{s}) \subseteq \mathbf{x}$ 164

$\mathbf{Val}$ — the set of values that can be stored in the stack and heap 14

$\mathbf{Var}$ — the set of (program and logical) variables 14

$\mathbf{WSL}$ — weak-separation logic 50

$\mathbf{WSL}^+$ — positive weak-separation logic 50

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Part I

INTRODUCTION & OVERVIEW

I introduce separation logic and provide an overview of the contributions of this thesis. I also introduce some notation and concepts that we will need in Parts ii and iii.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.

1 INTRODUCTION

"The job of verification is not to explore some immense search space, but to formalize the programmer's intuitions until any faults are revealed. This requires specifications and proofs that are succinct and intelligible which in turn require logics that go beyond predicate calculus (the assembly language of program proving)." [Rey11]

This is how John C. Reynolds argued for developing program logics that go beyond first-order logic in his invited talk at the *5th IEEE International Symposium on Theoretical Aspects of Software Engineering* in 2011. This was, of course, the same John C. Reynolds who is well known for co-inventing and popularizing *separation logic* [Rey02], a family of program logics that are very much in the spirit of the above edict.

Separation logics—I use the plural here because countless variants have been proposed [Par10]—are program logics for reasoning about dynamic resources such as heap-allocated memory or the locks in a concurrent program. Separation logics have found many different application areas, from compiler verification [App14] to the verification of concurrent algorithms [OHe07] and concurrent data structures [KSW18] to information-flow security [EM19], from deductive verification [Rey02; Jac+11; PWZ14b] to shape analysis [DOY06; Yan+08; Cal+11] to concolic execution [Pha+19a].

Separation logics generally have two things in common: They come with atomic predicates for concisely specifying the resource usage of the program; and they come with *separating connectives*, most importantly the *separating conjunction* , denoted $\star$. A separation-logic formula $\phi \star \psi$ expresses that the dynamic resources specified in the formulas ϕ and ψ must be disjoint—something which cannot be expressed concisely in first-order logic.

This property of the separating conjunction enables the succinct and intelligible specifications and proofs of programs that Reynolds envisioned. It is also the key to developing *compositional* proofs and *local* program analyses that have made possible the unprecedented scalability exhibited by tools such as Facebook's Infer [Cal+11; Cal+15; OHe19].

If you would like to use separation logic in your verification tool, program analyzer or mechanized proof, you have to solve two quite distinct tasks: to express the properties you would like to check and the relevant aspects of program behavior in separation-logic formulas; and to reason about these formulas. You might be familiar with this distinction in the first-order setting. For example, the frontend of a

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

program verifier such as `Dafny` [Lei10] generates *verification conditions* in a fragment of first-order logic. It then sends these formulas to a backend that determines whether the generated verification conditions hold, i.e., whether the first-order formulas are valid. `Dafny` and many other tools use *SMT solvers* such as `Z3` [MB08] for this purpose.

This thesis is about developing such backends—but for separation logic, not for first-order logic. Specifically, I present *decision procedures* for several variants of separation logics. These decision procedures can be used for *automated reasoning* about separation-logic formulas: to determine fully automatically whether or not a separation-logic formula is valid. This thesis is *not* about developing frontends. I do *not* present any verification tool or full-fledged program analysis.

Even more specifically, I limit my attention to two "flavors" of separation logic that target the analysis and verification of programs that use unbounded data structures such as lists or trees:

1. I propose *strong-separation logic with trees and data*, $\mathbf{SSL}_{\mathsf{data}}$, a novel separation logic that combines a nonstandard semantics of the separating conjunction with predicates for reasoning about list and tree structures in the heap and a mechanism for specifying properties of the content of such data structures. $\mathbf{SSL}_{\mathsf{data}}$ is expressive enough to axiomatize common data structures such as binary search trees or max heaps. At the same time, the entailment problem of **SSL**, the fragment without data constraints, is decidable in PSPACE and the entailment problem of $\mathbf{SSL}^{+}_{\mathsf{data}}$, a fragment with data predicates but without magic wands and guarded negation, is even decidable in coNP. This makes $\mathbf{SSL}_{\mathsf{data}}$ more tractable than many other program logics.

2. *Separation logic with inductive definition*, **SLID**, is a separation logic augmented with a powerful mechanism for modeling custom data structures by user-defined recursive definitions. I show that a large fragment of this logic, the 2EXPTIME-hard [EIP20] logic $\mathbf{SLID}_{\mathsf{btw}}$ [IRS13], is decidable in 2EXPTIME. I also propose an extension of this logic, $\mathbf{SLID}^{\mathsf{g}}_{\mathsf{btw}}$, that remains in 2EXPTIME despite supporting guarded variants of the *magic wand*, the *septraction* operator, and negation.

In contrast to many other publications about separation logic, you will not find any proof theory in this document. Instead, I take a model-theoretic approach, exploiting that the program heap is essentially a directed graph, so reasoning about separation-logic formulas often boils down to reasoning about classes of graphs.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

1.1 A SHORT GUIDE TO THIS THESIS

My thesis consists of four parts.

The first part, which you are currently reading, continues as follows. In Chapter 2, I establish notational conventions and introduce separation logic. I motivate and summarize the contributions of my thesis in Chapter 3. As this chapter provides an overview of the remainder of my thesis, it should be considered essential reading. The first part of the thesis ends with a discussion of the connections between the *stack–heap model* of separation logic and directed graphs in Chapter 4.

Each of the main parts of my thesis, Parts ii and iii, is centered around developing decision procedures for one of the aforementioned variants of separation logic: Part ii studies strong-separation logic with trees and data. Part iii studies separation logic with inductive definitions—in particular, with a fragment of inductive definitions introduced by Iosif et al. [IRS13].

I discuss related work and conclude in Part iv.

My thesis builds on joint work with Christina Jansen, Dejan Jovanović, Christoph Matheja, Thomas Noll, Georg Weissenbacher, and Florian Zuleger [Jan+17; KJW18a; Kat+18; KMZ19a; PMZ20; PZ20a; PZ20b].

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.

2 AN INTRODUCTION TO SEPARATION LOGIC

In the introduction, I promised that separation logic enables succinct specifications, intelligible proofs, and compositional verification. Succinctness and intelligibility should need no further justification (though I grant that intelligibility is, to a significant extent, in the eye of the beholder). It is less obvious that compositionality is desirable; or even what it means for verification to be compositional. Likewise, it isn't obvious why compositionality is difficult to achieve for programs that use dynamic memory. To make this document self contained, I will motivate separation logic by addressing these points before delving into the formal details. I first need to introduce some of the notation that I use throughout this thesis.

2.1 NOTATION

In this section, I establish the basic notational conventions that I use both later in this chapter and in the main parts of this thesis, Parts ii and iii.

ORDERED SEQUENCES. Ordered sequences are denoted in boldface, e.g., $\mathbf{x}$. The size of the sequence $\mathbf{x}$ is $|\mathbf{x}|$. To list the elements of a sequence, I write $\langle x_1, \ldots, x_k \rangle$. Further, I often omit the brackets around single-element sequences—i.e., write x instead of $\langle x \rangle$—to reduce clutter.

The empty sequence is ε and the concatenation of $\mathbf{x}$ and $\mathbf{y}$ is $\mathbf{x} \cdot \mathbf{y}$.

As usual, A^* denotes finite sequences over A, A^+ denotes nonempty finite sequences over A.

Moreover, I frequently treat sequences as sets when convenient and order does not matter. For example, $x \in \mathbf{x}$ states that the sequence $\mathbf{x}$ contains the element x, $\mathbf{x} \cup \mathbf{y}$ is the set that contains the elements of both sequences, etc.

FUNCTIONS. I write $f \colon A \rightharpoonup B$ to denote a partial function from A to B; $\mathrm{dom}(f)$ is the domain of (partial) function f and $\mathrm{img}(f)$ is the image of (partial) function f. The fact that partial function f is undefined on x is denoted $f(x) = \bot$. As usual, $f \circ g$ is the composition of functions f and g that maps x to $f(g(x))$.

We will also need notation to compare partial functions:

1. For functions $f, g \colon A \rightharpoonup B$, $f \subseteq g$ holds if $\mathrm{dom}(f) \subseteq \mathrm{dom}(g)$ and $f(a) = g(a)$ for all $a \in \mathrm{dom}(f)$.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

2. For functions $f, g\colon A \rightharpoonup 2^B$, $f \sqsubseteq g$ holds if $\mathrm{dom}(f) \subseteq \mathrm{dom}(g)$ and $f(a) \subseteq g(a)$ for all $a \in \mathrm{dom}(f)$.

I frequently use set notation to define and reason about partial functions. For example, $\{x_1 \mapsto y_1, \ldots, x_k \mapsto y_k\}$ is the partial function that maps x_i to y_i, $1 \leq i \leq k$, and is undefined on all other values. Consequently, $\emptyset$ represents a function that is undefined everywhere. $f \cup g$ is the (not necessarily disjoint) union of partial functions f and g; it is only defined if $f(x) = g(x)$ for all $x \in \mathrm{dom}(f) \cap \mathrm{dom}(g)$. The size of a partial function f is the size of its domain, i.e., $|f| \triangleq |\mathrm{dom}(f)|$.

Finally, the following conventions will simplify defining and using functions.

LIFTING FUNCTIONS TO SEQUENCES. For $f\colon A \rightharpoonup B$ and $\mathbf{a} \in A^*$, then $f(\mathbf{a})$ for the point-wise application of f to $\mathbf{a}$, i.e.,

$$f(\langle a_1, \ldots, a_k \rangle) \triangleq \langle f(a_1), \ldots, f(a_k) \rangle .$$

OMITTING BRACKETS. If $f\colon A^* \rightharpoonup B$, I sometimes write $f(a_1, \ldots, a_k)$ instead of $f(\langle a_1, \ldots, a_n \rangle)$ to improve readability.

LAMBDA NOTATION. The expression $\lambda x.\, f(x)$ defines an anonymous (partial) function that maps x to $f(x)$.

Additional information

Throughout this thesis, you will from time to time encounter boxes like this one. I use these boxes to shed light on the larger context or provide additional information that is connected to the content at hand. Reading the boxes is not necessary for following the technical development.

2.2 WHY SEPARATION LOGIC?

In this section, I provide a tutorial-like motivation for separation logic. If you already have some familiarity with separation logic, you can skip this section and continue with the formal introduction to separation logic in Section 2.3.

HOARE LOGIC AND COMPOSITIONALITY. To justify the development of separation logic, we must first take a step back and look at the shortcomings of "traditional" Hoare-style program verification when it comes to programs that use dynamic memory. Recall that a *Hoare triple*

$$\{\phi\}\, c\, \{\psi\}$$

states that whenever I execute the program c starting from a program state that satisfies the precondition ϕ, if the program terminates, the

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

resulting program satisfies the formula ψ.[1] If this is the case, we say that the Hoare triple is *valid*. Usually, ϕ and ψ are formulas from first-order logic. For example, $\{x = y\}\, x := x * 2\, \{x = 2y\}$ is a valid Hoare triple that expresses that when we execute the assignment $x := x * 2$ from a program state in which x and y are equal, then the value of x is twice the value of y once the assignment has terminated.

When verifying a program using Hoare logic, we often need to combine the Hoare triples of sub-programs into a Hoare triple for the complete program. For example, say we have already established the validity of the Hoare triple $\{\phi_2\}\, \mathsf{foo}(x, y)\, \{\psi_2\}$ for the function call $\mathsf{foo}(x, y)$ and we're now analyzing a piece of code that contains a call to foo, say $c; \mathsf{foo}(x, y)$, where c is a program and ; denotes sequential composition. Assume further that we've already discovered the valid Hoare triple $\{\phi_1\}\, c\, \{\psi_1\}$. The *sequence rule* of Hoare logic allows us to combine the Hoare triples of sequentially-composed programs with the following proof rule.

$$\frac{\{\phi_1\}\, c_1\, \{\phi_2\} \qquad \{\phi_2\}\, c_2\, \{\phi_3\}}{\{\phi_1\}\, c_1; c_2\, \{\phi_3\}}$$

I've listed the premises of the proof rule above the line and the conclusion below the line, as is standard.

Continuing our above example, the sequence rule only allows us to derive the Hoare triple $\{\zeta\}\, c; \mathsf{foo}(x, y)\, \{\psi\}$, if the postcondition of the triple for c and the precondition of the triple for $\mathsf{foo}(x, y)$ match, i.e., if ψ_1 and ϕ_2 are syntactically identical. In practice, this will often not be the case. There is a way out if ψ_1 implies ϕ_2: we can first apply the *rule of consequence*.

$$\frac{\phi_0 \models \phi_1 \qquad \{\phi_1\}\, c\, \{\phi_2\} \qquad \phi_2 \models \phi_3}{\{\phi_0\}\, c\, \{\phi_3\}}$$

Here, $\models$ represents logical implication or *entailment*, i.e., $\phi_0 \models \phi_1$ holds iff all program states that satisfy ϕ_0 also satisfy ϕ_1. The rule of consequence allows us to *weaken* postconditions and to *strengthen* preconditions.

If we can determine that $\psi_1 \models \phi_2$ holds in our example, we can construct a *proof tree* to derive a valid Hoare triple for $c; \mathsf{foo}(\mathrm{x}, \mathrm{y})$ as follows.

$$\frac{\dfrac{\phi_1 \models \phi_1 \qquad \{\phi_1\}\, c\, \{\psi_1\} \qquad \psi_1 \models \phi_2}{\{\phi_1\}\, c\, \{\phi_2\}} \qquad \{\phi_2\}\, \mathsf{foo}(x, y)\, \{\psi_2\}}{\{\phi_1\}\, c; \mathsf{foo}(x, y)\, \{\psi_2\}}$$

The rule of consequence can be a rather blunt tool, however: if the entailment $\psi_1 \models \phi_2$ holds, this means that ϕ_2 is logically weaker than

1 This is known as *partial correctness*. To show total correctness, we would additionally have to prove termination.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

ψ_1; replacing ψ_1 with ϕ_2 may thus result in losing information about the program state. To see why this matters, imagine a slightly different scenario. Assume we're given the Hoare triples

$$\{\phi\}\, c\, \{\psi_1 \wedge \psi_2\}\,, \quad \{\psi_1\}\, \mathsf{foo}(x, y)\, \{\zeta_1\}\,, \quad \{\psi_2\}\, \mathsf{bar}(x, y)\, \{\zeta_2\}\,.$$

Can we prove that the Hoare triple

$$\{\phi\}\, c; \mathsf{foo}(x, y); \mathsf{bar}(x, y)\, \{\zeta_1 \wedge \zeta_2\}$$

is valid? With the rules I've presented so far, this is impossible. As before, we can combine the specifications of c and $\mathsf{foo}(x, y)$ using the rule of consequence and the sequence rule:

$$\dfrac{\dfrac{\phi \models \phi \qquad \{\phi_1\}\, c\, \{\psi_1\} \qquad \psi_1 \wedge \psi_2 \models \psi_1}{\{\phi_1\}\, c\, \{\psi_1\}} \qquad \{\psi_1\}\, \mathsf{foo}(x, y)\, \{\zeta_1\}}{\{\phi_1\}\, c; \mathsf{foo}(x, y)\, \{\zeta_1\}}$$

Unfortunately, we are now stuck: when applying the rule of consequence, we discarded the fact that the state also satisfies ζ_2. But without this fact, we cannot apply the sequence rule to combine the above proof tree with the triple $\{\psi_2\}\, \mathsf{bar}(x, y)\, \{\zeta_2\}$. To avoid this problem, we would have to reprove foo and bar, establishing the following specifications.

$$\{\psi_1 \wedge \psi_2\}\, \mathsf{foo}(x, y)\, \{\zeta_1 \wedge \psi_2\}\,, \quad \{\zeta_1 \wedge \psi_2\}\, \mathsf{bar}(x, y)\, \{\zeta_1 \wedge \zeta_2\}$$

This is bad for multiple reasons: First, whenever foo is called in a new calling context, the formula describing the global state of the program will likely be different. As this global state must be used as the precondition of foo, we have to prove the validity of a Hoare triple for a new precondition, even if we've already discovered a valid Hoare triple for foo. In other words, we likely have to prove the validity of different Hoare triples for foo and bar every time they are called.

The second problem is that we obtain needlessly large specifications: for example, the formula ζ_1 is not relevant to the execution of $\mathsf{bar}(x, y)$, so why should we have to prove $\{\zeta_1 \wedge \psi_2\}\, \mathsf{bar}(x, y)\, \{\zeta_1 \wedge \zeta_2\}$? This not only violates our stated goal of succinct, intelligible specifications, but also means that our reasoning engine has to analyze larger formulas, incurring a potentially huge computational overhead.

This example showcases that Hoare logic, in the form discussed so far, is ill-equipped for *composing* specifications: there is no way to combine the specifications of components of the program (such as the functions foo and bar) without re-analyzing these components for every calling context.

To avoid these downsides, we somehow have to get around the need to state irrelevant information in the logical formulas that make up our specifications—a problem known as the *frame problem* in the field of artificial intelligence [MH69]. Naively, we could add the following *frame rule* to Hoare logic to solve the frame problem.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

$$\frac{\{\phi\}\, c\, \{\psi\}}{\{\phi \wedge \zeta\}\, c\, \{\psi \wedge \zeta\}}$$

Using this frame rule instead of the rule of consequence makes it possible to compose the specifications of c, foo, and bar:

$$\dfrac{\dfrac{\{\phi\}\, c\, \{\psi_1 \wedge \psi_2\} \qquad \dfrac{\{\psi_1\}\, \mathsf{foo}(x,y)\, \{\zeta_1\}}{\{\psi_1 \wedge \psi_2\}\, \mathsf{foo}(x,y)\, \{\zeta_1 \wedge \psi_2\}}}{\{\phi\}\, c; \mathsf{foo}(x,y)\, \{\zeta_1 \wedge \psi_2\}} \qquad \dfrac{\{\psi_2\}\, \mathsf{bar}(x,y)\, \{\zeta_2\}}{\{\zeta_1 \wedge \psi_2\}\, \mathsf{bar}(x,y)\, \{\zeta_1 \wedge \zeta_2\}}}{\{\phi\}\, c; \mathsf{foo}(x,y); \mathsf{bar}(x,y)\, \{\zeta_1 \wedge \zeta_2\}}$$

Unfortunately, the frame rule proposed above is unsound if the specifications contain pointer variables. Consider the following simple application of the frame rule.

$$\frac{\{x \mapsto v\}\, {}^*x := w\, \{x \mapsto w\}}{\{x \mapsto v \wedge y \mapsto v\}\, {}^*x := w\, \{x \mapsto w \wedge y \mapsto v\}}$$

Here, ${}^*x$ denotes the process of dereferencing pointer variable x and $x \mapsto v$ denotes that dereferencing the pointer variable x yields value v; put more succinctly, *x points to v*.

The above rule application is unsound if x and y are aliases, in which case the postcondition should be $x \mapsto w \wedge y \mapsto w$. This illustrates that compositional reasoning about dynamic memory is hard—with aliasing, seemingly local commands can have global effects.

We could try to fix the frame rule, for example by adding side conditions that require non-aliasing for all pairs of pointer variables in the formulas. The result would be neither succinct nor intelligible (because of all the distracting aliasing information we'd have to explicitly include in the specifications), nor compositional (because when we compute a Hoare triple for a function, we may not yet known which global aliasing information may be relevant in the calling context). We will not attempt such a fix here. Instead, we will follow John C. Reynold's advice quoted in Chapter 1 and go beyond first-order logic.

SEPARATION LOGIC. The key innovation of separation logic (SL) is the *separating conjunction*, denoted $\star$ [Rey02]. The formula $\phi \star \psi$, pronounced *ϕ and separately ψ*, expresses that the dynamic resources that occur in the formulas ϕ and ψ are disjoint. In this thesis, the only dynamic resource we will look at is heap-allocated memory, but in principle, the use of other dynamic resources is possible [Got+07]. In this thesis, the formula $\phi \star \psi$ therefore always means that the program heap can be split into two disjoint parts, one of which satisfies ϕ, the other of which satisfies ψ.

At this point, countless variants of separation logic have been proposed [DD15a; Par10], targeting many different programming languages and application areas; I will provide a partial overview when I discuss related work in Chapter 14. What the different separation logics generally have in common is the *monotonicity* of the separating

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

conjunction:[2] if the entailment $\phi \models \psi$ is valid, so is $\phi \star \zeta \models \psi \star \zeta$. More precisely, the models of separation logics form *separation algebras* [COY07], a concept that will appear in Chapter 5.

The monotonicity of the separating conjunction guarantees the soundness of the following variant of the frame rule.

$$\frac{\{\phi\}\, c\, \{\psi\}}{\{\phi \star \zeta\}\, c\, \{\psi \star \zeta\}}$$

For example, the inference

$$\frac{\{x \mapsto v\}\, {}^*x := w\, \{x \mapsto w\}}{\{x \mapsto v \star y \mapsto v\}\, {}^*x := w\, \{x \mapsto w \star y \mapsto v\}}$$

is sound, because the use of $\star$ instead of $\wedge$ enforces that x and y cannot alias.

More generally, the separating conjunction enables succinct and compositional specifications. To see this, it's enough to replace $\wedge$ with $\star$ in one of our previous examples:

$$\{\phi\}\, c\, \{\psi_1 \star \psi_2\}$$
$$\{\psi_1\}\, \mathsf{foo}(x, y)\, \{\zeta_1\}$$
$$\{\psi_2\}\, \mathsf{bar}(x, y)\, \{\zeta_2\}$$

The inferences in the following proof tree are now all sound:

$$\dfrac{\dfrac{\{\phi\}\, c\, \{\psi_1 \star \psi_2\} \qquad \dfrac{\{\psi_1\}\, \mathsf{foo}(x,y)\, \{\zeta_1\}}{\{\psi_1 \star \psi_2\}\, \mathsf{foo}(x,y)\, \{\zeta_1 \star \psi_2\}}}{\{\phi\}\, c; \mathsf{foo}(x,y)\, \{\zeta_1 \star \psi_2\}} \qquad \dfrac{\{\psi_2\}\, \mathsf{bar}(x,y)\, \{\zeta_2\}}{\{\zeta_1 \star \psi_2\}\, \mathsf{bar}(x,y)\, \{\zeta_1 \star \zeta_2\}}}{\{\phi\}\, c; \mathsf{foo}(x,y); \mathsf{bar}(x,y)\, \{\zeta_1 \star \zeta_2\}}$$

It thus suffices to derive Hoare triples for foo and bar whose preconditions mention only the part of the heap that is relevant to the execution of foo and bar. We then compose these specifications into larger specifications using the frame rule, without having to re-analyze foo or bar. The frame rule thus enables *compositional verification*. Similarly, it enables succinct specifications and proofs, because the specifications of functions need only describe the local state accessible to the function.

Taking a different angle, the frame rule is the key to *local*, *modular* program analyses [DOY06; Cal+11], because—absent recursion—we can analyze or prove the correctness of each function of the program in isolation. This explains the scalability and, ultimately, success of tools based on separation logic [OHe19], which now span diverse formal and semi-formal methods, including symbolic execution [BCO05b], shape analysis [Cal+11], deductive Hoare-style verification [Jac+11; PWZ14b], interactive theorem proving [KTB17; Jun+18], concolic execution [Pha+19a], and test-input generation [Pha+19b].

2 See [CCA17] for an in-depth examination of the shared properties of and differences between separation-logic variants.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

$$\begin{array}{lcl} \phi_{\text{atom}} & ::= & \mathbf{emp} \mid x \mapsto \langle y_1, \ldots, y_k \rangle \mid x \approx y \mid x \not\approx y \\ \phi & ::= & \phi_{\text{atom}} \mid \phi \star \phi \mid \phi \mathrel{-\!\!\star} \phi \mid \phi \wedge \phi \mid \phi \vee \phi \mid \neg \phi \mid \exists x.\, \phi \mid \forall x.\, \phi \end{array}$$

Figure 2.1: The syntax of a first-order separation logic

2.3 A BASIC FIRST-ORDER SEPARATION LOGIC

Over the course of this thesis, I will study multiple variants of separation logic. I will introduce the syntax and semantics of these variants as needed in Parts ii and iii of the thesis.

In this section, I formalize a simple first-order separation logic, $\mathbf{SL}_{\mathbf{base}}$. This gives me the chance to introduce the *stack–heap model* and formalize the separate connectives in a simple setting. Moreover, $\mathbf{SL}_{\mathbf{base}}$ will serve as the basis for a discussion about alternative semantics of the separating connectives in Chapter 3.

2.3.1 *The Syntax of Separation Logic*

In Fig. 2.1, I define the syntax of the first-order separation logic $\mathbf{SL}_{\mathbf{base}}$.

Informally, the meaning of the atomic formulas is as follows.

- The *empty-heap predicate* **emp** denotes the empty heap.
- The *points-to assertion* $x \mapsto \langle y_1, \ldots, y_k \rangle$ expresses that the pointer variable x points to a heap-allocated object that consists of k fields that store the values of the variables $y_1, \ldots, y_k$.
- (Dis-)equalities $x \approx y$ and $x \not\approx y$ express that the pointer variables x and y alias and that they don't alias, respectively.

The predicates **emp** and $\cdot \mapsto \cdot$ are often called *spatial* atoms because they describe the spatial layout of the heap, whereas (dis-)equalities are often called *pure* atoms [IO01].

The atomic formulas defined in Fig. 2.1 give rise to a rather minimalistic separation logic. Most separation logics provide additional atomic formulas. Later in this thesis, I will consider an SL variant with built-in atoms representing lists and trees as well as atoms for reasoning about the data stored inside the heap (Part ii). I will also consider an SL variant with a mechanism for user-defined predicates (Part iii).

In $\mathbf{SL}_{\mathbf{base}}$, formulas are built using classical propositional connectives $\wedge, \vee, \neg$, quantifiers $\exists, \forall$, as well as a set of *separating connectives*, the aforementioned separating conjunction $\star$ and its right adjoint, the *separating implication* $\mathrel{-\!\!\star}$. Because of the appearance of the separating implication, most people use the moniker *magic wand* instead. As usual, we can derive additional operators such as classical implication $\phi \Rightarrow \psi \triangleq \neg\phi \vee \psi$ and *septraction* $\phi \mathrel{-\!\!\circledast} \psi \triangleq \neg(\phi \mathrel{-\!\!\star} \neg\psi)$ [BDL12; TBR14].

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Let us contrast the meaning of the classical and separating connectives. While $\phi \wedge \psi$ means that the program state satisfies both ϕ and ψ simultaneously, $\phi \star \psi$ denotes that the program state can be split into two disjoint parts which separately satisfy ϕ and ψ. In particular, there can be no aliasing between the two disjoint parts of the program state, which implies the soundness of the frame rule introduced in Section 2.2. Similarly, while $\phi \Rightarrow \psi$ means that every program state that satisfies ϕ also satisfies ψ, $\phi \mathrel{-\!\!\star} \psi$ means that the *extension* of the program state with any program state that satisfies ϕ yields a program state that satisfies ψ.

The magic wand is particularly useful for weakest-precondition reasoning, for example to express memory allocation [IO01; Rey02]. In much of the separation-logic literature as well as in the bulk of the tools, the magic wand is not considered or only partially supported, however, because its inclusion in a separation logic quickly leads to undecidability [BDL12; App14; BH15; SS15].

2.3.2 *The Semantics of Separation Logic*

THE STACK–HEAP MODEL. In this thesis, I use *stack–heap pairs* as models of SL formulas. While clearly not the only possible model of separation logics for reasoning about unbounded data structures, it has been the most-widely used model ever since the first papers on SL were written [IO01; Rey02]. A stack–heap pair is a pair of partial functions $(\mathfrak{s}, \mathfrak{h})$. The *stack* $\mathfrak{s}$ is a partial function from (program and logical) variables to some set(s) of values. The set(s) of values may be identical to or subsume the set of addressable *memory locations*. The *heap* $\mathfrak{h}$ is a partial function from memory locations to ordered sequences of values.

Let us formalize these definitions. Throughout this thesis, **Var** denotes a countable infinite set of variables with $\mathsf{nil} \in \mathbf{Var}$. Here, nil can be seen as an auxiliary constant that we can use to refer to the null pointer in separation-logic formulas.

We also need a countably infinite set of addressable memory locations **Loc** and a set of values **Val** with $\mathbf{Loc} \subseteq \mathbf{Val}$. Note that interpreting **Loc** by an infinite set is a deliberate abstraction from the (finite, e.g. 64 bit) address space of a real program. In a "shape-only" separation logic whose formulas constrain only the shape, not the content of the heap, we can simply set $\mathbf{Val} \triangleq \mathbf{Loc}$. This will be the case in Part iii. If we want to combine reasoning about shape and content, we also need to model the data stored in the heap, so **Val** will subsume both the locations and a data domain. This will be the case in Part ii.

A *stack* is a finite partial function $\mathfrak{s}\colon \mathbf{Var} \rightharpoonup \mathbf{Val}$. I implicitly assume $\mathsf{nil} \in \mathrm{dom}(\mathfrak{s})$ for every stack, but unless nil is relevant for an example at hand, I often do not explicitly include nil when defining stacks. A *heap* is a finite partial function $\mathfrak{h}\colon \mathbf{Loc} \rightharpoonup \mathbf{Val}^{+}$ that maps locations to

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

$$
\begin{array}{lll}
(\mathfrak{s},\mathfrak{h}) \models \mathbf{emp} & \text{iff} & \mathfrak{h} = \emptyset \\
(\mathfrak{s},\mathfrak{h}) \models x \approx y & \text{iff} & \mathfrak{s}(x) = \mathfrak{s}(y) \text{ and } \mathfrak{h} = \emptyset \\
(\mathfrak{s},\mathfrak{h}) \models x \not\approx y & \text{iff} & \mathfrak{s}(x) \neq \mathfrak{s}(y) \text{ and } \mathfrak{h} = \emptyset \\
(\mathfrak{s},\mathfrak{h}) \models x \mapsto \mathbf{y} & \text{iff} & \mathfrak{h} = \{\mathfrak{s}(x) \mapsto \mathfrak{s}(\mathbf{y})\} \\
(\mathfrak{s},\mathfrak{h}) \models \phi \star \psi & \text{iff} & \text{ex. } \mathfrak{h}_1, \mathfrak{h}_2 \text{ s.t. } \mathfrak{h}_1 \cap \mathfrak{h}_2 = \emptyset, \mathfrak{h} = \mathfrak{h}_1 \cup \mathfrak{h}_2, \\
& & (\mathfrak{s},\mathfrak{h}_1) \models \phi, \text{ and } (\mathfrak{s},\mathfrak{h}_2) \models \psi \\
(\mathfrak{s},\mathfrak{h}) \models \phi \mathrel{-\!\!\star} \psi & \text{iff} & \text{for all } \mathfrak{h}_0, \text{ if } \mathfrak{h}_0 \cap \mathfrak{h} = \emptyset \text{ and } (\mathfrak{s},\mathfrak{h}_0) \models \phi \\
& & \text{then } (\mathfrak{s},\mathfrak{h}_0 \cup \mathfrak{h}) \models \psi \\
(\mathfrak{s},\mathfrak{h}) \models \phi \wedge \psi & \text{iff} & (\mathfrak{s},\mathfrak{h}) \models \phi \text{ and } (\mathfrak{s},\mathfrak{h}) \models \psi \\
(\mathfrak{s},\mathfrak{h}) \models \phi \vee \psi & \text{iff} & (\mathfrak{s},\mathfrak{h}) \models \phi \text{ or } (\mathfrak{s},\mathfrak{h}) \models \psi \\
(\mathfrak{s},\mathfrak{h}) \models \neg\phi & \text{iff} & (\mathfrak{s},\mathfrak{h}) \not\models \phi \\
(\mathfrak{s},\mathfrak{h}) \models \exists x.\,\phi & \text{iff} & \text{ex. } \ell \in \mathbf{Loc} \text{ s.t. } (\mathfrak{s} \cup \{x \mapsto \ell\}, \mathfrak{h}) \models \phi \\
(\mathfrak{s},\mathfrak{h}) \models \forall x.\,\phi & \text{iff} & \text{for all } \ell \in \mathbf{Loc}, (\mathfrak{s} \cup \{x \mapsto \ell\}, \mathfrak{h}) \models \phi
\end{array}
$$

Figure 2.2: A semantics of first-order separation logic. I implicitly assume for all formulas ϕ that $\mathsf{fvars}(\phi) \subseteq \mathrm{dom}(\mathfrak{s})$, i.e., $(\mathfrak{s},\mathfrak{h}) \models \phi$ is undefined if $\mathsf{fvars}(\phi) \not\subseteq \mathrm{dom}(\mathfrak{s})$.

ordered sequences of values. By mapping to a sequence rather than a single location, the heap maps every allocated memory location to the entire structure allocated at this location. This is a fairly standard (but far from ubiquitous [Rey02; Cal+06]) abstraction from actual memory layout; it simplifies the model and does not lose precision as long as we are not interested in pointer arithmetic or other word-level operations. I write **Stacks** for the set of all stacks and **Heaps** for the set of all heaps.

A *model* is a stack–heap pair $(\mathfrak{s},\mathfrak{h})$ with $\mathfrak{s}(\mathsf{nil}) \notin \mathrm{dom}(\mathfrak{h})$, reflecting that the null pointer cannot be allocated.

SEMANTICS. I define a semantics for our basic first-order separation logic $\mathbf{SL_{base}}$ in Fig. 2.2. In the definition of the model relation $(\mathfrak{s},\mathfrak{h}) \models \phi$, I implicitly assume that all variables that occur in a formula ϕ are defined in the stack $\mathfrak{s}$, i.e., $(\mathfrak{s},\mathfrak{h}) \models \phi$ is undefined if $\mathsf{fvars}(\phi) \not\subseteq \mathrm{dom}(\mathfrak{s})$. This assumption is necessary because of my nonstandard choice to define stacks as finite partial functions $\mathfrak{s} \colon \mathbf{Var} \rightharpoonup \mathbf{Val}$. (On the other hand, using partial functions will simplify some of the development in Parts ii and iii.)

The empty-heap predicate **emp** holds iff the heap is empty; equalities and disequalities between variables hold iff the stack maps the variables to identical and different values, respectively. For (dis-)equalities, I additionally require that the heap is empty. This is nonstandard, but not unprecedented [PWZ13], and will simplify some of the development later in this thesis.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

A points-to assertion $x \mapsto \langle y_1, \ldots, y_k \rangle$ holds in a singleton heap that allocates only the location $\mathfrak{s}(x)$ and stores the values $\mathfrak{s}(y_1), \ldots, \mathfrak{s}(y_k)$ in this location. This is often called a *precise* [COY07] semantics, because the heap contains precisely the object described by the points-to assertion, nothing else.

The separating connectives are formalized in terms of disjoint unions of heap functions: in the standard semantics presented here, $(\mathfrak{s}, \mathfrak{h}) \models \phi \star \psi$ if and only if there exist domain-disjoint functions $\mathfrak{h}_1$, $\mathfrak{h}_2$ such that their point-wise union is $\mathfrak{h}$ and such that $(\mathfrak{s}, \mathfrak{h}_1) \models \phi$ and $(\mathfrak{s}, \mathfrak{h}_2) \models \psi$. I will argue in Chapter 3 and Part ii that there are other reasonable semantics for $\star$ that may simplify automated reasoning about the logic.

While the separating conjunction is about splitting the heap, the magic wand is about extending the heap: $(\mathfrak{s}, \mathfrak{h}) \models \phi \mathrel{-\!\!\star} \psi$ if all ways to extend $\mathfrak{h}$ with a disjoint model of ϕ yields a model of ψ.

The semantics of the Boolean operators and the quantifiers is standard; for simplicity I only allow quantifying over locations, not over arbitrary values. This is necessary because variables in $\mathbf{SL_{base}}$ (as well as the other logics in this thesis) are not sorted—their interpretation is not constrained to either **Loc** or **Val** $\setminus$ **Loc**—and it rarely makes sense for a single variable to range both over memory locations and over the domain of data values stored in the heap.

Example 2.1.

1. $(x \mapsto y) \star (y \mapsto \mathsf{nil})$ *states that the heap consists of exactly two objects, one pointed to by x, the other pointed to by y; that the object pointed to by x contains a pointer to the object pointed to by y; and that the object pointed to by y contains a null pointer. Put less precisely but more concisely, x points to y, y points to* nil, *and x and y are separate objects in the heap. The precise semantics of points-to assertions guarantees that there are no other objects in the heap.*

2. $(x \mapsto y) \wedge (z \mapsto y)$ *states that (1) the heap consists of a single object x that points to y and that simultaneously (2) the heap consists of a single object z that points to y. This formula is only satisfiable in stacks $\mathfrak{s}$ with $\mathfrak{s}(x) = \mathfrak{s}(z)$.*

3. $(x \mapsto y) \mathrel{-\!\!\star} (z \mapsto y)$ *states that after adding a pointer from x to y to the heap, we obtain a heap that contains a single pointer from z to y. This formula is only satisfiable in the empty heap and in stacks $\mathfrak{s}$ with $\mathfrak{s}(x) = \mathfrak{s}(z)$.*

4. $\forall x.\, (x \mapsto \mathsf{nil}) \mathrel{-\!\!\star} ((\neg \mathbf{emp}) \star (\neg \mathbf{emp}))$ *states that the heap contains at least one pointer: No matter which variable we allocate additionally, the resulting heap can be split into two nonempty parts, so the original heap must itself have been nonempty. We could, of course, state this property more succinctly as* $\neg\mathbf{emp}$.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

2.3.3 *Automated Reasoning about Separation Logic*

While program verification and –analysis are the motivation for studying separation logic, they are not the focus of this thesis. Instead, I mainly focus on *automated reasoning* about separation-logic formulas. Roughly speaking, such automated reasoning forms the backend of an automated verifier or static analyzer based on SL: the verifier or analyzer generates *queries*, i.e., SL formulas, which we would like to answer automatically. This leads to the following two decision problems.

- The *satisfiability problem*: given an SL formula ϕ, is there a stack-heap pair $(\mathfrak{s}, \mathfrak{h})$ that satisfies ϕ?
- The *entailment problem*: given two formulas ϕ, ψ, is it the case for all stack–heap pairs with $(\mathfrak{s}, \mathfrak{h}) \models \phi$ and $\mathrm{dom}(\mathfrak{s}) \supseteq \mathsf{fvars}(\phi) \cup \mathsf{fvars}(\psi)$ that $(\mathfrak{s}, \mathfrak{h}) \models \psi$ holds as well?

We already saw in Section 2.2 that the entailment problem is central to Hoare-style deductive verification: the rule of consequence involves two instances of the entailment problem. This is in contrast to most other Hoare-logic rules, which can be applied purely syntactically. Given this central role of *entailment checking* in deductive verification, it is not surprising that the entailment problem has received the bulk of the attention in the corner of the separation-logic literature that is concerned with automated deduction [BCO04; BDP11; Ene+14; Ene+17; IRS13; IRV14; PR13; MJN15; SI18; Sig+19; Ta+18; TK15; TNK19].

For the first-order separation logic $\mathbf{SL}_{\mathbf{base}}$ defined in this section, satisfiability and entailment are, of course, undecidable. After all, the logic can be seen as an extension of first-order predicate logic with binary relations [CYO01], which is already undecidable.[3]

Many fragments of separation logic with decidable entailment problems have been proposed and implemented [BCO04; IRS13; IRV14; PWZ13; PWZ14a; SI18; TK15; TNK19]; usually, these fragments restrict or disallow quantifiers, the magic wand, and negation. I attempt a more-or-less thorough overview of the "decidability landscape" in Chapter 14.

The symbolic-heap fragment

Much effort has been put into automated reasoning about the *symbolic-heap fragment*. This fragment is suitable as an abstract domain in program analyses [BCO05b] and expressive enough to write interesting specifications. At the same time, reasoning about symbolic heaps is tractable [C00+11] or at least decidable, even

---

3 It is less easy to see that the logic remains remains undecidable even if we only allow unary points-to assertions, i.e., points-to assertions of the form $x \mapsto \mathbf{y}$. See [BDL12] for a proof.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

when considering quite general data structures [IRS13]; as such, this fragment has also been the main focus of the *Competition of Solvers for Separation Logic* [Sig+19].

A symbolic heap is a formula of the form

$$\exists \underbrace{x_1, \ldots, x_k}_{k \geq 0} . \underbrace{\phi_{\mathsf{atom}} \star \cdots \star \phi_{\mathsf{atom}}}_{1 \text{ or more atoms}} .$$

Since negation is not allowed in symbolic heaps, the entailment problem for symbolic heaps is genuinely different from the satisfiability problem—it is impossible to solve an entailment $\phi \models \psi$ by checking the unsatisfiability of $\phi \wedge \neg\psi$, because the latter formula is not a symbolic heap.

This thesis can be seen as a search for the right trade-off between expressiveness and tractability—to find a separation logic in which we can express and prove interesting properties, and for which automation of the entailment problem is also feasible. Obviously, there is not the one right trade-off; depending on your goals and target applications, quite different trade-offs may be appropriate. For example, if you need a lightning fast bug finder, you may go for a logic that is not very expressive and try to solve queries in this logic with a set of fast but incomplete heuristics [Cal+11]. Conversely, if you're interested in building a deductive-verification tool for code annotated with contracts and loop invariants, you may prefer a more expressive logic that allows you to prove more complex properties of the program [Chi+12; Qiu+13; PWZ14b].

2.3.4 *Further Reading*

So far, I've provided you with an informal motivation for separation logic as well as a formal definition of a simple separation logic. If you are interested specifically in an overview of decision procedures for separation logic, you can have a look at Chapter 14.

If you're looking for a more thorough introduction to separation logic and its use in program specification and verification, I still recommend "the classic", John C. Reynold's paper [Rey02]. For a more up-to-date overview that recounts much of the development of the past two decades, you should have a look at O'Hearn's very recent article [OHe19]. I also recommend Christoph Matheja's PhD thesis [Mat20], which contains a very detailed and accessible introduction to the analysis and verification of heap-manipulating programs.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Wien Bibliothek Your knowledge hub

2.4 REASONING ABOUT STACK–HEAP MODELS: LOCATIONS, ISOMORPHISM, ALIASING

Several concepts appear again and again throughout Parts ii and iii: we need to distinguish between several types of locations; we need to reason about *isomorphism*; and we need to keep track of *aliasing*. I formalize these terms below. This section is meant as a reference section; you can skim it on a first reading. All notation introduced below also appear in the *List of Symbols* at the beginning of this document.

LOCATIONS. We will frequently have to reason about the locations in a heap. Let $\mathfrak{s} \in \mathbf{Stacks}$ be a stack and let $\mathfrak{h} \in \mathbf{Heaps}$ be a heap. A location ℓ is *labeled* iff $\ell \in \mathrm{img}(\mathfrak{s})$. We let $\mathsf{locs}(\mathfrak{h}) \triangleq \mathrm{dom}(\mathfrak{h}) \cup ((\bigcup \mathrm{img}(\mathfrak{h})) \cap \mathbf{Loc})$. A location $\ell \in \mathbf{Loc}$ is an *allocated location* in $\mathfrak{h}$ if $\ell \in \mathrm{dom}(\mathfrak{h})$ and a *referenced location* if $\ell \in \mathrm{img}(\mathfrak{h})$. Similarly, a variable x is allocated or referenced if $\mathfrak{s}(x)$ is allocated or referenced, respectively.

A location ℓ is *dangling* if $\ell \in \mathsf{locs}(\mathfrak{h}) \setminus \mathrm{dom}(\mathfrak{h})$ and a variable x is dangling if $\mathfrak{s}(x)$ is dangling. We collect all dangling locations of a heap $\mathfrak{h}$ in $\mathsf{dangling}(\mathfrak{h})$.

We define $\mathsf{alloced}(\mathfrak{s},\mathfrak{h}) \triangleq \{x \mid \mathfrak{s}(x) \in \mathrm{dom}(\mathfrak{h})\}$ and $\mathsf{refed}(\mathfrak{s},\mathfrak{h}) \triangleq \{x \mid \mathfrak{s}(x) \in \mathrm{img}(\mathfrak{h})\}$, i.e., the sets of *allocated variables* and *referenced variables* of the model $(\mathfrak{s},\mathfrak{h})$.

ISOMORPHISM. Two models are isomorphic if one can be transformed into the other by renaming locations.

Definition 2.2 (Isomorphic models)**.** *Let* $(\mathfrak{s},\mathfrak{h}), (\mathfrak{s}',\mathfrak{h}')$ *be models.* $(\mathfrak{s},\mathfrak{h})$ *and* $(\mathfrak{s}',\mathfrak{h}')$ *are* isomorphic, $(\mathfrak{s},\mathfrak{h}) \cong (\mathfrak{s}',\mathfrak{h}')$, *iff there exists a bijection*

$$\sigma \colon \mathbf{Val} \to \mathbf{Val}$$

such that

1. *for all* $v \in \mathbf{Val} \setminus \mathbf{Loc}$, $\sigma(v) = v$,
2. *for all* x, $\mathfrak{s}'(x) = \sigma(\mathfrak{s}(x))$, *and*
3. $\mathfrak{h}' = \{\sigma(\ell) \mapsto \sigma(\mathfrak{h}(\ell)) \mid \ell \in \mathrm{dom}(\mathfrak{h})\}$.

We call σ *an isomorphism.*

By demanding in Definition 2.2 that the bijection σ is the identity on $\mathbf{Val} \setminus \mathbf{Loc}$, we capture that the two heaps must agree on the data stored inside the allocated heap locations.

Neither the logic $\mathbf{SL}_{\mathsf{base}}$, nor the logics $\mathbf{SSL}^{+}_{\mathsf{data}}$ and $\mathbf{SLID}^{\exists\forall}_{\mathsf{btw}}$ studied in Parts ii and iii can distinguish between isomorphic models.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

ALIASING. In general, the stack may map multiple variables to the same location, i.e., variables may be aliases.

We sometimes need to pick a variable from sets of aliases. To make this choice consistently, I assume a linear order on **Var** and denote by $\max(\mathbf{v})$ the maximal variable among a finite set of variables $\mathbf{v}$ according to this order.

For example, we need the *stack-choice function* of stack $\mathfrak{s}$, a partial function that maps every location $\ell \in \text{img}(\mathfrak{s})$ to a variable x with $\mathfrak{s}(x) = \ell$.

Definition 2.3 (Stack inverse and stack-choice function)**.** *Let $\mathfrak{s}$ be a stack.*

1. *The function* $\mathfrak{s}^{-1} \triangleq \{l \mapsto \{x \mid \mathfrak{s}(x) = l\}\}$ *is the* stack inverse
2. *The function* $\mathfrak{s}^{-1}_{\max} \triangleq \{l \mapsto \max(\mathfrak{s}^{-1}(l))\}$ *is the* stack-choice function *of* $\mathfrak{s}$.

Stack aliasing induces an equivalence relation, the *stack-aliasing constraint* of $\mathfrak{s}$ given by

$$\mathsf{aliasing}(\mathfrak{s}) \triangleq \{(x,y) \mid x,y \in \text{dom}(\mathfrak{s}) \text{ and } \mathfrak{s}(x) = \mathfrak{s}(y)\}\,.$$

The equivalence class in $\mathsf{aliasing}(\mathfrak{s})$ of variable x is

$$[x]^{\mathfrak{s}}_{=} \triangleq \{y \mid \mathfrak{s}(y) = \mathfrak{s}(x)\}\,.$$

Just like with other functions, I sometimes apply $[\cdot]^{\mathfrak{s}}_{=}$ to sequences in a point-wise manner, i.e., write $[\langle y_1, \dots, y_k\rangle]^{\mathfrak{s}}_{=}$ as shorthand for $\langle [y_1]^{\mathfrak{s}}_{=}, \dots, [y_k]^{\mathfrak{s}}_{=}\rangle$. The set of all equivalence classes of $\mathfrak{s}$ is

$$\mathsf{classes}(\mathfrak{s}) \triangleq \{[x]^{\mathfrak{s}}_{=} \mid x \in \text{dom}(\mathfrak{s})\}\,.$$

Finally, the following auxiliary notation will simplify reasoning about stack aliasing.

- I frequently denote a fixed stack-aliasing constraint by Σ.
- The *domain* of stack-aliasing constraint Σ are all variables on which Σ is defined, i.e., $\text{dom}(\Sigma) \triangleq \{x \mid (x,x) \in \Sigma\}$.
- The *restriction* of stack-aliasing constraint Σ to the variables $\mathbf{y}$ is $\Sigma|_{\mathbf{y}} \triangleq \Sigma \cap (\mathbf{y} \times \mathbf{y})$.
- $\mathbf{AC}^{\mathbf{x}} \triangleq \{\mathsf{aliasing}(\mathfrak{s}) \mid \text{dom}(\mathfrak{s}) = \mathbf{x}\}$ is the set of all stack-aliasing constraints over variables $\mathbf{x}$.

Observe that $\mathbf{AC}^{\mathbf{x}}$ is finite if $\mathbf{x}$ is finite—Its size is given by the $|\mathbf{x}|$-th Bell number.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

3 CONTRIBUTIONS & OVERVIEW

This thesis consists of two main technical parts. In each of these parts, I study the satisfiability and entailment problem of a variant of separation logic, prove (un-)decidability and complexity results, and develop decision procedures.

I briefly summarize the main contributions and then continue with a more detailed overview of both parts.

STRONG-SEPARATION LOGIC (PART II). While the magic wand is clearly useful—for example, for reasoning about memory allocation in weakest-precondition calculi—it is too powerful: separation logics that allow both the magic wand and inductive predicates are generally undecidable. For example, propositional separation logic with the singly-linked list predicate is undecidable [DLM18].

In this thesis, I show how to obtain a PSPACE-decidable separation logic with magic wand, negation, and inductive predicates for lists and trees. The key to the decidability result is assigning a more restrictive but natural semantics to the separating connectives. I call this more restrictive notion of separation *strong separation* and the resulting logic *strong-separation logic*.

DATA PREDICATES (PART II). Most decidability results for separation logics are limited to "shape-only" specifications, which describe the shape of the program heap, but not its content. In a shape-only specification, I can express that the heap contains a list, but not that this list is sorted.

In this thesis, I introduce a novel way of mixing reasoning about shape and data: I introduce a family of data predicates that can be passed as additional arguments to inductive predicates to constrain the data stored inside the data structures. I show that the resulting logic is expressive enough to axiomatize common data structures (such as sorted lists, binary search trees, max heaps, etc.), while retaining good computational properties. Specifically, entailment in an extension of the symbolic-heap fragment of the logic is decidable in coNP.

INDUCTIVE DEFINITIONS (PART III). In many variants of separation logic, systems of recursive equations can be used to define custom data structures. Among these, "shape-only" symbolic-heap separation logic with inductive definitions, a generalization of the original symbolic-heap fragment [BCO04; BCO05b], has

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

received a lot of attention [BDP11; Ene+14; IRV14; SI18; Ta+18; TK15; TNK19; Sig+19]. Entailment in this logic is undecidable in general [Ant+14], but a large fragment, which I denote $\mathbf{SLID}_{\mathsf{btw}}$ in this thesis, was shown decidable via reduction to monadic second-order logic by Iosif et al. [IRS13]. Unlike most other decidable separation logics, this fragment allows definitions of data structures "beyond trees," i.e., of data structures that are neither lists or trees. For example, binary trees with linked leaves can be defined in $\mathbf{SLID}_{\mathsf{btw}}$.

Iosif et al. [IRS13] did not propose a practical decision procedure for $\mathbf{SLID}_{\mathsf{btw}}$; the authors remarked in the follow-up paper [IRV14] that "the method from [IRS13] causes a blowup of several exponentials in the size of the input problem and is unlikely to produce an effective decision procedure." To circumvent this blowup, I develop a direct, model-theoretic decision procedure for an extension of $\mathbf{SLID}_{\mathsf{btw}}$. This decision procedure proved to have potential in practice [Sig+19], but takes double-exponential time in the size of the input in the worst case. In light of a recent 2EXPTIME-hardness proof [EIP20], we now know that this asymptotic behavior is unavoidable.

Let us look at each of these three contributions in more detail.

3.1 STRONG-SEPARATION LOGIC

Strong-separation logic (SSL) is the main object of study in Part ii of the thesis. The hyphen is important here. I do not claim to have made separation logic strong. Rather, I propose a logic with a more restrictive notion of separation than is commonly used—which yields stronger guarantees on the way a model of a separating conjunction $\phi_1 \star \phi_2$ can be decomposed into models of ϕ_1 and ϕ_2.

In this section, I give an informal introduction to SSL.[1] All formal details follow in Part ii.

STRONG SEPARATION. As explained in Chapter 2, the separating conjunction $\star$ is at the heart of the success of SL, because it supports concise statements about the disjointness of resources and enables compositional specification and verification.

The standard semantics of $\star$ allows splitting a heap into two arbitrary sub-heaps. The magic-wand operator $-\!\!\star$, which is the right adjoint of $\star$, allows extension by arbitrary heaps. The possibility of such arbitrary splits and extensions makes reasoning about SL formulas difficult, and quickly renders separation logic undecidable when inductive predicates for data structures are considered. For example,

1 Florian Zuleger contributed significantly to the informal introduction to strong-separation logic.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Wien Bibliothek Your knowledge hub

Demri et al. showed that adding only the singly-linked list-segment predicate to propositional separation logic (i.e., with $\star$, $\mathrel{-\!\!\star}$ and classical connectives $\wedge, \vee, \neg$) leads to undecidability [DLM18].

Most SL specifications used in automated verification do, however, not make use of arbitrary heap (de-)compositions. For example, as far as I am aware, all decidable variants of the widely used symbolic-heap fragments of separation logic (see e.g., [BCO04; BCO05b; Coo+11; IRS13; IRV14; MJN15; TK15; TNK19]) have the following property. To show that a model of a symbolic heap satisfies a separating conjunction, it is always sufficient to split the model at the locations that interpret program variables (for quantifier-free symbolic heaps) and logical variables (for existentially-quantified symbolic heaps).

This observation motivates a more restrictive separating conjunction that allows splitting the heap only at locations pointed to by variables.

The difference between the standard, "weak" semantics and the "strong" semantics is best explained by example. Let $(\mathfrak{s}, \mathfrak{h})$ be a stack–heap pair as introduced in Chapter 2. A *dangling pointer* is a variable x such that $\mathfrak{s}(x) \in \text{img}(\mathfrak{h}) \setminus \text{dom}(\mathfrak{h})$, i.e., a pointer variable that is the target of a pointer in the heap, but is not itself allocated.

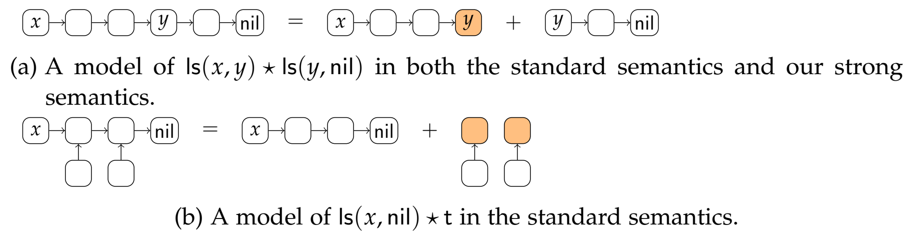
Figure 3.1a shows a graphical representation of a stack–heap pair $(\mathfrak{s}, \mathfrak{h})$ that satisfies the formula $\mathsf{ls}(x, y) \star \mathsf{ls}(y, \mathsf{nil})$. Here, ls denotes the list-segment predicate. As shown in Fig. 3.1a, $\mathfrak{h}$ can be split into two disjoint parts $\mathfrak{h}_1$ and $\mathfrak{h}_2$ such that $(\mathfrak{s}, \mathfrak{h}_1)$ is a model of $\mathsf{ls}(x, y)$ and $(\mathfrak{s}, \mathfrak{h}_2)$ is a model of $\mathsf{ls}(y, \mathsf{nil})$. Now, the sub-heap $\mathfrak{h}_1$ has a dangling pointer with target $\mathfrak{s}(y)$ (displayed with an orange background), while no pointer is dangling in the heap $\mathfrak{h}$.

The standard semantics of $\star$ allows splitting a heap into two arbitrary sub-heaps, which may result in the introduction of arbitrary dangling pointers into the sub-heaps. Note, however, that the introduction of dangling pointers is *not* arbitrary when splitting the models of $\mathsf{ls}(x, y) \star \mathsf{ls}(y, \mathsf{nil})$; there is only one way of splitting the models of this formula, namely at the location of program variable y.

Standard SL semantics also allows the introduction of dangling pointers without the use of variables. Fig. 3.1b shows a model of $\mathsf{ls}(x, \mathsf{nil}) \star \mathsf{t}$—assuming the standard semantics. Here, the formula t (for *true*) stands for any arbitrary heap. In particular, this includes heaps with arbitrary dangling pointers into the list segment $\mathsf{ls}(x, \mathsf{nil})$. This power of introducing arbitrary dangling pointers is what is used by Demri et al. for their undecidability proof of propositional separation logic with the singly-linked list-segment predicate [DLM18].

Using such a semantics of $\phi_1 \star \phi_2$ allows us to make stronger assumptions about the ways that the models of ϕ_1 and ϕ_2 may overlap. For this reason, Florian Zuleger and I decided to call the resulting logic strong-separation logic [PZ20b]. Strong-separation logic (SSL) shares many properties with standard separation-logic semantics. For example, the models of SSL formulas form a *separation algebra* [COY07], which

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN



(a) A model of $\mathsf{ls}(x, y) \star \mathsf{ls}(y, \mathsf{nil})$ in both the standard semantics and our strong semantics.

(b) A model of $\mathsf{ls}(x, \mathsf{nil}) \star \mathsf{t}$ in the standard semantics.

Figure 3.1: Two models and their decomposition into disjoint submodels. An orange background highlights the dangling pointers of submodels.

guarantees the soundness of the standard *frame rule* of separation logic [Rey02]. Consequently, SSL can be used instead of standard SL in a wide variety of (semi-)automated analyzers and verifiers, including Hoare-style verification [IO01; Rey02], symbolic execution [BCO05b], and Infer-style bi-abductive analyses [Cal+11; Cal+15].

At the same time, SSL has much better computational properties than standard SL—especially when formulas contain expressive features such as the magic wand, $-\!\!\star$, or negation. Specifically, I will present a PSpace decision procedure for the satisfiability problem of full propositional SSL with the singly-linked list-segment predicate, singly-linked tree predicate, and a notion of relational data predicates (discussed further below). Note that this matches the complexity of propositional separation logic without *any* inductive predicates, i.e., limited to points-to assertions [CYO01].

The PSpace result is in stark contrast to the aforementioned undecidability result obtained by Demri et al. [DLM18] for the logic without trees or data predicates, but assuming the standard semantics of the separating conjunction.

The decidability result makes SSL a very promising formalism for automating Hoare-style verification, as we can automatically discharge the verification conditions (VC) generated by strongest-postcondition and weakest-precondition calculi. This is much harder to achieve with standard SL; to quote from [App14, p. 131]: “VC-generators do not work especially well with separation logic, as they introduce magic-wand $-\!\!\star$ operators which are difficult to eliminate.”

3.2 DATA PREDICATES

I will also define an extension of SSL in which inductive predicates can be annotated with *data predicates*. Data predicates make it possible to specify universal and existential properties of the content of data structures. This allows us to go beyond memory correctness to automatically prove functional-correctness properties of programs with recursive data structures.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Data predicates can be unary or binary, and they can be existential or universal, for a total of four "flavors" of data predicates. A unary data predicate asserts that the data structure contains a value that satisfies the predicate (existential) or that all values in the data structure satisfy the predicate (universal). By convention, the variable α occurs in all unary data predicates and ranges over the data stored in the data structure. For example, consider the formula

$$\mathsf{ls}(x, [\alpha \approx c]^{\exists}) \star \mathsf{ls}(y, [\alpha \not\approx c]^{\forall}),$$

where c is a fresh constant. The formula expresses that the lists have different contents: the list with head x contains the data value c, as specified by the existential predicate $[\alpha \approx c]^{\exists}$. You should read it as "if I instantiate α with all data values stored in the list with head x, I will find at least one value d that satisfies $d \approx c$." Similarly, the list with head y does not contain c, as specified by the universal predicate $[\alpha \not\approx c]^{\forall}$, to be read as "all values d stored in the data structure satisfy the formula $d \not\approx c$."

A binary data predicate has two variables that are instantiated with the data stored in the data structure, α and β. Moreover, it always contains a *field identifier*. For example, the universal binary data predicate $[\mathsf{l}\colon \beta < \alpha]^{\forall}$ can be read as "for all pairs of locations ℓ_1, ℓ_2 in, if ℓ_2 can be reached from ℓ_1 by first dereferencing the field l (for left), if ℓ_2 stores d_2 and ℓ_1 stores d_1, then $d_2 < d_1$ holds." Put more simply, all values stored in the left subtree of ℓ_1 (for arbitrary tree location ℓ_1) are smaller than the value stored in ℓ_1. Analogously, $[\mathsf{r}\colon \beta > \alpha]^{\forall}$ expresses that all values in the right subtree (specified via the field identifier r) of a tree location ℓ are larger than the value stored in ℓ.

By combining the two predicates, we can thus specify that a tree is a binary search tree. To give a concrete example,

$$\mathsf{tree}(x, \left\{[\mathsf{l}\colon \beta < \alpha]^{\forall}, [\mathsf{r}\colon \beta > \alpha]^{\forall}\right\})$$

asserts that x is the root of a binary search tree defined over fields l and r.

The purpose of these examples was to give you a rough intuition for what you can express with data predicates in SSL, so it does not matter if you didn't get all the details. You will find more examples as well as the formal syntax and semantics of data predicates in Part ii. There, I also present a decision procedure of a fragment of the logic with data predicates but without magic wands or unguarded negations.

In Chapter 14, I will compare SSL with data predicates to other heap logics that support some form of data constraints.

3.3 INDUCTIVE DEFINITIONS

Initially, separation logic was used for manual proofs of program correctness [IO01; Yan01; Rey02]. The (unsurprising) undecidability

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

of first-order separation logic and a few decidability results were established early on [CYO01], but it took several more years until Berdine et al. first showed the applicability of SL-based reasoning in practice with the development of `Smallfoot` [BCO05a]. `Smallfoot` uses a fragment of separation-logic as symbolic states in symbolic execution [BCO05b]. This fragment is since known as the *symbolic-heap fragment* of separation logic (see also Section 2.3.3).

Many tools, including `Smallfoot`, have hardcoded support for a small number of data structures, typically variants of lists or binary trees [BCO04; Coo+11; PR13; PWZ13; PWZ14a]. There has, however, been significant theoretical [Ant+14; IRS13] and practical [Sig+19] work on the symbolic-heap fragment with user-defined inductive definitions. In this fragment, the user of the logic can define their own custom data structures by providing a set of recursive equations that I call *systems of inductive definitions* (SIDs) and define formally in Chapter 8. The formulas in the equations are themselves symbolic heaps. The semantics of the user-defined predicates is then given by the least fixed point of the equation system. For example, the equations

$$\begin{aligned} \mathsf{lseg}(x_1, x_2) &\Leftarrow x_1 \mapsto x_2 \\ \mathsf{lseg}(x_1, x_2) &\Leftarrow \exists y.\, (x_1 \mapsto y) \star \mathsf{lseg}(y, x_2) \end{aligned}$$

define a binary predicate lseg whose models are all nonempty singly-linked list segments.

If arbitrary symbolic heaps are allowed in SIDs, the entailment problem of the symbolic-heap fragment with inductive definitions is undecidable [Ant+14]. To obtain a decidable logic, it is thus necessary to restrict the inductive definitions; for example, restricting them to linear structures [GCW16] or tree structures [TK15; TNK19; IRV14].

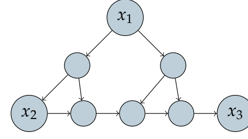
It is, however, also possible to go beyond trees. Iosif et al. [IRS13] proposed to restrict SIDs in a way that guarantees that all models of the user-defined predicates (when viewed as graphs) have bounded treewidth. I denote this fragment of SIDs by $\mathbf{ID}_{\mathsf{btw}}$ in this thesis. The restriction to $\mathbf{ID}_{\mathsf{btw}}$ made it possible to solve the entailment problem for symbolic heaps via a reduction to the satisfiability problem of monadic second-order logic (MSO) over graphs of bounded treewidth (BTW), which can be encoded into MSO over trees [Cou90], the decidability of which is a classical result by Rabin [Rab69].

In $\mathbf{ID}_{\mathsf{btw}}$ we can, for example, define a predicate for trees with linked leaves (TLL), which can be used to implement an efficient sorted-set data structure. I define such a predicate and display a small TLL in Fig. 3.2.

The satisfiability problem of MSO over graphs of BTW is nonelementary, but the reduction in [IRS13] has a bounded number of quantifier alternations, leading to an elementary recursive decision procedure that is, however, still several times exponential [IRV14].

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

$$\begin{aligned} \mathsf{tll}(x_1,x_2,x_3) &\Leftarrow (x_1 \mapsto \langle \mathsf{nil},\mathsf{nil},x_3\rangle) \star (x_1 \approx x_2) \\ \mathsf{tll}(x_1,x_2,x_3) &\Leftarrow \exists\langle l,r,m\rangle\,.\,(x_1 \mapsto \langle l,r,\mathsf{nil}\rangle) \\ &\qquad \star\, \mathsf{tll}(l,x_2,m) \\ &\qquad \star\, \mathsf{tll}(r,m,x_3) \end{aligned}$$



(a) The definition of the tll predicate representing trees with linked leaves.

(b) A model of $\mathsf{tll}(x_1,x_2,x_3)$.

Figure 3.2: A system of inductive definition (SID) defining trees with linked leaves. This SID is in the fragment $\mathbf{ID}_{\mathsf{btw}}$ introduced in [IRS13].

My primary goal in Part iii is to develop a more practical decision procedure for separation logic with inductive definitions from $\mathbf{ID}_{\mathsf{btw}}$. I present a direct model-theoretic decision procedure, as opposed to a reduction to a different formalism. This decision procedure runs in double-exponential time in the worst case. Granted, this does not sound all that practical; fortunately, the performance in practice is much better than the asymptotic complexity would suggest [Sig+19]. Moreover, there is no hope to obtain a decision procedure with better asymptotic behavior: The entailment problem for $\mathbf{ID}_{\mathsf{btw}}$ was recently shown to be hard for 2ExpTime [EIP20].

I also study whether we can allow arbitrary definitions from $\mathbf{ID}_{\mathsf{btw}}$, but go beyond the symbolic-heap fragment. I show that it is possible to allow classical conjunction and disjunction, as well as *guarded* variants of negation, magic wand, and septraction, i.e., formulas of the form $\phi \wedge \neg\psi$, $\phi \wedge (\psi_1 \mathrel{-\!\!\star} \psi_2)$ and $\phi \wedge (\psi_1 \mathrel{-\!\!\circledast} \psi_2)$. I also show that allowing any of these three operators without guards yields an undecidable logic. This leaves us with an almost complete picture of the decidability landscape for separation logics with $\mathbf{ID}_{\mathsf{btw}}$ definitions.

3.4 ABSTRACTION-BASED SATISFIABILITY CHECKING

As discussed earlier in this chapter, both Part ii and Part iii are centered around a decision procedure for a variant of SL. Both of these decision procedures are based on a *compositional abstraction* that *refines* the satisfaction relation of the respective SL variant. In this section, I briefly discuss this approach in a generic setting.

Let **SL** be a variant of separation logic. We are looking for a finite abstract domain $\mathbb{A}$ and an abstraction function $\mathsf{abst}\colon \mathbf{Stacks} \times \mathbf{Heaps} \to \mathbb{A}$ with the following key properties.

REFINEMENT. If $\mathsf{abst}(\mathfrak{s},\mathfrak{h}_1) = \mathsf{abst}(\mathfrak{s},\mathfrak{h}_2)$, then $(\mathfrak{s},\mathfrak{h}_1)$ and $(\mathfrak{s},\mathfrak{h}_2)$ satisfy the same **SL** formulas, so the equivalence relation induced by the abstraction function,

$$(\mathfrak{s},\mathfrak{h}_1) \equiv_{\mathsf{abst}} (\mathfrak{s},\mathfrak{h}_2) \text{ iff } \mathsf{abst}(\mathfrak{s},\mathfrak{h}_1) = \mathsf{abst}(\mathfrak{s},\mathfrak{h}_2),$$

refines the satisfaction relation.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

COMPOSITIONALITY. There exists an operation $\bullet\colon \mathbb{A} \times \mathbb{A} \rightharpoonup \mathbb{A}$ such that $\mathsf{abst}(\mathfrak{s}, \mathfrak{h}_1 \sqcup \mathfrak{h}_2) = \mathsf{abst}(\mathfrak{s}, \mathfrak{h}_1) \bullet \mathsf{abst}(\mathfrak{s}, \mathfrak{h}_2)$, where $\sqcup$ is the notion of disjoint union used to give the semantics to the separating conjunction. (This notion of disjoint union will differ between Parts ii and iii.)

Refinement and compositionality make it possible to go from the function $\mathsf{abst}\colon \mathbf{Stacks} \times \mathbf{Heaps} \to \mathbb{A}$ to a function

$$\mathsf{abst}_{\mathbf{SL}}\colon \mathbf{SL} \to \mathbb{A}, \quad \phi \mapsto \{\mathsf{abst}(\mathfrak{s}, \mathfrak{h}) \mid (\mathfrak{s}, \mathfrak{h}) \models \phi\},$$

because

1. if $a \in \mathsf{abst}_{\mathbf{SL}}(\phi_1) \cap \mathsf{abst}_{\mathbf{SL}}(\phi_2)$ then $a \in \mathsf{abst}_{\mathbf{SL}}(\phi_1 \wedge \phi_2)$ (by refinement), and
2. if $a_1 \in \mathsf{abst}_{\mathbf{SL}}(\phi_1)$ and $a_2 \in \mathsf{abst}_{\mathbf{SL}}(\phi_2)$ then $a_1 \bullet a_2 \in \mathsf{abst}_{\mathbf{SL}}(\phi_1 \star \phi_2)$ (by compositionality).

Provided we come up with a way to compute the abstraction of atomic formulas, we can then use $\mathsf{abst}_{\mathbf{SL}}$ for satisfiability checking: $\mathsf{abst}_{\mathbf{SL}}(\phi) \neq \emptyset$ iff ϕ is satisfiable.

Table 3.1 points you to the key definitions and results of Parts ii and iii. Besides the results explained above, I've included a few additional key lemmas.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

TU Bibliothek WIEN Your knowledge hub
Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.

	Part ii	Part iii
Abstract domain	AMS (Definition 6.9)	Φ-type (Definition 11.17)
Size of domain	Lemma 7.4	Lemma 11.39
SL Variant	**SSL**	$\mathbf{SLID}^{g}_{btw}$
Compositionality	Lemma 6.26	Corollary 11.32
Decomposability	Lemma 6.27	Lemma 12.7
Refinement	Theorems 6.28 and 6.38	Theorem 12.10
Computability	Fig. 6.4	Predicates: Fig. 12.1
		$\mathbf{SLID}^{g}_{btw}$: Fig. 12.2
Decidability	Theorem 6.45	Theorem 12.30

Table 3.1: Key definitions and results in Parts ii and iii.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.

4 HEAPS AS DIRECTED GRAPHS

Throughout this thesis, I take a model-theoretic perspective: all decision procedures in this thesis are based on abstracting or encoding models, rather than designing proof systems. When thinking about stack–heap pairs, it is often useful to view them as directed graphs.

FROM STACK–HEAP MODELS TO GRAPHS. All of the technical development in this thesis is, to a varying degree, informed by viewing the program heap as a *directed graph* whose nodes are the heap locations and whose edges are the pointers between the heap locations.

Sometimes, I don't care about the order of the pointers, i.e., about the order of the sequences in $\mathrm{img}(\mathfrak{h})$. In these cases, the following notion of directed graphs is sufficient.

Definition 4.1 (Directed graph)**.** *Let $V_\mathcal{G} \subseteq \mathbf{Loc}$ be a finite set of locations and $E_\mathcal{G} \subseteq V_\mathcal{G} \times V_\mathcal{G}$. The pair $\mathcal{G} = \langle V_\mathcal{G}, E_\mathcal{G} \rangle$ is a* directed graph.

When I do care about the order in $\mathrm{img}(\mathfrak{h})$, I label the edges of the directed graph according to this order, obtaining a *directed indexed graph*.

Definition 4.2 (Directed indexed graph)**.** *A* directed indexed graph *is a pair $\mathcal{G} = \langle V_\mathcal{G}, E_\mathcal{G} \rangle$, where $V_\mathcal{G} \subseteq \mathbf{Loc}$ is a finite set of locations and $E_\mathcal{G} \subseteq \mathbf{Loc} \times \mathbb{N} \times \mathbf{Loc}$.*

In both types of graphs, $V_\mathcal{G}$ are the *nodes* of the graph and $E_\mathcal{G}$ are the *edges* of the graph.

Let $\mathfrak{h}$ be a heap in the sense of Section 2.3.2 and recall from Section 2.4 that $\mathsf{locs}(\mathfrak{h})$ denotes the locations in the domain and image of the heap. The function graph of $\mathfrak{h}$ induces a directed graph as follows.

Definition 4.3 (Induced graphs)**.** *Let $\mathfrak{h} \colon \mathbf{Loc} \rightharpoonup \mathbf{Val}^*$ be a heap. The* induced graph *and* induced indexed graph *of $\mathfrak{h}$ are given by*

$$\mathsf{graph}(\mathfrak{h}) \triangleq \langle \mathsf{locs}(\mathfrak{h}), \bigcup \{ \{ \langle \ell_0, v \rangle \mid v \in \mathfrak{h}(\ell_0) \cap \mathbf{Loc} \} \mid \ell_0 \in \mathrm{dom}(\mathfrak{h}) \} \rangle$$

$$\mathsf{igraph}(\mathfrak{h}) \triangleq \Big\langle \mathsf{locs}(\mathfrak{h}), \bigcup \{ \{ \langle \ell_0, i, v_i \rangle \mid 1 \leq i \leq k, v_i \in \mathbf{Loc} \} \mid \ell_0 \in \mathrm{dom}(\mathfrak{h}), \mathfrak{h}(\ell_0) = \langle v_1, \ldots, v_k \rangle \} \Big\rangle$$

If $\mathsf{igraph}(\mathfrak{h}) = \langle V_\mathcal{G}, E_\mathcal{G} \rangle$ and $\langle \ell_1, i, \ell_2 \rangle \in E_\mathcal{G}$, then the i-th field of the structure allocated at location ℓ_1 in the heap is a pointer to location ℓ_2. The restriction to $\mathbf{Loc}$ in the induced graphs guarantees that we

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

only include locations, not data (i.e., elements from **Val** $\setminus$ **Loc**) in the induced graphs. The nodes of the induced graphs are all memory locations that are allocated or referenced in $\mathfrak{h}$; the edges of the induced graphs are all pointers between memory locations.

Example 4.4 (Induced graph). *Assume* $\mathbf{Loc} = \mathbb{N}$ *and* $\mathbf{Val} = \mathbf{Loc} \cup \{true, false\}$ *let* $\mathfrak{h} \triangleq \{1 \mapsto \langle 2, 3, true \rangle, 2 \mapsto \langle 0, 0, false \rangle, 3 \mapsto 4\}$.

1. $\mathsf{graph}(\mathfrak{h}) = \langle \{0, 1, 2, 3, 4\}, \{\langle 1, 2 \rangle, \langle 1, 3 \rangle, \langle 2, 0 \rangle, \langle 3, 4 \rangle\} \rangle$,
2. $\mathsf{igraph}(\mathfrak{h}) = \Big\langle \{0, 1, 2, 3, 4\}, \{\langle 1, 1, 2 \rangle, \langle 1, 2, 3 \rangle, \langle 2, 1, 0 \rangle, \langle 2, 2, 0 \rangle, \langle 3, 1, 4 \rangle\} \Big\rangle$.

REASONING ABOUT DIRECTED GRAPHS. Later in the thesis, we will need several graph-theoretic concepts like reachability, paths, cycles, etc. We will also need to reason about specific classes of graphs, such as trees. In the remainder of this section, I formalize these concepts. You can treat this section as a reference section and skim it on a first reading.

Definition 4.5 (Reachability and paths). *Let* $\mathcal{G} = \langle V_{\mathcal{G}}, E_{\mathcal{G}} \rangle$ *be a directed indexed graph.*

- $\ell_1 \xrightarrow{i}_{\mathcal{G}} \ell_2$ *holds iff* $\langle \ell_1, i, \ell_2 \rangle \in E_{\mathcal{G}}$.
- $\ell_1 \rightarrow_{\mathcal{G}} \ell_2$ *holds iff there exists an* i *such that* $\ell_1 \xrightarrow{i} \ell_2$ *holds.*
- $\ell_1 \xrightarrow{*}_{\mathcal{G}} \ell_2$ *holds if there exist* $\ell'_1, \dots, \ell'_k$, $k \geq 1$, *such that* $\ell_1 = \ell'_1$, $\ell_2 = \ell'_k$, *and for all* $1 \leq j < k$, $\ell'_j \rightarrow \ell'_{j+1}$ *holds. In this case, we say that* ℓ_2 *is* reachable *from* ℓ_1. *If* $\ell'_1, \dots, \ell'_k$ *are pairwise different, they are a* path *of* length $k - 1$.
- $\ell_1 \xrightarrow{+}_{\mathcal{G}} \ell_2$ *holds if there exists an* ℓ' *such that* $\ell_1 \rightarrow_{\mathcal{G}} \ell'$ *and* $\ell' \xrightarrow{*}_{\mathcal{G}} \ell_2$.
- $\ell_1 \xrightarrow{i*}_{\mathcal{G}} \ell_2$ *holds if there exist an* ℓ' *with* $\ell_1 \xrightarrow{i}_{\mathcal{G}} \ell'$ *and* $\ell' \xrightarrow{*}_{\mathcal{G}} \ell_2$.

We omit $\mathcal{G}$ *when it is clear from the context, writing, for example,* $\ell_1 \xrightarrow{+} \ell_2$ *instead of* $\ell_1 \xrightarrow{+}_{\mathcal{G}} \ell_2$.

Definition 4.6 ((A-)cyclicity). *Let* $\mathcal{G}$ *be a directed graph.* $\mathcal{G}$ *is* acyclic *if there does* not *exist a location* $\ell \in V_{\mathcal{G}}$ *with* $\ell \xrightarrow{+}_{\mathcal{G}} \ell$. *If* $\mathcal{G}$ *is not acyclic, we say that* $\mathcal{G}$ contains cycles.

Definition 4.7 (Source, sink, predecessor, successor). *Let* $\mathcal{G} = \langle V_{\mathcal{G}}, E_{\mathcal{G}} \rangle$ *be a directed graph and* $\ell \in V_{\mathcal{G}}$.

1. ℓ *is a* source *of* $\mathcal{G}$ *if* $\ell \rightarrow_{\mathcal{G}} \ell'$ *for some* $\ell' \in V_{\mathcal{G}}$, *but* $\ell' \not\rightarrow_{\mathcal{G}} \ell$ *for all* $\ell' \in V_{\mathcal{G}}$.
2. ℓ *is a* sink *of* $\mathcal{G}$ *if* $\ell' \rightarrow_{\mathcal{G}} \ell$ *for some* $\ell' \in V_{\mathcal{G}}$, *but* $\ell \not\rightarrow_{\mathcal{G}} \ell'$ *for all* $\ell' \in V_{\mathcal{G}}$.

TU Bibliothek Your knowledge hub WIEN
Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.

3. ℓ_2 *is a* successor *of* ℓ_1 *if* $\ell_1 \rightarrow_{\mathcal{G}} \ell_2$.

4. ℓ_2 *is a* predecessor *of* ℓ_1 *if* $\ell_2 \rightarrow_{\mathcal{G}} \ell_1$.

I next formalize the two key data structures of Part ii, singly-linked list segments and directed trees. In the definition of directed trees, I include a dedicated *terminal* node representing the null pointer. This node is special, because it is the only tree node that can have multiple predecessors.

Definition 4.8 (Lists, trees)**.** *Let $\mathcal{G} = \langle V_{\mathcal{G}}, E_{\mathcal{G}} \rangle$ be a directed indexed graph.*

1. *$\mathcal{G}$ is a* singly-linked list *if $\mathcal{G}$ has a single source ℓ, every node of $\mathcal{G}$ is reachable from ℓ, and every node of $\mathcal{G}$ has at most one successor. The* length *of the list is given by $|V_{\mathcal{G}}| - 1$.*

2. *$\mathcal{G}$ is a* directed tree with root ℓ and terminal ℓ' *iff (1) $\mathcal{G}$ is acyclic, (2) ℓ is the unique source of $\mathcal{G}$, (3) $\ell \xrightarrow{*} \ell''$ for all $\ell'' \in V_{\mathcal{G}}$, (4) if $\ell' \in V_{\mathcal{G}}$ then ℓ' is a sink of $\mathcal{G}$, and (5) every node except ℓ' has at most one predecessor. The* depth *of a tree is the length of the longest path in the tree.*

ACCESS PATH ORDERING. Later, we will need to order the sinks of a tree from left to right. To this end, we introduce an ordering $\prec$ on the nodes of a tree based on the fields we have to follow in the tree to get to the nodes. Specifically, $\ell \prec \ell'$ in $\mathcal{G}$ iff ℓ precedes ℓ' in a depth-first preorder traversal of the tree $\mathcal{G}$. We use $\prec$ to define the aforementioned left-to-right ordering of the sinks of a tree.

Definition 4.9 (Access path ordering)**.** *Let $\mathcal{G} = \langle V_{\mathcal{G}}, E_{\mathcal{G}} \rangle$ be a directed tree with root ℓ_0 and terminal ℓ_t and let $\ell \in V_{\mathcal{G}} \setminus \{\ell_t\}$. The* access path *of ℓ is the unique sequence of natural numbers $\langle n_1, \dots, n_k \rangle \in \mathbb{N}^*$ for which there exist nodes $\ell_1, \dots, \ell_k$ such that $\ell_k = \ell$ and for all $1 \leq i \leq k$, $(\ell_{i-1}, n_i, \ell_i) \in \mathcal{G}$. The* access path ordering, *$\prec$ on $\mathcal{G}$ is given by*

$\ell'_1 \prec \ell'_2$ iff the access path of ℓ'_1 is lexicographically smaller than the access path of ℓ'_2.

Definition 4.10 (Sink sequence)**.** *Let $\mathcal{G}$ be a directed tree with terminal ℓ. The sequence $\langle \ell_1, \dots, \ell_k \rangle$ is the* sink sequence *of $\mathcal{G}$, written $\mathsf{sinkseq}(\mathcal{G})$, if (1) it contains all and only the sinks of $\mathcal{G}$ except ℓ and (2) $\ell_i \prec \ell_j$ iff $i < j$.*

We further refine the notions of lists and trees.

Definition 4.11.

1. *A singly-linked list $\mathcal{G}$ is a* cycle *if every node of $\mathcal{G}$ has exactly one predecessor, exactly one successor, and $\ell_1 \xrightarrow{*}_{\mathcal{G}} \ell_2$ for all $\ell_1, \ell_2 \in V_{\mathcal{G}}$.*

2. *A singly-linked list $\mathcal{G}$ is a* lasso *if every node of $\mathcal{G}$ has exactly one successor.*

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

3. *Let $\mathcal{G}$ be a directed tree with root ℓ and terminal ℓ'. Then $\mathcal{G}$ is a* directed tree with root ℓ, terminal ℓ' and holes $\langle \ell_1, \ldots, \ell_k \rangle$ *iff* $\mathsf{sinkseq}(\mathcal{G}) = \langle \ell_1, \ldots, \ell_k \rangle$.

Example 4.12. 1. *Let $\mathcal{G} = \langle \{a,b\}, \{\langle a,1,b\rangle, \langle b,1,a\rangle\}\rangle$. $\mathcal{G}$ is a cycle.*

2. *Let $\mathcal{G} = \langle \{a,b,c\}, \{\langle a,1,b\rangle, \langle b,1,c\rangle, \langle c,1,b\rangle\}\rangle$. $\mathcal{G}$ is a lasso.*

3. *Let $\mathcal{G} = \langle \{a,b,c,d,e,n\}, \{\langle a,1,b\rangle, \langle a,2,c\rangle, \langle b,1,n\rangle, \langle b,2,d\rangle, \langle c,1,n\rangle, \langle c,2,e\rangle\}\rangle$. $\mathcal{G}$ is a directed tree with root a, terminal n, and holes $\langle d,e\rangle$.*

FROM GRAPHS TO STACK–HEAP MODELS. The reachability relations are adapted from directed graphs to heaps in the obvious way. For example, if $\mathcal{G} = \mathsf{igraph}(\mathfrak{h})$, then $\ell_1 \mapsto_{\mathfrak{h}} \ell_2$ iff $\ell_1 \to_{\mathcal{G}} \ell_2$ and $\ell_1 \xrightarrow{*}_{\mathfrak{h}} \ell_2$ iff $\ell_2 \xrightarrow{*}_{\mathcal{G}} \ell_2$. Assuming $\mathfrak{s}$ is clear from the context, we extend this notation to variables. For example, we write $x \to_{\mathfrak{h}} y$ if for two variables $x, y \in \mathrm{dom}(\mathfrak{s})$, it holds that $\mathfrak{s}(x) \to_{\mathfrak{h}} \mathfrak{s}(y)$.

We adapt the notion of sources and sinks to variables. First, we define $\mathsf{sinkseq}(\mathfrak{h}) \triangleq \mathsf{sinkseq}(\mathsf{igraph}(\mathfrak{h}))$.

Definition 4.13 (Source and sink variables)**.** *Let $(\mathfrak{s}, \mathfrak{h})$ be a model.*

- *If $(\mathfrak{s}, \mathfrak{h})$ is a directed tree with root ℓ,*

$$\mathsf{sourcevars}_{\mathfrak{s}}(\mathfrak{h}) \triangleq \mathfrak{s}^{-1}(\ell) \text{ and}$$
$$\mathsf{sinkvars}_{\mathfrak{s}}(\mathfrak{h}) \triangleq \mathfrak{s}^{-1}(\mathsf{sinkseq}(\mathfrak{h})).$$

- *If $(\mathfrak{s}, \mathfrak{h})$ is an acyclic singly-linked list with source ℓ and sink ℓ',* $\mathsf{sourcevars}_{\mathfrak{s}}(\mathfrak{h}) \triangleq \mathfrak{s}^{-1}(\ell)$ *and* $\mathsf{sinkvars}_{\mathfrak{s}}(\mathfrak{h}) \triangleq \langle \mathfrak{s}^{-1}(\ell') \rangle$.

- *If $\mathfrak{h}$ is* not *acyclic, but contains a unique location ℓ such that $\ell \xrightarrow{*}_{\mathfrak{h}} \ell'$ for all $\ell' \in \mathsf{locs}(\mathfrak{h})$, then*

$$\mathsf{sourcevars}_{\mathfrak{s}}(\mathfrak{h}) \triangleq \mathfrak{s}^{-1}(\ell) \text{ and}$$
$$\mathsf{sinkvars}_{\mathfrak{s}}(\mathfrak{h}) \triangleq \mathfrak{s}^{-1}(\mathrm{img}(\mathfrak{s}) \setminus \mathfrak{s}(\mathsf{nil})) \setminus \emptyset;$$

  In this case, we impose an arbitrary order on $\mathsf{sinkvars}_{\mathfrak{s}}(\mathfrak{h})$.

  In all other cases, $\mathsf{sourcevars}_{\mathfrak{s}}(\mathfrak{h}) = \bot$ *and* $\mathsf{sinkvars}_{\mathfrak{s}}(\mathfrak{h}) = \bot$.

Example 4.14 (Source and sink variables)**.** 1. *Let*

$$\mathfrak{s} \triangleq \{x \mapsto 1, y_1 \mapsto 6, y_2 \mapsto 7, y_3 \mapsto 5, z \mapsto 5\} \text{ and}$$
$$\mathfrak{h} \triangleq \{1 \mapsto \langle 2,3\rangle, 2 \mapsto \langle 4,5\rangle, 3 \mapsto \langle 0,0\rangle, 4 \mapsto \langle 6,7\rangle\}.$$

Then

$$\mathsf{sourcevars}_{\mathfrak{s}}(\mathfrak{h}) = \{\{x\}\} \text{ and}$$
$$\mathsf{sinkvars}_{\mathfrak{s}}(\mathfrak{h}) = \langle \{y_1\}, \{y_2\}, \{y_3, z\}\rangle.$$

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

2. *Let* $\mathfrak{s} \triangleq \{x \mapsto 1, y \mapsto 2\}$ *and* $\mathfrak{h} \triangleq \{1 \mapsto \langle 2,1\rangle\}$*. Then*

$$\mathsf{sourcevars}_{\mathfrak{s}}(\mathfrak{h}) = \{\{x\}\} \quad \textit{and} \quad \mathsf{sinkvars}_{\mathfrak{s}}(\mathfrak{h}) = \langle \{x\}, \{y\} \rangle .$$

3. *Let* $\mathfrak{s} \triangleq \{x \mapsto 1, y \mapsto 2\}$ *and* $\mathfrak{h} \triangleq \{1 \mapsto \langle 2,1\rangle, 3 \mapsto \langle 1,2\rangle\}$*. Then* $\mathsf{sourcevars}_{\mathfrak{s}}(\mathfrak{h}) = \bot$ *and* $\mathsf{sinkvars}_{\mathfrak{s}}(\mathfrak{h}) = \bot$*, because* $(\mathfrak{s},\mathfrak{h})$ *is neither acyclic, nor does* $\mathfrak{h}$ *contain a location from which all heap locations are reachable.*

Sources vs. roots

A *source* is any node (of directed graph $\mathcal{G}$) or location (of model $(\mathfrak{s},\mathfrak{h})$) that does not have predecessors, whereas I use the term *root* only for the roots of (sets of) trees.

Sinks vs. holes

A *sink* is any node (of directed graph $\mathcal{G}$) or location (of model $(\mathfrak{s},\mathfrak{h})$) that does not have successors. A *hole* is an explicitly specified sink. Specifically, in the case of a tree with holes, a hole is a sink that is different from the dedicated terminal node. The terminal node represents the null pointer and is the only sink that may occur multiple times in a tree.

Further, we also define several classes of stack–heap models related to the graph classes defined in Definition 4.11.

Definition 4.15. *Let* $(\mathfrak{s},\mathfrak{h})$ *be a model and* $\mathcal{G} \triangleq \mathsf{igraph}(\mathfrak{h})$*.*

1. $(\mathfrak{s},\mathfrak{h})$ *is a* singly-linked list *iff* $\mathcal{G}$ *is a singly-linked list. It is a* cycle *or* lasso *iff* $\mathcal{G}$ *is a cycle or lasso.*

2. $(\mathfrak{s},\mathfrak{h})$ *is a* connected list segment from x to y via $\langle x_1,\ldots,x_k\rangle$, $k \geq 0$*, if either of the following holds.*
   a) $\mathfrak{h} = \emptyset$*,* $k = 0$*, and* $\mathfrak{s}(x) = \mathfrak{s}(y)$*; or*
   b) $(\mathfrak{s},\mathfrak{h})$ *is a singly-linked list,* x *is a source variable of* $(\mathfrak{s},\mathfrak{h})$*,* y *is a sink variable of* $(\mathfrak{s},\mathfrak{h})$*, and if* $k > 0$ *then* $x \xrightarrow{+}_{\mathfrak{h}} x_1$*,* $x_i \xrightarrow{+} x_{i+1}$ *for all* i*, and* $x_k \xrightarrow{+}_{\mathfrak{h}} y$*.*

3. $(\mathfrak{s},\mathfrak{h})$ *is a* directed tree with root x and holes $\langle x_1,\ldots,x_k\rangle$ *if either of the following holds.*
   a) $\mathfrak{h} = \emptyset$*,* $k = 0$*, and* $\mathfrak{s}(x) = \mathfrak{s}(\mathsf{nil})$*; or*
   b) $\mathfrak{h} = \emptyset$*,* $k = 1$*, and* $\mathfrak{s}(x) = \mathfrak{s}(x_1)$*; or*
   c) $\mathsf{igraph}(\mathfrak{h})$ *is a directed tree with root* $\mathfrak{s}(x)$*, terminal* $\mathfrak{s}(\mathsf{nil})$*, and holes* $\langle \mathfrak{s}(x_1),\ldots,\mathfrak{s}(x_k)\rangle$*.*

Finally, I will need the *connected components* of directed graphs, which I will define in terms of the following auxiliary definition.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Definition 4.16 (Symmetry closure)**.** *The* symmetry closure *of a directed graph* $\mathcal{G} = \langle V_\mathcal{G}, E_\mathcal{G} \rangle$ *is the directed graph* $\langle V_\mathcal{G}, E_\mathcal{G} \cup \{\langle \ell', \ell \rangle \mid \langle \ell, \ell' \rangle \in E_\mathcal{G}\} \rangle$.

Definition 4.17 (Connected component)**.** *Let* $\mathcal{G} = \langle V_\mathcal{G}, E_\mathcal{G} \rangle, \mathcal{G}' = \langle V'_\mathcal{G}, E'_\mathcal{G} \rangle$ *be directed graphs and let* $\mathcal{S}$ *be the symmetry closure of* $\mathcal{G}$. $\mathcal{G}'$ *is a* connected component *of* $\mathcal{G}$ *if (1)* $V'_\mathcal{G}$ *is a maximal, nonempty subset of* $V_\mathcal{G}$ *such that for all* $\ell, \ell' \in V'_\mathcal{G}$, $\ell \xrightarrow{*}_\mathcal{S} \ell'$ *and (2)* $E'_\mathcal{G} = E_\mathcal{G} \cap V'_\mathcal{G} \times V'_\mathcal{G}$.

I deliberately allow that $E'_\mathcal{G}$ is empty in the above definition, i.e., that a connected component does not contain any edges.

Example 4.18 (Connected components)**.** *Let* $\mathcal{G} = \langle \{a, b, c, d, e, f, g\}, \{(a, b), (b, c), (d, e), (e, d), (f, d)\} \rangle$. *The connected components of* $\mathcal{G}$ *are*

1. $\langle \{a, b, c\}, \{(a, b), (b, c)\} \rangle$,
2. $\langle \{d, e, f\}, \{(d, e), (e, d), (f, d)\} \rangle$,
3. $\langle \{g\}, \emptyset \rangle$.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Part II

DECIDING STRONG-SEPARATION LOGIC WITH TREES AND DATA

I study separation logic with built-in list and tree predicates, but with two twists: First, I propose a more restrictive semantics of the separating conjunction. This semantics does not change the meaning of symbolic heaps, but makes the (usually undecidable [DLM18]) full propositional logic, including negation and magic wands, decidable. Second, I consider universal and existential relational constraints on the data stored inside the data structures to express functional-correctness properties such as "h is the head of a sorted list" or "t is the root of a binary search tree." I present a PSpace decision procedure for the full propositional logic *without* data predicates and SMT-based NP and coNP decision procedures for the satisfiability and entailment problem of the *positive* fragment of the logic (with data predicates, guarded negation and without magic wands). Some of the work presented in this part was previously published in [KJW18a; PZ20b].

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.

5 STRONG-SEPARATION LOGIC WITH LISTS, TREES, AND DATA PREDICATES

In Chapter 3, I motivated the *strong-separation* semantics of the separating connectives and proposed *data predicates* that enable a combined shape–value analysis with strong-separation logic.

In this chapter, I formally introduce a separation logic with these two features, *strong-separation logic with lists, trees, and data predicates*, $\mathbf{SSL}_{\mathsf{data}}$. I begin with the logic **SSL** without data predicates, introducing its syntax and semantics and contrasting it to the semantics of standard, "weak-separation" logic in Section 5.1. In Section 5.2, I define $\mathbf{SSL}_{\mathsf{data}}$ as the extension of SSL with data predicates.

Over the course of the following two chapters, I develop decision procedures for **SSL** (Chapter 6) and $\mathbf{SSL}^{+}_{\mathsf{data}}$, the positive fragment of $\mathbf{SSL}_{\mathsf{data}}$ (Chapter 7).

HOW PART II RELATES TO [KJW18A; PZ20B]. Strong-separation logic with lists, trees and data predicates is an amalgamation of the logic $\mathbf{SL}^{*}_{\mathsf{data}}$ that Dejan Jovanović, Georg Weissenbacher and I introduced in [KJW18a] and of the logic SSL that Florian Zuleger and I introduced in [PZ20b]: $\mathbf{SL}^{*}_{\mathsf{data}}$ features data predicates, but assumes standard, "weak" separation, allows no Boolean connectives below $\star$ and no magic wands. SSL as defined in [PZ20b] is a full propositional separation logic built on top of strong separation, but only supports lists, not trees, and is a "shape-only" logic without support for data predicates.

As such, this part of the thesis has grown out of joint work with Dejan Jovanović, Georg Weissenbacher and Florian Zuleger [KJW18a; PZ20b]. Sections that are due in large part to one of my contributors contain an explicit attribution.

ADDING A DATA THEORY TO SEPARATION LOGIC. In Chapter 2, we made the distinction between memory locations, **Loc**, and values in the heap, **Val**, but did not specify the set $\mathbf{Val} \setminus \mathbf{Loc}$ of "non-location" values.

In this part of the thesis, I assume a set **Data** of data values, with $\mathbf{Data} \cap \mathbf{Loc} = \emptyset$, and define the set of values as $\mathbf{Val} \triangleq \mathbf{Loc} \cup \mathbf{Data}$.

SSL is parametric with respect to a background theory $\mathcal{T}_{\mathbf{Data}}$ of the data domain **Data**. I denote by $\mathcal{F}_{\mathbf{Data}}$ the set of all quantifier-free $\mathcal{T}_{\mathbf{Data}}$-formulas. The background theory can be instantiated with any first-order theory with equality, as usual in satisfiability modulo theories (see, e.g., [Bar+09]). We assume a model relation $\models_{\mathbf{Data}}$ that

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

captures the semantics of the background theory, i.e., $\mathfrak{s} \models_{\mathbf{Data}} F$ holds if and only if the formula $F \in \mathcal{F}_{\mathbf{Data}}$ is true in stack $\mathfrak{s}$.

Disjointness of Loc and Data

I make the disjointness assumption $\mathbf{Data} \cap \mathbf{Loc} = \emptyset$ purely for technical convenience. In practice, this assumption is unrealistic. For example, both the memory locations and the data stored in the heap might be fixed-width integers. We can still guarantee disjointness in our mathematical representation of the heap, for example by adding a "tag" to each value that identifies it as a location or data value.

5.1 STRONG-SEPARATION LOGIC WITHOUT DATA PREDICATES

In this section, I introduce the core fragment of strong-separation logic (SSL) with lists, trees, and data, **SSL**. In this core fragment, it is only possible to reason about the shape of the heap, not about the data stored inside the heap. I will add support for such reasoning in Section 5.2.

5.1.1 *Data Structures*

We will consider separation-logic over data structures $\mathcal{D} = \{\mathsf{ls}, \mathsf{tree}\}$ representing *singly-linked list segments* and *binary-tree segments*. Before I formalize SSL, I explain how these data structures are modeled in the stack–heap model as introduced in Section 2.3.2.

We associate with each data structure $\mathsf{ds} \in \mathcal{D}$ a *node signature*, $\mathsf{sig}(\mathsf{ds})$, which captures the shape of each element (node) of the data structure.

$$\mathsf{sig}(\mathsf{ls}) \triangleq \mathbf{Loc} \times \mathbf{Data}$$
$$\mathsf{sig}(\mathsf{tree}) \triangleq \mathbf{Loc} \times \mathbf{Loc} \times \mathbf{Data}$$

A node in a list consists of a pointer to its successor (which is an element from **Loc**) and the data stored in the list node (an element from **Data**). A node in a tree consists of pointers to its left and right successors (from **Loc**) and the data stored in the node (from **Data**).

Since heap locations are not typed in our variant of the stack–heap model, I assume that every heap location that corresponds to the signature of a data structure ds is in fact of the type defining this data structure. For example, if $\mathfrak{h}(\ell) \in \mathbf{Loc} \times \mathbf{Data}$, ℓ is a list location. The following definitions encapsulate this assumption.

$$\mathsf{loc}_{\mathsf{ls}}(\mathfrak{h}) \triangleq \{\ell \in \mathbf{Loc} \mid \ell \in \mathrm{dom}(\mathfrak{h}) \text{ and } \mathfrak{h}(\ell) \in \mathsf{sig}(\mathsf{ls})\}$$
$$\mathsf{loc}_{\mathsf{tree}}(\mathfrak{h}) \triangleq \{\ell \in \mathbf{Loc} \mid \ell \in \mathrm{dom}(\mathfrak{h}) \text{ and } \mathfrak{h}(\ell) \in \mathsf{sig}(\mathsf{tree})\}$$

For convenience, we additionally define partial functions $\mathsf{n}, \mathsf{l}, \mathsf{r}, \mathsf{d}$ for accessing the (next, left, right, data) *fields* of the data structures:

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

- If $\ell \in \mathsf{loc}_{\mathsf{ls}}(\mathfrak{h})$ and $\mathfrak{h}(\ell) = \langle \ell', v \rangle$, then $\mathsf{n}(\ell) = \ell'$, $\mathsf{l}(\ell) = \bot$, $\mathsf{r}(\ell) = \bot$, $\mathsf{d}(\ell) = v$.
- If $\ell \in \mathsf{loc}_{\mathsf{tree}}(\mathfrak{h})$ and $\mathfrak{h}(\ell) = \langle \ell_1, \ell_2, v \rangle$, then $\mathsf{n}(\ell) = \bot$, $\mathsf{l}(\ell) = \ell_1$, $\mathsf{r}(\ell) = \ell_2$, $\mathsf{d}(\ell) = v$.

Further, we define $\mathsf{sort}(\mathsf{f})$ as the associated sort of field f,

$$\mathsf{sort}(\mathsf{n}) \triangleq \mathbf{Loc},\ \mathsf{sort}(\mathsf{l}) \triangleq \mathbf{Loc},\ \mathsf{sort}(\mathsf{r}) \triangleq \mathbf{Loc},\ \mathsf{sort}(\mathsf{d}) \triangleq \mathbf{Data}.$$

Extending SSL with additional data structures

The results in this part of the thesis can easily be extended to other variants of lists and trees by extending $\mathcal{D}$ and defining an appropriate node signature. For example, it would be possible to support doubly-linked structures or structures that store multiple values, possibly from multiple types of data. This would, however, significantly complicate the formalization without providing any additional insights, so I stick to ls and tree in this thesis.

Unfortunately, I do not have a formal characterization of the data structures that my approach can handle. In particular, I do not know if there is a class of data structures that allow limited sharing and thus go "beyond trees" (cf. [MS01; IRS13]), while still allowing abstractions of polynomial size (cf. Chapter 6).

5.1.2 *Syntax of Strong-Separation Logic*

In Fig. 5.1, I've defined the syntax of "shape-only" *strong-separation logic*, **SSL** and its *positive fragment*, $\mathbf{SSL}^{+}$. Both SL variants are constructed from the same atomic formulas, ϕ_{atom}:

- A *points-to predicate* $x \mapsto_{\mathsf{n}} y$, expressing that x is an allocated list node that contains a pointer to y.
- A points-to predicate $x \mapsto_{\mathsf{l,r}} \langle y_1, y_2 \rangle$, expressing that x is an allocated tree node that contains pointers to left child y_1 and right child y_2.
- The logic includes the four inductive predicates ls, $\mathsf{ls}_{\geq 2}$, tree, and $\mathsf{tree}_{\geq 2}$. These predicates correspond to (possibly empty) list segments, list segments of length at least two, (possibly empty) tree segments, and tree segments of depth at least two. The inclusion of $\mathsf{ls}_{\geq 2}$ and $\mathsf{tree}_{\geq 2}$ in the logic is mostly for technical convenience, as we will see later. The second parameter of each inductive predicate, $\mathbf{x}$, is a sequence of *holes* delineating the data-structure segment. To reduce clutter, I write $\mathsf{ds}(x)$ instead of $\mathsf{ds}(x, \varepsilon)$ and $\mathsf{ds}(x, y)$ instead of $\mathsf{ds}(x, \langle y \rangle)$.
- (Dis)equalities between variables, $x \approx y$ and $x \not\approx y$, are atomic separation-logic formulas as well.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

$$\begin{array}{rcl}
\phi_{\mathsf{atom}} & ::= & x \mapsto_{\mathsf{n}} y \mid x \mapsto_{\mathsf{l,r}} \langle y_1, y_2 \rangle \\
& & \mid \mathsf{ls}(x, \mathbf{y}) \mid \mathsf{tree}(x, \mathbf{y}) \mid \mathsf{ls}_{\geq 2}(x, \mathbf{y}) \mid \mathsf{tree}_{\geq 2}(x, \mathbf{y}) \\
& & \mid x \approx y \mid x \not\approx y \\
\phi & ::= & \phi_{\mathsf{atom}} \mid \phi \star \phi \mid \phi \mathrel{-\!\!\star} \phi \mid \phi \wedge \phi \mid \phi \vee \phi \mid \neg \phi \\
\phi_{\mathsf{spatial}} & ::= & \phi_{\mathsf{atom}} \mid \phi_{\mathsf{spatial}} \star \phi_{\mathsf{spatial}} \\
\phi_{\mathsf{pos}} & ::= & \phi_{\mathsf{spatial}} \mid \phi_{\mathsf{pos}} \vee \phi_{\mathsf{pos}} \mid \phi_{\mathsf{pos}} \wedge \phi_{\mathsf{pos}} \mid \phi_{\mathsf{pos}} \wedge \neg \phi_{\mathsf{pos}}
\end{array}$$

Figure 5.1: Syntax: ϕ defines formulas of strong-separation logic with lists, trees, and data, **SSL**; and ϕ_{pos} defines the formulas of its positive fragment, $\mathbf{SSL}^{+}$. We assume $x, y, y_1, y_2 \in \mathbf{Var}$, $\mathbf{y} \in \mathbf{Var}^{*}$.

emp	$\triangleq$	$\mathsf{nil} \approx \mathsf{nil}$	empty-heap predicate
$\phi \circledast \psi$	$\triangleq$	$\neg(\phi \mathrel{-\!\!\star} \neg\psi)$	septraction
t	$\triangleq$	$\mathbf{emp} \vee \neg\mathbf{emp}$	true
f	$\triangleq$	$\neg\mathsf{t}$	false
$\mathsf{alloc}(x)$	$\triangleq$	$(x \mapsto_{\mathsf{n}} \mathsf{nil}) \mathrel{-\!\!\star} \mathsf{f}$	x is allocated

Figure 5.2: Derived **SSL** formulas.

In "full" propositional SSL, we can freely combine atoms with all the operators introduced in Part i, i.e., the separating conjunction $\star$, the magic wand $\mathrel{-\!\!\star}$, and Boolean connectives $\wedge$, $\vee$, and $\neg$. In positive SSL, we build *spatial formulas*, ϕ_{spatial}, using the separating conjunction $\star$ and then combine spatial formulas using conjunction, disjunction, and *guarded negation*.

Before we turn to the semantics, allow me to provide some context.

1. I have not included **emp**, representing the empty heap, in the syntax. This is because in our semantics, it is possible to introduce **emp** as syntactic sugar, see Fig. 5.2.

2. The spatial formulas ϕ_{spatial} are quantifier-free *symbolic heaps*.

3. Guarded negation has, as far as I know, not been studied previously in the context of separation logic, but for example for first-order logic [BCS15].

4. Guarded negation is sufficient for reducing entailment checking to (un-)satisfiability checking: $\phi \models \psi$ iff the guarded negation $\phi \wedge \neg\psi$ is unsatisfiable.

In Fig. 5.2, I introduce notation for several derived formulas.

Example 5.1 (Syntax). *Let w, x, y, z be variables.*

- $\mathsf{ls}(x, y) \star \mathsf{ls}(y)$ *are disjoint list segments from x to y and from y to* nil.
- $\mathsf{ls}(x, y) \wedge (\neg x \mapsto_{\mathsf{n}} y) \wedge \neg\mathbf{emp}$ *is equivalent to* $\mathsf{ls}_{\geq 2}(x, y)$.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

- $\mathsf{tree}(x, \langle y, z \rangle) \star \mathsf{tree}(y) \star \mathsf{tree}(z)$ *represents a binary tree rooted in* x *that contains two subtrees* y *and* z *ordered from left to right, as specified by the sequence of holes,* $\langle y, z \rangle$.
- $\mathsf{tree}(y) \mathrel{-\!\!\star} \mathsf{tree}(x)$ *is equivalent to the formula* $\mathsf{tree}(x, y)$, *i.e., a tree segment with root* x *and a single hole* y.

The first three formulas are $\mathbf{SSL}^{+}$ *formulas, the last formula is in* $\mathbf{SSL} \setminus \mathbf{SSL}^{+}$.

ADDITIONAL NOTATION. For $\Psi = \{\psi_1, \ldots, \psi_n\}$, we define $\bigstar\Psi \triangleq \psi_1 \star \psi_2 \star \cdots \star \psi_n$ if $n \geq 1$ and $\bigstar\Psi \triangleq \mathbf{emp}$ if $n = 0$.

By $\mathsf{fvars}(\phi)$ we denote the set of free variables of ϕ. As we are only dealing with quantifier-free logics in this part of the thesis, these are in fact *all* variables that occur in ϕ.

We define the *size* of the formula ϕ as $|\phi| = 1$ for atomic formulas ϕ, $|\phi_1 \times \phi_2| \triangleq |\phi_1| + |\phi_2| + 1$ for $\times \in \{\wedge, \vee, \star, \mathrel{-\!\!\star}\}$ and $|\neg\phi_1| \triangleq |\phi_1| + 1$.

5.1.3 *Semantics of Strong-Separation Logic*

We use the stack–heap model introduced in Section 2.3.2 to give a semantics to SSL. Recall that in SSL, $\mathbf{Val}$ is $\mathbf{Loc} \cup \mathbf{Data}$. Further, I will make the simplifying assumption that all allocated heap locations are a node of one of the data structures $\mathcal{D} = \{\mathsf{ls}, \mathsf{tree}\}$. This implies that a stack is a finite partial function $\mathfrak{s} \colon \mathbf{Var} \rightharpoonup \mathbf{Loc} \cup \mathbf{Data}$ and that a *heap* is a partial function $\mathfrak{h} \colon \mathbf{Loc} \rightharpoonup \bigcup_{\mathsf{ds} \in \mathcal{D}} \mathsf{sig}(\mathsf{ds})$. I call x a *location variable* if $\mathfrak{s}(x) \in \mathbf{Loc}$ and a *data variable* if $\mathfrak{s}(x) \in \mathbf{Data}$.

TWO NOTIONS OF DISJOINT UNION OF HEAPS. We write $\mathfrak{h}_1 + \mathfrak{h}_2$ for the union of disjoint heaps, i.e.,

$$\mathfrak{h}_1 + \mathfrak{h}_2 \triangleq \begin{cases} \mathfrak{h}_1 \cup \mathfrak{h}_2, & \text{if } \mathrm{dom}(\mathfrak{h}_1) \cap \mathrm{dom}(\mathfrak{h}_2) = \emptyset \\ \bot, & \text{otherwise.} \end{cases}$$

This standard notion of disjoint union is commonly used to assign a semantics to the separating conjunction and magic wand. It requires that $\mathfrak{h}_1$ and $\mathfrak{h}_2$ are domain-disjoint, but does not impose any restrictions on the *images* of the heaps. In particular, the dangling pointers of $\mathfrak{h}_1$ may alias arbitrarily with the domain and image of $\mathfrak{h}_2$ and vice-versa.

Let $\mathfrak{s}$ be a stack. We write $\mathfrak{h}_1 \uplus^{\mathfrak{s}} \mathfrak{h}_2$ for the disjoint union of $\mathfrak{h}_1$ and $\mathfrak{h}_2$ that restricts aliasing of dangling pointers to the locations in the image of stack $\mathfrak{s}$. This yields an infinite family of union operators: one for each stack. Formally,

$$\mathfrak{h}_1 \uplus^{\mathfrak{s}} \mathfrak{h}_2 \triangleq \begin{cases} \mathfrak{h}_1 + \mathfrak{h}_2, & \text{if } \mathsf{locs}(\mathfrak{h}_1) \cap \mathsf{locs}(\mathfrak{h}_2) \subseteq \mathrm{img}(\mathfrak{s}) \\ \bot, & \text{otherwise.} \end{cases}$$

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Wien Bibliothek Your knowledge hub

Intuitively, $\mathfrak{h}_1 \uplus^{\mathfrak{s}} \mathfrak{h}_2$ is the (disjoint) union of heaps that share only locations that are in the image of the stack $\mathfrak{s}$. Note that if $\mathfrak{h}_1 \uplus^{\mathfrak{s}} \mathfrak{h}_2$ is defined then $\mathfrak{h}_1 + \mathfrak{h}_2$ is defined, but not vice-versa.

Just like the standard disjoint union $+$, the operator $\uplus^{\mathfrak{s}}$ gives rise to a *separation algebra*, i.e., a cancellative, commutative partial monoid [COY07].

Lemma 5.2. *Let $\mathfrak{s}$ be a stack and write $u \triangleq \lambda x. \bot$. The triple $(\mathbf{Heaps}, \uplus^{\mathfrak{s}}, u)$ is a separation algebra.*

Proof. Trivially, the operation $\uplus^{\mathfrak{s}}$ is commutative and associative with unit u. Let $\mathfrak{h} \in \mathbf{Heaps}$. Let $\mathfrak{h}_1 \neq \mathfrak{h}_2$ such that $\mathfrak{h} \uplus^{\mathfrak{s}} \mathfrak{h}_1 = \mathfrak{h} \uplus^{\mathfrak{s}} \mathfrak{h}_2 \neq \bot$. Since the domain $\mathfrak{h}$ is disjoint from both the domain of $\mathfrak{h}_1$ and the domain of $\mathfrak{h}_2$, it follows that for all ℓ, $\mathfrak{h}_1(\ell) = \mathfrak{h}_2(\ell)$ and thus $\mathfrak{h}_1 = \mathfrak{h}_2$. As $\mathfrak{h}_1$ and $\mathfrak{h}_2$ were chosen arbitrarily, we obtain that the function $\mathfrak{h} \uplus^{\mathfrak{s}} (\cdot)$ is injective. Consequently, the monoid is cancellative. □

Whether or not the models form a separation algebra can be viewed as the litmus test for separation logics: it is a prerequisite for the logic to satisfy the frame rule (cf. Chapter 2).[1] The fact that $\uplus^{\mathfrak{s}}$ gives rise to a separation algebra is thus a strong indicator that $\uplus^{\mathfrak{s}}$ is suitable for defining the semantics of $\star$.[2]

SEMANTICS OF ATOMIC FORMULAS. Figure 5.3 defines the semantics of SSL. Just like in Chapter 2, I implicitly assume that all variables that occur in a formula ϕ are defined in the stack $\mathfrak{s}$, i.e., $(\mathfrak{s}, \mathfrak{h}) \models \phi$ is undefined if $\mathsf{fvars}(\phi) \not\subseteq \mathsf{dom}(\mathfrak{s})$. (Dis-)equalities only hold if the heap is empty. Points-to assertions hold in single-element heaps, i.e., we use a *precise* semantics (see e.g. [BCO04]). Since we cannot reason about data in **SSL** (as opposed to $\mathbf{SSL}_{\mathsf{data}}$), we only state that there must exist *some* data value stored in the heap location, without constraining *which* data value is stored in the location.

The holes of lists and trees are required to be pairwise different and different from nil, expressed using the auxiliary formula $\mathsf{distinct}(\mathbf{z})$. The semantics of lists and trees are defined recursively. You can read the auxiliary predicate $\mathsf{ds}^i_{\mathbf{z}}(x, \mathbf{y})$, $\mathsf{ds} \in \mathcal{D}$, as "the data structure consists of i allocated elements, its root is x, its holes are $\mathbf{y}$, and it is part of a larger data structure with holes $\mathbf{z}$." We use the reference to the original holes $\mathbf{z}$ to enforce that none of the holes is allocated inside the data structure, thus enforcing acyclicity.

1 Whether the frame rule actually holds then depends on the operational semantics of the programming language underlying the Hoare logic: the statements of the programming language need to have local effects for the frame rule to hold; see [COY07].
2 This is, perhaps, an overly simplistic view. If you are interested in a more nuanced discussion of the semantics of separation logic, I suggest you read [CCA17].

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

$$
\begin{array}{lll}
(\mathfrak{s},\mathfrak{h}) \models x \approx y & \text{iff} & \mathfrak{s}(x) = \mathfrak{s}(y) \text{ and } \mathrm{dom}(\mathfrak{h}) = \emptyset \\
(\mathfrak{s},\mathfrak{h}) \models x \not\approx y & \text{iff} & \mathfrak{s}(x) \neq \mathfrak{s}(y) \text{ and } \mathrm{dom}(\mathfrak{h}) = \emptyset \\
(\mathfrak{s},\mathfrak{h}) \models x \mapsto_{\mathsf{n}} y & \text{iff} & \exists d \in \mathbf{Data}.\, \mathfrak{h} = \{\mathfrak{s}(x) \mapsto \langle \mathfrak{s}(y), d\rangle\} \\
(\mathfrak{s},\mathfrak{h}) \models x \mapsto_{\mathsf{l,r}} \langle y_1, y_2\rangle & \text{iff} & \exists d \in \mathbf{Data}.\, \mathfrak{h} = \{\mathfrak{s}(x) \mapsto \langle \mathfrak{s}(y_1), \mathfrak{s}(y_2), d\rangle\} \\
(\mathfrak{s},\mathfrak{h}) \models \mathsf{distinct}(\mathbf{z}) & \text{iff} & (\mathfrak{s},\mathfrak{h}) \models \bigstar_{y_1,y_2 \in \mathbf{z}, y_1 \neq y_2} y_1 \not\approx y_2 \\
(\mathfrak{s},\mathfrak{h}) \models \mathsf{ls}(x,\mathbf{y}) & \text{iff} & \exists i \geq 0.\, (\mathfrak{s},\mathfrak{h}) \models \mathsf{ls}^i_{\mathbf{y}}(x,\mathbf{y}) \star \mathsf{distinct}(\mathbf{y} \cup \{\mathsf{nil}\}) \\
(\mathfrak{s},\mathfrak{h}) \models \mathsf{ls}_{\geq 2}(x,\mathbf{y}) & \text{iff} & \exists i \geq 2.\, (\mathfrak{s},\mathfrak{h}) \models \mathsf{ls}^i_{\mathbf{y}}(x,\mathbf{y}) \star \mathsf{distinct}(\mathbf{y} \cup \{\mathsf{nil}\}) \\
(\mathfrak{s},\mathfrak{h}) \models \mathsf{tree}(x,\mathbf{y}) & \text{iff} & \exists i \geq 0.\, (\mathfrak{s},\mathfrak{h}) \models \mathsf{tree}^i_{\mathbf{y}}(x,\mathbf{y}) \star \mathsf{distinct}(\mathbf{y} \cup \{\mathsf{nil}\}) \\
(\mathfrak{s},\mathfrak{h}) \models \mathsf{tree}_{\geq 2}(x,\mathbf{y}) & \text{iff} & \exists i \geq 2.\, (\mathfrak{s},\mathfrak{h}) \models \mathsf{tree}^i_{\mathbf{y}}(x,\mathbf{y}) \star \mathsf{distinct}(\mathbf{y} \cup \{\mathsf{nil}\}) \\
(\mathfrak{s},\mathfrak{h}) \models \mathsf{ls}^i_{\mathbf{z}}(x,\mathbf{y}) & \text{iff} & \exists \ell \in \mathbf{Loc}, \mathfrak{s}' \in \mathbf{Stacks}. \\
 & & \mathfrak{s}' = \mathfrak{s} \cup \{v \mapsto \ell\}, \mathfrak{s}'(x) \notin \mathfrak{s}'(\mathbf{z}) \text{ and} \\
 & & (\mathfrak{s}',\mathfrak{h}) \models (x \mapsto_{\mathsf{n}} v) \star \mathsf{ls}^{i-1}_{\mathbf{z}}(v,\mathbf{y}) \\
(\mathfrak{s},\mathfrak{h}) \models \mathsf{tree}^i_{\mathbf{z}}(x,\mathbf{y}) & \text{iff} & \exists i_1, i_2 \in \mathbb{N}, \mathbf{y}_1, \mathbf{y}_2 \in \mathbf{Var}^*, \ell_1, \ell_2 \in \mathbf{Loc}, \mathfrak{s}' \in \mathbf{Stacks}. \\
 & & \mathbf{y} = \mathbf{y}_1 \cdot \mathbf{y}_2, i = i_1 + i_2 + 1, \\
 & & \mathfrak{s}' = \mathfrak{s} \cup \{v_1, v_2 \mapsto \ell_1, \ell_2\}, \mathfrak{s}'(x) \notin \mathfrak{s}'(\mathbf{z}), \text{ and} \\
 & & (\mathfrak{s}',\mathfrak{h}) \models (x \mapsto_{\mathsf{l,r}} \langle v_1, v_2\rangle) \star \mathsf{tree}^{i_1}_{\mathbf{z}}(v_1,\mathbf{y}_1) \star \mathsf{tree}^{i_2}_{\mathbf{z}}(v_2,\mathbf{y}_2) \\
(\mathfrak{s},\mathfrak{h}) \models \mathsf{ds}^0_{\mathbf{z}}(x,\epsilon) & \text{iff} & (\mathfrak{s},\mathfrak{h}) \models x \approx \mathsf{nil} \\
(\mathfrak{s},\mathfrak{h}) \models \mathsf{ds}^0_{\mathbf{z}}(x,\langle y\rangle) & \text{iff} & (\mathfrak{s},\mathfrak{h}) \models x \approx y \\
(\mathfrak{s},\mathfrak{h}) \models \phi_1 \star \phi_2 & \text{iff} & \exists \mathfrak{h}_1, \mathfrak{h}_2 \in \mathbf{Heaps}.\, \mathfrak{h} = \mathfrak{h}_1 \uplus^{\mathfrak{s}} \mathfrak{h}_2, \\
 & & (\mathfrak{s},\mathfrak{h}_1) \models \phi_1, \text{ and } (\mathfrak{s},\mathfrak{h}_2) \models \phi_2 \\
(\mathfrak{s},\mathfrak{h}) \models \phi_1 \mathrel{-\!\!\star} \phi_2 & \text{iff} & \forall \mathfrak{h}_1 \in \mathbf{Heaps}. \text{ if } (\mathfrak{s},\mathfrak{h}_1) \models \phi_1 \text{ and } \mathfrak{h} \uplus^{\mathfrak{s}} \mathfrak{h}_1 \neq \bot \\
 & & \text{then } (\mathfrak{s}, \mathfrak{h} \uplus^{\mathfrak{s}} \mathfrak{h}_1) \models \phi_2 \\
(\mathfrak{s},\mathfrak{h}) \models \phi_1 \wedge \phi_2 & \text{iff} & (\mathfrak{s},\mathfrak{h}) \models \phi_1 \text{ and } (\mathfrak{s},\mathfrak{h}) \models \phi_2 \\
(\mathfrak{s},\mathfrak{h}) \models \phi_1 \vee \phi_2 & \text{iff} & (\mathfrak{s},\mathfrak{h}) \models \phi_1 \text{ or } (\mathfrak{s},\mathfrak{h}) \models \phi_2 \\
(\mathfrak{s},\mathfrak{h}) \models \neg \phi & \text{iff} & (\mathfrak{s},\mathfrak{h}) \not\models \phi
\end{array}
$$

Figure 5.3: The semantics of strong-separation logic **SSL**. Variables v, v_1, v_2 are fresh. For brevity we denote with ds either ls or tree.

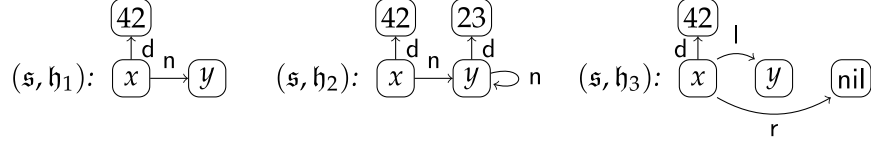
Cyclic data structures

Note that our semantics implies that $\mathsf{ls}(x,x)$ only holds if the heap is empty, following e.g. [DLM18]; similar for $\mathsf{tree}(x,x)$. If you need to specify that your data structure contains a cycle, you have to model this fact explicitly, for example $\mathsf{ls}(x,z) \star (z \mapsto_{\mathsf{n}} x)$, where z is a fresh variable. All results in this part of the thesis can be easily adapted to a semantics that allows cyclic list segments $\mathsf{ls}(x,x)$ or cyclic tree segments $\mathsf{tree}(x,x)$, where one of the paths loops back to the root.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

SEMANTICS OF NON-ATOMIC FORMULAS. We use $\uplus^{\mathfrak{s}}$ to give a semantics to $\star$. The semantics of the magic wand $\mathrel{-\!\!\star}$ is also based on $\uplus^{\mathfrak{s}}$ as opposed to $+$. This guarantees that $\mathrel{-\!\!\star}$ is the right adjoint of $\star$, as usual. The semantics of the Boolean connectives is standard.

Example 5.3 (Semantics). *Consider the following graphical representations of three stack–heap models.*



I've labeled the edges with n, l, r, *and* d *if they correspond to the successor of a list node, left or right child of a tree node, or data field of a node; and I've included the labels of heap locations as node labels. In these models we have that*

$$(\mathfrak{s},\mathfrak{h}_1) \models \mathsf{ls}(x,y), \quad (\mathfrak{s},\mathfrak{h}_2) \not\models \mathsf{ls}(x,y), \quad (\mathfrak{s},\mathfrak{h}_2) \not\models \mathsf{ls}(x,y) \star \mathsf{ls}(y,y),$$
$$(\mathfrak{s},\mathfrak{h}_2) \models \mathsf{ls}(x,y) \star (y \mapsto_{\mathsf{n}} y), \qquad (\mathfrak{s},\mathfrak{h}_3) \models \mathsf{tree}(x,y) \star (x \not\approx y).$$

CHARACTERIZING LIST AND TREE MODELS. We relate the models of list and tree predicates to the heap classes of Definition 4.15.

In the list characterization, I explicitly include the intermediate variables of *connected list segments* (cf. Definition 4.15), as this information will be useful for abstracting lists in Chapter 6.

Lemma 5.4 (Characterization of list models). *Let $(\mathfrak{s},\mathfrak{h})$ be a model, $x \in$ **Var** and $\mathbf{y} \in$ **Var*** with $|\mathbf{y}| < 1$. Moreover, let y be* nil *if* $\mathbf{y}$ *is empty and the unique element in* $\mathbf{y}$ *otherwise. Then $(\mathfrak{s},\mathfrak{h}) \models \mathsf{ls}(x,\mathbf{y})$ iff there exist $x_1,\ldots,x_k \in$ **Var**, $k \geq 0$, such that $(\mathfrak{s},\mathfrak{h})$ is a connected list segment from x to y via $\langle x_1,\ldots,x_k\rangle$.*

Proof. We just show one direction, as the proof of the other direction is similar. Assume $(\mathfrak{s},\mathfrak{h}) \models \mathsf{ls}^i_{\mathbf{y}}(x,y)$. We proceed by induction on i.

CASE $i = 0$. In this case, $x \approx y$ holds. Consequently, $\mathfrak{h} = \emptyset$ and $\mathfrak{s}(x) = \mathfrak{s}(y)$, implying that $(\mathfrak{s},\mathfrak{h})$ is a connected list segment (via zero intermediate variables).

CASE $i \geq 1$. Assume z is a fresh variable. By definition, there exists a location $\ell \in$ **Loc** such that for $\mathfrak{s}' = \mathfrak{s} \cup \{z \mapsto \ell\}$. $(\mathfrak{s}',\mathfrak{h}) \models (x \mapsto_{\mathsf{n}} z) \star \mathsf{ls}^{i-1}_{\mathbf{z}}(z,\mathbf{y})$ holds.

Let $\mathfrak{h}_1,\mathfrak{h}_2$ be such that $\mathfrak{h} = \mathfrak{h}_1 \uplus^{\mathfrak{s}'} \mathfrak{h}_2$, $(\mathfrak{s}',\mathfrak{h}_1) \models (x \mapsto_{\mathsf{n}} z)$ and $(\mathfrak{s}',\mathfrak{h}_2) \models \mathsf{ls}^{i-1}_{\mathbf{z}}(z,\mathbf{y})$.

By the induction hypothesis, $(\mathfrak{s}',\mathfrak{h}_2)$ is a connected list segment from z to y via some variables $x_1,\ldots,x_k$. In particular, $(\mathfrak{s}',\mathfrak{h}_2)$ is a singly-linked list from z to y. As $x \mapsto_{\mathfrak{h}} z$ holds, it holds that

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

$(\mathfrak{s}', \mathfrak{h})$ is a singly-linked list from x to y and thus $(\mathfrak{s}, \mathfrak{h})$ is a singly-linked list from x to y. If $\mathfrak{s}'(z) \in \mathrm{img}(\mathfrak{s})$, let $x_0 \in \mathfrak{s}^{-1}(\mathfrak{s}'(z))$ and $\mathbf{x} \triangleq \langle x_0 \rangle \cdot \langle x_1, \ldots, x_k \rangle$. Otherwise, let $\mathbf{x} \triangleq \langle x_1, \ldots, x_k \rangle$. It follows that $(\mathfrak{s}, \mathfrak{h})$ is a connected list segment from x to y via $\mathbf{x}$. □

Lemma 5.5 (Characterization of tree models)**.** *Let $(\mathfrak{s}, \mathfrak{h})$ be a model, $x \in \mathbf{Var}$, $\mathbf{y} \in \mathbf{Var}^*$. Then $(\mathfrak{s}, \mathfrak{h}) \models \mathsf{tree}(x, \mathbf{y})$ iff $(\mathfrak{s}, \mathfrak{h})$ is a directed tree with root x and holes $\mathbf{y}$.*

Proof. We prove the more general claim that it holds for all $\mathfrak{s}$ and $\mathfrak{h}$ and all $\mathbf{z} \supseteq \mathbf{y}$ that $(\mathfrak{s}, \mathfrak{h}) \models \mathsf{tree}^i_{\mathbf{z}}(x, \mathbf{y})$ iff $(\mathfrak{s}, \mathfrak{h})$ is a directed tree with root x and holes $\mathbf{y}$.

Since $(\mathfrak{s}, \mathfrak{h}) \models \mathsf{tree}^i_{\mathbf{z}}(x, \mathbf{y})$ implies that $(\mathfrak{s}, \mathfrak{h}) \models \mathsf{tree}^i_{\mathbf{y}}(x, \mathbf{y})$ and thus also that $(\mathfrak{s}, \mathfrak{h}) \models \mathsf{tree}(x, \mathbf{y})$, the original claim follows.

We prove just one direction, as the proof of the other direction is similar. We proceed by mathematical induction on i. Assume $(\mathfrak{s}, \mathfrak{h}) \models \mathsf{tree}^i_{\mathbf{z}}(x, \mathbf{y})$.

CASE $i = 0$. In this case, either $x \approx \mathsf{nil}$ holds or $\mathbf{y} = \langle y \rangle$ and $x \approx y$ holds. Consequently, $\mathfrak{h} = \emptyset$ and $\mathfrak{s}(x) = \mathfrak{s}(\mathsf{nil})$ or $\mathfrak{s}(x) = \mathfrak{s}(y)$, implying that $(\mathfrak{s}, \mathfrak{h})$ is a directed tree with root x and holes $\mathbf{y}$.

CASE $i \geq 1$. Assume z_1, z_2 are fresh variables. By definition, there exist numbers i_1, i_2, sequences of variables $\mathbf{y}_1, \mathbf{y}_2 \in \mathbf{Var}^*$ and locations location $\ell_1, \ell_2 \in \mathbf{Loc}$ such that

1. $\mathbf{y} = \mathbf{y}_1 \cdot \mathbf{y}_2$,
2. $i = i_1 + i_2 + 1$ and thus in particular $i_1, i_2 < i$,
3. for $\mathfrak{s}' = \mathfrak{s} \cup \{z_1, z_2 \mapsto \ell_1, \ell_2\}$

for $\mathfrak{s}' = \mathfrak{s} \cup \{z \mapsto \ell\}$. $(\mathfrak{s}', \mathfrak{h}) \models (x \mapsto_{\mathsf{n}} z) \star \mathsf{ls}^{i-1}_{\mathbf{z}}(z, \mathbf{y})$ holds.

Let $\mathfrak{h}_0, \mathfrak{h}_1, \mathfrak{h}_2$ be such that $\mathfrak{h} = \mathfrak{h}_0 \uplus^{\mathfrak{s}'} \mathfrak{h}_1 \uplus^{\mathfrak{s}'} \mathfrak{h}_2$, $(\mathfrak{s}', \mathfrak{h}_0) \models (x \mapsto_{\mathsf{n}} z)$ and $(\mathfrak{s}', \mathfrak{h}_1) \models \mathsf{tree}^{i_1}_{\mathbf{z}}(z_1, \mathbf{y}_1)$ and $(\mathfrak{s}', \mathfrak{h}_1) \models \mathsf{tree}^{i_2}_{\mathbf{z}}(z_2, \mathbf{y}_2)$.

Note that for $j \in \{1, 2\}$, it holds that (1) $i_j < i$ and (2) $\mathbf{y}_1 \subseteq \mathbf{y} \subseteq \mathbf{z}$. Consequently, we have by the induction hypotheses that $(\mathfrak{s}', \mathfrak{h}_j)$ is a directed tree with rot z_j and holes $\mathbf{y}_j$. Moreover, $x \notin \mathbf{z}$ by definition of $\mathsf{tree}^i_{\mathbf{z}}$, so $x \notin \mathbf{y}_1$ and $x \notin \mathbf{y}_2$. Further, $x \mapsto_{\mathfrak{h}} z_1$ and $x \mapsto_{\mathfrak{h}} z_2$ hold. It follows that $(\mathfrak{s}', \mathfrak{h})$ is a directed tree with root x and holes $\mathbf{y}$. Since $x \in \mathrm{dom}(\mathfrak{s})$ and $\mathbf{y} \subseteq \mathrm{dom}(\mathfrak{s})$, it follows that $(\mathfrak{s}, \mathfrak{h})$ is a directed tree with root x and holes $\mathbf{y}$. □

ISOMORPHISM. Recall from Definition 2.2 the notion of isomorphic models. **SSL** cannot distinguish between isomorphic models.

We first show this for list and tree predicates.

Lemma 5.6. *Let $(\mathfrak{s}, \mathfrak{h}), (\mathfrak{s}', \mathfrak{h}')$ be models with $(\mathfrak{s}, \mathfrak{h}) \cong (\mathfrak{s}', \mathfrak{h}')$. Let $\mathsf{pred} \in \{\mathsf{ls}, \mathsf{ls}_{\geq 2}, \mathsf{tree}, \mathsf{tree}_{\geq 2}\}$. Then $(\mathfrak{s}, \mathfrak{h}) \models \mathsf{pred}(x, \mathbf{y})$ iff $(\mathfrak{s}', \mathfrak{h}') \models \mathsf{pred}(x, \mathbf{y})$.*

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Proof. Let $\sigma\colon \mathbf{Val} \to \mathbf{Val}$ be a witness of the isomorphism.

We show the claim for $\mathsf{pred} = \mathsf{ls}$. We show just one direction of the proof, as the proof of the other direction is completely analogous.

Assume $(\mathfrak{s},\mathfrak{h}) \models \mathsf{ls}(x,\mathbf{y})$. Then there exists an i such that $(\mathfrak{s},\mathfrak{h}) \models \mathsf{ls}^i_{\mathbf{y}}(x,\mathbf{y}) \star \mathsf{distinct}(\mathbf{y} \cup \{\mathsf{nil}\})$. As σ is a bijection and $\mathfrak{s}'(x) = \sigma(\mathfrak{s}(x))$ for all x, it follows that

$$(\mathfrak{s},\mathfrak{h}) \models \mathsf{distinct}(\mathbf{y} \cup \{\mathsf{nil}\}) \text{ iff } (\mathfrak{s}',\mathfrak{h}') \models \mathsf{distinct}(\mathbf{y} \cup \{\mathsf{nil}\}).$$

It remains to be shown that $(\mathfrak{s}',\mathfrak{h}') \models \mathsf{ls}^i_{\mathbf{y}}(x,\mathbf{y})$. We show the more general claim that for all $(\mathfrak{s},\mathfrak{h})$ and $(\mathfrak{s}',\mathfrak{h}')$ with $(\mathfrak{s},\mathfrak{h}) \cong (\mathfrak{s}',\mathfrak{h}')$ and for all sequences of variables $\mathbf{z}$ and $\mathbf{y}$, if $(\mathfrak{s},\mathfrak{h}) \models \mathsf{ls}^i_{\mathbf{z}}(x,\mathbf{y})$ then $(\mathfrak{s}',\mathfrak{h}') \models \mathsf{ls}^i_{\mathbf{z}}(x,\mathbf{y})$. We proceed by mathematical induction on i.

CASE $i = 0$. In this case, $\mathfrak{h} = \emptyset$, so $\mathfrak{h}' = \emptyset$. Moreover, $(\mathfrak{s},\mathfrak{h}) \models x \approx y$ iff $(\mathfrak{s}',\mathfrak{h}') \models x \approx y$, because σ is a bijection. Consequently, $(\mathfrak{s}',\mathfrak{h}') \models \mathsf{ls}^0_{\mathbf{z}}(x,\mathbf{y})$.

CASE $i > 0$. Let z be a fresh variable. There exist $\ell \in \mathbf{Loc}$ and $\mathfrak{s}_1 \in \mathbf{Stacks}$, $\mathfrak{s}_1 = \mathfrak{s} \cup \{z \mapsto \ell\}$ and $\mathfrak{s}_1(x) \notin \mathfrak{s}_1(\mathbf{z})$ and $(\mathfrak{s}_1,\mathfrak{h}) \models (x \mapsto_{\mathsf{n}} z) \star \mathsf{ls}^{i-1}_{\mathbf{z}}(z,\mathbf{y})$.

Define $\mathfrak{h}_1 \triangleq \{\mathfrak{s}(x) \mapsto \mathfrak{h}(\mathfrak{s}(x))\}$ and $\mathfrak{h}_2 \triangleq \mathfrak{h} - \mathfrak{s}(x)$. Observe that $(\mathfrak{s}_1,\mathfrak{h}_1) \models (x \mapsto_{\mathsf{n}} z)$ (and in particular $\mathfrak{h}_1 = \{\mathfrak{s}(x) \mapsto \langle \ell, d\rangle\}$) and $(\mathfrak{s}_1,\mathfrak{h}_2) \models \mathsf{ls}^{i-1}_{\mathbf{z}}(z,\mathbf{y})$.

Let $\ell' \triangleq \sigma(\ell)$ and $\mathfrak{s}'_1 \triangleq \mathfrak{s}' \cup \{z \mapsto \ell', w \mapsto d\}$. Because σ is a bijection, $\mathfrak{s}'_1(x) \notin \mathfrak{s}'_1(\mathbf{z})$.

Let $\mathfrak{h}'_1 \triangleq \{\mathfrak{s}'(x) \mapsto \mathfrak{h}'(\mathfrak{s}'(x))\}$ and $\mathfrak{h}'_2 \triangleq \mathfrak{h}' - \mathfrak{s}'(x)$. By construction, $\mathfrak{h}'_1 = \{\mathfrak{s}'(x) \mapsto \langle \ell', d\rangle\} = \{\sigma(\mathfrak{s}(x)) \mapsto \langle \sigma(\ell), \sigma(d)\rangle\}$. In particular, $(\mathfrak{s}'_1,\mathfrak{h}'_1) \models (x \mapsto_{\mathsf{n}} z)$.

Moreover, $\mathfrak{h}'_2 = \{\sigma(\bar{\ell}) \mapsto \sigma(\mathfrak{h}(\bar{\ell})) \mid \bar{\ell} \in \mathrm{dom}(\mathfrak{h}) \setminus \{\mathfrak{s}(x)\}\}$. Therefore, $(\mathfrak{s}_1,\mathfrak{h}_2) \cong (\mathfrak{s}'_1,\mathfrak{h}'_2)$. It follows by the induction hypothesis that $(\mathfrak{s}'_1,\mathfrak{h}'_2) \models \mathsf{ls}^{i-1}_{\mathbf{z}}(z,\mathbf{y})$, and thus $(\mathfrak{s}'_1,\mathfrak{h}') \models (x \mapsto_{\mathsf{n}} z) \star \mathsf{ls}^{i-1}_{\mathbf{z}}(z,\mathbf{y})$, which in turn implies that $(\mathfrak{s}',\mathfrak{h}') \models \mathsf{ls}^i_{\mathbf{z}}(x,\mathbf{y})$ by the semantics of ls^i.

The identical proof strategy can be used to prove the claim for $\mathsf{pred} \in \{\mathsf{ls}_{\geq 2}, \mathsf{tree}, \mathsf{tree}_{\geq 2}\}$. □

Lemma 5.7. *Let $(\mathfrak{s},\mathfrak{h}),(\mathfrak{s}',\mathfrak{h}')$ be models with $(\mathfrak{s},\mathfrak{h}) \cong (\mathfrak{s}',\mathfrak{h}')$ and let $\phi \in$ **SSL**. Then $(\mathfrak{s},\mathfrak{h}) \models \phi$ iff $(\mathfrak{s}',\mathfrak{h}') \models \phi$.*

Proof. We prove the claim by induction on the structure of the formula ϕ. Clearly, the claim holds for the base cases $x \mapsto_{\mathsf{n}} y$, $x \mapsto_{\mathsf{l,r}} \langle y_1, y_2\rangle$, $x \approx y$ and $x \not\approx y$. For $\mathsf{pred} \in \{\mathsf{ls}, \mathsf{ls}_{\geq 2}, \mathsf{tree}, \mathsf{tree}_{\geq 2}\}$, the claim was proved in Lemma 5.6.

Further, the claim immediately follows from the induction assumption for the cases $\phi_1 \wedge \phi_2$, $\phi_1 \vee \phi_2$ and $\neg\phi$. It remains to consider the

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

cases $\phi_1 \star \phi_2$ and $\phi_1 \mathrel{-\!\!\star} \phi_2$. Let $(\mathfrak{s}, \mathfrak{h})$ and $(\mathfrak{s}', \mathfrak{h}')$ be two stack-heap pairs with $(\mathfrak{s}, \mathfrak{h}) \cong (\mathfrak{s}', \mathfrak{h}')$.

We will show that $(\mathfrak{s}, \mathfrak{h}) \models \phi_1 \star \phi_2$ implies $(\mathfrak{s}', \mathfrak{h}') \models \phi_1 \star \phi_2$; the other direction is completely symmetric. We assume that $(\mathfrak{s}, \mathfrak{h}) \models \phi_1 \star \phi_2$. Then, there are $\mathfrak{h}_1, \mathfrak{h}_2$ with $\mathfrak{h}_1 \uplus^{\mathfrak{s}} \mathfrak{h}_2 = \mathfrak{h}$ and $(\mathfrak{s}, \mathfrak{h}_i) \models \phi_i$ for $i = 1, 2$. We consider a bijection σ that is an isomorphism between $(\mathfrak{s}, \mathfrak{h})$ and $(\mathfrak{s}', \mathfrak{h}')$. Let $\mathfrak{h}'_1$ respectively $\mathfrak{h}'_2$ be the sub-heap of $\mathfrak{h}'$ restricted to $\sigma(\mathrm{dom}(\mathfrak{h}_1))$ resp. $\sigma(\mathrm{dom}(\mathfrak{h}_2))$. It is easy to verify that $\mathfrak{h}'_1 \uplus^{\mathfrak{s}} \mathfrak{h}'_2 = \mathfrak{h}'$ and $(\mathfrak{s}, \mathfrak{h}_i) \cong (\mathfrak{s}', \mathfrak{h}'_i)$ for $i = 1, 2$. Hence, we can apply the induction assumption and get that $(\mathfrak{s}', \mathfrak{h}'_i) \models \phi_i$ for $i = 1, 2$. Because of $\mathfrak{h}'_1 \uplus^{\mathfrak{s}} \mathfrak{h}'_2 = \mathfrak{h}'$ we get $(\mathfrak{s}', \mathfrak{h}') \models \phi_1 \star \phi_2$.

We will show that $(\mathfrak{s}, \mathfrak{h}) \models \phi_1 \mathrel{-\!\!\star} \phi_2$ implies $(\mathfrak{s}', \mathfrak{h}') \models \phi_1 \mathrel{-\!\!\star} \phi_2$; the other direction is completely symmetric. We assume that $(\mathfrak{s}, \mathfrak{h}) \models \phi_1 \mathrel{-\!\!\star} \phi_2$. Let $\mathfrak{h}'_0$ be a heap with $(\mathfrak{s}', \mathfrak{h}'_0) \models \phi_1$ and $\mathfrak{h}'_0 \uplus^{\mathfrak{s}} \mathfrak{h}' \neq \bot$. We consider a bijection σ that is an isomorphism between $(\mathfrak{s}, \mathfrak{h})$ and $(\mathfrak{s}', \mathfrak{h}')$. Let $L \subseteq \mathbf{Loc}$ be some subset of locations with $L \cap (\mathsf{locs}(\mathfrak{h}) \cup \mathrm{img}(\mathfrak{s})) = \emptyset$, $L \cap (\mathsf{locs}(\mathfrak{h}') \cup \mathrm{img}(\mathfrak{s}')) = \emptyset$, and $|L| = \mathsf{locs}(\mathfrak{h}'_0) \setminus (\mathsf{locs}(\mathfrak{h}') \cup \mathrm{img}(\mathfrak{s}'))$. We can assume w.l.o.g. that the restriction of σ to L is a bijection from L to $\mathsf{locs}(\mathfrak{h}'_0)$. Then, σ induces a heap $\mathfrak{h}_0$ such that $(\mathfrak{s}, \mathfrak{h}_0) \cong (\mathfrak{s}', \mathfrak{h}'_0)$, $\mathfrak{h}_0 \uplus^{\mathfrak{s}} \mathfrak{h} \neq \bot$ and $(\mathfrak{s}, \mathfrak{h}_0 \uplus^{\mathfrak{s}} \mathfrak{h}) \cong (\mathfrak{s}', \mathfrak{h}'_0 \uplus^{\mathfrak{s}} \mathfrak{h}')$. By induction assumption we get that $(\mathfrak{s}, \mathfrak{h}_0) \models \phi_1$. From the assumption $(\mathfrak{s}, \mathfrak{h}) \models \phi_1 \mathrel{-\!\!\star} \phi_2$ and $\mathfrak{h}_0 \uplus^{\mathfrak{s}} \mathfrak{h} \neq \bot$ we now get that $(\mathfrak{s}, \mathfrak{h}_0 \uplus^{\mathfrak{s}} \mathfrak{h}) \models \phi_2$. Again from the induction assumption we finally get that $(\mathfrak{s}', \mathfrak{h}'_0 \uplus^{\mathfrak{s}} \mathfrak{h}') \models \phi_2$. □

SEMANTIC CONSEQUENCE. We denote by $\phi \models \psi$ that ϕ entails ψ, i.e., that for all $(\mathfrak{s}, \mathfrak{h})$, if $(\mathfrak{s}, \mathfrak{h}) \models \phi$ then also $(\mathfrak{s}, \mathfrak{h}) \models \psi$. Moreover, we define a restricted notion of entailment to stacks $\mathfrak{s}$ with $\mathrm{dom}(\mathfrak{s}) \subseteq \mathbf{x}$. Formally, $\phi \models_{\mathbf{x}} \psi$ iff for all $(\mathfrak{s}, \mathfrak{h})$, if $\mathrm{dom}(\mathfrak{s}) \subseteq \mathbf{x}$ and $(\mathfrak{s}, \mathfrak{h}) \models \phi$ then also $(\mathfrak{s}, \mathfrak{h}) \models \psi$.

Because of the strong-separation semantics, the number of possible shapes of the heap of the models of ϕ over $\mathbf{x}$ may be smaller than over $\mathbf{y} \supseteq \mathbf{x}$. For this reason, it is possible that $\phi \models_{\mathbf{x}} \psi$, but $\phi \not\models_{\mathbf{y}} \psi$.

Example 5.8 (Indexed entailment)**.** *It holds that*

$$\mathsf{ls}(x, y) \models_{\{x,y\}} \neg((\neg\mathbf{emp}) \star (\neg\mathbf{emp})),$$

because it is impossible to split a list into two non-empty sub-lists using $\uplus^{\mathfrak{s}}$ without access to a variable at which we can split the list. (Recall that our semantics precludes cyclic lists, so a model of $\mathsf{ls}(x, y)$ is never a model of $\mathsf{ls}(x, y) \star \mathsf{ls}(y, y)$.) Conversely,

$$\mathsf{ls}(x, y) \not\models_{\{x,y,z\}} \neg((\neg\mathbf{emp}) \star (\neg\mathbf{emp})).$$

For example, if $\mathfrak{s} = \{x \mapsto \ell_1, y \mapsto \ell_2, z \mapsto \ell_3\}$ and $\mathfrak{h} = \{\ell_1 \mapsto \ell_3, \ell_3 \mapsto \ell_2\}$, it holds that $\mathfrak{h} = \{\ell_1 \mapsto \ell_3\} \uplus^{\mathfrak{s}} \{\ell_3 \mapsto \ell_2\}$, and thus both $(\mathfrak{s}, \mathfrak{h}) \models \mathsf{t}$ and $(\mathfrak{s}, \mathfrak{h}) \models (\neg\mathbf{emp}) \star (\neg\mathbf{emp})$.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

POSITIVE FORMULAS VS. SYMBOLIC HEAPS.

Definition 5.9 (Positive model)**.** *Let $(\mathfrak{s}, \mathfrak{h})$ be a model. $(\mathfrak{s}, \mathfrak{h})$ is a* positive model *if there exists a formula $\phi \in \mathbf{SSL}^+$ such that $(\mathfrak{s}, \mathfrak{h}) \models \phi$.*

Clearly, every model of a positive formula satisfies at least one symbolic heap.

Lemma 5.10. *Let $(\mathfrak{s}, \mathfrak{h})$ be a positive model. Then there exist $k \geq 0$ and spatial atoms $\tau_1, \ldots, \tau_k$ such that $(\mathfrak{s}, \mathfrak{h}) \models \tau_1 \star \cdots \star \tau_k$.*

Proof. By definition, there exists a formula $\phi \in \mathbf{SSL}^+$ such that $(\mathfrak{s}, \mathfrak{h}) \models \phi$. We proceed by induction on ϕ. If ϕ is spatial, there is nothing to show. If $\phi = \phi_1 \wedge \phi_2$ or $\phi = \phi_1 \wedge \neg\phi_2$, then $(\mathfrak{s}, \mathfrak{h}) \models \phi_1$ by the semantics of $\wedge$ and the result follows immediately from the induction hypothesis for ϕ_1. If $\phi = \phi_1 \vee \phi_2$ then either $(\mathfrak{s}, \mathfrak{h}) \models \phi_1$ or $(\mathfrak{s}, \mathfrak{h}) \models \phi_2$. Assume w.l.o.g. that $(\mathfrak{s}, \mathfrak{h}) \models \phi_1$. The result follows immediately from the induction hypothesis for ϕ_1. □

5.1.4 *Strong-Separation Logic vs Weak-Separation Logic*

We contrast the semantics of strong-separation logic and the standard "weak-separation" semantics.

The syntax of weak-separation logic **WSL** is like the syntax of **SSL**, except that we use weak versions of the separating connectives, denoted by a **w** superscript. In particular, positive weak-separation logic, $\mathbf{WSL}^+$, is like $\mathbf{SSL}^+$ except that we use $\star^{\mathbf{w}}$ instead of $\star$ as separating conjunction. Formally, for ϕ_{atom} as in Fig. 5.1, we collect formulas of the form $\phi^{\mathbf{w}}_{\mathsf{pos}}$, defined below, in the set $\mathbf{WSL}^+$.

$$\phi^{\mathbf{w}}_{\mathsf{spatial}} ::= \phi_{\mathsf{atom}} \mid \phi_{\mathsf{spatial}} \star^{\mathbf{w}} \phi_{\mathsf{spatial}}$$
$$\phi^{\mathbf{w}}_{\mathsf{pos}} ::= \phi^{\mathbf{w}}_{\mathsf{spatial}} \mid \phi^{\mathbf{w}}_{\mathsf{pos}} \vee \phi^{\mathbf{w}}_{\mathsf{pos}} \mid \phi^{\mathbf{w}}_{\mathsf{pos}} \wedge \phi^{\mathbf{w}}_{\mathsf{pos}} \mid \phi^{\mathbf{w}}_{\mathsf{pos}} \wedge \neg\phi^{\mathbf{w}}_{\mathsf{pos}}$$

We use "normal" disjoint union, $+$, to give a semantics to the weak separating conjunction, $\star^{\mathbf{w}}$:

$$(\mathfrak{s}, \mathfrak{h}) \models \phi_1 \star^{\mathbf{w}} \phi_2 \quad \text{iff} \quad \exists \mathfrak{h}_1, \mathfrak{h}_2 \in \mathbf{Heaps}.\, \mathfrak{h} = \mathfrak{h}_1 + \mathfrak{h}_2, \; (\mathfrak{s}, \mathfrak{h}_1) \models \phi_1, \text{ and } (\mathfrak{s}, \mathfrak{h}_2) \models \phi_2$$

In other words, what I call *weak* here is really the *standard* semantics of $\star$ in the separation-logic literature [Reyo2; OHe19], which I also used in the semantics of $\mathbf{SL}_{\mathbf{base}}$ in Part i. I use the adjective *weak* in this part of the thesis to explicitly contrast this semantics to the *strong* semantics of $\star$, which imposes stronger requirements on sub-heap composition: sub-heaps may only overlap at locations that are stored in the stack.

Example 5.11 (SSL vs. WSL)**.** *Let $\phi \triangleq (\mathsf{ls}(a, \mathsf{nil}) \star \mathsf{t}) \wedge (\mathsf{ls}(b, \mathsf{nil}) \star \mathsf{t})$, where* t *is the derived formula representing true. In Fig. 5.4, we show two*

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

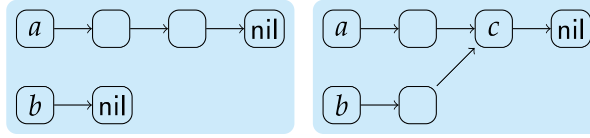


Figure 5.4: Models of $(\mathsf{ls}(a, \mathsf{nil}) \star \mathsf{t}) \wedge (\mathsf{ls}(b, \mathsf{nil}) \star \mathsf{t})$ for a stack with domain a, b and a stack with domain a, b, c. I do not show data fields to reduce clutter.

models of ϕ. On the left, we assume that a, b are the only variables in $\mathrm{dom}(\mathfrak{s})$, *whereas on the right, we assume that there is a third stack variable c.*

Note that the latter model, where the two lists overlap, is possible in SSL only *because the lists come together at the location labeled by c. If we removed the variable c from the stack, the model would still satisfy* $(\mathsf{ls}(a, \mathsf{nil}) \star^{\mathbf{w}} \mathsf{t}) \wedge (\mathsf{ls}(b, \mathsf{nil}) \star^{\mathbf{w}} \mathsf{t})$, *but no longer satisfy ϕ according to the strong semantics, because $\uplus^{\mathfrak{s}}$ would no longer allow splitting the heap at that location.*

As I mentioned in Chapter 2, a lot of work on automated deduction for and program analysis with separation logic is limited to *symbolic heaps*. Interestingly, the semantics of WSL and SSL coincide on the symbolic-heap fragment—in fact, they coincide on the positive fragment.

Let us make this claim precise. Let $\phi \in \mathbf{SSL}^+$. We denote by $\mathsf{weaken}(\phi)$ the $\mathbf{WSL}^+$ formula obtained by replacing every occurrence of $\star$ with $\star^{\mathbf{w}}$.

Theorem 5.12. *Let $\phi \in \mathbf{SSL}^+$ and let $(\mathfrak{s}, \mathfrak{h})$ be a model. Then $(\mathfrak{s}, \mathfrak{h}) \models \phi$ iff $(\mathfrak{s}, \mathfrak{h}) \models \mathsf{weaken}(\phi)$.*

Theorem 5.12 holds because in models of positive formulas, all dangling locations are labeled by variables of the formula or nil.

Lemma 5.13. *Let $\phi \in \mathbf{SSL}^+ \cup \mathbf{WSL}^+$ be positive and $(\mathfrak{s}, \mathfrak{h}) \models \phi$. Then* $\mathsf{dangling}(\mathfrak{h}) \subseteq \mathfrak{s}(\mathsf{fvars}(\phi)) \cup \{\mathfrak{s}(\mathsf{nil})\}$.

Proof. By a straightforward structural induction on ϕ: let $\phi \in \mathbf{SSL}^+ \cup \mathbf{WSL}^+$ and let $(\mathfrak{s}, \mathfrak{h})$ be a model with $(\mathfrak{s}, \mathfrak{h}) \models \phi$.

CASE $\phi = x \approx y$ OR $\phi = x \not\approx y$. It holds that $\mathfrak{h} = \emptyset$, so trivially

$$\mathsf{dangling}(\mathfrak{h}) = \emptyset \subseteq \mathfrak{s}(\mathsf{fvars}(\phi)) \cup \{\mathfrak{s}(\mathsf{nil})\}.$$

CASE $\phi = x \mapsto_{\mathsf{n}} y$. It holds that $\mathsf{dangling}(\mathfrak{h}) \subseteq \{\mathfrak{s}(y)\} \subseteq \mathfrak{s}(\mathsf{fvars}(\phi)) \cup \{\mathfrak{s}(\mathsf{nil})\}$.

CASE $\phi = x \mapsto_{\mathsf{l},\mathsf{r}} \langle y_1, y_2 \rangle$. It holds that $\mathsf{dangling}(\mathfrak{h}) \subseteq \{\mathfrak{s}(y_1), \mathfrak{s}(y_2)\} \subseteq \mathfrak{s}(\mathsf{fvars}(\phi)) \cup \{\mathfrak{s}(\mathsf{nil})\}$.

CASE $\phi = \mathsf{ds}(x, \mathbf{y})$. The semantics enforce that $\mathsf{dangling}(\mathfrak{h}) \subseteq \mathfrak{s}(\mathbf{y}) \cup \{\mathfrak{s}(\mathsf{nil})\} \subseteq \mathfrak{s}(\mathsf{fvars}(\phi)) \cup \{\mathfrak{s}(\mathsf{nil})\}$.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

CASE $\phi = \phi_1 \star \phi_2$. There exist $\mathfrak{h}_1, \mathfrak{h}_2$ with $(\mathfrak{s}, \mathfrak{h}_1) \models \phi_1$ and $(\mathfrak{s}, \mathfrak{h}_2) \models \phi_2$ and $\mathfrak{h} = \mathfrak{h}_1 \uplus^{\mathfrak{s}} \mathfrak{h}_2$. By the induction hypotheses, $\mathsf{dangling}(\mathfrak{h}_i) \subseteq \mathfrak{s}(\mathsf{fvars}(\phi)) \cup \{\mathfrak{s}(\mathsf{nil})\}$. Since

$$\mathsf{dangling}(\mathfrak{h}) \subseteq \mathsf{dangling}(\mathfrak{h}_1) \cup \mathsf{dangling}(\mathfrak{h}_2),$$

the claim follows.

CASE $\phi = \phi_1 \star^{\mathbf{w}} \phi_2$. Analogously, except that $\mathfrak{h} = \mathfrak{h}_1 + \mathfrak{h}_2$.

CASE $\phi = \phi_1 \wedge \phi_2$, $\phi = \phi_1 \vee \phi_2$ The claim follows immediately from the induction hypotheses. □

As every location that is shared between heaps $\mathfrak{h}_1 + \mathfrak{h}_2$ is dangling either in $\mathfrak{h}_1$ or in $\mathfrak{h}_2$ (or both), the union operations $+$ and $\uplus^{\mathfrak{s}}$ coincide on models of positive formulas.

Lemma 5.14. *Let $(\mathfrak{s}, \mathfrak{h}_1) \models \phi_1$ and $(\mathfrak{s}, \mathfrak{h}_2) \models \phi_2$ for positive formulas ϕ_1, ϕ_2. Then $\mathfrak{h}_1 + \mathfrak{h}_2 \neq \bot$ iff $\mathfrak{h}_1 \uplus^{\mathfrak{s}} \mathfrak{h}_2 \neq \bot$.*

Proof. If $\mathfrak{h}_1 \uplus^{\mathfrak{s}} \mathfrak{h}_2 \neq \bot$, then $\mathfrak{h}_1 + \mathfrak{h}_2 \neq \bot$ by definition.

Conversely, assume $\mathfrak{h}_1 + \mathfrak{h}_2 \neq \bot$. We need to show that $\mathsf{locs}(\mathfrak{h}_1) \cap \mathsf{locs}(\mathfrak{h}_2) \subseteq \mathrm{img}(\mathfrak{s})$. To this end, let $\ell \in \mathsf{locs}(\mathfrak{h}_1) \cap \mathsf{locs}(\mathfrak{h}_2)$. Then there exists an $i \in \{1, 2\}$ such that $i \in \mathrm{img}(\mathfrak{h}_i) \setminus \mathrm{dom}(\mathfrak{h}_i)$—otherwise ℓ would be in $\mathrm{dom}(\mathfrak{h}_1) \cap \mathrm{dom}(\mathfrak{h}_2)$ and $\mathfrak{h}_1 + \mathfrak{h}_2 = \bot$. By Lemma 5.13, we thus have $\ell \in \mathrm{img}(\mathfrak{s})$. □

Since the semantics coincide on atomic formulas by definition and on $\star$ by Lemma 5.13, we can easily show that they coincide on all positive formulas.

Proof of Theorem 5.12. We proceed by structural induction on ϕ. If ϕ is atomic, $\mathsf{weaken}(\phi) = \phi$, so there is nothing to show.

For $\phi = \phi_1 \star \phi_2$, $\mathsf{weaken}(\phi) = \phi_1 \star^{\mathbf{w}} \phi_2$, and the claim follows from the induction hypotheses and Lemma 5.14. For $\phi = \phi_1 \wedge \phi_2$, $\phi = \phi_1 \vee \phi_2$, and $\phi = \phi_1 \wedge \neg\phi_2$, the claim follows immediately from the induction hypotheses and the semantics of $\wedge$, $\vee$. □

A difference between WSL and SSL: Size formulas

At this point, you might be wondering what the differences in the expressiveness of WSL and SSL actually are. One such difference is that the former allows reasoning about heap size, whereas the latter does not. Given

$$(\mathsf{size} \geq k) \triangleq \underbrace{(\neg\mathbf{emp}) \star^{\mathbf{w}} \cdots \star^{\mathbf{w}} (\neg\mathbf{emp})}_{k \text{ times}} \text{ and}$$

$$(\mathsf{size} = k) \triangleq (\mathsf{size} \geq k) \wedge (\neg(\mathsf{size} \geq k+1)),$$

it holds that $(\mathfrak{s}, \mathfrak{h}) \models (\mathsf{size} \geq k)$ iff $|\mathfrak{h}| \geq k$ and $(\mathfrak{s}, \mathfrak{h}) \models (\mathsf{size} = k)$ iff $|\mathfrak{h}| = k$. Such size formulas are important ingredients of some of

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Wien Bibliothek Your knowledge hub

the undecidability proofs for SL with the standard weak semantics, see e.g. [DD14; DD15b; DLM18; EIP19b]. It is only possible to express exact heap size in SSL using explicit points-to assertions.

5.2 ADDING DATA PREDICATES TO SSL

In this section, we add to **SSL** the possibility to constrain the data values stored inside the heap by extending the atomic formulas of **SSL** with three new types of atomic formulas.

1. We add quantifier-free formulas from the data theory $\mathcal{T}_{\mathbf{Data}}$.
2. We add a variant of the points-to assertions that allows specifying the content of the data field.
3. We allow annotating list and tree predicates with *data predicates* as proposed in Section 3.2.

We call the extended logic $\mathbf{SSL}_{\mathsf{data}}$. The motivation behind $\mathbf{SSL}_{\mathsf{data}}$ is to be able to reason about properties of inductive structures that appear frequently in practice. For example, we would like to be able to express that a data structure contains a certain value, a list is sorted, or a tree is a binary search tree.

I begin with a semi-formal introduction to data predicates in Section 5.2.1, then formalize the extension to $\mathbf{SSL}_{\mathsf{data}}$ in Section 5.2.2.

5.2.1 *A Semi-Formal Introduction to SSL with Data Predicates*

We assume two dedicated fresh variables α and β from **Var** of sort **Data** to be used exclusively in data predicates. We consider four kinds of data predicates:

1. *Existential unary data predicates* $[F]^{\exists}$, where $F \in \mathcal{F}_{\mathbf{Data}}$, $\alpha \in \mathsf{fvars}(F)$, $\beta \notin \mathsf{fvars}(F)$.
2. *Existential binary data predicates* $[\mathsf{f} \colon F]^{\exists}$, where $F \in \mathcal{F}_{\mathbf{Data}}$, $\alpha, \beta \in \mathsf{fvars}(F)$, $\mathsf{f} \in \{\mathsf{n}, \mathsf{l}, \mathsf{r}\}$.
3. *Universal unary data predicates* $[F]^{\forall}$, where $F \in \mathcal{F}_{\mathbf{Data}}$, $\alpha \in \mathsf{fvars}(F)$, $\beta \notin \mathsf{fvars}(F)$.
4. *Universal binary data predicates* $[\mathsf{f} \colon F]^{\forall}$, where $F \in \mathcal{F}_{\mathbf{Data}}$, $\alpha, \beta \in \mathsf{fvars}(F)$, $\mathsf{f} \in \{\mathsf{n}, \mathsf{l}, \mathsf{r}\}$.

All four types of predicates may also contain other variables from $\mathbf{Var} \setminus \{\alpha, \beta\}$. We pass a set of data predicates $\mathcal{P}$ as additional parameter to the list and tree predicates, obtaining ternary predicates $\mathsf{ls}(x, \mathbf{y}, \mathcal{P})$, $\mathsf{tree}(x, \mathbf{y}, \mathcal{P})$, and so on. As before, if either of $\mathbf{y}$ or $\mathcal{P}$ is empty, we omit them; and if either of them contain only a single element, we omit the

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

braces. Semi-formally, the semantics of an inductive data predicate $\mathsf{ds}(x, \mathbf{y}, \mathcal{P})$ are as follows:

1. The predicate holds in $(\mathfrak{s}, \mathfrak{h})$ only if it holds without the data constraints, i.e., $(\mathfrak{s}, \mathfrak{h}) \models \mathsf{ds}(x, \mathbf{y})$ holds and $(\mathfrak{s}, \mathfrak{h})$ therefore describes a ds data structure.

2. Let $\circlearrowright \in \{\exists, \forall\}$. For each unary data predicate $[F(\alpha)]^{\circlearrowright} \in \mathcal{P}$,
   - If $\circlearrowright = \exists$, then there must exist a location $\ell \in \mathrm{dom}(\mathfrak{h})$ whose data field satisfies F, i.e., there exists an $\ell \in \mathrm{dom}(\mathfrak{h})$ such that $(\mathfrak{s} \cup \{\alpha \mapsto \mathsf{d}(\ell)\}, \mathfrak{h}) \models_{\mathbf{Data}} F$ holds.
   - If $\circlearrowright = \forall$, then the data fields of all locations $\ell \in \mathrm{dom}(\mathfrak{h})$ must satisfy F, i.e., for all $\ell \in \mathrm{dom}(\mathfrak{h})$,

     $$(\mathfrak{s} \cup \{\alpha \mapsto \mathsf{d}(\ell)\}, \mathfrak{h}) \models_{\mathbf{Data}} F$$

     holds.

3. Let $\circlearrowright \in \{\exists, \forall\}$ and $\mathsf{f} \in \{\mathsf{n}, \mathsf{l}, \mathsf{r}\}$. For each binary data predicate $[\mathsf{f}\colon F(\alpha, \beta)]^{\circlearrowright} \in \mathcal{P}$,
   - If $\circlearrowright = \exists$, there must exist a location ℓ_1 and an f-descendant ℓ_2 of ℓ_1 such that F holds when evaluated on the data fields of ℓ_1 and ℓ_2. Formally, $\mathsf{f}(\ell_1) \mapsto^*_{\mathfrak{h}} \ell_2$ and $(\mathfrak{s} \cup \{\alpha \mapsto \mathsf{d}(\ell_1), \beta \mapsto \mathsf{d}(\ell_2)\}, \mathfrak{h}) \models_{\mathbf{Data}} F$ holds.
   - If $\circlearrowright = \forall$, then it must hold for all pairs of locations ℓ_1 and ℓ_2 such that ℓ_2 is an f-descendant of ℓ_1 that F holds when evaluated on the data fields of ℓ_1 and ℓ_2. Formally, for all $\ell_1, \ell_2 \in \mathrm{dom}(\mathfrak{h})$, if $\mathsf{f}(\ell_1) \mapsto^*_{\mathfrak{h}} \ell_2$ then $(\mathfrak{s} \cup \{\alpha \mapsto \mathsf{d}(\ell_1), \beta \mapsto \mathsf{d}(\ell_2)\}, \mathfrak{h}) \models_{\mathbf{Data}} F$ holds.

Example 5.15 (Data predicates). *I illustrate the use of data predicates through representative examples of data predicates over lists and trees. The predicates*

$$\mathsf{ls}(x, [\alpha = 0]^{\forall}), \quad \mathsf{ls}(x, [\mathsf{n}\colon \alpha \not\approx \beta]^{\forall}, \quad \mathsf{ls}(x, [\mathsf{n}\colon \alpha < \beta]^{\forall})$$

describe lists with all data values equal to 0, lists with all data values distinct, and lists with data values that are strictly increasing, i.e., sorted lists with distinct values. The predicates

$$\mathsf{ls}(x, [\alpha = 17]^{\exists}), \quad \mathsf{ls}(x, [\mathsf{n}\colon \alpha \geq \beta]^{\exists})$$

describe lists that contain the data value 17 *and lists whose values are* not *increasing, i.e., lists that are not sorted. The predicates*

$$\mathsf{tree}(x, \left\{[\mathsf{l}\colon \beta < \alpha]^{\forall}, [\mathsf{r}\colon \beta > \alpha]^{\forall}\right\}),$$
$$\mathsf{tree}(x, \left\{[\mathsf{l}\colon \beta < \alpha]^{\forall}, [\mathsf{r}\colon \beta < \alpha]^{\forall}\right\})$$

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Wien Bibliothek Your knowledge hub

describe a binary search tree, and a max-heap.

The formula $\mathsf{ls}(x, m, [\alpha < M]^{\forall}) \star (m \mapsto_{\mathsf{ls}} \langle y, M\rangle) \star \mathsf{ls}(y, [\alpha > M]^{\forall})$ *describes a partitioned list. The left partition contains elements smaller than the pivot m and the right partition contains elements larger than the pivot m. Here, M is a fresh variable of sort* **Data** *that represents the value* $\mathsf{d}(m)$.

The formula $\mathsf{ls}(x, [\alpha \approx c]^{\exists}) \star \mathsf{ls}(y, [\alpha \not\approx c]^{\forall})$, *where c is a fresh variable of sort* **Data**, *describes lists that have different contents: the list x contains the data value c and the list y does not contain c.*

Restrictions on formulas in data predicates?

You may be wondering if we have to impose restrictions on the $\mathcal{F}_{\mathbf{Data}}$ formulas that can be used in data predicates. In particular, universal binary data predicates mostly make sense for transitive properties such as $<$, $\leq$, or $\not\approx$. We do *not* impose any such restrictions, however. The use of non-transitive universal binary predicates will simply limit the size of the models. For example, $\mathsf{ls}(x, [\mathsf{n}\colon \beta = \alpha + 1]^{\forall})$ is satisfiable only in lists of length at most 2, because it is impossible for all pairs of three or more distinct integers to differ by 1.

5.2.2 *Formal Syntax and Semantics of SSL with Data*

$\mathbf{SSL}_{\mathsf{data}}$ is the extension of **SSL** with the following atomic formulas:

- All $\mathcal{F}_{\mathbf{Data}}$ formulas, i.e., quantifier-free formulas from the data theory.
- Points-to assertions $x \mapsto_{\mathsf{ls}} \langle y_1, z\rangle$ and $x \mapsto_{\mathsf{tree}} \langle y_1, y_2, z\rangle$. In contrast to the assertions $x \mapsto_{\mathsf{n}} y$ and $x \mapsto_{\mathsf{l,r}} \langle y_1, y_2\rangle$ of **SSL**, these assertions enforce that $\mathsf{d}(\mathfrak{s}(x)) = \mathfrak{s}(z)$.
- Predicate calls $\mathsf{pred}(x, \mathbf{y}, \mathcal{P})$, where $\mathsf{pred} \in \{\mathsf{ls}, \mathsf{tree}, \mathsf{ls}_{\geq 2}, \mathsf{tree}_2\}$, $x \in \mathbf{Var}$, $\mathbf{y} \in \mathbf{Var}^*$, and $\mathcal{P}$ is a set of data predicates.

We denote by $\mathbf{SSL}^{+}_{\mathsf{data}}$ the positive fragment of $\mathbf{SSL}_{\mathsf{data}}$.

The formal semantics of these new atoms is defined in Fig. 5.5.

An inductive predicate $\mathsf{pred}(x, \mathbf{y}, \mathcal{P})$ holds only if its shape-only variant $\mathsf{pred}(x, \mathbf{y})$ holds. The semantics of data predicates use the reachability predicates from Chapter 4 in a straightforward way. I would like to emphasize that $\ell_1 \xrightarrow{\mathsf{f}*}_{\mathfrak{h}} \ell_2$ holds if ℓ_2 is reachable from ℓ_1 by *first* following the f-field; for the remainder of the path from ℓ_1 to ℓ_2, we can follow arbitrary fields. For example, $\ell_1 \xrightarrow{\mathsf{l}*}_{\mathfrak{h}} \ell_2$ holds for all nodes ℓ_2 in the left subtree of the tree with root ℓ_1.

$\mathbf{SSL}_{\mathsf{data}}$ inherits many properties from **SSL**. For example, it cannot distinguish between isomorphic models.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

$$
\begin{array}{lll}
(\mathfrak{s},\mathfrak{h}) \models \phi, \phi \in \mathcal{F}_{\mathbf{Data}} & \text{iff} & \mathfrak{s} \models_{\mathbf{Data}} \phi \text{ and } \mathrm{dom}(\mathfrak{h}) = \emptyset \\
(\mathfrak{s},\mathfrak{h}) \models x \mapsto_{\mathsf{ds}} \mathbf{y} & \text{iff} & \mathfrak{h} = \{\mathfrak{s}(x) \mapsto \mathfrak{s}(\mathbf{y})\} \text{ and } \mathfrak{s}(\mathbf{y}) \in \mathsf{sig}(\mathsf{ds}) \\
(\mathfrak{s},\mathfrak{h}) \models \mathsf{pred}(x,\mathbf{y},\mathcal{P}) & \text{iff} & (\mathfrak{s},\mathfrak{h}) \models \mathsf{pred}(x,\mathbf{y}) \text{ and} (\mathfrak{s},\mathfrak{h}) \models \bigwedge \mathcal{P} \\
(\mathfrak{s},\mathfrak{h}) \models [F]^{\exists} & \text{iff} & \exists \ell \in \mathrm{dom}(\mathfrak{h}).\, \mathfrak{s} \cup \{\alpha \mapsto \mathsf{d}(\ell)\} \models_{\mathbf{Data}} F \\
(\mathfrak{s},\mathfrak{h}) \models [F]^{\forall} & \text{iff} & \forall \ell \in \mathrm{dom}(\mathfrak{h}).\, \mathfrak{s} \cup \{\alpha \mapsto \mathsf{d}(\ell)\} \models_{\mathbf{Data}} F \\
(\mathfrak{s},\mathfrak{h}) \models [\mathsf{f}\colon F]^{\exists} & \text{iff} & \exists \ell_1,\ell_2 \in \mathrm{dom}(\mathfrak{h}).\, \ell_1 \xrightarrow{\mathsf{f}*}_{\mathfrak{h}} \ell_2 \text{ and} \\
& & \mathfrak{s} \cup \{\alpha \mapsto \mathsf{d}(\ell_1), \beta \mapsto \mathsf{d}(\ell_2)\} \models_{\mathbf{Data}} F \\
(\mathfrak{s},\mathfrak{h}) \models [\mathsf{f}\colon F]^{\forall} & \text{iff} & \forall \ell_1,\ell_2 \in \mathrm{dom}(\mathfrak{h}).\ \text{if } \ell_1 \xrightarrow{\mathsf{f}*}_{\mathfrak{h}} \ell_2 \\
& & \text{then } \mathfrak{s} \cup \{\alpha \mapsto \mathsf{d}(\ell_1), \beta \mapsto \mathsf{d}(\ell_2)\} \models_{\mathbf{Data}} F
\end{array}
$$

Figure 5.5: Semantics of $\mathbf{SSL}_{\mathsf{data}}$ formulas. For brevity, we denote with pred any of the inductive predicates ls, $\mathsf{ls}_{\geq 2}$, tree, $\mathsf{tree}_{\geq 2}$.

Lemma 5.16. *Let $(\mathfrak{s},\mathfrak{h}),(\mathfrak{s}',\mathfrak{h}')$ be models with $(\mathfrak{s},\mathfrak{h}) \cong (\mathfrak{s}',\mathfrak{h}')$ and let $\phi \in \mathbf{SSL}_{\mathsf{data}}$. Then $(\mathfrak{s},\mathfrak{h}) \models \phi$ iff $(\mathfrak{s}',\mathfrak{h}') \models \phi$.*

Proof. Analogous to Lemma 5.7, noting that because isomorphisms are defined to be identity functions on **Data**, it holds that

$$\mathsf{d}(\mathfrak{h}(\ell)) = \mathsf{d}(\mathfrak{h}'(\sigma(\ell))) \text{ for all } \ell \in \mathrm{dom}(\mathfrak{h}),$$

where σ is an isomorphism. Consequently, neither data atoms nor data predicates can differentiate between $(\mathfrak{s},\mathfrak{h})$ and $(\mathfrak{s}',\mathfrak{h}')$. □

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

6 DECIDING SSL WITHOUT DATA PREDICATES

The goal of this chapter is to develop a decision procedure for the full propositional logic **SSL**, i.e., the logic with Boolean operators and magic wand, but without data predicates. Specifically, we will prove the following theorem.

Theorem 6.1 (Satisfiability checking for **SSL**). *Let $\phi \in \mathbf{SSL}$ and let $\mathbf{x} \subseteq \mathbf{Var}$ be a finite set of variables. It is decidable in* PSpace *(in $|\phi|$ and $|\mathbf{x}|$) whether there exists a model $(\mathfrak{s}, \mathfrak{h})$ with $\mathrm{dom}(\mathfrak{s}) \subseteq \mathbf{x}$ and $(\mathfrak{s}, \mathfrak{h}) \models \phi$.*

Note the parameterization on $\mathbf{x}$ in Theorem 6.1, which reflects that the SSL semantics changes as we increase the stack size.

Example 6.2 (Parameterization on $\mathbf{x}$). *Recall the formulas* $\mathsf{alloc}(x)$ *and* $-\!\circledast$ *from Fig. 5.2. Let $\phi = \mathsf{alloc}(x) -\!\circledast \mathsf{ls}_{\geq 2}(x, \mathsf{nil})$. In standard, weak SL, $(\mathfrak{s}, \mathfrak{h}) \models \phi$ iff*

$$\mathfrak{h} = \{\ell_1 \mapsto \langle \ell_2, d_2 \rangle, \ell_2 \mapsto \langle \ell_3, d_3 \rangle, \ldots, \ell_{n-1} \mapsto \langle \ell_n, d_n \rangle\},$$

$n \geq 1$, with $\mathfrak{s}(\mathsf{nil}) = \ell_n$ and $\mathfrak{s}(x) \notin \mathrm{dom}(\mathfrak{h})$: all such list segments can be extended to a list segment from x to nil *by prepending a list segment from $\mathfrak{s}(x)$ to ℓ_1. This is not the case in SSL, because $\uplus^{\mathfrak{s}}$ only allows concatenating the two list segments if $\ell_1 \in \mathrm{img}(\mathfrak{s})$. Consequently, ϕ is unsatisfiable in stacks with $\mathrm{dom}(\mathfrak{s}) \subseteq \{x\}$, but is satisfiable in stacks with $\mathrm{dom}(\mathfrak{s}) = \{x, y\}$. In the latter case, we can pick $\mathfrak{s}(y) = \ell_1$, which allows us to prepend the list segment as in the WSL case, illustrated in Fig. 6.1.*

Our approach is based on abstracting stack–heap models by *abstract memory states* (AMS). AMSs have the following two key properties.

REFINEMENT (THEOREM 6.28). If $(\mathfrak{s}_1, \mathfrak{h}_1)$ and $(\mathfrak{s}_2, \mathfrak{h}_2)$ have the same AMS, then they satisfy the same **SSL** formulas. As such, the AMS abstraction *refines* the satisfaction relation of SSL.

COMPUTABILITY (FIG. 6.4). For every formula $\phi \in \mathbf{SSL}$ we can compute (in PSpace) the set of all AMS of all models of ϕ. A formula $\phi \in \mathbf{SSL}$ is satisfiable if this set is nonempty.

These two properties together imply Theorem 6.1; see also Section 3.4 for a high-level summary of the approach of *abstraction-based satisfiability checking* that I take in both main parts of this thesis.

The AMS abstraction is motivated by the following insights.

1. The strong union operator, $\uplus^{\mathfrak{s}}$, induces a unique decomposition of the heap into at most $|\mathfrak{s}|$ minimal *chunks* of memory that cannot be further decomposed.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

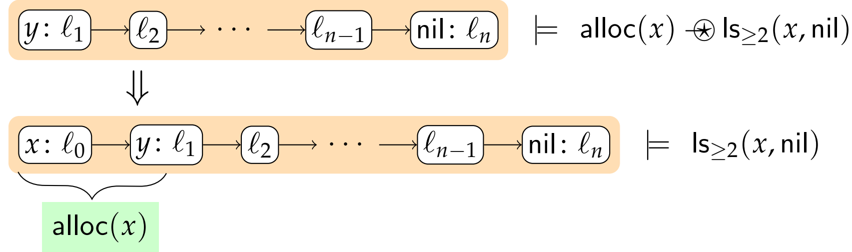


Figure 6.1: A model of $\mathsf{alloc}(x) \circledast \mathsf{ls}_{\geq 2}(x, \mathsf{nil})$ with a stack of size 2. In **SSL**, this is no longer a model if y is removed from the stack.

2. To decide whether $(\mathfrak{s}, \mathfrak{h}) \models \phi$ holds, it is sufficient to know for each chunk of $\mathfrak{h}$ (1) which atomic formulas the chunk satisfies and (2) which variables (if any) are allocated in the chunk.

We proceed as follows. In Section 6.1, we will make precise the notion of memory chunks and define the AMS abstraction. We will prove the *refinement theorem* for SSL in Section 6.2. This makes it possible to map every SSL formula to the AMS of its models, as we will see in Section 6.3. In Section 6.4, we will build upon these results to develop a procedure for deciding satisfiability of SSL formulas by means of AMS computation. Finally, we will show the PSPACE-completeness result in Section 6.5.

While the idea for strong-separation logic and the AMS abstraction were mine, I would like to stress that large parts of this chapter are closely based on joint work with Florian Zuleger [PZ20b].

6.1 ABSTRACT MEMORY STATES

An abstract memory state consists of an abstraction of every *chunk* of memory of a stack–heap model. For our purposes, a chunk of $\mathfrak{h}$ is a *minimal* nonempty sub-heap of $\mathfrak{h}$ that can be split off of $\mathfrak{h}$ according to the strong-separation semantics.

Definition 6.3 (Chunk). *Let* $(\mathfrak{s}, \mathfrak{h})$ *be a model. A sub-heap* $\mathfrak{h}_1 \subseteq \mathfrak{h}$ *is a* chunk of $(\mathfrak{s}, \mathfrak{h})$ *if*

1. *there exists a heap* $\mathfrak{h}_0$ *such that* $\mathfrak{h} = \mathfrak{h}_1 \uplus^{\mathfrak{s}} \mathfrak{h}_0$*, and*
2. *for all* $\mathfrak{h}_2 \subsetneq \mathfrak{h}_1$ *with* $\mathfrak{h}_2 \neq \emptyset$*, there does* not *exist a heap* $\mathfrak{h}_0$ *with* $\mathfrak{h} = \mathfrak{h}_2 \uplus^{\mathfrak{s}} \mathfrak{h}_0$.

We collect the set of all chunks of $(\mathfrak{s}, \mathfrak{h})$ *in* $\mathsf{chunks}(\mathfrak{s}, \mathfrak{h})$.

Example 6.4 (Chunks). *Let* $d \in$ **Data** *be an arbitrary value from the data domain. Consider the following model* $(\mathfrak{s}, \mathfrak{h})$*:*

$$\begin{aligned}\mathfrak{s} =& \{x \mapsto 1, y \mapsto 3, z \mapsto 3, w \mapsto 5, v \mapsto 8\}\\ \mathfrak{h} =& \{1 \mapsto \langle 2, d\rangle, 2 \mapsto \langle 3, d\rangle, 3 \mapsto \langle 7, d\rangle, 4 \mapsto \langle 6, d\rangle, 5 \mapsto \langle 6, d\rangle,\\ & 6 \mapsto \langle 3, d\rangle, 8 \mapsto \langle 8, d\rangle, 9 \mapsto \langle 10, d\rangle, 10 \mapsto \langle 9, d\rangle\}\end{aligned}$$

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

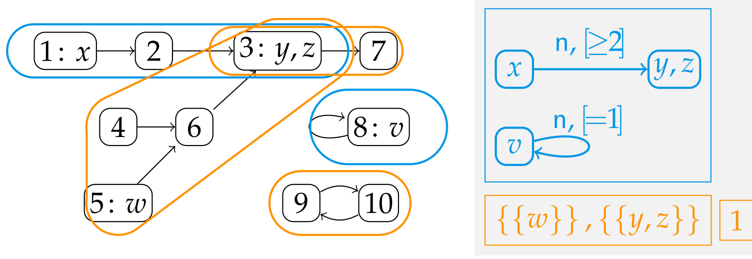


Figure 6.2: Graphical representation of a model consisting of five chunks (left, see Example 6.4; data fields omitted for clarity) and its induced AMS (right, see Example 6.12).

The model $(\mathfrak{s},\mathfrak{h})$ is illustrated in Fig. 6.2. I've deliberately omitted the data fields from the graphical representation to make it easier to parse the figure.

Like in the previous chapter, I include both the identities and the labels of locations in the graphical representation; e.g., $3\colon y,z$ represents location 3 with $\mathfrak{s}(y)=3$ and $\mathfrak{s}(z)=3$.

The model consists of five chunks, $\mathfrak{h}_1 \triangleq \{1 \mapsto \langle 2,d\rangle, 2 \mapsto \langle 3,d\rangle\}$, $\mathfrak{h}_2 \triangleq \{8 \mapsto \langle 8,d\rangle\}$, $\mathfrak{h}_3 \triangleq \{4 \mapsto \langle 6,d\rangle, 5 \mapsto \langle 6,d\rangle, 6 \mapsto \langle 3,d\rangle\}$, $\mathfrak{h}_4 \triangleq \{3 \mapsto \langle 7,d\rangle\}$, and $\mathfrak{h}_5 \triangleq \{9 \mapsto \langle 10,d\rangle, 10 \mapsto \langle 9,d\rangle\}$.

Every model can be decomposed into its chunks:

Lemma 6.5 (Decomposability into chunks)**.** *Let $(\mathfrak{s},\mathfrak{h})$ be a model and let* $\mathsf{chunks}(\mathfrak{s},\mathfrak{h}) = \{\mathfrak{h}_1,\ldots,\mathfrak{h}_n\}$. *Then, $\mathfrak{h} = \mathfrak{h}_1 \uplus^{\mathfrak{s}} \mathfrak{h}_2 \uplus^{\mathfrak{s}} \cdots \uplus^{\mathfrak{s}} \mathfrak{h}_n$.*

Proof. We prove the claim by providing a graph-theoretic representation of the chunks of $(\mathfrak{s},\mathfrak{h})$. We consider the directed graph

$$\mathcal{G} = \{\mathrm{dom}(\mathfrak{h}), \{(\ell,\ell') \mid \ell' \in \mathfrak{h}(\ell) \cap \mathbf{Loc}, \ell' \notin \mathrm{img}(\mathfrak{s})\}.$$

This graph is like the induced graph $\mathsf{graph}(\mathfrak{h})$, except that we remove all the edges that end in a labeled location.

Let $\mathcal{C}_1,\ldots,\mathcal{C}_n$ be the connected components of $\mathcal{G}$. Recall that connected components may consist of a single location and no edges.

We now consider the sub-heaps $\mathfrak{h}_1,\ldots,\mathfrak{h}_n$ of $\mathfrak{h}$ induced by the connected components, where we define $\mathfrak{h}_i$ to be the heap $\mathfrak{h}$ restricted to the locations of $\mathcal{C}_i$.

We now prove that $\mathfrak{h}_1,\ldots,\mathfrak{h}_n$ are indeed the chunks of $\mathfrak{h}$: We consider two heaps $\mathfrak{h}_S$ and $\mathfrak{h}_T$ such that $\mathfrak{h}_S \uplus^{\mathfrak{s}} \mathfrak{h}_T = \mathfrak{h}$. Then we have that each sub-heap $\mathfrak{h}_i$ is fully contained in either $\mathfrak{h}_S$ or $\mathfrak{h}_T$ because only edges that end in a labeled location can be used to connect the two heaps $\mathfrak{h}_S$ and $\mathfrak{h}_T$, but by construction, only the sources and the sinks of $\mathfrak{h}_i$ can be labeled locations.

Further, for all $i \neq j$, the locations in $\mathsf{locs}(\mathfrak{h}_i) \cap \mathsf{locs}(\mathfrak{h}_j)$ are labeled locations, i.e., $\mathsf{locs}(\mathfrak{h}_i) \cap \mathsf{locs}(\mathfrak{h}_j) \subseteq \mathrm{img}(\mathfrak{s})$, because these locations are the targets of the removed edges and thus in $\mathrm{img}(\mathfrak{s})$ by construction. Consequently, $\mathfrak{h}_1 \uplus^{\mathfrak{s}} \mathfrak{h}_2 \uplus^{\mathfrak{s}} \cdots \uplus^{\mathfrak{s}} \mathfrak{h}_n$ is defined. Moreover, every location

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

in $\text{dom}(\mathfrak{h})$ is contained in some connected component C_i and thus in some sub-heap $\mathfrak{h}_i$. Hence, $\mathfrak{h} = \mathfrak{h}_1 \uplus^{\mathfrak{s}} \mathfrak{h}_2 \uplus^{\mathfrak{s}} \cdots \uplus^{\mathfrak{s}} \mathfrak{h}_n$. □

We distinguish two types of chunks: those that satisfy **SSL** atoms and those that don't.

Definition 6.6 (Positive and negative chunk). *Let $\mathfrak{h}_c \subseteq \mathfrak{h}$ be a chunk of $(\mathfrak{s}, \mathfrak{h})$. $\mathfrak{h}_c$ is a* positive chunk *if there exists an atomic SSL formula τ such that $(\mathfrak{s}, \mathfrak{h}_c) \models \tau$. Otherwise, $\mathfrak{h}_c$ is a* negative chunk. *We collect the positive and negative chunks of $(\mathfrak{s}, \mathfrak{h})$ in* $\mathsf{chunks}^+(\mathfrak{s}, \mathfrak{h})$ *and* $\mathsf{chunks}^-(\mathfrak{s}, \mathfrak{h})$, *respectively.*

As the name suggests, every positive chunk satisfies at least one positive formula, whereas negative chunks don't satisfy any positive formula.

Example 6.7. *Recall the chunks $\mathfrak{h}_1$ through $\mathfrak{h}_5$ from Example 6.4. $\mathfrak{h}_1$ and $\mathfrak{h}_2$ are positive chunks (blue in Fig. 6.2), $\mathfrak{h}_3$ to $\mathfrak{h}_5$ are negative chunks (orange).*

Negative chunks can be classified into four (not mutually-exclusive) categories, illustrated in Fig. 6.3.

CYCLIC STRUCTURES. A cyclic list or lasso (cf. Definition 4.15) of length at least two[1] or tree segments of depth at least two whose root is among its holes.

UNLABELED JOIN POINTS. Overlaid lists or trees that cannot be separated via $\uplus^{\mathfrak{s}}$ because they are joined at locations that are not in $\text{img}(\mathfrak{s})$.

GARBAGE. Chunks that contain locations that are inaccessible via stack variables.

UNLABELED DANGLING POINTERS. Chunks that contain an unlabeled sink, i.e., a dangling location that is not in $\text{img}(\mathfrak{s})$ and thus *cannot* be "made non-dangling" via composition using $\uplus^{\mathfrak{s}}$.

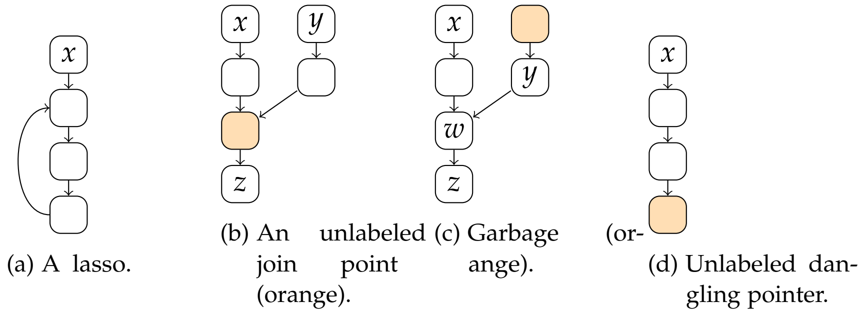
Example 6.8 (Negative chunks). *The chunk $\mathfrak{h}_3$ from Example 6.4 can be viewed as two overlaid list segments (from 4 to 3 and 5 to 3), and it contains garbage, namely the location 4 that cannot be reached via stack variables. The chunk $\mathfrak{h}_4$ has an unlabeled dangling pointer. The chunk $\mathfrak{h}_5$ is cyclic.*

By definition, every positive chunk satisfies at least one atomic formula (a points-to assertion, a list predicate, or a tree predicate), whereas negative chunks do not satisfy any positive SSL formula.

In *abstract memory states* (AMSs), we retain for every chunk enough information to (1) determine which atomic formulas the chunk satisfies, and (2) keep track of which variables are allocated within each chunk.

1 Such lists are negative chunks because our semantics of ls does not allow cyclic list segments. Consequently, heaps that correspond to cyclic lists do not satisfy any atomic SSL formula and are thus negative.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub



(a) A lasso.

(b) An unlabeled join point (orange).

(c) Garbage (orange).

(d) Unlabeled dangling pointer.

Figure 6.3: Types of negative chunks.

An AMS is a graph abstraction that uses an *abstract edge relation* with the following signature.

$$E\colon (V \times \{\mathsf{l},\mathsf{r},\mathsf{n}\}) \rightharpoonup (V^{+} \times \{\llbracket =1 \rrbracket, \llbracket \geq 2 \rrbracket\})$$

A hyperedge

$$\langle v, \mathsf{f}\rangle \mapsto \langle \mathbf{w}, \llbracket \sim n \rrbracket\rangle\,, \quad \text{where } \llbracket \sim n \rrbracket \in \{\llbracket =1 \rrbracket, \llbracket \geq 2 \rrbracket\}\,,$$

is a partial encoding of a positive chunk, stating that if I follow field f from node v, I can reach sinks $\mathbf{w}$; and that this part of the chunk is either of size exactly one, $\llbracket =1 \rrbracket$, or of size at least two, $\llbracket \geq 2 \rrbracket$. For example, $\langle v, \mathsf{l}\rangle \mapsto \langle \mathbf{w}, \llbracket \geq 2 \rrbracket\rangle$ expresses that the left subtree of v is of depth at least two and contains the sinks $\mathbf{w}$.

We define AMSs in terms of the following auxiliary notation.

- $\mathrm{dom}_V(E) \triangleq \{v \mid \text{ex. } \mathsf{f} \text{ s.t. } \langle v, \mathsf{f}\rangle \in \mathrm{dom}(E)\}$.
- $\mathsf{fields}_E(v) \triangleq \{\mathsf{f} \mid \langle v, \mathsf{f}\rangle \in \mathrm{dom}(E)\}$.

We use sets of variables—specifically, stack equivalence classes—as the nodes of an AMS. Recall from Section 2.4 that $\mathsf{classes}(\mathfrak{s})$ denotes the equivalence classes of stack $\mathfrak{s}$.

Definition 6.9. *Let $\mathfrak{s}$ be a stack. Let $V \triangleq \mathsf{classes}(\mathfrak{s})$, $E\colon (V \times \{\mathsf{l},\mathsf{r},\mathsf{n}\}) \rightharpoonup (V^{+} \times \{\llbracket =1 \rrbracket, \llbracket \geq 2 \rrbracket\})$, $\rho \subseteq 2^{V}$, and $\gamma \in \mathbb{N}$. Then $\mathcal{A} = \langle V, E, \rho, \gamma\rangle$ is an* abstract memory state *(AMS) for stack $\mathfrak{s}$ if all of the following* consistency conditions *hold.*

1. *Consistent data structures: For all $v \in V$,*

$$\mathsf{fields}_E(v) \in \{\emptyset, \{\mathsf{n}\}, \{\mathsf{l},\mathsf{r}\}\}\,.$$

2. *No double allocation: $(\bigcup \mathrm{dom}_V(E)) \cap \bigcup \rho = \emptyset$ and for all $\mathbf{v_1}, \mathbf{v_2} \in \rho$, if $\mathbf{v_1} \neq \mathbf{v_2}$ then $\mathbf{v_1} \cap \mathbf{v_2} = \emptyset$.*

3. *No* unrealizable edges*:*
   a) *If $E(v,\mathsf{n}) = \langle\langle v_1, \ldots, v_k\rangle, \iota\rangle$ then $k = 1$.*
   b) *For all f, if $E(v,\mathsf{f}) = \langle\langle v_1, \ldots, v_k\rangle, \iota\rangle$ and $k \geq 2$ then $\iota = \llbracket \geq 2 \rrbracket$.*

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek, Your knowledge hub

We call V the nodes, *E the* hyperedges, *ρ the* negative-allocation constraint *and γ the* garbage-chunk count *of $\mathcal{A}$. The* size *of $\mathcal{A}$ is given by $|\mathcal{A}| \triangleq |V| + \sum_{v \in \mathrm{dom}(E), E(v)=\langle \mathbf{y}, \iota \rangle} |\mathbf{y}| + \gamma$. Finally, the* allocated variables *of an AMS are given by* $\mathbf{alloc}(\mathcal{A}) \triangleq \mathrm{dom}_V(E) \cup \bigcup \rho$.

We collect the set of all AMS in **AMS** and the set of AMS with nodes $\mathsf{classes}(\mathfrak{s})$ in $\mathbf{AMS}_{\mathfrak{s}}$.

The consistency conditions enforce that every AMS is the abstraction of at least one stack–heap pair, as we will see below. The rationale behind each of the conditions is as follows.

1. An edge label f corresponds to allocated fields of the location labeled by v. We must thus ensure that we don't allocate both the list field n and a tree field l, r on the same variable; and we must ensure that we either allocate both l and r or neither of them.

2. Both $\mathrm{dom}_V(E)$ and $\bigcup \rho$ correspond to (equivalence classes of) allocated variables, so we demand that these sets are disjoint to preclude double allocation.

3. A list can only ever have one hole; and only structures of size at least two can have more than one hole per field.

Every model induces an AMS based on its decomposition into chunks. First, every positive chunk induces hyperedges. Informally, each hyperedge maps the source of a positive chunk and a field f to the sinks of the chunk reachable via f (including nil, if applicable) and to a tag that indicates whether or not the size of the chunk is exactly one or at least two.

Definition 6.10 (Induced hyperedge). *Let $\mathfrak{h}_c \in \mathsf{chunks}^+(\mathfrak{s}, \mathfrak{h})$ for a model $(\mathfrak{s}, \mathfrak{h})$, let ℓ be the unique source of $\mathfrak{h}_c$, and let $v \triangleq \mathfrak{s}^{-1}(\ell)$. Define*

$$\iota \triangleq \begin{cases} [=1] & \text{if } |\mathfrak{h}_c| = 1 \\ [\geq 2] & \text{otherwise.} \end{cases}$$

The induced hyperedges of $\mathfrak{h}_c$,

$$\mathsf{edges}_{\mathfrak{s}}(\mathfrak{h}_c) \colon V \times \{\mathsf{l}, \mathsf{r}, \mathsf{n}\} \rightharpoonup V^+ \times \{[=1], [\geq 2]\},$$

are then defined as follows.

CASE $\ell \in \mathsf{loc}_{\mathsf{ls}}(\mathfrak{h})$. *If $\mathsf{sinkvars}_{\mathfrak{s}}(\mathfrak{h}_c) = \varepsilon$, we let $\mathbf{w} \triangleq \langle [\mathsf{nil}]^{\mathfrak{s}}_{=} \rangle$. Otherwise, we let $\mathbf{w} \triangleq \mathsf{sinkvars}_{\mathfrak{s}}(\mathfrak{h}_c)$. Then, $\mathsf{edges}_{\mathfrak{s}}(\mathfrak{h}_c) \triangleq \{\langle v, \mathsf{n} \rangle \mapsto \langle \mathbf{w}, \iota \rangle\}$.*

CASE $\ell \in \mathsf{loc}_{\mathsf{tree}}(\mathfrak{h})$. *Let $\mathbf{w}_{\mathsf{l}}, \mathbf{w}_{\mathsf{r}}$ be such that $\mathbf{w}_{\mathsf{l}} \cdot \mathbf{w}_{\mathsf{r}} = \mathsf{sinkvars}_{\mathfrak{s}}(\mathfrak{h}_c)$ and such that $\mathbf{w}_{\mathsf{l}}$ are those sink variables in the left subtree of ℓ, i.e., $\mathbf{w}_{\mathsf{l}} = \left\{ \mathfrak{s}^{-1}(\ell') \mid \ell' \in \mathsf{sinkseq}(\mathfrak{s}, \mathfrak{h}_c) \text{ s.t. } \ell \xrightarrow{\mathsf{l}*}_{\mathfrak{h}_c} \ell' \right\}$. Moreover,*

$$\mathbf{w}'_{\mathsf{l}} \triangleq \langle [\mathsf{nil}]^{\mathfrak{s}}_{=} \rangle \text{ if } \mathbf{w}_{\mathsf{l}} = \varepsilon \text{ and } \mathbf{w}'_{\mathsf{l}} \triangleq \mathbf{w}_{\mathsf{l}} \text{ otherwise,}$$
$$\mathbf{w}'_{\mathsf{r}} \triangleq \langle [\mathsf{nil}]^{\mathfrak{s}}_{=} \rangle \text{ if } \mathbf{w}_{\mathsf{r}} = \varepsilon \text{ and } \mathbf{w}'_{\mathsf{r}} \triangleq \mathbf{w}_{\mathsf{r}} \text{ otherwise.}$$

TU Bibliothek Your knowledge hub
Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.

Then $\mathsf{edges}_{\mathfrak{s}}(\mathfrak{h}_c) = \{\langle v, \mathsf{l}\rangle \mapsto \langle \mathbf{w}'_{\mathsf{l}}, \iota\rangle, \langle v, \mathsf{r}\rangle \mapsto \langle \mathbf{w}'_{\mathsf{r}}, \iota\rangle\}$.

In the above definition, we explicitly include (the equivalence class of) nil among the targets of the hyperedges that would otherwise have no target. This reflects that every chunk that does not contain explicitly specified sinks must be null-terminated.

Inclusion of nil in induced hyperedges

You may have noticed that we only include nil among the targets if no target is explicitly specified, i.e., for null-terminated lists and for sub-trees without holes. It would be possible to also include nil for sub-trees in which both nil and one or more holes occur as dangling pointers. This is not necessary, however, because **SSL** formulas cannot distinguish between trees of depth at least two that contain holes and nil and trees that contain only holes.

We denote the sets of variables allocated in negative chunks by

$$\mathsf{alloc}^-(\mathfrak{s}, \mathfrak{h}) \triangleq \{\mathfrak{s}^{-1}(\mathrm{dom}(\mathfrak{h}_c) \cap \mathrm{img}(\mathfrak{s})) \mid \mathfrak{h}_c \in \mathsf{chunks}^-(\mathfrak{s}, \mathfrak{h})\} \setminus \emptyset.$$

Now we are ready to define the *induced AMS* of a model.

Definition 6.11 (Induced AMS). *Let* $(\mathfrak{s}, \mathfrak{h})$ *be a model. Let*

$$V \triangleq \mathsf{classes}(\mathfrak{s}),$$
$$E \triangleq \bigcup \{\mathsf{edges}_{\mathfrak{s}}(\mathfrak{h}_c) \mid \mathfrak{h}_c \in \mathsf{chunks}^+(\mathfrak{s}, \mathfrak{h})\},$$
$$\rho \triangleq \mathsf{alloc}^-(\mathfrak{s}, \mathfrak{h}),$$
$$\gamma \triangleq |\mathsf{chunks}^-(\mathfrak{s}, \mathfrak{h})| - |\mathsf{alloc}^-(\mathfrak{s}, \mathfrak{h})|.$$

Then $\mathsf{ams}(\mathfrak{s}, \mathfrak{h}) \triangleq \langle V, E, \rho, \gamma\rangle$ *is the* induced AMS *of* $(\mathfrak{s}, \mathfrak{h})$.

Example 6.12. *The induced AMS of the model* $(\mathfrak{s}, \mathfrak{h})$ *from Example 6.4 is illustrated on the right-hand side of Fig. 6.2. The blue box depicts the graph* (V, E) *induced by the positive chunks* $\mathfrak{h}_1, \mathfrak{h}_2$. *The negative chunk* $\mathfrak{h}_3$ *allocates the variables* $\{[w]^{\mathfrak{s}}_{=}\} = \{\{w\}\}$, *negative chunk* $\mathfrak{h}_4$ *allocates the variables* $\{[y]^{\mathfrak{s}}_{=}\} = \{\{y, z\}\}$. *Consequently, the negative-allocation constraint is* $\{\{\{w\}\}, \{\{y, z\}\}\}$. *Finally, and the garbage-chunk count is* 1, *because* $\mathfrak{h}_5$ *is the only negative chunk that does not allocate stack variables.*

Observe that the induced AMS is indeed an AMS.

Lemma 6.13. *Let* $(\mathfrak{s}, \mathfrak{h})$ *be a model. Then* $\mathsf{ams}(\mathfrak{s}, \mathfrak{h}) \in$ **AMS**.

Proof. Let $\langle V, E, \rho, \gamma\rangle \triangleq \mathsf{ams}(\mathfrak{s}, \mathfrak{h})$. We show that all conditions hold.

1. Every set of induced hyperedges defines either the field n or both fields l, r, guaranteeing data-structure consistency.
2. Every variable is allocated in at most one chunk, so no variable can occur both in $\mathrm{dom}(E)$ and in ρ.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

3. Every hyperedge is realizable:
   a) List segments have exactly one hole.
   b) Trees can only have more than one sink in a subtree if they are of size at least two. □

Conversely, every AMS is the induced AMS of at least one model.

Lemma 6.14 (Realizability of AMS)**.** *Let* $\mathcal{A} \in \mathbf{AMS}$*. There exists a model* $(\mathfrak{s}, \mathfrak{h})$ *with* $\mathsf{ams}(\mathfrak{s}, \mathfrak{h}) = \mathcal{A}$*.*

Proof. The claim follows easily from the consistency criteria of AMS. I defer a detailed proof, including a polynomial bound on the size of the minimal model $(\mathfrak{s}, \mathfrak{h})$ with $\mathsf{ams}(\mathfrak{s}, \mathfrak{h}) = \mathcal{A}$, until Section 7.1, where I return to the problem of realizability in the larger context of proving a *small-model property* for **SSL**. □

We abstract SSL formulas by the set of AMS of their models.

Definition 6.15 (Abstraction of **SSL** formulas)**.** *Let* $\mathfrak{s}$ *be a stack. The* **SSL** abstraction *w.r.t.* $\mathfrak{s}$, $\mathsf{ams}_{\mathfrak{s}} \colon \mathbf{SSL} \to 2^{\mathbf{AMS}}$*, is given by*

$$\mathsf{ams}_{\mathfrak{s}}(\phi) \triangleq \{\mathsf{ams}(\mathfrak{s}, \mathfrak{h}) \mid \mathfrak{h} \in \mathbf{Heaps} \text{ and } (\mathfrak{s}, \mathfrak{h}) \models \phi\}.$$

We lift **SSL** abstraction from individual stacks to all stacks over a set of variables: $\mathsf{ams}_{\mathbf{x}}(\phi) \triangleq \{\mathsf{ams}_{\mathfrak{s}}(\phi) \mid \mathrm{dom}(\mathfrak{s}) = \mathbf{x}\}$. Because AMS do not retain any information about heap locations, just about aliasing, abstractions do not differ for stacks with the same equivalence classes.

Lemma 6.16. *Let* $\mathfrak{s}, \mathfrak{s}'$ *be stacks with* $\mathsf{classes}(\mathfrak{s}) = \mathsf{classes}(\mathfrak{s}')$*. It then holds for all formulas* $\phi \in \mathbf{SSL}$ *that* $\mathsf{ams}_{\mathfrak{s}}(\phi) = \mathsf{ams}_{\mathfrak{s}'}(\phi)$*.*

Proof. Let $\mathcal{A} \in \mathsf{ams}_{\mathfrak{s}}(\phi)$. There exists a heap $\mathfrak{h}$ such that $\mathsf{ams}(\mathfrak{s}, \mathfrak{h}) = \mathcal{A}$ and $(\mathfrak{s}, \mathfrak{h}) \models \phi$. Let $\mathfrak{h}'$ be such that $(\mathfrak{s}, \mathfrak{h}) \cong (\mathfrak{s}', \mathfrak{h}')$. By Lemma 5.7, $(\mathfrak{s}', \mathfrak{h}') \models \phi$. Moreover, $\mathsf{ams}(\mathfrak{s}', \mathfrak{h}') = \mathcal{A}$. Consequently, $\mathcal{A} \in \mathsf{ams}_{\mathfrak{s}'}(\phi)$. The other direction is proved analogously. □

We thus have $|\mathsf{ams}_{\mathbf{x}}(\phi)| = B_{|\mathbf{x}|}$ if B_k denotes the k-th Bell number, i.e., the number of partitions of a set with k elements.

Garbage-free AMS

It makes sense to view an AMS $\mathcal{A} = \langle V, E, \rho, \gamma \rangle$ with $\rho = \emptyset$ and $\gamma = 0$ as a *garbage-free AMS*. By definition of induced AMS, we have that if $(\mathfrak{s}, \mathfrak{h}) \models \phi$ for a positive formula $\phi \in \mathbf{SSL}^{+}$, then $\mathsf{ams}(\mathfrak{s}, \mathfrak{h})$ is garbage free in this sense. The purpose of the ρ and γ component thus is to deal correctly with formulas that contain negation and/or the magic wand and may thus be satisfied by models that contain garbage.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

6.2 THE REFINEMENT THEOREM FOR SSL

The main goal of this section is to show the following *refinement theorem* for **SSL**.

Theorem (Refinement theorem for **SSL**). *Let $\phi \in$ **SSL** and let $(\mathfrak{s}, \mathfrak{h}_1)$, $(\mathfrak{s}, \mathfrak{h}_2)$ be models with $\mathsf{ams}(\mathfrak{s}, \mathfrak{h}_1) = \mathsf{ams}(\mathfrak{s}, \mathfrak{h}_2)$. Then $(\mathfrak{s}, \mathfrak{h}_1) \models \phi$ iff $(\mathfrak{s}, \mathfrak{h}_2) \models \phi$.*

I will go about this step by step, characterizing the SSL abstraction, $\mathsf{ams}_{\mathfrak{s}}(\phi)$, of all atomic formulas and of the separating conjunction before proving the refinement theorem. In the remainder of this section, let $(\mathfrak{s}, \mathfrak{h})$ be an arbitrary but fixed model.

6.2.1 *Abstract Memory States of Atomic Formulas*

The empty-heap predicate **emp** is only satisfied by the empty heap, i.e., by a heap that consists of zero chunks. Consequently,

Lemma 6.17. $(\mathfrak{s}, \mathfrak{h}) \models$ **emp** *iff* $\mathsf{ams}(\mathfrak{s}, \mathfrak{h}) = \langle \mathsf{classes}(\mathfrak{s}), \emptyset, \emptyset, 0 \rangle$

Proof. $(\mathfrak{s}, \mathfrak{h}) \models$ **emp** iff $\mathfrak{h} = \emptyset$ iff $\mathsf{chunks}(\mathfrak{s}, \mathfrak{h}) = \emptyset$ iff $\mathsf{ams}(\mathfrak{s}, \mathfrak{h}) = \langle \mathsf{classes}(\mathfrak{s}), \emptyset, \emptyset, 0 \rangle$. □

Lemma 6.18. *1.* $(\mathfrak{s}, \mathfrak{h}) \models x \approx y$ *iff* $\mathsf{ams}(\mathfrak{s}, \mathfrak{h}) = \langle \mathsf{classes}(\mathfrak{s}), \emptyset, \emptyset, 0 \rangle$ *and* $[x]^{\mathfrak{s}}_{=} = [y]^{\mathfrak{s}}_{=}$.

2. $(\mathfrak{s}, \mathfrak{h}) \models x \not\approx y$ *iff* $\mathsf{ams}(\mathfrak{s}, \mathfrak{h}) = \langle \mathsf{classes}(\mathfrak{s}), \emptyset, \emptyset, 0 \rangle$ *and* $[x]^{\mathfrak{s}}_{=} \neq [y]^{\mathfrak{s}}_{=}$.

Proof. I only show the first claim, as the proof of the second claim is completely analogous. $(\mathfrak{s}, \mathfrak{h}) \models x \approx y$ iff ($\mathfrak{s}(x) = \mathfrak{s}(y)$ and $\mathfrak{h} = \emptyset$) iff ($[x]^{\mathfrak{s}}_{=} = [y]^{\mathfrak{s}}_{=}$ and $(\mathfrak{s}, \mathfrak{h}) \models$ **emp**) iff, by Lemma 6.17, ($[x]^{\mathfrak{s}}_{=} = [y]^{\mathfrak{s}}_{=}$ and $\mathsf{ams}(\mathfrak{s}, \mathfrak{h}) = \langle \mathsf{classes}(\mathfrak{s}), \emptyset, \emptyset, 0 \rangle$). □

Models of points-to assertions consist of a single positive chunk of size 1.

Lemma 6.19. *1. Let* $E \triangleq \{\langle [x]^{\mathfrak{s}}_{=}, \mathsf{n} \rangle \mapsto \langle \langle [y]^{\mathfrak{s}}_{=} \rangle, [=1] \rangle\}$. $(\mathfrak{s}, \mathfrak{h}) \models x \mapsto_{\mathsf{n}} y$ *iff* $\mathsf{ams}(\mathfrak{s}, \mathfrak{h}) = \langle \mathsf{classes}(\mathfrak{s}), E, \emptyset, 0 \rangle$.

2. *Let* $E \triangleq \{\langle [x]^{\mathfrak{s}}_{=}, \mathsf{l} \rangle \mapsto \langle \langle [y]^{\mathfrak{s}}_{=} \rangle, [=1] \rangle, \langle [x]^{\mathfrak{s}}_{=}, \mathsf{r} \rangle \mapsto \langle \langle [z]^{\mathfrak{s}}_{=} \rangle, [=1] \rangle\}$. $(\mathfrak{s}, \mathfrak{h}) \models x \mapsto_{\mathsf{l},\mathsf{r}} \langle y, z \rangle$ *iff* $\mathsf{ams}(\mathfrak{s}, \mathfrak{h}) = \langle \mathsf{classes}(\mathfrak{s}), E, \emptyset, 0 \rangle$.

Proof. We only show the claim for $\mapsto_{\mathsf{n}}$, as the argument for $\mapsto_{\mathsf{l},\mathsf{r}}$ is analogous. If $(\mathfrak{s}, \mathfrak{h}) \models x \mapsto_{\mathsf{n}} y$ then $\mathfrak{h} = \{\mathfrak{s}(x) \mapsto \langle \mathfrak{s}(y), d \rangle\}$ for some $d \in$ **Data**. In particular, it then holds that $\mathfrak{h}$ is a positive chunk with $|\mathfrak{h}| = 1$, source variables $[x]^{\mathfrak{s}}_{=}$, and sink variables $\langle [y]^{\mathfrak{s}}_{=} \rangle$. Consequently, $\mathsf{edges}_{\mathfrak{s}}(\mathfrak{h}) = E$. It follows that $\mathsf{ams}(\mathfrak{s}, \mathfrak{h}) = \langle \mathsf{classes}(\mathfrak{s}), E, \emptyset, 0 \rangle$.

Conversely, assume $\mathsf{ams}(\mathfrak{s}, \mathfrak{h}) = \langle \mathsf{classes}(\mathfrak{s}), E, \emptyset, 0 \rangle$. Because $|E| = 1$, $\rho = \emptyset$ and $\gamma = 0$, $\mathfrak{h}$ consists of a single positive chunk and no negative chunks; and it holds for that $|\mathfrak{h}| = 1$, the source variables of $\mathfrak{h}$ are

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

$[x]^{\mathfrak{s}}_{=}$, and the sink variables of $\mathfrak{h}$ are $\langle [y]^{\mathfrak{s}}_{=} \rangle$. In other words, $\mathfrak{h}$ consists of a single list location and it holds that $\mathfrak{h} = \{\mathfrak{s}(x) \mapsto \langle \mathfrak{s}(y), d \rangle\}$ for some $d \in \mathbf{Data}$. By the semantics of points-to assertions, we then have $(\mathfrak{s}, \mathfrak{h}) \models x \mapsto_{\mathsf{n}} y$. □

We capture the abstractions of list and tree predicates in the following sets of *abstract lists* and *abstract trees*.

Definition 6.20 (Abstract List). *Let $x \in \mathbf{Var}$, $\mathbf{y} \in \mathbf{Var}^*$ with $|\mathbf{y}| < 1$. Moreover, let y be* nil *if $\mathbf{y}$ is empty and the unique element in $\mathbf{y}$ otherwise.*

An AMS $\mathcal{A}$ is an abstract list *w.r.t. $\mathfrak{s}$, x, and $\mathbf{y}$ if there exist $x_1, \dots, x_k$ and a heap $\mathfrak{h}$ such that such that $(\mathfrak{s}, \mathfrak{h})$ is a connected list segment from x to y via $\langle x_1, \dots, x_k \rangle$ and $\mathsf{ams}(\mathfrak{s}, \mathfrak{h}) = \mathcal{A}$.*

We collect all abstract lists in $\mathbf{AbstLists}(\mathfrak{s}, x, \mathbf{y})$. All abstract lists induced by heaps $\mathfrak{h}$ with $|\mathfrak{h}| \geq 2$ are collected in $\mathbf{AbstLists}_{\geq 2}(\mathfrak{s}, x, \mathbf{y})$.

Definition 6.21 (Abstract Tree). *Let $x \in \mathbf{Var}$, $\mathbf{y} \in \mathbf{Var}^*$. An AMS $\mathcal{A}$ is an* abstract tree *w.r.t. $\mathfrak{s}$, x, and $\mathbf{y}$ if there exists a heap $\mathfrak{h}$ such that $(\mathfrak{s}, \mathfrak{h})$ is a directed tree with root x and holes $\mathbf{y}$ and $\mathsf{ams}(\mathfrak{s}, \mathfrak{h}) = \mathcal{A}$.*

We collect all abstract trees in $\mathbf{AbstTrees}(\mathfrak{s}, x, \mathbf{y})$. All abstract trees induced by heaps $\mathfrak{h}$ with $|\mathfrak{h}| \geq 2$ are collected in $\mathbf{AbstTrees}_{\geq 2}(\mathfrak{s}, x, \mathbf{y})$.

Since we already know from Lemmas 5.4 and 5.5 that connected list segments and directed trees characterize the models of the list and tree predicates, it is no surprise that abstract lists and trees characterize the abstractions of these models.

Lemma 6.22 (Abstraction of predicate calls).

1. $(\mathfrak{s}, \mathfrak{h}) \models \mathsf{ls}(x, \mathbf{y})$ *iff* $\mathsf{ams}(\mathfrak{s}, \mathfrak{h}) \in \mathbf{AbstLists}(\mathfrak{s}, x, \mathbf{y})$.
2. $(\mathfrak{s}, \mathfrak{h}) \models \mathsf{ls}_{\geq 2}(x, \mathbf{y})$ *iff* $\mathsf{ams}(\mathfrak{s}, \mathfrak{h}) \in \mathbf{AbstLists}_{\geq 2}(\mathfrak{s}, x, \mathbf{y})$.
3. $(\mathfrak{s}, \mathfrak{h}) \models \mathsf{tree}(x, \mathbf{y})$ *iff* $\mathsf{ams}(\mathfrak{s}, \mathfrak{h}) \in \mathbf{AbstTrees}(\mathfrak{s}, x, \mathbf{y})$.
4. $(\mathfrak{s}, \mathfrak{h}) \models \mathsf{tree}_{\geq 2}(x, \mathbf{y})$ *iff* $\mathsf{ams}(\mathfrak{s}, \mathfrak{h}) \in \mathbf{AbstTrees}_{\geq 2}(\mathfrak{s}, x, \mathbf{y})$.

Proof. Immediate consequence of Lemmas 5.4 and 5.5 and the definition of abstract lists and abstract trees. □

So far, defining abstract lists and abstract trees may seem like a pointless exercise. This is not quite true. Yes, we have proved the trivial fact that the abstraction of a model of a list is an abstract list—but we have also proved that an abstract list *only* abstracts list models. In other words, all models whose abstraction is an abstract list do satisfy a list predicate. This is crucial for showing the refinement theorem.

Besides, having dedicated notation such as $\mathbf{AbstTrees}(\mathfrak{s}, x, \mathbf{y})$ will be useful later when reasoning about the computability and complexity of AMS-based satisfiability checking.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

6.2.2 *Composing Abstract Memory States*

Our next goal is to lift the union operator $\uplus^{\mathfrak{s}}$ to the abstract domain **AMS**. We will define an operator $\bullet$ with the following property.

$$\text{if } \mathfrak{h}_1 \uplus^{\mathfrak{s}} \mathfrak{h}_2 \neq \bot \text{ then } \mathsf{ams}(\mathfrak{s}, \mathfrak{h}_1 \uplus^{\mathfrak{s}} \mathfrak{h}_2) = \mathsf{ams}(\mathfrak{s}, \mathfrak{h}_1) \bullet \mathsf{ams}(\mathfrak{s}, \mathfrak{h}_2).$$

This will allow us to extend the results that we showed for the abstractions of atomic formulas Section 6.2.1 to arbitrary SSL formulas—and thus prove the refinement theorem.

AMS composition is a partial operation defined only on *compatible* AMS. Compatibility enforces (1) that there is no double allocation and (2) that the AMSs were obtained for equivalent stacks (i.e., for stacks $\mathfrak{s}, \mathfrak{s}'$ with $\mathsf{classes}(\mathfrak{s}) = \mathsf{classes}(\mathfrak{s}')$).

Definition 6.23 (Compatibility of AMSs). *AMSs $\mathcal{A}_1 = \langle V_1, E_1, \rho_1, \gamma_1 \rangle$ and $\mathcal{A}_2 = \langle V_2, E_2, \rho_2, \gamma_2 \rangle$ are* compatible *iff (1) $V_1 = V_2$ and (2) $\mathbf{alloc}(\mathcal{A}_1) \cap \mathbf{alloc}(\mathcal{A}_2) = \emptyset$.*

Note that if $(\mathfrak{s}, \mathfrak{h}_1) \uplus^{\mathfrak{s}} (\mathfrak{s}, \mathfrak{h}_2)$ is defined, then $\mathsf{ams}(\mathfrak{s}, \mathfrak{h}_1)$ and $\mathsf{ams}(\mathfrak{s}, \mathfrak{h}_2)$ are compatible. The converse is not true, because $\mathsf{ams}(\mathfrak{s}, \mathfrak{h}_1)$ and $\mathsf{ams}(\mathfrak{s}, \mathfrak{h}_2)$ may be compatible even if $\mathrm{dom}(\mathfrak{h}_1) \cap \mathrm{dom}(\mathfrak{h}_2) \neq \emptyset$.

AMS composition is defined in a point-wise manner on compatible AMSs and undefined otherwise.

Definition 6.24 (AMS composition). *Let $\mathcal{A}_1 = \langle V_1, E_1, \rho_1, \gamma_1 \rangle$, $\mathcal{A}_2 = \langle V_2, E_2, \rho_2, \gamma_2 \rangle$ be AMS. The* composition *of $\mathcal{A}_1, \mathcal{A}_2$ is then given by*

$$\mathcal{A}_1 \bullet \mathcal{A}_2 \triangleq \begin{cases} \langle V_1, E_1 \cup E_2, \rho_1 \cup \rho_2, \gamma_1 + \gamma_2 \rangle, & \textit{if } \mathcal{A}_1, \mathcal{A}_2 \textit{ compatible} \\ \bot, & \textit{otherwise.} \end{cases}$$

Lemma 6.25. *Let $\mathfrak{s}$ be a stack and let $\mathfrak{h}_1, \mathfrak{h}_2$ be heaps. If $\mathfrak{h}_1 \uplus^{\mathfrak{s}} \mathfrak{h}_2 \neq \bot$ then $\mathsf{ams}(\mathfrak{s}, \mathfrak{h}_1) \bullet \mathsf{ams}(\mathfrak{s}, \mathfrak{h}_2) \neq \bot$.*

Proof. Since the same stack $\mathfrak{s}$ underlies both abstractions, we have $V_1 = V_2$. Furthermore, $\mathrm{dom}(\mathfrak{h}_1) \cap \mathrm{dom}(\mathfrak{h}_2) = \emptyset$ implies that $\mathbf{alloc}(\mathcal{A}_1) \cap \mathbf{alloc}(\mathcal{A}_2) = \emptyset$. □

We next show that $\mathsf{ams}(\mathfrak{s}, \mathfrak{h}_1 \uplus^{\mathfrak{s}} \mathfrak{h}_2) = \mathsf{ams}(\mathfrak{s}, \mathfrak{h}_1) \bullet \mathsf{ams}(\mathfrak{s}, \mathfrak{h}_2)$ whenever $\mathfrak{h}_1 \uplus^{\mathfrak{s}} \mathfrak{h}_2$ is defined.

Lemma 6.26 (Homomorphism of composition). *Let $(\mathfrak{s}, \mathfrak{h}_1), (\mathfrak{s}, \mathfrak{h}_2)$ be models and assume $\mathfrak{h}_1 \uplus^{\mathfrak{s}} \mathfrak{h}_2 \neq \bot$. Then $\mathsf{ams}(\mathfrak{s}, \mathfrak{h}_1 \uplus^{\mathfrak{s}} \mathfrak{h}_2) = \mathsf{ams}(\mathfrak{s}, \mathfrak{h}_1) \bullet \mathsf{ams}(\mathfrak{s}, \mathfrak{h}_2)$.*

Proof. The result follows easily from the observation that

$$\mathsf{chunks}(\mathfrak{s}, \mathfrak{h}_1 \uplus^{\mathfrak{s}} \mathfrak{h}_2) = \mathsf{chunks}(\mathfrak{s}, \mathfrak{h}_1) \cup \mathsf{chunks}(\mathfrak{s}, \mathfrak{h}_2),$$

which, in turn, is an immediate consequence of Lemma 6.5.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

To show this in more detail, we need to look at the components of

$$\begin{aligned}\langle V_1, E_1, \rho_1, \gamma_1\rangle &\triangleq \mathsf{ams}(\mathfrak{s}, \mathfrak{h}_1),\\ \langle V_2, E_2, \rho_2, \gamma_2\rangle &\triangleq \mathsf{ams}(\mathfrak{s}, \mathfrak{h}_2), \text{ and}\\ \langle V, E, \rho, \gamma\rangle &\triangleq \mathsf{ams}(\mathfrak{s}, \mathfrak{h}_1 \uplus^{\mathfrak{s}} \mathfrak{h}_2).\end{aligned}$$

- As all models have the same stack, it holds that $V = V_1$.
- As $\mathsf{chunks}^+(\mathfrak{s}, \mathfrak{h}_1 \uplus^{\mathfrak{s}} \mathfrak{h}_2) = \mathsf{chunks}^+(\mathfrak{s}, \mathfrak{h}_1) \cup \mathsf{chunks}^+(\mathfrak{s}, \mathfrak{h}_2)$, it follows that

  $$\begin{aligned}&\left\{\mathsf{edges}_{\mathfrak{s}}(\mathfrak{h}_c) \mid \mathfrak{h}_c \in \mathsf{chunks}^+(\mathfrak{s}, \mathfrak{h}_1 \uplus^{\mathfrak{s}} \mathfrak{h}_2)\right\}\\ =&\left\{\mathsf{edges}_{\mathfrak{s}}(\mathfrak{h}_c) \mid \mathfrak{h}_c \in \mathsf{chunks}^+(\mathfrak{s}, \mathfrak{h}_1)\right\}\\ &\cup \left\{\mathsf{edges}_{\mathfrak{s}}(\mathfrak{h}_c) \mid \mathfrak{h}_c \in \mathsf{chunks}^+(\mathfrak{s}, \mathfrak{h}_2)\right\}\end{aligned}$$

  and thus $E = E_1 \cup E_2$.
- As $\mathsf{chunks}^-(\mathfrak{s}, \mathfrak{h}_1 \uplus^{\mathfrak{s}} \mathfrak{h}_2) = \mathsf{chunks}^-(\mathfrak{s}, \mathfrak{h}_1) \cup \mathsf{chunks}^-(\mathfrak{s}, \mathfrak{h}_2)$, it follows that

  $$\begin{aligned}\rho =&\mathsf{alloc}^-(\mathfrak{s}, \mathfrak{h}_1 \uplus^{\mathfrak{s}} \mathfrak{h}_2)\\ =&\mathsf{alloc}^-(\mathfrak{s}, \mathfrak{h}_1) \cup \mathsf{alloc}^-(\mathfrak{s}, \mathfrak{h}_2) = \rho_1 \cup \rho_2\end{aligned}$$

  and

  $$\begin{aligned}\gamma =& \left|\mathsf{chunks}^-(\mathfrak{s}, \mathfrak{h}_1 \uplus^{\mathfrak{s}} \mathfrak{h}_2)\right| - \left|\mathsf{alloc}^-(\mathfrak{s}, \mathfrak{h}_1 \uplus^{\mathfrak{s}} \mathfrak{h}_2)\right|\\ =&\left(\left|\mathsf{chunks}^-(\mathfrak{s}, \mathfrak{h}_1)\right| - \left|\mathsf{alloc}^-(\mathfrak{s}, \mathfrak{h}_1)\right|\right)\\ &+ \left(\left|\mathsf{chunks}^-(\mathfrak{s}, \mathfrak{h}_2)\right| - \left|\mathsf{alloc}^-(\mathfrak{s}, \mathfrak{h}_2)\right|\right) = \gamma_1 + \gamma_2.\end{aligned}$$

Combining these observations, we obtain

$$\mathsf{ams}(\mathfrak{s}, \mathfrak{h}_1 \uplus^{\mathfrak{s}} \mathfrak{h}_2) = \mathsf{ams}(\mathfrak{s}, \mathfrak{h}_1) \bullet \mathsf{ams}(\mathfrak{s}, \mathfrak{h}_2). \qquad \square$$

6.2.3 *Proving the Refinement Theorem*

The results from Sections 6.2.1 and 6.2.2 imply the *refinement theorem* that I stated at the beginning of this chapter: models with the same AMS abstraction satisfy the same SSL formulas.

To show the refinement theorem, we need one additional property of AMS composition. If an AMS $\mathcal{A}$ of a model $(\mathfrak{s}, \mathfrak{h})$ can be decomposed into two smaller AMS $\mathcal{A} = \mathcal{A}_1 \bullet \mathcal{A}_2$, it is also possible to decompose the heap $\mathfrak{h}$ into smaller heaps $\mathfrak{h}_1, \mathfrak{h}_2$ with $\mathsf{ams}(\mathfrak{s}, \mathfrak{h}_i) = \mathcal{A}_i$.

Lemma 6.27 (Decomposability of AMS). *Let* $\mathsf{ams}(\mathfrak{s}, \mathfrak{h}) = \mathcal{A}_1 \bullet \mathcal{A}_2$. *There exist* $\mathfrak{h}_1, \mathfrak{h}_2$ *with* $\mathfrak{h} = \mathfrak{h}_1 \uplus^{\mathfrak{s}} \mathfrak{h}_2$, $\mathsf{ams}(\mathfrak{s}, \mathfrak{h}_1) = \mathcal{A}_1$ *and* $\mathsf{ams}(\mathfrak{s}, \mathfrak{h}_2) = \mathcal{A}_2$.

Proof. Let $\mathfrak{h}_c \in \mathsf{chunks}(\mathfrak{s}, \mathfrak{h})$. Since chunks cannot be split into smaller heaps using $\uplus^{\mathfrak{s}}$, there are only two possibilities: either there exists an $\mathcal{A}_1'$ such that $\mathcal{A}_1 = \mathsf{ams}(\mathfrak{s}, \mathfrak{h}_c) \bullet \mathcal{A}_1'$ or there exists an $\mathcal{A}_2'$ such that

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

$\mathcal{A}_2 = \mathsf{ams}(\mathfrak{s}, \mathfrak{h}_c) \bullet \mathcal{A}'_2$. In the former case, we add $\mathfrak{h}_c$ to $\mathfrak{h}_1$, in the latter case we add it to $\mathfrak{h}_2$.

By construction, we obtain for $1 \leq i \leq 2$ that $\mathcal{A}_i$ is the composition of the abstractions of all chunks of $\mathfrak{h}_i$, i.e., $\mathsf{ams}(\mathfrak{s}, \mathfrak{h}_i) = \mathcal{A}_i$. □

Theorem 6.28 (Refinement theorem for **SSL**). *Let* $\phi \in$ **SSL** *and let* $(\mathfrak{s}, \mathfrak{h}_1)$, $(\mathfrak{s}, \mathfrak{h}_2)$ *be models with* $\mathsf{ams}(\mathfrak{s}, \mathfrak{h}_1) = \mathsf{ams}(\mathfrak{s}, \mathfrak{h}_2)$. *Then* $(\mathfrak{s}, \mathfrak{h}_1) \models \phi$ *iff* $(\mathfrak{s}, \mathfrak{h}_2) \models \phi$.

Proof. Let $\mathcal{A} \triangleq \langle V, E, \rho, \gamma \rangle$ be the AMS for which $\mathsf{ams}(\mathfrak{s}, \mathfrak{h}_1) = \mathcal{A} = \mathsf{ams}(\mathfrak{s}, \mathfrak{h}_2)$ holds. We proceed by induction on the structure of ϕ. I present only one direction of the proof, as the proof of the other direction is completely analogous.

Assume that the claim holds for all subformulas of ϕ and assume that $(\mathfrak{s}, \mathfrak{h}_1) \models \phi$. In particular, this implies that $\mathcal{A} \in \mathsf{ams}_{\mathfrak{s}}(\phi)$—a fact that we will use throughout this proof. We show that $(\mathfrak{s}, \mathfrak{h}_2) \models \phi$.

CASE **emp**. Immediate consequence of Lemma 6.17.

CASE $x \approx y$. As both models have the same stack, this is an immediate consequence of Lemma 6.18.

CASE $x \not\approx y$. Analogous.

CASE $x \mapsto_{\mathsf{n}} y$, $x \mapsto_{\mathsf{l,r}} \mathbf{y}$. Immediate consequence of Lemma 6.19.

CASE $\mathsf{pred}(x, \mathbf{y})$, $\mathsf{pred} \in \{\mathsf{ls}, \mathsf{ls}_{\geq 2}, \mathsf{tree}, \mathsf{tree}_{\geq 2}\}$. Immediate consequence of Lemma 6.22.

CASE $\phi_1 \star \phi_2$. By the semantics of $\star$, there exist $\mathfrak{h}_{1,1}, \mathfrak{h}_{1,2}$ with $\mathfrak{h}_1 = \mathfrak{h}_{1,1} \uplus^{\mathfrak{s}} \mathfrak{h}_{1,2}$, $(\mathfrak{s}, \mathfrak{h}_{1,1}) \models \phi_1$, and $(\mathfrak{s}, \mathfrak{h}_{1,2}) \models \phi_2$. Let $\mathcal{A}_1 \triangleq \mathsf{ams}(\mathfrak{s}, \mathfrak{h}_{1,1})$ and $\mathcal{A}_2 \triangleq \mathsf{ams}(\mathfrak{s}, \mathfrak{h}_{1,2})$. By Lemma 6.33, $\mathsf{ams}(\mathfrak{s}, \mathfrak{h}_1) = \mathcal{A}_1 \bullet \mathcal{A}_2 = \mathsf{ams}(\mathfrak{s}, \mathfrak{h}_2)$. We can thus apply Lemma 6.27 to $\mathsf{ams}(\mathfrak{s}, \mathfrak{h}_2)$, $\mathcal{A}_1$, and $\mathcal{A}_2$ to obtain heaps $\mathfrak{h}_{2,1}, \mathfrak{h}_{2,2}$ with $\mathfrak{h}_2 = \mathfrak{h}_{2,1} \uplus^{\mathfrak{s}} \mathfrak{h}_{2,2}$, $\mathsf{ams}(\mathfrak{s}, \mathfrak{h}_{2,1}) = \mathcal{A}_1$ and $\mathsf{ams}(\mathfrak{s}, \mathfrak{h}_{2,2}) = \mathcal{A}_2$.

For $1 \leq i \leq 2$, we apply the induction hypotheses for ϕ_i, $\mathfrak{h}_{1,i}$ and $\mathfrak{h}_{2,i}$ to obtain that $\mathfrak{h}_{2,i} \models \phi_i$. By the semantics of $\star$, we then have $(\mathfrak{s}, \mathfrak{h}_2) = (\mathfrak{s}, \mathfrak{h}_{2,1} \uplus^{\mathfrak{s}} \mathfrak{h}_{2,2}) \models \phi_1 \star \phi_2$.

CASE $\phi_1 \mathrel{-\!\!\star} \phi_2$. Since $(\mathfrak{s}, \mathfrak{h}_1) \models \phi_1 \mathrel{-\!\!\star} \phi_2$, it holds for all heaps $\mathfrak{h}_0$ with $(\mathfrak{s}, \mathfrak{h}_0) \models \phi_1$ and $\mathfrak{h}_1 \uplus^{\mathfrak{s}} \mathfrak{h}_0 \neq \bot$ that $(\mathfrak{s}, \mathfrak{h}_1 \uplus^{\mathfrak{s}} \mathfrak{h}_0) \models \phi_2$. Let $\mathfrak{h}_0$ be an arbitrary such heap. We assume w.l.o.g. that $\mathfrak{h}_2 \uplus^{\mathfrak{s}} \mathfrak{h}_0 \neq \bot$—if this is not the case, simply replace $\mathfrak{h}_0$ with a heap $\mathfrak{h}'_0$ with $(\mathfrak{s}, \mathfrak{h}_0) \cong (\mathfrak{s}, \mathfrak{h}'_0)$ for which $\mathfrak{h}_2 \uplus^{\mathfrak{s}} \mathfrak{h}'_0 \neq \bot$. This is always possible, because all locations in $\mathsf{locs}(\mathfrak{h}_0) \setminus \mathrm{img}(\mathfrak{s})$ can be renamed arbitrarily, so in particular to locations that are neither in $\mathsf{locs}(\mathfrak{h}_1) \setminus \mathrm{img}(\mathfrak{s})$ nor in $\mathsf{locs}(\mathfrak{h}_2) \setminus \mathrm{img}(\mathfrak{s})$. By Lemma 5.7, $(\mathfrak{s}, \mathfrak{h}_1 \uplus^{\mathfrak{s}} \mathfrak{h}'_0) \models \phi_2$.

Note that $\mathsf{ams}(\mathfrak{s}, \mathfrak{h}_1 \uplus^{\mathfrak{s}} \mathfrak{h}_0) = \mathsf{ams}(\mathfrak{s}, \mathfrak{h}_1) \bullet \mathsf{ams}(\mathfrak{s}, \mathfrak{h}_0) = \mathsf{ams}(\mathfrak{s}, \mathfrak{h}_2) \bullet \mathsf{ams}(\mathfrak{s}, \mathfrak{h}_0) = \mathsf{ams}(\mathfrak{s}, \mathfrak{h}_2 \uplus^{\mathfrak{s}} \mathfrak{h}_0)$ (by assumption and Lemma 6.33). It therefore follows from the induction hypothesis for ϕ_2, $(\mathfrak{s}, \mathfrak{h}_1 \uplus^{\mathfrak{s}}$

TU Bibliothek, Your knowledge hub. WIEN
Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.

$\mathfrak{h}_0)$, and $(\mathfrak{s}, \mathfrak{h}_2 \uplus^{\mathfrak{s}} \mathfrak{h}_0)$ that $(\mathfrak{s}, \mathfrak{h}_2 \uplus^{\mathfrak{s}} \mathfrak{h}_0) \models \phi_2$ and thus $(\mathfrak{s}, \mathfrak{h}_2) \models \phi_1 \circledast \phi_2$.

CASE $\phi_1 \wedge \phi_2$. By the semantics of $\wedge$, we have both $(\mathfrak{s}, \mathfrak{h}_1) \models \phi_1$ and $(\mathfrak{s}, \mathfrak{h}_1) \models \phi_2$. We apply the induction hypotheses for ϕ_1 and ϕ_2 to obtain $(\mathfrak{s}, \mathfrak{h}_2) \models \phi_1$ and $(\mathfrak{s}, \mathfrak{h}_2) \models \phi_2$. By the semantics of $\wedge$, we then have $(\mathfrak{s}, \mathfrak{h}_2) \models \phi_1 \wedge \phi_2$.

CASE $\phi_1 \vee \phi_2$. By the semantics of $\vee$, we have $(\mathfrak{s}, \mathfrak{h}_1) \models \phi_1$ or $(\mathfrak{s}, \mathfrak{h}_1) \models \phi_2$. Assume w.l.o.g. that $(\mathfrak{s}, \mathfrak{h}_1) \models \phi_1$. We apply the induction hypothesis for ϕ_1 to obtain $(\mathfrak{s}, \mathfrak{h}_2) \models \phi_1$. By the semantics of $\vee$, we then have $(\mathfrak{s}, \mathfrak{h}_2) \models \phi_1 \vee \phi_2$.

CASE $\neg\phi_1$. By the semantics of $\neg$, we have $(\mathfrak{s}, \mathfrak{h}_1) \not\models \phi_1$. By the induction hypothesis for ϕ_1 we then obtain $(\mathfrak{s}, \mathfrak{h}_2) \not\models \phi_1$. By the semantics of $\neg$, we have $(\mathfrak{s}, \mathfrak{h}_2) \models \neg\phi_1$. □

Corollary 6.29. *Let $(\mathfrak{s}, \mathfrak{h})$ be a model and $\phi \in$ **SSL**. $(\mathfrak{s}, \mathfrak{h}) \models \phi$ iff $\mathsf{ams}(\mathfrak{s}, \mathfrak{h}) \in \mathsf{ams}_{\mathfrak{s}}(\phi)$.*

Proof. Let $\mathcal{A} \triangleq \mathsf{ams}(\mathfrak{s}, \mathfrak{h})$. Because $\mathcal{A} \in \mathsf{ams}_{\mathfrak{s}}(\phi)$, there exists by definition of $\mathsf{ams}_{\mathfrak{s}}$ a model $(\mathfrak{s}, \mathfrak{h}')$ such that $(\mathfrak{s}, \mathfrak{h}') \models \phi$ and $\mathsf{ams}(\mathfrak{s}, \mathfrak{h}') = \mathcal{A}$. By applying Theorem 6.28 to ϕ, $(\mathfrak{s}, \mathfrak{h})$ and $(\mathfrak{s}, \mathfrak{h}')$, it then follows that $(\mathfrak{s}, \mathfrak{h}) \models \phi$. □

6.3 COMPUTING ABSTRACT MEMORY STATES

In this section, I show how to compute the AMS of arbitrary **SSL** formulas. I proceed in two steps. First, I derive recursive equations that reduce the set of AMS $\mathsf{ams}_{\mathfrak{s}}(\phi)$ for arbitrary compound **SSL** formulas to the set of AMS of the constituent formulas of ϕ. I then show that we can actually evaluate these equations, as well as the set of AMS we found for atomic formulas in Section 6.2.1, thus obtaining an algorithm for computing the abstraction of arbitrary **SSL** formulas.

6.3.1 *Abstracting Non-Atomic SSL Formulas*

ABSTRACTING THE BOOLEAN OPERATORS. The refinement theorem ensures that we can express the sets of AMS of Boolean operators $\wedge$, $\vee$ and $\neg$ in terms of intersection, union, and complement.

Lemma 6.30. $\mathsf{ams}_{\mathfrak{s}}(\phi_1 \wedge \phi_2) = \mathsf{ams}_{\mathfrak{s}}(\phi_1) \cap \mathsf{ams}_{\mathfrak{s}}(\phi_2)$.

Proof. ($\Rightarrow$) Assume $\mathcal{A} \in \mathsf{ams}_{\mathfrak{s}}(\phi_1 \wedge \phi_2)$. Let $(\mathfrak{s}, \mathfrak{h})$ be a model with $\mathsf{ams}(\mathfrak{s}, \mathfrak{h}) = \mathcal{A}$. By Corollary 6.29, $(\mathfrak{s}, \mathfrak{h}) \models \phi_1 \wedge \phi_2$. Consequently, $(\mathfrak{s}, \mathfrak{h}) \models \phi_1$ and $(\mathfrak{s}, \mathfrak{h}) \models \phi_2$. Thus, $\mathcal{A} \in \mathsf{ams}_{\mathfrak{s}}(\phi_1)$ and $\mathcal{A} \in \mathsf{ams}_{\mathfrak{s}}(\phi_2)$, i.e., $\mathcal{A} \in \mathcal{A} \in \mathsf{ams}_{\mathfrak{s}}(\phi_1) \cap \mathsf{ams}_{\mathfrak{s}}(\phi_2)$.

($\Leftarrow$) Let $\mathcal{A} \in \mathsf{ams}_{\mathfrak{s}}(\phi_1) \cap \mathsf{ams}_{\mathfrak{s}}(\phi_2)$. Let $(\mathfrak{s}, \mathfrak{h})$ be such that $\mathsf{ams}(\mathfrak{s}, \mathfrak{h}) = \mathcal{A}$. By Corollary 6.29, $(\mathfrak{s}, \mathfrak{h}) \models \phi_1$ and $(\mathfrak{s}, \mathfrak{h}) \models \phi_2$ and thus $(\mathfrak{s}, \mathfrak{h}) \models \phi_1 \wedge \phi_2$. It follows that $\mathcal{A} \in \mathsf{ams}_{\mathfrak{s}}(\phi_1 \wedge \phi_2)$. □

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Lemma 6.31. $\mathsf{ams}_{\mathfrak{s}}(\phi_1 \vee \phi_2) = \mathsf{ams}_{\mathfrak{s}}(\phi_1) \cup \mathsf{ams}_{\mathfrak{s}}(\phi_2)$.

Proof. ($\Rightarrow$) Assume $\mathcal{A} \in \mathsf{ams}_{\mathfrak{s}}(\phi_1 \vee \phi_2)$. Let $(\mathfrak{s}, \mathfrak{h})$ be a model with $\mathsf{ams}(\mathfrak{s}, \mathfrak{h}) = \mathcal{A}$. By Corollary 6.29, $(\mathfrak{s}, \mathfrak{h}) \models \phi_1 \vee \phi_2$. Consequently, $(\mathfrak{s}, \mathfrak{h}) \models \phi_1$ or $(\mathfrak{s}, \mathfrak{h}) \models \phi_2$. Assume w.l.o.g. that $(\mathfrak{s}, \mathfrak{h}) \models \phi_1$. Thus, $\mathcal{A} \in \mathsf{ams}_{\mathfrak{s}}(\phi_1) \subseteq \mathsf{ams}_{\mathfrak{s}}(\phi_1) \cup \mathsf{ams}_{\mathfrak{s}}(\phi_2)$.

($\Leftarrow$) Let $\mathcal{A} \in \mathsf{ams}_{\mathfrak{s}}(\phi_1) \cup \mathsf{ams}_{\mathfrak{s}}(\phi_2)$. Assume w.l.o.g. that $\mathcal{A} \in \mathsf{ams}_{\mathfrak{s}}(\phi_1)$. Let $(\mathfrak{s}, \mathfrak{h})$ be a model with $\mathsf{ams}(\mathfrak{s}, \mathfrak{h}) = \mathcal{A}$. By Corollary 6.29, $(\mathfrak{s}, \mathfrak{h}) \models \phi_1$ and thus $(\mathfrak{s}, \mathfrak{h}) \models \phi_1 \vee \phi_2$. It follows that $\mathcal{A} \in \mathsf{ams}_{\mathfrak{s}}(\phi_1 \vee \phi_2)$. □

Lemma 6.32. $\mathsf{ams}_{\mathfrak{s}}(\neg\phi_1) = \{\mathsf{ams}(\mathfrak{s}, \mathfrak{h}) \mid \mathfrak{h} \in \mathbf{Heaps}\} \setminus \mathsf{ams}_{\mathfrak{s}}(\phi_1)$.

Proof. ($\Rightarrow$) Assume $\mathcal{A} \in \mathsf{ams}_{\mathfrak{s}}(\neg\phi_1)$. Let $(\mathfrak{s}, \mathfrak{h})$ be such that $\mathsf{ams}(\mathfrak{s}, \mathfrak{h}) = \mathcal{A}$. By Corollary 6.29, $(\mathfrak{s}, \mathfrak{h}) \models \neg\phi_1$ and thus $(\mathfrak{s}, \mathfrak{h}) \not\models \phi_1$. By definition, we then have that $\mathcal{A} \notin \mathsf{ams}_{\mathfrak{s}}(\phi_1)$, i.e.,

$$\mathcal{A} \in \{\mathsf{ams}(\mathfrak{s}, \mathfrak{h}) \mid \mathfrak{h} \in \mathbf{Heaps}\} \setminus \mathsf{ams}_{\mathfrak{s}}(\phi_1).$$

($\Leftarrow$) Let $\mathcal{A} \in \{\mathsf{ams}(\mathfrak{s}, \mathfrak{h}) \mid \mathfrak{h} \in \mathbf{Heaps}\} \setminus \mathsf{ams}_{\mathfrak{s}}(\phi_1)$. Let $(\mathfrak{s}, \mathfrak{h})$ be such that $\mathsf{ams}(\mathfrak{s}, \mathfrak{h}) = \mathcal{A}$. Because $\mathcal{A} \notin \mathsf{ams}_{\mathfrak{s}}(\phi_1)$, it follows by Corollary 6.29 that $(\mathfrak{s}, \mathfrak{h}) \not\models \phi_1$. By the semantics of $\neg$, $(\mathfrak{s}, \mathfrak{h}) \models \neg\phi_1$. It follows that $\mathcal{A} \in \mathsf{ams}_{\mathfrak{s}}(\neg\phi_1)$. □

ABSTRACTING THE SEPARATING CONJUNCTION. In Section 6.2.2, we defined the composition operation, $\bullet$, on pairs of AMS. We now lift this operation to sets of AMS in a point-wise manner. Let $\mathbf{A}_1, \mathbf{A}_2 \subseteq \mathbf{AMS}_{\mathfrak{s}}$ be sets of AMS w.r.t. some stack $\mathfrak{s}$. We define

$$\mathbf{A}_1 \bullet \mathbf{A}_2 \triangleq \{\mathcal{A}_1 \bullet \mathcal{A}_2 \mid \mathcal{A}_1 \in \mathbf{A}_1, \mathcal{A}_2 \in \mathbf{A}_2, \mathcal{A}_1 \bullet \mathcal{A}_2 \neq \bot\}.$$

Lemma 6.26 implies that $\mathsf{ams}_{\mathfrak{s}}$ is a homomorphism from formulas and the separating conjunction, $\star$, to sets of AMS and the composition operator, $\bullet$:

Lemma 6.33 (Correctness of AMS composition)**.** *Let* $\phi_1, \phi_2 \in \mathbf{SSL}$. *Then* $\mathsf{ams}_{\mathfrak{s}}(\phi_1 \star \phi_2) = \mathsf{ams}_{\mathfrak{s}}(\phi_1) \bullet \mathsf{ams}_{\mathfrak{s}}(\phi_2)$.

Proof. Let $\mathcal{A} \in \mathsf{ams}_{\mathfrak{s}}(\phi_1 \star \phi_2)$. There then exists a heap $\mathfrak{h}$ such that $(\mathfrak{s}, \mathfrak{h}) \models \phi_1 \star \phi_2$ and $\mathsf{ams}(\mathfrak{s}, \mathfrak{h}) = \mathcal{A}$. By the semantics of $\star$, we can split $\mathfrak{h}$ into $\mathfrak{h}_1 \uplus^{\mathfrak{s}} \mathfrak{h}_2$ with $(\mathfrak{s}, \mathfrak{h}_i) \models \phi_i$ (and thus $\mathsf{ams}(\mathfrak{s}, \mathfrak{h}_i) \in \mathsf{ams}_{\mathfrak{s}}(\phi_i)$). By Lemma 6.26, $\mathcal{A} = \mathsf{ams}(\mathfrak{s}, \mathfrak{h}_1) \bullet \mathsf{ams}(\mathfrak{s}, \mathfrak{h}_2)$ for $\mathfrak{h}_1$, $\mathfrak{h}_2$ as above. Consequently, $\mathcal{A} \in \mathsf{ams}_{\mathfrak{s}}(\phi_1) \bullet \mathsf{ams}_{\mathfrak{s}}(\phi_2)$ by definition of $\bullet$.

Conversely, let $\mathcal{A} \in \mathsf{ams}_{\mathfrak{s}}(\phi_1) \bullet \mathsf{ams}_{\mathfrak{s}}(\phi_2)$. By definition of $\bullet$, there then exist $\mathcal{A}_i \in \mathsf{ams}_{\mathfrak{s}}(\phi_i)$ such that $\mathcal{A} = \mathcal{A}_1 \bullet \mathcal{A}_2$. Let $\mathfrak{h}_1, \mathfrak{h}_2$ be witnesses of this fact, i.e., such that $(\mathfrak{s}, \mathfrak{h}_i) \models \phi_i$ and $\mathsf{ams}(\mathfrak{s}, \mathfrak{h}_i) = \mathcal{A}_i$. Assume w.l.o.g. that $\mathfrak{h}_1 \uplus^{\mathfrak{s}} \mathfrak{h}_2 \neq \bot$. (Otherwise, replace $\mathfrak{h}_2$ with an $\mathfrak{h}_2'$ such that $(\mathfrak{s}, \mathfrak{h}_2) \cong (\mathfrak{s}, \mathfrak{h}_2')$ and $\mathfrak{h}_1 \uplus^{\mathfrak{s}} \mathfrak{h}_2' \neq \bot$.)

By the semantics of $\star$, $(\mathfrak{s}, \mathfrak{h}_1 \uplus^{\mathfrak{s}} \mathfrak{h}_2) \models \phi_1 \star \phi_2$. Therefore, $\mathsf{ams}(\mathfrak{s}, \mathfrak{h}_1 \uplus^{\mathfrak{s}} \mathfrak{h}_2) \in \mathsf{ams}_{\mathfrak{s}}(\phi_1 \star \phi_2)$. By Lemma 6.33, $\mathsf{ams}(\mathfrak{s}, \mathfrak{h}_1 \uplus^{\mathfrak{s}} \mathfrak{h}_2) = \mathcal{A}$. The claim follows. □

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

ABSTRACT MAGIC WANDS. Finally, we formalize the abstraction of magic wands in terms of an *abstract magic wand*, $-\!\!\bullet$, which relates to $\bullet$ in the same way that $-\!\!\star$ relates to $\star$.

Definition 6.34. *Let* $\mathbf{A}_1, \mathbf{A}_2 \subseteq \mathbf{AMS}$*. The* abstract magic wand, $\mathbf{A}_1 -\!\!\bullet \mathbf{A}_2$, *is given by*

$$\mathbf{A}_1 -\!\!\bullet \mathbf{A}_2 \triangleq \{\mathcal{A} \in \mathbf{AMS} \mid \textit{for all } \mathcal{A}_1 \in \mathbf{A}_1 \textit{ if } \mathcal{A} \bullet \mathcal{A}_1 \neq \bot \\ \textit{then } \mathcal{A} \bullet \mathcal{A}_1 \in \mathbf{A}_2\}$$

Using the refinement theorem, in the form of Corollary 6.29, and the properties of $\bullet$ showed in Lemmas 6.26 and 6.27, we can show that $\mathsf{ams}_\mathfrak{s}$ is a homomorphism from SSL formulas and $-\!\!\star$ to sets of AMS and the abstract magic wand, $-\!\!\bullet$.

Lemma 6.35. *For all* $\phi_1, \phi_2 \in \mathbf{SSL}$*, it holds that*

$$\mathsf{ams}_\mathfrak{s}(\phi_1 -\!\!\star \phi_2) = \mathsf{ams}_\mathfrak{s}(\phi_1) -\!\!\bullet \mathsf{ams}_\mathfrak{s}(\phi_2).$$

Proof. Define $\mathbf{A}_i \triangleq \mathsf{ams}_\mathfrak{s}(\phi_i)$, $1 \leq i \leq 2$. Let $\mathcal{A} \in \mathsf{ams}_\mathfrak{s}(\phi_1 -\!\!\star \phi_2)$. Then there exists a model $(\mathfrak{s}, \mathfrak{h})$ with $\mathsf{ams}(\mathfrak{s}, \mathfrak{h}) = \mathcal{A}$ and $(\mathfrak{s}, \mathfrak{h}) \models \phi_1 -\!\!\star \phi_2$.

Let $\mathcal{A}_1 \in \mathbf{A}_1$ be an arbitrary AMS with $\mathcal{A} \bullet \mathcal{A}_1 \neq \bot$. By definition of $\mathbf{A}_1$ there exists a heap $\mathfrak{h}_1$ such that $(\mathfrak{s}, \mathfrak{h}_1) \models \phi_1$ and $\mathsf{ams}(\mathfrak{s}, \mathfrak{h})$. Since $\mathcal{A} \bullet \mathcal{A}_1 \neq \bot$, we can assume w.l.o.g. that $\mathfrak{h} \uplus^\mathfrak{s} \mathfrak{h}_1 \neq \bot$—otherwise, simply replace $\mathfrak{h}_1'$ with an appropriate isomorphic heap, i.e., such that $(\mathfrak{s}, \mathfrak{h}_1) \cong (\mathfrak{s}, \mathfrak{h}_1')$ and $\mathfrak{h} \uplus^\mathfrak{s} \mathfrak{h}_1' \neq \bot$.

By the semantics of $-\!\!\star$, it follows that $(\mathfrak{s}, \mathfrak{h} \uplus^\mathfrak{s} \mathfrak{h}_1) \models \phi_2$. Consequently, $\mathsf{ams}(\mathfrak{s}, \mathfrak{h} \uplus^\mathfrak{s} \mathfrak{h}_1) \in \mathbf{A}_2$. By Lemma 6.26, $\mathsf{ams}(\mathfrak{s}, \mathfrak{h} \uplus^\mathfrak{s} \mathfrak{h}_1) = \mathsf{ams}(\mathfrak{s}, \mathfrak{h}) \bullet \mathsf{ams}(\mathfrak{s}, \mathfrak{h}_1) = \mathcal{A} \bullet \mathcal{A}_1$, so $\mathcal{A} \bullet \mathcal{A}_1 \in \mathbf{A}_2$.

Since $\mathcal{A}_1$ was arbitrary, it follows that $\mathcal{A} \in \mathbf{A}_1 -\!\!\bullet \mathbf{A}_2$.

Conversely, let $\mathcal{A} \in \mathbf{A}_1 -\!\!\bullet \mathbf{A}_2$. Let $\mathfrak{h}, \mathfrak{h}_1$ be arbitrary heaps with $\mathsf{ams}(\mathfrak{s}, \mathfrak{h}) = \mathcal{A}$, $\mathsf{ams}(\mathfrak{s}, \mathfrak{h}_1) \in \mathbf{A}_1$, $(\mathfrak{s}, \mathfrak{h}_1) \models \phi_1$, and $\mathfrak{h} \uplus^\mathfrak{s} \mathfrak{h}_1 \neq \bot$. Let $\mathcal{A}_1 \triangleq \mathsf{ams}(\mathfrak{s}, \mathfrak{h}_1)$. Note that $\mathcal{A} \bullet \mathcal{A}_1 \neq \bot$. Since $\mathcal{A}_1 \in \mathbf{A}_1$ and $\mathcal{A} \in \mathbf{A}_1 -\!\!\bullet \mathbf{A}_2$, it follows by definition of $-\!\!\bullet$ that $\mathcal{A} \bullet \mathcal{A}_1 \in \mathbf{A}_2$. Further, by Lemma 6.26, we then have $\mathsf{ams}(\mathfrak{s}, \mathfrak{h} \uplus^\mathfrak{s} \mathfrak{h}_1) = \mathcal{A} \bullet \mathcal{A}_1 \in \mathbf{A}_2$. By Corollary 6.29, this allows us to conclude that $(\mathfrak{s}, \mathfrak{h} \uplus^\mathfrak{s} \mathfrak{h}_1) \models \phi_2$. Consequently, $(\mathfrak{s}, \mathfrak{h}) \models \phi_1 -\!\!\star \phi_2$. Since $\mathfrak{h}_1$ was arbitrary, it follows that $\mathcal{A} \in \mathsf{ams}_\mathfrak{s}(\phi_1 -\!\!\star \phi_2)$. □

6.3.2 *Refining the Refinement Theorem: Bounding Garbage*

Even though we have now characterized the set $\mathsf{ams}_\mathfrak{s}(\phi)$ for every formula ϕ, we do not yet have a way to implement AMS computation: while $\mathsf{ams}_\mathfrak{s}(\phi)$ is finite if ϕ is a spatial atom, the set is infinite if the formula contains negation (Lemma 6.32) or magic wands (Lemma 6.35).

As every node in V has to contain at least one stack variable, there are for every fixed stack $\mathfrak{s}$ only finitely many ways to pick the nodes V, the edges E and the negative-allocation constraint ρ. The garbage-chunk count can, however, be an arbitrary natural number. Fortunately,

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

to decide the satisfiability of any fixed formula ϕ, it is *not* necessary to keep track of arbitrarily large garbage-chunk counts.

We introduce a notion of *chunk size* of formula ϕ, $\lceil\phi\rceil$, that provides an upper bound on the number of chunks that may be necessary to satisfy and/or falsify the formula.

Definition 6.36 (Chunk size). *Let $\phi \in$ **SSL**. The* chunk size *of ϕ, denoted $\lceil\phi\rceil$, is defined as follows.*

- $\lceil\tau\rceil \triangleq 1$ *if τ is atomic*
- $\lceil\phi \star \psi\rceil \triangleq \lceil\phi\rceil + \lceil\psi\rceil$
- $\lceil\phi \mathrel{-\!\!\star} \psi\rceil \triangleq \lceil\psi\rceil$
- $\lceil\phi \wedge \psi\rceil \triangleq \max(\lceil\phi\rceil, \lceil\psi\rceil)$, $\lceil\phi \vee \psi\rceil \triangleq \max(\lceil\phi\rceil, \lceil\psi\rceil)$
- $\lceil\neg\phi\rceil \triangleq \lceil\phi\rceil$.

Observe that $\lceil\phi\rceil \leq |\phi|$ for all ϕ. Intuitively, $\lceil\phi\rceil - 1$ is an upper bound on the number of times the operation $\uplus^{\mathfrak{s}}$ is applied when checking whether $(\mathfrak{s}, \mathfrak{h}) \models \phi$.

Example 6.37 (Chunk size). *Let $\psi = (x \mapsto_{\mathsf{n}} y) \star ((b \mapsto_{\mathsf{n}} c) \mathrel{-\!\!\circledast} \mathsf{ls}(a, c))$. Then $\lceil\psi\rceil = 2$; and to check that ψ holds in a model that consists of a pointer from x to y and a list segment from a to b, it suffices to split this model once—i.e., $(\lceil\psi\rceil - 1)$ many times—using $\uplus^{\mathfrak{s}}$, obtaining sub-heaps that correspond to the points-to assertion and the list segment.*

We generalize the refinement theorem, Theorem 6.28, to models whose AMS differ in their garbage-chunk count, provided both garbage-chunk counts exceed the chunk size of the formula.

Theorem 6.38 (Refined refinement theorem for **SSL**). *Let $\phi \in$ **SSL** be a formula with $\lceil\phi\rceil = k$. Let $m \geq k$, $n \geq k$ and let $(\mathfrak{s}, \mathfrak{h}_1), (\mathfrak{s}, \mathfrak{h}_2)$ be models with $\mathsf{ams}(\mathfrak{s}, \mathfrak{h}_1) = \langle V, E, \rho, m\rangle$, $\mathsf{ams}(\mathfrak{s}, \mathfrak{h}_2) = \langle V, E, \rho, n\rangle$. Then $(\mathfrak{s}, \mathfrak{h}_1) \models \phi$ iff $(\mathfrak{s}, \mathfrak{h}_2) \models \phi$.*

Proof. If $m = n$, the result follows from Theorem 6.28. If $m \neq n$, assume w.l.o.g. that $m < n$. We proceed by structural induction on ϕ. We only prove one implication, as the proof of the other direction is very similar.

ATOMIC FORMULAS. As we saw in Section 6.2.1, the AMS of all models of atomic SSL formulas have a garbage-chunk count of 0. Moreover, by definition of chunk size, $\lceil\phi\rceil = 1$ for all atomic ϕ. Since $m \geq \lceil\phi\rceil = 1$ and $n > m$, it follows that $(\mathfrak{s}, \mathfrak{h}_1) \not\models \phi$ and $(\mathfrak{s}, \mathfrak{h}_2) \not\models \phi$.

CASE $\phi_1 \star \phi_2$. Assume $(\mathfrak{s}, \mathfrak{h}_1) \models \phi_1 \star \phi_2$. Let $\mathfrak{h}_{1,1}, \mathfrak{h}_{1,2}$ be such that $\mathfrak{h}_1 = \mathfrak{h}_{1,1} \uplus^{\mathfrak{s}} \mathfrak{h}_{1,2}$, $(\mathfrak{s}, \mathfrak{h}_{1,1}) \models \phi_1$, and $(\mathfrak{s}, \mathfrak{h}_{1,2}) \models \phi_2$. Let

$$\mathcal{A}_1 = \langle V_1, E_1, \rho_1, m_1\rangle \triangleq \mathsf{ams}(\mathfrak{s}, \mathfrak{h}_{1,1}) \text{ and}$$
$$\mathcal{A}_2 = \langle V_2, E_2, \rho_2, m_2\rangle \triangleq \mathsf{ams}(\mathfrak{s}, \mathfrak{h}_{1,2}).$$

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Since $m \geq \lceil \phi_1 \rceil + \lceil \phi_2 \rceil$, it follows that, either $m_1 \geq \lceil \phi_1 \rceil$ or $m_2 \geq \lceil \phi_2 \rceil$ (or both). Assume w.l.o.g. that $m_1 \geq \lceil \phi_1 \rceil$ and let $\mathcal{A}_1' \triangleq \langle V_1, E_1, \rho_1, n - m_2 \rangle$. Observe that $\mathsf{ams}(\mathfrak{s}, \mathfrak{h}_2) = \mathcal{A}_1' \bullet \mathcal{A}_2$. There thus exist by Lemma 6.27 heaps $\mathfrak{h}_{2,1}, \mathfrak{h}_{2,2}$ such that $(\mathfrak{s}, \mathfrak{h}_2) = \mathfrak{h}_{2,1} \uplus^{\mathfrak{s}} \mathfrak{h}_{2,2}$, $\mathsf{ams}(\mathfrak{s}, \mathfrak{h}_{2,1}) = \mathcal{A}_1'$ and $\mathsf{ams}(\mathfrak{s}, \mathfrak{h}_{2,1}) = \mathcal{A}_2$. As both $m_1 \geq \lceil \phi_1 \rceil$ and $n - m_2 > m - m_2 = m_1 \geq \lceil \phi_1 \rceil$, we have by the induction hypothesis for ϕ_1 that $(\mathfrak{s}, \mathfrak{h}_{2,1}) \models \phi_1$. Additionally, we have $\mathfrak{h}_{2,2} \models \phi_2$ by Theorem 6.28. Consequently, $(\mathfrak{s}, \mathfrak{h}_2) \models \phi_1 \star \phi_2$.

CASE $\phi_1 \mathrel{-\!\!\star} \phi_2$. Assume $(\mathfrak{s}, \mathfrak{h}_1) \models \phi_1 \mathrel{-\!\!\star} \phi_2$. Let $\mathfrak{h}'$ be such that $(\mathfrak{s}, \mathfrak{h}') \models \phi_1$ and $(\mathfrak{s}, \mathfrak{h}' \uplus^{\mathfrak{s}} \mathfrak{h}_1) \models \phi_2$. Assume w.l.o.g. that $\mathfrak{h}_2 \uplus^{\mathfrak{s}} \mathfrak{h}' \neq \bot$—if this is not the case simply replace $\mathfrak{h}'$ with an isomorphic heap that has this property. Let

$$\mathcal{A}_2 = \langle V_2, E_2, \rho_2, m + m_2 \rangle = \mathsf{ams}(\mathfrak{s}, \mathfrak{h}' \uplus^{\mathfrak{s}} \mathfrak{h}_1) \text{ and}$$
$$\mathcal{A}_2' = \langle V_2, E_2, \rho_2, n + m_2 \rangle = \mathsf{ams}(\mathfrak{s}, \mathfrak{h}' \uplus^{\mathfrak{s}} \mathfrak{h}_2).$$

Trivially, $m + m_2 \geq m = \lceil \phi_2 \rceil$ and $n + m_2 \geq n > m = \lceil \phi_2 \rceil$. It thus follows from the induction hypothesis for ϕ_2 that $(\mathfrak{s}, \mathfrak{h}' \uplus^{\mathfrak{s}} \mathfrak{h}_2) \models \phi_2$. As $\mathfrak{h}'$ was arbitrary, it follows that $(\mathfrak{s}, \mathfrak{h}_2) \models \phi_1 \mathrel{-\!\!\star} \phi_2$.

CASE $\phi_1 \wedge \phi_2$. Assume $(\mathfrak{s}, \mathfrak{h}_1) \models \phi_1 \wedge \phi_2$ and hence $(\mathfrak{s}, \mathfrak{h}_1) \models \phi_1$ and $(\mathfrak{s}, \mathfrak{h}_1) \models \phi_2$. For $1 \leq i \leq 2$, we have by definition of $\lceil \phi_1 \wedge \phi_2 \rceil$ that $m \geq \max(\lceil \phi_1 \rceil, \lceil \phi_2 \rceil) \geq \lceil \phi_i \rceil$ and hence $n \geq \lceil \phi_i \rceil$. We thus conclude from the induction hypotheses that $(\mathfrak{s}, \mathfrak{h}_2) \models \phi_i$ and therefore, by the semantics of $\wedge$, $(\mathfrak{s}, \mathfrak{h}_2) \models \phi_1 \wedge \phi_2$.

CASE $\phi_1 \vee \phi_2$. Assume $(\mathfrak{s}, \mathfrak{h}_1) \models \phi_1 \vee \phi_2$. W.l.o.g., we have $(\mathfrak{s}, \mathfrak{h}_1) \models \phi_1$. By definition of $\lceil \phi_1 \vee \phi_2 \rceil$, it follows that

$$m \geq \max(\lceil \phi_1 \rceil, \lceil \phi_2 \rceil) \geq \lceil \phi_1 \rceil$$

and hence $n \geq \lceil \phi_1 \rceil$. We thus conclude from the induction hypothesis for ϕ_1 that $(\mathfrak{s}, \mathfrak{h}_2) \models \phi_1$ and therefore, by the semantics of $\vee$, $(\mathfrak{s}, \mathfrak{h}_2) \models \phi_1 \vee \phi_2$.

CASE $\neg \phi_1$. Assume $(\mathfrak{s}, \mathfrak{h}_1) \models \neg \phi_1$. Consequently, $(\mathfrak{s}, \mathfrak{h}_2) \not\models \phi_1$. Since $m \geq \lceil \neg \phi_1 \rceil = \lceil \phi_1 \rceil$ and $n > m$, it follows by induction that $(\mathfrak{s}, \mathfrak{h}_2) \not\models \phi_1$. By the semantics of negation, we conclude that $(\mathfrak{s}, \mathfrak{h}_2) \models \neg \phi_1$. □

This implies that ϕ is satisfiable over stack $\mathfrak{s}$ iff ϕ is satisfiable by a heap that contains at most $\lceil \phi \rceil$ garbage chunks.

Corollary 6.39. *Let ϕ be an* **SSL** *formula with $\lceil \phi \rceil = k$. Then ϕ is satisfiable over stack $\mathfrak{s}$ iff there exists a heap $\mathfrak{h}$ such that (1) $\mathsf{ams}(\mathfrak{s}, \mathfrak{h}) = (V, E, \rho, \gamma)$ for some $\gamma \leq k$ and (2) $(\mathfrak{s}, \mathfrak{h}) \models \phi$.*

Proof. Assume ϕ is satisfiable and let $(\mathfrak{s}, \mathfrak{h})$ be a model with $(\mathfrak{s}, \mathfrak{h}) \models \phi$. Let $\mathcal{A} = \langle V, E, \rho, \gamma \rangle \triangleq \mathsf{ams}(\mathfrak{s}, \mathfrak{h})$. If $\gamma \leq k$, there is nothing to show.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Otherwise, let $\mathcal{A}' \triangleq \langle V, E, \rho, k \rangle$. Let $\mathfrak{h}'$ be a heap such that $\mathsf{ams}(\mathfrak{s}, \mathfrak{h}') = \mathcal{A}'$; such a heap exists by Lemma 6.14. By Theorem 6.38, $(\mathfrak{s}, \mathfrak{h}') \models \phi$.

Conversely, if ϕ is not satisfiable over $\mathfrak{s}$, it is not satisfied by any heap; in particular, it is not satisfied by heaps with garbage-chunk count of at most k. □

6.4 DECIDING SSL BY AMS COMPUTATION

In light of Corollary 6.39, we can decide the SSL satisfiability problem by means of a function $\mathsf{abst}_{\mathfrak{s}}(\phi)$ that computes the (finite) intersection of the (possibly infinite) set $\mathsf{ams}_{\mathfrak{s}}(\phi)$ and the (finite) set

$$\mathbf{AMS}_{k,\mathfrak{s}} \triangleq \{\langle V, E, \rho, \gamma \rangle \in \mathbf{AMS} \mid V = \mathsf{classes}(\mathfrak{s}) \text{ and } \gamma \leq k\},$$

setting $k \triangleq \lceil \phi \rceil$. We define $\mathsf{abst}_{\mathfrak{s}}(\phi)$ in Fig. 6.4. The definition is straightforward except for the cases for $\star$, $\mathrel{-\!\!\star}$, $\wedge$, and $\vee$, which rely on *lifting* the bound on the garbage-chunk count from m to $n \geq m$.

The idea behind the lifting operation, formalized below, is to adapt the set computed for the chunk size of the subformulas, m, to the chunk size of the larger formula, $n \geq m$, by adding for every AMS of the form $\langle V, E, \rho, m \rangle$ all the AMS of the form $\langle V, E, \rho, j \rangle$, $m < j \leq n$. This is sound by Theorem 6.38.

Definition 6.40. *Let $m, n \in \mathbb{N}$ with $m \leq n$ and let $\mathcal{A} = \langle V, E, \rho, \gamma \rangle \in$ **AMS**. The* bound-lifting *of $\mathcal{A}$ from m to n is*

$$\mathsf{lift}_{m \nearrow n}(\mathcal{A}) \triangleq \begin{cases} \{\mathcal{A}\} & \text{if } \gamma < m \\ \{\langle V, E, \rho, k \rangle \mid m \leq k \leq n\} & \text{if } \gamma = m. \end{cases}$$

We apply bound-lifting to sets of AMS in a point-wise manner, i.e.,

$$\mathsf{lift}_{m \nearrow n}(\mathbf{A}) \triangleq \bigcup_{\mathcal{A} \in \mathbf{A}} \mathsf{lift}_{m \nearrow n}(\mathcal{A}).$$

Note that as a consequence of Theorem 6.38,

$$\mathsf{lift}_{\lceil \phi \rceil \nearrow n}(\mathsf{ams}_{\mathfrak{s}}(\phi) \cap \mathbf{AMS}_{\lceil \phi \rceil, \mathfrak{s}}) = \mathsf{ams}_{\mathfrak{s}}(\phi) \cap \mathbf{AMS}_{n,\mathfrak{s}}$$

for all $n \geq \lceil \phi \rceil$. By combining this observation with the lemmas characterizing $\mathsf{ams}_{\mathfrak{s}}$ (Lemmas 6.17 to 6.19, 6.22, 6.30 to 6.33 and 6.35), we obtain the correctness of $\mathsf{abst}_{\mathfrak{s}}(\phi)$.

Theorem 6.41 (Correctness of AMS computation)**.** *Let $\mathfrak{s}$ be a stack and let $\phi \in$ **SSL** be a formula. Then $\mathsf{abst}_{\mathfrak{s}}(\phi) = \mathsf{ams}_{\mathfrak{s}}(\phi) \cap \mathbf{AMS}_{\lceil \phi \rceil, \mathfrak{s}}$.*

Proof. We proceed by induction on the structure of ϕ.

CASE **emp**. By Lemma 6.17, $\mathsf{abst}_{\mathfrak{s}}(\mathbf{emp}) = \mathsf{ams}_{\mathfrak{s}}(\mathbf{emp})$. Moreover, as $\mathsf{ams}_{\mathfrak{s}}(\mathbf{emp}) \subseteq \mathbf{AMS}_{0,\mathfrak{s}}$, we also have $\mathsf{ams}_{\mathfrak{s}}(\mathbf{emp}) \subseteq \mathbf{AMS}_{\lceil \phi \rceil, \mathfrak{s}}$.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

$$
\begin{aligned}
\mathsf{abst}_{\mathfrak{s}}(\mathbf{emp}) &\triangleq \{\langle \mathsf{classes}(\mathfrak{s}), \emptyset, \emptyset, 0\rangle\}\\
\mathsf{abst}_{\mathfrak{s}}(x \approx y) &\triangleq \text{if } \mathfrak{s}(x) = \mathfrak{s}(y) \text{ then } \mathsf{abst}_{\mathfrak{s}}(\mathbf{emp}) \text{ else } \emptyset\\
\mathsf{abst}_{\mathfrak{s}}(x \not\approx y) &\triangleq \text{if } \mathfrak{s}(x) \neq \mathfrak{s}(y) \text{ then } \mathsf{abst}_{\mathfrak{s}}(\mathbf{emp}) \text{ else } \emptyset\\
\mathsf{abst}_{\mathfrak{s}}(x \mapsto_{\mathsf{n}} y) &\triangleq \{\langle \mathsf{classes}(\mathfrak{s}), \{[x]^{\mathfrak{s}}_{=} \mapsto [\mathbf{y}]^{\mathfrak{s}}_{=}\}, \emptyset, 0\rangle\}\\
\mathsf{abst}_{\mathfrak{s}}(x \mapsto_{\mathsf{l,r}} \mathbf{y}) &\triangleq \{\langle \mathsf{classes}(\mathfrak{s}), \{[x]^{\mathfrak{s}}_{=} \mapsto [\mathbf{y}]^{\mathfrak{s}}_{=}\}, \emptyset, 0\rangle\}\\
\mathsf{abst}_{\mathfrak{s}}(\mathsf{ls}(x, \mathbf{y})) &\triangleq \mathbf{AbstLists}(\mathfrak{s}, x, \mathbf{y})\\
\mathsf{abst}_{\mathfrak{s}}(\mathsf{ls}_{\geq 2}(x, \mathbf{y})) &\triangleq \mathbf{AbstLists}_{\geq 2}(\mathfrak{s}, x, \mathbf{y})\\
\mathsf{abst}_{\mathfrak{s}}(\mathsf{tree}(x, \mathbf{y})) &\triangleq \mathbf{AbstTrees}(\mathfrak{s}, x, \mathbf{y})\\
\mathsf{abst}_{\mathfrak{s}}(\mathsf{tree}_{\geq 2}(x, \mathbf{y})) &\triangleq \mathbf{AbstTrees}_{\geq 2}(\mathfrak{s}, x, \mathbf{y})\\
\mathsf{abst}_{\mathfrak{s}}(\phi_1 \star \phi_2) &\triangleq \mathbf{AMS}_{\lceil \phi_1 \star \phi_2 \rceil, \mathfrak{s}} \cap (\mathsf{lift}_{\lceil \phi_1 \rceil \nearrow \lceil \phi_1 \star \phi_2 \rceil}(\mathsf{abst}_{\mathfrak{s}}(\phi_1))\\
&\qquad \bullet \mathsf{lift}_{\lceil \phi_2 \rceil \nearrow \lceil \phi_1 \star \phi_2 \rceil}(\mathsf{abst}_{\mathfrak{s}}(\phi_2)))\\
\mathsf{abst}_{\mathfrak{s}}(\phi_1 \mathrel{-\!\!\star} \phi_2) &\triangleq \mathbf{AMS}_{\lceil \phi_1 \mathrel{-\!\!\star} \phi_2 \rceil, \mathfrak{s}} \cap\\
&\qquad (\mathsf{abst}_{\mathfrak{s}}(\phi_1) \mathrel{-\!\!\bullet} \mathsf{lift}_{\lceil \phi_2 \rceil \nearrow \lceil \phi_1 \rceil + \lceil \phi_2 \rceil}(\mathsf{abst}_{\mathfrak{s}}(\phi_2)))\\
\mathsf{abst}_{\mathfrak{s}}(\phi_1 \wedge \phi_2) &\triangleq \mathsf{lift}_{\lceil \phi_1 \rceil \nearrow \lceil \phi_1 \wedge \phi_2 \rceil}(\mathsf{abst}_{\mathfrak{s}}(\phi_1))\\
&\qquad \cap \mathsf{lift}_{\lceil \phi_2 \rceil \nearrow \lceil \phi_1 \wedge \phi_2 \rceil}(\mathsf{abst}_{\mathfrak{s}}(\phi_2))\\
\mathsf{abst}_{\mathfrak{s}}(\phi_1 \vee \phi_2) &\triangleq \mathsf{lift}_{\lceil \phi_1 \rceil \nearrow \lceil \phi_1 \vee \phi_2 \rceil}(\mathsf{abst}_{\mathfrak{s}}(\phi_1))\\
&\qquad \cup \mathsf{lift}_{\lceil \phi_2 \rceil \nearrow \lceil \phi_1 \vee \phi_2 \rceil}(\mathsf{abst}_{\mathfrak{s}}(\phi_2))\\
\mathsf{abst}_{\mathfrak{s}}(\neg \phi_1) &\triangleq \mathbf{AMS}_{\lceil \phi_1 \rceil, \mathfrak{s}} \setminus \mathsf{abst}_{\mathfrak{s}}(\phi_1)
\end{aligned}
$$

Figure 6.4: Computing the abstract memory states of the models of ϕ with stack $\mathfrak{s}$.

CASE $x = y$. If $\mathfrak{s}(x) = \mathfrak{s}(y)$ then $\mathsf{ams}_{\mathfrak{s}}(x = y) = \{\langle V, E, \rho, \gamma\rangle \in \mathbf{AMS} \mid V = \mathsf{classes}(\mathfrak{s})\}$ by Lemma 6.18. Thus, in particular, $\mathsf{ams}_{\mathfrak{s}}(x = y) \supseteq \mathbf{AMS}_{1,\mathfrak{s}} = \mathbf{AMS}_{\lceil x=y \rceil,\mathfrak{s}}$ and the claim follows. If, instead, $\mathfrak{s}(x) \neq \mathfrak{s}(y)$ then $\mathsf{ams}_{\mathfrak{s}}(x = y) \cap \mathbf{AMS}_{\lceil x=y \rceil,\mathfrak{s}} = \emptyset = \mathsf{abst}_{\mathfrak{s}}(x = y)$.

CASE $x \mapsto_{\mathsf{n}} y$, $x \mapsto_{\mathsf{l,r}} \mathbf{y}$. Apply Lemma 6.19 and proceed as for **emp**.

CASE $\mathtt{pred}(x, \mathbf{y})$, $\mathtt{pred} \in \{\mathsf{ls}, \mathsf{ls}_{\geq 2}, \mathtt{tree}, \mathtt{tree}_{\geq 2}\}$. Apply Lemma 6.22 and proceed as for **emp**.

CASE $\phi_1 \star \phi_2$. By the induction hypotheses, we have for $1 \leq i \leq 2$ that $\mathsf{abst}_{\mathfrak{s}}(\phi_i) = \mathsf{ams}_{\mathfrak{s}}(\phi_i) \cap \mathbf{AMS}_{\lceil \phi_i \rceil,\mathfrak{s}}$ Let

$$\mathbf{A}_i \triangleq \mathsf{lift}_{\lceil \phi_1 \rceil \nearrow \lceil \phi_1 \star \phi_2 \rceil}(\mathsf{abst}_{\mathfrak{s}}(\phi_i)).$$

By Theorem 6.38, it follows that $\mathbf{A}_i = \mathsf{ams}_{\mathfrak{s}}(\phi_i) \cap \mathbf{AMS}_{\lceil \phi_1 \star \phi_2 \rceil,\mathfrak{s}}$. By Lemma 6.33, it thus follows that $\mathbf{A}_1 \bullet \mathbf{A}_2$ contains all AMS in $\mathsf{ams}_{\mathfrak{s}}(\phi_1 \star \phi_2)$ that can be obtained by composing AMS with a garbage-chunk count of at most $\lceil \phi_i \star \phi_2 \rceil$. Thus, in particular, (1) $\mathbf{A}_1 \bullet \mathbf{A}_2 \subseteq \mathsf{ams}_{\mathfrak{s}}(\phi_1 \star \phi_2)$ and (2) $\mathbf{A}_1 \bullet \mathbf{A}_2 \supseteq \mathsf{ams}_{\mathfrak{s}}(\phi_1 \star \phi_2) \cap \mathbf{AMS}_{\lceil \phi_i \star \phi_2 \rceil,\mathfrak{s}}$. The claim follows.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Wien Bibliothek Your knowledge hub

CASE $\phi_1 \mathrel{-\!\!\star} \phi_2$. By the induction hypotheses, we have for $1 \leq i \leq 2$ that $\mathsf{abst}_{\mathfrak{s}}(\phi_i) = \mathsf{ams}_{\mathfrak{s}}(\phi_i) \cap \mathbf{AMS}_{\lceil \phi_i \rceil, \mathfrak{s}}$ Let

$$\mathbf{A}_2 \triangleq \mathsf{lift}_{\lceil \phi_2 \rceil \nearrow \lceil \phi_1 + \phi_2 \rceil}(\mathsf{abst}_{\mathfrak{s}}(\phi_2)).$$

By Theorem 6.38 and the definition of $\lceil \phi_1 \star \phi_2 \rceil$, it follows that $\mathbf{A}_2 = \mathsf{ams}_{\mathfrak{s}}(\phi_2) \cap \mathbf{AMS}_{\lceil \phi_1 \star \phi_2 \rceil, \mathfrak{s}}$. Thus, in particular, $\mathbf{A}_2$ contains every AMS in $\mathsf{ams}_{\mathfrak{s}}(\phi_2)$ that can be obtained by composing an AMS in $\mathbf{AMS}_{\lceil \phi_1 \mathrel{-\!\circledast} \phi_2 \rceil} = \mathbf{AMS}_{\lceil \phi_2 \rceil}$ with an AMS from $\mathsf{ams}_{\mathfrak{s}}(\phi_1) \cap \mathbf{AMS}_{\lceil \phi_1 \rceil, \mathfrak{s}}$. It thus follows from Lemma 6.35 that $\mathsf{ams}_{\mathfrak{s}}(\phi_1) \mathrel{-\!\bullet} \mathbf{A}_2$ is precisely the set of AMS $\mathsf{ams}_{\mathfrak{s}}(\phi_1 \mathrel{-\!\circledast} \phi_2) \cap \mathbf{AMS}_{\lceil \phi_1 \mathrel{-\!\circledast} \phi_2 \rceil}$.

CASE $\phi_1 \wedge \phi_2$. By the induction hypotheses, we have for $1 \leq i \leq 2$ that $\mathsf{abst}_{\mathfrak{s}}(\phi_i) = \mathsf{ams}_{\mathfrak{s}}(\phi_i) \cap \mathbf{AMS}_{\lceil \phi_i \rceil, \mathfrak{s}}$. For $1 \leq i \leq 2$, let $\mathbf{A}_i \triangleq \mathsf{lift}_{\lceil \phi_1 \rceil \nearrow \lceil \phi_1 \wedge \phi_2 \rceil}(\mathsf{abst}_{\mathfrak{s}}(\phi_i))$. By Theorem 6.38, we have $\mathbf{A}_i = \mathsf{ams}_{\mathfrak{s}}(\phi_i) \cap \mathbf{AMS}_{\lceil \phi_1 \wedge \phi_2 \rceil, \mathfrak{s}}$. The claim follows from Lemma 6.30.

CASE $\phi_1 \vee \phi_2$. By the induction hypotheses, we have for $1 \leq i \leq 2$ that $\mathsf{abst}_{\mathfrak{s}}(\phi_i) = \mathsf{ams}_{\mathfrak{s}}(\phi_i) \cap \mathbf{AMS}_{\lceil \phi_i \rceil, \mathfrak{s}}$. For $1 \leq i \leq 2$, let $\mathbf{A}_i \triangleq \mathsf{lift}_{\lceil \phi_1 \rceil \nearrow \lceil \phi_1 \vee \phi_2 \rceil}(\mathsf{abst}_{\mathfrak{s}}(\phi_i))$. By Theorem 6.38, we have $\mathbf{A}_i = \mathsf{ams}_{\mathfrak{s}}(\phi_i) \cap \mathbf{AMS}_{\lceil \phi_1 \vee \phi_2 \rceil, \mathfrak{s}}$. The claim follows from Lemma 6.31.

CASE $\neg \phi_1$. By the induction hypothesis, we have that $\mathsf{abst}_{\mathfrak{s}}(\phi_1) = \mathsf{ams}_{\mathfrak{s}}(\phi_1) \cap \mathbf{AMS}_{\lceil \phi_1 \rceil, \mathfrak{s}}$. From Lemma 6.32, it follows that

$$\begin{aligned} & \mathsf{ams}_{\mathfrak{s}}(\neg \phi_1) \cap \mathbf{AMS}_{\lceil \neg \phi_1 \rceil, \mathfrak{s}} \\ = & \mathbf{AMS}_{\lceil \neg \phi_1 \rceil, \mathfrak{s}} \setminus \mathsf{ams}_{\mathfrak{s}}(\phi_1) \\ = & \mathbf{AMS}_{\lceil \neg \phi_1 \rceil, \mathfrak{s}} \setminus (\mathsf{ams}_{\mathfrak{s}}(\phi_1) \cap \mathbf{AMS}_{\lceil \phi_1 \rceil, \mathfrak{s}}). \end{aligned}$$

The claim follows. □

COMPUTABILITY. As all sets that occur in the definition of $\mathsf{abst}_{\mathfrak{s}}(\phi)$ are finite, and the operators $\bullet, \mathrel{-\!\bullet}, \cap, \cup$ and $\setminus$ are all computable, it remains to be shown that we can compute the set of AMS for all atomic formulas. This is trivial for **emp**, (dis-)equalities, and points-to assertions. It remains to be shown that we can compute the sets of abstract lists and abstract trees.

Lemma 6.42. *The sets* $\mathbf{AbstLists}(\mathfrak{s}, x, \mathbf{y})$ *and* $\mathbf{AbstLists}_{\geq 2}(\mathfrak{s}, x, \mathbf{y})$ *can be computed in* PSPACE *in* $|\mathfrak{s}|$.

Proof. We need to enumerate all abstractions of connected list segments from x to y via $\langle x_1, \ldots, x_k \rangle$. First, observe that every such abstraction is of the form $\langle \mathsf{classes}(\mathfrak{s}), E, \emptyset, 0 \rangle$ for some set of hyperedges E. Our goal is to enumerate all possible sets E in polynomial space.

To this end, observe that every connected list segment consists of $k+1$ positive chunks: One chunk with source x and sink x_1, one chunk with source x_i and sink x_{i+1} for all $1 \leq i < k$, and one chunk with source x_k and sink y.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Each of these chunks is abstracted by a single edge of the form $\langle \mathbf{z_1}, \mathsf{n}\rangle \mapsto \langle \mathbf{z_2}, [=1]\rangle$ or $\langle \mathbf{z_1}, \mathsf{n}\rangle \mapsto \langle \mathbf{z_2}, [\geq 2]\rangle$. Here, $\mathbf{z_1}$ and $\mathbf{z_2}$ are the stack-equivalence classes of source and sink.

Since x, y, and all x_i are pairwise different, $k < |\mathfrak{s}|$. Consequently, every abstract list consists of at most $\mathcal{O}(|\mathfrak{s}|)$ many edges. Since each of these edges is of the forms illustrated above, each edge can be encoded by $(\log(|\mathfrak{s}|))$ many bits, so every abstract list can be stored in polynomial space.

Further, it is easy to systematically enumerate (with a polynomial number of additional bits for bookkeeping) all these AMS: We sequentially consider all possible ways to pick $x_1, \ldots, x_k$; and for each such pick, systematically construct the induced edges of the corresponding positive chunks.

To enumerate $\mathbf{AbstLists}_{\geq 2}(\mathfrak{s}, x, \mathbf{y})$, we simply skip all AMS that contain no edge or that contain only a single edge of the form $\langle [x]^{\mathfrak{s}}_{=}, \mathsf{n}\rangle \mapsto \langle [y]^{\mathfrak{s}}_{=}, [=1]\rangle$. □

Lemma 6.43. *The sets* $\mathbf{AbstTrees}(\mathfrak{s}, x, \mathbf{y})$ *and* $\mathbf{AbstTrees}_{\geq 2}(\mathfrak{s}, x, \mathbf{y})$ *can be computed in* PSPACE *in* $|\mathfrak{s}|$.

Proof sketch. We proceed as in Lemma 6.42, except that (1) we need to consider all directed trees with root x and holes $\mathbf{y}$, which can each consist of at most $|\mathfrak{s}|$ many positive chunks and (2) each chunk is abstracted by two hyperedges rather than one hyperedge, as positive chunks of tree segments induce edges for both field l and field r. □

Corollary 6.44 (Computability of $\mathsf{abst}_{\mathfrak{s}}(\phi)$)**.** *Let* $\mathfrak{s}$ *be a (finite) stack. Then* $\mathsf{abst}_{\mathfrak{s}}(\phi)$ *is computable for all formulas* ϕ.

To decide whether ϕ is satisfiable in a stack of size n, we can thus proceed as follows. We pick a set of variables $\mathbf{x} \supseteq \mathsf{fvars}(\phi)$ with $|x| = n$. By Lemma 6.16, there are only finitely many stacks over $\mathbf{x}$ that have different abstractions. For each of these stacks, we can decide the nonemptiness of $\mathsf{ams}_{\mathfrak{s}}$ by Theorem 6.41 and Corollary 6.44. The formula is satisfiable iff $\mathsf{ams}_{\mathfrak{s}}$ is nonempty for any of the stacks.

Theorem 6.45 (Decidability of **SSL** satisfiability checking)**.** *It is decidable whether* ϕ *is satisfiable in a stack of size at most* n.

Proof. Observe that ϕ is satisfiable in a stack of size at most n iff ϕ is satisfiable in a stack $\mathfrak{s}$ with $\mathrm{dom}(\mathfrak{s}) \subseteq \{\mathsf{fvars}(\phi)\} \cup \left\{a_1, \ldots, a_{n-|\mathsf{fvars}(\phi)|}\right\}$, where we assume w.l.o.g. that $a_i \notin \mathsf{fvars}(\phi)$ for all i. Observe that $\mathbf{C} \triangleq \{\mathsf{classes}(\mathfrak{s}) \mid \mathrm{dom}(\mathfrak{s}) \subseteq \mathbf{x}\}$ is finite; and that all stacks $\mathfrak{s}, \mathfrak{s}'$ with $\mathsf{classes}(\mathfrak{s}) = \mathsf{classes}(\mathfrak{s}')$ have the same abstraction by Lemma 6.16. Consequently, we can compute the set $\mathbf{R} \triangleq \{\mathsf{abst}_{\mathfrak{s}}(\phi) \mid \mathrm{dom}(\mathfrak{s}) \subseteq \mathbf{x}\}$ by picking for each element $\mathsf{X} \in \mathbf{C}$ one stack $\mathfrak{s}$ with $\mathsf{classes}(\mathfrak{s}) = \mathsf{X}$ and computing $\mathsf{abst}_{\mathfrak{s}}(\phi)$ for this stack. Moreover, by Corollary 6.44, $\mathsf{abst}_{\mathfrak{s}}(\phi)$ is computable for every such stack. By Theorem 6.41 and Corollary 6.39, ϕ is satisfiable over stack $\mathfrak{s}$ iff $\mathsf{abst}_{\mathfrak{s}}(\phi)$ is nonempty. Putting

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

all this together, we obtain ϕ is satisfiable in stacks of size n if and only if any of finitely many computable sets $\mathsf{abst}_{\mathfrak{s}}(\phi)$ is nonempty. □

As our logic is closed under negation, we can, of course, also decide the entailment problem for SSL formulas.

Corollary 6.46 (Decidability of indexed **SSL** entailments)**.** *It is decidable whether $\phi \models_{\mathbf{x}} \psi$ for all finite sets of variables $\mathbf{x}$ and all formulas $\phi, \psi \in \mathbf{SSL}$.*

Proof. $\phi \models_{\mathbf{x}} \psi$ iff $\phi \land \neg\psi$ is unsatisfiable over stacks of size $|\mathbf{x}|$, which is decidable by Theorem 6.45. □

6.5 COMPLEXITY OF THE SSL SATISFIABILITY PROBLEM

It is easy to see that the algorithm $\mathsf{abst}_{\mathfrak{s}}(\phi)$ runs in exponential time. We conclude this section with a proof that SSL satisfiability and entailment are actually PSPACE-complete.

PSPACE-HARDNESS. An easy reduction from quantified Boolean formulas (QBF) shows that the SSL satisfiability problem is PSPACE-hard. I assume w.l.o.g. that every QBF formula is fully quantified, i.e., every variable is either existentially quantified or universally quantified.

The reduction is presented in Fig. 6.5. For convenience, we make use of **emp**, t and $\circledast$ as defined in Fig. 5.2. We encode positive literals x by $(x \mapsto_{\mathsf{n}} \mathsf{nil}) \star \mathsf{t}$ (the heap contains the pointer $x \mapsto_{\mathsf{n}} \mathsf{nil}$) and negative literals by a septraction $(x \mapsto_{\mathsf{n}} \mathsf{nil}) \circledast \mathsf{t}$ (it is possible to extend the heap with a pointer $x \mapsto_{\mathsf{n}} \mathsf{nil}$, implying that the heap does not allocate x). The magic wand is used to simulate universals (i.e., to enforce that we consider both the case $x \mapsto_{\mathsf{n}} \mathsf{nil}$ and the case **emp**, thus setting x both to true and to false). Analogously, septraction is used to simulate existentials.

As we only encode fully quantified formulas F, every variable occurs on the left-hand side of a magic wand or septraction in the encoding $\mathsf{qbf_to_sl}(F)$. For this reason, we may require that the model of $\mathsf{qbf_to_sl}(F)$ is empty, as we do in $\mathbf{emp} \land \mathsf{aux}(F)$. This restriction is actually necessary for the correctness of the encoding of universal quantifiers. To see this, consider the formula $F \triangleq \forall x. F'$ and let $(\mathfrak{s}, \mathfrak{h})$ be a stack–heap pair with $\mathfrak{s}(x) \in \mathrm{dom}(\mathfrak{h})$. Then $(\mathfrak{s}, \mathfrak{h}) \models (x \mapsto_{\mathsf{n}} \mathsf{nil}) \mathrel{-\!\!\star} \mathsf{aux}(F')$ for all F', so $\mathsf{aux}(F)$ does not correctly encode the universal quantifier. By setting $\mathsf{qbf_to_sl}(F) \triangleq \mathbf{emp} \land \mathsf{aux}(F)$, we can exclude models such as $(\mathfrak{s}, \mathfrak{h})$ that break the encoding of the universal.

Similar reductions from QBF to SL can already be found (for standard, "weak" SL) in [CYO01].

Lemma 6.47 (Hardness of **SSL** satisfiability checking)**.** *The satisfiability problem for* **SSL** *without list predicates, without tree predicates and without* $\mapsto_{\mathsf{tree}}$ *is* PSPACE*-hard.*

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

$$
\begin{array}{lcl}
\mathsf{qbf_to_sl}(F) & \triangleq & \mathbf{emp} \wedge \mathsf{aux}(F) \\
\mathsf{aux}(x) & \triangleq & (x \mapsto_{\mathsf{n}} \mathsf{nil}) \star \mathsf{t} \\
\mathsf{aux}(\neg x) & \triangleq & (x \mapsto_{\mathsf{n}} \mathsf{nil}) \circledast \mathsf{t} \\
\mathsf{aux}(F \wedge G) & \triangleq & \mathsf{aux}(F) \wedge \mathsf{aux}(G) \\
\mathsf{aux}(F \vee G) & \triangleq & \mathsf{aux}(F) \vee \mathsf{aux}(G) \\
\mathsf{aux}(\exists x.\, F) & \triangleq & ((x \mapsto_{\mathsf{n}} \mathsf{nil}) \vee \mathbf{emp}) \circledast \mathsf{aux}(F) \\
\mathsf{aux}(\forall x.\, F) & \triangleq & ((x \mapsto_{\mathsf{n}} \mathsf{nil}) \vee \mathbf{emp}) \mathrel{-\!\!\star} \mathsf{aux}(F)
\end{array}
$$

Figure 6.5: Translation $\mathsf{qbf_to_sl}(F)$ from closed QBF formula F (in negation normal form) to an SSL formula that is satisfiable iff F is true.

Note that this reduction simultaneously proves the PSpace-hardness of SSL *model checking*, i.e., of answering the question whether for a given model $(\mathfrak{s}, \mathfrak{h})$ and given formula ϕ, $(\mathfrak{s}, \mathfrak{h}) \models \phi$ holds.

Lemma 6.48 (Complexity of **SSL** model checking)**.** *The model checking problem for* **SSL** *without list predicates, without tree predicates and without* $\mapsto_{\mathsf{tree}}$ *is* PSpace*-hard.*

Proof. If F is a QBF formula over variables $x_1, \ldots, x_k$, then $\mathsf{qbf_to_sl}(F)$ is satisfiable iff

$$(\{x_i \mapsto l_i \mid 1 \leq i \leq n\}, \emptyset) \models \mathsf{qbf_to_sl}(F). \qquad \square$$

PSpace-membership. For every stack $\mathfrak{s}$ and every bound on the garbage-chunk count of the AMS we consider, it is possible to encode every AMS by a string of polynomial length.

Lemma 6.49 (Size of AMS)**.** *Let* $k \in \mathbb{N}$*, let* $\mathfrak{s}$ *be a stack and* $n := k + |\mathfrak{s}|$*. There exists an injective function* $\mathsf{encode} \colon \mathbf{AMS}_{k,\mathfrak{s}} \to \{0,1\}^*$ *such that*

$$\max \{|\mathsf{encode}(\mathcal{A})| \mid \mathcal{A} \in \mathbf{AMS}_{k,\mathfrak{s}}\} \in \mathcal{O}(n \log(n)).$$

Proof. Let $\mathcal{A} = \langle V, E, \rho, \gamma \rangle \in \mathbf{AMS}_{k,\mathfrak{s}}$. Each of the $|\mathfrak{s}| \leq n$ variables that occur in A can be encoded by a logarithmic number of bits.

Observe that $|\bigcup V| \leq |\mathfrak{s}|$, so V can be encoded by $\mathcal{O}(n \log(n) + n)$ symbols (using a constant-length delimiter between the nodes). Each of the at most $|V|$ edges can be encoded by $\mathcal{O}(\log(n))$ bits, encoding the position of the source and target nodes in the encoding of V by $\mathcal{O}(\log(n))$ bits each and expending another bit to differentiate between $[=1]$ and $[\geq 2]$ edges. ρ can be encoded like V. Since $\gamma \leq k \leq n$, γ can be encoded by at most $\log(n)$ bits. In total, we thus have an encoding of length $\mathcal{O}(n \log(n))$. $\square$

An enumeration-based implementation of the algorithm in Fig. 6.4 (that has to keep in memory at most one AMS per subformula at any point in the computation) therefore runs in polynomial space.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Lemma 6.50. *Let $\phi \in \mathbf{SSL}$ and let $n \in \mathbb{N}$. It is decidable in polynomial space in $|\phi| + n$ whether ϕ is satisfiable in a stack of size at most n.*

Proof. A simple induction on the structure of ϕ shows that it is possible to enumerate the set $\mathsf{abst}_{\mathfrak{s}}(\phi)$ in polynomial space.

For lists and trees, we showed this in Lemmas 6.42 and 6.43. All other base cases are obvious.

The most interesting induction step is $\phi_1 \mathrel{-\!\!\star} \phi_2$. Assume that we can enumerate the sets $\mathsf{abst}_{\mathfrak{s}}(\phi_1) = \{\mathcal{A}_1, \dots, \mathcal{A}_m\}$ and $\mathsf{abst}_{\mathfrak{s}}(\phi_2) = \{\mathcal{B}_1, \dots, \mathcal{B}_n\}$ in polynomial space. Use additional memory to successively enumerate all $\mathcal{A} \in \mathbf{AMS}_{\lceil \phi_1 \mathrel{-\!\!\star} \phi_2 \rceil, \mathfrak{s}}$. This is possible in space $\mathcal{O}(n \log(n))$ by Lemma 6.49. For each such $\mathcal{A}$, we enumerate all pairs of AMS $(\mathcal{A}_i, \mathcal{B}_j)$, $1 \leq i \leq m$, $2 \leq j \leq n$. $\mathcal{A} \in \mathsf{abst}_{\mathfrak{s}}(\phi_1 \mathrel{-\!\!\star} \phi_2)$ iff for all $\mathcal{A}_i$ such that $\mathcal{A} \bullet \mathcal{A}_i \neq \bot$, $\mathcal{B}_j = \mathcal{A} \bullet \mathcal{A}_i$ holds for at least one the pairs $(\mathcal{A}_i, \mathcal{B}_j)$. Thus we only need to store a constant number of additional AMS to enumerate $\mathbf{AMS}_{\lceil \phi_1 \mathrel{-\!\!\circledast} \phi_2 \rceil, \mathfrak{s}}$. Similar results hold for all binary operators and negation. □

The PSPACE-completeness result for **SSL**, stated at the beginning of this chapter as Theorem 6.1, follows by combining Lemmas 6.47 and 6.50.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.

7 DECIDING POSITIVE SSL

In this chapter, I develop a decision procedure for $\mathbf{SSL}^{+}_{\text{data}}$, the positive fragment of SSL with data predicates. The restriction to the positive fragment allows us to bring down the complexity of the decision procedure by exploiting properties such as Lemma 5.13. I conjecture that the AMS abstraction of Chapter 6 can be extended to "full" SSL with, for example, transitive data predicates such as sortedness, but do not address such extensions here.

I show in Section 7.1 that every satisfiable $\mathbf{SSL}^{+}_{\text{data}}$ formula has models of polynomial size. This *small-model property* allows us to solve the satisfiability problem of $\mathbf{SSL}^{+}_{\text{data}}$ with a *guess-and-check procedure*: guess a small model (nondeterministically or by enumeration) and check whether it satisfies the formula. Since the model-checking problem for $\mathbf{SSL}^{+}_{\text{data}}$ is in PTime, this immediately yields an NP decision procedure. This is in contrast to full $\mathbf{SSL}_{\text{data}}$, which also has a small-model property, but a PSpace-complete model-checking problem.

Rather than implementing a custom guess-and-check procedure, I define an SMT encoding that exploits the small-model property in Section 7.2.

This chapter is based to a large extent on joint work with Dejan Jovanović and Georg Weissenbacher [KJW18a]. In particular, I would like to emphasize that the SMT encoding in Section 7.2 is based on the SMT encoding of $\mathbf{SL}^{*}_{\text{data}}$, which was presented in [KJW18a] and was mainly developed by Dejan Jovanović. I've adapted this encoding to $\mathbf{SSL}^{+}_{\text{data}}$, fixed a few small bugs in the encoding, and completely overhauled its correctness proof.

Besides rewriting or extending most proofs, I've also made several other changes compared to [KJW18a]. In particular, (1) I use the standard stack–heap model instead of *heap interpretations*; (2) I've dropped support for "fully negative formulas" (which do not seem particularly useful in the context of **SSL** and would thus be an unnecessary complication); and (3) I've dropped support for per-field allocation to simplify the presentation.

7.1 THE SMALL-MODEL PROPERTY OF SSL

In this section, we take the following steps to prove that satisfiability of $\mathbf{SSL}^{+}_{\text{data}}$ formulas is decidable in NP:

1. We show that every AMS is induced by a polynomial model, which implies that every satisfiable $\mathbf{SSL}^{+}$ formula has a polynomial model.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

2. We show that the minimal models of $\mathbf{SSL}^+_{\mathsf{data}}$ formulas differ from to the minimal models of $\mathbf{SSL}^+$ formulas only by a polynomial number of *witnesses*, i.e., locations that determine the truth or falsity of data predicates.

3. We show that the model-checking problem for $\mathbf{SSL}^+$ is in PTIME.

Together, these results imply that the satisfiability problems of both $\mathbf{SSL}^+$ and $\mathbf{SSL}^+_{\mathsf{data}}$ can be solved by a *guess-and-check* procedure that is in NP.

REALIZABILITY SIZE OF AMS. Every AMS corresponds to a model. More precisely, every AMS is *realizable* in a model of linear size. We call this size bound the *realizability size* of the AMS $\mathcal{A}$. We first define the realizability size of individual positive chunks abstracted by $\mathcal{A}$.

Definition 7.1 (Chunk realizability size). *Let $\mathcal{A} = \langle V, E, \rho, \gamma \rangle$ be an AMS and $v \in \mathrm{dom}_V(E)$.*

1. *If $\mathsf{fields}_E(v) = \{\mathsf{n}\}$, then $\mathsf{rsize}(v) \triangleq 1$ iff $E(v, \mathsf{n}) = \langle \mathbf{w}, [=1] \rangle$ and $\mathsf{rsize}(v) \triangleq 2$ otherwise.*

2. *If $\mathsf{fields}_E(v) = \{\mathsf{l}, \mathsf{r}\}$, let $\mathbf{w}_1$, $\mathbf{w}_2$ and ι be such that $E(v, \mathsf{l}) = \langle \mathbf{w}_1, \iota \rangle$ and $E(v, \mathsf{r}) = \langle \mathbf{w}_2, \iota \rangle$. If $\iota = [=1]$, $\mathsf{rsize}(v) \triangleq 1$. Otherwise, $\mathsf{rsize}(v) \triangleq \max(2, |\mathbf{w}_1| + |\mathbf{w}_2| - 1)$.*

Definition 7.2 (Realizability size). *Let $\mathcal{A} = \langle V, E, \rho, \gamma \rangle$ be an AMS. The* realizability size *of $\mathcal{A}$ is then given by*

$$\mathsf{rsize}(\mathcal{A}) \triangleq \left(\sum_{v \in \mathrm{dom}_V(E)} \mathsf{rsize}(v) \right) + \left(\sum_{r \in \rho} |r| \right) + \gamma.$$

Lemma 7.3. *Let $\mathcal{A} = \langle V, E, \rho, \gamma \rangle \in \mathbf{AMS}$. Then there exists a model $(\mathfrak{s}, \mathfrak{h})$ with $|\mathfrak{h}| = \mathsf{rsize}(\mathcal{A})$ and $\mathsf{ams}(\mathfrak{s}, \mathfrak{h}) = \mathcal{A}$.*

Proof. Let $d \in \mathbf{Data}$ be arbitrary, $\{v_1, \ldots, v_n\} \triangleq V$ and $\mathfrak{s} \triangleq \{x \mapsto \ell_i \mid 1 \leq i \leq n, x \in v_i\}$. In the following we write $\mathfrak{s}(v)$ for the single location corresponding to stack-equivalence class v. We pick $\mathfrak{h}$ as the union of the following chunks $\mathfrak{h}_c$.

- For each $v_i \in \mathrm{dom}_V(E)$, we construct a positive chunk as follows.
  - If $\mathsf{fields}_E(v) = \{\mathsf{n}\}$, let $\langle \langle w \rangle, \iota \rangle \triangleq E(v_i, \mathsf{n})$. If $\iota = [=1]$, we define $\mathfrak{h}_c \triangleq \{\mathfrak{s}(v_i) \mapsto \langle \mathfrak{s}(w), e \rangle\}$. Otherwise, we let ℓ be a fresh location and define $\mathfrak{h}_c \triangleq \{\mathfrak{s}(v_i) \mapsto \ell, \ell \mapsto \langle \mathfrak{s}(w), d \rangle\}$. Note that $|\mathfrak{h}_c| = \mathsf{rsize}(E(v_i))$ and $\mathsf{edges}_{\mathfrak{s}}(\mathfrak{h}_c) = \{E(v_i, \mathsf{n})\}$.
  - If $\mathsf{fields}_E(v) = \{\mathsf{l}, \mathsf{r}\}$, we let

$$\langle \langle w_1, \ldots, w_m \rangle, \iota \rangle \triangleq E(v_i, \mathsf{l}),$$
$$\langle \langle w_{m+1}, \ldots, w_k \rangle, \iota \rangle \triangleq E(v_i, \mathsf{r}).$$

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

If $\iota = (\![=1]\!)$, we know that $m = 1$, $k = 2$ and let $\mathfrak{h}_c \triangleq \{\mathfrak{s}(v_i) \mapsto \langle \mathfrak{s}(w_1), \mathfrak{s}(w_{m+1}), d\rangle\}$. Otherwise, we let $\mathfrak{h}_c$ be a tree segment with root $\mathfrak{s}(v_i)$, sink sequence in the left subtree $\mathfrak{s}(w_1), \ldots, \mathfrak{s}(w_m)$, and sink sequence in the right subtree, $\mathfrak{s}(w_{m+1}), \ldots, \mathfrak{s}(w_k)$. The smallest such tree has $k-1$ inner nodes (including the root).

If $k = 2$, this means we have to introduce some arbitrary inner node to bring the size of the tree to two, thus satisfying the constraint ι.

If $k \geq 3$, we introduce fresh locations ℓ'_2, ℓ'_{k-1} and construct a tree with root $\mathfrak{s}(v_i)$, inner nodes ℓ'_2, ℓ'_{k-1}, and holes $w_1, \ldots, w_k$. Note that $\mathsf{edges}_{\mathfrak{s}}(\mathfrak{h}_c) = \{E(v_i, \mathsf{l}), E(v_i, \mathsf{r})\}$ and $|\mathfrak{h}_c| = \mathsf{rsize}(v_i)$.

- For each set $\{w_1, \ldots, w_m\} \in \rho$, we let ℓ be a fresh location and construct a negative chunk $\mathfrak{h}_c \triangleq \{\mathfrak{s}(w_i) \mapsto \langle \ell, d\rangle\}$. Note that $\mathsf{ams}(\mathfrak{s}, \mathfrak{h}_c) = \langle V, \emptyset, \{\{w_1, \ldots, w_m\}\}, 0\rangle$ Further, observe that one location is allocated for each element of $\bigcup \rho$, yielding a sub-heap of size $\left(\sum_{r\in\rho} |r|\right)$ as desired.
- For $1 \leq i \leq \gamma$, we let ℓ_i be a fresh location and construct a negative chunk $\mathfrak{h}_c \triangleq \{\ell_i \mapsto \langle \ell_i, d\rangle\}$. Thus, exactly γ additional locations are allocated.

The above case analysis reveals that $\mathsf{ams}(\mathfrak{s}, \mathfrak{h}) = \mathcal{A}$ and $|\mathfrak{h}| = \mathsf{rsize}(\mathcal{A})$. □

We can easily bound $\mathsf{rsize}(\mathcal{A})$ by a small polynomial in (1) the size of the stack of the abstracted model and (2) an upper bound on the garbage-chunk count.

Lemma 7.4 (AMS realizability size is polynomial)**.** *Let $\mathcal{A} \in \mathbf{AMS}_{k,\mathfrak{s}}$ be an AMS. Then* $\mathsf{rsize}(\mathcal{A}) \leq |\mathfrak{s}|^2 + 3|\mathfrak{s}| + k$.

Proof. Observe that $|V| \leq |\mathfrak{s}|$ and that for all $v \in V$, $\mathsf{rsize}(v) \leq \max(2, |v|) \leq \max(2, |\mathfrak{s}|)$. Thus:

$$\begin{aligned}
\mathsf{rsize}(\mathcal{A}) &= \left(\sum_{v\in \mathrm{dom}_V(E)} \mathsf{rsize}(v)\right) + \left(\sum_{r\in\rho} |r|\right) + \gamma \\
&\leq |\mathfrak{s}|\,(\max(2, |\mathfrak{s}|)) + |V| + \gamma \\
&\leq \max(2\,|\mathfrak{s}|\,, |\mathfrak{s}|^2) + |\mathfrak{s}| + k \\
&\leq |\mathfrak{s}|^2 + 3\,|\mathfrak{s}| + k.
\end{aligned}$$
□

REALIZABILITY OF $\mathbf{SSL}^+_{\mathsf{data}}$ FORMULAS. We will define a similar notion of realizability size for $\mathbf{SSL}^+_{\mathsf{data}}$ formulas. This size notion will only take into account the size of the heap of the minimal models, because the size of the stack of minimal models of a positive formula ϕ is always given by $|\mathsf{fvars}(\phi)|$.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Lemma 7.5. *Let* $\phi \in \mathbf{SSL}^+_{\mathsf{data}}$ *and let* $(\mathfrak{s}, \mathfrak{h})$ *be a model of* ϕ*. Let* $\mathfrak{s}'$ *be the restriction of* $\mathfrak{s}$ *to* $\mathsf{fvars}(\phi)$*. Then* $(\mathfrak{s}', \mathfrak{h}) \models \phi$*.*

Proof. An immediate consequence of Lemma 5.13. □

Note that Lemma 7.5 *cannot* be generalized to full SSL because of the strong-separation semantics; see Example 6.2.

In light of Lemma 7.5, it makes sense to define the *realizability size* of a formula ϕ simply as the minimal heap size among the models of ϕ.

Definition 7.6 ($\mathbf{SSL}_{\mathsf{data}}$ realizability size)**.** *Let* $\phi \in \mathbf{SSL}_{\mathsf{data}}$*. The* realizability size *of* ϕ *is given by*

$$\mathsf{rsize}(\phi) \triangleq \min \{|\mathfrak{h}| \mid \mathfrak{s} \in \mathbf{Stacks}, \mathfrak{h} \in \mathbf{Heaps}, (\mathfrak{s}, \mathfrak{h}) \models \phi\}.$$

The realizability size of $\mathbf{SSL}^+$ formulas (i.e., formulas without data constraints) is closely connected to the realizability size of AMS, because every model induces an AMS and all models with the same AMS satisfy the same $\mathbf{SSL}^+$ formulas.

Lemma 7.7 (Polynomial realizability of $\mathbf{SSL}^+$ formulas)**.** *Let* $\phi \in \mathbf{SSL}^+$ *with* $\mathsf{fvars}(\phi) = \mathbf{x}$*. If* ϕ *is satisfiable, it holds that* $\mathsf{rsize}(\phi) \leq |\mathbf{x}|^2 + 3|\mathbf{x}|$*.*

Proof. Let $(\mathfrak{s}, \mathfrak{h})$ be an arbitrary model with $(\mathfrak{s}, \mathfrak{h}) \models \phi$. Define $\mathcal{A} \triangleq \mathsf{ams}(\mathfrak{s}, \mathfrak{h})$. Observe that $\mathcal{A} \in \mathbf{AMS}_{0,\mathfrak{s}}$. By Lemma 7.4, there exists a model $(\mathfrak{s}, \mathfrak{h}')$ with $\mathsf{ams}(\mathfrak{s}, \mathfrak{h}') = \mathcal{A}$ and $|\mathfrak{h}'| \leq |\mathbf{x}|^2 + 3|\mathbf{x}|$. Moreover, $(\mathfrak{s}, \mathfrak{h}') \models \phi$ by Theorem 6.28, implying the claim. □

Corollary 7.8 (Small-model property of $\mathbf{SSL}^+$)**.** *Let* $\phi \in \mathbf{SSL}^+$*. Then* ϕ *is satisfiable iff* ϕ *is satisfiable in a model* $(\mathfrak{s}, \mathfrak{h})$ *with* $|\mathfrak{s}| \leq |\phi|$ *and* $|\mathfrak{h}| \in \mathcal{O}(|\phi|^2)$*.*

Proof. As $|\mathsf{fvars}(\phi)| \leq |\phi|$, the claim follows immediately from Lemmas 7.5 and 7.7. □

REALIZABILITY SIZE IN THE PRESENCE OF DATA PREDICATES. We next adapt the small-model property of $\mathbf{SSL}^+$, Corollary 7.8, to $\mathbf{SSL}^+_{\mathsf{data}}$, i.e., to positive formulas with data predicates. In the presence of data predicates it is impossible to bound the realizability size solely based on the size of AMS, because every model of a predicate $\mathsf{ds}(x, \mathbf{y}, \mathcal{P})$ must contain *witnesses* of the existential data predicates in $\mathcal{P}$. Symmetrically, a model of $\mathsf{ds}(x, \mathbf{y}) \wedge \neg\mathsf{ds}(x, \mathbf{y}, \mathcal{P})$ may have to contain witnesses of the *falsity* of a universal data predicate in $\mathcal{P}$.

Example 7.9 (Witnesses)**.** *Let*

$$\phi = \mathsf{ls}(x_1, x_2, [\mathsf{n}\colon \alpha < \beta]^{\forall}) \wedge \neg\mathsf{ls}(x_1, x_2, [\mathsf{n}\colon 2\alpha < \beta]^{\forall}).$$

Every model of ϕ *satisfies the "dual" of* $[\mathsf{n}\colon 2\alpha < \beta]^{\forall}$*, i.e., the existential data predicate* $[\mathsf{n}\colon 2\alpha \geq \beta]^{\exists}$*. Consequently, every model of* ϕ *contains a pair of witnesses for this data predicate. For example, this is the case in the model* $(\mathfrak{s}, \mathfrak{h})$ *in Fig. 7.1. We have that* $(\mathfrak{s}, \mathfrak{h}) \models \phi$*, with the two highlighted locations* ℓ_2, ℓ_3 *witnessing the existential.*

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

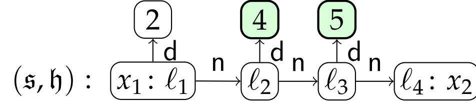


Figure 7.1: A model that contains witnesses for the data predicate $[\mathsf{n}\colon 2\alpha \geq \beta]^{\exists}$.

Intuitively, the size bound for $\mathbf{SSL}^{+}_{\mathsf{data}}$ formulas can thus be obtained from the size bound for $\mathbf{SSL}^{+}$ by adding a bound on the number of witnesses that are necessary to satisfy the formulas.

To formalize the small-model property for $\mathbf{SSL}^{+}_{\mathsf{data}}$, we define a projection function, $\mathsf{dropdata}(\phi)$, from $\mathbf{SSL}^{+}_{\mathsf{data}}$ formulas onto $\mathbf{SSL}^{+}$ formulas without negation:

$$
\begin{aligned}
\mathsf{dropdata}(\mathsf{pred}(x,\mathbf{y},\mathcal{P})) &\triangleq \mathsf{pred}(x,\mathbf{y}),\ \mathsf{pred} \in \{\mathsf{ls}, \mathsf{ls}_{\geq 2}, \mathsf{tree}, \mathsf{tree}_{\geq 2}\}\\
\mathsf{dropdata}(x \mapsto_{\mathsf{ls}} \langle n, d\rangle) &\triangleq x \mapsto_{\mathsf{n}} n\\
\mathsf{dropdata}(x \mapsto_{\mathsf{tree}} \langle l, r, d\rangle) &\triangleq x \mapsto_{\mathsf{l,r}} \langle l, r\rangle\\
\mathsf{dropdata}(F) &\triangleq \mathbf{emp},\ F \in \mathcal{F}_{\mathbf{Data}}\\
\mathsf{dropdata}(\tau) &\triangleq \tau,\ \text{for all other atoms } \tau\\
\mathsf{dropdata}(\phi \wedge \neg\psi) &\triangleq \mathsf{dropdata}(\phi)\\
\mathsf{dropdata}(\phi \times \psi) &\triangleq \mathsf{dropdata}(\phi) \times \mathsf{dropdata}(\psi),\\
&\quad\ \times \in \{\star, \wedge, \vee\}
\end{aligned}
$$

Lemma 7.10. *If* $(\mathfrak{s},\mathfrak{h}) \models \phi$ *then* $(\mathfrak{s},\mathfrak{h}) \models \mathsf{dropdata}(\phi)$.

Proof. A straightforward induction on ϕ, recalling that (1) $(\mathfrak{s},\mathfrak{h}) \models \mathsf{pred}(x,\mathbf{y},\mathcal{P}) \implies (\mathfrak{s},\mathfrak{h}) \models \mathsf{pred}(x,\mathbf{y})$ and (2) $(\mathfrak{s},\mathfrak{h}) \models \phi \wedge \neg\psi \implies (\mathfrak{s},\mathfrak{h}) \models \phi$. □

For technical convenience, we encapsulate the size bound for minimal models of $\mathbf{SSL}^{+}_{\mathsf{data}}$ formulas in $\mathsf{bound}(\phi)$.

Definition 7.11 (Model-size bound)**.** *Let* $\phi \in \mathbf{SSL}^{+}_{\mathsf{data}}$. *The* model-size bound *of* ϕ *is* $\mathsf{bound}(\phi) \triangleq |\phi|^2 + 5\,|\phi|$.

Lemma 7.12 (Small-model property of $\mathbf{SSL}_{\mathsf{data}}$)**.** *Let* $\phi \in \mathbf{SSL}^{+}_{\mathsf{data}}$. *Then* ϕ *is satisfiable iff* ϕ *is satisfiable in a model* $(\mathfrak{s},\mathfrak{h})$ *with* $|\mathfrak{s}| \leq |\phi|$ *and* $|\mathfrak{h}| \leq \mathsf{bound}(\phi)$.

Proof sketch. Let $(\mathfrak{s},\mathfrak{h})$ be an arbitrary model with $(\mathfrak{s},\mathfrak{h}) \models \phi$ and let $\mathcal{A} \triangleq \mathsf{ams}(\mathfrak{s},\mathfrak{h})$. By Lemma 7.5, we can assume that $\mathsf{dom}(\mathfrak{s}) = \mathsf{fvars}(\phi)$ and thus $|\mathfrak{s}| \leq |\phi|$ as desired.

I claim that we can transform $(\mathfrak{s},\mathfrak{h})$ into a model that is isomorphic to the model of size $\mathsf{rsize}(\mathcal{A})$ that we constructed in Lemma 7.3 through *location removal*, as illustrated in Fig. 7.2.

Specifically, we can remove all locations except for:

1. Labeled locations, as this would change $\mathsf{dom}_V(E)$.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

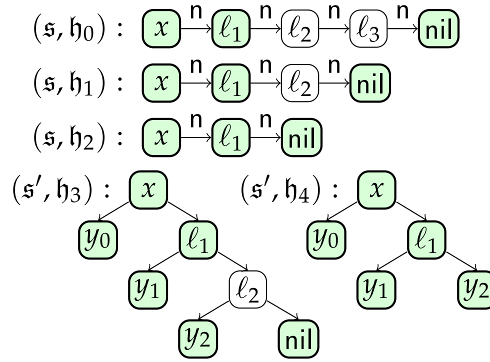


Figure 7.2: Subsequent removal of list locations ℓ_3 and ℓ_2 and tree location ℓ_2, transforming $(\mathfrak{s}, \mathfrak{h}_1)$ via $(\mathfrak{s}, \mathfrak{h}_1)$ into $(\mathfrak{s}, \mathfrak{h}_2)$ and $(\mathfrak{s}', \mathfrak{h}_3)$ into $(\mathfrak{s}', \mathfrak{h}_4)$, without changing the induced AMS.

2. Immediate successors of the head of a list or root of a tree, because removing these could change a $[\geq 2]$ hyperedge into an $[=1]$ hyperedge.

3. Enough inner nodes of a tree to still accommodate all k holes of a tree, i.e., $k - 2$ nodes beside the root.

Such a location removal never invalidates a universal data predicate, i.e., if a universal data predicate holds in a sub-heap of the original model, it continues to hold in the minimized model.

Conversely, location removal that removes all witnesses of an existential data predicate falsifies that data predicate, and may thus result in a model that no longer satisfies ϕ.

Note further that a universal data predicate that occurs under negation corresponds to an existential data predicate and vice-versa.

Consequently, we exempt from the iterative location removal:

1. one location per unary existential data predicate that occurs in ϕ under an even number of negations,

2. two locations per binary existential data predicate that occurs in ϕ under an even number of negations,

3. one location per unary universal data predicate that occurs in ϕ under an odd number of negations, and

4. two locations per binary existential data predicate that occurs in ϕ under an odd number of negations.

It follows that (1) the minimized model still induces the AMS $\mathcal{A}$, (2) the minimized model still satisfies ϕ.

Moreover, Lemma 7.10 implies that $\mathcal{A}$ is an AMS of $\mathsf{dropdata}(\phi)$. Since we retained at most two witnesses per occurrence of a data

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

predicate in ϕ, and the number of data-predicate occurrences is limited by $|\phi|$, Lemma 7.7 yields that the minimized model is of size at most

$$\begin{aligned}
&\mathsf{rsize}(\mathcal{A}) + 2\,|\phi| \\
\leq &\mathsf{rsize}(\mathsf{dropdata}(\phi)) + 2\,|\phi| \\
\leq &|\mathsf{fvars}(\phi)|^2 + 3\,|\mathsf{fvars}(\phi)| + 2\,|\phi| \\
\leq &|\phi|^2 + 5\,|\phi| = \mathsf{bound}(\phi).
\end{aligned}$$

□

Tighter size bounds?

Clearly, the model-size bound $\mathsf{bound}(\phi)$ is not tight for most formulas. For example, the bound overapproximates the required number of witnesses by assuming two witnesses for every occurrence of a data predicate. Moreover, $\mathsf{rsize}(\phi)$ is linear (as opposed to quadratic) in $|\phi|$ if we have a constant bound on the number of holes that occur in a tree segment—which is a very reasonable assumption in practice. In approaches that are based on the size bound (as is the case for the SMT encoding defined in Section 7.2), it is worthwhile to compute tighter bounds, as this may significantly improve the performance of an implementation in practice.

The final step to obtaining an NP decision procedure is a polynomial-time algorithm for model checking. Our model-checking algorithm will exploit that **SSL** predicates are *precise* (see e.g. [COY07]), i.e., that they satisfy the following *unique footprint property*.

Lemma 7.13 (Unique footprint property)**.** *Let* $(\mathfrak{s}, \mathfrak{h})$ *be a model such that* $(\mathfrak{s}, \mathfrak{h}) \models \mathsf{pred}(x, \mathbf{y}) \star \mathsf{t}$ *for* $\mathsf{pred} \in \{\mathsf{ls}, \mathsf{ls}_{\geq 2}, \mathsf{tree}, \mathsf{tree}_{\geq 2}\}$*. Then there exists a unique heap* $\mathfrak{h}'$ *such that* $\mathfrak{h}' \subseteq \mathfrak{h}$ *and* $(\mathfrak{s}, \mathfrak{h}') \models \mathsf{pred}(x, \mathbf{y})$*.*

Proof. For $\mathsf{pred} = \mathsf{ls}$ and $\mathsf{pred} = \mathsf{ls}_{\geq 2}$, the result holds trivially. We prove the claim for tree; the proof for $\mathsf{tree}_{\geq 2}$ is completely analogous.

By the semantics of $\star$, there exists at least one such $\mathfrak{h}'$. We show that this $\mathfrak{h}'$ is unique by systematically constructing all candidates for $\mathfrak{h}'$.

Let $\mathfrak{h}_1, \ldots, \mathfrak{h}_k$ be the positive chunks of $\mathfrak{h}$. Assume $\mathbf{y} = \langle y_1, \ldots, y_n \rangle$. It suffices to show that there is a unique subset of these chunks whose union satisfies $\mathsf{tree}(x, \mathbf{y})$.

W.l.o.g., $\mathfrak{h}_1$ allocates x. Every sub-heap $\mathfrak{h}'$ with $(\mathfrak{s}, \mathfrak{h}') \models \mathsf{tree}(x, \mathbf{y})$ must contain $\mathfrak{h}_1$, so we make $\mathfrak{h}_1$ the current candidate for $\mathfrak{h}'$. Let $\mathbf{z} = \langle z_1, \ldots, z_m \rangle$ be the sink sequence of $\mathfrak{h}_1$. If $\mathbf{z} = \mathbf{y}$, we are done: even though it might be possible to add other $\mathfrak{h}_i$ to obtain the same sink sequence $\mathbf{y}$, this would only be possible by allocating at least one of the $\mathbf{y}$, which contradicts the semantics of the tree predicate.

Otherwise, let i be the first index with $[z_i]^{\mathfrak{s}}_{=} \neq [y_i]^{\mathfrak{s}}_{=}$. Let $\mathfrak{h}_j$ be the chunk that allocates z_i. Every sub-heap $\mathfrak{h}'$ with $(\mathfrak{s}, \mathfrak{h}') \models \mathsf{tree}(x, \mathbf{y})$ must contain $\mathfrak{h}_j$. We therefore add $\mathfrak{h}_j$ to the current candidate for $\mathfrak{h}'$

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

This yields a larger model with root x and new sink sequence $\langle z'_1, \ldots, z'_{m'} \rangle$. If $\langle z'_1, \ldots, z'_{m'} \rangle = \mathbf{y}$, we are done. Otherwise, we iterate the previous step. Since there are only k positive chunks, this process terminates. Since $(\mathfrak{s}, \mathfrak{h}) \models \mathsf{tree}(x, \mathbf{y}) \star \mathsf{t}$, we must arrive at a model with sink sequence $\mathbf{y}$.

Moreover, at no step did we have a choice: all the chunks we added to the candidate for $\mathfrak{h}'$ *had* to be added to this candidate to arrive at a model of $\mathsf{tree}(x, \mathbf{y})$. □

Lemma 7.14 (Polynomial-time model checking for $\mathbf{SSL}^{+}_{\mathsf{data}}$)**.** *Assume that $\mathcal{T}_{\mathbf{Data}}$ is decidable in* NP. *Let $(\mathfrak{s}, \mathfrak{h})$ be a model and $\phi \in \mathbf{SSL}^{+}_{\mathsf{data}}$. It is decidable in* PTime *(in $|(\mathfrak{s}, \mathfrak{h})|$ as well as $|\phi|$) whether $(\mathfrak{s}, \mathfrak{h}) \models \phi$ holds.*

Proof sketch. Since ϕ is a Boolean combination of symbolic heaps, we need to check whether $(\mathfrak{s}, \mathfrak{h}) \models \psi$ for each symbolic heap ψ in ϕ. Take such a symbolic heap $\psi = \tau_1 \star \cdots \star \tau_k$.

Because of the unique footprint property of **SSL** (Lemma 7.13), there is at most one way to split $\mathfrak{h}$ into $\mathfrak{h}_1 \uplus^{\mathfrak{s}} \cdots \uplus^{\mathfrak{s}} \mathfrak{h}_k$ such that for each i, $(\mathfrak{s}, \mathfrak{h}_i) \models \tau_i$. We can discover such a split in a greedy way: $\mathfrak{h}_i$ must consist of the root of τ_i and everything reachable from this root without going through the holes of τ_i.

If no such split exists, ψ is unsatisfiable. Otherwise, we must simply check for each $\mathfrak{h}_i$ whether $(\mathfrak{s}, \mathfrak{h}_i) \models \tau_i$ holds. This can clearly be done in polynomial time, as it is possible to check whether $\mathfrak{h}_i$ is a list or a tree in polynomial time and it is possible to evaluate data predicates on $\mathfrak{h}_i$ in polynomial time. □

Crucially, Lemma 7.14 holds only for $\mathbf{SSL}^{+}_{\mathsf{data}}$, not for the full logic $\mathbf{SSL}_{\mathsf{data}}$, whose model-checking problem we proved to be PSpace-hard in Lemma 6.48.

Theorem 7.15. *Assume that $\mathcal{T}_{\mathbf{Data}}$ is decidable in* NP. *Then the satisfiability problem of $\mathbf{SSL}^{+}_{\mathsf{data}}$ is decidable in* NP.

Proof. Let $\phi \in \mathbf{SSL}^{+}_{\mathsf{data}}$. By Lemma 7.12, ϕ is satisfiable iff it is satisfied by a model $(\mathfrak{s}, \mathfrak{h})$ with $\mathsf{dom}(\mathfrak{s}) = \mathsf{fvars}(\phi)$ and $|\mathfrak{h}| \leq \mathsf{bound}(\phi) \in \mathcal{O}(|\phi|^2)$. Consequently, we guess a stack–heap pair within these (polynomial) size bounds and then check (by Lemma 7.14) in deterministic polynomial time in the size of $(\mathfrak{s}, \mathfrak{h})$ whether $(\mathfrak{s}, \mathfrak{h}) \models \phi$ holds. We thus obtain a guess-and-check procedure that runs in NP. □

In principle, we could use the guess-and-check procedure from Theorem 7.15 to implement the oracle decide^{+} and thus conclude the chapter at this point. Towards a more practical implementation, I will instead propose an SMT encoding of $\mathbf{SSL}^{+}_{\mathsf{data}}$ formulas based on the small-model property.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

7.2 ENCODING POSITIVE FORMULAS INTO SMT

In this section, I assume that $\mathcal{T}_{\mathbf{Data}}$ is stably infinite. Under this assumption, we can decide $\mathbf{SSL}^{+}_{\mathsf{data}}$ formulas by encoding them into SMT formulas that can be discharged by off-the-shelf SMT solvers.

Specifically, I show how to model the spatial constraints of $\mathbf{SSL}^{+}_{\mathsf{data}}$ formulas in the theory of arrays, thus obtaining an SMT formula over the combination of $\mathcal{T}_{\mathbf{Data}}$ and the theory of arrays that is satisfiable if and only if ϕ is satisfiable. This encoding can be performed in polynomial time (and space).

Since $\mathbf{SSL}^{+}_{\mathsf{data}}$ is decidable in NP if $\mathcal{T}_{\mathbf{Data}}$ is decidable in NP, as we just saw in Theorem 7.15, an encoding into SAT (i.e., the satisfiability problem of propositional logic) rather than SMT would also be conceivable, but by encoding into SMT, we can make use of the highly optimized decision procedures that are available in state-of-the-art SMT solvers for many common choices of $\mathcal{T}_{\mathbf{Data}}$.

THE THEORY OF ARRAYS. Our approach relies on the theory of arrays extended with combinators that can express constant arrays and express point-wise array operations [MB09]. We denote this theory by $\mathcal{T}_{\mathsf{array}}$.

The basic theory of arrays defines functions store and $\cdot[\cdot]$ that satisfy the following axioms [McC62; BM07].

$$\begin{aligned}
&\forall i, j.\, i = j \implies \mathbf{a}[i] = \mathbf{a}[j] \\
&\forall \mathbf{a}, i, j, v.\, i = j \implies \mathsf{store}(\mathbf{a}, i, v)[j] = v \\
&\forall \mathbf{a}, i, j, v.\, i \neq j \implies \mathsf{store}(\mathbf{a}, i, v)[j] = \mathbf{a}[j]
\end{aligned}$$

The generalized theory $\mathcal{T}_{\mathsf{array}}$ adds a constant combinator $\mathbf{K}$ and a map combinator map such that

$$\begin{aligned}
&\forall c, i.\, \mathbf{K}(c)[i] = c \\
&\forall n, f, \mathbf{a}_1, \ldots, \mathbf{a}_n, i.\, \mathsf{map}_f(\mathbf{a}_1, \ldots, \mathbf{a}_n)[i] = f(\mathbf{a}_1[i], \ldots, \mathbf{a}_n[i]),
\end{aligned}$$

where c ranges over constants and f over functions of arity n.

We write array s_1 s_2 for the sort of arrays with indices of sort s_1 and elements of sort s_2.

With $\mathbf{K}$ and `map`, it is possible to express universal statements about array elements without relying on quantifiers. Moreover, the satisfiability of generalized array formulas is decidable in NP with effective decision procedures implemented in popular SMT solvers such as `Z3` [MB08] and `Boolector` [PNB13].

The array combinators in $\mathcal{T}_{\mathsf{array}}$ are powerful enough to express basic set-theoretic operations. For example, we can view a set of locations $\mathbf{l}$

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

as an array $\mathbf{a_l}$ mapping **Loc** to **Bool** $= \{\mathsf{t}, \mathsf{f}\}$, where $\mathbf{a_l}[l] = \mathsf{t}$ iff $l \in \mathbf{l}$. Taking this view, set operations can be defined as follows.

$$\{x\} \triangleq \mathsf{store}(\mathbf{K}(\mathsf{f}), x, \mathsf{t}) \qquad x \in \mathbf{x} \triangleq \mathbf{x}[x]$$
$$\mathbf{x} = \emptyset \triangleq (\mathbf{x} = \mathbf{K}(\mathsf{f})) \qquad \mathbf{x} \subseteq \mathbf{y} \triangleq \mathsf{map}_{\Rightarrow}(\mathbf{x}, \mathbf{y}) = \mathbf{K}(\mathsf{t})$$
$$\mathbf{x} \cup \mathbf{y} \triangleq \mathsf{map}_{\vee}(\mathbf{x}, \mathbf{y}) \qquad \mathbf{x} \cap \mathbf{y} \triangleq \mathsf{map}_{\wedge}(\mathbf{x}, \mathbf{y})$$

In the following, I use the set notation as a shorthand for the equivalent array encoding. I denote array variables that represent sets by boldface letters (e.g., $\mathbf{x}$), and vectors of array variables by uppercase boldface letters (e.g., $\mathbf{X} = \langle \mathbf{x}_1, \ldots \mathbf{x}_n \rangle$). To ease notation, I overload predicates over sets to predicates over vectors of sets in a point-wise manner. For example, $\mathbf{X} = \emptyset$ represents $\bigwedge \mathbf{x}_i = \emptyset$, and $\mathbf{X} = \mathbf{Y} \cup \mathbf{Z}$ for $\bigwedge \mathbf{x}_i = \mathbf{y}_i \cup \mathbf{z}_i$.

FROM STACK–HEAP PAIRS TO FIRST-ORDER MODELS. To each stack–heap model $(\mathfrak{s}, \mathfrak{h})$, with $|\mathfrak{h}| = n$, we associate an equivalent first-order model $\mathcal{M}_{\text{SMT}}$ in the theory combination $\mathcal{T}_{\mathbf{Array}} \oplus \mathcal{T}_{\mathbf{Data}}$ as follows. $\mathcal{M}_{\text{SMT}}$ uses the same interpretations of the sorts **Loc** and **Data** as the stack–heap model.[1] For each field $\mathsf{f} \in \{\mathsf{n}, \mathsf{l}, \mathsf{r}, \mathsf{d}\}$, $\mathcal{M}_{\text{SMT}}$ contains an array of sort array **Loc** $\mathsf{sort}(\mathsf{f})$ The interpretation $\mathsf{f}^{\mathcal{M}_{\text{SMT}}}$ of a field f is an array mapping each $\ell \in \mathrm{dom}(\mathfrak{h})$ to $\mathsf{f}(\ell)$,if $\mathsf{f}(\ell) \neq \bot$, and to an arbitrary well-sorted value otherwise.

For each data structure $\mathsf{ds} \in \{\mathsf{ls}, \mathsf{tree}\}$, $\mathcal{M}_{\text{SMT}}$ also contains a dedicated set variable $\mathbf{x}_{\mathsf{ds}}$. The interpretation $\mathbf{x}_{\mathsf{ls}}^{\mathcal{M}_{\text{SMT}}}$ is an array representing the set $\mathsf{loc}_{\mathsf{ls}}(\mathfrak{h})$. Analogously, $\mathbf{x}_{\mathsf{tree}}^{\mathcal{M}_{\text{SMT}}}$ is an array representing the set $\mathsf{loc}_{\mathsf{tree}}(\mathfrak{h})$.

The interpretation $\mathcal{M}_{\text{SMT}}$ also includes n dedicated location variables $x_1, \ldots, x_n$, and a set of locations $\mathbf{x}_{\mathsf{global}}$ interpreted so that (1) $\mathbf{x}_{\mathsf{global}} = \mathbf{x}_{\mathsf{ls}} \cup \mathbf{x}_{\mathsf{tree}}$ and (2) $\mathbf{x}_{\mathsf{global}} \subseteq \{x_1, \ldots, x_n\}$ holds. The variables $\mathrm{dom}(\mathfrak{s})$ and nil are interpreted in $\mathcal{M}_{\text{SMT}}$ as they are in $(\mathfrak{s}, \mathfrak{h})$.

The following SMT formula $\Delta^n_{\mathbf{SSL}}$ defines positive stack–heap models of size at most n.

$$\Delta^n_{\mathbf{SSL}} \triangleq (\mathbf{x}_{\mathsf{global}} = \bigcup_{\mathsf{ds} \in \mathcal{D}} \mathbf{x}_{\mathsf{ds}}) \wedge (\mathbf{x}_{\mathsf{global}} \subseteq \{x_1, \ldots, x_n\})$$
$$\wedge\, (\mathsf{nil} \notin \mathbf{x}_{\mathsf{global}}) \wedge (\mathbf{x}_{\mathsf{ls}} \cap \mathbf{x}_{\mathsf{tree}} = \emptyset)$$

Formula $\Delta^n_{\mathbf{SSL}}$ makes sure that every allocated location is either a list location or a tree location, that the allocated heap size is at most n, that nil is not allocated, and that no variable is treated as both a list and a tree location.

To guide the development of the SMT translation, we formalize the *model correspondence* between stack–heap models and the SMT models

[1] This somewhat limits the interpretation of **Loc** in practice, because the theory of arrays need not work for indexes of arbitrary sort. It usually makes sense to interpret **Loc** by the set of integers. This allows us to translate the heap to integer-indexed arrays.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

described above. The idea is to define a translation from an $\mathbf{SSL}^+$ formula ϕ to an SMT formula F such that $(\mathfrak{s},\mathfrak{h}) \models \phi$ if and only if the SMT models $\mathcal{M}_{\text{SMT}}$ that correspond to $(\mathfrak{s},\mathfrak{h})$ satisfy F.

Definition 7.16 (Model correspondence)**.** *Let $(\mathfrak{s},\mathfrak{h})$ be a stack–heap pair with $|\mathfrak{h}| = n$ and $\mathcal{M}_{\text{SMT}}$ an SMT model such that $\mathcal{M}_{\text{SMT}} \models \Delta^n_{\mathbf{SSL}}$. $\mathcal{M}_{\text{SMT}}$* corresponds to *$(\mathfrak{s},\mathfrak{h})$ iff all of the following hold.*

1. $\mathbf{x}_{\mathsf{global}}^{\mathcal{M}_{\text{SMT}}} = \mathrm{dom}(\mathfrak{h})$,
2. $y^{\mathcal{M}_{\text{SMT}}} = \mathfrak{s}(y)$ *for all* $y \in \mathrm{dom}(\mathfrak{h})$,
3. $\mathsf{f}^{\mathcal{M}_{\text{SMT}}}[\ell] = \mathsf{f}(\ell)$ *for all fields* f *and all* $\ell \in \mathrm{dom}(\mathfrak{h})$ *with* $\mathsf{f}(\ell) \neq \bot$.

Informally, an SMT model $\mathcal{M}_{\text{SMT}}$ corresponds to a stack–heap model $(\mathfrak{s},\mathfrak{h})$ if $\mathcal{M}_{\text{SMT}}$ interprets both the stack variables and all heap locations in the same way as $(\mathfrak{s},\mathfrak{h})$.

Lemma 7.17. *For every stack–heap model $(\mathfrak{s},\mathfrak{h})$ with $|\mathfrak{h}| = n$, there exists an SMT model $\mathcal{M}_{\text{SMT}}$ such that $\mathcal{M}_{\text{SMT}} \models \Delta^n_{\mathbf{SSL}}$ and $\mathcal{M}_{\text{SMT}}$ corresponds to $(\mathfrak{s},\mathfrak{h})$. Likewise, for every SMT model $\mathcal{M}_{\text{SMT}}$ with $\mathcal{M}_{\text{SMT}} \models \Delta^n_{\mathbf{SSL}}$, there exists a stack–heap model $(\mathfrak{s},\mathfrak{h})$ with $|\mathfrak{h}| = n$ such that $\mathcal{M}_{\text{SMT}}$ corresponds to $(\mathfrak{s},\mathfrak{h})$.*

Proof. By construction. □

Note that there is no unique SMT model that corresponds to a stack–heap model $(\mathfrak{s},\mathfrak{h})$, as the interpretation of each f-array can be arbitrary outside of $\mathsf{loc}_{\mathsf{ls}}(\mathfrak{h})$ (for $\mathsf{f} \in \mathcal{F}^{\mathbf{Data}}_{\mathsf{ls}}$) or $\mathsf{loc}_{\mathsf{tree}}(\mathfrak{h})$ (for $\mathsf{f} \in \mathcal{F}^{\mathbf{Data}}_{\mathsf{tree}}$).

SMT TRANSLATION. The encoding function T_n that translates basic $\mathbf{SSL}^+_{\mathsf{data}}$ formulas to SMT is shown in Fig. 7.3. We start without inductive predicates, following the approach from [PWZ13]. The function T_n takes an $\mathbf{SSL}^+_{\mathsf{data}}$ formula ϕ and translates ϕ into an SMT formula $F = \mathsf{T}_n(\phi)$ such that F is satisfiable if and only if ϕ is satisfiable in a model of size at most n.

The translation relies on two auxiliary functions: $\mathsf{T}^b_n(\phi)$, which translates the Boolean structure of ϕ recursively; and $\mathsf{T}^s_n(\phi, \mathbf{Y})$, which translates spatial formulas. Here, $\mathbf{Y}$, contains the footprint variables $\mathbf{y}_{\mathsf{ls}}$ and $\mathbf{y}_{\mathsf{tree}}$ that the translation will define. Both functions return a triple $\langle A, B, Z\rangle$, where A and B together define the semantics of ϕ and Z is the set of all fresh variables introduced (recursively) by the translation. The encoding is straightforward, with the exception of negation. Let $\mathsf{T}^b_n(\phi) = \langle A, B, Z\rangle$. In order for our encoding to be correct, we will prove that it has the following three properties.

CORRECTNESS: If $\mathcal{M}_{\text{SMT}}$ corresponds to $(\mathfrak{s},\mathfrak{h})$ then $(\mathfrak{s},\mathfrak{h}) \models \phi$ iff $\mathcal{M}_{\text{SMT}} \models \exists Z.\, A \wedge B$;

Z-EXISTENCE: $\exists Z.\, B$ is valid; and

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

$$
\begin{array}{lcl}
\mathsf{T}^s_n(F_{\mathbf{Data}}, \mathbf{Y}) & \triangleq & \langle F_{\mathbf{Data}}, (\mathbf{Y} = \emptyset), \emptyset\rangle \\
\mathsf{T}^s_n(x \mapsto_{\mathsf{n}} \langle y \rangle, \mathbf{Y}) & \triangleq & \langle(\mathsf{n}[x] = y), \\
 & & \quad (\mathbf{y}_{\mathsf{ls}} = \{x\} \wedge \mathbf{y}_{\mathsf{tree}} = \emptyset), \emptyset\rangle \\
\mathsf{T}^s_n(x \mapsto_{\mathsf{l,r}} \langle y_1, y_2 \rangle, \mathbf{Y}) & \triangleq & \langle(\mathsf{l}[x] = y_1 \wedge \mathsf{r}[x] = y_2), \\
 & & \quad (\mathbf{y}_{\mathsf{ls}} = \emptyset \wedge \mathbf{y}_{\mathsf{tree}} = \{x\}), \emptyset\rangle \\
\mathsf{T}^s_n(x \mapsto_{\mathsf{ls}} \langle y, d \rangle, \mathbf{Y}) & \triangleq & \langle(\mathsf{n}[x] = y \wedge \mathsf{d}[x] = d), \\
 & & \quad (\mathbf{y}_{\mathsf{ls}} = \{x\} \wedge \mathbf{y}_{\mathsf{tree}} = \emptyset), \emptyset\rangle \\
\mathsf{T}^s_n(x \mapsto_{\mathsf{tree}} \langle y_1, y_2, d \rangle, \mathbf{Y}) & \triangleq & \langle(\mathsf{l}[x] = y_1 \wedge \mathsf{r}[x] = y_2 \wedge \mathsf{d}[x] = d), \\
 & & \quad (\mathbf{y}_{\mathsf{ls}} = \emptyset \wedge \mathbf{y}_{\mathsf{tree}} = \{x\}), \emptyset\rangle \\
\mathsf{T}^s_n(\phi_1 \star \phi_2, \mathbf{Y}) & \triangleq & \text{let } \mathbf{Y}_1, \mathbf{Y}_2 \text{ be fresh} \\
 & & \quad \langle A_1, B_1, Z_1\rangle = \mathsf{T}^s_n(\phi_1, \mathbf{Y}_1), \\
 & & \quad \langle A_2, B_2, Z_2\rangle = \mathsf{T}^s_n(\phi_2, \mathbf{Y}_2) \\
 & & \quad Z = Z_1 \cup Z_2 \cup \mathbf{Y}_1 \cup \mathbf{Y}_2 \\
 & & \text{in } \langle A_1 \wedge A_2 \wedge \mathbf{Y}_1 \cap \mathbf{Y}_2 = \emptyset, \\
 & & \quad B_1 \wedge B_2 \wedge \mathbf{Y} = \mathbf{Y}_1 \cup \mathbf{Y}_2, Z\rangle \\
\mathsf{T}^b_n(\phi) & \triangleq & \text{let } \mathbf{Y} \text{ be fresh, } \langle A, B, Z\rangle = \mathsf{T}^s_n(\phi, \mathbf{Y}) \\
 & & \text{in } \langle A \wedge \mathbf{X} = \mathbf{Y}, B, Z \cup \mathbf{Y}\rangle \\
\mathsf{T}^b_n(\neg\phi) & \triangleq & \text{let } \langle A, B, Z\rangle = \mathsf{T}^b_n(\phi) \text{ in } \langle \neg A, B, Z\rangle \\
\mathsf{T}^b_n(\phi_1 \wedge \phi_2) & \triangleq & \text{let } \langle A_1, B_1, Z_1\rangle = \mathsf{T}^b_n(\phi_1), \\
 & & \quad \langle A_2, B_2, Z_2\rangle = \mathsf{T}^b_n(\phi_2) \\
 & & \text{in } \langle A_1 \wedge A_2, B_1 \wedge B_2, Z_1 \cup Z_2\rangle \\
\mathsf{T}^b_n(\phi_1 \vee \phi_2) & \triangleq & \text{let } \langle A_1, B_1, Z_1\rangle = \mathsf{T}^b_n(\phi_1), \\
 & & \quad \langle A_2, B_2, Z_2\rangle = \mathsf{T}^b_n(\phi_2) \text{ in} \\
 & & \langle A_1 \vee A_2, B_1 \wedge B_2, Z_1 \cup Z_2\rangle \\
\mathsf{T}_n(\phi) & \triangleq & \text{let } \mathsf{T}^b_n(\phi) = \langle A, B, Z\rangle \text{ in } A \wedge B \wedge \Delta^n_{\mathbf{SSL}}
\end{array}
$$

Figure 7.3: SMT encoding for the core fragment of $\mathbf{SSL}^+$ without inductive predicates. We assume arrays $\mathbf{X}$ that represent the global footprint of the model.

Z-EQUIVALENCE: $B(Z_1) \wedge B(Z_2) \implies A(Z_1) = A(Z_2)$ is valid.

The correctness property ensures that the encoding correctly encodes the $\mathbf{SSL}^+_{\mathsf{data}}$ semantics: ϕ is true in a stack–heap model of size at most n iff it is true in the corresponding SMT model. Z-Existence and Z-Equivalence make sure that the encoding can accommodate negation: the B part of the translation is a "definition" of the fresh variables Z: the variables Z can be assigned in each model $\mathcal{M}_{\mathrm{SMT}}$ in such a way that B is satisfied; and if there are multiple ways to assign Z that make B true, then either both assignments or neither assignment make A true, i.e., the A part cannot distinguish between assignments to Z.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

These properties allow us to ensure correctness of the translation of negation $\neg\phi$.

Lemma 7.18 (Encoding negation). *Assume* $T_n^b(\phi) = \langle A, B, Z\rangle$ *and* $\mathcal{M}_{\text{SMT}}$ *corresponds to* $(\mathfrak{s}, \mathfrak{h})$. *Then* $(\mathfrak{s}, \mathfrak{h}) \models \neg\phi$ *iff* $\mathcal{M}_{\text{SMT}} \models \exists Z.\, B \wedge \neg A$.

Proof.

$$\begin{array}{llll} (\mathfrak{s}, \mathfrak{h}) \models \neg\phi & \text{iff} & (\mathfrak{s}, \mathfrak{h}) \not\models \phi & \text{iff} \\ \mathcal{M}_{\text{SMT}} \not\models \exists Z.\, A \wedge B & \text{iff} & \mathcal{M}_{\text{SMT}} \models \neg\exists Z.\, A \wedge B & \text{iff} \\ \mathcal{M}_{\text{SMT}} \models \forall Z.\, B \Rightarrow \neg A & \text{iff} & \mathcal{M}_{\text{SMT}} \models \exists Z.\, B \wedge \neg A \ , & \end{array}$$

where the last equivalence follows from Z-existence and Z-equivalence as follows:

($\Rightarrow$) Assume $\mathcal{M}_{\text{SMT}} \models \forall Z.\, B \Rightarrow \neg A$. Because of Z-existence, we then have $\mathcal{M}_{\text{SMT}} \models (\forall Z.\, B \Rightarrow \neg A) \wedge \exists Z.\, B$ and thus $\mathcal{M}_{\text{SMT}} \models \exists Z.\, B \wedge \neg A$.

($\Leftarrow$) Assume $\mathcal{M}_{\text{SMT}} \models \exists Z.\, B \wedge \neg A$. Assume towards a contradiction that $\mathcal{M}_{\text{SMT}} \models \exists Z.\, B \wedge A$. We can remove the quantifier prefix by adding an appropriate number of constants to $\mathcal{M}_{\text{SMT}}$. Specifically, there exists a model $\mathcal{M}'_{\text{SMT}}$ that is like $\mathcal{M}_{\text{SMT}}$, except that $\mathcal{M}'_{\text{SMT}}$ is defined on additional constants Z_1 and Z_2 such that both $\mathcal{M}'_{\text{SMT}} \models B(Z_1) \wedge \neg A(Z_1)$ and $\mathcal{M}'_{\text{SMT}} \models B(Z_2) \wedge A(Z_2)$ hold. In this case, we have $\mathcal{M}'_{\text{SMT}} \models B(Z_1) \wedge B(Z_2) \wedge A(Z_1) \neq A(Z_2)$. This contradicts Z-equivalence. □

LISTS AND TREES. To translate a predicate $\mathsf{T}_n^s(\mathsf{ds}(x, \mathbf{s}, \mathcal{P}), \mathbf{Y})$, with $\mathbf{s} = \langle s_1, \ldots, s_k\rangle$, for models of size at most n, we introduce fresh binary predicates $r_1^{\mathbf{z}}, \ldots, r_n^{\mathbf{z}}$, and a fresh set of locations $\mathbf{z}$. These fresh predicates are meant to represent reachability in up to n steps within the set $\mathbf{z}$. Location set $\mathbf{z}$ will represent all nodes reachable from the source of the data structure, x, and allocated within the sub-heap that is the model of $\mathsf{ds}(x, \mathbf{s}, \mathcal{P})$.

Throughout the remainder of the section, we assume that ds, x, $\mathbf{s}$, $\mathcal{P}$, $\mathbf{Y}$, $\mathbf{z}$, $r_i^{\mathbf{z}}$ and n are fixed and in scope of all definitions. We also assume sets of fields $\mathcal{F}_{\mathsf{ds}}$ and $\mathcal{F}_{\mathsf{ds}}^{\mathbf{Data}}$, defined as $\mathcal{F}_{\mathsf{ls}} = \{\mathsf{n}\}$ and $\mathcal{F}_{\mathsf{ls}}^{\mathbf{Data}} = \{\mathsf{n}, \mathsf{d}\}$, or $\mathcal{F}_{\mathsf{tree}} = \{\mathsf{l}, \mathsf{r}\}$ and $\mathcal{F}_{\mathsf{tree}}^{\mathbf{Data}} = \{\mathsf{l}, \mathsf{r}, \mathsf{d}\}$.

To improve readability of the translation function T_n^s for inductive data structures, we define several auxiliary formulas. To start with, we define the following functions for convenience.

$$\mathsf{isleaf}(y) \triangleq y = \mathsf{nil} \vee \bigvee_{s \in \mathbf{s}} y = s$$

$$\mathsf{succ}(y_1, y_2) \triangleq \bigvee_{\mathsf{f} \in \mathcal{F}_{\mathsf{ds}}} \mathsf{f}[y_1] = y_2$$

These formulas capture whether x is a leaf (a hole or null) and whether y is a direct successor of x.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

As the variable $\mathbf{z}$ represents the footprint of the data structure, we need to connect it to the appropriate footprint parameter of T^s_n.

$$\mathsf{defineY} \triangleq (\mathbf{y}_{\mathsf{ds}} = \mathbf{z}) \wedge \bigwedge_{\mathsf{ds}' \in \mathcal{D} \setminus \{\mathsf{ds}\}} \mathbf{y}_{\mathsf{ds}'} = \emptyset$$

Next, we define reachability predicates, which we will need both to define the footprint of the data structure and to enforce its acyclicity. Although reachability is not expressible in first-order logic, since we are only interested in finite reachability with respect to the (locations interpreting the) variables $x_1, \ldots, x_n$, we can define m-step reachability predicates $r^{\mathbf{z}}_m$, for $1 \leq m \leq n$, as follows.

$$R_1 \triangleq \bigwedge_{1 \leq i,j \leq n} r^{\mathbf{z}}_1(x_i, x_j) \Leftrightarrow (x_i \in \mathbf{z} \wedge \neg\mathsf{isleaf}(x_j) \wedge \mathsf{succ}(x_i, x_j))$$

$$R_m \triangleq \bigwedge_{1 \leq i,j \leq n} r^{\mathbf{z}}_m(x_i, x_j) \Leftrightarrow (r^{\mathbf{z}}_{m-1}(x_i, x_j) \vee \bigvee_{1 \leq k \leq n} (r^{\mathbf{z}}_{m-1}(x_i, x_k) \wedge r^{\mathbf{z}}_1(x_k, x_j))$$

$$\mathsf{reachability} \triangleq R_1 \wedge R_2 \wedge \cdots \wedge R_n$$

By requiring $\neg\mathsf{isleaf}(x_j)$ for the target x_j, we ensure that only the locations that are allocated within ds are in the reachability predicate. This means that we can define the footprint of ds in terms of reachability: We define a formula footprint that asserts that the set $\mathbf{z}$ (the footprint of ds) is defined as the set of allocated locations reachable from x.

$$\begin{aligned}\mathsf{footprint} \triangleq\ & \mathbf{z} \subseteq \{x_1, \ldots, x_n\} \wedge (\mathsf{isleaf}(x) \Rightarrow \mathbf{z} = \emptyset) \wedge \\ & \wedge \big(\neg\mathsf{isleaf}(x) \Rightarrow \\ & \quad \textstyle\bigwedge_{1 \leq i \leq n} ((x_i \in \mathbf{z}) \Leftrightarrow ((x_i = x) \vee r^{\mathbf{z}}_n(x, x_i)))\big)\end{aligned}$$

Next, the formula structure ensures that the elements of the data structure ds are part of an acyclic data structure, starting at x, with no sharing of non-nil nodes.

$$\mathsf{rootloc} \triangleq (\bigvee_{1 \leq i \leq n} x = x_i) \vee (x = \mathsf{nil}) \vee (\bigvee_{s \in \mathbf{s}} x = s)$$

$$\begin{aligned}\mathsf{oneparent} \triangleq \bigwedge_{1 \leq i \leq n} x_i \in \mathbf{z} \Rightarrow \Bigg(& \bigwedge_{\mathsf{f} \neq \mathsf{g} \in \mathcal{F}_{\mathsf{ds}}} (\mathsf{f}[x_i] = \mathsf{g}[x_i] \Rightarrow \mathsf{f}[x_i] = \mathsf{nil}) \\ & \wedge \bigwedge_{1 \leq j \leq n} (x_j \in \mathbf{z} \wedge x_i \neq x_j \Rightarrow \\ & \quad \bigwedge_{\mathsf{f},\mathsf{g} \in \mathcal{F}_{\mathsf{ds}}} (\mathsf{f}[x_i] = \mathsf{g}[x_j] \Rightarrow \mathsf{f}[x_i] = \mathsf{nil}))\Bigg)\end{aligned}$$

$$\mathsf{structure} \triangleq \mathsf{rootloc} \wedge (\neg\mathsf{isleaf}(x) \Rightarrow x \in \mathbf{z}) \wedge \mathsf{oneparent} \wedge \neg r^{\mathbf{z}}_n(x, x)$$

The formula rootloc enforces that the root of the data structure, x, is either allocated (and hence among $\mathbf{x}_{\mathsf{global}} \subseteq \{x_1, \ldots, x_n\}$) or a leaf,

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

as required by the data-structure semantics. The formula oneparent expresses that (1) the non-null successors of node x_i are pairwise different or null (i.e., the left and right child of a tree node cannot be the same) and (2) the non-null successors of distinct nodes x_i, x_j are pairwise different. Together, this guarantees that every node has at most one incoming pointer.

The formula structure then expresses that the data structure has a nonempty footprint unless its source is also a sink; that every node has at most one incoming pointer; and that x is not reachable from itself. Combined, these constraints guarantee that $\mathsf{ls}(x, \mathbf{s}, \mathcal{P})$ can only be interpreted by an acyclic list from x to $\mathbf{s}$ and that $\mathsf{tree}(x, \mathbf{s}, \mathcal{P})$ can only be interpreted by a tree with root x and holes $\mathbf{s}$.

We still need to axiomatize the properties of holes: we must assert that the holes of ds are pairwise different, occur exactly once, are the only (non-null) leaves of the structure, and, for trees, occur in the same order as prescribed by the vector $\mathbf{s} = \langle s_1, \ldots, s_k \rangle$.

$$
\begin{aligned}
\mathsf{holeseq} &\triangleq (\mathsf{isleaf}(x) \Rightarrow \bigwedge_{s \in \mathbf{s}} x = s) \wedge \bigwedge_{1 \leq i < j \leq k} s_i \neq s_j \\
\mathsf{holesoccur} &\triangleq \neg \mathsf{isleaf}(x) \implies \bigwedge_{s \in \mathbf{s}} \bigvee_{1 \leq p \leq n} (x_p \in \mathbf{z} \wedge \mathsf{succ}(x_p, s)) \\
\mathsf{correctleaves} &\triangleq \bigwedge_{1 \leq i \leq n} \bigwedge_{\mathsf{f} \in \mathcal{F}_{\mathsf{ds}}} (x_i \in \mathbf{z} \wedge \mathsf{f}[x_i] \notin \mathbf{z}) \Rightarrow \mathsf{isleaf}(\mathsf{f}[x_i]) \\
r_n^{\mathbf{z}}(y_1, y_2, \mathsf{f}) &\triangleq \mathsf{f}[y_1] = y_2 \vee (\mathsf{f}[y_1] \in \mathbf{z} \wedge r_n^{\mathbf{z}}(\mathsf{f}[y_1], y_2)) \\
\mathsf{fldhole}_{\mathsf{f}}(x_p, s) &\triangleq \mathsf{f}[x_p] = s \\
&\quad \vee \bigvee_{1 \leq c \leq n} (r_n^{\mathbf{z}}(x_p, x_c, \mathsf{f}) \wedge x_c \in \mathbf{z} \wedge \mathsf{succ}(x_c, s)) \\
\mathsf{before}(x_p, s_1, s_2) &\triangleq \mathsf{fldhole}_{\mathsf{l}}(x_p, s_1) \wedge \mathsf{fldhole}_{\mathsf{r}}(x_p, s_2) \\
\mathsf{ordered} &\triangleq \bigwedge_{1 \leq i < k} \bigvee_{1 \leq p \leq n} (x_p \in \mathbf{z} \wedge \mathsf{before}(x_p, s_i, s_{i+1}))
\end{aligned}
$$

The formula holeseq asserts that if the source x of the data structure is itself a leaf, then it is equal to *all* holes; and that the holes are pairwise different. This guarantees, for example, that the translation of a formula such as $\mathsf{ls}(x, \langle s_1, s_2 \rangle)$ is unsatisfiable, in accordance with $\mathbf{SSL}^+$ semantics.

The formula holesoccur states that every hole has a direct predecessor in $\mathbf{z}$, i.e., in the data-structure footprint, ensuring that all holes occur. Note that it is necessary to state this property in such a roundabout way, because the holes themselves are *not* in the footprint $\mathbf{z}$.

correctleaves ensures that all successors of variables in the footprint that are not themselves in the footprints, i.e., all leaves of the data structure, are indeed among the set of leaves (either nil or a hole).

Finally, the formula ordered expresses that the holes occur in the correct left-to-right order. It makes use of several auxiliary formulas. In particular, the formula $\mathsf{fldhole}_{\mathsf{f}}(x_p, s)$ states that the hole s is in

TU Bibliothek, Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar. The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.

the f-subtree of x_p and $r_n^{\mathbf{z}}(x, y, \mathsf{f})$ denotes that y is reachable from x through f as the first step.

We combine the above constraints into

$$\mathsf{leafconstraints}^{\mathsf{ls}} \triangleq \mathsf{holesoccur} \wedge \mathsf{holeseq} \wedge \mathsf{correctleaves} \text{ and}$$
$$\mathsf{leafconstraints}^{\mathsf{tree}} \triangleq \mathsf{holesoccur} \wedge \mathsf{holeseq} \wedge \mathsf{correctleaves} \wedge \mathsf{ordered}$$

Finally, we define the alldata formula that ensures that the data fields of the allocated nodes of the data structures respects the given (unary and binary, existential and universal) data predicates. In the encoding of binary data predicates, we reuse the ternary function $r_n^{\mathbf{z}}$ that expresses reachability through the field f. Otherwise, the encoding of data predicates is straightforward.

$$\mathsf{data}([F]^{\forall}) \triangleq \mathsf{map}_{\Rightarrow}(\mathbf{z}, \mathsf{map}_F(\mathsf{d})) = \mathbf{K}(\mathsf{t})$$
$$\mathsf{data}([F]^{\exists}) \triangleq \mathsf{map}_{\Rightarrow}(\mathbf{z}, \mathsf{map}_F(\mathsf{d})) \neq \mathbf{K}(\mathsf{f})$$
$$\mathsf{data}([\mathsf{f}\colon F]^{\forall}) \triangleq \bigwedge_{1 \leq i,j \leq n} x_i, x_j \in \mathbf{z} \wedge r_n^{\mathbf{z}}(x_i, x_j, \mathsf{f}) \Rightarrow F(\mathsf{d}[x_i], \mathsf{d}[x_j])$$
$$\mathsf{data}([\mathsf{f}\colon F]^{\exists}) \triangleq \bigvee_{1 \leq i,j \leq n} x_i, x_j \in \mathbf{z} \wedge r_n^{\mathbf{z}}(x_i, x_j, \mathsf{f}) \wedge F(\mathsf{d}[x_i], \mathsf{d}[x_j])$$
$$\mathsf{alldata} \triangleq \bigwedge_{F \in \mathcal{P}} \mathsf{data}(F)$$

Putting all the auxiliary formulas together, we define the translation of inductive predicates $\mathsf{ds} \in \{\mathsf{ls}, \mathsf{tree}\}$ to SMT as follows.

$$\begin{aligned} \mathsf{T}_n^s(\mathsf{ds}(x, \mathbf{s}, \mathcal{P}), \mathbf{Y}) \triangleq\ & \text{let } r_1^{\mathbf{z}}, \ldots, r_n^{\mathbf{z}}, \mathbf{z} \text{ be fresh} \\ & \text{let } A = \mathsf{structure} \wedge \mathsf{leafconstraints}^{\mathsf{ds}} \wedge \mathsf{alldata} \\ & \text{let } B = \mathsf{reachability} \wedge \mathsf{footprint} \wedge \mathsf{defineY} \text{ in} \\ & \langle A, B, \{r_1^{\mathbf{z}}, \ldots, r_n^{\mathbf{z}}, \mathbf{z}\} \rangle \end{aligned}$$

I would like to reiterate that the reachability constraint reachability only ensures that the predicates $r_k^{\mathbf{z}}$ are fully defined on the set $\{x_1, \ldots, x_n\}$ and can be interpreted arbitrarily elsewhere. Nevertheless, this is sufficient for the translation to be correct, because the A part of the translation cannot distinguish two interpretations of $r_k^{\mathbf{z}}$ that differ only outside of $\{x_1, \ldots, x_n\}$. This is crucial for the correctness of the encoding as it supports the Z-Equivalence property of the translation.

For the predicates $\mathsf{ds}_{\geq 2}$, we extend the encoding with a depth constraint.

$$\mathsf{depth} \geq 2 \triangleq \bigvee_{1 \leq i \leq n} (r_2^{\mathbf{z}}(x, x_i) \wedge \neg r_1^{\mathbf{z}}(x, x_i))$$
$$\begin{aligned} \mathsf{T}_n^s(\mathsf{ds}_{\geq 2}(x, \mathbf{s}, \mathcal{P}), \mathbf{Y}) \triangleq\ & \text{let } \langle A, B, Z \rangle = \mathsf{T}_n^s(\mathsf{ds}(x, \mathbf{s}, \mathcal{P}), \mathbf{Y}) \\ & \text{in } \langle A \wedge \mathsf{depth} \geq 2, B, Z \rangle \end{aligned}$$

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Theorem 7.19. *Let ϕ be a* $\mathbf{SSL}^{+}_{\mathsf{data}}$ *formula and let* $n \triangleq \mathsf{bound}(\phi)$ *be the realizability size of* ϕ*. Then* ϕ *is* $\mathbf{SSL}^{+}_{\mathsf{data}}$*-satisfiable if and only if the SMT translation* $F = T_n(\phi)$ *is satisfiable. Moreover, the translation* F *is polynomial in the size of* ϕ*.*

As $\mathcal{T}_{\mathsf{array}}$ is in NP, this yields an NP decision procedure for $\mathbf{SSL}^{+}_{\mathsf{data}}$ if $\mathcal{T}_{\mathbf{Data}}$ is in NP, matching the complexity result of Theorem 7.15 in Section 7.1 that we derived from the small-model property.

In the remainder of this section, I give a detailed proof of Theorem 7.19.

Evaluation of the SMT Encoding

I've implemented an almost identical SMT encoding for the logic $\mathbf{SL}^{*}_{\mathsf{data}}$ in my tool `Sloth`. A preliminary evaluation of `Sloth` showed that the SMT encoding can be used to solve small but intricate entailments about all data structures discussed here—binary search trees, heaps, etc.—but does not yet scale to large queries. Since I only evaluated the encoding of $\mathbf{SL}^{*}_{\mathsf{data}}$, not of $\mathbf{SSL}^{+}_{\mathsf{data}}$, I do not include the experimental results here. They are available in the (informal) proceedings of the First Workshop on Automated Deduction for Separation Logics (ADSL) [KJW18b].

7.2.1 *Correctness of the SMT Encoding of Lists and Trees*

I begin by showing the correctness of the encoding of lists and trees. I focus on trees, as the list encoding is essentially a simpler special case of the tree encoding. Disregarding data predicates for the moment, the correctness argument can be summarized as follows.

1. The tree predicate $\mathsf{tree}(x, \mathbf{y})$ holds in $(\mathfrak{s}, \mathfrak{h})$ iff $(\mathfrak{s}, \mathfrak{h})$ is a directed tree with root x and holes $\mathbf{y}$ (Lemma 5.5).

2. A model $(\mathfrak{s}, \mathfrak{h})$ and its corresponding models induce the same directed graph (Lemma 7.21).

3. The encoding of $\mathsf{tree}(x, \mathbf{y})$ holds in a model $\mathcal{M}_{\text{SMT}}$ if and only if $\mathcal{M}_{\text{SMT}}$ is a directed tree with root $x^{\mathcal{M}_{\text{SMT}}}$, terminal $\mathsf{nil}^{\mathcal{M}_{\text{SMT}}}$, and holes $\mathbf{y}^{\mathcal{M}_{\text{SMT}}}$ (Lemma 7.22).

This implies the correctness of the encoding, because by Lemma 7.17, every stack–heap model has a corresponding SMT model and vice versa, allowing us to transform every model of $\mathsf{tree}(x, \mathbf{y})$ into an SMT model of the tree encoding and vice versa.

We must first clarify what we mean by the directed graph induced by an SMT model.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Definition 7.20. *Let $\mathcal{M}_{\text{SMT}}$ be an SMT model. The directed graph of $\mathcal{M}_{\text{SMT}}$ is given by*

$$\mathcal{G}_{\mathcal{M}_{\text{SMT}}} \triangleq \left\{ \langle \ell, 0, \ell' \rangle \mid \ell \in \mathbf{x}_{\mathsf{global}}^{\mathcal{M}_{\text{SMT}}}, \mathsf{l}^{\mathcal{M}_{\text{SMT}}}[\ell] = \ell' \right\}$$
$$\cup \left\{ \langle \ell, 1, \ell' \rangle \mid \ell \in \mathbf{x}_{\mathsf{global}}^{\mathcal{M}_{\text{SMT}}}, \mathsf{r}^{\mathcal{M}_{\text{SMT}}}[\ell] = \ell' \right\}$$

Lemma 7.21. *Let $(\mathfrak{s}, \mathfrak{h})$ be a stack–heap model and let $\mathcal{M}_{\text{SMT}}$ be a corresponding SMT model. Then* $\mathsf{igraph}(\mathfrak{h}) = \mathcal{G}_{\mathcal{M}_{\text{SMT}}}$.

Proof. Follows immediately from the definition of model correspondence. □

We show that the (directed graphs of) the first-order models of $\mathsf{T}_n^s(\mathsf{tree}(x, \langle s_1, \ldots, s_k \rangle, \mathcal{P}))$ are directed trees with holes.

Lemma 7.22. *Let $\mathcal{M}_{\text{SMT}}$ be an SMT model.*

$$\mathcal{M}_{\text{SMT}} \models \Delta_{\mathbf{SSL}}^n \wedge \exists \mathbf{Y} \exists r_1^{\mathbf{z}}, \ldots, \exists r_n^{\mathbf{z}} \exists \mathbf{z}.\ T_n^s(\mathsf{tree}(x, \langle y_1, \ldots, y_k \rangle))$$

if and only if $\mathcal{G}_{\mathcal{M}_{\text{SMT}}}$ is a directed tree with root $x^{\mathcal{M}_{\text{SMT}}}$, terminal $\mathsf{nil}^{\mathcal{M}_{\text{SMT}}}$, and holes $\left\langle y_1^{\mathcal{M}_{\text{SMT}}}, \ldots, y_k^{\mathcal{M}_{\text{SMT}}} \right\rangle$.

Proof. Let $\mathbf{y}^{\mathcal{M}_{\text{SMT}}} \triangleq \left\langle y_1^{\mathcal{M}_{\text{SMT}}}, \ldots, y_k^{\mathcal{M}_{\text{SMT}}} \right\rangle$. First observe that the formula is satisfied by a model with $\mathbf{x}_{\mathsf{global}}^{\mathcal{M}_{\text{SMT}}} = \emptyset$ iff $x^{\mathcal{M}_{\text{SMT}}}$ is equal to the unique hole (if any) or to $\mathsf{nil}^{\mathcal{M}_{\text{SMT}}}$ iff $\mathcal{G}_{\mathcal{M}_{\text{SMT}}}$ is an empty directed tree with root $x^{\mathcal{M}_{\text{SMT}}}$, terminal $\mathsf{nil}^{\mathcal{M}_{\text{SMT}}}$, and holes $\mathbf{y}^{\mathcal{M}_{\text{SMT}}}$.

Now assume that $\mathbf{x}_{\mathsf{global}}^{\mathcal{M}_{\text{SMT}}} \neq \emptyset$. As per Definition 4.11, we need to prove that $\mathcal{G}_{\mathcal{M}_{\text{SMT}}}$ satisfies the following properties.

1. $\mathcal{G}_{\mathcal{M}_{\text{SMT}}}$ is acyclic.
2. $x^{\mathcal{M}_{\text{SMT}}}$ is the unique source of $\mathcal{G}_{\mathcal{M}_{\text{SMT}}}$.
3. $\mathsf{nil}^{\mathcal{M}_{\text{SMT}}}$ is a sink of $\mathcal{G}_{\mathcal{M}_{\text{SMT}}}$ or does not occur in the nodes of $\mathcal{G}_{\mathcal{M}_{\text{SMT}}}$.
4. Every node except $\mathsf{nil}^{\mathcal{M}_{\text{SMT}}}$ has at most one predecessor.
5. $\mathsf{sinkseq}(\mathcal{G}_{\mathcal{M}_{\text{SMT}}}) = \mathbf{y}^{\mathcal{M}_{\text{SMT}}}$.

We prove these properties one by one. Let $\mathcal{M}_{\text{SMT1}}$ be the extension of $\mathcal{M}_{\text{SMT}}$ with symbols $\mathbf{Y}, r_1^{\mathbf{z}}, \ldots, r_n^{\mathbf{z}}, \mathbf{z}$ such that $\mathcal{M}_{\text{SMT1}} \models \Delta_{\mathbf{SSL}}^n \wedge \mathsf{T}_n^s(\mathsf{tree}(x, \mathbf{y}))$. Observe that $\mathcal{G}_{\mathcal{M}_{\text{SMT1}}} = \mathcal{G}_{\mathcal{M}_{\text{SMT}}}$.

1. It suffices to show that none of the nodes $x_1, \ldots, x_n$ is on a cycle.

   If $x_i = x$, we note that $\mathcal{M}_{\text{SMT1}} \models \mathsf{structure}$, so $\mathcal{M}_{\text{SMT1}} \models \neg r_n^{\mathbf{z}}(x, x)$, guaranteeing that x is not reachable from itself, i.e., not on a cycle.

   Since $\mathcal{M}_{\text{SMT1}} \models \mathsf{footprint}$, every other node $x_i \in \mathbf{x}_{\mathsf{global}}$, $x_i \neq x$, is reachable from x. Since every node except nil has at most one predecessor, the only path to x_i then is the path from x. It follows that x_i cannot be on a cycle. Since x_i was arbitrary, it follows that $\mathcal{G}_{\mathcal{M}_{\text{SMT}}}$ is acyclic.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

2. Since $\mathcal{M}_{\text{SMT1}} \models \mathsf{footprint}$, every other node $x_i^{\mathcal{M}_{\text{SMT}}} \in \mathbf{x}_{\text{global}}^{\mathcal{M}_{\text{SMT}}}$ is reachable from $x^{\mathcal{M}_{\text{SMT}}}$, so no other node except $x^{\mathcal{M}_{\text{SMT}}}$ can possible be a source of $\mathcal{G}_{\mathcal{M}_{\text{SMT}}}$. Because $\mathcal{G}_{\mathcal{M}_{\text{SMT}}}$ is acyclic, it follows that $x^{\mathcal{M}_{\text{SMT}}}$ has no predecessors, so $x^{\mathcal{M}_{\text{SMT}}}$ is indeed a source of $\mathcal{G}_{\mathcal{M}_{\text{SMT}}}$.

3. $\mathcal{M}_{\text{SMT1}} \models \Delta^n_{\mathbf{SSL}}$, so in particular $\mathcal{M}_{\text{SMT1}} \models \mathsf{nil} \notin \mathbf{x}_{\text{global}}$. By definition, only the nodes in $\mathbf{x}_{\text{global}}^{\mathcal{M}_{\text{SMT}}}$ have successors in $\mathcal{G}_{\mathcal{M}_{\text{SMT}}}$. Consequently, nil is a sink of $\mathcal{G}_{\mathcal{M}_{\text{SMT}}}$ or does not occur in the nodes of $\mathcal{G}_{\mathcal{M}_{\text{SMT}}}$.

4. $\mathcal{M}_{\text{SMT1}} \models \mathsf{oneparent}$, so it holds that (1) the successors of all nodes in $\mathbf{x}_{\text{global}}^{\mathcal{M}_{\text{SMT}}}$ are either both nil or distinct and (2) the successors of all pairs of nodes may only overlap on nil. Consequently, every node except for $\mathsf{nil}^{\mathcal{M}_{\text{SMT}}}$ has at most one predecessor.

5. The reachability constraint guarantees that the nodes x_j in $\mathbf{z}$ (and thus in $\mathbf{x}_{\text{global}}^{\mathcal{M}_{\text{SMT}}}$) do not satisfy $\mathsf{isleaf}(x_j)$, implying that the nodes $\mathbf{y}^{\mathcal{M}_{\text{SMT}}}$ do not have any successors in $\mathcal{G}_{\mathcal{M}_{\text{SMT}}}$, i.e., they are sinks of the graphs if they occur in the graph As $\mathcal{M}_{\text{SMT1}} \models \mathsf{holesoccur}$, every node in $\mathbf{y}^{\mathcal{M}_{\text{SMT}}}$ occurs, so the sinks of $\mathcal{G}^{\mathcal{M}_{\text{SMT}}}$ are precisely the nodes $\mathbf{y}^{\mathcal{M}_{\text{SMT}}}$.

   As $\mathcal{M}_{\text{SMT1}} \models \mathsf{ordered}$, we have that $y_i^{\mathcal{M}_{\text{SMT}}} \prec y_j^{\mathcal{M}_{\text{SMT}}}$ (i.e., that $y_i^{\mathcal{M}_{\text{SMT}}}$ occurs in the access path ordering before $y_j^{\mathcal{M}_{\text{SMT}}}$) iff $i < j$, implying that $\mathsf{sinkseq}(\mathcal{G}_{\mathcal{M}_{\text{SMT}}}) = \mathbf{y}^{\mathcal{M}_{\text{SMT}}}$.

Consequently, $\mathcal{G}_{\mathcal{M}_{\text{SMT}}}$ is a directed tree with root $x^{\mathcal{M}_{\text{SMT}}}$, terminal $\mathsf{nil}^{\mathcal{M}_{\text{SMT}}}$ and holes $\mathbf{y}^{\mathcal{M}_{\text{SMT}}}$. □

To finish the correctness proof, we need to show that the data predicates $\mathcal{P}$ in $\mathsf{tree}(x, \mathbf{y}, \mathcal{P})$ hold in $(\mathfrak{s}, \mathfrak{h})$ iff they hold in the corresponding SMT model.

Lemma 7.23. *Let $(\mathfrak{s}, \mathfrak{h})$ be a model and $\mathcal{M}_{\text{SMT}}$ its corresponding SMT model. Let P be a data predicate. Then $(\mathfrak{s}, \mathfrak{h}) \models P$ iff $\mathcal{M}_{\text{SMT}} \models \mathsf{data}(P)$.*

Proof. We make a case distinction based on the type of data predicate.

CASE $[F]^{\exists}$. $(\mathfrak{s}, \mathfrak{h}) \models [F]^{\exists}$
iff there ex. a location $\ell \in \mathrm{dom}(\mathfrak{h})$ s.t. $F(\mathsf{d}(\ell))$ holds
iff there exists an $\ell \in \mathbf{x}_{\text{global}}^{\mathcal{M}_{\text{SMT}}}$ such that $F(\mathsf{d}^{\mathcal{M}_{\text{SMT}}}[\ell])$ holds
iff $\mathcal{M}_{\text{SMT}} \models \mathsf{map}_{\Rightarrow}(\mathbf{z}, \mathsf{map}_F(\mathsf{d})) \neq \mathbf{K}(\mathsf{f})$.

CASE $[F]^{\forall}$. Analogously.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

CASE $[\mathsf{f}\colon F]^{\exists}$. $(\mathfrak{s},\mathfrak{h}) \models [\mathsf{f}\colon F]^{\exists}$

iff there exist locations $\ell_1, \ell_2 \in \mathrm{dom}(\mathfrak{h})$ such that

$\ell_1 \xrightarrow{\mathsf{f}*}_{\mathfrak{h}} \ell_2$ and $F(\mathsf{d}(\ell_1), \mathsf{d}(\ell_2))$ holds

iff there exist locations $\ell_1, \ell_2 \in \mathbf{z}^{\mathcal{M}_{\text{SMT}}}$ such that

$\ell_1 \xrightarrow{\mathsf{f}*}_{\mathcal{M}_{\text{SMT}}} \ell_2$ and $F(\mathsf{d}^{\mathcal{M}_{\text{SMT}}}[\ell_1], \mathsf{d}^{\mathcal{M}_{\text{SMT}}}[\ell_2])$ holds

iff there exist $x_1^{\mathcal{M}_{\text{SMT}}}, x_2^{\mathcal{M}_{\text{SMT}}} \in \mathbf{z}^{\mathcal{M}_{\text{SMT}}}$ such that

$x_1^{\mathcal{M}_{\text{SMT}}} \xrightarrow{\mathsf{f}*}_{\mathcal{M}_{\text{SMT}}} x_2^{\mathcal{M}_{\text{SMT}}}$ and

$F(\mathsf{d}^{\mathcal{M}_{\text{SMT}}}[x_1^{\mathcal{M}_{\text{SMT}}}], \mathsf{d}^{\mathcal{M}_{\text{SMT}}}[x_2^{\mathcal{M}_{\text{SMT}}}])$ holds

iff $\mathcal{M}_{\text{SMT}} \models \bigvee_{1 \le i,j \le n} x_i, x_j \in \mathbf{z} \wedge r_n^{\mathbf{z}}(x_i, x_j, \mathsf{f}) \wedge F(\mathsf{d}[x_i], \mathsf{d}[x_j])$.

CASE $[\mathsf{f}\colon F]^{\forall}$. Analogously. □

We now have all the necessary ingredients for proving the correctness of the SMT encoding of trees.

Lemma 7.24. *Let* $\langle A, B, Z\rangle = \mathcal{T}_n^s(\mathsf{tree}(x, \langle s_1, \dots, s_k\rangle, \mathcal{P}), \mathbf{Y})$, *let* $F = A \wedge B \wedge \Delta_{\mathbf{SSL}}^n$. *Let* $(\mathfrak{s},\mathfrak{h})$ *be a stack–heap pair and* $\mathcal{M}_{\text{SMT}}$ *an SMT model that corresponds to* $(\mathfrak{s},\mathfrak{h})$. *Then:*

1. *B satisfies* $(\mathbf{Y} \cup Z)$*-existence,*
2. *A and B satisfy* $(\mathbf{Y} \cup Z)$*-equivalence,*
3. *Assume* $\mathcal{M}_{\text{SMT}}$ *corresponds to* $(\mathfrak{s},\mathfrak{h})$. *Then:*
   a) $((\mathfrak{s},\mathfrak{h}) \models \mathsf{tree}(x, \langle s_1, \dots, s_k\rangle, \mathcal{P})$ *and* $|\mathfrak{h}| \le n)$ *iff* $\mathcal{M}_{\text{SMT}} \models \exists \mathbf{Y} \exists Z.\, F$
   b) $((\mathfrak{s},\mathfrak{h}) \models \mathsf{tree}_{\ge 2}(x, \langle s_1, \dots, s_k\rangle, \mathcal{P})$ *and* $|\mathfrak{h}| \le n)$ *iff* $(\mathcal{M}_{\text{SMT}} \models \exists \mathbf{Y} \exists Z.\, F$ *and* $\left|\mathbf{x}_{\mathsf{global}}^{\mathcal{M}_{\text{SMT}}}\right| \ge 2)$.

Proof. Observe that $\mathbf{Y} \cup Z = \langle \mathbf{y}_{\mathsf{ls}}, \mathbf{y}_{\mathsf{tree}}, \mathbf{z}, r_1^{\mathbf{z}}, \dots, r_n^{\mathbf{z}}\rangle$.

We consider each of the three statements separately:

1. The formula $B = \mathsf{reachability} \wedge \mathsf{footprint} \wedge \mathsf{defineY}$ is always satisfied in the assignment that assigns to the sets $\mathbf{z}$ and $\mathbf{y}_{\mathsf{ds}}$ those locations among $\{x_1, \dots, x_n\}$ that are reachable from x without going through a hole, and to $\mathbf{y}_{\mathsf{ds}'}$ the empty set for $\mathsf{ds}' \neq \mathsf{ds}$.

   In this interpretation, we interpret each $r_i^{\mathbf{z}}$ as the predicate representing i-step reachability on the set $\mathbf{z}$. Note, in particular, that $\mathbf{z}$ is empty if and only if the tree root x itself is a hole; and that if $\mathbf{z}$ is empty all the $r_i^{\mathbf{z}}$ are empty as well.

2. Assignments to $(\mathbf{Y} \cup Z)$ are not unique, but they are unique on the interpretation of $\{x_1, \dots, x_n\}$, which are the only locations that influence the truth value of A.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

3. By Lemma 5.5, $(\mathfrak{s}, \mathfrak{h}) \models \mathsf{tree}(x, \mathbf{y}, \mathcal{P})$ if and only if $(\mathfrak{s}, \mathfrak{h})$ is a directed tree with holes $\mathbf{s}$ and satisfies $\mathcal{P}$. By Lemma 7.22, $\mathcal{M}_{\text{SMT}}$ satisfies the tree encoding if and only if $\mathcal{M}_{\text{SMT}}$ is a directed tree with holes $\mathbf{s}$ and satisfies $\mathcal{P}$. The first sub-claim follows because by Lemma 7.21, $(\mathfrak{s}, \mathfrak{h})$ is a directed tree with holes $\mathbf{s}$ iff its corresponding models $\mathcal{M}_{\text{SMT}}$ are directed trees with holes $\mathbf{s}$.

   The second sub-claim follows because $\mathcal{M}_{\text{SMT}} \models \mathsf{depth} \geq 2$ iff $\left|\mathbf{x}_{\mathsf{global}}^{\mathcal{M}_{\text{SMT}}}\right| \geq 2$ iff $\mathcal{M}_{\text{SMT}}$ corresponds to a model with $|\mathfrak{h}| \geq 2$. □

I omit a proof of the correctness of the list encoding, as it is a simpler special case of the tree encoding.

Lemma 7.25. *Let* $\langle A, B, Z \rangle = T_n^{\mathfrak{s}}(\mathsf{ls}(x, \langle s_1, \ldots, s_k \rangle, \mathcal{P}), \mathbf{Y})$ *and let* $F = A \wedge B \wedge \Delta^n_{\mathbf{SSL}}$. *Then:*

1. *B satisfies* $(\mathbf{Y} \cup Z)$*-existence,*
2. *A and B satisfy* $(\mathbf{Y} \cup Z)$*-equivalence,*
3. *Assume* $\mathcal{M}_{\text{SMT}}$ *corresponds to* $(\mathfrak{s}, \mathfrak{h})$*. Then:*
   a) $((\mathfrak{s}, \mathfrak{h}) \models \mathsf{ls}(x, \mathbf{y}, \mathcal{P})$ *and* $|\mathfrak{h}| \leq n)$ *iff* $\mathcal{M}_{\text{SMT}} \models \exists \mathbf{Y} \exists Z.\, F$
   b) $((\mathfrak{s}, \mathfrak{h}) \models \mathsf{ls}_{\geq 2}(x, \mathbf{y}, \mathcal{P})$ *and* $|\mathfrak{h}| \leq n)$ *iff* $(\mathcal{M}_{\text{SMT}} \models \exists \mathbf{Y} \exists Z.\, F$ *and* $\left|\mathbf{x}_{\mathsf{global}}^{\mathcal{M}_{\text{SMT}}}\right| \geq 2)$.

7.2.2 *Correctness of the SMT Encoding of Arbitrary Positive Formulas*

To show the correctness of the full encoding, we also need a composition operation on SMT models that mimics the heap-composition operator $\uplus^{\mathfrak{s}}$. Such an operation need only be defined on SMT models that are *compatible* in the following sense.

Definition 7.26. *Let* $\mathfrak{s}$ *be a stack and let* $\mathcal{M}_{\text{SMT1}}$ *and* $\mathcal{M}_{\text{SMT2}}$ *be two first-order models. We say that* $\mathcal{M}_{\text{SMT1}}$ *and* $\mathcal{M}_{\text{SMT2}}$ *are* $\mathfrak{s}$-compatible *if they have disjoint global footprints (i.e.,* $\mathbf{x}_{\mathsf{global}}^{\mathcal{M}_{\text{SMT1}}} \cap \mathbf{x}_{\mathsf{global}}^{\mathcal{M}_{\text{SMT2}}} = \emptyset$*), but agree on the interpretation of all* $\mathbf{SSL}^{+}_{\mathsf{data}}$*-common elements (all field arrays* f*, all variables in* $\mathrm{dom}(\mathfrak{s})$*,* nil *and all variables* x_i*).*

We define a partial composition operation on compatible models.

Definition 7.27. *If* $\mathcal{M}_{\text{SMT1}}$ *and* $\mathcal{M}_{\text{SMT2}}$ *are compatible, we define the combined model* $\mathcal{M}_{\text{SMT1}} \oplus \mathcal{M}_{\text{SMT2}}$ *as the model that interprets* $\mathbf{x}_{\mathsf{global}}$ *by* $\mathbf{x}_{\mathsf{global}}^{\mathcal{M}_{\text{SMT1}}} \cup \mathbf{x}_{\mathsf{global}}^{\mathcal{M}_{\text{SMT2}}}$ *and interprets all variables that occur in* $\mathcal{M}_{\text{SMT1}}$ *or* $\mathcal{M}_{\text{SMT2}}$ *(or both) like they are interpreted in* $\mathcal{M}_{\text{SMT1}}$ *and* $\mathcal{M}_{\text{SMT2}}$*. If* $\mathcal{M}_{\text{SMT1}}$ *and* $\mathcal{M}_{\text{SMT2}}$ *are not compatible,* $\mathcal{M}_{\text{SMT1}} \oplus \mathcal{M}_{\text{SMT2}} = \bot$.

Composition is well defined because of the assumption that $\mathcal{M}_{\text{SMT1}}$ and $\mathcal{M}_{\text{SMT2}}$ are compatible.

As desired, there is a close correspondence between the composition of heaps via $\uplus^{\mathfrak{s}}$ and the composition of first-order models via $\oplus$:

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Lemma 7.28 (Correspondence of composition)**.** *Let $\mathfrak{s}$ be a stack and let $\mathfrak{h}_1, \mathfrak{h}_2$ be heaps such that $|\mathfrak{h}_1| + |\mathfrak{h}_2| = n$. Then $\mathfrak{h}_1 \uplus^{\mathfrak{s}} \mathfrak{h}_2 \neq \bot$ if and only if there exist compatible SMT models $\mathcal{M}_{\text{SMT}1}, \mathcal{M}_{\text{SMT}2}$ such that (1) $\mathcal{M}_{\text{SMT}i} \models \Delta^n_{\mathbf{SSL}}$, (2) $\mathcal{M}_{\text{SMT}i}$ corresponds to $(\mathfrak{s}, \mathfrak{h}_i)$, and (3) $\mathcal{M}_{\text{SMT}1} \oplus \mathcal{M}_{\text{SMT}2}$ corresponds to $\mathfrak{h}_1 \uplus^{\mathfrak{s}} \mathfrak{h}_2$.*

Proof. We pick models $\mathcal{M}_{\text{SMT}1}$ and $\mathcal{M}_{\text{SMT}2}$ such that they correspond to $(\mathfrak{s}, \mathfrak{h}_1)$ and $(\mathfrak{s}, \mathfrak{h}_2)$ and such that they interpret the variables $x_1, \ldots, x_n$ as well as the arrays n, l, r, and d in an identical way. The former is possible because we can assign to the x_i pairwise different elements of $\text{dom}(\mathfrak{h}_1) \cup \text{dom}(\mathfrak{h}_2)$; the latter is possible because $\mathfrak{h}_1 \uplus^{\mathfrak{s}} \mathfrak{h}_2 \neq \bot$, so we can interpret the array f simply by $\mathsf{f}(\ell)$ for all $\ell \in \text{dom}(\mathfrak{h}_1) \cup \text{dom}(\mathfrak{h}_2)$ and arbitrarily (but in the same way in both SMT models) on all other locations. By construction, $\mathcal{M}_{\text{SMT}1} \oplus \mathcal{M}_{\text{SMT}2}$ corresponds to $\mathfrak{h}_1 \uplus^{\mathfrak{s}} \mathfrak{h}_2$. ◻

Lemma 7.29. *Let ϕ be a spatial formula. Let $(\mathfrak{s}, \mathfrak{h})$ be a stack–heap model of size $n = |\mathfrak{h}|$ and let $\mathcal{M}_{\text{SMT}}$ be any corresponding SMT model. Let $\mathbf{Y} = \{\mathbf{y}_{\mathsf{ls}}, \mathbf{y}_{\mathsf{tree}}\}$ be fresh set variables and $\langle A, B, Z\rangle = T^{\mathfrak{s}}_n(\phi, \mathbf{Y})$. Then*

1. *$(\mathfrak{s}, \mathfrak{h}) \models \phi$ iff $\mathcal{M}_{\text{SMT}} \models \exists \mathbf{Y} \exists Z.\, A \wedge B \wedge \mathbf{X} = \mathbf{Y}$.*
2. *B satisfies $(\mathbf{Y} \cup Z)$-existence*
3. *A and B satisfy $(\mathbf{Y} \cup Z)$-equivalence.*

Proof. Let ϕ be a spatial formula without tree and ls predicates. We proceed by structural induction on ϕ.

$T^{\mathfrak{s}}_n(F), F \in \mathcal{F}_{\mathbf{Data}}$: The translation defines no fresh variables, i.e., $Z = \emptyset$, and so $(\mathbf{Y} \cup Z)$-existence is just $\mathbf{Y}$-existence. There is a unique interpretation of $\mathbf{Y}$ that satisfies B, namely $\mathbf{Y} = \emptyset$. Hence both $\mathbf{Y}$-existence and $\mathbf{Y}$-uniqueness are satisfied. Due to $\mathbf{SSL}^+_{\mathsf{data}}$ semantics we also have that $(\mathfrak{s}, \mathfrak{h}) \models F$ iff $\mathcal{M}_{\text{SMT}} \models F \wedge \mathbf{X} = \emptyset$ iff $\mathcal{M}_{\text{SMT}} \models \exists \mathbf{Y}.\, A \wedge B \wedge \mathbf{X} = \mathbf{Y}$.

$T^{\mathfrak{s}}_n(x \mapsto_{\mathsf{ls}} \langle y, d\rangle)$: Again, the translation defines no fresh variables, i.e., $Z = \emptyset$, and there is a unique interpretation of $\mathbf{Y}$ that satisfies B, namely $\mathbf{y}_{\mathsf{ls}} = \{\mathfrak{s}(x)\}$ and $\mathbf{y}_{\mathsf{tree}} = \emptyset$. Hence $\mathbf{Y}$-existence and $\mathbf{Y}$-uniqueness are satisfied. If $(\mathfrak{s}, \mathfrak{h}) \models x \mapsto_{\mathsf{ls}} y, d$, then $\mathfrak{h}(\mathfrak{s}(x)) = \langle \mathfrak{s}(y), \mathfrak{s}(d)\rangle$ and thus $\mathsf{n}(x) = \mathfrak{s}(y)$, $\mathsf{d}(x) = \mathfrak{s}(d)$. Because $\mathcal{M}_{\text{SMT}}$ corresponds to $(\mathfrak{s}, \mathfrak{h})$, $\mathsf{n}^{\mathcal{M}_{\text{SMT}}}(x^{\mathcal{M}_{\text{SMT}}}) = y^{\mathcal{M}_{\text{SMT}}}$ and $\mathsf{d}^{\mathcal{M}_{\text{SMT}}}(x^{\mathcal{M}_{\text{SMT}}}) = d^{\mathcal{M}_{\text{SMT}}}$, so $\mathcal{M}_{\text{SMT}} \models A$. Due to $\mathbf{Y}$-existence, also $\mathcal{M}_{\text{SMT}} \models \exists \mathbf{Y}.\, A \wedge B \wedge \mathbf{X} = \mathbf{Y}$.

For the other direction, if $\mathcal{M}_{\text{SMT}} \models \exists \mathbf{Y}.\, A \wedge B \wedge \mathbf{X} = \mathbf{Y}$, then $\mathbf{x}_{\mathsf{ls}}^{\mathcal{M}_{\text{SMT}}} = \mathbf{y}_{\mathsf{ls}}^{\mathcal{M}_{\text{SMT}}} = \{x^{\mathcal{M}_{\text{SMT}}}\} = \{\mathfrak{s}(x)\}$ and $\mathbf{x}_{\mathsf{tree}}^{\mathcal{M}_{\text{SMT}}} = \mathbf{y}_{\mathsf{tree}}^{\mathcal{M}_{\text{SMT}}} = \emptyset$. Because $\mathcal{M}_{\text{SMT}}$ corresponds to $(\mathfrak{s}, \mathfrak{h})$, this implies $\text{dom}(\mathfrak{h}) = \{\mathfrak{s}(x)\}$. Moreover, because A holds in $\mathcal{M}_{\text{SMT}}$, $\mathsf{n}^{\mathcal{M}_{\text{SMT}}}(x^{\mathcal{M}_{\text{SMT}}}) = y^{\mathcal{M}_{\text{SMT}}}$ and $\mathsf{d}^{\mathcal{M}_{\text{SMT}}}(x^{\mathcal{M}_{\text{SMT}}}) = d^{\mathcal{M}_{\text{SMT}}}$, so by model correspondence, $\mathfrak{h}(\mathfrak{s}(x)) = \langle \mathfrak{s}(y), \mathfrak{s}(d)\rangle$. It follows that $(\mathfrak{s}, \mathfrak{h}) \models x \mapsto_{\mathsf{ls}} \langle y_1, \ldots, y_k\rangle$.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

$\mathrm{T}^s_n(x \mapsto_{\mathsf{tree}} \langle y_1, y_2, d\rangle)$: Analogous to the previous case.

$\mathrm{T}^s_n(\mathsf{ds}(x, \mathbf{s}, \mathcal{P}))$: See Lemmas 7.24 and 7.25.

$\mathrm{T}^s_n(\phi_1 \star \phi_2)$: We introduce names for the parts of the sub-encodings.

$$\langle A_1, B_1, Z_1\rangle \triangleq \mathrm{T}^s_n(\phi_1, \mathbf{Y}_1),$$
$$\langle A_2, B_2, Z_2\rangle \triangleq \mathrm{T}^s_n(\phi_2, \mathbf{Y}_2),$$
$$Z \triangleq Z_1 \cup Z_2 \cup \mathbf{Y_1} \cup \mathbf{Y_2}.$$

By induction, there exist assignments to Z_1 and Z_2, $\mathbf{Y}_1$ and $\mathbf{Y_2}$ such that B_1 and B_2 are satisfied. Consequently, $B_1 \wedge \mathcal{B}_2$ satisfies Z-existence.

Also by induction, we have for $1 \leq i \leq 2$ that A_i cannot distinguish between different assignments that make the B_i true. Since the $(\mathbf{Y}_1 \cup Z_1) \cap (\mathbf{Y}_2 \cup Z_2) = \emptyset$, it follows that $A_1 \wedge A_2$ and $B_1 \wedge B_2$ satisfy Z-equivalence.

Because the variables $\mathbf{Y}$ are fresh and do not occur in the A_i and B_i, $(\mathbf{Y} \cup Z)$-existence and $(\mathbf{Y} \cup Z)$-equivalence follow for $A = A_1 \wedge A_2$ and $B = B_1 \wedge B_2 \wedge \mathbf{Y} = \mathbf{Y}_1 \cup \mathbf{Y}_2$.

Assume that $(\mathfrak{s}, \mathfrak{h}) \models \phi$. By the semantics of $\star$, there exist $\mathfrak{h}_1$ and $\mathfrak{h}_2$ such that $\mathfrak{h} = \mathfrak{h}_1 \uplus^{\mathfrak{s}} \mathfrak{h}_2$ and $(\mathfrak{s}, \mathfrak{h}_i) \models \phi_i$. By induction, $\mathcal{M}_{\mathrm{SMT}i} \models \exists \mathbf{Y}_i \exists Z_i.\, A_i \wedge B_i$, with $\mathcal{M}_{\mathrm{SMT}i}$ the model corresponding to $(\mathfrak{s}, \mathfrak{h}_i)$. By Lemma 7.28, we can assume that $\mathcal{M}_{\mathrm{SMT}} = \mathcal{M}_{\mathrm{SMT}1} \oplus \mathcal{M}_{\mathrm{SMT}2}$ is the first-order model associated with $(\mathfrak{s}, \mathfrak{h})$, and we have directly that $\mathcal{M}_{\mathrm{SMT}} \models \exists \mathbf{Y} \exists \mathbf{Y}_1 \exists Z_1 \exists \mathbf{Y}_2 \exists Z_2 . A_1 \wedge A_2 \wedge \mathsf{empty}(\mathbf{Y}_1 \cap \mathbf{Y}_2) \wedge B_1 \wedge B_2 \wedge \mathbf{Y} = \mathbf{Y}_1 \cup \mathbf{Y}_2 \cup \mathbf{X} = \mathbf{Y}$. For the other direction, assume that $\mathcal{M}_{\mathrm{SMT}} \models \exists \mathbf{Y} \exists \mathbf{Y}_1 \exists Z_1 \exists \mathbf{Y}_2 \exists Z_2.\, A_1 \wedge A_2 \wedge \mathbf{Y}_1 \cap \mathbf{Y}_2 = \emptyset \wedge B_1 \wedge B_2 \wedge \mathbf{Y} = \mathbf{Y}_1 \cup \mathbf{Y}_2 \wedge \mathbf{x}_{\mathsf{global}} = \mathbf{Y}$. Let $\mathbf{Y}_i^{\exists}$ be the values that witness above existential satisfiability. Observe that A_i and B_i refer only to fresh variables $\mathbf{Y_i}$ and Z_i, for $i \in \{1, 2\}$. We can therefore decompose $\mathcal{M}_{\mathrm{SMT}}$ into $\mathcal{M}_{\mathrm{SMT}1} \oplus \mathcal{M}_{\mathrm{SMT}2}$, where $\mathcal{M}_{\mathrm{SMT}i}$ interprets variables $\mathbf{Y}_i$ and Z_i and interprets $\mathbf{x}_{\mathsf{global}}$ as $\mathbf{Y}_i^{\exists}$. We have that $\mathcal{M}_{\mathrm{SMT}i} \models \exists \mathbf{Y}_i \exists Z_i\, .A_i \wedge B_i \wedge \mathbf{x}_{\mathsf{global}} = \mathbf{Y}_i$.

By Lemma 7.17 we can assume that $\mathcal{M}_{\mathrm{SMT}i}$ is the first-order model associated with some $\mathbf{SSL}^{+}_{\mathsf{data}}$ model $(\mathfrak{s}, \mathfrak{h}_i)$, for $i \in \{1, 2\}$. By the induction hypotheses, we have that $(\mathfrak{s}, \mathfrak{h}_i) \models \phi_i$. By the semantics of $\star$, we have for $\mathfrak{h} = \mathfrak{h}_1 \uplus^{\mathfrak{s}} \mathfrak{h}_2$ that $(\mathfrak{s}, \mathfrak{h}) \models \phi_1 \star \phi_2$. (Note that $\uplus^{\mathfrak{s}}$ is defined because $\mathbf{Y}_1 \cap \mathbf{Y}_2 = \emptyset$ holds in the SMT model $\mathcal{M}_{\mathrm{SMT}}$.) Moreover, by Lemma 7.28, we can assume that $\mathcal{M}_{\mathrm{SMT}}$ corresponds to corresponds to $\mathcal{M}_{\mathrm{SMT}}$. □

Lemma 7.30 (Correctness of the SMT encoding)**.** *Let $\phi \in \mathbf{SSL}^{+}_{\mathsf{data}}$ and let $T^b_n(\phi) = \langle A, B, Z\rangle$. $T_n(\phi)$ satisfies correctness, Z-existence and Z-equivalence.*

Proof. Again, we proceed by structural induction and analyze the relevant cases:

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Wien Bibliothek Your knowledge hub

- If ϕ is a spatial formula, the result is an immediate consequence of Lemma 7.29, noting that $(\mathbf{Y} \cup Z)$-existence implies Z-existence and $(\mathbf{Y} \cup Z)$-equivalence implies Z-equivalence.

- If $\phi = \phi_1 \wedge \phi_2$, or $\phi = \phi_1 \vee \phi_2$, the results follows immediately by induction. Note that both for conjunctions and disjunctions, B_1 and B_2 are conjoined, guaranteeing that both existence and equivalence are satisfied by $\mathsf{T}^b_n(\phi)$.

- If $\phi = \neg\phi_1$ then, by induction, ϕ_1 satisfies Z-equivalence and Z-existence. Since B and Z are not changed by $\mathsf{T}^b_n(\neg\phi_1)$, Z-existence is preserved. Since $\mathcal{M}_{\text{SMT}} \models \mathsf{T}^b_n(\neg\phi_1)$ iff $\mathcal{M}_{\text{SMT}} \not\models \mathsf{T}^b_n(\phi_1)$, the truth value of the A-component is simply flipped, so Z-equivalence is preserved as well. It then follows by Lemma 7.18 that $(\mathfrak{s}, \mathfrak{h}) \models \neg\phi_1$ iff $\mathcal{M}_{\text{SMT}} \models \exists Z.\, \neg A \wedge B$, yielding correctness. □

CORRECTNESS OF THE DATA-STRUCTURE ENCODING. We now have all the tools to show that Theorem 7.19 holds.

Proof of Theorem 7.19. We first show that ϕ is $\mathbf{SSL}^+_{\mathsf{data}}$-satisfiable if and only if the SMT translation $F = \mathsf{T}_n(\phi)$ is satisfiable, for $n \triangleq \mathsf{bound}(\phi)$. Since n is an upper bound on the size of the minimal model of ϕ by Lemma 7.12, ϕ is satisfiable if and only if it is satisfiable in a model with at most n allocated locations. It follows from Lemma 7.30 that ϕ and $\exists Z.\, \mathsf{T}^b_n(\phi) \wedge \Delta^n_{\mathbf{SSL}}$ are satisfiability equivalent. Since $\mathsf{T}_n(\phi) = \mathsf{T}^b_n(\phi) \wedge \Delta^n_{\mathbf{SSL}}$, the desired satisfiability equivalence between ϕ and $\mathsf{T}_n(\phi)$ follows.

It remains to be shown that the reduction is polynomial. The size of the encoding $\mathsf{T}_n(\phi)$ is dominated by the encoding of the tree and ls predicates, which in turn are dominated by the formula $\mathsf{reachability}$. As the formula $\mathsf{reachability}$ is of order $\mathcal{O}(n^4)$, the total size of the encoding of ϕ is bounded by $\mathcal{O}((n_{\mathsf{ls}} + n_{\mathsf{tree}})n^4)$. Since n is itself polynomial in $|\phi|$ by Lemma 7.12, the encoding is polynomial in $|\phi|$. □

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Part III

DECIDING SEPARATION LOGIC WITH INDUCTIVE DEFINITIONS

I study separation logic with a powerful mechanism for modeling custom data structures by inductive definitions. I give an asymptotically optimal 2ExpTime decision procedure for the entailment problem of a quantifier-free separation logic in which negation, septraction and magic wand are *guarded* and in which inductive definitions are restricted to $\mathbf{ID}_{\mathsf{btw}}$ [IRS13]. $\mathbf{ID}_{\mathsf{btw}}$ imposes restrictions on inductive definitions that guarantee that all models are of *bounded treewidth*. The resulting formalism is expressive enough to, for example, reason about trees with linked leaves, a data structure used to implement sorted sets. Some of the work presented in this part was previously published in [Jan+17; Kat+18; KMZ19a; PMZ20; PZ20a].

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.

8 SEPARATION LOGIC WITH INDUCTIVE DEFINITIONS

In this part of the thesis, I study *separation logic with inductive definitions* (SLID). In contrast to Part ii, we no longer assume that predicates for data structures, such as ls and tree, are built into the logic. Instead, the user of SLID can define custom predicates by providing a set of recursive equations called a *system of inductive definitions* (SID). Our main focus will be on (an extension of) the decidable fragment $\mathbf{SLID}_{\mathsf{btw}}$ [IRS13], which I introduced informally in Chapter 3 and will formalize later in the present chapter.

OUTLINE. In this chapter, I introduce the syntax and semantics of SLID and define the *guarded fragment* $\mathbf{SLID}^{\mathsf{g}}_{\mathsf{btw}}$, the extension of $\mathbf{SLID}_{\mathsf{btw}}$ that is my main object of study. The following three chapters center around an abstraction for $\mathbf{SLID}^{\mathsf{g}}_{\mathsf{btw}}$ formulas, Φ*-types*. Specifically, Chapter 9 contains an informal motivation of the abstraction, Chapter 10 introduces Φ*-forests* and their *projections* onto formulas, and Chapter 11 uses such forest projections to define the Φ-type abstraction. A decision procedure for $\mathbf{SLID}^{\mathsf{g}}_{\mathsf{btw}}$ based on the Φ-type abstraction follows in Chapter 12. In Chapter 13, I conclude this part of the thesis by proving that all logics that all extensions of $\mathbf{SLID}^{\mathsf{g}}_{\mathsf{btw}}$ that lift one of the *guardedness* restrictions are undecidable.

HOW PART II RELATES TO [JAN+17; KAT+18; KMZ19A; PMZ20; PZ20A]. We began studying abstraction-based entailment checking for SLID, in the guise of *compositional heap automata* and *robustness properties*, in [Jan+17]. In that paper, we provided anecdotal evidence that our approach might lend itself to deciding entailment in large fragments of SLID. In [KMZ19a], we proposed the Φ*-profile* abstraction, including an implementation in our `Harrsh` tool [Kat+18], claiming that this approach yields a 2EXPTIME decision procedure for $\mathbf{SLID}_{\mathsf{btw}}$.

Unfortunately, we later discovered that the approach of [KMZ19a] is incomplete, which led to the development of the Φ*-type* abstraction introduced in [PMZ20]. Chapters 10 to 12 are based on this article. Our recent article [PZ20a] contains a more accessible introduction to the Φ-type abstraction, which has informed both Chapter 9 and the presentation of Chapters 10 to 12. Further, [PZ20a] contains an abridged version of the undecidability proofs in Chapter 13.

To sum up, this part of the thesis has grown out of joint work with Christina Jansen, Christoph Matheja, Thomas Noll and Florian Zuleger [Jan+17; Kat+18; KMZ19a; PMZ20; PZ20a]. Sections that are

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

$$
\begin{array}{lll}
 & & u, v, u_i, v_i', \ldots \in \mathbf{Var} \cup \mathbf{Loc}, \\
 & & \mathbf{u}, \mathbf{v}, \mathbf{w_i}, \ldots \in (\mathbf{Var} \cup \mathbf{Loc})^*, \mathbf{e}, \mathbf{a_i} \in \mathbf{Var}^* \\
\tau & ::= & \mathbf{emp} \mid u \mapsto \mathbf{v} \mid \mathsf{pred}(\mathbf{u}) \mid u \approx v \mid u \not\approx v \\
\phi_{\mathsf{g}} & ::= & \tau \mid \phi_{\mathsf{g}} \star \phi_{\mathsf{g}} \mid \phi_{\mathsf{g}} \wedge \phi_{\mathsf{g}} \mid \phi_{\mathsf{g}} \vee \phi_{\mathsf{g}} \\
 & & \mid \phi_{\mathsf{g}} \wedge \neg \phi_{\mathsf{g}} \mid \phi_{\mathsf{g}} \wedge (\phi_{\mathsf{g}} \circledast\!\!- \phi_{\mathsf{g}}) \mid \phi_{\mathsf{g}} \wedge (\phi_{\mathsf{g}} -\!\!\star \phi_{\mathsf{g}}) \\
\phi_{\mathsf{qf}} & ::= & \tau \mid \phi_{\mathsf{qf}} \star \phi_{\mathsf{qf}} \mid \phi_{\mathsf{qf}} -\!\!\star \phi_{\mathsf{qf}} \mid \phi_{\mathsf{qf}} \wedge \phi_{\mathsf{qf}} \mid \phi_{\mathsf{qf}} \vee \phi_{\mathsf{qf}} \mid \neg \phi_{\mathsf{qf}} \\
\phi_{\mathsf{sh}} & ::= & \exists \mathbf{e}.\, (u_1 \mapsto \mathbf{v}_1) \star \cdots \star (u_k \mapsto \mathbf{v}_k) \\
 & & \qquad \star\, \mathsf{pred}_1(\mathbf{w}_1) \star \cdots \star \mathsf{pred}_l(\mathbf{w}_l) \star \Pi, \\
 & & \text{with } \Pi = u_1 \approx v_1 \star \cdots \star u_m \approx v_m \star u_1' \not\approx v_1' \star \cdots \star u_n' \not\approx v_n'
\end{array}
$$

Figure 8.1: The syntax of the separation-logic fragments studied in this part of the thesis: Guarded formulas ϕ_{g}, collected in $\mathbf{SLID}^{\mathsf{g}}$; quantifier-free formulas ϕ_{qf}, collected in $\mathbf{SLID}^{\mathsf{qf}}$; and existentially-quantified symbolic heaps, ϕ_{sh}, collected in $\mathbf{SH}^{\exists}$.

due in large part to one of my contributors contain an explicit attribution.

8.1 SYNTAX OF SEPARATION LOGIC WITH INDUCTIVE DEFINITIONS

We assume a set **Preds** of *predicate identifiers*. Each predicate $\mathsf{pred} \in \mathbf{Preds}$ is equipped with an arity $\mathsf{ar}(\mathsf{pred}) \in \mathbb{N}$, representing the number of parameters to be passed to the predicate.

The grammar in Fig. 8.1 defines three variants of separation logic with inductive definitions:

- *Guarded* quantifier-free separation logic, formulas of the form ϕ_{g}, collected in the set $\mathbf{SLID}^{\mathsf{g}}$.
- Quantifier-free separation logic, formulas of the form ϕ_{qf}, collected in $\mathbf{SLID}^{\mathsf{qf}}$.
- *Existentially-quantified symbolic heaps*, $\mathbf{SH}^{\exists}$.

In Fig. 8.1, $\mathsf{pred} \in \mathbf{Preds}$ is a predicate identifier and $|\mathbf{u}| = \mathsf{ar}(\mathsf{pred})$.

The first line of Fig. 8.1 defines the atomic formulas, τ, common to all three SL variants. As with the first-order separation logic $\mathbf{SL}_{\mathbf{base}}$ from Chapter 2, $\mathbf{emp}$ is the *empty heap*, $x \mapsto \mathbf{y}$ asserts that x *points to* the locations captured by $\mathbf{y}$, $x \approx y$ asserts the equality between variables x and y, and $x \not\approx y$ asserts the disequality of x and y. Guarded formulas, ϕ_{g}, are built from atomic formulas using the separating conjunction $\star$, conjunction $\wedge$, disjunction $\vee$, *guarded negation* $\phi_{\mathsf{g}} \wedge \neg\phi_{\mathsf{g}}$, *guarded septraction* $\phi_{\mathsf{g}} \wedge (\phi_{\mathsf{g}} \circledast\!\!- \phi_{\mathsf{g}})$, and *guarded magic wands* $\phi_{\mathsf{g}} \wedge (\phi_{\mathsf{g}} -\!\!\star \phi_{\mathsf{g}})$.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

In (not necessarily guarded) quantifier-free formulas, ϕ_{qf}, negation and magic wands may occur unguarded. Moreover, we omit the septraction operator in ϕ_{qf}, as it is definable as $\phi \circledast \psi \triangleq \neg(\phi \mathrel{-\!\!*} \neg\psi)$.

Finally, ϕ_{sh} formulas are existentially-quantified symbolic heaps, consisting of points-to assertions, predicate calls, and a *pure constraint*, i.e., a conjunction of (dis-)equalities. In the literature, the pure constraint, Π, is usually defined with the classical conjunction $\wedge$ [BCO04]. Just like in Parts i and ii of the thesis, we instead use the separating conjunction $\star$ to express the pure constraint, because our semantics of (dis-)equalities, defined in Section 8.2, forces the heap to be empty in models of (dis-)equalities.

As in Part ii, $\bigstar\{\psi_1, \ldots, \psi_k\}$ is the formula $\psi_1 \star \cdots \star \psi_k$.

Example 8.1 (Syntax of SLID). *Assume the predicate* $\mathsf{lseg}(x_1, x_2)$ *is interpreted by the set of all nonempty list segments from* x_1 *to* x_2.

1. *The symbolic heap* $\exists y.\, \mathsf{lseg}(x,y) \star \mathsf{lseg}(z,y) \star \mathsf{lseg}(y,\mathsf{nil}) \in \mathbf{SH}^{\exists}$ *states that the heap consists of two overlaid null-terminated lists, one with head* x*, the other with head* z.

2. *The guarded formula* $\mathsf{lseg}(x,y) \wedge \neg x \mapsto y \in \mathbf{SLID}^{\mathsf{g}}_{\mathsf{btw}}$ *states that the heap consists of a list of length at least two.*

3. *The guarded formula* $\mathsf{lseg}(x,y) \wedge (\mathsf{ls}(y,z) \circledast \mathsf{ls}(x,x)) \in \mathbf{SLID}^{\mathsf{g}}_{\mathsf{btw}}$ *states that the heap consists of a list segment from* x *to* y *that can be extended to a cyclic list by adding a (nonempty) list from* y *to* z*. This formula is only satisfied by models in which* $x \approx z$ *holds.*

4. *The unguarded formula* $(x \mapsto \mathsf{nil}) \mathrel{-\!\!*} x \not\approx x \in \mathbf{SLID}^{\mathsf{qf}}_{\mathsf{btw}}$ *states that in all models obtained by extending the current model with a pointer from* x *to* nil*,* x *is different from* x*. As it is impossible that* x *is different from* x*, the formula is only satisfied by models in which there is* no *way to extend the heap by a pointer from* x *to* nil*, i.e., by models in which* x *is already allocated.*

An atom representing *true*?

It is quite common to include an atomic formula that holds in every model, often denoted *true* or t [Rey02], even in symbolic-heap fragments [Cal+11]. We do *not* include a spatial atom *true* in any of the logics; while *true* can of course be defined in $\mathbf{SLID}^{\mathsf{qf}}$, for example by $\mathbf{emp} \vee \neg\mathbf{emp}$, it is *not* definable in $\mathbf{SLID}^{\mathsf{g}}$ because of the aforementioned semantics of (dis-)equalities. This is crucial: If *true* were definable, $\mathbf{SLID}^{\mathsf{qf}}$ and $\mathbf{SLID}^{\mathsf{g}}$ would be equivalent, because we could simply use *true* for all guards.

LOCATIONS IN FORMULAS. I make the nonstandard choice to allow locations as terms in formulas. While I make this choice purely for technical convenience, it is in fact quite natural when considering

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

languages such as C that allow accessing concrete memory addresses. Given a formula ϕ, we denote by $\mathsf{locs}(\phi)$ the set of all locations that occur in ϕ.

INDUCTIVE DEFINITIONS. Predicates are defined by a *system of inductive definitions* (SID). An SID is a finite set Φ of *rules* of the form $\mathsf{pred}(\mathbf{x}) \Leftarrow \phi$, where $\mathsf{pred} \in \mathbf{Preds}$ is a predicate symbol, $\mathbf{x}$ are the *parameters* of pred, and—as is standard [IRS13; Ant+14]—$\phi \in \mathbf{SH}^{\exists}$ is an existentially-quantified symbolic heap as defined in Fig. 8.1. We assume that all rules of the same predicate pred have the same parameters. We collect these *free variables* of pred in the set $\mathsf{fvars}(\mathsf{pred})$. We collect all predicates that occur in SID Φ in the set $\mathbf{Preds}(\Phi)$. The size of an SID Φ, $|\Phi|$, is the sum of the sizes of the formulas in its rules. We give a few examples of SIDs; the formal semantics of SIDs will be defined in Section 8.2.

Example 8.2 (SID). *1. Let Φ_{ls} be the following SID.*

$$\begin{aligned}
\mathsf{lseg}(x_1, x_2) &\Leftarrow x_1 \mapsto x_2 \\
\mathsf{lseg}(x_1, x_2) &\Leftarrow \exists y.\, x_1 \mapsto y \star \mathsf{lseg}(y, x_2) \\
\mathsf{ls}(x_1) &\Leftarrow x_1 \mapsto \mathsf{nil} \\
\mathsf{ls}(x_1) &\Leftarrow \exists y.\, (x_1 \mapsto y) \star \mathsf{ls}(y)
\end{aligned}$$

*The predicates lseg and ls correspond to the built-in ls predicate of the logic **SSL** from Part ii with one and zero holes, respectively, except that they do* not *allow empty lists. The formulas $\mathsf{lseg}(x_1, \mathsf{nil})$ and $\mathsf{ls}(x_1)$ are equivalent w.r.t. Φ_{ls}.*

2. *Let $\Phi_{\mathsf{odd/even}}$ be the following SID.*

$$\begin{aligned}
\mathsf{odd}(x_1, x_2) &\Leftarrow x_1 \mapsto x_2 \\
\mathsf{odd}(x_1, x_2) &\Leftarrow \exists y.\, (x_1 \mapsto y) \star \mathsf{even}(y, x_2) \\
\mathsf{even}(x_1, x_2) &\Leftarrow \exists y.\, (x_1 \mapsto y) \star \mathsf{odd}(y, x_2)
\end{aligned}$$

$\Phi_{\mathsf{odd/even}}$ defines all lists of odd and even length, respectively.

3. *Let Φ_{tree} be the following SID.*

$$\begin{aligned}
\mathsf{tree}(x_1) &\Leftarrow x_1 \mapsto \langle \mathsf{nil}, \mathsf{nil} \rangle \\
\mathsf{tree}(x_1) &\Leftarrow \exists \langle l, r \rangle \,.\, (x_1 \mapsto \langle l, r \rangle) \star \mathsf{tree}(l) \star \mathsf{tree}(r)
\end{aligned}$$

*Φ_{tree} defines the set of null-terminated binary trees, equivalent to trees without holes in **SSL**.*

INSTANTIATING VARIABLES. Let ϕ be a formula and let $\mathbf{y}, \mathbf{z} \in (\mathbf{Var} \cup \mathbf{Loc})^*$ be sequences of the same length, $\mathbf{y}$ repetition free. We de-

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

note by $\phi[\mathbf{y}/\mathbf{z}]$ the instantiation of $\mathbf{y} = \langle y_1, \ldots, y_k \rangle$ by $\mathbf{z} = \langle z_1, \ldots, z_k \rangle$ in ϕ. Formally,

$$
\begin{array}{lll}
y_i[\mathbf{y}/\mathbf{z}] & \triangleq & z_i, \quad 1 \leq i \leq k \\
a[\mathbf{y}/\mathbf{z}] & \triangleq & a, \quad \text{iff } a \notin \mathbf{y} \\
\langle a_1, \ldots, a_k \rangle[\mathbf{y}/\mathbf{z}] & \triangleq & \langle a_1[\mathbf{y}/\mathbf{z}], \ldots, a_k[\mathbf{y}/\mathbf{z}] \rangle \\
\mathbf{emp}[\mathbf{y}/\mathbf{z}] & \triangleq & \mathbf{emp} \\
a \mapsto \mathbf{b}[\mathbf{y}/\mathbf{z}] & \triangleq & a[\mathbf{y}/\mathbf{z}] \mapsto \mathbf{b}[\mathbf{y}/\mathbf{z}] \\
\mathsf{pred}(\mathbf{x})[\mathbf{y}/\mathbf{z}] & \triangleq & \mathsf{pred}(\mathbf{x}[\mathbf{y}/\mathbf{z}]) \\
\neg\phi[\mathbf{y}/\mathbf{z}] & \triangleq & \neg(\phi[\mathbf{y}/\mathbf{z}]) \\
\phi \times \psi[\mathbf{y}/\mathbf{z}] & \triangleq & \phi[\mathbf{y}/\mathbf{z}] \times \psi[\mathbf{y}/\mathbf{z}], \text{ where } \times \in \{\wedge, \vee, \star, -\!\!\star, \circledast\} \\
\exists e.\, \phi[\mathbf{y}/\mathbf{z}] & \triangleq & \exists e[\mathbf{y}/\mathbf{z}].\, \phi[\mathbf{y}/\mathbf{z}]
\end{array}
$$

For example, for $\mathbf{y} = \langle x \rangle$ and $\mathbf{z} = \langle w \rangle$, we have $(x \mapsto v)[\mathbf{y}/\mathbf{z}] = w \mapsto v$; and $\exists x.\, \mathsf{ls}(z, x)[\mathbf{y}/\mathbf{z}] = \exists w.\, \mathsf{ls}(z, w)$.

Finally, assuming we have an order on the free variables of a formula (as we do with predicate calls, for example) we abbreviate $\phi(\mathbf{z}) \triangleq \phi[\mathsf{fvars}(\phi)/\mathbf{z}]$.

8.2 SEMANTICS OF SEPARATION LOGIC WITH INDUCTIVE DEFINITIONS

As usual, we define the semantics in terms of stack–heap pairs, $(\mathfrak{s}, \mathfrak{h})$ (cf. Chapter 2). In this part of the thesis, we only consider variants of SLID *without* data constraints. We thus define $\mathbf{Val} \triangleq \mathbf{Loc}$ for this part of the thesis. Given this assumption, we have $\mathfrak{s}\colon \mathbf{Var} \rightharpoonup \mathbf{Loc}$ and $\mathfrak{h}\colon \mathbf{Loc} \rightharpoonup \mathbf{Loc}^+$, i.e., all values stored in the stack and the heap are locations. Figure 8.2 defines the semantics of separation logic formulas ϕ w.r.t. a fixed SID Φ. To simplify the technical development in later chapters, we deviate from the standard presentation of the semantics (see e.g. [Rey02]) by syntactically replacing variables by their corresponding heap locations. This is possible because we allow locations as terms in formulas. When writing $\phi[\mathrm{dom}(\mathfrak{s}) / \mathrm{img}(\mathfrak{s})]$, I assume that an arbitrary but consistent order is imposed on the domain and image of the stack, i.e., every variable $x \in \mathrm{dom}(\mathfrak{s})$ is replaced by $\mathfrak{s}(x)$.

As in Parts i and ii, the semantics of equalities and disequalities require that the heap is empty. This ensures that *true* is not definable in guarded formulas (e.g., as $x \approx x$). Apart from this choice, our semantics is equivalent to the standard semantics of separation logic with inductive definitions [IRS13; Ant+14] on formulas that do not contain location terms.

Just like in Parts i and ii, we use a *precise* [COY07] semantics of the points-to assertion: $(\mathfrak{s}, \mathfrak{h}) \models_\Phi x \mapsto \mathbf{y}$ holds only in single-pointer heaps.

A heap is a model of a predicate call $\mathsf{pred}(\mathbf{v})$ iff it is a model of a rule $\mathsf{pred}(\mathbf{x}) \Leftarrow \psi$, in which the parameters $\mathbf{x}$ have been replaced

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

$(\mathfrak{s}, \mathfrak{h}) \models_\Phi \phi$	iff	$\mathsf{fvars}(\phi) \subseteq \mathrm{dom}(\mathfrak{s})$ and $\mathfrak{h} \models_\Phi \phi[\mathrm{dom}(\mathfrak{s}) / \mathrm{img}(\mathfrak{s})]$
$\mathfrak{h} \models_\Phi \mathbf{emp}$	iff	$\mathrm{dom}(\mathfrak{h}) = \emptyset$
$\mathfrak{h} \models_\Phi \ell_1 \approx \ell_2$	iff	$\mathrm{dom}(\mathfrak{h}) = \emptyset$ and $\ell_1 = \ell_2$
$\mathfrak{h} \models_\Phi \ell_1 \not\approx \ell_2$	iff	$\mathrm{dom}(\mathfrak{h}) = \emptyset$ and $\ell_1 \neq \ell_2$
$\mathfrak{h} \models_\Phi v \mapsto w$	iff	$\mathfrak{h} = \{v \mapsto w\}$
$\mathfrak{h} \models_\Phi \mathsf{pred}(\mathbf{v})$	iff	$\mathfrak{h} \models_\Phi \phi[\mathbf{x}/\mathbf{v}]$ for some rule $(\mathsf{pred}(\mathbf{x}) \Leftarrow \psi) \in \Phi$
$\mathfrak{h} \models_\Phi \phi_1 \star \phi_2$	iff	ex. $\mathfrak{h}_1, \mathfrak{h}_2$ s.t. $\mathfrak{h} = \mathfrak{h}_1 + \mathfrak{h}_2$ and $\mathfrak{h}_1 \models_\Phi \phi_1$ and $\mathfrak{h}_2 \models_\Phi \phi_2$
$\mathfrak{h} \models_\Phi \phi_1 \mathrel{-\!\!\star} \phi_2$	iff	for all $\mathfrak{h}_1, \mathfrak{h}_2$ s.t. $\mathfrak{h}_2 = \mathfrak{h}_1 + \mathfrak{h}$ it holds that if $\mathfrak{h}_1 \models_\Phi \phi_1$ then $\mathfrak{h}_2 \models_\Phi \phi_2$
$\mathfrak{h} \models_\Phi \phi_1 \wedge \phi_2$	iff	$\mathfrak{h} \models_\Phi \phi_1$ and $\mathfrak{h} \models_\Phi \phi_2$
$\mathfrak{h} \models_\Phi \phi_1 \vee \phi_2$	iff	$\mathfrak{h} \models_\Phi \phi_1$ or $\mathfrak{h} \models_\Phi \phi_2$
$\mathfrak{h} \models_\Phi \neg\phi_1$	iff	$\mathfrak{h} \not\models_\Phi \phi_1$
$\mathfrak{h} \models_\Phi \exists e.\, \phi$	iff	exists $v \in \mathbf{Loc}$ s.t. $\mathfrak{h} \models_\Phi \phi[e/v]$

Figure 8.2: Semantics of separation logic with inductive definitions.

by the actual arguments, $\mathbf{v}$. Note that this semantics of predicates corresponds to the least fixed-point semantics as formalized e.g. in [Bro+14].

In this part of the thesis, we use the standard, "weak" semantics of the separating conjunction: $\mathfrak{h} \models_\Phi \phi_1 \star \phi_2$ iff $\mathfrak{h}$ can be split into disjoint heaps that are models of ϕ_1 and ϕ_2; and $\mathfrak{h} \models_\Phi \phi_1 \mathrel{-\!\!\star} \phi_2$ iff extending $\mathfrak{h}$ with a model of ϕ_1 always yields a model of ϕ_2, provided the extension is defined. The semantics of the (classical) Boolean connectives and the existential $\exists$ are standard. In particular, $\exists$ corresponds to stack extension, even though this is not explicit in our semantics.

Lemma 8.3. *Let $\phi \in \mathbf{SH}^{\exists}$ and $e \in \mathbf{Var}$. Let $(\mathfrak{s}, \mathfrak{h})$ be a stack–heap pair with $e \notin \mathrm{dom}(\mathfrak{s})$. Then $(\mathfrak{s}, \mathfrak{h}) \models_\Phi \exists e.\, \phi$ if and only if there exists a location $v \in \mathbf{Loc}$ such that $(\mathfrak{s} \cup \{e \mapsto v\}, \mathfrak{h}) \models_\Phi \phi$.*

Proof. $(\mathfrak{s}, \mathfrak{h}) \models_\Phi \exists e.\, \phi$

iff $\mathfrak{h} \models_\Phi (\exists e.\, \phi)[\mathrm{dom}(\mathfrak{s}) / \mathrm{img}(\mathfrak{s})]$

iff $\mathfrak{h} \models_\Phi \exists e.\, (\phi[\mathrm{dom}(\mathfrak{s}) / \mathrm{img}(\mathfrak{s})])$

iff ex. $v \in \mathbf{Loc}$ s.t. $\mathfrak{h} \models_\Phi \phi[\mathrm{dom}(\mathfrak{s}) / \mathrm{img}(\mathfrak{s})][e/v]$

iff ex. $v \in \mathbf{Loc}$ s.t. $\mathfrak{h} \models_\Phi \phi[\mathrm{dom}(\mathfrak{s}) \cdot \langle e \rangle \,/\, \mathrm{img}(\mathfrak{s}) \cdot \langle v \rangle]$

iff ex. $v \in \mathbf{Loc}$ s.t.
$\mathfrak{h} \models_\Phi \phi[\mathrm{dom}(\mathfrak{s} \cup \{e \mapsto v\}) / \mathrm{img}(\mathfrak{s} \cup \{e \mapsto v\})]$

iff ex. $v \in \mathbf{Loc}$ s.t. $(\mathfrak{s} \cup \{e \mapsto v\}, \mathfrak{h}) \models_\Phi \phi$ □

SATISFIABILITY AND ENTAILMENT. As the semantics of an SLID formula depends on the given SID, the satisfiability and entailment

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

problems are parameterized by an SID. Let ϕ, ψ be SLID formulas with $\mathsf{fvars}(\psi) \subseteq \mathsf{fvars}(\phi)$. We say that ϕ is *satisfiable* w.r.t. SID Φ if there exists a model $(\mathfrak{s}, \mathfrak{h})$ such that $(\mathfrak{s}, \mathfrak{h}) \models_\Phi \phi$. We say that ϕ *entails* ψ w.r.t. Φ, denoted $\phi \models_\Phi \psi$, iff for all models $(\mathfrak{s}, \mathfrak{h})$, $(\mathfrak{s}, \mathfrak{h}) \models_\Phi \phi$ implies $(\mathfrak{s}, \mathfrak{h}) \models_\Phi \psi$.

ISOMORPHISM. Formulas without location terms cannot distinguish between isomorphic models. I restate the definition of isomorphism, simplifying it (compared to Definition 2.2) based on the identity **Val** = **Loc** that holds in this part of the thesis.

Definition 8.4 (Isomorphic stack–heap pairs)**.** *Two stack–heap pairs* $(\mathfrak{s}, \mathfrak{h})$ *and* $(\mathfrak{s}', \mathfrak{h}')$ *are* isomorphic, *written* $(\mathfrak{s}, \mathfrak{h}) \cong (\mathfrak{s}', \mathfrak{h}')$, *if there exists a bijection* $\sigma \colon (\mathsf{locs}(\mathfrak{h}) \cup \mathsf{img}(\mathfrak{s})) \to (\mathsf{locs}(\mathfrak{h}') \cup \mathsf{img}(\mathfrak{s}'))$ *such that (1) for all* x, $\mathfrak{s}'(x) = \sigma(\mathfrak{s}(x))$ *and (2)* $\mathfrak{h}' = \{\sigma(l) \mapsto \sigma(\mathfrak{h}(l)) \mid l \in \mathrm{dom}(\mathfrak{h})\}$.

Lemma 8.5. *Let* ϕ *be an SLID formula with* $\mathsf{locs}(\phi) = \emptyset$. *Let* $(\mathfrak{s}, \mathfrak{h})$ *and* $(\mathfrak{s}', \mathfrak{h}')$ *be models with* $(\mathfrak{s}, \mathfrak{h}) \cong (\mathfrak{s}', \mathfrak{h}')$. *Then* $(\mathfrak{s}, \mathfrak{h}) \models_\Phi \phi$ *iff* $(\mathfrak{s}', \mathfrak{h}') \models_\Phi \phi$.

Proof sketch. Let σ be an isomorphism between $(\mathfrak{s}, \mathfrak{h})$ and $(\mathfrak{s}', \mathfrak{h}')$. Then:

$$
\begin{aligned}
&(\mathfrak{s}, \mathfrak{h}) \models_\Phi \phi \\
\text{iff } & \mathfrak{h} \models_\Phi \phi[\mathrm{dom}(\mathfrak{s}) / \mathrm{img}(\mathfrak{s})] \\
\text{iff } & \mathfrak{h}' \models_\Phi \phi[\mathrm{dom}(\mathfrak{s}) / \mathrm{img}(\mathfrak{s})][\mathrm{dom}(\sigma) / \mathrm{img}(\sigma)] && (\dagger) \\
\text{iff } & \mathfrak{h}' \models_\Phi \phi[\mathrm{dom}(\mathfrak{s}') / \mathrm{img}(\mathfrak{s}')] && (\text{because } \mathsf{locs}(\phi) = \emptyset) \\
\text{iff } & (\mathfrak{s}', \mathfrak{h}') \models_\Phi \phi.
\end{aligned}
$$

If we wanted to be fully formal, we would have to prove the step marked $(\dagger)$ by a structural induction on ϕ. □

8.3 SIDS WITH MODELS OF BOUNDED TREEWIDTH

As discussed in Section 3.3, the entailment problem even of the symbolic-heap fragment of SLID is undecidable in general [Ant+14; IRV14]. To obtain a decidable logic, we restrict SIDs to the fragment $\mathbf{ID}_{\mathrm{btw}}$ that was introduced by Iosif et al. in [IRS13]. $\mathbf{ID}_{\mathrm{btw}}$ imposes restrictions on SIDs that guarantee that all models of the predicates of the SID are of *bounded treewidth*.[1]

The relationship between $\mathbf{ID}_{\mathrm{btw}}$ and monadic second-order logic over graphs of bounded treewidth.

The subscript btw is slightly misleading, as $\mathbf{ID}_{\mathrm{btw}}$ does *not* have the same expressive power as MSO over connected graphs of bounded treewidth. It is easy to find MSO formulas that do not have equivalent $\mathbf{ID}_{\mathrm{btw}}$ formulas. For example, the progress

1 More precisely, when viewed as graphs, all models of the SIDs in $\mathbf{ID}_{\mathrm{btw}}$ have bounded treewidth. For a definition of treewidth, see e.g. [Die16].

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

property of $\mathbf{ID}_{\mathsf{btw}}$ only allows “forward edges” starting in a free variable, whereas MSO can also model “backward edges.” It is an open question whether there is a separation logic that captures MSO over connected graphs of bounded treewidth.

In $\mathbf{ID}_{\mathsf{btw}}$, all SIDs have to satisfy *progress*, *connectivity*, and *establishment*. To formalize these assumptions, we introduce a couple of auxiliary definitions for symbolic heaps ϕ: the *local allocation of* ϕ, $\mathsf{lalloc}(\phi)$; and the *local references of* ϕ, $\mathsf{lref}(\phi)$. In the following definition, $\mathsf{lfun} \in \{\mathsf{lalloc}, \mathsf{lref}\}$.

$$\mathsf{lalloc}(x \mapsto \mathbf{y}) \triangleq \{x\} \quad \mathsf{lref}(x \mapsto \mathbf{y}) \triangleq \mathbf{y}$$
$$\mathsf{lfun}(\mathbf{emp}) \triangleq \emptyset$$
$$\mathsf{lfun}(\mathsf{pred}(\mathbf{y})) \triangleq \emptyset$$
$$\mathsf{lfun}(\phi_1 \star \phi_2) \triangleq \mathsf{lfun}(\phi_1) \cup \mathsf{lfun}(\phi_2)$$
$$\mathsf{lfun}(\exists e.\, \phi) \triangleq \mathsf{lfun}(\phi)$$

PROGRESS. A predicate pred satisfies *progress* iff every rule of pred contains exactly one points-to assertion and there is a variable $x \in \mathsf{fvars}(\mathsf{pred})$ such that for all rules $(\mathsf{pred}(\mathbf{x}) \Leftarrow \phi) \in \Phi$, $\mathsf{lalloc}(\phi) = \{x\}$. In this case, we define $\mathsf{lalloc}(\mathsf{pred}(\mathbf{y})) \triangleq \{x[\mathbf{x}/\mathbf{y}]\}$ and call x the *root of the predicate*. Moreover, if the i-th parameter of pred is the root of pred, then $\mathsf{predroot}(\mathsf{pred}(z_1, \ldots, z_k)) \triangleq z_i$.

CONNECTIVITY. A predicate pred satisfies *connectivity* iff for all rules of pred, all variables that are allocated in the recursive calls of the rule are referenced in the rule. Formally, for all rules $(\mathsf{pred} \Leftarrow \phi) \in \Phi$ and for all predicate calls $\mathsf{pred}'(\mathbf{y}) \in \phi$, it holds that $\mathsf{lalloc}(\mathsf{pred}'(\mathbf{y})) \subseteq \mathsf{lref}(\phi)$.

ESTABLISHMENT. A predicate pred is *established* iff all existentially quantified variables across all rules of pred are eventually allocated. Formally, for all rules $(\mathsf{pred} \Leftarrow \exists \mathbf{y}.\, \phi) \in \Phi$ and for all models $(\mathfrak{s}, \mathfrak{h})$, if $(\mathfrak{s}, \mathfrak{h}) \models_{\Phi} \phi$ then $\mathfrak{s}(\mathbf{y}) \subseteq \mathrm{dom}(\mathfrak{h})$.

We collect in $\mathbf{ID}_{\mathsf{btw}}$ all SIDs that satisfy *progress*, *connectivity*, and *establishment*. Unless stated otherwise, we assume $\Phi \in \mathbf{ID}_{\mathsf{btw}}$ for all SIDs Φ in this thesis.

Example 8.6 (SID Assumptions). *All SIDs in Example 8.2 satisfy progress, connectivity, and establishment and are thus elements of* $\mathbf{ID}_{\mathsf{btw}}$.

Throughout this thesis, I assume w.l.o.g. that rules whose right-hand side does not contain predicate calls do not contain existential quantifiers, because in $\mathbf{ID}_{\mathsf{btw}}$, such existentials can always be eliminated: all existentially-quantified variables in such “non-recursive” rules must be provably equal to the root of the predicate—and thus to a parameter of the predicate—to satisfy establishment. Consequently, we can replace all occurrences of existentially-quantified variables in non-recursive rules with the corresponding root variable.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

RESTRICTING SLID TO SIDS FROM $\mathbf{ID}_{\mathsf{btw}}$. I write $\mathbf{SLID}^{\mathsf{g}}_{\mathsf{btw}}$ and $\mathbf{SLID}^{\mathsf{qf}}_{\mathsf{btw}}$ for the restriction of the logics $\mathbf{SLID}^{\mathsf{g}}$ and $\mathbf{SLID}^{\mathsf{qf}}$ in which only SIDs from $\mathbf{ID}_{\mathsf{btw}}$ may be used to interpret predicate calls.

ADDING PREDICATES FOR POINTS-TO ASSERTIONS TO SIDS. To avoid dedicated reasoning about points-to assertions, we sometimes (specifically, in Section 8.4 and Chapter 12) exploit that we can add predicates that simulate points-to assertions to any SID. I call the resulting SIDs *pointer-closed*.

Definition 8.7 (Pointer-closed SID). *An SID Φ is* pointer-closed *w.r.t. ϕ iff it contains for all points-to assertions of arity $k+1$ that occur in ϕ a predicate ptr_k defined by the rule $\mathsf{ptr}_k(\mathbf{x}) \Leftarrow x_1 \mapsto \langle x_2, \ldots, x_{k+1}\rangle$.*

8.4 GUARDED MODELS AND DANGLING POINTERS

GUARDED MODELS. Much of the development in the remainder of this thesis exploits that guarded formulas interpreted over SIDs from $\mathbf{ID}_{\mathsf{btw}}$, i.e., formulas in $\mathbf{SLID}^{\mathsf{g}}_{\mathsf{btw}}$, only have *guarded* models in the following sense.

Definition 8.8. *Let $(\mathfrak{s},\mathfrak{h})$ be a model. We call $(\mathfrak{s},\mathfrak{h})$* guarded *w.r.t. SID Φ if there exist predicate calls $\mathsf{pred}_1(\mathbf{x_1}),\ldots,\mathsf{pred}_k(\mathbf{x_k})$, $k \geq 0$, such that $(\mathfrak{s},\mathfrak{h}) \models_{\Phi} \mathsf{pred}_1(\mathbf{x_1}) \star \cdots \star \mathsf{pred}_k(\mathbf{x_k})$.*

We write $\mathbf{Models}^{\mathsf{g}}_{\Phi} \triangleq \{(\mathfrak{s},\mathfrak{h}) \mid (\mathfrak{s},\mathfrak{h}) \text{ is guarded w.r.t. } \Phi\}$ for the set of all guarded models w.r.t. SID Φ.

The use of the adjective *guarded* is justified by the following lemma.

Lemma 8.9. *Let $\Phi \in \mathbf{ID}_{\mathsf{btw}}$ be a pointer-closed SID. Let $\phi \in \mathbf{SLID}^{\mathsf{g}}_{\mathsf{btw}}$ and let $(\mathfrak{s},\mathfrak{h})$ be a model with $(\mathfrak{s},\mathfrak{h}) \models_{\Phi} \phi$. Then $(\mathfrak{s},\mathfrak{h}) \in \mathbf{Models}^{\mathsf{g}}_{\Phi}$.*

Proof. By induction on ϕ.

CASE $\phi = \mathbf{emp}$. Trivial, as $(\mathfrak{s},\mathfrak{h})$ satisfies the separating conjunction of $k=0$ predicate calls.

CASE $\phi = x \approx y$, $\phi = x \not\approx y$. Analogously.

CASE $\phi = x \mapsto \mathbf{y}$. Because Φ is pointer-closed, there exists a predicate that $(\mathfrak{s},\mathfrak{h})$ satisfies.

CASE $\phi = \mathsf{pred}(\mathbf{x})$. Trivial.

CASE $\phi = \phi_1 \star \phi_2$. Split $\mathfrak{h}$ into $\mathfrak{h}_1 + \mathfrak{h}_2$ such that $(\mathfrak{s},\mathfrak{h}_1) \models_{\Phi} \phi_1$ and $(\mathfrak{s},\mathfrak{h}_2) \models_{\Phi} \phi_2$, which is possible by the semantics of $\star$. By the induction hypotheses, there exist $\mathsf{pred}_{1,1}(\mathbf{x_1}),\ldots,\mathsf{pred}_{1,m}(\mathbf{x_m})$ and $\mathsf{pred}_{2,1}(\mathbf{y_1}),\ldots,\mathsf{pred}_{2,n}(\mathbf{y_n})$ with

$$(\mathfrak{s},\mathfrak{h}_1) \models_{\Phi} \bigstar_{1\leq i\leq m} \mathsf{pred}_{1,i}(\mathbf{x_i}) \text{ and}$$
$$(\mathfrak{s},\mathfrak{h}_2) \models_{\Phi} \bigstar_{1\leq j\leq m} \mathsf{pred}_{2,j}(\mathbf{y_j}).$$

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

As $\mathfrak{h} = \mathfrak{h}_1 + \mathfrak{h}_2$, we obtain

$$(\mathfrak{s}, \mathfrak{h}) \models_{\Phi} \bigstar_{1 \leq i \leq m} \mathsf{pred}_{1,i}(\mathbf{x_i}) \star \bigstar_{1 \leq j \leq m} \mathsf{pred}_{2,j}(\mathbf{y_j}).$$

Thus, $(\mathfrak{s}, \mathfrak{h})$ is guarded.

CASE $\phi = \phi_1 \wedge \phi_2$. In this case, in particular $(\mathfrak{s}, \mathfrak{h}) \models_{\Phi} \phi_1$, so the claim follows immediately from the induction hypothesis for ϕ_1.

CASE $\phi = \phi_1 \vee \phi_2$. Assume w.l.o.g. that $(\mathfrak{s}, \mathfrak{h}) \models_{\Phi} \phi_1$. The claim follows immediately from the induction hypothesis for ϕ_1. □

DANGLING POINTERS IN $\mathbf{SLID}_{\mathsf{btw}}$. If a predicate pred satisfies establishment, and $(\mathfrak{s}, \mathfrak{h}) \models_{\Phi} \mathsf{pred}(\mathbf{y})$, then only the locations $\mathfrak{s}(\mathbf{y})$ can be dangling in $(\mathfrak{s}, \mathfrak{h})$.

Recall from Section 2.4 that we denote the dangling locations of the heap $\mathfrak{h}$ by $\mathsf{dangling}(\mathfrak{h}) \triangleq \{l \in \bigcup \mathrm{img}(\mathfrak{h}) \mid l \notin \mathrm{dom}(\mathfrak{h})\}$.

Lemma 8.10. *Let $\mathfrak{h}$ be a heap, let $\mathbf{l}$ be a sequence of locations, let $\Phi \in \mathbf{ID}_{\mathsf{btw}}$, and $\mathsf{pred} \in \mathbf{Preds}(\Phi)$. If $\mathfrak{h} \models_{\Phi} \mathsf{pred}(\mathbf{l})$ then $\mathsf{dangling}(\mathfrak{h}) \subseteq \mathbf{l}$.*

Proof. A simple consequence of establishment. □

Lemma 8.11. *Let $(\mathfrak{s}, \mathfrak{h})$ be a stack–heap pair, let $\mathbf{z} \in \mathbf{Var}^*$, let $\Phi \in \mathbf{ID}_{\mathsf{btw}}$, and $\mathsf{pred} \in \mathbf{Preds}(\Phi)$. If $(\mathfrak{s}, \mathfrak{h}) \models_{\Phi} \mathsf{pred}(\mathbf{z})$ then $\mathsf{dangling}(\mathfrak{h}) \subseteq \mathfrak{s}(\mathbf{z})$.*

Proof. Follows from the semantics and Lemma 8.10. □

This property extends to arbitrary $\mathbf{SLID}^{\mathsf{g}}_{\mathsf{btw}}$ formulas over $\mathbf{ID}_{\mathsf{btw}}$ SIDs and hence to guarded models.

Lemma 8.12. *Let $\Phi \in \mathbf{ID}_{\mathsf{btw}}$ and $(\mathfrak{s}, \mathfrak{h}) \in \mathbf{Models}^{\mathsf{g}}_{\Phi}$. Then $\mathsf{dangling}(\mathfrak{h}) \subseteq \mathrm{img}(\mathfrak{s})$.*

Proof. By Lemma 8.9, there exist predicate calls $\mathsf{pred}_1(\mathbf{x_1}), \ldots, \mathsf{pred}_k(\mathbf{x_k})$ such that $(\mathfrak{s}, \mathfrak{h}) \models_{\Phi} \bigstar_{1 \leq i \leq k} \mathsf{pred}_i(\mathbf{x_i})$. By the semantics of $\star$, there thus exist $\mathfrak{h}_1, \ldots \mathfrak{h}_k$ with $\mathfrak{h} = \mathfrak{h}_1 + \cdots + \mathfrak{h}_k$ and $(\mathfrak{s}, \mathfrak{h}_i) \models_{\Phi} \mathsf{pred}_i(\mathbf{x_i})$. Lemma 8.11 yields $\mathsf{dangling}(\mathfrak{h}_i) \subseteq \mathfrak{s}(\mathbf{x_i})$ for all $1 \leq i \leq k$. It follows that

$$\begin{aligned} \mathsf{dangling}(\mathfrak{h}) &= \mathsf{dangling}(\mathfrak{h}_1) \cup \cdots \mathsf{dangling}(\mathfrak{h}_k) \\ &\subseteq \mathfrak{s}(\mathbf{x_1}) \cup \cdots \cup \mathfrak{s}(\mathbf{x_k}) \subseteq \mathrm{img}(\mathfrak{s}), \end{aligned}$$

where the last inequality holds because $\mathbf{x_i} \subseteq \mathrm{dom}(\mathfrak{s})$ for all i. □

Corollary 8.13. *Let $\phi_1, \phi_2 \in \mathbf{SLID}^{\mathsf{g}}_{\mathsf{btw}}$ be guarded formulas and let $\Phi \in \mathbf{ID}_{\mathsf{btw}}$. Let $(\mathfrak{s}, \mathfrak{h}) \models_{\Phi} \phi_1 \star \phi_2$. Then there exist heaps $\mathfrak{h}_1, \mathfrak{h}_2$ such that $(\mathfrak{s}, \mathfrak{h}_1), (\mathfrak{s}, \mathfrak{h}_2) \in \mathbf{Models}^{\mathsf{g}}_{\Phi}$, $\mathfrak{h} = \mathfrak{h}_1 + \mathfrak{h}_2$, $(\mathfrak{s}, \mathfrak{h}_1) \models_{\Phi} \phi_1$ and $(\mathfrak{s}, \mathfrak{h}_2) \models_{\Phi} \phi_2$.*

Proof. By the semantics of $\star$, there exist $\mathfrak{h}_1$, $\mathfrak{h}_2$ with $\mathfrak{h} = \mathfrak{h}_1 + \mathfrak{h}_2$, $(\mathfrak{s}, \mathfrak{h}_1) \models_{\Phi} \phi_1$ and $(\mathfrak{s}, \mathfrak{h}_2) \models_{\Phi} \phi_2$. By Lemma 8.9, $(\mathfrak{s}, \mathfrak{h}_1), (\mathfrak{s}, \mathfrak{h}_2) \in \mathbf{Models}^{\mathsf{g}}_{\Phi}$. □

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

We will exploit this property of $\mathbf{SLID}^{\mathrm{g}}_{\mathrm{btw}}$ in the decision procedure we develop later in the thesis.

The importance of guardedness for limiting dangling pointers

For all models to be guarded (Lemma 8.9), it is crucial that all occurrences of negation, magic wand, and septraction in $\mathbf{SLID}^{\mathrm{g}}_{\mathrm{btw}}$ are guarded. For negation and the magic wand, this is straightforward, as they can be used to define *true* (e.g. by $\mathbf{emp} \vee (\neg\mathbf{emp})$ and $((x \mapsto \mathsf{nil}) \star (x \mapsto \mathsf{nil})) \mathrel{-\!\!\star} \mathbf{emp}$), and *true* is satisfied by all models, including non-guarded models.

For septraction, consider the following SID.

$$
\begin{aligned}
\mathsf{tll}(r,h,t) &\Leftarrow h \mapsto t \star r \approx h \\
\mathsf{tll}(r,h,t) &\Leftarrow \exists \langle s_1, s_2, m\rangle \,.\, (r \mapsto \langle s_1, s_2\rangle) \star \mathsf{tll}(s_1,h,m) \star \mathsf{tll}(s_2,m,t) \\
\mathsf{lseg}(h,t) &\Leftarrow h \mapsto t \\
\mathsf{lseg}(h,t) &\Leftarrow \exists n.\, (h \mapsto n) \star \mathsf{lseg}(n,t)
\end{aligned}
$$

The tll predicate encodes a binary tree with root r and leftmost leaf h overlaid with a singly-linked list segment from h to t whose nodes are the leaves of the tree. Now assume that $(\mathfrak{s},\mathfrak{h}) \models_{\Phi} \mathsf{lseg}(h,t) \mathrel{-\!\circledast} \mathsf{tll}(r,h,t)$. Then there exists a heap $\mathfrak{h}_1$ with $(\mathfrak{s},\mathfrak{h}_1) \models_{\Phi} \mathsf{lseg}(h,t)$ and $(\mathfrak{s},\mathfrak{h}+\mathfrak{h}_1) \models_{\Phi} \mathsf{tll}(r,h,t)$. It is easy to see that $\mathsf{dangling}(\mathfrak{h}) = \mathrm{dom}(\mathfrak{h}_1)$, contradicting Lemma 8.12, which in turn implies that $\mathsf{lseg}(h,t) \mathrel{-\!\circledast} \mathsf{tll}(r,h,t)$ has non-guarded models.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.

9 TOWARDS A COMPOSITIONAL ABSTRACTION FOR SLID

In Section 3.4, I gave a high-level argument that a finite *compositional abstraction* that *refines* the satisfaction relation can be used to implement satisfiability checking for separation logic. In Part ii, we followed this approach to decide **SSL** and $\mathbf{SSL}_{\mathsf{data}}$. In this part, we take the same approach to decide $\mathbf{SLID}^{\mathsf{g}}_{\mathsf{btw}}$.

Complications and simplifications compared to the AMS abstraction

If you've read Part ii, you are familiar with the *AMS abstraction*. The AMS abstraction only needs to keep track of the built-in list and tree predicates, which allowed a fairly simple abstraction of *chunks of memory* by a small set of hyperedges. Such an abstraction is obviously not possible for $\mathbf{SLID}^{\mathsf{g}}_{\mathsf{btw}}$, where we have to deal with arbitrary user-defined predicates from $\mathbf{ID}_{\mathsf{btw}}$, leading to a more complicated abstraction. At the same time, AMSs had to keep track of *garbage* in the model, because **SSL** allowed unrestricted use of negations and magic wands. This is not necessary when abstracting $\mathbf{SLID}^{\mathsf{g}}_{\mathsf{btw}}$, since guardedness guarantees that models of $\mathbf{SLID}^{\mathsf{g}}_{\mathsf{btw}}$ formulas cannot contain garbage.

Clearly, the key challenge is to develop an abstraction mechanism that can deal with arbitrary user-defined predicates from the $\mathbf{ID}_{\mathsf{btw}}$ fragment. To get an abstraction that satisfies refinement, we need to be able to deduce from the abstraction which predicate calls hold in the underlying model. To this end, we will abstract every model by a set of formulas that relates the model to predicates of the SID.

9.1 FIRST ATTEMPT: ABSTRACTING MODELS BY SYMBOLIC HEAPS

Our first idea is to abstract a model by the quantifier-free symbolic heaps that it satisfies.

$$\mathsf{abst}_1(\mathfrak{s},\mathfrak{h}) \triangleq \{\phi \text{ quantifier-free symbolic heap} \mid (\mathfrak{s},\mathfrak{h}) \models_\Phi \phi\}\,.$$

Let us analyze the properties of this abstraction function.

FINITENESS. For the moment, let us not worry whether we can actually compute this abstraction. At least, it is finite, because there are only finitely many quantifier-free symbolic heaps up to logical equivalence: if $\Phi \in \mathbf{ID}_{\mathsf{btw}}$, every predicate call in a symbolic heap ϕ

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

has to allocate at least one free variable because of the *progress* property; trivially, the same holds for every points-to assertion. Consequently, every satisfiable quantifier-free formula can contain at most $|\mathsf{fvars}(\phi)|$ many predicate calls and points-to assertions. In principle, we can, of course, "blow up" a satisfiable formula ϕ to arbitrary size by adding **emp** atoms and (dis-)equalities, but any fixed *stack-aliasing constraint* (cf. Section 2.4) over $\mathsf{fvars}(\phi)$ can be expressed by fewer than $|\mathsf{fvars}(\phi)|^2$ such atoms. For this reason, it is not necessary to consider larger symbolic heaps in the abstraction.

REFINEMENT. Trivially, abst_1 guarantees refinement at least on the symbolic-heap fragment of $\mathbf{SLID}^{\mathsf{g}}_{\mathsf{btw}}$.

COMPOSITIONALITY. Can we *compose* abstractions, i.e., can we find a (computable) operator $\bullet$ with $\mathsf{abst}_1(\mathfrak{s}, \mathfrak{h}_1) \bullet \mathsf{abst}_1(\mathfrak{s}, \mathfrak{h}_2) = \mathsf{abst}_1(\mathfrak{s}, \mathfrak{h}_1 + \mathfrak{h}_2)$? A very simple example shows that this is a difficult problem. Assume Φ defines the list-segment predicate lseg (cf. Example 8.2), $(\mathfrak{s}, \mathfrak{h}_1) \models_\Phi \mathsf{lseg}(x, y)$, $(\mathfrak{s}, \mathfrak{h}_2) \models_\Phi \mathsf{lseg}(y, z)$ and $\mathfrak{s}(x) \neq \mathfrak{s}(y) \neq \mathfrak{s}(z)$. Then (not including pure constraints for simplicity):

- $\mathsf{abst}_1(\mathfrak{s}, \mathfrak{h}_1) = \{\mathsf{lseg}(x, y)\}$
- $\mathsf{abst}_1(\mathfrak{s}, \mathfrak{h}_2) = \{\mathsf{lseg}(y, z)\}$
- $\mathsf{abst}_1(\mathfrak{s}, \mathfrak{h}_1 + \mathfrak{h}_2) = \{\mathsf{lseg}(x, z)\}$.

That $\mathsf{lseg}(x, z)$ holds in the composed model cannot be inferred from the abstractions of the individual model in a syntactic way. Instead, we have to look at the semantics of the predicate, i.e., at the Φ. In fact, the above composition operation $\bullet$ boils down to an entailment check

$$\mathsf{lseg}(x, y) \star \mathsf{lseg}(y, z) \models_\Phi \mathsf{lseg}(x, z)$$

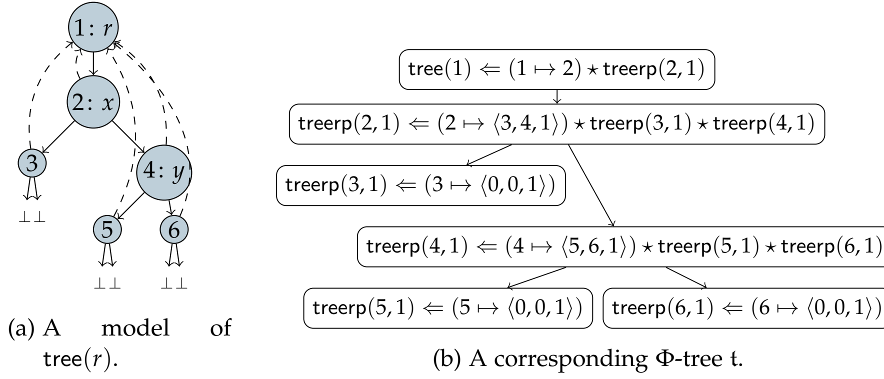
So we have a chicken-and-egg problem: we need an entailment checker to implement the composition operator $\bullet$ that we would like to use in the implementation of our abstraction-based (satisfiability and) entailment checker.

9.2 SECOND ATTEMPT: UNFOLDING PREDICATES INTO FORESTS

Can we extend the abstraction abst_1 to get a "more syntactic" composition operation that can be implemented without an entailment check? Yes, we can.[1][2] To explain how, we need to take a step back and think about the semantics of SIDs.

1 Change we can believe in.
2 Or, as suggested by Thomas Wies, let's make abstractions great again!

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub



(a) A model of $\mathsf{tree}(r)$.

(b) A corresponding Φ-tree $\mathfrak{t}$.

Figure 9.1: A model $(\mathfrak{s}, \mathfrak{h}) \models_\Phi \mathsf{tree}(r)$ and the Φ-tree $\mathfrak{t}$ corresponding to this model.

UNFOLDING PREDICATE CALLS. According to the semantics, the model relationship $(\mathfrak{s}, \mathfrak{h}) \models_\Phi \mathsf{pred}(\mathbf{z})$ holds iff there exists a rule $\mathsf{pred}(\mathbf{x}) \Leftarrow \psi(\mathbf{x}) \in \Phi$ such that $(\mathfrak{s}, \mathfrak{h}) \models_\Phi \psi(\mathbf{z})$. We say that we have *unfolded* the predicate pred by its rule ψ. In general, ψ may itself contain predicate calls. To show that $(\mathfrak{s}, \mathfrak{h}) \models_\Phi \psi(\mathbf{z})$, we must unfold each of these predicate calls by a rule of the respective predicates. This unfolding process continues until every predicate has been unfolded by a nonrecursive rule.

It is natural to visualize such an unfolding process as a tree. In fact, it is quite possible to define the semantics of inductive predicates explicitly in terms of such *unfolding trees* (cf. [IRS13; IRV14; Jan+17]). In this thesis, I define a variant of unfolding trees, Φ-*trees*.

Example 9.1 (Φ-tree)**.** *Consider the following SID* Φ*.*

$$\begin{aligned}
\mathsf{tree}(r) &\Leftarrow \exists x.\, (r \mapsto x) \star \mathsf{treerp}(x, r) \\
\mathsf{treerp}(x, r) &\Leftarrow (x \mapsto \langle \mathsf{nil}, \mathsf{nil}, r \rangle) \\
\mathsf{treerp}(x, r) &\Leftarrow \exists \langle c_1, c_2 \rangle \,.\, (x \mapsto \langle c_1, c_2, r \rangle) \star \mathsf{treerp}(c_1, r) \star \mathsf{treerp}(c_2, r)
\end{aligned}$$

Let $\mathfrak{s} = \{r \mapsto 1, x \mapsto 2, y \mapsto 4\}$, $\mathfrak{h} = \{1 \mapsto 2, 2 \mapsto \langle 3, 4, 1 \rangle, 3 \mapsto \langle 0, 0, 1 \rangle, 4 \mapsto \langle 5, 6, 1 \rangle, 5 \mapsto \langle 0, 0, 1 \rangle, 6 \mapsto \langle 0, 0, 1 \rangle\}$*, displayed in Fig. 9.1a. Note that* $(\mathfrak{s}, \mathfrak{h}) \models_\Phi \mathsf{tree}(r)$*. Each node is labeled with a location and the stack variable interpreted by the location (if any). The first two outgoing pointers of each node are displayed by solid edges, the third pointer by a dashed edge.*

Figure 9.1b shows a Φ*-tree* $\mathfrak{t}$ *that corresponds to this model. Each node of the* Φ*-tree* $\mathfrak{t}$ *is labeled with a* rule instance*, i.e., a rule of the SID in which all variables—both formal parameters and existentially-quantified variables—have been instantiated with the locations of the model. This is a difference to other notions of unfolding trees, in which nodes are labeled by rules, not rule instances [IRS13; IRV14; Jan+17]. Note that* $\mathfrak{t}$ *induces the heap* $\mathfrak{h}$ *in a very direct way:* $\mathfrak{h}$ *is the union of all the points-to assertions that occur in the node labels of* $\mathfrak{t}$*. We denote this as* $\mathfrak{h} = \mathsf{heap}(\mathfrak{t})$*.*

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Now that we are familiar with Φ-trees, we can read the entailment

$$\mathsf{lseg}(x, y) \star \mathsf{lseg}(y, z) \models_\Phi \mathsf{lseg}(x, z)$$

as follows: the entailment is valid iff whenever $(\mathfrak{s}, \mathfrak{h}) \models_\Phi \mathsf{lseg}(x, y) \star \mathsf{lseg}(y, z)$ holds, it is possible to find a Φ-tree $\mathfrak{t}$ with $\mathsf{heap}(\mathfrak{t}) = \mathfrak{h}$ and with root $\mathsf{lseg}(\mathfrak{s}(x), \mathfrak{s}(z))$.

ABSTRACTING MODELS BY FORESTS. Our next abstraction attempt is to encode the existence of such a Φ-tree in the abstraction.

More precisely, we need to encode that the models of $\mathsf{lseg}(x, y)$ and $\mathsf{lseg}(y, z)$ each correspond to *partial* unfolding trees of $\mathsf{lseg}(x, z)$ that can be combined into an unfolding tree of $\mathsf{lseg}(x, z)$. We model such partial unfolding trees by allowing *holes* in Φ-trees: we allow the unfolding process to stop at any point, i.e., that one or more of the predicate calls introduced (by means of recursive rules) in the unfolding process remain folded. The calls that remain folded form the holes of the tree.

Example 9.2 (Φ-trees with holes)**.** *Recall the entailment* $\mathsf{lseg}(x, y) \star \mathsf{lseg}(y, z) \models_\Phi \mathsf{lseg}(x, z)$ *from above. Figure 9.2a shows* $(\mathfrak{s}, \mathfrak{h}_1), (\mathfrak{s}, \mathfrak{h}_2)$ *with* $(\mathfrak{s}, \mathfrak{h}_1) \models_\Phi \mathsf{lseg}(x, y)$ *and* $(\mathfrak{s}, \mathfrak{h}_2) \models_\Phi \mathsf{lseg}(y, z)$*. By the semantics of* $\star$*, it holds for* $\mathfrak{h} \triangleq \mathfrak{h}_1 + \mathfrak{h}_2$ *that* $(\mathfrak{s}, \mathfrak{h}) \models_\Phi \mathsf{lseg}(x, y) \star \mathsf{lseg}(y, z)$*.*

We would like to argue that $(\mathfrak{s}, \mathfrak{h}) \models_\Phi \mathsf{lseg}(x, z)$*. To this end, Fig. 9.2b shows a* Φ-forest *consisting of two trees,* $\mathfrak{t}_1$ *and* $\mathfrak{t}_2$*. The tree* $\mathfrak{t}_1$ *corresponds to the sub-heap* $\mathfrak{h}_1$ *(i.e,* $\mathsf{heap}(\mathfrak{t}_1) = \mathfrak{h}_1$*) and* $\mathfrak{t}_2$ *corresponds to* $\mathfrak{h}_2$ *(i.e.,* $\mathsf{heap}(\mathfrak{t}_2) = \mathfrak{h}_2$*. The tree* $\mathfrak{t}_1$ *contains a* hole*: the recursive call* $\mathsf{lseg}(4, 6)$ *is not unfolded in the tree. The hole, location* 4*, is not allocated in the tree, even though it is the root parameter of the* hole predicate $\mathsf{lseg}(4, 6)$*. Put differently, the tree* $\mathfrak{t}_1$ *witnesses that* $\mathfrak{h}_1$ *is a partial model of* $\mathsf{lseg}(1, 6)$*, because* $\mathfrak{h}_1$ *corresponds to a partial unfolding tree (i.e., an unfolding tree with holes) with root* $\mathsf{lseg}(1, 6)$*.*

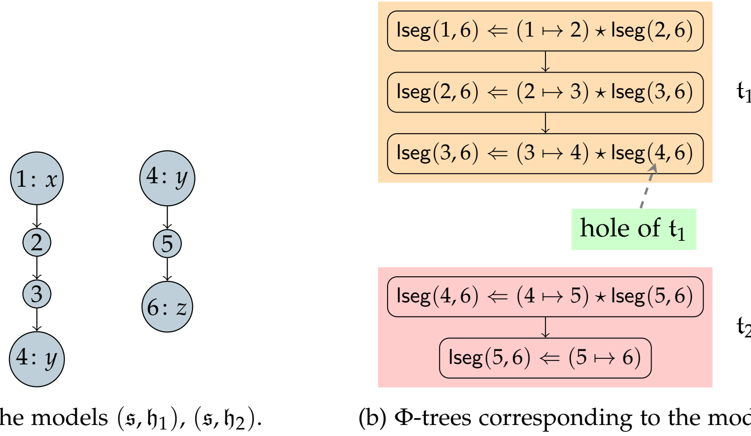
We can merge $\mathfrak{t}_1$ *and* $\mathfrak{t}_2$ *into a larger tree by adding an edge from the hole of* $\mathfrak{t}_1$ *and the root of* $\mathfrak{t}_2$*. Adding such an edge makes sense, because the root of* $\mathfrak{t}_2$ *is labeled with the aforementioned hole predicate,* $\mathsf{lseg}(4, 6)$*. The resulting tree is a* Φ*-tree for* $\mathsf{lseg}(x, z)$*—that is, a tree without any holes whose root is labeled with a rule instance for* $\mathsf{lseg}(\mathfrak{s}(x), \mathfrak{s}(z))$*.*

By merging the two trees—i.e., identifying the root of the tree $\mathfrak{t}_3$ *with the hole of* $\mathfrak{t}_1$*—we have thus shown that the model of* $\mathsf{lseg}(x, y) \star \mathsf{lseg}(y, z)$ *is also a model of* $\mathsf{lseg}(x, z)$*.*

We formalize Φ-*trees* in Definition 10.1 in the next chapter. We can go one step further and consider *partial unfolding forests* or simply Φ-*forests* (cf. Definition 10.3), i.e., forests consisting of Φ-trees. For example, $\{\mathfrak{t}_1, \mathfrak{t}_2\}$ in Fig. 9.2 are a Φ-forest. Forests can contain an arbitrary finite number of trees.

Example 9.3 (Φ-forest)**.** *Recall* Φ *and* $(\mathfrak{s}, \mathfrak{h})$ *from Example 9.1. Fig. 9.3 shows a* Φ-forest $\mathfrak{f} = \{\mathfrak{t}_1, \mathfrak{t}_2, \mathfrak{t}_3\}$ *that encodes one way to derive the model*

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub



(a) The models $(\mathfrak{s}, \mathfrak{h}_1)$, $(\mathfrak{s}, \mathfrak{h}_2)$. (b) Φ-trees corresponding to the models.

Figure 9.2: Models $(\mathfrak{s}, \mathfrak{h}_1) \models_\Phi \mathsf{lseg}(x, y)$ and $(\mathfrak{s}, \mathfrak{h}_2) \models_\Phi \mathsf{lseg}(y, z)$ and Φ-trees $\mathfrak{t}_1, \mathfrak{t}_2$ for these models. The tree $\mathfrak{t}_1$ contains one predicate call that is not unfolded, the *hole predicate* $\mathsf{lseg}(4, 6)$. We say that 4, the root of this folded predicate call, is a hole of the tree.

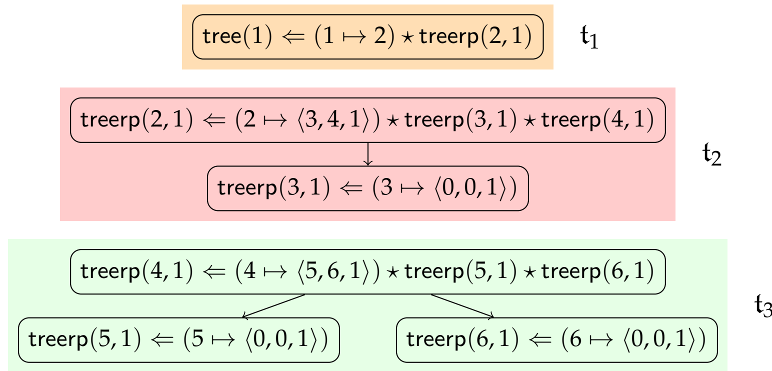


Figure 9.3: A Φ-forest $\mathfrak{f} = \{\mathfrak{t}_1, \mathfrak{t}_2, \mathfrak{t}_3\}$ for the model from Example 9.1, used in Example 9.3.

$(\mathfrak{s}, \mathfrak{h})$ *by unfolding predicates of the SID. Both* $\mathfrak{t}_1$ *and* $\mathfrak{t}_2$ *only* partially *unfold the predicates at their roots, leaving* 2 *and* 4 *as holes, respectively. By merging the three trees, we get the tree* $\mathfrak{t}$ *from Example 9.1.*

Denote by $\mathsf{abst}_2(\mathfrak{s}, \mathfrak{h})$ the set of all Φ-forests of the model $(\mathfrak{s}, \mathfrak{h})$.

COMPOSITIONALITY. We can easily define a suitable composition operation: the operation $\mathsf{abst}_2(\mathfrak{s}, \mathfrak{h}_1) \bullet \mathsf{abst}_2(\mathfrak{s}, \mathfrak{h}_2)$ consists in computing all ways to merge the Φ-forests of $\mathsf{abst}_2(\mathfrak{s}, \mathfrak{h}_1)$ and $\mathsf{abst}_2(\mathfrak{s}, \mathfrak{h}_2)$. It is fairly easy to see that this process yields precisely the set of all Φ-forests of $(\mathfrak{s}, \mathfrak{h}_1 + \mathfrak{h}_2)$, i.e., the set $\mathsf{abst}_2(\mathfrak{s}, \mathfrak{h}_1 + \mathfrak{h}_2)$, as required by the compositionality property.

FINITENESS. There are two reasons that abst_2 gives rise to an infinite abstraction.

ISSUE 1 The tree nodes are labeled with concrete locations, so the value of abst_2 differs even for isomorphic models—not exactly

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

what we would expect of an abstraction, and one reason that the image of abst_2 is an infinite set.

ISSUE 2 If we keep track of all unfolding forests, the size of $\mathsf{abst}_2(\mathfrak{s}, \mathfrak{h})$ grows with the size of $\mathfrak{h}$, without any upper bound. For example, I could map a list sequence of size n to a forest that contains n one-node trees. This is another reason that the image of abst_2 is an infinite set.

This part of the thesis is to a large extent concerned with overcoming these two issues to obtain a viable abstraction for $\mathbf{SLID}^{\mathsf{g}}_{\mathsf{btw}}$.

9.3 THIRD ATTEMPT: FOREST PROJECTIONS

Above, I introduced Φ-trees as unfolding trees with holes. Let's say $\mathfrak{t}$ is a Φ-tree with root $\mathsf{rootpred}(\mathfrak{t})$ and with hole predicates $\mathsf{allholepreds}(\mathfrak{t})$.

Example 9.4. *1. Let $\mathfrak{t}_1, \mathfrak{t}_2$ be the Φ-trees from Example 9.2. Then*

- $\mathsf{rootpred}(\mathfrak{t}_1) = \mathsf{lseg}(1, 6)$, $\mathsf{allholepreds}(\mathfrak{t}_1) = \{\mathsf{lseg}(4, 6)\}$,
- $\mathsf{rootpred}(\mathfrak{t}_2) = \mathsf{lseg}(4, 6)$, $\mathsf{allholepreds}(\mathfrak{t}_2) = \emptyset$.

2. *Let $\mathfrak{t}_1, \mathfrak{t}_2, \mathfrak{t}_3$ be the Φ-trees from Example 9.3. Then*

- $\mathsf{rootpred}(\mathfrak{t}_1) = \mathsf{tree}(1)$, $\mathsf{allholepreds}(\mathfrak{t}_1) = \{\mathsf{treerp}(2, 1)\}$,
- $\mathsf{rootpred}(\mathfrak{t}_2) = \mathsf{treerp}(2, 1)$, $\mathsf{allholepreds}(\mathfrak{t}_2) = \{\mathsf{treerp}(4, 1)\}$,
- $\mathsf{rootpred}(\mathfrak{t}_3) = \mathsf{treerp}(4, 1)$, $\mathsf{allholepreds}(\mathfrak{t}_3) = \emptyset$.

The main insight behind the Φ-*type* abstraction is that every Φ-tree $\mathfrak{t}$ can be viewed as encoding a model of $\mathsf{rootpred}(\mathfrak{t})$ from which models of $\mathsf{allholepreds}(\mathfrak{t})$ have been subtracted. The tree $\mathfrak{t}$ can thus be *projected* onto the formula

$$(\bigstar \mathsf{allholepreds}(\mathfrak{t})) \mathrel{-\!\!\star} \mathsf{rootpred}(\mathfrak{t}). \qquad (\dagger)$$

Observe that the above formula contains locations rather than variables, because the parameters of predicate calls in Φ-trees are locations, not variables. If we simply used the above magic wand in our abstraction, we would thus get a different abstraction even of Φ-trees that encode the same model up to isomorphism—in other words, we have not yet solved Issue 1 as formulated in the previous section.

REPLACING LOCATIONS WITH VARIABLES. To avoid this problem, we replace the locations in the formula $(\dagger)$ with variables. Our first attempt is as follows. Say $\mathfrak{t}$ is a Φ-tree of the model $(\mathfrak{s}, \mathfrak{h})$. Then we replace the locations in the formula $(\dagger)$ in the following way.

1. Every location $v \in \mathrm{img}(\mathfrak{s})$ is replaced by an arbitrary variable in $\mathfrak{s}^{-1}(v)$.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

2. Every location in $\mathsf{locs}(\mathfrak{h}) \setminus \mathrm{img}(\mathfrak{s})$ is replaced by an existentially-quantified variable, because there exists a location in the heap $\mathfrak{h}$ that corresponds to the location in the formula (†).

3. All other locations are replaced by a universally-quantified variable, because these locations do not occur in the heap $\mathfrak{h}$ and can thus be picked in an arbitrary way.

Example 9.5 (Forest projection (first attempt)). *1. Let $\mathfrak{s}$ be the stack and let $\mathfrak{t}_1, \mathfrak{t}_2$ be the Φ-trees from Example 9.2. The projection of $\mathfrak{s}$ and Φ-forest $\{\mathfrak{t}_1, \mathfrak{t}_2\}$ is*

$$(\mathsf{lseg}(y,z) \mathrel{-\!\!\star} \mathsf{lseg}(x,z)) \star (\mathbf{emp} \mathrel{-\!\!\star} \mathsf{lseg}(y,z)).$$

2. Let $\mathfrak{t}_1$ be as above and let $\mathfrak{s}'$ be the restriction of $\mathfrak{s}$ to $\{x,y\}$. The projection of $\mathfrak{s}'$ and Φ-forest $\{\mathfrak{t}_1\}$ is

$$\forall a.\, \mathsf{lseg}(y,a) \mathrel{-\!\!\star} \mathsf{lseg}(x,a)$$

3. Let $\mathfrak{s}$ and $\mathfrak{f}$ be the stack and Φ-forest from Example 9.3. The projection of $\mathfrak{s}$ and $\mathfrak{f}$ is

$$\begin{aligned} &(\mathsf{treerp}(x,r) \mathrel{-\!\!\star} \mathsf{tree}(x,r)) \\ \star\, &(\mathsf{treerp}(y,r) \mathrel{-\!\!\star} \mathsf{treerp}(x,r)) \\ \star\, &(\mathbf{emp} \mathrel{-\!\!\star} \mathsf{treerp}(y,r)). \end{aligned}$$

4. Let $\mathfrak{f}$ be as above and let $\mathfrak{s}'$ be the restriction of $\mathfrak{s}$ to $\{r\}$. The projection of $\mathfrak{s}'$ and $\mathfrak{f}$ is

$$\begin{aligned} \exists \langle e_1, e_2 \rangle .(&\mathsf{treerp}(e_1,r) \mathrel{-\!\!\star} \mathsf{tree}(e_1,r)) \\ &\star (\mathsf{treerp}(e_2,r) \mathrel{-\!\!\star} \mathsf{treerp}(e_1,r)) \\ &\star (\mathbf{emp} \mathrel{-\!\!\star} \mathsf{treerp}(e_2,r)). \end{aligned}$$

This is not quite the right way to define forest projections, for reasons that go beyond the scope of this overview chapter; but it is close enough to the actual definition to serve as a guide for the next two chapters, Chapters 10 and 11.

9.4 BEYOND THE THIRD ATTEMPT

To sum up, I propose to abstract the model $(\mathfrak{s}, \mathfrak{h})$ in the following way.

1. We compute all Φ-forests of $(\mathfrak{s}, \mathfrak{h})$.

2. We project these forests onto formulas as explained above.

3. The abstraction of $(\mathfrak{s}, \mathfrak{h})$ is the set of all these formulas.

There are two important complications that bar us from implementing this approach.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

1. The use of quantifiers as proposed above fails for certain SIDs that use pure constraints. For this reason, we have to use *guarded* variants of the quantifiers. I will introduce these quantifiers and justify their use in Section 10.2.

2. We still haven't solved Issue 2 from Section 9.2: just like the number of forests, the number of projections grows as the model grows.

   To obtain a finite abstraction, we thus have to limit the Φ-forests that we consider for projection. In Section 11.1, I will identify the set of *delimited Φ-forests* as a finite set of Φ-forests suitable for defining a finite and compositional abstraction.

OUTLINE OF THIS PART OF THE THESIS. In Chapter 10, I formalize Φ-forests and their projections. Chapter 11 defines the Φ-*type* abstraction as the set of projections of all *delimited Φ-forests*. In Chapter 12, I show the *refinement theorem* for the Φ-type abstraction and $\mathbf{SLID}^{\mathsf{g}}_{\mathsf{btw}}$ formulas, and develop a decision procedure for $\mathbf{SLID}^{\mathsf{g}}_{\mathsf{btw}}$ based on computing Φ-types. Finally, I prove the undecidability of all unguarded extensions of $\mathbf{SLID}^{\mathsf{g}}_{\mathsf{btw}}$ in Chapter 13.

TU WIEN Bibliothek Your knowledge hub
Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.

10 FORESTS AND THEIR PROJECTIONS

In this chapter, I formalize the concepts that I informally introduced in the previous chapter: Φ-*forests* (Section 10.1), their projection onto formulas (Section 10.2), and the composition of these objects (Section 10.3). This will prepare us for defining the Φ-*type* abstraction in Chapter 11. The decision procedures for $\mathbf{SLID}^{g}_{\mathsf{btw}}$ based on Φ-types follow in Chapter 12.

10.1 FORESTS

Our main objects of study in this section are Φ-*forests* (Definition 10.3) made up of Φ-*trees* (Definition 10.1). As motivated in Chapter 9, a Φ-tree encodes one fixed way to unfold a predicate call by means of the rules of the SID Φ. The differences between the unfolding trees of [IRS13; IRV14; Jan+17] and our Φ-trees are (1) that we instantiate variables with locations and (2) that Φ-trees can have *holes*, because we allow that one or more of the predicate calls introduced (by means of recursive rules) in the unfolding process remain folded.

RULE INSTANCES. We annotate every node of a Φ-tree with a *rule instance* of the SID Φ, i.e., with a formula obtained from a rule of the SID by instantiating both the formal arguments of the predicates and the existentially quantified variables of the rule with locations:

$$\begin{aligned}\mathbf{RuleInst}(\Phi) \triangleq \{&\mathsf{pred}(\mathbf{v}) \Leftarrow \phi[\mathbf{x}\cdot\mathbf{y}/\mathbf{v}\cdot\mathbf{w}] \mid \\ &(\mathsf{pred}(\mathbf{x}) \Leftarrow \exists \mathbf{y}.\,\phi) \in \Phi, \\ &\mathbf{v} \in \mathbf{Loc}^{\mathsf{ar}(\mathsf{pred})}, \mathbf{w} \in \mathbf{Loc}^{|\mathbf{y}|} \text{ and all} \\ &\text{(dis)equalities in } \phi[\mathbf{x}\cdot\mathbf{y}/\mathbf{v}\cdot\mathbf{w}] \text{ are valid}\}\end{aligned}$$

Here, I mean the (dis-)equalities that occur explicitly in the formula, not those implied by recursive calls or by the separating conjunction. Note that because all variables have been instantiated with locations, it is trivial to check whether these (dis)equalities hold. Furthermore, whenever $\mathfrak{h} \models_{\Phi} \mathsf{pred}(\mathbf{v})$, there is at least one rule instance $(\mathsf{pred}(\mathbf{v}) \Leftarrow \psi) \in \mathbf{RuleInst}(\Phi)$ such that $\mathfrak{h} \models_{\Phi} \psi$.

Φ-TREES. We model Φ-trees as partial functions

$$\mathfrak{t}\colon \mathbf{Loc} \rightharpoonup \left(2^{\mathbf{Loc}} \times \mathbf{RuleInst}(\Phi)\right).$$

The set of locations $\mathbf{Loc}$ serves as the nodes of the tree; and every node is mapped to its successors in the (directed) tree as well as to

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

its label, a rule instance. For $\mathfrak{t}$ to be a Φ-tree, it must satisfy a certain set of consistency criteria. To make it easier to work with Φ-trees and formalize the consistency criteria, we first introduce some additional notation.

Let $\Phi \in \mathbf{ID}_{\mathsf{btw}}$ and let

$$\mathfrak{t}(l) = \langle \mathbf{w}, (\mathsf{pred}(\mathbf{v}) \Leftarrow (a \mapsto \mathbf{b}) \star \mathsf{pred}_1(\mathbf{v_1}) \star \cdots \star \mathsf{pred}_m(\mathbf{v_m}) \star \Pi) \rangle ,$$

where Π is a pure constraint. Note that, by progress of the SID Φ, all rule instances are of this form. We define:

- $\mathsf{succ}_{\mathfrak{t}}(l) \triangleq \mathbf{w}$
- $\mathsf{head}_{\mathfrak{t}}(l) \triangleq \mathsf{pred}(\mathbf{v})$,
- $\mathsf{heap}_{\mathfrak{t}}(l) \triangleq \{a \mapsto \mathbf{b}\}$,
- $\mathsf{calls}_{\mathfrak{t}}(l) \triangleq \{\mathsf{pred}_1(\mathbf{v_1}), \ldots, \mathsf{pred}_m(\mathbf{v_m})\}$, and
- $\mathsf{rule}_{\mathfrak{t}}(l) \triangleq \mathsf{pred}(\mathbf{v}) \Leftarrow (a \mapsto \mathbf{b}) \star \mathsf{pred}_1(\mathbf{v_1}) \star \cdots \star \mathsf{pred}_m(\mathbf{v_m}) \star \Pi$.

Moreover, we define the *hole predicates* of l as those predicate calls in $\mathsf{calls}_{\mathfrak{t}}(l)$ whose root does not occur in $\mathsf{succ}_{\mathfrak{t}}(l)$; and the *holes* as the corresponding root locations:

- $\mathsf{holepreds}_{\mathfrak{t}}(l) \triangleq \{\mathsf{pred}'(\mathbf{z}') \in \mathsf{calls}_{\mathfrak{t}}(l) \mid \forall c \in \mathsf{succ}_{\mathfrak{t}}(l).\, \mathsf{head}_{\mathfrak{t}}(c) \neq \mathsf{pred}'(\mathbf{z}')\}$, and
- $\mathsf{holes}_{\mathfrak{t}}(l) \triangleq \left\{\mathsf{predroot}(\mathsf{pred}'(\mathbf{z}')) \mid \mathsf{pred}'(\mathbf{z}') \in \mathsf{holepreds}_{\mathfrak{t}}(l)\right\}$.

We lift some of the definitions from individual locations l to entire trees $\mathfrak{t}$.

- $\mathsf{heap}(\mathfrak{t}) \triangleq \bigcup_{c \in \mathrm{dom}(\mathfrak{t})} \mathsf{heap}_{\mathfrak{t}}(c)$
- $\mathsf{ptrlocs}(\mathfrak{t}) \triangleq \bigcup_{(c \mapsto \mathbf{d}) \in \mathsf{heap}(\mathfrak{t})} \{c\} \cup \mathbf{d}$
- $\mathsf{allholes}(\mathfrak{t}) \triangleq \bigcup_{l \in \mathrm{dom}(\mathfrak{t})} \mathsf{holes}_{\mathfrak{t}}(l)$
- $\mathsf{allholepreds}(\mathfrak{t}) \triangleq \bigcup_{c \in \mathrm{dom}(\mathfrak{t})} \mathsf{holepreds}_{\mathfrak{t}}(c)$

Finally, we define the projection of $\mathfrak{t}$ onto the *directed graph* (cf. Definition 4.1), $\mathsf{graph}(\mathfrak{t})$, induced by its first component,

$$\mathsf{graph}(\mathfrak{t}) \triangleq \langle \mathrm{dom}(\mathfrak{t}), \{\langle x, y \rangle \mid x \in \mathrm{dom}(\mathfrak{t}), y \in \mathsf{succ}_{\mathfrak{t}}(x)\} \rangle .$$

We denote by $\mathsf{height}(\mathfrak{t})$ the length of the longest path in $\mathsf{graph}(\mathfrak{t})$.

Definition 10.1 (Φ-Tree). *Let $\Phi \in \mathbf{ID}_{\mathsf{btw}}$. A partial function*

$$\mathfrak{t} \colon \mathbf{Loc} \rightharpoonup \left(2^{\mathbf{Loc}} \times \mathbf{RuleInst}(\Phi)\right)$$

is a Φ-tree *iff*

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

1. $\mathsf{graph}(\mathfrak{t})$ *is a directed tree and*

2. $\mathfrak{t}$ *is* Φ-consistent, *i.e., for all* $l \in \mathrm{dom}(\mathfrak{t})$, *if* $\mathsf{succ}_\mathfrak{t}(l) = \langle v_1, \ldots, v_k \rangle$, $\mathsf{calls}_\mathfrak{t}(l) = \{\mathsf{pred}_1(\mathbf{v_1}), \ldots, \mathsf{pred}_m(\mathbf{v_m})\}$, $\mathsf{head}_\mathfrak{t}(l) = \mathsf{pred}(\mathbf{v})$, *and* $\mathsf{heap}_\mathfrak{t}(l) = \{a \mapsto \mathbf{b}\}$ *then*
   - $l = a$,
   - $\mathsf{succ}_\mathfrak{t}(l) \subseteq \mathbf{b}$, *and*
   - $\{\mathsf{head}_\mathfrak{t}(v_1), \ldots, \mathsf{head}_\mathfrak{t}(v_k)\} \subseteq \{\mathsf{pred}_1(\mathbf{v_1}), \ldots, \mathsf{pred}_m(\mathbf{v_m})\}$.

Let $\mathfrak{t}$ *be a* Φ*-tree. As* $\mathfrak{t}$ *is a directed tree, it has a root, which we denote by* $\mathsf{root}(\mathfrak{t})$. *We set* $\mathsf{rootpred}(\mathfrak{t}) \triangleq \mathsf{head}_\mathfrak{t}(\mathsf{root}(\mathfrak{t}))$.

Example 10.2 (Φ-Tree). 1. *Let*

$$\mathfrak{t}(l) \triangleq \begin{cases} \langle b, \mathsf{even}(l_1, a) \Leftarrow (l_1 \mapsto b) \star \mathsf{odd}(b, a) \rangle & \text{if } l = l_1 \\ \langle \emptyset, \mathsf{odd}(b, a) \Leftarrow (b \mapsto l_2) \star \mathsf{even}(l_2, a) \rangle & \text{if } l = b \\ \bot & \text{otherwise.} \end{cases}$$

Then $\mathfrak{t}$ *is a* Φ*-tree with* $\mathrm{dom}(\mathfrak{t}) = \{l_1, b\}$, $\mathsf{succ}_\mathfrak{t}(l_1) = b$, $\mathsf{head}_\mathfrak{t}(l_1) = \mathsf{even}(l_1, a)$, $\mathsf{calls}_\mathfrak{t}(l_1) = \{\mathsf{odd}(b, a)\}$, $\mathsf{heap}(\mathfrak{t}) = \{l_1 \mapsto b, b \mapsto l_2\}$, $\mathsf{heap}_\mathfrak{t}(l_1) = \{l_1 \mapsto b\}$, $\mathsf{ptrlocs}(\mathfrak{t}) = \{l_1, b, l_2\}$, $\mathsf{allholes}(\mathfrak{t}) = \{l_2\}$, *and* $\mathsf{allholepreds}(\mathfrak{t}) = \{\mathsf{even}(l_2, a)\}$.

2. *Chapter 9 contains several examples of* Φ*-trees.*

Handling duplicate holes?

You may have observed that the definitions of holes and hole predicates do not work correctly if a rule instance contains multiple predicate calls with the same root parameter, or even multiple identical predicate calls. We will see in Chapter 11 that we do not need to consider such trees. We can thus get away with this imprecision, obtaining a simpler formalization of Φ-trees.

We combine zero or more Φ-trees into Φ-*forests*.

Definition 10.3 (Φ-Forest). *Let* $\Phi \in \mathbf{ID}_{\mathsf{btw}}$. *Let* $\mathfrak{t}_1, \ldots, \mathfrak{t}_k$ *be* Φ*-trees. The set* $\mathfrak{f} = \{\mathfrak{t}_1, \ldots, \mathfrak{t}_k\}$ *is a* Φ-forest *iff* $\mathrm{dom}(\mathfrak{t}_i) \cap \mathrm{dom}(\mathfrak{t}_j) = \emptyset$ *for* $i \neq j$.

We assume that all definitions are lifted from Φ-trees to Φ-forests in the obvious way. In particular, for forest $\mathfrak{f} = \{\mathfrak{t}_1, \ldots, \mathfrak{t}_k\}$, we define

- the *induced heap of* $\mathfrak{f}$ as $\mathsf{heap}(\mathfrak{f}) \triangleq \bigcup_{\mathfrak{t} \in \mathfrak{f}} \mathsf{heap}(\mathfrak{t})$;
- $\mathrm{dom}(\mathfrak{f}) \triangleq \bigcup_i \mathrm{dom}(\mathfrak{t}_i)$;
- $\mathsf{graph}(\mathfrak{f}) \triangleq \big\langle \mathrm{dom}(\mathfrak{f}), \{\langle x, y \rangle \mid 1 \leq i \leq k, x \in \mathrm{dom}(\mathfrak{t}_i), y \in \mathsf{succ}_{\mathfrak{t}_i}(x)\} \big\rangle$;

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

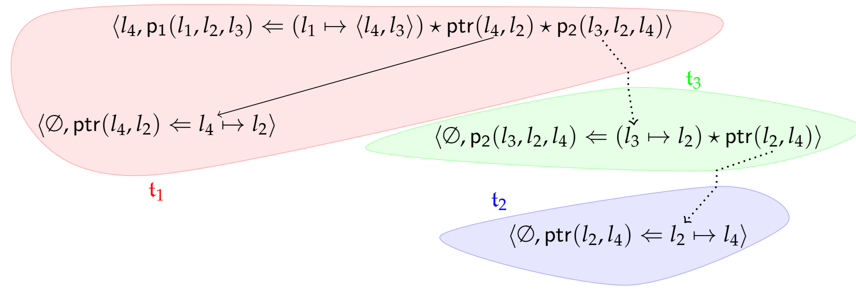


Figure 10.1: The Φ-forest defined in Example 10.4. The edge within $\mathfrak{t}_1$ illustrates that $l_4 \in \mathsf{succ}_{\mathfrak{t}_1}(l_1)$. The holes of $\mathfrak{t}_1$ and $\mathfrak{t}_3$ are the roots of $\mathfrak{t}_3$ and $\mathfrak{t}_2$, respectively, indicated by the dotted edges.

- $\mathsf{roots}(\mathfrak{f}) \triangleq \{\mathsf{root}(\mathfrak{t}_i) \mid 1 \le i \le k\}$;
- $\mathsf{allholes}(\mathfrak{f}) \triangleq \bigcup_{1\le i\le k} \mathsf{allholes}(\mathfrak{t}_i)$;
- and if $l \in \mathrm{dom}(\mathfrak{t}_i)$ then $\mathsf{rule}_{\mathfrak{f}}(l) = \mathsf{rule}_{\mathfrak{t}_i}(l)$.

Example 10.4 (Φ-Forest). *1. Example 9.3 defines a Φ-forest.*

2. Consider the following SID Φ.

$$\begin{array}{lcl}
\mathsf{ptr}(x_1,x_2) & \Leftarrow & x_1 \mapsto x_2 \\
\mathsf{p}_1(x_1,x_2,x_3) & \Leftarrow & \exists y.\,(x_1 \mapsto \langle y, x_3\rangle) \star \mathsf{ptr}(y,x_2) \star \mathsf{p}_2(x_3,x_2,y) \\
\mathsf{p}_2(x_1,x_2,x_3) & \Leftarrow & (x_1 \mapsto x_2) \star \mathsf{ptr}(x_2,x_3) \\
\mathsf{q}_1(x_1,x_2,x_3) & \Leftarrow & \exists y.\,(x_1 \mapsto \langle y, x_3\rangle) \star \mathsf{q}_2(y,x_2) \\
\mathsf{q}_2(x_1,x_2) & \Leftarrow & (x_1 \mapsto x_2) \star \mathsf{ptr}(x_2,x_1)
\end{array}$$

Figure 10.1 displays the Φ-forest $\mathfrak{f} = \{\mathfrak{t}_1, \mathfrak{t}_2, \mathfrak{t}_3\}$, where

- $\mathfrak{t}_1(l_1) = \langle l_4, \mathsf{p}_1(l_1,l_2,l_3) \Leftarrow (l_1 \mapsto \langle l_4, l_3\rangle) \star \mathsf{ptr}(l_4,l_2) \star \mathsf{p}_2(l_3,l_2,l_4)\rangle$.
- $\mathfrak{t}_1(l_4) = \langle \emptyset, \mathsf{ptr}(l_4,l_2) \Leftarrow l_4 \mapsto l_2\rangle$
- $\mathfrak{t}_2(l_2) = \langle \emptyset, \mathsf{ptr}(l_2,l_4) \Leftarrow l_2 \mapsto l_4\rangle$
- $\mathfrak{t}_3(l_3) = \langle \emptyset, \mathsf{p}_2(l_3,l_2,l_4) \Leftarrow (l_3 \mapsto l_2) \star \mathsf{ptr}(l_2,l_4)\rangle$
- *The trees $\mathfrak{t}_i, 1 \le i \le 3$, are undefined on all other locations.*

Every model of a predicate call corresponds to (at least one) Φ-tree.

Lemma 10.5. *If $(\mathfrak{s},\mathfrak{h}) \models_\Phi \mathsf{pred}(z_1,\ldots,z_k)$, then there exists a Φ-tree $\mathfrak{t}$ with $\mathsf{rootpred}(\mathfrak{t}) = \mathsf{pred}(\mathfrak{s}(z_1),\ldots,\mathfrak{s}(z_k))$, $\mathsf{allholepreds}(\mathfrak{t}) = \emptyset$, and $\mathsf{heap}(\{\mathfrak{t}\}) = \mathfrak{h}$.*

Proof. The statement directly follows by induction on the number of rules applied to derive $(\mathfrak{s},\mathfrak{h}) \models_\Phi \mathsf{pred}(z_1,\ldots,z_k)$. □

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

10.1.1 *Composing Forests*

As motivated in Chapter 9, forests are composed by (1) taking their disjoint union and (2) optionally merging pairs of trees of the resulting forest by identifying the root of one tree with a hole of another tree.

DISJOINT UNION OF FORESTS. The union of two Φ-forests corresponds to ordinary set union, provided no location is in the domain of both forests; otherwise, it is undefined.

Definition 10.6 (Union of Φ-forests). *Let $\mathfrak{f}_1, \mathfrak{f}_2$ be Φ-forests. The* union *of $\mathfrak{f}_1$, $\mathfrak{f}_2$ is given by*

$$\mathfrak{f}_1 \uplus \mathfrak{f}_2 \triangleq \begin{cases} \mathfrak{f}_1 \cup \mathfrak{f}_2 & \text{if } \mathsf{dom}(\mathfrak{f}_1) \cap \mathsf{dom}(\mathfrak{f}_2) = \emptyset, \\ \bot, & \text{otherwise.} \end{cases}$$

Lemma 10.7. *Let $\mathfrak{f} = \mathfrak{f}_1 \uplus \mathfrak{f}_2$. Then $\mathsf{heap}(\mathfrak{f}) = \mathsf{heap}(\mathfrak{f}_1) + \mathsf{heap}(\mathfrak{f}_2)$.*

Proof. $\mathsf{heap}(\mathfrak{f}) = \bigcup_{\mathfrak{t}\in\mathfrak{f}} \mathsf{heap}(\mathfrak{t}) = (\bigcup_{\mathfrak{t}\in\mathfrak{f}_1} \mathsf{heap}(\mathfrak{t})) \cup (\bigcup_{\mathfrak{t}\in\mathfrak{f}_2} \mathsf{heap}(\mathfrak{t})) = \mathsf{heap}(\mathfrak{f}_1) + \mathsf{heap}(\mathfrak{f}_2)$. (Where we have $+$ rather than $\cup$ because $\mathfrak{f}_1 \uplus \mathfrak{f}_2$ is defined.) □

SPLITTING FORESTS. We formalize the process of merging Φ-trees in a roundabout way: we first define a way to *split* the trees of a forest into sub-trees at a fixed set of locations—the reverse operation of the merge operation. This may seem like an arbitrary choice, but will simplify the development in later sections and chapters.

I introduce the split operation by example before formalizing it in Definition 10.9.

Example 10.8 (Splitting forests).
1. *Let $\mathfrak{t}$ be the Φ-tree from Example 10.2. The $\{b\}$-split of $\{\mathfrak{t}\}$ is given by $\{\mathfrak{t}_1, \mathfrak{t}_2\}$, for*
   - $\mathfrak{t}_1 = \{l_1 \mapsto \langle \emptyset, \mathsf{even}(l_1, a) \Leftarrow (l_1 \mapsto b) \star \mathsf{odd}(b, a)\rangle\}$,
   - $\mathfrak{t}_2 = \{b \mapsto \langle \emptyset, \mathsf{odd}(b, a) \Leftarrow (b \mapsto l_2) \star \mathsf{even}(l_2, a)\rangle\}$.

   In fact, $\{\mathfrak{t}_1, \mathfrak{t}_2\}$ is the $\mathbf{l}$-split of $\{\mathfrak{t}\}$ for all $\mathbf{l} \supseteq \{b\}$: in our definition of $\mathbf{l}$-split we will not require for the locations in $\mathbf{l}$ to occur in the forest.

2. *Recall the forest $\mathfrak{f} = \{\mathfrak{t}_1, \mathfrak{t}_2, \mathfrak{t}_3\}$ from Example 9.3 and the tree $\mathfrak{t}$ from Example 9.1. Then $\mathfrak{f}$ is the $\{2,4\}$-split of $\{\mathfrak{t}\}$. Likewise, $\mathfrak{f}$ is the $\{1,2,4,7\}$-split of $\{\mathfrak{t}\}$, because 1 is already the root of a tree and 7 does not occur in the forest. In contrast, $\mathfrak{f}$ is* not *the $\{1,2,5\}$-split of $\{\mathfrak{t}\}$, because $5 \in \mathsf{dom}(\mathfrak{f}) \setminus \mathsf{roots}(\mathfrak{f})$.*

Definition 10.9 ($\mathbf{l}$-split). *Let $\mathfrak{f}, \bar{\mathfrak{f}}$ be forests and $\mathbf{l} \subseteq \mathbf{Loc}$. $\bar{\mathfrak{f}}$ is an $\mathbf{l}$-split of $\mathfrak{f}$ if (1) $\mathsf{dom}(\mathfrak{f}) = \mathsf{dom}(\bar{\mathfrak{f}})$, (2) $\mathsf{rule}_{\mathfrak{f}}(d) = \mathsf{rule}_{\bar{\mathfrak{f}}}(d)$ for all $d \in \mathsf{dom}(\mathfrak{f})$, and (3) $\mathsf{graph}(\bar{\mathfrak{f}}) = \mathsf{graph}(\mathfrak{f}) \setminus \{(a,b) \mid a \in \mathbf{Loc}, b \in \mathbf{l}\}$.*

Lemma 10.10 (Uniqueness of $\mathbf{l}$-split). *Every Φ-forest $\mathfrak{f}$ has a unique $\mathbf{l}$-split.*

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Proof. Let $\langle V_{\mathcal{G}}, E_{\mathcal{G}} \rangle \triangleq \mathsf{graph}(\mathfrak{f})$. We consider the graph

$$\mathcal{G} \triangleq \langle V_{\mathcal{G}}, E_{\mathcal{G}} \setminus \{\langle a, b\rangle \mid a \in \mathbf{Loc}, b \in \mathbf{l}\}\rangle .$$

We now consider the connected components (cf. Definition 4.17) $\mathcal{C}_1, \ldots, \mathcal{C}_k$ of $\mathcal{G}$. Since $\mathsf{graph}(\mathfrak{f})$ is a forest and $\mathcal{G} \subseteq \mathsf{graph}(\mathfrak{f})$, $\mathcal{G}$ is a forest, i.e., all the connected components $\mathcal{C}_i$ of $\mathcal{G}$ are trees. Hence, the connected components induce a Φ-forest $\bar{\mathfrak{f}}$: Let $\mathsf{locs}(\mathcal{C}_i)$ be all locations that occur in $\mathcal{C}_i$ and let $\mathsf{succ}_{\mathcal{C}_i}(a)$, $a \in \mathbf{Loc}$, be the maximal set of locations such that $\{a\} \times \mathsf{succ}_{\mathcal{C}_i}(a) \subseteq \mathcal{C}_i$. Define

$$\begin{aligned} \mathfrak{t}_i &\triangleq \{a \mapsto \langle \mathsf{succ}_{\mathcal{C}_i}(a), \mathsf{rule}_{\mathfrak{f}}(a)\rangle \mid a \in \mathsf{locs}(\mathcal{C}_i)\} , \\ \bar{\mathfrak{f}} &\triangleq \{\mathfrak{t}_1, \ldots, \mathfrak{t}_n\} . \end{aligned}$$

By construction, the forest $\bar{\mathfrak{f}}$ is an $\mathbf{l}$-split of $\mathfrak{f}$. Moreover, because every $\mathbf{l}$-split must have the same domain and same rule instances as $\mathfrak{f}$ and because every connected component necessarily gives rise to a single Φ-tree, the $\mathbf{l}$-split is unique. □

From now on, we denote by $\mathsf{split}(\mathfrak{f}, \mathbf{l})$ the unique $\mathbf{l}$-split of $\mathfrak{f}$.

Definition 10.11 (Forest derivation). *Let $\mathfrak{f}_1, \mathfrak{f}_2$ be forests. $\mathfrak{f}_2$ is* one-step derivable *from $\mathfrak{f}_1$, denoted $\mathfrak{f}_1 \blacktriangleright \mathfrak{f}_2$ iff there exists a location $l \in \mathrm{dom}(\mathfrak{f})$ with* $\mathfrak{f}_1 = \mathsf{split}(\mathfrak{f}_2, \{l\})$.

We denote by $\blacktriangleright^*$ the reflexive–transitive closure of $\blacktriangleright$ and say that $\mathfrak{f}_2$ is *derivable* from $\mathfrak{f}_1$ if $\mathfrak{f}_1 \blacktriangleright^* \mathfrak{f}_2$ holds. Intuitively, $\mathfrak{f}_1 \blacktriangleright^* \mathfrak{f}_2$ holds if splitting the trees in $\mathfrak{f}_2$ at zero or more locations yields $\mathfrak{f}_1$; or, equivalently, if "merging" zero or more trees of $\mathfrak{f}_1$ yields $\mathfrak{f}_2$.

Lemma 10.12. *$\mathfrak{f}_1 \blacktriangleright^* \mathfrak{f}_2$ iff there exists a set of locations $\mathbf{l}$ with $\mathfrak{f}_1 =$ $\mathsf{split}(\mathfrak{f}_2, \mathbf{l})$.*

Proof. This follows from the observation that

$$\mathsf{split}(\mathfrak{f}, \{l_1, \ldots, l_k\}) = \mathsf{split}(\ldots \mathsf{split}(\mathsf{split}(\mathfrak{f}, \{l_1\}), \{l_2\}), \ldots, \{l_k\}). \square$$

Example 10.13. *Let*

$$\begin{aligned} \mathfrak{t}_1 &\triangleq \{l_1 \mapsto \langle \emptyset, \mathsf{odd}(l_1, l_4) \Leftarrow (l_1 \mapsto l_2) \star \mathsf{even}(l_2, l_4)\rangle\} \\ \mathfrak{t}_2 &\triangleq \{l_2 \mapsto \langle l_3, \mathsf{even}(l_2, l_4) \Leftarrow (l_2 \mapsto l_3) \star \mathsf{odd}(l_3, l_4)\rangle , \\ &\qquad l_3 \mapsto \langle \emptyset, \mathsf{odd}(l_3, l_4) \Leftarrow (l_3 \mapsto l_4)\rangle\} \\ \mathfrak{f} &\triangleq \{\mathfrak{t}_1, \mathfrak{t}_2\}. \end{aligned}$$

Let $\bar{\mathfrak{f}} \triangleq \{\bar{\mathfrak{t}}\}$ for

$$\begin{aligned} \bar{\mathfrak{t}} \triangleq \{&l_1 \mapsto \langle l_2, \mathsf{odd}(l_1, l_4) \Leftarrow (l_1 \mapsto l_2) \star \mathsf{even}(l_2, l_4)\rangle , \\ &l_2 \mapsto \langle l_3, \mathsf{even}(l_2, l_4) \Leftarrow (l_2 \mapsto l_3) \star \mathsf{odd}(l_3, l_4)\rangle , \\ &l_3 \mapsto \langle \emptyset, \mathsf{odd}(l_3, l_4) \Leftarrow (l_3 \mapsto l_4)\rangle .\} \end{aligned}$$

Then $\mathfrak{f} \blacktriangleright \bar{\mathfrak{f}}$.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Forests that are in the $\blacktriangleright^*$ relation have the same induced model.

Lemma 10.14. *Let* $\mathfrak{f}$ *be a* Φ*-forest and* $\bar{\mathfrak{f}} \blacktriangleright^* \mathfrak{f}$. *Then* $\mathsf{heap}(\bar{\mathfrak{f}}) = \mathsf{heap}(\mathfrak{f})$.

Proof. Since $\bar{\mathfrak{f}} \blacktriangleright^* \mathfrak{f}$, there exists by Lemma 10.12 a set of locations $\mathbf{l}$ with $\bar{\mathfrak{f}} = \mathsf{split}(\mathfrak{f}, \mathbf{l})$. By definition of $\mathbf{l}$-splits, we have (1) $\mathrm{dom}(\bar{\mathfrak{f}}) = \mathrm{dom}(\mathfrak{f})$ and (2) for every location $l \in \mathrm{dom}(\bar{\mathfrak{f}})$ that $\mathsf{rule}_{\bar{\mathfrak{f}}}(l) = \mathsf{rule}_{\mathfrak{f}}(l)$. Consequently, $\mathsf{heap}(\bar{\mathfrak{f}}) = \mathsf{heap}(\mathfrak{f})$. □

The derivation relation $\blacktriangleright^*$ induces the *composition* operation on pairs of forests that we motivated at the beginning of this section.

Definition 10.15 (Forest composition). *Let* $\mathfrak{f}_1, \mathfrak{f}_2$ *be* Φ*-forests. The* composition *of* $\mathfrak{f}_1$ *and* $\mathfrak{f}_2$ *is given by* $\mathfrak{f}_1 \bullet_{\mathbf{F}} \mathfrak{f}_2 \triangleq \{\mathfrak{f} \mid \mathfrak{f}_1 \uplus \mathfrak{f}_2 \blacktriangleright^* \mathfrak{f}\}$.

10.2 FOREST PROJECTIONS

In this section, we define the *projection* of Φ-forests onto formulas. In Chapter 9, I gave a "morally correct" introduction to these projections. In the present chapter, I will employ a form of *guarded quantification* in projections to turn morally correct projections into actually correct projections. I formalize these quantifiers in Section 10.2.1, then define *stack–forest projection* in two steps in Sections 10.2.2 and 10.2.3.

In the end, we will obtain a projection function for stack–forest pairs that works as follows. Given a stack $\mathfrak{s}$ and a Φ-forest $\mathfrak{f} = \{\mathfrak{t}_1, \ldots, \mathfrak{t}_k\}$,

1. we compute the formula

$$\phi \triangleq \bigstar_{1 \leq i \leq k} \left(\bigstar \mathsf{allholepreds}(\mathfrak{t}_i)\right) \mathrel{-\!\!\star} \mathsf{rootpred}(\mathfrak{t}_i),$$

   in which all parameters of all predicate calls are locations;

2. we replace in ϕ every location $v \in \mathrm{img}(\mathfrak{s})$ by an arbitrary but fixed variable x for which $\mathfrak{s}(x) = v$ holds;

3. we replace every location $v \in \mathsf{locs}(\mathsf{heap}(\mathfrak{f}))$ with $v \notin \mathrm{img}(\mathfrak{s})$ by a *guarded existential*;

4. we replace every other location by a *guarded universal*.

This process is exactly like the process proposed in Chapter 9, except for the use of guarded quantifiers instead of standard quantifiers.

10.2.1 *Interlude: Guarded Quantifiers*

We introduce *guarded* versions of both existential and universal quantifiers, which we denote $\exists\!\!\exists$ and $\forall\!\!\forall$. Specifically, we consider formulas of the form

$$\exists\!\!\exists \mathbf{e}.\, (\forall\!\!\forall \mathbf{a_1}.\, \phi_{\mathsf{qf}}) \star \cdots \star (\forall\!\!\forall \mathbf{a_k}.\, \phi_{\mathsf{qf}}),$$

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

where ϕ_{qf} denotes $\mathbf{SLID}^{\mathsf{qf}}_{\mathsf{btw}}$ formulas as defined in Section 8.1. We collect all formulas of this form in the set $\mathbf{SLID}^{\exists\forall}_{\mathsf{btw}}$; $\mathbf{SLID}^{\exists\forall}_{\mathsf{btw}}$ formulas without guarded existentials are collected in $\mathbf{SLID}^{\forall}_{\mathsf{btw}}$.

The guarded quantifiers have the following semantics.

- $(\mathfrak{s},\mathfrak{h}) \models_\Phi \exists \langle e_1,\ldots,e_k\rangle \,.\, \phi$ iff there exist pairwise different locations
  $$v_1,\ldots,v_k \in \mathrm{dom}(\mathfrak{h}) \setminus \mathrm{img}(\mathfrak{s})$$
  such that
  $$(\mathfrak{s} \cup \{e_1 \mapsto v_1,\ldots,e_k \mapsto v_k\}, \mathfrak{h}) \models_\Phi \phi.$$
- $(\mathfrak{s},\mathfrak{h}) \models_\Phi \forall \langle a_1,\ldots,a_k\rangle \,.\, \phi$ iff for all pairwise different locations
  $$v_1,\ldots,v_k \in \mathbf{Loc} \setminus (\mathsf{locs}(\mathfrak{h}) \cup \mathrm{img}(\mathfrak{s})),$$
  it holds that
  $$(\mathfrak{s} \cup \{a_1 \mapsto v_1,\ldots,a_k \mapsto v_k\}, \mathfrak{h}) \models_\Phi \phi.$$

I would like to stress three things. First, quantifiers cannot be instantiated with locations that are already in the stack. Second, it is crucial that the locations are pairwise different. Third, the quantifiers are *not* dual, i.e., $\exists \mathbf{x}.\,\phi$ is *not* equivalent to $\neg\forall\mathbf{x}.\,\neg\phi$. In particular, $\forall$ ranges over *all* locations that are not in $\mathsf{locs}(\mathfrak{h})$, whereas $\exists$ ranges only over *some* locations that are in $\mathsf{locs}(\mathfrak{h})$, namely $\mathrm{dom}(\mathfrak{h})$.

Location terms in a formula can be replaced by a guarded universal if the locations are not used in the model.

Lemma 10.16. *Let* $\phi \in \mathbf{SLID}^{\mathsf{qf}}$*, let* $\mathbf{v} \in (\mathbf{Loc} \setminus (\mathsf{locs}(\mathfrak{h}) \cup \mathrm{img}(\mathfrak{s})))^*$ *be repetition free, and assume that* $(\mathfrak{s},\mathfrak{h}) \models_\Phi \phi$*. Let* $\mathbf{a} \triangleq \{a_1,\ldots,a_{|\mathbf{v}|}\}$ *such that* $\mathbf{a} \cap \mathrm{dom}(\mathfrak{s}) = \emptyset$*. Then* $(\mathfrak{s},\mathfrak{h}) \models_\Phi \forall\mathbf{a}.\,\phi[\mathbf{v}/\mathbf{a}]$*.*

Proof sketch. Let $\mathbf{w} \in (\mathbf{Loc} \setminus (\mathsf{locs}(\mathfrak{h}) \cup \mathrm{img}(\mathfrak{s})))^*$ be a repetition-free sequence of locations with $|\mathbf{v}| = |\mathbf{w}|$. Since neither $\mathbf{v}$ nor $\mathbf{w}$ intersect with $\mathsf{locs}(\mathfrak{h})$ or $\mathrm{img}(\mathfrak{s})$, it follows that $(\mathfrak{s},\mathfrak{h}) \models_\Phi \phi[\mathbf{v}/\mathbf{w}]$ Since $\mathbf{w}$ was arbitrary, $(\mathfrak{s},\mathfrak{h}) \models_\Phi \forall\mathbf{a}.\,\phi[\mathbf{v}/\mathbf{a}]$ by the semantics of $\forall$. ☐

The semi-distributivity of quantifiers in separation logic[1] also holds for guarded quantifiers.

Lemma 10.17. *Let* ϕ,ψ *be formulas with* $\mathsf{fvars}(\psi) \cap \mathbf{z} = \emptyset$*. Let* $\mathbb{Q} \in \{\exists,\forall\}$ *and* $(\mathfrak{s},\mathfrak{h}) \models_\Phi (\mathbb{Q}\mathbf{z}.\,\phi) \star \psi$*. Then* $(\mathfrak{s},\mathfrak{h}) \models_\Phi \mathbb{Q}\mathbf{z}.\,(\phi \star \psi)$*.*

Proof. CASE $\mathbb{Q} = \exists$. Assume $(\mathfrak{s},\mathfrak{h}) \models_\Phi (\exists\mathbf{z}.\,\phi) \star \psi$. Then there exist $\mathfrak{h}_1,\mathfrak{h}_2$ with $\mathfrak{h} = \mathfrak{h}_1 + \mathfrak{h}_2$, $(\mathfrak{s},\mathfrak{h}_1) \models_\Phi (\exists\mathbf{z}.\,\phi)$ and $(\mathfrak{s},\mathfrak{h}_2) \models_\Phi \psi$. By

1 See, for example, [Rey02].

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

the semantics of $\mathbb{H}$, there then exists a sequence of pairwise-different locations $\mathbf{v} \in (\mathrm{dom}(\mathfrak{h}_1) \setminus \mathrm{img}(\mathfrak{s}))^*$ such that

$$(\mathfrak{s} \cup \{\mathbf{z} \mapsto \mathbf{v}\}, \mathfrak{h}_1) \models_\Phi \phi$$

holds. As $\mathsf{fvars}(\psi) \cap \mathbf{z} = \emptyset$, it follows that $(\mathfrak{s} \cup \{\mathbf{z} \mapsto \mathbf{v}\}, \mathfrak{h}) \models_\Phi \phi \star \psi$. Since $\mathrm{dom}(\mathfrak{h}_1) \subseteq \mathrm{dom}(\mathfrak{h})$, $\mathbf{v} \in (\mathrm{dom}(\mathfrak{h}_1) \setminus \mathrm{img}(\mathfrak{s}))^*$ implies that $\mathbf{v} \in (\mathrm{dom}(\mathfrak{h}) \setminus \mathrm{img}(\mathfrak{s}))^*$. It follows by the semantics of $\mathbb{H}$ that $(\mathfrak{s}, \mathfrak{h}) \models_\Phi \mathbb{H}\mathbf{z}.\, \phi \star \psi$.

CASE $\mathbb{Q} = \mathbb{V}$. Assume $(\mathfrak{s}, \mathfrak{h}) \models_\Phi (\mathbb{V}\mathbf{z}.\, \phi) \star \psi$. Then there exist $\mathfrak{h}_1, \mathfrak{h}_2$ with $\mathfrak{h} = \mathfrak{h}_1 + \mathfrak{h}_2$, $(\mathfrak{s}, \mathfrak{h}_1) \models_\Phi (\mathbb{V}\mathbf{z}.\, \phi)$ and $(\mathfrak{s}, \mathfrak{h}_2) \models_\Phi \psi$. By the semantics of $\mathbb{V}$, it then holds for all sequences of pairwise-different locations $\mathbf{v} \in (\mathbf{Loc} \setminus (\mathsf{locs}(\mathfrak{h}_1) \cup \mathrm{img}(\mathfrak{s})))^*$ that $(\mathfrak{s} \cup \{\mathbf{z} \mapsto \mathbf{v}\}, \mathfrak{h}_1) \models_\Phi \phi$. As $\mathsf{fvars}(\psi) \cap \mathbf{z} = \emptyset$, it also holds for all such $\mathbf{z}$ that $(\mathfrak{s} \cup \{\mathbf{z} \mapsto \mathbf{v}\}, \mathfrak{h}) \models_\Phi \phi \star \psi$. Now observe that

$$(\mathbf{Loc} \setminus (\mathsf{locs}(\mathfrak{h}_1) \cup \mathrm{img}(\mathfrak{s}))) \supseteq (\mathbf{Loc} \setminus (\mathsf{locs}(\mathfrak{h}) \cup \mathrm{img}(\mathfrak{s}))).$$

Consequently, $\phi \star \psi$ holds in $(\mathfrak{s} \cup \{\mathbf{z} \mapsto \mathbf{v}\}, \mathfrak{h})$ for all sequences $\mathbf{v}$ drawn from $(\mathbf{Loc} \setminus (\mathsf{locs}(\mathfrak{h}) \cup \mathrm{img}(\mathfrak{s})))$. It follows by the semantics of $\mathbb{V}$ that $(\mathfrak{s}, \mathfrak{h}) \models_\Phi \mathbb{V}\mathbf{z}.\, \phi \star \psi$. □

Moving any guarded quantifier out results in a *strictly* weaker formula. In particular, $\mathbb{H}\mathbf{z}.\, (\phi \star \psi) \not\models (\mathbb{H}\mathbf{z}.\, \phi) \star \psi$ even if $\mathbf{z} \cap \mathsf{fvars}(\psi) = \emptyset$. For example, $\mathbb{H}z.\, (y \mapsto z) \star \mathsf{lseg}(x, y)$ is satisfiable (by a model in which y points to an inner node of the list segment $\mathsf{lseg}(x, y)$, i.e., z is identified with an inner node of the list), whereas $(\mathbb{H}z.\, y \mapsto z) \star \mathsf{lseg}(x, y)$ is unsatisfiable (because the only allocated location in a model of $(\mathbb{H}z.\, y \mapsto z)$ would be $\mathfrak{s}(y)$, but guarded existentials can only be instantiated with locations that are not in $\mathrm{img}(\mathfrak{s})$). In contrast, $\exists \mathbf{z}.\, (\phi \star \psi) \models (\exists \mathbf{z}.\, \phi) \star \psi$ holds if $\mathbf{z} \cap \mathsf{fvars}(\psi) = \emptyset$.

Many of the standard equivalences of separation logic continue to hold for formulas with guarded quantifiers, however. For the sake of completeness, I define in Fig. 10.2 the *rewriting equivalence*, an equivalence relation $\equiv$ on formulas with guarded quantifiers that includes all the rewriting rules we need in the remainder of the thesis.

Lemma 10.18 (Soundness of rewriting equivalence)**.** *If* $\phi_1 \equiv \phi_2$ *then* $\phi_1 \models_\Phi \phi_2$.

10.2.2 *Forest Projections without Stacks*

We are now ready to define the projection functions. We begin with the projection of single trees. Besides the tree, we pass a set of locations $\mathbf{v}$ to the projection function. The locations $\mathbf{v}$ are blocked from being replaced by guarded universals. As we will see later, we can use this parameter to avoid introducing universals for the locations in $\mathrm{img}(\mathfrak{s})$ and $\mathsf{locs}(\mathfrak{h})$—in accordance with the semantics of $\mathbb{V}$.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

$$\frac{\phi_1 \equiv \phi_2}{\phi_1 \star \psi \equiv \phi_2 \star \psi}\text{(mono)} \qquad \frac{}{\phi_1 \star \mathbf{emp} \equiv \phi_1}\text{(emp)}$$

$$\frac{\phi_1 \equiv \phi_2}{\phi_2 \mathrel{-\!\!\star} \psi \equiv \phi_1 \mathrel{-\!\!\star} \psi}\text{(anti)} \qquad \frac{}{\phi_1 \star (\phi_2 \star \phi_3) \equiv (\phi_1 \star \phi_2) \star \phi_3}\text{(assoc)}$$

$$\frac{}{\phi_1 \star \phi_2 \equiv \phi_2 \star \phi_1}\text{(comm)} \qquad \frac{\mathbb{Q} \in \{\mathbb{V}, \mathbb{E}\} \qquad z \notin \mathsf{vars}(\phi)}{\mathbb{Q}y.\,\phi \equiv \mathbb{Q}z.\,\phi[y/z]}\text{(ren)}$$

$$\frac{\phi_1 \equiv \phi_2}{\mathbb{E}\mathbf{y}.\,\phi_1 \equiv \mathbb{E}\mathbf{y}.\,\phi_2}(\mathbb{E}\text{-intro}) \qquad \frac{\phi_1 \equiv \phi_2}{\mathbb{V}\mathbf{y}.\,\phi_1 \equiv \mathbb{V}\mathbf{y}.\,\phi_2}(\mathbb{V}\text{-intro})$$

$$\frac{\phi_1 \equiv \phi_2}{\phi_2 \equiv \phi_1}\text{(sym)} \qquad \frac{\phi_1 \equiv \phi_3 \qquad \phi_3 \equiv \phi_2}{\phi_1 \equiv \phi_2}\text{(trans)}$$

Figure 10.2: A set of rules for rewriting $\mathbf{SLID}_{\mathsf{btw}}^{\mathbb{EV}}$ formulas into equivalent formulas.

Definition 10.19 (Tree projection). *Let* $\mathfrak{t}$ *be a* Φ*-tree and let* $\mathbf{v} \subseteq \mathbf{Loc}$*. The* tree projection *of* $\mathfrak{t}$ *w.r.t.* $\mathbf{v}$, $\mathsf{project}^{\mathbf{Loc}}(\mathbf{v}, \mathfrak{t})$*, is given by*

$$\begin{aligned}
\mathsf{project}^{\mathbf{Loc}}(\mathbf{v}, \mathfrak{t}) &\triangleq \mathbb{V}\mathbf{a}.\,\psi[\mathbf{w}/\mathbf{a}] \\
&\quad \textit{where } \psi \triangleq (\bigstar \mathsf{allholepreds}(\mathfrak{t})) \mathrel{-\!\!\star} \mathsf{rootpred}(\mathfrak{t}) \\
&\quad\quad \mathbf{w} \triangleq \mathsf{locs}(\psi) \setminus (\mathsf{ptrlocs}(\mathfrak{t}) \cup \mathbf{v}) \\
&\quad\quad \mathbf{a} \triangleq \langle a_1, \ldots, a_{|\mathbf{w}|} \rangle .
\end{aligned}$$

Example 10.20 (Tree projection)**.** *Let* $\mathfrak{t}$ *be the* Φ*-tree from Example 10.2. Then*

1. $\mathsf{project}^{\mathbf{Loc}}(\emptyset, \mathfrak{t}) = \mathbb{V}a_1.\,\mathsf{even}(l_2, a_1) \mathrel{-\!\!\star} \mathsf{even}(l_1, a_1)$.
2. $\mathsf{project}^{\mathbf{Loc}}(\{l_1, l_2\}, \mathfrak{t}) = \mathbb{V}a_1.\,\mathsf{even}(l_2, a_1) \mathrel{-\!\!\star} \mathsf{even}(l_1, a_1)$.
3. $\mathsf{project}^{\mathbf{Loc}}(\{l_1, l_2, a\}, \mathfrak{t}) = \mathsf{even}(l_2, a) \mathrel{-\!\!\star} \mathsf{even}(l_1, a)$.

Tree projection is sound in the sense that the induced heap of a tree satisfies the projection of the tree. To show this soundness result, we need the following variant of modus ponens.

Lemma 10.21 (Generalized modus ponens)**.**

$$\begin{aligned}
&((\mathsf{pred}_2(\mathbf{x_2}) \star \psi) \mathrel{-\!\!\star} \mathsf{pred}_1(\mathbf{x_1})) \star (\psi' \mathrel{-\!\!\star} \mathsf{pred}_2(\mathbf{x_2})) \\
&\textit{implies } (\psi \star \psi') \mathrel{-\!\!\star} \mathsf{pred}_1(\mathbf{x_1}).
\end{aligned}$$

We write $\phi \overset{\mathsf{mp}}{\Longrightarrow} \psi$ if ϕ implies ψ by Lemma 10.21.

Lemma 10.22 (Soundness of tree projection)**.** *Let* $\mathfrak{t}$ *be a* Φ*-tree, let* $\mathbf{v} \subseteq \mathbf{Loc}$*, and let* $\mathfrak{s}$ *be a stack with* $\mathsf{img}(\mathfrak{s}) \subseteq \mathbf{v}$*. Then* $(\mathfrak{s}, \mathsf{heap}(\mathfrak{t})) \models_{\Phi} \mathsf{project}^{\mathbf{Loc}}(\mathbf{v}, \mathfrak{t})$.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Proof. We proceed by mathematical induction on $\mathsf{height}(\mathfrak{t})$. Let

$$\begin{aligned}
&r \triangleq \mathsf{root}(\mathfrak{t}),\\
&\langle s_1, \ldots, s_m \rangle \triangleq \mathsf{succ}_{\mathfrak{t}}(r),\\
&(\mathsf{pred}(\mathbf{z}) \Leftarrow (a \mapsto \mathbf{b}) \star \mathsf{pred}_1(\mathbf{z_1}) \star \cdots \star \mathsf{pred}_k(\mathbf{z_k})) \triangleq \mathsf{rule}_{\mathfrak{t}}(r),\\
&\{\mathsf{pred}_{1,1}(\mathbf{z_{1,1}}), \ldots, \mathsf{pred}_{1,m}(\mathbf{z_{1,m}})\}\\
&\quad \triangleq \{\mathsf{pred}_1(\mathbf{z_1}), \ldots, \mathsf{pred}_k(\mathbf{z_k})\} \setminus \mathsf{holepreds}_{\mathfrak{t}}(r),\\
&\{\mathsf{pred}_{2,1}(\mathbf{z_{2,1}}), \ldots, \mathsf{pred}_{2,n}(\mathbf{z_{2,n}})\} \triangleq \mathsf{holepreds}_{\mathfrak{t}}(r).
\end{aligned}$$

Moreover, let $\mathfrak{t}_i$ be the sub-tree of $\mathfrak{t}$ rooted in s_i. By the semantics of $-\!\!\star$, we have

$$\{a \mapsto \mathbf{b}\} \models_{\Phi} (\mathsf{pred}_1(\mathbf{z_1}) \star \cdots \star \mathsf{pred}_k(\mathbf{z_k})) -\!\!\star \mathsf{pred}(\mathbf{z}).$$

By the induction hypotheses for $\mathfrak{t}_1, \ldots, \mathfrak{t}_m$, we additionally have that

$$\begin{aligned}
(\mathfrak{s}, \mathsf{heap}(\{\mathfrak{t}_i\})) &\models_{\Phi} \mathsf{project}(\mathfrak{t}_i)\\
&= \forall \mathbf{a_i}.\, \underbrace{((\bigstar \mathsf{allholepreds}(\mathfrak{t}_i)) -\!\!\star \mathsf{pred}_{1,i}(\mathbf{z_{1,i}}))}_{\triangleq \psi_i}[\mathbf{v_i}/\mathbf{a_i}]
\end{aligned}$$

for appropriate choices of $\mathbf{v_i}$ and $\mathbf{a_i}$.

By definition, $\mathsf{heap}(\mathfrak{f}) = \{a \mapsto \mathbf{b}\} \bigcup \mathsf{heap}(\{\mathfrak{t}_i\})$. By the semantics of the separating conjunction, we therefore obtain

$$\begin{aligned}
\mathsf{heap}(\mathfrak{f}) \models_{\Phi} &((\mathsf{pred}_1(\mathbf{z_1}) \star \cdots \star \mathsf{pred}_k(\mathbf{z_k})) -\!\!\star \mathsf{pred}(\mathbf{z}))\\
&\star \bigstar_{1 \leq i \leq m} \psi_i[\mathbf{v_i}/\mathbf{a_i}].
\end{aligned}$$

I claim that $\mathsf{project}^{\mathbf{Loc}}(\mathbf{v}, \mathfrak{f})$ is implied by this formula, which would imply that $\mathsf{heap}(\mathfrak{f}) \models_{\Phi} \mathsf{project}^{\mathbf{Loc}}(\mathbf{v}, \mathfrak{f})$ holds.

To this end, we first instantiate the variables $\mathbf{a_i}$ with $\mathbf{v_i}$, reversing the introduction of the guarded universals. Since the $\mathbf{v_i}$ are pairwise different and occur neither in $\mathrm{img}(\mathfrak{s})$ nor in the sub-heaps of $\mathsf{heap}(\mathfrak{f})$ that satisfy $\psi_i[\mathbf{v_i}/\mathbf{a_i}]$, we are guaranteed by the semantics of guarded universals that $(\mathfrak{s}, \mathsf{heap}(\mathfrak{f}))$ satisfies the resulting formula, i.e.,

$$\begin{aligned}
(\mathfrak{s}, \mathsf{heap}(\mathfrak{f})) \models_{\Phi} &((\mathsf{pred}_1(\mathbf{z_1}) \star \cdots \star \mathsf{pred}_k(\mathbf{z_k})) -\!\!\star \mathsf{pred}(\mathbf{z}))\\
&\star \bigstar_{1 \leq i \leq m} \psi_i.
\end{aligned}$$

Let $\mathbf{v}'$ be all those locations that occur in this formula but not in $\mathsf{heap}(\mathfrak{f})$ and $\mathbf{a}' \triangleq \langle a_1, \ldots, a_{|\mathbf{v}'|} \rangle$. By Lemma 10.16,

$$\begin{aligned}
(\mathfrak{s}, \mathsf{heap}(\mathfrak{f})) \models_{\Phi} \forall \mathbf{a}'.\, \Big(&((\mathsf{pred}_1(\mathbf{z_1}) \star \cdots \star \mathsf{pred}_k(\mathbf{z_k})) -\!\!\star \mathsf{pred}(\mathbf{z}))\\
&\star \bigstar_{1 \leq i \leq m} \psi_i \Big)[\mathbf{v}'/\mathbf{a}']. \qquad (\dagger)
\end{aligned}$$

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Note that for every $1 \leq i \leq m$ there exists a $1 \leq j \leq k$ with $\mathsf{pred}_{1,i}(\mathbf{z_{1,i}}) = \mathsf{pred}_j(\mathbf{z_j})$. We apply Lemma 10.21 to (†) m times, setting $\mathsf{pred}_1(\mathbf{x_1}) \triangleq \mathsf{pred}(\mathbf{z}) = \mathsf{rootpred}(\mathfrak{t})$ and $\mathsf{pred}_2(\mathbf{x_2}) \triangleq \mathsf{pred}_{1,i}(\mathbf{z_{1,i}}) = \mathsf{rootpred}(\mathfrak{t}_i)$.

Let $\mathbf{v}'' \subseteq \mathbf{v}'$ all those locations among $\mathbf{v}'$ that still occur in the formula at the end of this process and let $\mathbf{a}'' \subseteq \mathbf{a}'$ be the corresponding variables. We obtain the following identities.

$$
\begin{aligned}
(\mathfrak{s}, \mathsf{heap}(\mathfrak{f})) \models_\Phi \mathbb{V}\mathbf{a}''.\Big(& (\bigstar_{1\leq i\leq m}(\bigstar\mathsf{allholepreds}(\mathfrak{t}_i) \star \bigstar_{1\leq j\leq n}\mathsf{pred}_{2,j}(\mathbf{z_{2,j}}))) \\
& \mathrel{-\!\!\star} \mathsf{pred}(\mathbf{z})\Big)[\mathbf{v}''/\mathbf{a}''] \\
= \mathbb{V}\mathbf{a}''.\Big(& (\bigstar_{1\leq i\leq m}(\bigstar\mathsf{allholepreds}(\mathfrak{t}_i) \star \bigstar\mathsf{holepreds}_{\mathfrak{t}}(r))) \\
& \mathrel{-\!\!\star} \mathsf{pred}(\mathbf{z})\Big)[\mathbf{v}''/\mathbf{a}''] \\
= \mathbb{V}\mathbf{a}''.\Big(& (\bigstar_{1\leq i\leq m}(\bigstar\mathsf{allholepreds}(\mathfrak{t}_i) \star \bigstar\mathsf{holepreds}_{\mathfrak{t}}(\mathsf{root}(\mathfrak{t})))) \\
& \mathrel{-\!\!\star} \mathsf{pred}(\mathbf{z})\Big)[\mathbf{v}''/\mathbf{a}''] \\
= \mathbb{V}\mathbf{a}''.\, & ((\bigstar\mathsf{allholepreds}(\mathfrak{t})) \mathrel{-\!\!\star} \mathsf{rootpred}(\mathfrak{t}))[\mathbf{v}''/\mathbf{a}''] \\
= \mathsf{project}&^{\mathbf{Loc}}(\mathbf{v}, \mathfrak{t}).
\end{aligned}
$$

□

Why we need guarded quantifiers in tree projections

Lemma 10.22 only holds because tree projection (Definition 10.19) introduces guarded universals rather than "normal," unguarded universals. For example, assume that

$$
\begin{aligned}
\Phi &= \{\mathsf{pred}(x_1, x_2, x_3) \Leftarrow (x_1 \mapsto \mathsf{nil}) \star x_1 \not\approx x_2 \star x_2 \not\approx x_3\} \\
\mathfrak{t} &= \{l_1 \mapsto \langle \emptyset, \mathsf{pred}(l_1, l_2, l_3) \Leftarrow (l_1 \mapsto 0) \star l_1 \not\approx l_2 \star l_2 \not\approx l_3 \rangle\}\,.
\end{aligned}
$$

In this case,

$$\mathsf{heap}(\mathfrak{t}) \models_\Phi \mathbb{V}\,\langle a_1, a_2\rangle\,.\,\mathbf{emp} \mathrel{-\!\!\star} \mathsf{pred}(l_1, a_1, a_2) = \mathsf{project}^{\mathbf{Loc}}(\emptyset, \mathfrak{t}),$$

but

$$
\begin{aligned}
\mathsf{heap}(\mathfrak{t}) &\not\models_\Phi \mathbf{emp} \mathrel{-\!\!\star} \mathsf{pred}(l_1, l_1, l_1) \\
\mathsf{heap}(\mathfrak{t}) &\not\models_\Phi \mathbf{emp} \mathrel{-\!\!\star} \mathsf{pred}(l_1, l_2, l_2),
\end{aligned}
$$

so in particular $\mathsf{heap}(\mathfrak{t}) \not\models_\Phi \forall\,\langle a_1, a_2\rangle\,.\,\mathbf{emp} \mathrel{-\!\!\star} \mathsf{pred}(l_1, a_1, a_2)$.[a]

[a] Adapted from an example by Nicolas Peltier.

We lift tree projections to forest projections in the obvious way.

Definition 10.23 (Forest projection). *Let $\mathfrak{f} = \{\mathfrak{t}_1, \ldots, \mathfrak{t}_k\}$ be a Φ-forest. The* forest projection *of $\mathfrak{f}$, $\mathsf{project}^{\mathbf{Loc}}(\mathbf{v}, \mathfrak{f})$, is given by*

$$\mathsf{project}^{\mathbf{Loc}}(\mathbf{v}, \mathfrak{f}) \triangleq \bigstar_{1\leq i\leq k}\, \mathsf{project}^{\mathbf{Loc}}(\mathbf{v}, \mathfrak{t}_i).$$

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

The terms in tree and forest projections are a mix of locations and variables. Strictly speaking, the forest projection is not unique, as it involves imposing an (arbitrary) order on the trees $\mathfrak{t}_1, \ldots, \mathfrak{t}_k$. Because of the commutativity and associativity of $\star$, this order does not matter, however. Put differently, changing the order of the trees in the projection yields a formula that is equivalent w.r.t. the rewriting equivalence $\equiv$ defined in Fig. 10.2.

Example 10.24 (Forest projection)**.** *Let $\mathfrak{f}$ be the forest from Example 10.13. Then*

$$\begin{aligned}\mathsf{project}^{\mathbf{Loc}}(\emptyset, \mathfrak{f}) = &(\forall a_1 . \mathsf{even}(l_2, a_1) \mathrel{-\!\!\star} \mathsf{odd}(l_1, a_1)) \\ &\star (\mathbf{emp} \mathrel{-\!\!\star} \mathsf{even}(l_2, l_4)).\end{aligned}$$

Observe that even though $l_4 \in \mathsf{ptrlocs}(\mathfrak{t}_2)$, because $l_4 \notin \mathsf{ptrlocs}(\mathfrak{t}_1)$, it is replaced with the universally-quantified variable a_1 in the part of the formula corresponding to $\mathfrak{t}_1$. In contrast, $\mathsf{project}^{\mathbf{Loc}}(\{l_4\}, \mathfrak{f}) = (\mathsf{even}(l_2, l_4) \mathrel{-\!\!\star} \mathsf{odd}(l_1, l_4)) \star (\mathbf{emp} \mathrel{-\!\!\star} \mathsf{even}(l_2, l_4))$, i.e., we can avoid the introduction of the universal by adding l_4 to the first parameter of the projection function.

Note that the separating conjunction of the projections of two forests is equivalent to the projection of the union of the forests.

Lemma 10.25. *Let $\mathfrak{f} = \mathfrak{f}_1 \uplus \mathfrak{f}_2$. Then $\mathsf{project}(\mathfrak{f}_1) \star \mathsf{project}(\mathfrak{f}_2) \equiv \mathsf{project}(\mathfrak{f})$.*

Proof. Follows from (comm) and (assoc) (combining sub-proofs as necessary via (mono) and (trans)). ☐

Using this result, the soundness of tree projections, Lemma 10.22, can be lifted from trees to forests.

Lemma 10.26. *Let $\mathfrak{f}$ be a Φ-forest, $\mathbf{v} \subseteq \mathbf{Loc}$, and let $\mathfrak{s}$ be a stack with* $\mathrm{img}(\mathfrak{s}) \subseteq \mathbf{v}$*. Then $(\mathfrak{s}, \mathsf{heap}(\mathfrak{f})) \models_\Phi \mathsf{project}^{\mathbf{Loc}}(\mathbf{v}, \mathfrak{f})$.*

Proof. Let $\mathfrak{f} = \{\mathfrak{t}_1, \ldots, \mathfrak{t}_k\}$. For each i, we have by Lemma 10.22 that $(\mathfrak{s}, \mathsf{heap}(\mathfrak{t}_i)) \models_\Phi \mathsf{project}^{\mathbf{Loc}}(\mathbf{v}, \mathfrak{t}_i)$. By definition, $\mathsf{heap}(\mathfrak{f}) = \mathsf{heap}(\mathfrak{t}_1) + \cdots + \mathsf{heap}(\mathfrak{t}_k)$. It thus follows by the semantics of $\star$ that

$$(\mathfrak{s}, \mathsf{heap}(\mathfrak{f})) \models_\Phi \mathsf{project}^{\mathbf{Loc}}(\mathbf{v}, \mathfrak{t}_1) \star \cdots \star \mathsf{project}^{\mathbf{Loc}}(\mathbf{v}, \mathfrak{t}_k).$$

By Lemma 10.25,

$$\mathsf{project}^{\mathbf{Loc}}(\mathbf{v}, \mathfrak{t}_1) \star \cdots \star \mathsf{project}^{\mathbf{Loc}}(\mathbf{v}, \mathfrak{t}_k) \equiv \mathsf{project}^{\mathbf{Loc}}(\mathbf{v}, \mathfrak{f}).$$

Finally, the soundness of $\equiv$ (Lemma 10.18) yields $(\mathfrak{s}, \mathsf{heap}(\mathfrak{f})) \models_\Phi \mathsf{project}^{\mathbf{Loc}}(\mathbf{v}, \mathfrak{f})$. ☐

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

10.2.3 *Stack–Forest Projection*

Forest projections as defined in the previous section contain locations. We now get rid of these locations based on the stack $\mathfrak{s}$. Specifically, the projection of stack–forest pairs replaces every location l in the forest projection by a variable: a stack variable, if l is in the image of the stack and an existentially-quantified variable otherwise.

In the following, we assume for all stacks that

$$\mathrm{dom}(\mathfrak{s}) \cap (\{a_1, a_2, \ldots\} \cup \{e_1, e_2, \ldots\}) = \emptyset.$$

We also need the *stack-choice function*, $\mathfrak{s}^{-1}_{\max}$, defined in Definition 2.3 on page 20.

Definition 10.27 (Stack–forest projection). *Let $\mathfrak{f} = \{\mathfrak{t}_1, \ldots, \mathfrak{t}_k\}$ be a Φ-forest and let $\mathfrak{s}$ be a stack. Let $\mathbf{w} \triangleq \mathsf{locs}(\mathsf{heap}(\mathfrak{f})) \setminus \mathrm{img}(\mathfrak{s})$ be the (arbitrarily ordered) sequence of locations that occur in a pointer in $\mathsf{heap}(\mathfrak{f})$ but are not the value of any stack variable, and let $\mathbf{e} \triangleq \langle e_1, e_2, \ldots, e_{|\mathbf{w}|} \rangle$ be a sequence of fresh variables. The* stack–forest projection *of $\mathfrak{s}$ and $\mathfrak{f}$, $\mathsf{project}(\mathfrak{s}, \mathfrak{f})$, is given by $\exists\!\!\exists \mathbf{e}.\, \mathsf{project}^{\mathbf{Loc}}(\mathrm{img}(\mathfrak{s}) \cup \mathbf{w}, \mathfrak{f})[\mathrm{dom}(\mathfrak{s}^{-1}_{\max}) \cdot \mathbf{w} / \mathrm{img}(\mathfrak{s}^{-1}_{\max}) \cdot \mathbf{e}]$.*

Note that by construction (1) $\mathrm{dom}(\mathfrak{s}^{-1}_{\max}) \cap \mathbf{w} = \emptyset$ and (2) the sequence $\mathrm{img}(\mathfrak{s}^{-1}_{\max}) \cdot \mathbf{e}$ is repetition free. Further, I am aware that the notation $\phi[\mathrm{dom}(\mathfrak{s}^{-1}_{\max}) \cdot \mathbf{w} / \mathrm{img}(\mathfrak{s}^{-1}_{\max}) \cdot \mathbf{e}]$ is not fully formal, as $\mathrm{dom}(\mathfrak{s}^{-1}_{\max})$ and $\mathrm{img}(\mathfrak{s}^{-1}_{\max})$ are sets, not sequences. Just like in earlier chapters, I assume that a suitable order is imposed on these sets to instantiate every location $v \in \mathrm{dom}(\mathfrak{s}^{-1}_{\max})$ with the variable $\mathfrak{s}^{-1}_{\max}(v)$.

Intuitively, stack–forest projection replaces all locations in the image of the stack with stack variables; and all other location terms with existentially-quantified variables. Consequently, the resulting formula no longer contains any location terms. By passing $\mathrm{img}(\mathfrak{s}) \cup \mathbf{w}$ to the call $\mathsf{project}^{\mathbf{Loc}}(\mathrm{img}(\mathfrak{s}) \cup \mathbf{w}, \mathfrak{f})$, we ensure that uses of these locations are not replaced by universals. In other words, it is guaranteed that all occurrences of locations in $\mathrm{img}(\mathfrak{s}) \cup \mathbf{w}$ are replaced by a (unique) variable in $\mathrm{dom}(\mathfrak{s}) \cup \mathbf{e}$ in the stack–forest projection.

The use of guarded existentials (as opposed to $\exists$) is for compatibility with the guarded universal: if ϕ and ψ are (the quantifier-free part of) stack–forest projections, the implication

$$(\forall\!\!\forall a.\, \phi) \star (\exists\!\!\exists e.\, \psi) \implies \exists\!\!\exists e.\, \phi[a/e] \star \psi$$

is valid. This will be crucial when *composing* projections later in this chapter (cf. Definition 10.32). In contrast,

$$(\forall\!\!\forall a.\, \phi) \star (\exists e.\, \psi) \not\Longrightarrow \exists e.\, \phi[a/e] \star \psi.$$

is *not* valid for multiple reasons: The quantifier $\exists e$ admits interpreting e by a location in $\mathrm{img}(\mathfrak{s})$ or by a location that is shared between the sub-models of $\forall\!\!\forall a.\, \phi$ and $\exists e.\, \psi$. Further, multiple existentials could be

TU Bibliothek WIEN Your knowledge hub
Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.

interpreted by the same location. In all these cases, it would not be allowed to instantiate the quantifier $\forall a$ with the location interpreting the existential, showing that the substitution $\exists e.\, \phi[a/e] \star \psi$ is indeed invalid.

Example 10.28 (Stack–forest projection). *1. Let $\mathfrak{t}$ be the Φ-tree from Example 10.2. Let $\mathfrak{f} = \{\mathfrak{t}\}$ and $\mathfrak{s} = \{x_1 \mapsto l_1, x_2 \mapsto l_2\}$. Then*

$$\mathsf{heap}(\mathfrak{f}) = \{l_1 \mapsto b, b \mapsto l_2\} \text{ and}$$
$$\mathsf{project}^{\mathbf{Loc}}(\{l_1, l_2\}, \mathfrak{f}) = \forall a_1.\, \mathsf{even}(l_2, a_1) \mathrel{-\!\!\star} \mathsf{even}(l_1, a_1).$$

As all locations in this formula are in the image of the stack, we have

$$\begin{aligned}\mathsf{project}(\mathfrak{s}, \mathfrak{f}) &= \mathsf{project}^{\mathbf{Loc}}(\{l_1, l_2\}, \mathfrak{f})[\mathrm{dom}(\mathfrak{s}_{\max}^{-1}) / \mathrm{img}(\mathfrak{s}_{\max}^{-1})] \\ &= \forall a_1.\, \mathsf{even}(x_2, a_1) \mathrel{-\!\!\star} \mathsf{even}(x_1, a_1).\end{aligned}$$

Observe that

$$(\mathfrak{s}, \mathsf{heap}(\mathfrak{f})) \models_\Phi \mathsf{project}(\mathfrak{s}, \mathfrak{f}).$$

2. *Let $\mathfrak{s}$ and $\mathfrak{f}$ be the stack and Φ-forest from Example 9.3. Then*

$$\begin{aligned}\mathsf{project}(\mathfrak{s}, \mathfrak{f}) =& (\mathsf{treerp}(x, r) \mathrel{-\!\!\star} \mathsf{tree}(r)) \\ & \star (\mathsf{treerp}(y, r) \mathrel{-\!\!\star} \mathsf{treerp}(x, r)) \\ & \star (\mathbf{emp} \mathrel{-\!\!\star} \mathsf{treerp}(y, r)).\end{aligned}$$

3. *Let $\mathfrak{s} = \{x_1 \mapsto l_1, x_2 \mapsto l_2, x_3 \mapsto l_3\}$ and let $\mathfrak{f}$ be the Φ-forest from Example 10.4.*

$$\mathsf{heap}(\mathfrak{f}) = \{l_1 \mapsto \langle l_4, l_3\rangle, l_2 \mapsto l_4, l_3 \mapsto l_2, l_4 \mapsto l_2\},$$
$$\mathrm{img}(\mathfrak{s}) \cup (\mathsf{locs}(\mathsf{heap}(\mathfrak{f})) \setminus \mathrm{img}(\mathfrak{s})) = \{l_1, l_2, l_3, l_4\}$$

and

$$\begin{aligned}\mathsf{project}^{\mathbf{Loc}}(\{l_1, l_2, l_3, l_4\}, \mathfrak{f}) =& (\mathsf{p}_2(l_3, l_2, l_4) \mathrel{-\!\!\star} \mathsf{p}_1(l_1, l_2, l_3)) \\ & \star (\mathbf{emp} \mathrel{-\!\!\star} \mathsf{ptr}(l_2, l_4)) \\ & \star (\mathsf{ptr}(l_2, l_4) \mathrel{-\!\!\star} \mathsf{p}_2(l_3, l_2, l_4)).\end{aligned}$$

In Fig. 10.3a, we display the model $(\mathfrak{s}, \mathsf{heap}(\mathfrak{f}))$. The corresponding stack–forest projection is given by

$$\mathsf{project}^{\mathbf{Loc}}(\{l_1, l_2, l_3, l_4\}, \mathfrak{f})[\mathrm{dom}(\mathfrak{s}_{\max}^{-1}) \cdot l_4 / \mathrm{img}(\mathfrak{s}_{\max}^{-1}) \cdot e_1],$$

i.e.,

$$\begin{aligned}\mathsf{project}(\mathfrak{s}, \mathfrak{f}) = \exists e_1.(& \mathsf{p}_2(x_3, x_2, e_1) \mathrel{-\!\!\star} \mathsf{p}_1(x_1, x_2, x_3)) \\ & \star (\mathbf{emp} \mathrel{-\!\!\star} \mathsf{ptr}(x_2, e_1)) \\ & \star (\mathsf{ptr}(x_2, e_1) \mathrel{-\!\!\star} \mathsf{p}_2(x_3, x_2, e_1)).\end{aligned}$$

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

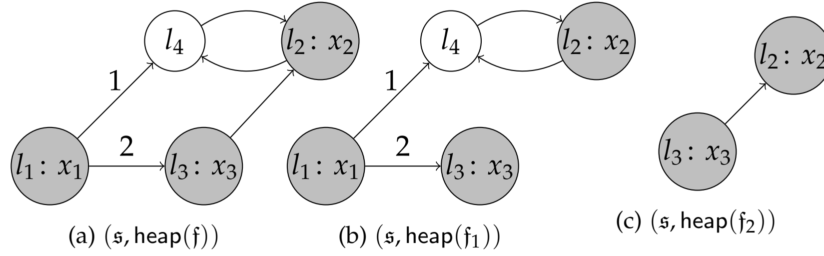


Figure 10.3: The models obtained via the projection of the Φ-forest $\mathfrak{f}$ defined in Examples 10.4 and 10.28 and the forests $\mathfrak{f}_1$ and $\mathfrak{f}_2$ as defined in Example 10.30.

Observe that $(\mathfrak{s}, \mathsf{heap}(\mathfrak{f})) \models_\Phi \mathsf{project}(\mathfrak{s}, \mathfrak{f})$. *Note also that the entailment*

$$\mathsf{project}(\mathfrak{s}, \mathfrak{f}) \models_\Phi \mathsf{p}_1(x_1, x_2, x_3)$$

is valid—in fact, $(\mathfrak{s}, \mathsf{heap}(\mathfrak{f}))$ *is the unique model (up to isomorphism) of the predicate* $\mathsf{p}_1(x_1, x_2, x_3)$.

In Example 10.28, we observed that $(\mathfrak{s}, \mathsf{heap}(\mathfrak{f})) \models_\Phi \mathsf{project}(\mathfrak{s}, \mathfrak{f})$. This is not an accident: the same property holds for all stacks and forests.

Lemma 10.29 (Soundness of stack–forest projection)**.** *Let* $\mathfrak{f}$ *be a* Φ*-forest and let* $\mathfrak{s}$ *be a stack. Then* $(\mathfrak{s}, \mathsf{heap}(\mathfrak{f})) \models_\Phi \mathsf{project}(\mathfrak{s}, \mathfrak{f})$.

Proof. Let $\mathbf{v} \triangleq \mathsf{locs}(\mathsf{heap}(\mathfrak{f})) \setminus \mathrm{img}(\mathfrak{s})$ and $\mathbf{e} \triangleq \langle e_1, e_2, \ldots, e_{|\mathbf{v}|} \rangle$. By Lemma 10.26, $(\mathfrak{s}, \mathsf{heap}(\mathfrak{f})) \models_\Phi \mathsf{project}^{\mathbf{Loc}}(\mathbf{w}, \mathfrak{f})$ for all $\mathbf{w}$ with $\mathrm{img}(\mathfrak{s}) \subseteq \mathbf{w} \subseteq \mathbf{Loc}$; in particular, this holds for $\mathbf{w} = \mathrm{img}(\mathfrak{s}) \cup \mathbf{v}$. The claim then follows by the implications in Fig. 10.4. In Fig. 10.4, we exploit that even if $\mathfrak{s}$ is *not* injective, $\mathfrak{s} \circ \mathfrak{s}_{\max}^{-1}$ is the identity function—unlike $\mathfrak{s}_{\max}^{-1} \circ \mathfrak{s}$, which is only the identity if $\mathfrak{s}$ is injective. □

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

TU Bibliothek, WIEN Your knowledge hub
Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.

$$
\begin{aligned}
& (\mathfrak{s}, \mathsf{heap}(\mathfrak{f})) \models_\Phi \mathsf{project}^{\mathbf{Loc}}(\mathrm{img}(\mathfrak{s}) \cup \mathbf{v}, \mathfrak{f}) \\
\Longrightarrow\ & \mathsf{heap}(\mathfrak{f}) \models_\Phi \mathsf{project}^{\mathbf{Loc}}(\mathrm{img}(\mathfrak{s}) \cup \mathbf{v}, \mathfrak{f})[\mathrm{dom}(\mathfrak{s}) \,/\, \mathrm{img}(\mathfrak{s})] && \text{(semantics)} \\
\Longrightarrow\ & \mathsf{heap}(\mathfrak{f}) \models_\Phi \mathsf{project}^{\mathbf{Loc}}(\mathrm{img}(\mathfrak{s}) \cup \mathbf{v}, \mathfrak{f}) && (\mathrm{dom}(\mathfrak{s}) \cap \mathsf{fvars}(\mathsf{project}^{\mathbf{Loc}}(\mathrm{img}(\mathfrak{s}) \cup \mathbf{v}, \mathfrak{f})) = \emptyset) \\
\Longrightarrow\ & \mathsf{heap}(\mathfrak{f}) \models_\Phi \mathsf{project}^{\mathbf{Loc}}(\mathrm{img}(\mathfrak{s}) \cup \mathbf{v}, \mathfrak{f})[\mathrm{dom}(\mathfrak{s}_{\max}^{-1}) \,/\, \mathrm{img}(\mathfrak{s}_{\max}^{-1})][\mathrm{dom}(\mathfrak{s}) \,/\, \mathrm{img}(\mathfrak{s})] && (\mathfrak{s} \circ \mathfrak{s}_{\max}^{-1} \text{ is identity}) \\
\Longrightarrow\ & (\mathfrak{s}, \mathsf{heap}(\mathfrak{f})) \models_\Phi (\mathsf{project}^{\mathbf{Loc}}(\mathrm{img}(\mathfrak{s}) \cup \mathbf{v}, \mathfrak{f}))[\mathrm{dom}(\mathfrak{s}_{\max}^{-1}) \,/\, \mathrm{img}(\mathfrak{s}_{\max}^{-1})] && \text{(stack–heap semantics)} \\
\Longrightarrow\ & (\mathfrak{s}, \mathsf{heap}(\mathfrak{f})) \models_\Phi (\mathsf{project}^{\mathbf{Loc}}(\mathrm{img}(\mathfrak{s}) \cup \mathbf{v}, \mathfrak{f}))[\mathbf{v} / \mathbf{e}][\mathbf{e} / \mathbf{v}][\mathrm{dom}(\mathfrak{s}_{\max}^{-1}) \,/\, \mathrm{img}(\mathfrak{s}_{\max}^{-1})] && (\mathbf{e} \text{ is fresh, so } \phi[\mathbf{v}/\mathbf{e}][\mathbf{e}/\mathbf{v}] \text{ is the identity}) \\
\Longrightarrow\ & (\mathfrak{s}, \mathsf{heap}(\mathfrak{f})) \models_\Phi \Big(\exists \mathbf{e}.\, (\mathsf{project}^{\mathbf{Loc}}(\mathrm{img}(\mathfrak{s}) \cup \mathbf{v}, \mathfrak{f}))[\mathbf{v} / \mathbf{e}] \Big) [\mathrm{dom}(\mathfrak{s}_{\max}^{-1}) \,/\, \mathrm{img}(\mathfrak{s}_{\max}^{-1})] \\
& && \text{(semantics of } \exists\text{, all locations in } \mathbf{v} \text{ allocated by Definition 10.27)} \\
\Longrightarrow\ & (\mathfrak{s}, \mathsf{heap}(\mathfrak{f})) \models_\Phi \exists \mathbf{e}.\, (\mathsf{project}^{\mathbf{Loc}}(\mathrm{img}(\mathfrak{s}) \cup \mathbf{v}, \mathfrak{f}))[\mathrm{dom}(\mathfrak{s}_{\max}^{-1}) \cdot \mathbf{v} \,/\, \mathrm{img}(\mathfrak{s}_{\max}^{-1}) \cdot \mathbf{e}] && (\mathbf{v} \cap \mathrm{dom}(\mathfrak{s}^{-1}) = \emptyset,\ \mathbf{e} \cap \mathrm{dom}(\mathfrak{s}^{-1}) = \emptyset) \\
\Longrightarrow\ & (\mathfrak{s}, \mathsf{heap}(\mathfrak{f})) \models_\Phi \mathsf{project}(\mathfrak{s}, \mathfrak{f}) && \text{(Definition 10.27)}
\end{aligned}
$$

Figure 10.4: Soundness of stack–forest projection.

10.3 COMPOSING PROJECTIONS

10.3.1 *Motivation*

Recall that we defined a composition operation on forests, $\mathfrak{f}_1 \bullet_{\mathbf{F}} \mathfrak{f}_2$, in Definition 10.15. Since our abstraction of models will consist of projections of forests, not of the forests themselves, we need to adapt the operation from forests to projections. This operation should derive from $\mathsf{project}(\mathfrak{s}, \mathfrak{f}_1)$ and $\mathsf{project}(\mathfrak{s}, \mathfrak{f}_2)$ all and only the projections of forests $\mathfrak{f} \in \mathfrak{f}_1 \bullet_{\mathbf{F}} \mathfrak{f}_2$, i.e., forests with $\mathfrak{f}_1 \uplus \mathfrak{f}_2 \blacktriangleright^* \mathfrak{f}$. In other words, we are looking for an operation $\bullet_{\mathbf{P}}$ that satisfies the following identity.

$$\mathsf{project}(\mathfrak{s}, \mathfrak{f}_1) \bullet_{\mathbf{P}} \mathsf{project}(\mathfrak{s}, \mathfrak{f}_2) \stackrel{?}{=} \{\mathsf{project}(\mathfrak{s}, \mathfrak{f}) \mid \mathfrak{f} \in \mathfrak{f}_1 \bullet_{\mathbf{F}} \mathfrak{f}_2\}. \qquad (\dagger)$$

Put differently, we are looking for an operation $\bullet_{\mathbf{P}}$ such that $\mathsf{project}(\mathfrak{s}, \cdot)$ is a homomorphism from the set of Φ-forests and $\bullet_{\mathbf{F}}$ to the set of projections and $\bullet_{\mathbf{P}}$.

How can we define such an operation $\bullet_{\mathbf{P}}$? Intuitively, we need to conjoin the projections via $\star$ (to simulate the operation $\mathfrak{f}_1 \uplus \mathfrak{f}_2$) and subsequently simulate the merge operation $\blacktriangleright$ at the level of projections. Recall that $\blacktriangleright$ identifies a hole of a tree $\mathfrak{t}_1$ with the root of another tree $\mathfrak{t}_2$. This is only possible if the hole and root are labeled with the same predicate call, say $\mathsf{pred}_2(\mathbf{l_2})$. To simulate this merge operation on projections, we define a derivation operation on projections, $\triangleright$, that rewrites formulas based on the variant of modus ponens, Lemma 10.21: by applying Lemma 10.21, we can merge the hole of the (projection of) tree $\mathfrak{t}_1$ with the root of the (projection of) tree $\mathfrak{t}_2$.

There is a further complication, however: the projections contain quantifiers. In particular, $\mathsf{project}(\mathfrak{s}, \mathfrak{f}_1) \star \mathsf{project}(\mathfrak{s}, \mathfrak{f}_2)$ is of the form $(\boxminus\mathbf{e_1}.\, \psi_1) \star (\boxminus\mathbf{e_2}.\, \psi_2)$, whereas $\mathsf{project}(\mathfrak{s}, \mathfrak{f}_1 \uplus \mathfrak{f}_2)$ is of the form $\boxminus\mathbf{e}.\, \psi$, where ψ_1, ψ_2, ψ do not contain guarded existentials. In other words, $\bullet_{\mathbf{P}}$ has to push the guarded existentials out before applying the merge operation.

To sum up, to define $\bullet_{\mathbf{P}}$, we need two ingredients: (1) a variant of the separating conjunction, which we will denote $\bar{\star}$, that captures all sound ways to move the existential quantifiers to the front of the formula $\mathsf{project}(\mathfrak{s}, \mathfrak{f}_1) \star \mathsf{project}(\mathfrak{s}, \mathfrak{f}_2)$ (i.e., "re-scopes the existentials") and (2) a derivation operation on projections, $\triangleright$, that rewrites formulas based on generalized modus ponens, $\stackrel{\mathsf{mp}}{\Longrightarrow}$ (Lemma 10.21).

Example 10.30 (Composing projections). *Recall the SID Φ and the trees $\mathfrak{t}_1, \mathfrak{t}_2, \mathfrak{t}_3$ from Example 10.4.*

Let

$$\mathfrak{s} = \{x_1 \mapsto l_1, x_2 \mapsto l_2, x_3 \mapsto l_3\},$$
$$\mathfrak{h}_1 \triangleq \{l_1 \mapsto \langle l_4, l_3\rangle, l_2 \mapsto l_4, l_4 \mapsto l_2\}, \textit{ and}$$
$$\mathfrak{h}_2 \triangleq \{l_3 \mapsto l_2\}.$$

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

The models $(\mathfrak{s}, \mathfrak{h}_1)$ *and* $(\mathfrak{s}, \mathfrak{h}_2)$ *are displayed in Figs. 10.3b and 10.3c on p. 144. Let* $\mathfrak{f}_1 \triangleq \{\mathfrak{t}_1, \mathfrak{t}_2\}$ *and* $\mathfrak{f}_2 \triangleq \{\mathfrak{t}_3\}$*. Note that* $\mathfrak{h}_i = \mathsf{heap}(\mathfrak{f}_i)$*. Let*

$$\begin{aligned}\mathfrak{t}_0 = \{ l_1 &\mapsto \langle\, \langle l_4, l_3\rangle\,, \mathsf{p}_1(l_1, l_2, l_3) \\ &\qquad\qquad \Leftarrow (l_1 \mapsto \langle l_4, l_3\rangle) \star \mathsf{ptr}(l_4, l_2) \star \mathsf{p}_2(l_3, l_2, l_4)\rangle, \\ l_2 &\mapsto \langle \emptyset, \mathsf{ptr}(l_2, l_4) \Leftarrow l_2 \mapsto l_4\rangle\,, \\ l_3 &\mapsto \langle l_2, \mathsf{p}_2(l_3, l_2, l_4) \Leftarrow (l_3 \mapsto l_2) \star \mathsf{ptr}(l_2, l_4)\rangle\,, \\ l_4 &\mapsto \langle \emptyset, \mathsf{ptr}(l_4, l_2) \Leftarrow l_4 \mapsto l_2\rangle\}\end{aligned}$$

and define $\mathfrak{f}_0 \triangleq \{\mathfrak{t}_0\}$*. Note that* $\mathfrak{f}_1 \uplus \mathfrak{f}_2 \blacktriangleright^* \mathfrak{f}_0$*. We would thus expect for*

$$\begin{aligned}\mathsf{project}(\mathfrak{s}, \mathfrak{f}_0) &= \mathbf{emp} \mathrel{-\!\!\star} \mathsf{p}_1(x_1, x_2, x_3), \\ \mathsf{project}(\mathfrak{s}, \mathfrak{f}_1) &= \boxminus e_1.\, (\mathsf{p}_2(x_3, x_2, e_1) \mathrel{-\!\!\star} \mathsf{p}_1(x_1, x_2, x_3)) \\ &\qquad\qquad \star (\mathbf{emp} \mathrel{-\!\!\star} \mathsf{ptr}(x_2, e_1)), \\ \mathsf{project}(\mathfrak{s}, \mathfrak{f}_2) &= \mathbb{V} a_1.\, \mathsf{ptr}(x_2, a_1) \mathrel{-\!\!\star} \mathsf{p}_2(x_3, x_2, a_1)\end{aligned}$$

that

$$\mathsf{project}(\mathfrak{s}, \mathfrak{f}_0) \in \mathsf{project}(\mathfrak{s}, \mathfrak{f}_1) \bullet_{\mathbf{P}} \mathsf{project}(\mathfrak{s}, \mathfrak{f}_2).$$

To this end, $\bullet_{\mathbf{P}}$ *must execute the following two steps.*

RE-SCOPING. *We push the existential out using the operation* $\bar{\star}$*, in the process instantiating the universal* a_1 *with* e_1*:*

$$\begin{aligned}\phi &\triangleq \boxminus e_1.\, (\mathsf{p}_2(x_3, x_2, e_1) \mathrel{-\!\!\star} \mathsf{p}_1(x_1, x_2, x_3)) \\ &\qquad \star (\mathbf{emp} \mathrel{-\!\!\star} \mathsf{ptr}(x_2, e_1)) \\ &\qquad \star (\mathsf{ptr}(x_2, e_1) \mathrel{-\!\!\star} \mathsf{p}_2(x_3, x_2, e_1)) \\ &\in (\boxminus e_1.\, (\mathsf{p}_2(x_3, x_2, e_1) \mathrel{-\!\!\star} \mathsf{p}_1(x_1, x_2, x_3)) \\ &\qquad \star (\mathbf{emp} \mathrel{-\!\!\star} \mathsf{ptr}(x_2, e_1))) \\ &\quad \bar{\star}\ (\mathbb{V} a_1.\, \mathsf{ptr}(x_2, a_1) \mathrel{-\!\!\star} \mathsf{p}_2(x_3, x_2, a_1)).\end{aligned}$$

MERGE STEPS. *We apply* $\overset{\mathsf{mp}}{\Longrightarrow}$ *(cf. Lemma 10.21) twice to subformulas of the re-scoped formula* ϕ*, each step denoted by* $\rhd$*:*

$$\begin{aligned}\phi &\rhd \boxminus e_1.\, (\mathsf{p}_2(x_3, x_2, e_1) \mathrel{-\!\!\star} \mathsf{p}_1(x_1, x_2, x_3)) \\ &\qquad \star (\mathbf{emp} \mathrel{-\!\!\star} \mathsf{p}_2(x_3, x_2, e_1)) \\ &\rhd \mathbf{emp} \mathrel{-\!\!\star} \mathsf{p}_1(x_1, x_2, x_3) = \mathsf{project}(\mathfrak{s}, \mathfrak{f}_0).\end{aligned}$$

Assume we define the operation $\bullet_{\mathbf{P}}$ *as rescoping* $\bar{\star}$ *followed by arbitrarily many merge steps:*

$$\begin{aligned}\phi_1 \bullet_{\mathbf{P}} \phi_2 \triangleq \{\phi \mid & \textit{there exist a } k \geq 1 \textit{ and } \zeta_1, \ldots, \zeta_k \\ & \textit{s.t. } \zeta_1 \textit{ is a re-scoping of } \phi_1 \textit{ and } \phi_2, \\ & \zeta_i \rhd \zeta_{i+1} \textit{ for all } 1 \leq i < k, \textit{ and } \zeta_k = \phi\}.\end{aligned}$$

Then we obtain

$$\mathsf{project}(\mathfrak{s}, \mathfrak{f}_0) \in \mathsf{project}(\mathfrak{s}, \mathfrak{f}_1) \bullet_{\mathbf{P}} \mathsf{project}(\mathfrak{s}, \mathfrak{f}_2),$$

as desired.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

In the remainder of this section, I formalize the operations $\bar{\star}$ and $\triangleright$, and then prove that the operation $\bullet_{\mathbf{P}}$ that combines these operations indeed corresponds to forest composition. This will "almost" yield the homomorphism (†) as proposed at the beginning of this section—there is one minor complication that will force us to change the statement slightly.

Sections 10.3.2 and 10.3.3 do not contain any deep insights. If you aren't interested in the formal details of $\bullet_{\mathbf{P}}$, you can safely skip the remainder of this section and continue with Chapter 11.

10.3.2 *Formal Definition of Projection Composition*

We need the following auxiliary notation. We write $\forall \mathbf{a}.\, \phi \xrightarrow{\exists \mathbf{e}/\forall \mathbf{u}} \forall \mathbf{b}.\, \phi'$ if there exist disjoint subsets $\bar{\mathbf{a_1}}, \bar{\mathbf{a_2}} \subseteq \mathbf{a}$ and subsets $\bar{\mathbf{u}} \subseteq \mathbf{u}$, $\bar{\mathbf{e}} \subseteq \mathbf{e}$ such that $\phi' = \phi[\bar{\mathbf{a_1}} \cdot \bar{\mathbf{a_2}} / \bar{\mathbf{u}} \cdot \bar{\mathbf{e}}]$ and $\mathbf{b} = \left(\mathbf{a} \setminus (\bar{\mathbf{a_1}} \cup \bar{\mathbf{a_2}})\right) \cup \bar{\mathbf{u}}$.

Example 10.31. *Let*

$$\phi \triangleq \forall \langle a_1, a_2 \rangle . (\mathsf{odd}(y, a_2) \mathrel{-\!\!\star} \mathsf{odd}(e_1, a_2)) \star (\mathsf{even}(e_1, a_1) \mathrel{-\!\!\star} \mathsf{odd}(x, a_1)).$$

Then

$$\phi \xrightarrow{\exists e_2/\forall u_1} \forall u_1.(\mathsf{odd}(y, e_2) \mathrel{-\!\!\star} \mathsf{odd}(e_1, e_2)) \star (\mathsf{even}(e_1, u_1) \mathrel{-\!\!\star} \mathsf{odd}(x, u_1)) \text{ but also}$$

$$\phi \xrightarrow{\exists e_2/\forall u_1} \forall u_1.(\mathsf{odd}(y, u_1) \mathrel{-\!\!\star} \mathsf{odd}(e_1, u_1)) \star (\mathsf{even}(e_1, e_2) \mathrel{-\!\!\star} \mathsf{odd}(x, e_2)), \text{ and}$$

$$\phi \xrightarrow{\exists e_2/\forall u_1} \forall a_1.(\mathsf{odd}(y, e_2) \mathrel{-\!\!\star} \mathsf{odd}(e_1, e_2)) \star (\mathsf{even}(e_1, a_1) \mathrel{-\!\!\star} \mathsf{odd}(x, a_1)) \text{ etc.}$$

We formalize the two ingredients of $\bullet_{\mathbf{P}}$, the notions of *re-scoping* and *derivability*.

Definition 10.32 (Re-scoping)**.** *Let $n \in \mathbb{N}$ and $\phi_i = \exists \mathbf{e_i}.\, \bigstar_{1 \le j \le n} \psi_{i,j}$ for $1 \le i \le 2$ such that $\mathbf{e_1} \cap \mathbf{e_2} = \emptyset$. We say that ζ is a* re-scoping *of ϕ_1 and ϕ_2, written $\zeta \in (\phi_1 \bar{\star} \phi_2)$, if there exist formulas $\psi'_{1,1}, \ldots, \psi'_{2,n_2}$ such that (1) $\psi_{i,j} \xrightarrow{\exists \mathbf{e_{3-i}}/\forall \varepsilon} \psi'_{i,j}$ for all i, j; and (2) $\zeta = \exists \mathbf{e_1} \cdot \mathbf{e_2}.\, \bigstar_{1 \le j \le n_1} \psi'_{1,j} \star \bigstar_{1 \le j \le n_2} \psi'_{2,j}$.*

Re-scoping is sound for projections that correspond to guarded models.

Lemma 10.33 (Soundness of re-scoping)**.** *Let $\mathfrak{s}$ be a stack and let $\mathfrak{f}_1, \mathfrak{f}_2$ be Φ-forests with $(\mathfrak{s}, \mathsf{heap}(\mathfrak{f}_1)), (\mathfrak{s}, \mathsf{heap}(\mathfrak{f}_2)) \in \mathbf{Models}^{\mathfrak{g}}_{\Phi}$. Define $\phi_i \triangleq \mathsf{project}(\mathfrak{s}, \mathfrak{f}_i)$. Finally, assume $\phi_3 \in (\phi_1 \bar{\star} \phi_2)$. Then $\phi_1 \star \phi_2 \models_{\Phi} \phi_3$.*

Proof. By Lemma 10.17, it is sound to move the guarded existential quantifiers to the front.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

It remains to be shown that instantiation of guarded universals with guarded existentials as in Definition 10.32 is sound.

Specifically, re-scoping instantiates the universals in ϕ_1 only with existentials from ϕ_2 and vice-versa. We show the soundness of the former; the argument for the latter is completely analogous.

Let $(\mathfrak{s}, \mathfrak{h}') \models_\Phi (\exists\mathbf{e_1}.\, \psi_1) \star (\exists\mathbf{e_2}.\, \psi_2)$. Then there exist models $\mathfrak{h}'_1, \mathfrak{h}'_2$ such that $(\mathfrak{s}, \mathfrak{h}'_i) \models_\Phi \exists\mathbf{e_i}.\, \psi_i$ and $\mathfrak{h}' = \mathfrak{h}'_1 + \mathfrak{h}'_2$.

We have to show that the interpretation of the existentials $\mathbf{e_2}$ does not overlap with $\mathsf{locs}(\mathfrak{h}_1)$, because guarded universals may only be instantiated with locations that don't occur in the model.

Let $e \in \mathbf{e_2}$. By the semantics of $\exists$, there then exists a location $l \in \mathrm{dom}(\mathfrak{h}'_2) \setminus \mathrm{img}(\mathfrak{s})$ such that $(\mathfrak{s}, \mathfrak{h}'_2) \models_\Phi \psi_2[e/l]$. Because $\mathfrak{h}'_1 + \mathfrak{h}'_2 \neq \bot$, it follows that $l \notin \mathrm{dom}(\mathfrak{h}'_1)$. Because $(\mathfrak{s}, \mathfrak{h}'_1) \in \mathbf{Models}^{\mathfrak{g}}_\Phi$ and $l \notin \mathrm{img}(\mathfrak{s})$, we have by Lemma 8.12 that $l \notin \mathsf{dangling}(\mathfrak{h}'_1)$. Consequently, $l \notin \mathsf{locs}(\mathfrak{h}'_1)$. It is thus sound w.r.t. the semantics of guarded universals to instantiate a universal quantifier a by e in ψ_1. □

Definition 10.34 (Derivability). *We say that ψ is* derivable *from $\zeta = \exists\mathbf{e}.\, \bigstar_{1 \le i \le n} \forall\mathbf{a_i}.\, \zeta_i$, written $\zeta \rhd \psi$, iff there exist indices m_1 and m_2, variable sequences $\mathbf{u_1}, \mathbf{u_2}, \mathbf{b_1}, \mathbf{b_2}, \mathbf{b}$, and formulas ϕ_1, ϕ_2, ϕ such that (1) $\forall\mathbf{a_{m_i}}.\, \zeta_{m_i} \xrightarrow{\exists\varepsilon/\forall\mathbf{u_i}} \forall\mathbf{b}_i.\, \phi_i$ for $1 \le i \le 2$, (2) $\phi_1 \star \phi_2 \overset{\mathsf{mp}}{\Longrightarrow} \phi$, (3) $\mathbf{b} = (\mathbf{b_1} \cup \mathbf{b_2}) \cap \mathsf{fvars}(\phi)$ and (4) ψ is obtained from ζ by removing the subformulas $\forall\mathbf{a_{m_i}}.\, \zeta_{m_i}$ and adding the subformula $\forall\mathbf{b}.\, \phi$.*

We write $\rhd^*$ for the reflexive–transitive closure of $\rhd$.

Lemma 10.35 (Soundness of $\rhd$). *Let ϕ_1, ϕ_2 be such that $\phi_1 = \mathsf{project}(\mathfrak{s}, \mathfrak{f})$ for some choice of $\mathfrak{s}$, $\mathfrak{f}$ and such that $\phi_1 \rhd \phi_2$. Then $\phi_1 \models_\Phi \phi_2$.*

Proof sketch. Because of the soundness of the modus-ponens variant $\overset{\mathsf{mp}}{\Longrightarrow}$ used in $\rhd$, it suffices to show that the instantiation

$$\forall\mathbf{a_{m_i}}.\, \zeta_{m_i} \xrightarrow{\exists\varepsilon/\forall\mathbf{u_i}} \forall\mathbf{b_i}.\, \phi_i, \tag{‡}$$

carried out to obtain matching predicate calls prior to applying $\overset{\mathsf{mp}}{\Longrightarrow}$ is sound. Since we only rename guarded universals to other guarded universals, this follows immediately from the injectivity of the instantiation carried out by (‡): every variable among $\mathbf{u_i}$ instantiates at most one of the variables $\mathbf{a_{m_i}}$, ensuring that we do not "merge" distinct variables from $\mathbf{a_{m_i}}$ by renaming them to the same variable in $\mathbf{b_i}$. This is necessary for soundness, because distinct universals have to be interpreted by distinct locations according to the semantics of guarded universals. □

As explained in Section 10.3.1, we integrate re-scoping, $\bar{\star}$, and derivability, $\rhd^*$, into the composition operation $\bullet_{\mathbf{P}}$.

Definition 10.36 (Projection composition). *Let $\phi_1 = \mathsf{project}(\mathfrak{s}, \mathfrak{f}_1), \phi_2 = \mathsf{project}(\mathfrak{s}, \mathfrak{f}_2)$ for some choice of $\mathfrak{s}$, $\mathfrak{f}_1$, $\mathfrak{f}_2$. Then*

$$\phi_1 \bullet_{\mathbf{P}} \phi_2 \triangleq \{\phi \mid \text{there ex. } \zeta \text{ s.t. } \zeta \in (\phi_1 \bar{\star} \phi_2) \text{ and } \zeta \rhd^* \phi\}.$$

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Corollary 10.37 (Soundness of $\bullet_{\mathbf{P}}$)**.** *Let $\mathfrak{s}$ be a stack and let $\mathfrak{f}_1, \mathfrak{f}_2$ be Φ-forests. If $\mathfrak{f}_1 \uplus \mathfrak{f}_2 \neq \bot$ and $(\mathfrak{s}, \mathsf{heap}(\mathfrak{f}_1)), (\mathfrak{s}, \mathsf{heap}(\mathfrak{f}_2)) \in \mathbf{Models}^{\mathfrak{g}}_{\Phi}$ then $\mathsf{project}(\mathfrak{s}, \mathfrak{f}_1) \star \mathsf{project}(\mathfrak{s}, \mathfrak{f}_2) \models_{\Phi} \psi$ for all $\psi \in \phi_1 \bullet_{\mathbf{P}} \phi_2$.*

Proof. Thanks to the transitivity of logical implication, this follows immediately from Lemmas 10.33 and 10.35. □

Example 10.38.
- *For $\phi_1 = \mathsf{ls}(x_2, x_3) \mathrel{-\!\!\star} \mathsf{ls}(x_1, x_3)$ and $\phi_2 = \mathbf{emp} \mathrel{-\!\!\star} \mathsf{ls}(x_2, x_3)$, it holds that $\phi_1 \star \phi_2 \rhd \mathbf{emp} \mathrel{-\!\!\star} \mathsf{ls}(x_1, x_3)$. Hence, $(\mathbf{emp} \mathrel{-\!\!\star} \mathsf{ls}(x_1, x_3)) \in \phi_1 \bullet_{\mathbf{P}} \phi_2$.*
- *Let*
$$\begin{aligned} \phi_1 &= \forall a.\, \mathsf{ls}(x_2, a) \mathrel{-\!\!\star} \mathsf{ls}(x_1, a), \\ \phi_2 &= \forall b.\, \mathsf{ls}(x_3, b) \mathrel{-\!\!\star} \mathsf{ls}(x_2, b), \\ \phi_1' &= \forall c.\, \mathsf{ls}(x_2, c) \mathrel{-\!\!\star} \mathsf{ls}(x_1, c) \text{ and} \\ \phi_2' &= \forall c.\, \mathsf{ls}(x_3, c) \mathrel{-\!\!\star} \mathsf{ls}(x_2, c). \end{aligned}$$
Observe that $\phi_i \xrightarrow{\exists\varepsilon/\forall c} \phi_i'$ and that $(\mathsf{ls}(x_2, c) \mathrel{-\!\!\star} \mathsf{ls}(x_1, c)) \star (\mathsf{ls}(x_3, c) \mathrel{-\!\!\star} \mathsf{ls}(x_2, c)) \rhd \mathsf{ls}(x_3, c) \mathrel{-\!\!\star} \mathsf{ls}(x_1, c)$. Hence, $(\forall c.\, \mathsf{ls}(x_3, c) \mathrel{-\!\!\star} \mathsf{ls}(x_1, c)) \in \phi_1 \bullet_{\mathbf{P}} \phi_2$.
- *Let*
$$\begin{aligned} \phi_1 &= \exists e_1.\, (\mathsf{treerp}(y, e_1) \mathrel{-\!\!\star} \mathsf{treerp}(x, e_1)) \\ &\qquad \star (\mathbf{emp} \mathrel{-\!\!\star} \mathsf{treerp}(y, e_1)) \text{ and} \\ \phi_2 &= \mathbf{emp}. \end{aligned}$$
Then
$$\psi = \exists e_1.\, (\mathsf{treerp}(y, e_1) \mathrel{-\!\!\star} \mathsf{treerp}(x, e_1)) \star (\mathbf{emp} \mathrel{-\!\!\star} \mathsf{treerp}(y, e_1))$$
is a re-scoping of ϕ_1 and ϕ_2 and $(\mathsf{treerp}(y, e_1) \mathrel{-\!\!\star} \mathsf{treerp}(x, e_1)) \star (\mathbf{emp} \mathrel{-\!\!\star} \mathsf{treerp}(y, e_1)) \rhd \mathbf{emp} \mathrel{-\!\!\star} \mathsf{treerp}(x, e_1)$, so $(\exists e_1.\, \mathbf{emp} \mathrel{-\!\!\star} \mathsf{treerp}(x, e_1)) \in \phi_1 \bullet_{\mathbf{P}} \phi_2$.
- *The semi-formal example in Section 10.3.1, Example 10.30, also works for $\bullet_{\mathbf{P}}$ as formalized in this section.*

10.3.3 *Relating Forest Composition and Projection Composition*

At the beginning of this section, we stated a design goal for projection composition: The projection function $\mathsf{project}(\mathfrak{s}, \cdot)$ should be a homomorphism from forests and $\bullet_{\mathbf{F}}$ to projections and $\bullet_{\mathbf{P}}$, i.e.,

$$\mathsf{project}(\mathfrak{s}, \mathfrak{f}_1) \bullet_{\mathbf{P}} \mathsf{project}(\mathfrak{s}, \mathfrak{f}_2) \stackrel{?}{=} \{\mathsf{project}(\mathfrak{s}, \mathfrak{f}) \mid \mathfrak{f} \in \mathfrak{f}_1 \bullet_{\mathbf{F}} \mathfrak{f}_2\}.$$

Is this the case for the operation $\bullet_{\mathbf{P}}$ as defined in Definition 10.36? Not quite.

Example 10.39 (Projection is not homomorphic)**.** *Let*

$$\begin{aligned} \mathfrak{t}_1 &\triangleq \{l_1 \mapsto \langle \emptyset, (\mathsf{odd}(l_1, m_1) \Leftarrow (l_1 \mapsto l_2) \star \mathsf{even}(l_2, m_1)) \rangle\} \\ \mathfrak{t}_2 &\triangleq \{l_2 \mapsto \langle \emptyset, (\mathsf{even}(l_2, m_2) \Leftarrow (l_2 \mapsto l_3) \star \mathsf{odd}(l_3, m_2)) \rangle\} \\ \mathfrak{f}_1 &\triangleq \{\mathfrak{t}_1\}, \mathfrak{f}_2 \triangleq \{\mathfrak{t}_2\} \\ \mathfrak{s} &\triangleq \{x_1 \mapsto l_1, x_2 \mapsto l_2, x_3 \mapsto l_3\} \end{aligned}$$

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Then

$$\begin{aligned}
&\mathsf{project}(\mathfrak{s},\mathfrak{f}_1) = \forall a_1.\,\mathsf{even}(x_2,a_1) \multimap \mathsf{odd}(x_1,a_1),\\
&\mathsf{project}(\mathfrak{s},\mathfrak{f}_2) = \forall a_1.\,\mathsf{odd}(x_3,a_1) \multimap \mathsf{even}(x_2,a_1),\ \textit{and}\\
&\forall a_1.\,\mathsf{odd}(x_3,a_1) \multimap \mathsf{odd}(x_1,a_1) \in \mathsf{project}(\mathfrak{s},\mathfrak{f}_1) \bullet_{\mathbf{P}} \mathsf{project}(\mathfrak{s},\mathfrak{f}_2).
\end{aligned}$$

Because different locations m_1, m_2 are unused in the two forests, there is only one forest in $\mathfrak{f}_1 \bullet_{\mathbf{F}} \mathfrak{f}_2$: the forest $\{\mathfrak{t}_1, \mathfrak{t}_2\}$. It is not possible to merge the trees using $\blacktriangleright$, because the hole predicate of the first tree, $\mathsf{even}(l_2, m_1)$, is different from the root of the second tree, $\mathsf{even}(l_2, m_2)$. In particular, there does not *exist a forest $\mathfrak{f}$ with $\mathfrak{f} \in \mathfrak{f}_1 \bullet_{\mathbf{F}} \mathfrak{f}_2$ and $\mathsf{project}(\mathfrak{s},\mathfrak{f}) = \forall a_1.\,\mathsf{odd}(x_3,a_1) \multimap \mathsf{odd}(x_1,a_1)$.*

The essence of Example 10.39 is that while $\bullet_{\mathbf{P}}$ allows renaming universals, $\bullet_{\mathbf{F}}$ does not allow renaming unused locations, breaking the homomorphism. To get a correspondence between the two notions of composition, we therefore allow renaming the location terms in forests that do *not* occur in any points-to assertion. We capture this in the notion of *α-equivalence*.

In the following, we write

$$(\mathsf{pred}(\mathbf{l}) \Leftarrow \phi)[\mathbf{v_1}/\mathbf{v_2}] \triangleq \mathsf{pred}(\mathbf{l}[\mathbf{v_1}/\mathbf{v_2}]) \Leftarrow \phi[\mathbf{v_1}/\mathbf{v_2}].$$

Definition 10.40 (α-equivalence)**.** *Two Φ-trees $\mathfrak{t}_1, \mathfrak{t}_2$ are* α-equivalent, *denoted $\mathfrak{t}_1 \equiv_\alpha \mathfrak{t}_2$, iff there exist sequences $\mathbf{v_1}, \mathbf{v_2} \in (\mathbf{Loc} \setminus \mathsf{locs}(\mathsf{heap}(\mathfrak{t}_1)))^*$ such that*

$$\mathfrak{t}_2 = \{l \mapsto \langle \mathsf{succ}_{\mathfrak{t}_1}(l), \mathsf{rule}_{\mathfrak{t}_1}(l)[\mathbf{v_1}/\mathbf{v_2}] \mid l \in \mathrm{dom}(\mathfrak{t}_1) \rangle\}.$$

Two Φ-forests $\mathfrak{f}_1 = \{\mathfrak{t}_1, \dots, \mathfrak{t}_k\}, \mathfrak{f}_2 = \{\bar{\mathfrak{t}}_1, \dots \bar{\mathfrak{t}}_k\}$ are α-equivalent, *also denoted $\mathfrak{f}_1 \equiv_\alpha \mathfrak{f}_2$, iff $\mathfrak{t}_i \equiv_\alpha \bar{\mathfrak{t}}_i$ for all $1 \leq i \leq k$.*

Intuitively, $\mathfrak{f}_1 \equiv_\alpha \mathfrak{f}_2$ if it is possible to rename the location terms in $\mathfrak{f}_1$ that do *not* occur in any points-to assertions in such a way that we obtain $\mathfrak{f}_2$. Consequently, α-equivalent forests induce the same heap.

Lemma 10.41. *If $\mathfrak{f}_1 \equiv_\alpha \mathfrak{f}_2$ then $\mathsf{heap}(\mathfrak{f}_1) = \mathsf{heap}(\mathfrak{f}_2)$.*

Proof. The forest $\mathfrak{f}_2$ differs from $\mathfrak{f}_1$ only in locations that are not in $\mathsf{heap}(\mathfrak{f}_1)$. □

Our final goal for this chapter is to prove

$$\begin{aligned}
&\mathsf{project}(\mathfrak{s},\mathfrak{f}_1) \bullet_{\mathbf{P}} \mathsf{project}(\mathfrak{s},\mathfrak{f}_2)\\
&= \{\mathsf{project}(\mathfrak{s},\mathfrak{f}) \mid \text{ex. } \bar{\mathfrak{f}}_1, \bar{\mathfrak{f}}_2 \text{ s.t. } \mathfrak{f}_1 \equiv_\alpha \bar{\mathfrak{f}}_1, \mathfrak{f}_2 \equiv_\alpha \bar{\mathfrak{f}}_2 \text{ and } \mathfrak{f} \in \bar{\mathfrak{f}}_1 \bullet_{\mathbf{F}} \bar{\mathfrak{f}}_2\}.
\end{aligned}$$

FROM COMPOSITION OF FORESTS TO COMPOSITION OF PROJECTIONS. Every $\blacktriangleright^*$ derivation on forests of guarded models induces a $\triangleright^*$ derivation on forest projections. We first show this for single-step derivations $\blacktriangleright$ and $\triangleright$.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Lemma 10.42. *Let $\mathfrak{f}_1, \mathfrak{f}_2$ be Φ-forests with $(\mathfrak{s}, \mathsf{heap}(\mathfrak{f}_1)), (\mathfrak{s}, \mathsf{heap}(\mathfrak{f}_2)) \in$* $\mathbf{Models}^{g}_{\Phi}$. *If $\mathfrak{f}_1 \uplus \mathfrak{f}_2 \blacktriangleright \mathfrak{f}$ then $\mathsf{project}(\mathfrak{s}, \mathfrak{f}) \in \mathsf{project}(\mathfrak{s}, \mathfrak{f}_1) \bullet_{\mathbf{P}} \mathsf{project}(\mathfrak{s}, \mathfrak{f}_2)$.*

Proof sketch. Let $\mathfrak{f}' \triangleq \mathfrak{f}_1 \uplus \mathfrak{f}_2$. Since $\mathfrak{f}' \blacktriangleright \mathfrak{f}$, there exists a location $l \in \mathrm{dom}(\mathfrak{f})$ with $\mathfrak{f}' = \mathsf{split}(\mathfrak{f}, \{l\})$. In particular, there then exist trees $\mathfrak{t}_1, \mathfrak{t}_2 \in \mathfrak{f}'$ and $\mathfrak{t} \in \mathfrak{f}$ such that

1. $\mathrm{dom}(\mathfrak{t}_1) \cup \mathrm{dom}(\mathfrak{t}_2) = \mathrm{dom}(\mathfrak{t})$,
2. $\mathsf{rule}_{\{\mathfrak{t}_1,\mathfrak{t}_2\}}(l') = \mathsf{rule}_{\{\mathfrak{t}\}}(l')$ for all l',
3. $l = \mathsf{root}(\mathfrak{t}_1)$, $l \in \mathsf{allholes}(\mathfrak{t}_2)$ and $(\mathsf{allholes}(\mathfrak{t}_1) \cup \mathsf{allholes}(\mathfrak{t}_2)) \setminus \{l\} = \mathsf{allholes}(\mathfrak{t})$.

This implies that the projections of $\mathfrak{t}_1$ and $\mathfrak{t}_2$ are mergeable into the projection of $\mathfrak{t}$ via generalized modus ponens (Lemma 10.21). Let us examine this claim in more detail. By definition of stack–forest projection, we have for $1 \leq i \leq 2$ and appropriate choices of $\mathbf{w_i}$, $\mathbf{e_i}$ that

$$\begin{aligned}\mathsf{project}(\mathfrak{s}, \mathfrak{f}_i) =& \exists \mathbf{e_i}.\, \zeta[\mathrm{dom}(\mathfrak{s}^{-1}_{\max}) \cdot \mathbf{w_i} / \mathrm{img}(\mathfrak{s}^{-1}_{\max}) \cdot \mathbf{e_i}], \text{ for} \\ \zeta =& (\mathbb{V} a_{i,1}.\, \psi_{i,1}[\mathbf{v_{i,1}} / \mathbf{a_{i,1}}]) \\ & \star \cdots \star (\mathbb{V} a_{i,n_1}.\, \psi_{i,n_1}[\mathbf{v_{i,n_1}} / \mathbf{a_{i,n_1}}]).\end{aligned}$$

We construct a formula $\psi \in (\mathsf{project}(\mathfrak{s}, \mathfrak{f}_1) \bar{\star} \mathsf{project}(\mathfrak{s}, \mathfrak{f}_2))$ as follows:

1. We push $\mathbf{e_1}$ and $\mathbf{e_2}$ out.
2. For all $1 \leq j \leq n_1$ and every location $v \in \mathbf{v_{1,j}}$, if $v \in \mathbf{w_2}$—that is, if v is a location that is replaced by an existential quantifier in $\mathsf{project}(\mathfrak{s}, \mathfrak{f}_2)$ and thus replaced by a universal quantifier in $\mathsf{project}(\mathfrak{s}, \mathfrak{f}_1)$—then we instantiate the universal $v[\mathbf{v_{1,j}} / \mathbf{a_{1,j}}]$ with the existential $v[\mathbf{w_2} / \mathbf{e_2}]$ in $\psi_{1,j}$.
3. Symmetrically, for all $1 \leq j \leq n_2$ and every location $v \in \mathbf{v_{2,j}}$, if $v \in \mathbf{w_1}$ then we instantiate $v[\mathbf{v_{2,j}} / \mathbf{a_{2,j}}]$ with $v[\mathbf{w_1} / \mathbf{e_1}]$ in $\psi_{2,j}$.

Observe that in the resulting formula ψ, it holds for every location $v \in \mathsf{locs}(\mathsf{heap}(\mathfrak{f}_1 \uplus \mathfrak{f}_2))$ that this location corresponds to a fixed variable in ψ, namely one of the variables in $\mathrm{dom}(\mathfrak{s}) \cup \mathbf{e_1} \cup \mathbf{e_2}$. In particular, this means that the thus-renamed projections of $\mathfrak{t}_1$ and $\mathfrak{t}_2$ can be merged via $\overset{\mathsf{mp}}{\Longrightarrow}$, implying $\psi \rhd \mathsf{project}(\mathfrak{s}, \mathfrak{f})$. Hence, $\mathsf{project}(\mathfrak{s}, \mathfrak{f}) \in \mathsf{project}(\mathfrak{s}, \mathfrak{f}_1) \bullet_{\mathbf{P}} \mathsf{project}(\mathfrak{s}, \mathfrak{f}_2)$. □

A straightforward inductive argument allows us to lift Lemma 10.42 from $\blacktriangleright$ to $\blacktriangleright^*$ and thus to $\bullet_{\mathbf{F}}$.

Corollary 10.43. *Let $\mathfrak{f}_1, \mathfrak{f}_2$ be Φ-forests with $(\mathfrak{s}, \mathsf{heap}(\mathfrak{f}_1)), (\mathfrak{s}, \mathsf{heap}(\mathfrak{f}_2)) \in$* $\mathbf{Models}^{g}_{\Phi}$. *If $\mathfrak{f} \in \mathfrak{f}_1 \bullet_{\mathbf{F}} \mathfrak{f}_2$ then $\mathsf{project}(\mathfrak{s}, \mathfrak{f}) \in \mathsf{project}(\mathfrak{s}, \mathfrak{f}_1) \bullet_{\mathbf{P}} \mathsf{project}(\mathfrak{s}, \mathfrak{f}_2)$.*

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

FROM COMPOSITION OF PROJECTIONS TO FOREST COMPOSITION. Up to α-equivalence of forests $\mathfrak{f}_1, \mathfrak{f}_2$, every element in $\mathsf{project}(\mathfrak{s}, \mathfrak{f}_1) \bullet_{\mathbf{P}} \mathsf{project}(\mathfrak{s}, \mathfrak{f}_2)$ corresponds to an element in the composition $\mathfrak{f}_1 \bullet_{\mathbf{F}} \mathfrak{f}_2$.

At a high level, merging two subformulas in $\rhd$ corresponds exactly to merging two trees in $\mathfrak{f}_1$, which then yields the desired forest $\mathfrak{f}_2$. It might, however, be necessary to rename some locations that occur in predicate calls in $\mathfrak{f}_1$ that end up universally quantified so as to obtain trees to which we can apply $\blacktriangleright$; hence the need to apply $\blacktriangleright$ to an α-equivalent forest $\mathfrak{f}_1'$, not directly to $\mathfrak{f}_1$, in the following lemma.

Lemma 10.44. *Let $\mathfrak{f}_1$ be a Φ-forest with $\mathsf{heap}(\mathfrak{f}_1) \in \mathbf{Models}^{\mathfrak{g}}_{\Phi}$ and let ϕ be such that $\mathsf{project}(\mathfrak{s}, \mathfrak{f}_1) \rhd \phi$. Then there exist forests $\mathfrak{f}_1', \mathfrak{f}_2$ with $\mathfrak{f}_1 \equiv_\alpha \mathfrak{f}_1'$, $\mathfrak{f}_1' \blacktriangleright \mathfrak{f}_2$ and $\mathsf{project}(\mathfrak{s}, \mathfrak{f}_2) = \phi$.*

Proof. Assume $\mathsf{project}(\mathfrak{s}, \mathfrak{f}_1) = \exists\mathbf{e_1}.\, \psi_{1,1} \star \cdots \star \psi_{1,n_1}$ and $\phi = \exists\mathbf{e_2}.\, \psi_{2,1} \star \cdots \star \psi_{2,n_2}$. Because $\mathsf{project}(\mathfrak{s}, \mathfrak{f}_1) \rhd \phi$, there exist indices m_1, m_2, m_3, sequences of variables $\mathbf{u_1}, \mathbf{u_2}$, and formulas $\psi'_{1,m_1}, \psi'_{1,m_2}$ such that

1. $\psi_{1,m_i} \xrightarrow{\exists\varepsilon/\forall\mathbf{u_i}} \psi'_{1,m_i}$ for $1 \leq i \leq 2$,
2. $\psi'_{1,m_1} \star \psi'_{1,m_1} \overset{\mathsf{mp}}{\Longrightarrow} \psi_{2,m_2}$, and
3. $\phi \equiv \bigstar((\{\psi_{1,1}, \ldots, \psi_{1,n_1}\} \setminus \{\psi_{1,m_1}, \psi_{1,m_2}\}) \cup \psi_{2,m_2})$.

Let $\mathfrak{t}_1, \mathfrak{t}_2 \in \mathfrak{f}_1$ and $\mathbf{v}, \mathbf{e_1}$ be such that

$$\psi_{1,m_i} = \mathsf{project}^{\mathbf{Loc}}(\mathrm{img}(\mathfrak{s}) \cup \mathbf{v}, \mathfrak{t}_i)[\mathbf{v}/\mathbf{e_1}],$$

$1 \leq i \leq 2$. There then exist variable sequences $\mathbf{a_1}, \mathbf{a_2}$ and location sequences $\mathbf{v_1}, \mathbf{v_2}$ such that

$$\psi_{1,m_i} = \forall\mathbf{a_i}.\, (\bigstar\mathsf{allholepreds}(\mathfrak{t}_i)) \mathrel{-\!\!\star} \mathsf{rootpred}(\mathfrak{t}_i)[\mathbf{v} \cdot \mathbf{v_i}/\mathbf{e_1} \cdot \mathbf{a_i}]$$

We assume w.l.o.g. that in the derivation $\mathsf{project}(\mathfrak{s}, \mathfrak{f}_1) \rhd \phi$, the predicate call $\mathsf{rootpred}(\mathfrak{t}_1)$ is merged with a call $\mathsf{pred}(\mathbf{l}) \in \mathsf{allholepreds}(\mathfrak{t}_2)$. (If this is not the case, simply swap $\mathfrak{t}_1$ and $\mathfrak{t}_2$.)

Because $\psi_{1,m_i} \xrightarrow{\exists\varepsilon/\forall\mathbf{u_i}} \psi'_{1,m_i}$ for $1 \leq i \leq 2$ and $\psi'_{1,m_1} \star \psi'_{1,m_1} \Longrightarrow \psi_{2,m_2}$ by Lemma 10.21 there exist variable sequences $\mathbf{b_1} \subseteq \mathbf{a_1} \cup \mathbf{u_1}$ and $\mathbf{b_2} \subseteq \mathbf{a_2} \cup \mathbf{u_2}$ such that

$$\mathsf{rootpred}(\mathfrak{t}_1)[\mathbf{v} \cdot \mathbf{v_1}/\mathbf{e_1} \cdot \mathbf{a_1}][\mathbf{a_1}/\mathbf{b_1}] = \mathsf{pred}(\mathbf{l})[\mathbf{v} \cdot \mathbf{v_2}/\mathbf{e_1} \cdot \mathbf{a_2}][\mathbf{a_2}/\mathbf{b_2}].$$

A simplification step yields

$$\mathsf{rootpred}(\mathfrak{t}_1)[\mathbf{v} \cdot \mathbf{v_1}/\mathbf{e_1} \cdot \mathbf{b_1}] = \mathsf{pred}(\mathbf{l})[\mathbf{v} \cdot \mathbf{v_2}/\mathbf{e_1} \cdot \mathbf{b_2}].$$

In general, there is, however, no guarantee that $\mathsf{rootpred}(\mathfrak{t}_1) = \mathsf{pred}(\mathbf{l})$: the calls only need to match after variable introduction.

To ensure that the predicate calls in the trees match exactly, which is required to apply $\blacktriangleright$, we therefore rename the locations in $\mathbf{v_1}$ and

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

$\mathbf{v_2}$ to fresh locations $\mathbf{v}_1'$ and $\mathbf{v}_2'$ in such a way that for all $w_1 \in \mathbf{v_1}$ and $w_2 \in \mathbf{v_2}$, if $w_1[\mathbf{v} \cdot \mathbf{v_1} / \mathbf{e_1} \cdot \mathbf{b_1}] = w_2[\mathbf{v} \cdot \mathbf{v_2} / \mathbf{e_1} \cdot \mathbf{b_2}]$ then also $w_1[\mathbf{v_1}/\mathbf{v_1}'] = w_2[\mathbf{v_2}/\mathbf{v_2}']$.

We then rename the locations $\mathbf{v_1}$ in $\mathfrak{t}_1$ to $\mathbf{v_1}'$ and the locations $\mathbf{v_2}$ in $\mathfrak{t}_2$ to $\mathbf{v_2}'$, obtaining $\mathfrak{t}_1'$ and $\mathfrak{t}_2'$. We define $\mathfrak{f}_1' \triangleq (\mathfrak{f}_1 \setminus \{\mathfrak{t}_1, \mathfrak{t}_2\}) \cup \{\mathfrak{t}_1', \mathfrak{t}_2'\}$. Observe that $\mathfrak{f}_1 \equiv_\alpha \mathfrak{f}_1'$, because we have only renamed locations that were replaced by universal quantifiers, i.e., locations that are *not* in $\mathsf{locs}(\mathsf{heap}(\mathfrak{f}_1))$.

Now that the root call and hole call in the renamed trees match exactly, i.e., now that there exists a call $\mathsf{pred}(\mathbf{l}') \in \mathsf{allholepreds}(\mathfrak{t}_2')$ with $\mathsf{pred}(\mathbf{l}') = \mathsf{rootpred}(\mathfrak{t}_1')$, we can apply $\blacktriangleright$ to the forest with the renamed trees, $\mathfrak{f}_1'$. We call the result $\mathfrak{f}_2$. It is easy to verify that the projection of the merged trees is exactly ψ_{2,m_3}. Consequently, $\mathsf{project}(\mathfrak{s}, \mathfrak{f}_2) \equiv \phi$. $\square$

Lemma 10.45. *If* $\mathsf{project}(\mathfrak{s}, \mathfrak{f}_1) \rhd^* \phi$ *then there exist forests* $\mathfrak{f}_1', \mathfrak{f}_2$ *with (1)* $\mathfrak{f}_1 \equiv_\alpha \mathfrak{f}_1'$*, (2)* $\mathfrak{f}_1' \blacktriangleright^* \mathfrak{f}_2$*, and (3)* $\mathsf{project}(\mathfrak{s}, \mathfrak{f}_2) \equiv \phi$*.*

Proof. We proceed by induction on the number n of steps of the derivation $\mathsf{project}(\mathfrak{s}, \mathfrak{f}_1) \rhd^* \phi$. If $n = 0$, set $\mathfrak{f}_1' = \mathfrak{f}_2 = \mathfrak{f}_1$.

If $n > 0$, let ψ be such that $\mathsf{project}(\mathfrak{s}, \mathfrak{f}_1) \rhd^* \psi \rhd \phi$. By the induction hypothesis, there exist forests $\mathfrak{f}_1'', \mathfrak{f}_3$ with $\mathfrak{f}_1 \equiv_\alpha \mathfrak{f}_1''$, $\mathfrak{f}_1'' \blacktriangleright^* \mathfrak{f}_3$ and $\mathsf{project}(\mathfrak{s}, \mathfrak{f}_3) = \psi$. By Lemma 10.44, there exist forests $\mathfrak{f}_3', \mathfrak{f}_2$ with $\mathfrak{f}_3 \equiv_\alpha \mathfrak{f}_3'$, $\mathfrak{f}_3' \blacktriangleright \mathfrak{f}_2$ and $\mathsf{project}(\mathfrak{s}, \mathfrak{f}_2) = \phi$.

Let $\mathbf{v_1}, \mathbf{v_2}$ be the location sequences that witness $\mathfrak{f}_3 \equiv_\alpha \mathfrak{f}_3'$, i.e., such that renaming $\mathbf{v_1}$ to $\mathbf{v_2}$ in $\mathfrak{f}_3$ yields $\mathfrak{f}_3'$. Also rename $\mathbf{v_1}$ to $\mathbf{v_2}$ in $\mathfrak{f}_1''$ to obtain a forest $\mathfrak{f}_1'$ with $\mathfrak{f}_1'' \equiv_\alpha \mathfrak{f}_1'$. Observe that because $\mathfrak{f}_1'' \blacktriangleright^* \mathfrak{f}_3$, it holds that $\mathfrak{f}_1' \blacktriangleright^* \mathfrak{f}_3'$. Combining this derivation with $\mathfrak{f}_3' \blacktriangleright \mathfrak{f}_2$, we obtain $\mathfrak{f}_1' \blacktriangleright^* \mathfrak{f}_2$. As we have already shown that $\mathsf{project}(\mathfrak{s}, \mathfrak{f}_2) = \phi$, this proves the claim. $\square$

Lemma 10.46. *Let* $\mathfrak{f}_1, \mathfrak{f}_2$ *be* Φ*-forests with* $(\mathfrak{s}, \mathsf{heap}(\mathfrak{f}_1)), (\mathfrak{s}, \mathsf{heap}(\mathfrak{f}_2)) \in \mathbf{Models}_\Phi^{\mathfrak{g}}$ *and* $\mathfrak{f}_1 \uplus \mathfrak{f}_2 \neq \bot$*. If* $\phi \in \mathsf{project}(\mathfrak{s}, \mathfrak{f}_1) \bullet_{\mathbf{P}} \mathsf{project}(\mathfrak{s}, \mathfrak{f}_2)$ *then there exist forests* $\mathfrak{f}_1', \mathfrak{f}_2', \mathfrak{f}$ *with (1)* $\mathfrak{f}_1 \equiv_\alpha \mathfrak{f}_1'$*, (2)* $\mathfrak{f}_2 \equiv_\alpha \mathfrak{f}_2'$*, (3)* $\mathfrak{f} \in \mathfrak{f}_1' \bullet_{\mathbf{F}} \mathfrak{f}_2'$ *and (4)* $\phi = \mathsf{project}(\mathfrak{s}, \mathfrak{f})$*.*

Proof. Because $\phi \in \mathsf{project}(\mathfrak{s}, \mathfrak{f}_1) \bullet_{\mathbf{P}} \mathsf{project}(\mathfrak{s}, \mathfrak{f}_2)$, there exists a formula ψ with $\psi \in (\mathsf{project}(\mathfrak{s}, \mathfrak{f}_1) \bar{\ast} \mathsf{project}(\mathfrak{s}, \mathfrak{f}_2))$ and $\psi \rhd^* \phi$.

We first construct forests $\mathfrak{f}_1''$ and $\mathfrak{f}_2''$ that are α-equivalent to $\mathfrak{f}_1$ and $\mathfrak{f}_2$ in a way that corresponds to the instantiation steps that produced the re-scoping ψ: For example, let $\mathfrak{t} \in \mathfrak{f}_1$ and let $\mathbb{V}\mathbf{a}.\,\zeta$ be the formula corresponding to $\mathfrak{t}$ in $\mathsf{project}(\mathfrak{s}, \mathfrak{f}_1)$. Assume that in the re-scoping, the variable $a \in \mathbf{a}$ was instantiated with the variable $e \in \mathbf{e_2}$, where $\mathbf{e_2}$ are the existential quantifiers in $\mathsf{project}(\mathfrak{s}, \mathfrak{f}_2)$. Assume further that the variable a replaced the location v_1 in $\mathsf{project}(\mathfrak{s}, \mathfrak{f}_1)$ and that the variable e replaced the location v_2 in $\mathsf{project}(\mathfrak{s}, \mathfrak{f}_2)$.

Note that $v_2 \notin \mathsf{locs}(\mathsf{heap}(\mathfrak{f}_1))$: By the semantics of the guarded existential $\mathbb{E}$, $v_2 \in \mathrm{dom}(\mathsf{heap}(\mathfrak{f}_2))$, so $v_2 \notin \mathrm{dom}(\mathsf{heap}(\mathfrak{f}_1))$. Moreover, because $(\mathfrak{s}, \mathsf{heap}(\mathfrak{f}_1)) \in \mathbf{Models}_\Phi^{\mathfrak{g}}$ and $v_2 \notin \mathrm{img}(\mathfrak{s})$, we have by

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Lemma 8.12 that $v_2 \notin \mathsf{dangling}(\mathfrak{h}_1)$. Consequently, $v_2 \notin \mathsf{locs}(\mathsf{heap}(\mathfrak{f}_1))$. This implies that renaming v_1 to v_2 in $\mathfrak{t}$ yields a tree $\mathfrak{t}'$ with $\mathfrak{t} \equiv_\alpha \mathfrak{t}'$.

We repeat this process until we have renamed the appropriate location in $\mathfrak{f}_1 \uplus \mathfrak{f}_2$ for every instantiation in the re-scoping. This yields a forest $\mathfrak{f}_1'' \uplus \mathfrak{f}_2''$ with $\mathfrak{f}_1 \equiv_\alpha \mathfrak{f}_1''$, $\mathfrak{f}_2 \equiv_\alpha \mathfrak{f}_2''$, and $\mathsf{project}(\mathfrak{s}, \mathfrak{f}_1'' \uplus \mathfrak{f}_2'') = \psi$.

We apply Lemma 10.45 and obtain forests $\mathfrak{f}_1'$, $\mathfrak{f}_2'$, and $\mathfrak{f}$ with $\mathfrak{f}_1'' \uplus \mathfrak{f}_2'' \equiv_\alpha \mathfrak{f}_1' \uplus \mathfrak{f}_2'$ (and thus $\mathfrak{f}_1'' \equiv_\alpha \mathfrak{f}_1'$ and $\mathfrak{f}_2'' \equiv_\alpha \mathfrak{f}_2'$), $\mathfrak{f}_1' \uplus \mathfrak{f}_2' \blacktriangleright^* \mathfrak{f}$, and $\mathsf{project}(\mathfrak{s}, \mathfrak{f}) = \phi$. By definition, $\mathfrak{f} \in \mathfrak{f}_1' \bullet_{\mathbf{F}} \mathfrak{f}_2'$ and by transitivity of $\equiv_\alpha$, $\mathfrak{f}_1 \equiv_\alpha \mathfrak{f}_1'$ and $\mathfrak{f}_1 \equiv_\alpha \mathfrak{f}_2'$, proving the claim. □

THE CORRESPONDENCE BETWEEN $\bullet_{\mathbf{F}}$ AND $\bullet_{\mathbf{P}}$. By combining the previous results, we obtain a variation of the homomorphism stated as (†) at the beginning of this section.

Theorem 10.47. *Let $\mathfrak{f}_1, \mathfrak{f}_2$ be Φ-forests with $(\mathfrak{s}, \mathsf{heap}(\mathfrak{f}_1)), (\mathfrak{s}, \mathsf{heap}(\mathfrak{f}_2)) \in$ $\mathbf{Models}_\Phi^{\mathfrak{g}}$ and $\mathfrak{f}_1 \uplus \mathfrak{f}_2 \neq \bot$. Then*

$$\begin{aligned} &\mathsf{project}(\mathfrak{s}, \mathfrak{f}_1) \bullet_{\mathbf{P}} \mathsf{project}(\mathfrak{s}, \mathfrak{f}_2) \\ &= \{\mathsf{project}(\mathfrak{s}, \mathfrak{f}) \mid \textit{ex. } \bar{\mathfrak{f}}_1, \bar{\mathfrak{f}}_2 \textit{ s.t. } \mathfrak{f}_1 \equiv_\alpha \bar{\mathfrak{f}}_1, \mathfrak{f}_2 \equiv_\alpha \bar{\mathfrak{f}}_2 \textit{ and } \mathfrak{f} \in \bar{\mathfrak{f}}_1 \bullet_{\mathbf{F}} \bar{\mathfrak{f}}_2\}\,. \end{aligned}$$

Proof. Immediate from Corollary 10.43 and Lemma 10.46. □

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.

11 THE TYPE ABSTRACTION

In this chapter, we develop the Φ-*type abstraction*, which abstracts every (guarded) model by a set of stack–forest projections. In Chapter 12, we will develop a decision procedure for $\mathbf{SLID}^{\mathsf{g}}_{\mathsf{btw}}$ by systemically computing the Φ-types of all models of arbitrary $\mathbf{SLID}^{\mathsf{g}}_{\mathsf{btw}}$ formulas.

Recall that all stack–forest projections can be obtained by "partially unfolding" symbolic heaps (and adding appropriate quantifiers). Henceforth, we thus call such formulas *unfolded symbolic heaps* (USHs) w.r.t. Φ.

Unfolded symbolic heaps vs. inductive wands

Note that USHs are a generalization of the *inductive wands* proposed in [TNK19]: Inductive wands also correspond to partial unfolding of predicates, but Tatsuta et al. only consider quantifier-free formulas with inductive wands, whereas we allow a (guarded) exists–forall prefix.

Our idea for abstracting models is as follows: because the USHs that hold in a model $(\mathfrak{s}, \mathfrak{h})$ capture all the ways that $(\mathfrak{s}, \mathfrak{h})$ relates to the predicates of the SID, we would like to abstract every model by a subset of the USHs of the model $(\mathfrak{s}, \mathfrak{h})$. Let us formalize this.

Definition 11.1 (Forests of a heap). *The* forests *of a heap* $\mathfrak{h} \in \mathbf{Heaps}$ *are* $\mathsf{forests}_{\Phi}(\mathfrak{h}) \triangleq \{\mathfrak{f} \mid \mathsf{heap}(\mathfrak{f}) = \mathfrak{h}\}$, *i.e., the set of all Φ-forests whose induced heap is $\mathfrak{h}$.*

The abstraction should map $(\mathfrak{s}, \mathfrak{h})$ to a *subset* of the set of USHs

$$\{\mathsf{project}(\mathfrak{s}, \mathfrak{f}) \mid \mathfrak{f} \in \mathsf{forests}_{\Phi}(\mathfrak{h})\}\,.$$

However, we cannot use the entire set as abstraction: while this set is finite for every fixed model, the set of all USHs w.r.t. an SID Φ is infinite and thus not suitable for defining a finite abstraction of the set of all models.

Example 11.2. *Let Φ be an SID that defines the list-segment predicate* lseg. *Let* $(\mathfrak{s}, \mathfrak{h}) \models_{\Phi} \mathsf{lseg}(x, \mathsf{nil})$ *with* $|\mathfrak{h}| > n$. *Then there exists a forest* $\mathfrak{f}$ *with* $\mathsf{heap}(\mathfrak{f}) = \mathfrak{h}$ *and*

$$\begin{aligned}\mathsf{project}(\mathfrak{s}, \mathfrak{f}) = \exists y_1, \ldots, y_n . \mathsf{lseg}(y_n, \mathsf{nil}) \\ \star (\mathsf{lseg}(y_n, \mathsf{nil}) \mathrel{-\!\!\star} \mathsf{lseg}(y_{n-1}, \mathsf{nil})) \\ \star \cdots \star (\mathsf{lseg}(y_2, \mathsf{nil}) \mathrel{-\!\!\star} \mathsf{lseg}(y_1, \mathsf{nil})) \\ \star (\mathsf{lseg}(y_1, \mathsf{nil}) \mathrel{-\!\!\star} \mathsf{lseg}(x, \mathsf{nil}))\end{aligned}$$

As there exist such models $(\mathfrak{s}, \mathfrak{h})$ for arbitrary $n \in \mathbb{N}$, there are infinitely many USHs w.r.t. Φ.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

To obtain a finite abstraction, we only consider *delimited USHs* (DUSHs), in which (1) all root parameters of predicate calls are free variables and (2) every variable occurs at most once as a root parameter on the left-hand side of a magic wand. We formalize DUSHs in Section 11.1.

The Φ-type of a model $(\mathfrak{s}, \mathfrak{h})$, formally introduced in Section 11.2, then is the set of all DUSHs that can be obtained via stack–forest projection of the forests $\mathsf{forests}_\Phi(\mathfrak{h})$.

11.1 DELIMITED UNFOLDED SYMBOLIC HEAPS

11.1.1 *Interfaces*

To obtain a finite set of USHs, we will restrict the *interface* of the forests we consider.

Definition 11.3 (Interface). *The* interface *of* $\mathfrak{f} = \{\mathfrak{t}_1, \ldots, \mathfrak{t}_k\}$ *is given by*

$$\mathsf{interface}(\mathfrak{f}) \triangleq \bigcup_{1 \le i \le k} (\{\mathsf{root}(\mathfrak{t}_i)\} \cup \mathsf{allholes}(\mathfrak{t}_i)).$$

Example 11.4. *Recall the forest* $\mathfrak{f}$ *from Example 10.4. We have* $\mathsf{interface}(\mathfrak{f}) = \{l_1, l_2, l_3\}$*: the locations* l_1, l_2, l_3 *all occur as the roots of a tree;* l_2 *and* l_3 *additionally occur as holes (of* $\mathfrak{t}_3$ *and* $\mathfrak{t}_1$*, respectively); and* l_4 *occurs neither as root nor as hole of a tree.*

When we split a forest $\mathfrak{f}$ at locations $\mathbf{l}$, this adds the locations $\mathbf{l}$ to the interface of the forest—provided those locations actually occur in the forest $\mathfrak{f}$ to begin with.

Lemma 11.5. *Let* $\mathfrak{f}$ *be a forest and* $\mathbf{l} \subseteq \mathbf{Loc}$*. Then* $\mathsf{interface}(\mathsf{split}(\mathfrak{f}, \mathbf{l})) = \mathsf{interface}(\mathfrak{f}) \cup (\mathbf{l} \cap \mathrm{dom}(\mathfrak{f}))$.

Proof. In the following, let $\mathsf{locs}(\mathsf{graph}(\mathfrak{f}))$ denote all those locations that occur in the relation $\mathsf{graph}(\mathfrak{f})$.

$$
\begin{aligned}
&\mathsf{roots}(\mathsf{split}(\mathfrak{f}, \mathbf{l}))\\
=&\mathsf{roots}(\mathfrak{f}) \cup \{b \in \mathbf{l} \mid \exists a.\,(a,b) \in \mathsf{graph}(\mathfrak{f})\}\\
=&\mathsf{roots}(\mathfrak{f}) \cup \{b \in \mathbf{l} \cap \mathrm{dom}(\mathfrak{f}) \mid \exists a.\,(a,b) \in \mathsf{graph}(\mathfrak{f})\}\\
&\hfill (\mathsf{locs}(\mathsf{graph}(\mathfrak{f})) \subseteq \mathrm{dom}(\mathfrak{f}))\\
=&\{b \in \mathrm{dom}(\mathfrak{f}) \mid \forall a.\,(a,b) \notin \mathsf{graph}(\mathfrak{f})\}\\
&\quad \cup \{b \in \mathbf{l} \cap \mathrm{dom}(\mathfrak{f}) \mid \exists a.\,(a,b) \in \mathsf{graph}(\mathfrak{f})\}\\
&\hfill \text{(all and only roots have no predecessor)}\\
=&\{b \in \mathrm{dom}(\mathfrak{f}) \mid \forall a.\,(a,b) \notin \mathsf{graph}(\mathfrak{f})\}\\
&\quad \cup \{b \in \mathbf{l} \cap \mathrm{dom}(\mathfrak{f}) \mid \forall a.\,(a,b) \notin \mathsf{graph}(\mathfrak{f})\}\\
&\quad \cup \{b \in \mathbf{l} \cap \mathrm{dom}(\mathfrak{f}) \mid \exists a.\,(a,b) \in \mathsf{graph}(\mathfrak{f})\}\\
&\hfill \text{(second set subset of first set)}
\end{aligned}
$$

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

$$
\begin{aligned}
&= \mathsf{roots}(\mathfrak{f}) \cup \{b \in \mathbf{l} \cap \mathrm{dom}(\mathfrak{f}) \mid \forall a.\,(a,b) \notin \mathsf{graph}(\mathfrak{f})\} \\
&\quad \cup \{b \in \mathbf{l} \cap \mathrm{dom}(\mathfrak{f}) \mid \exists a.\,(a,b) \in \mathsf{graph}(\mathfrak{f})\} \\
&= \mathsf{roots}(\mathfrak{f}) \cup \{b \in \mathbf{l} \cap \mathrm{dom}(\mathfrak{f})\}
\end{aligned}
$$

Similarly,

$$
\begin{aligned}
\mathsf{allholes}(\mathsf{split}(\mathfrak{f},\mathbf{l})) &= \mathsf{allholes}(\mathfrak{f}) \cup \{b \in \mathbf{l} \mid \exists a.\,(a,b) \in \mathsf{graph}(\mathfrak{f})\} \\
&= \mathsf{allholes}(\mathfrak{f}) \cup \{b \in \mathbf{l} \cap \mathrm{dom}(\mathfrak{f})\}\,.
\end{aligned}
$$

By definition of interfaces, we thus obtain

$$
\begin{aligned}
\mathsf{interface}(\mathsf{split}(\mathfrak{f},\mathbf{l})) &= \mathsf{roots}(\mathsf{split}(\mathfrak{f},\mathbf{l})) \cup \mathsf{allholes}(\mathsf{split}(\mathfrak{f},\mathbf{l})) \\
&= \mathsf{roots}(\mathfrak{f}) \cup \{t \in \mathbf{l} \cap \mathrm{dom}(\mathfrak{f})\} \\
&\quad \cup \mathsf{allholes}(\mathfrak{f}) \cup \{b \in \mathbf{l} \cap \mathrm{dom}(\mathfrak{f})\} \\
&= \mathsf{interface}(\mathfrak{f}) \cup \{b \in \mathbf{l} \cap \mathrm{dom}(\mathfrak{f})\}\,. \qquad \square
\end{aligned}
$$

11.1.2 *Delimited Forests and Their Projections*

$\mathfrak{s}$-DELIMITED FORESTS. An $\mathfrak{s}$-*delimited forest* is a forest whose interface is contained in $\mathrm{img}(\mathfrak{s})$ and which, additionally, does not have any duplicate holes.

Definition 11.6. *A forest $\mathfrak{f}$ is $\mathfrak{s}$-*delimited *iff (1) $\mathsf{interface}(\mathfrak{f}) \subseteq \mathrm{img}(\mathfrak{s})$ and (2) for all $l \in \mathsf{allholes}(\mathfrak{f})$ there exist exactly one tree $\mathfrak{t} \in \mathfrak{f}$ and exactly one location l' such that $l \in \mathsf{holes}_{\mathfrak{t}}(l')$.*

That every hole occurs at most once in a delimited forest $\mathfrak{f}$ is a prerequisite for "removing" all holes of a delimited forest via forest composition and $\blacktriangleright^*$. In other words, only if there are *no* duplicate holes can there exist forests $\mathfrak{f}', \bar{\mathfrak{f}}$ such that $\mathfrak{f} \uplus \mathfrak{f}' \blacktriangleright^* \bar{\mathfrak{f}}$ and $\mathsf{allholes}(\bar{\mathfrak{f}}) = \emptyset$. Conversely, in forests with duplicate holes, eliminating all holes is impossible, because any attempt to do so would lead to double allocation.

Example 11.7. *Let Φ_{tree} be the SID that defines binary trees from Example 8.2. Let $\mathfrak{s} = x \mapsto l_1, y \mapsto l_2, z \mapsto l_3$.*

1. *Let $\mathfrak{t}_1 = \{l_1 \mapsto \langle \emptyset, \mathsf{tree}(l_1) \Leftarrow (l_1 \mapsto \langle l_2, l_3\rangle) \star \mathsf{tree}(l_2) \star \mathsf{tree}(l_3)\rangle\}$. Observe that $\mathsf{allholes}(\{\mathfrak{t}_1\}) = \{l_2, l_3\}$. Since both holes occur only once and $\mathsf{interface}(\mathfrak{f}) = \{l_1, l_2, l_3\} \subseteq \mathrm{img}(\mathfrak{s})$, $\{\mathfrak{t}_1\}$ is $\mathfrak{s}$-delimited. We can eliminate these holes as follows. We let*

$$
\begin{aligned}
\mathfrak{t}_2 &\triangleq \{l_2 \mapsto \langle \emptyset, \mathsf{tree}(l_2) \Leftarrow l_2 \mapsto \langle \mathsf{nil}, \mathsf{nil}\rangle\rangle\}, \\
\mathfrak{t}_3 &\triangleq \{l_3 \mapsto \langle \emptyset, \mathsf{tree}(l_3) \Leftarrow l_3 \mapsto \langle \mathsf{nil}, \mathsf{nil}\rangle\rangle\}, \\
\bar{\mathfrak{t}} &\triangleq \{l_1 \mapsto \langle \langle l_2, l_3\rangle, \mathfrak{t}_1(l_1)\rangle\} \cup \mathfrak{t}_2 \cup \mathfrak{t}_3.
\end{aligned}
$$

Then $\{\mathfrak{t}_1\} \cup \{\mathfrak{t}_2, \mathfrak{t}_3\} \blacktriangleright^* \{\bar{\mathfrak{t}}\}$ and $\mathsf{allholes}(\bar{\mathfrak{t}}) = \emptyset$, i.e., we have successfully eliminated all the holes of $\{\mathfrak{t}_1\}$.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

2. *Let*

$$\mathfrak{t}' = \{l_1 \mapsto \langle \{l_2\}, \mathsf{tree}(l_1) \Leftarrow (l_1 \mapsto \langle l_2, l_3 \rangle) \star \mathsf{tree}(l_2) \star \mathsf{tree}(l_3) \rangle, \\ l_2 \mapsto \langle \{l_4\}, \mathsf{tree}(l_1) \Leftarrow (l_2 \mapsto \langle l_3, l_4 \rangle) \star \mathsf{tree}(l_3) \star \mathsf{tree}(l_4) \rangle, \\ l_4 \mapsto \langle \emptyset, \mathsf{tree}(l_3) \Leftarrow (l_3 \mapsto \langle \mathsf{nil}, \mathsf{nil} \rangle) \rangle\}.$$

We have $\mathsf{allholes}(\{\mathfrak{t}'\}) = \{l_4\}$, *but this hole now appears twice. We can eliminate one occurrence of this hole. For example, there exists a tree* $\bar{\mathfrak{t}}'$ *such that with* $\mathfrak{t}_3$ *as in the previous example, we have that* $\{\mathfrak{t}'\} \cup \{\mathfrak{t}_3\} \blacktriangleright^* \{\bar{\mathfrak{t}}'\}$. *In the derivation step we can, however, only eliminate one occurrence of the hole, so* $\mathsf{allholes}(\bar{\mathfrak{t}}') = \{l_3\}$. *We cannot eliminate this remaining occurrence of the hole by merging with another forest, because* l_3 *would have to be the root of that forest and thus in the domain of that forest, but we already have* $l_3 \in \mathrm{dom}(\bar{\mathfrak{t}}')$.

If $\mathsf{heap}(\mathfrak{f}) = \mathfrak{h}_1 + \mathfrak{h}_2$ for $(\mathfrak{s}, \mathfrak{h}_1), (\mathfrak{s}, \mathfrak{h}_2) \in \mathbf{Models}^{\mathfrak{g}}_{\Phi}$, we can always find $\mathfrak{s}$-delimited forests $\mathfrak{f}_1, \mathfrak{f}_2$ with $\mathsf{heap}(\mathfrak{f}_i) = \mathfrak{h}_i$ and $\mathfrak{f}_1 \uplus \mathfrak{f}_2 \blacktriangleright^* \mathfrak{f}$. Specifically, we can always obtain $\mathfrak{f}_1$ and $\mathfrak{f}_2$ via the $\mathfrak{s}$-*decomposition of* $\mathfrak{f}$ via an $\mathrm{img}(\mathfrak{s})$-split.

Definition 11.8. *Let* $\mathfrak{f}$ *be an* $\mathfrak{s}$*-delimited forest. We call* $\mathsf{split}(\mathfrak{f}, \mathrm{img}(\mathfrak{s}))$ *the* $\mathfrak{s}$-decomposition *of* $\mathfrak{f}$. *We call* $\mathfrak{f}$ $\mathfrak{s}$-decomposed *iff* $\mathfrak{f} = \mathsf{split}(\mathfrak{f}, \mathrm{img}(\mathfrak{s}))$.

Lemma 11.9 (Decompositions are delimited)**.** *Let* $\bar{\mathfrak{f}}$ *be the* $\mathfrak{s}$*-decomposition of an* $\mathfrak{s}$*-delimited forest* $\mathfrak{f}$. *Then* $\bar{\mathfrak{f}}$ *is* $\mathfrak{s}$*-delimited.*

Proof. By definition, $\bar{\mathfrak{f}} = \mathsf{split}(\mathfrak{f}, \mathrm{img}(\mathfrak{s}))$. By Lemma 11.5, we then have $\mathsf{interface}(\bar{\mathfrak{f}}) \subseteq \mathsf{interface}(\mathfrak{f}) \cup \mathrm{img}(\mathfrak{s})$. Since $\mathfrak{f}$ is $\mathfrak{s}$-delimited, $\mathsf{interface}(\mathfrak{f}) \subseteq \mathrm{img}(\mathfrak{s})$. Overall, we thus obtain $\mathsf{interface}(\bar{\mathfrak{f}}) \subseteq \mathrm{img}(\mathfrak{s})$, i.e., $\bar{\mathfrak{f}}$ is $\mathfrak{s}$-delimited. □

Because the $\mathfrak{s}$-decomposition of a forest is obtained by splitting the trees of the forest at all locations in $\mathrm{img}(\mathfrak{s})$, only the roots of the trees in an $\mathfrak{s}$-decomposition of forest $\mathfrak{f}$ can be locations in $\mathrm{img}(\mathfrak{s})$.

Lemma 11.10. *Let* $\bar{\mathfrak{f}}$ *be the* $\mathfrak{s}$*-decomposition of an* $\mathfrak{s}$*-delimited forest* $\mathfrak{f}$ *and let* $\bar{\mathfrak{t}} \in \bar{\mathfrak{f}}$. *Then* $\mathrm{img}(\mathfrak{s}) \cap \mathrm{dom}(\bar{\mathfrak{t}}) = \{\mathsf{root}(\bar{\mathfrak{t}})\}$.

Proof. Since $\bar{\mathfrak{f}}$ is $\mathfrak{s}$-delimited by Lemma 11.9, we have $\{\mathsf{root}(\bar{\mathfrak{t}})\} \subseteq \mathrm{img}(\mathfrak{s})$. Since $\mathsf{root}(\bar{\mathfrak{t}}) \in \mathrm{dom}(\bar{\mathfrak{t}})$, $\{\mathsf{root}(\bar{\mathfrak{t}})\} \subseteq \mathrm{img}(\mathfrak{s}) \cap \mathrm{dom}(\bar{\mathfrak{t}})$.

Conversely, since $\bar{\mathfrak{f}} = \mathsf{split}(\mathfrak{f}, \mathrm{img}(\mathfrak{s}))$, we have $\mathsf{roots}(\bar{\mathfrak{f}}) = \mathsf{roots}(\mathfrak{f}) \cup (\mathrm{img}(\mathfrak{s}) \cap \mathrm{dom}(\mathfrak{f}))$, i.e., every location in $\mathrm{img}(\mathfrak{s}) \cap \mathrm{dom}(\bar{\mathfrak{f}})$ is a root of $\bar{\mathfrak{f}}$. Consequently, $\mathrm{img}(\mathfrak{s}) \cap \mathrm{dom}(\bar{\mathfrak{t}}) \subseteq \{\mathsf{root}(\bar{\mathfrak{t}})\}$. □

This implies in particular that the stack-allocated variables of $(\mathfrak{s}, \mathfrak{h})$ correspond precisely to the roots of the $\mathfrak{s}$-decomposed forests of $\mathfrak{h}$.

Lemma 11.11. *Let* $(\mathfrak{s}, \mathfrak{h})$ *be a model and* $\mathfrak{f} \in \mathsf{forests}_{\Phi}(\mathfrak{h})$ $\mathfrak{s}$*-delimited. Let* $\bar{\mathfrak{f}}$ *be the* $\mathfrak{s}$*-decomposition of* $\mathfrak{f}$. *Then* $\mathsf{alloced}(\mathfrak{s}, \mathfrak{h}) = \{x \mid \mathfrak{s}(x) \in \mathsf{roots}(\bar{\mathfrak{f}})\}$.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Proof. By Lemma 10.14, $\mathsf{heap}(\bar{\mathfrak{f}}) = \mathfrak{h}$ and thus, in particular, $\mathrm{dom}(\bar{\mathfrak{f}}) = \mathrm{dom}(\mathfrak{h})$. Consequently,

$$\mathfrak{s}(\mathsf{alloced}(\mathfrak{s}, \mathfrak{h})) = \mathrm{img}(\mathfrak{s}) \cap \mathrm{dom}(\bar{\mathfrak{f}}).$$

By Lemma 11.10, $\mathrm{img}(\mathfrak{s}) \cap \mathrm{dom}(\bar{\mathfrak{t}}) = \{\mathsf{root}(\bar{\mathfrak{t}})\}$ for all $\bar{\mathfrak{t}} \in \bar{\mathfrak{f}}$, so $\mathrm{img}(\mathfrak{s}) \cap \mathrm{dom}(\bar{\mathfrak{f}}) = \bigcup\{\{\mathsf{root}(\bar{\mathfrak{t}})\} \mid \bar{\mathfrak{t}} \in \bar{\mathfrak{f}}\} = \mathsf{roots}(\bar{\mathfrak{f}})$. Overall, we thus have $\mathfrak{s}(\mathsf{alloced}(\mathfrak{s}, \mathfrak{h})) = \mathsf{roots}(\bar{\mathfrak{f}})$. On both sides, we replace every location l with $\{x \mid \mathfrak{s}(x) = l\}$ and obtain $\mathsf{alloced}(\mathfrak{s}, \mathfrak{h}) = \{x \mid \mathfrak{s}(x) \in \mathsf{roots}(\bar{\mathfrak{f}})\}$. □

Lemma 11.12. *Let $\mathfrak{s}$ be a stack, let $\mathfrak{h}_1, \mathfrak{h}_2$ be heaps such that $(\mathfrak{s}, \mathfrak{h}_1), (\mathfrak{s}, \mathfrak{h}_2) \in$ $\mathbf{Models}^{\mathrm{g}}_{\Phi}$, and let $\bar{\mathfrak{f}}$ be the $\mathfrak{s}$-decomposition of $\mathfrak{s}$-delimited forest $\mathfrak{f}$ with $\mathsf{heap}(\mathfrak{f}) = \mathfrak{h}_1 + \mathfrak{h}_2$. Then there exist forests $\mathfrak{f}_1, \mathfrak{f}_2$ with $\mathfrak{f}_1 \uplus \mathfrak{f}_2 = \bar{\mathfrak{f}}$ and $\mathsf{heap}(\mathfrak{f}_i) = \mathfrak{h}_i$.*

Proof. We let $\mathfrak{f}_i \triangleq \{\bar{\mathfrak{t}} \in \bar{\mathfrak{f}} \mid \mathsf{root}(\bar{\mathfrak{t}}) \in \mathrm{dom}(\mathfrak{h}_i)\}$. Since $\mathfrak{f}_1 \uplus \mathfrak{f}_2 = \mathfrak{f}$ and thus $\mathsf{heap}(\mathfrak{f}_1) + \mathsf{heap}(\mathfrak{f}_2) = \mathsf{heap}(\mathfrak{f})$ by Lemma 10.7, it suffices to show that for every tree $\bar{\mathfrak{t}}$ in $\mathfrak{f}_i$, $\mathsf{heap}(\bar{\mathfrak{t}}) \subseteq \mathfrak{h}_i$

To this end, let $\bar{\mathfrak{t}} \in \mathfrak{f}_i$. Assume towards a contradiction that $\mathrm{dom}(\bar{\mathfrak{t}}) \cap \mathrm{dom}(\mathfrak{h}_{3-i}) \neq \emptyset$. Then there exist locations $l_1 \in \mathrm{dom}(\bar{\mathfrak{t}}) \cap \mathrm{dom}(\mathfrak{h}_i)$ and $l_2 \in \mathrm{dom}(\bar{\mathfrak{t}}) \cap \mathrm{dom}(\mathfrak{h}_{3-i})$ with $l_2 \in \mathsf{succ}_{\bar{\mathfrak{t}}}(l_1)$—otherwise, we would have $\mathsf{root}(\bar{\mathfrak{t}}) \in \mathrm{dom}(\mathfrak{h}_{3-i})$, but $\mathsf{root}(\bar{\mathfrak{t}}) \in \mathrm{dom}(\mathfrak{h}_i)$ by construction. In particular, $l_2 \in \mathrm{img}(\mathfrak{h}_i)$ and $l_2 \in \mathrm{dom}(\mathfrak{h}_{3-i})$, implying that $l_2 \in \mathsf{dangling}(\mathfrak{h}_i)$. Since $(\mathfrak{s}, \mathfrak{h}_1), (\mathfrak{s}, \mathfrak{h}_2) \in \mathbf{Models}^{\mathrm{g}}_{\Phi}$, we have by Lemma 8.12 that $l_2 \in \mathrm{img}(\mathfrak{s})$. Since $l_2 \neq \mathsf{root}(\bar{\mathfrak{t}})$, this contradicts Lemma 11.10. □

Corollary 11.13. *Let $(\mathfrak{s}, \mathfrak{h}_1), (\mathfrak{s}, \mathfrak{h}_2) \in \mathbf{Models}^{\mathrm{g}}_{\Phi}$ and $\mathfrak{h}_1 + \mathfrak{h}_2 \neq \bot$. Let $\mathfrak{f} \in \mathsf{forests}_{\Phi}(\mathfrak{h}_1 + \mathfrak{h}_2)$ be an $\mathfrak{s}$-delimited forest. Then there exist $\mathfrak{s}$-delimited forests $\mathfrak{f}_1, \mathfrak{f}_2$ with $\mathsf{heap}(\mathfrak{f}_i) = \mathfrak{h}_i$ and $\mathfrak{f} \in \mathfrak{f}_1 \bullet_{\mathbf{F}} \mathfrak{f}_2$.*

Proof. Let $\bar{\mathfrak{f}}$ be the $\mathfrak{s}$-decomposition of $\mathfrak{f}$. In particular, we then have $\bar{\mathfrak{f}} \blacktriangleright^* \mathfrak{f}$ by definition of $\blacktriangleright^*$. Let $\mathfrak{f}_1, \mathfrak{f}_2$ be such that $\mathfrak{f}_1 \uplus \mathfrak{f}_2 = \bar{\mathfrak{f}}$ and $\mathsf{heap}(\mathfrak{f}_i) = \mathfrak{h}_i$. Such forests exist by Lemma 11.12. We then have $\mathfrak{f}_1 \uplus \mathfrak{f}_2 = \bar{\mathfrak{f}} \blacktriangleright^* \mathfrak{f}$, i.e., $\mathfrak{f} \in \mathfrak{f}_1 \bullet_{\mathbf{F}} \mathfrak{f}_2$. Since $\bar{\mathfrak{f}}$ is $\mathfrak{s}$-delimited (by Lemma 11.9), so are $\mathfrak{f}_1$ and $\mathfrak{f}_2$. □

DELIMITED USHS. The projections of $\mathfrak{s}$-delimited forests give rise to a fragment of unfolded symbolic heaps: USHs where all root parameters of predicate calls are free variables; and every free variable occurs at most once as a root parameter on the left-hand side of a magic wand. Consequently, we call such projections delimited as well:

Definition 11.14. *An unfolded symbolic heap ϕ is* delimited *iff*

1. *for all $\mathsf{pred}(\mathbf{z}) \in \phi$, $\mathsf{predroot}(\mathsf{pred}(\mathbf{z})) \in \mathsf{fvars}(\phi)$, and*
2. *for all z there exists at most one predicate call $\mathsf{pred}(\mathbf{z}) \in \phi$ such that $z = \mathsf{predroot}(\mathsf{pred}(\mathbf{z}))$ and $\mathsf{pred}(\mathbf{z})$ occurs on the left-hand side of a magic wand.*

A forest is $\mathfrak{s}$-delimited precisely when its projection is delimited.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Lemma 11.15. *Let $\mathfrak{f}$ be a forest and let $\mathfrak{s}$ be a stack. Then $\mathfrak{f}$ is $\mathfrak{s}$-delimited iff* $\mathsf{project}(\mathfrak{s},\mathfrak{f})$ *is delimited.*

Proof. Recall that the projection contains predicate calls corresponding to the roots and holes of the forest. It thus holds for all forests that

$$\mathsf{interface}(\mathfrak{f}) = \{\mathsf{predroot}(\mathsf{pred}(\mathbf{z})) \mid \mathsf{pred}(\mathbf{z}) \in \mathsf{project}(\mathfrak{f})\}\,. \qquad (\dagger)$$

We show that if $\mathfrak{f}$ is $\mathfrak{s}$-delimited then $\mathsf{project}(\mathfrak{s},\mathfrak{f})$ is delimited. The proof of the other direction is completely analogous.

If $\mathfrak{f}$ is $\mathfrak{s}$-delimited then $\mathsf{interface}(\mathfrak{f}) \subseteq \mathrm{img}(\mathfrak{s})$ and thus, by $(\dagger)$,

$$\{\mathsf{predroot}(\mathsf{pred}(\mathbf{z})) \mid \mathsf{pred}(\mathbf{z}) \in \mathsf{project}(\mathfrak{f})\} \subseteq \mathrm{img}(\mathfrak{s}).$$

Trivially, the set of root locations in the projection is a subset of the set of *all* locations in the projection.

$$\{\mathsf{predroot}(\mathsf{pred}(\mathbf{z})) \mid \mathsf{pred}(\mathbf{z}) \in \mathsf{project}(\mathfrak{f})\} \subseteq \mathsf{locs}(\mathsf{project}(\mathfrak{f})).$$

Combining the above two observations, we conclude

$$\begin{aligned} &\{\mathsf{predroot}(\mathsf{pred}(\mathbf{z})) \mid \mathsf{pred}(\mathbf{z}) \in \mathsf{project}(\mathfrak{f})\} \\ \subseteq\ &\mathrm{img}(\mathfrak{s}) \cap \mathsf{locs}(\mathsf{project}(\mathfrak{f})). \end{aligned}$$

We apply $\mathfrak{s}_{\max}^{-1}$ on both sides to obtain that

$$\begin{aligned} &\mathfrak{s}_{\max}^{-1}(\{\mathsf{predroot}(\mathsf{pred}(\mathbf{z})) \mid \mathsf{pred}(\mathbf{z} \in \mathsf{project}(\mathfrak{f}))\}) \\ \subseteq\ &\mathrm{dom}(\mathfrak{s}) \cap \underbrace{(\mathsf{fvars}(\mathsf{project}(\mathfrak{s},\mathfrak{f}))}_{\{\mathfrak{s}_{\max}^{-1}(l) \mid l \in \mathrm{img}(\mathfrak{s}) \cap \mathsf{locs}(\mathsf{project}(\mathfrak{f}))\}} \subseteq \mathsf{fvars}(\mathsf{project}(\mathfrak{s},\mathfrak{f})). \end{aligned}$$

Moreover, since there are no duplicate holes in $\mathfrak{f}$, and the holes of $\mathfrak{f}$ are mapped to the predicate calls on the left-hand side of magic wands in $\mathsf{project}(\mathfrak{s},\mathfrak{f})$, no variable can occur twice as root parameter on the left-hand side of magic wands in $\mathsf{project}(\mathfrak{s},\mathfrak{f})$.

Consequently, $\mathsf{project}(\mathfrak{s},\mathfrak{f})$ is delimited. □

We collect the set of delimited unfolded symbolic heaps (DUSH) over SID Φ in

$$\begin{aligned} \mathbf{DUSH}_\Phi \triangleq \{\mathsf{project}(\mathfrak{s},\mathfrak{f}) \mid \mathfrak{s} \in \mathbf{Stacks}, \mathfrak{f}\ \Phi\text{-forest}, \\ \mathsf{project}(\mathfrak{s},\mathfrak{f}) \text{ is delimited}\} \end{aligned}$$

and denote by $\mathbf{DUSH}_\Phi^{\mathbf{x}}$ the restriction of $\mathbf{DUSH}_\Phi$ to formulas ϕ with $\mathsf{fvars}(\phi) \subseteq \mathbf{x}$.

11.1.3 *Finiteness of the DUSH Fragment*

As explained before, the number of all unfolded symbolic heaps is infinite. In contrast, the set $\mathbf{DUSH}_\Phi^{\mathbf{x}}$ is finite for every fixed SID Φ and every fixed finite set of free variables $\mathbf{x}$, because (1) every root variable in a DUSH is among $\mathbf{x}$ and (2) every variable can appear at most twice as root variable (once as the projection of a hole, once as the projection of the root of a tree).

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Lemma 11.16. *Let $\Phi \in \mathbf{ID}_{\mathsf{btw}}$ and let $\mathbf{x} \in 2^{\mathbf{Var}}$ be a finite set of variables. Let $n \triangleq |\Phi| + |\mathbf{x}|$. Then $|\mathbf{DUSH}_{\Phi}^{\mathbf{x}}| \in 2^{\mathcal{O}(n^2 \log(n))}$.*

Proof. We first note that every element of $\mathbf{DUSH}_{\Phi}^{\mathbf{x}}$ can be encoded as a string of length $\mathcal{O}(n)$ over the alphabet $Z \triangleq \mathbf{Preds}(\Phi) \cup \mathbf{x} \cup \{e_1, \dots, e_{n^2}\} \cup \{a_1, \dots, a_{n^2}\} \cup \{\mathbf{emp}, \star, -\!\!\star, (,)\}$ (†), where we assume that $\{e_1, \dots, e_{n^2}, a_1, \dots, a_{n^2}\} \cap \mathbf{x} = \emptyset$.

To this end, let $\phi \in \mathbf{DUSH}_{\Phi}^{\mathbf{x}}$ be a DUSH and let $\mathbf{e}$, ψ_i, pred_i etc. be such that

$$\begin{aligned} \phi &= \exists\mathbf{e}.\, \psi_1 \star \cdots \star \psi_m, \\ \psi_i &= \forall\mathbf{a}_i.\, \zeta_i -\!\!\star \mathsf{pred}_i(\mathbf{z_i}) \text{ for } 1 \leq i \leq m. \end{aligned}$$

Because ϕ is delimited, $\mathsf{predroot}(\mathsf{pred}_i(\mathbf{z_i})) \in \mathbf{x}$. As ϕ is the projection of a Φ-forest $\mathfrak{f}$, every variable $x \in \mathbf{x}$ can appear as root parameter for at most one choice of i—otherwise, the location corresponding to x would be in the domain of at least two trees in $\mathfrak{f}$, contradicting the assumption that $\mathfrak{f}$ is a Φ-forest. Consequently, $m \leq n$.

Furthermore, because $\mathsf{predroot}(\mathsf{pred}'(\mathbf{z}'))$ is a hole for all $\mathsf{pred}'(\mathbf{z}')$ that occur in a ζ_i sub-formula, and the forest is delimited, it follows for all $\mathsf{pred}'(\mathbf{z}')$ that occur in a ζ_i sub-formula that $\mathsf{predroot}(\mathsf{pred}'(\mathbf{z}')) \in \mathbf{x}$. Since no hole may occur more than once in a delimited USH, this implies that the *total* number of predicate calls across all ζ_i is also bounded by $|\mathbf{x}| \leq n$.

Overall, it is therefore guaranteed that ϕ contains at most $2n \in \mathcal{O}(n)$ predicate calls. Each predicate call takes at most $|\Phi| \leq n$ many parameters. This implies that the formula can contain at most $n^2 - |\mathbf{x}| \leq n^2$ different variables. We can thus assume w.l.o.g. that all existentially-quantified variables in ϕ are among the variables $e_1, \dots, e_{n^2}$ and all universally-quantified variables are among $a_1, \dots, a_{n^2}$. There then is no need to include the quantifiers explicitly in the string encoding. After dropping the quantifiers, we obtain a formula ϕ' that consists exclusively of letters from the alphabet Z. Moreover, this formula consists of at most $\mathcal{O}(n^2)$ letters. This concludes the proof of (†).

Now observe that $|Z| \in \mathcal{O}(n^2)$. Consequently, every letter of Z can be encoded by $\mathcal{O}(\log(n^2)) = \mathcal{O}(\log(n))$ bits. Therefore, every $\phi \in \mathbf{DUSH}_{\Phi}^{\mathbf{x}}$ can be encoded by a bit string of length $\mathcal{O}(n^2 \log(n))$. Since there are $2^{\mathcal{O}(n^2 \log(n))}$ such strings, the claim follows. □

11.2 DEFINING THE TYPE ABSTRACTION

We call the set of all DUSHs of a model $(\mathfrak{s}, \mathfrak{h})$ the Φ-*type* of the model.

Definition 11.17 (Φ-Type). *Let $(\mathfrak{s}, \mathfrak{h}) \in \mathbf{Models}_{\Phi}^{\mathfrak{g}}$ be a model and $\Phi \in \mathbf{ID}_{\mathsf{btw}}$. The Φ-type of $(\mathfrak{s}, \mathfrak{h})$ is*

$$\mathsf{type}_{\Phi}(\mathfrak{s}, \mathfrak{h}) \triangleq \{\mathsf{project}(\mathfrak{s}, \mathfrak{f}) \mid \mathfrak{f} \in \mathsf{forests}_{\Phi}(\mathfrak{h})\} \cap \mathbf{DUSH}_{\Phi}.$$

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Note that because we require models to be guarded in the definition of types, we only have to deal with nonempty types.

Lemma 11.18. *Let* $(\mathfrak{s},\mathfrak{h}) \in \mathbf{Models}^{\mathrm{g}}_{\Phi}$. *It holds that* $\mathsf{type}_{\Phi}(\mathfrak{s},\mathfrak{h}) \neq \emptyset$.

Proof. By definition, $(\mathfrak{s},\mathfrak{h}) \models_{\Phi} \bigstar_{1\leq i\leq k}\mathsf{pred}_i(\mathbf{x_i})$ for appropriate predicate calls. We split $\mathfrak{h}$ into $\mathfrak{h}_1,\ldots,\mathfrak{h}_k$ such that $\mathfrak{h} = \mathfrak{h}_1 + \cdots + \mathfrak{h}_k$ and $(\mathfrak{s},\mathfrak{h}_i) \models_{\Phi} \mathsf{pred}_i(\mathbf{x_i})$. For $1 \leq i \leq k$, let $\mathfrak{t}_i$ be a tree with $\mathsf{heap}(\mathfrak{t}_i) = \mathfrak{h}_i$; such trees exist by Lemma 10.5. Observe further that these trees are delimited, as they do not have any holes and since their root is in $\mathfrak{s}(\mathbf{x_i})$. Let $\mathfrak{f} \triangleq \{\mathfrak{t}_1,\ldots,\mathfrak{t}_k\}$. Lemma 10.7 yields that $\mathsf{heap}(\mathfrak{f}) = \mathfrak{h}$. We apply Lemma 10.29 to obtain that $(\mathfrak{s},\mathfrak{h}) \models_{\Phi} \mathsf{project}(\mathfrak{s},\mathfrak{f})$. Thus, $\mathsf{project}(\mathfrak{s},\mathfrak{f}) \in \mathsf{type}_{\Phi}(\mathfrak{s},\mathfrak{h})$ implying $\mathsf{type}_{\Phi}(\mathfrak{s},\mathfrak{h}) \neq \emptyset$. □

We need the set of all types over an SID Φ and the restriction of this set to stack $\mathfrak{s}$,

$$\mathbf{Types}(\Phi) \triangleq \{\mathsf{type}_{\Phi}(\mathfrak{s},\mathfrak{h}) \mid \mathfrak{s} \in \mathbf{Stacks}, \mathfrak{h} \in \mathbf{Heaps}\},$$
$$\mathbf{Types}^{\mathfrak{s}}(\Phi) \triangleq \{\mathsf{type}_{\Phi}(\mathfrak{s},\mathfrak{h}) \mid \mathfrak{h} \in \mathbf{Heaps}\} \subseteq \mathbf{Types}(\Phi).$$

We use Φ-types to define an abstraction of $\mathbf{SLID}^{\mathrm{g}}_{\mathsf{btw}}$ formulas. This is completely analogous to how we defined an abstraction of **SSL** formulas by sets of AMS in Definition 6.15 in Part ii.

Definition 11.19 ($\mathfrak{s}$-Types of a formula)**.** *Let* $\phi \in \mathbf{SLID}^{\mathrm{g}}_{\mathsf{btw}}$ *be a guarded formula. The* $\mathfrak{s}$-types *of* ϕ *are given by*

$$\mathbf{Types}^{\mathfrak{s}}_{\Phi}(\phi) \triangleq \{\mathsf{type}_{\Phi}(\mathfrak{s},\mathfrak{h}) \mid \mathfrak{h} \in \mathbf{Heaps}, (\mathfrak{s},\mathfrak{h}) \models_{\Phi} \phi\}.$$

Note that $\mathbf{Types}^{\mathfrak{s}}_{\Phi}(\phi) \subseteq \mathbf{Types}^{\mathfrak{s}}(\Phi)$. Finally, the $\mathbf{x}$-*types* of a formula are the union over all stacks $\mathfrak{s}$ with $\mathrm{dom}(\mathfrak{s}) = \mathbf{x}$ of the $\mathfrak{s}$-types of that formula.

Definition 11.20 ($\mathbf{x}$-Types of a Formula)**.** *Let* $\mathbf{x} \in 2^{\mathbf{Var}}$ *be finite. We define* $\mathbf{Types}^{\mathbf{x}}_{\Phi}(\phi) \triangleq \bigcup\{\mathbf{Types}^{\mathfrak{s}}_{\Phi}(\phi) \mid \mathrm{dom}(\mathfrak{s}) \subseteq \mathbf{x}\}$.

Note that it is always possible to infer from $\mathsf{type}_{\Phi}(\mathfrak{s},\mathfrak{h})$ the set of variables $\mathsf{alloced}(\mathfrak{s},\mathfrak{h})$.

Definition 11.21 (Allocated variables of a type)**.** *Let* $\mathcal{T}$ *be a* Φ*-type. We define the set of* allocated variables *of* $\mathcal{T}$ *as*

$$\mathsf{alloced}(\mathcal{T}) \triangleq \{x \mid \textit{there ex. } \phi \in \mathcal{T} \textit{ and } (\psi \mathrel{-\!\!\star} \mathsf{pred}(\mathbf{z})) \in \phi \;\; \textit{s.t. } x = \mathsf{predroot}(\mathsf{pred}(\mathbf{z}))\}.$$

Lemma 11.22. *For all models* $(\mathfrak{s},\mathfrak{h}) \in \mathbf{Models}^{\mathrm{g}}_{\Phi}$, *it holds that*

$$\mathsf{alloced}(\mathfrak{s},\mathfrak{h}) = \mathsf{alloced}(\mathsf{type}_{\Phi}(\mathfrak{s},\mathfrak{h})).$$

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Proof. By definition of DUSHs, all root parameters of all DUSHs in $\mathsf{alloced}(\mathsf{type}_\Phi(\mathfrak{s},\mathfrak{h}))$ are in $\mathrm{img}(\mathfrak{s})$. Consequently, $\mathsf{alloced}(\mathfrak{s},\mathfrak{h}) \supseteq \mathsf{alloced}(\mathsf{type}_\Phi(\mathfrak{s},\mathfrak{h}))$

For the other implication, let $\mathfrak{f}$ be a forest with $\mathsf{heap}(\mathfrak{f}) = \mathfrak{h}$ and $\mathsf{project}(\mathfrak{s},\mathfrak{f}) \in \mathsf{type}_\Phi(\mathfrak{s},\mathfrak{h})$. Such a forest must exist, as $\mathsf{type}_\Phi(\mathfrak{s},\mathfrak{h}) \neq \emptyset$ by Lemma 11.18. Let $\bar{\mathfrak{f}}$ be the $\mathfrak{s}$-decomposition of $\mathfrak{f}$. By Lemma 11.9, $\bar{\mathfrak{f}}$ is delimited and by Lemma 10.14, $\mathsf{heap}(\bar{\mathfrak{f}}) = \mathfrak{h}$, implying $\mathsf{project}(\mathfrak{s},\bar{\mathfrak{f}}) \in \mathsf{type}_\Phi(\mathfrak{s},\mathfrak{h})$ and we can apply Lemma 11.11 to obtain that

$$\mathsf{alloced}(\mathfrak{s},\mathfrak{h}) = \{x \mid \mathfrak{s}(x) \in \mathsf{roots}(\bar{\mathfrak{f}})\}\,.$$

Consequently, all variables in $\mathsf{alloced}(\mathfrak{s},\mathfrak{h})$ occur as root parameters on the right-hand side of magic wands in $\mathsf{project}(\mathfrak{s},\bar{\mathfrak{f}})$. Therefore, $\mathsf{alloced}(\mathfrak{s},\mathfrak{h}) \subseteq \mathsf{alloced}(\mathsf{type}_\Phi(\mathfrak{s},\mathfrak{h}))$. □

11.2.1 *From Stacks to Stack-Aliasing Constraints*

Recall from Section 2.4 the notion of *stack-aliasing constraints*. It is not necessary to differentiate between the types of stacks with identical stack-aliasing constraints. To be able to formalize this observation, we define $\mathbf{Types}^{\Sigma}_{\Phi}(\phi) \triangleq \bigcup\{\mathbf{Types}^{\mathfrak{s}}_{\Phi}(\phi) \mid \mathfrak{s} \in \mathbf{Stacks} \text{ and } \mathsf{aliasing}(\mathfrak{s}) = \Sigma\}$. The above claim can then be expressed as the identity $\mathbf{Types}^{\mathfrak{s}}_{\Phi}(\phi) = \mathbf{Types}^{\mathsf{aliasing}(\mathfrak{s})}_{\Phi}(\phi)$ (cf. Corollary 11.25). Intuitively, the identity holds because isomorphic models have the same type.

Lemma 11.23 (Isomorphic models have the same type)**.** *Let* $(\mathfrak{s},\mathfrak{h})$, $(\mathfrak{s}',\mathfrak{h}')$ *be models with* $\mathrm{dom}(\mathfrak{s}) = \mathrm{dom}(\mathfrak{s}')$ *and* $(\mathfrak{s},\mathfrak{h}) \cong (\mathfrak{s}',\mathfrak{h}')$. *Then* $\mathsf{type}_\Phi(\mathfrak{s},\mathfrak{h}) = \mathsf{type}_\Phi(\mathfrak{s}',\mathfrak{h}')$.

Proof. Let $\mathcal{T} \triangleq \mathsf{type}_\Phi(\mathfrak{s},\mathfrak{h})$ and $\mathcal{T}' \triangleq \mathsf{type}_\Phi(\mathfrak{s}',\mathfrak{h}')$. Let σ be an isomorphism, i.e., such that (1) for all x, $\mathfrak{s}'(x) = \sigma(\mathfrak{s}(x))$ and (2) $\mathfrak{h}' = \{\sigma(l) \mapsto \sigma(\mathfrak{h}(l)) \mid l \in \mathrm{dom}(\mathfrak{h})\}$.

Let $\phi \in \mathsf{type}_\Phi(\mathfrak{s},\mathfrak{h})$. Then there exists a forest $\mathfrak{f} \in \mathsf{forests}_\Phi(\mathfrak{h})$ such that $\phi \in \mathsf{project}(\mathfrak{s},\mathfrak{f})$. We let $\mathfrak{f}'$ be the forest obtained from $\mathfrak{f}$ by renaming every location in $\mathfrak{f}$ with σ, i.e.,

$$\mathfrak{f}' \triangleq \{\sigma[\mathfrak{t}] \mid \mathfrak{t} \in \mathfrak{f}\}\,, \text{ where}$$

$$\sigma[\mathfrak{t}] \triangleq \Big\{\sigma(a) \mapsto \langle \sigma(\mathsf{succ}_{\mathfrak{t}}(a)), \sigma[\mathsf{head}_{\mathfrak{t}}(a)] \Leftarrow \sigma[a \mapsto \mathbf{b}] \star \sigma[\bigstar \mathsf{calls}_{\mathfrak{t}}(a)]\,\rangle \mid a \in \mathrm{dom}(\mathfrak{t}), \mathsf{heap}_{\mathfrak{t}}(a) = \{a \mapsto \mathbf{b}\}\Big\}.$$

Note that by construction, $\mathsf{heap}(\mathfrak{f}') = \sigma \circ \mathsf{heap}(\mathfrak{f}) = \sigma \circ \mathfrak{h} = \mathfrak{h}'$. Consequently, we have forest $\mathfrak{f}' \in \mathsf{forests}_\Phi(\mathfrak{h}')$ and $\phi = \mathsf{project}(\mathfrak{s}',\mathfrak{f}')$ and thus $\phi \in \mathsf{type}_\Phi(\mathfrak{s}',\mathfrak{h}')$. Since ϕ was arbitrary, this proves that $\mathsf{type}_\Phi(\mathfrak{s},\mathfrak{h}) \subseteq \mathsf{type}_\Phi(\mathfrak{s}',\mathfrak{h}')$. By a completely symmetric argument, we can show the other inclusion, $\mathsf{type}_\Phi(\mathfrak{s},\mathfrak{h}) \supseteq \mathsf{type}_\Phi(\mathfrak{s}',\mathfrak{h}')$. □

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Lemma 11.24. *Let $\mathfrak{s}, \mathfrak{s}'$ be stacks with* $\mathsf{aliasing}(\mathfrak{s}) = \mathsf{aliasing}(\mathfrak{s}')$. *It then holds for all formulas ϕ with* $\mathsf{locs}(\phi) = \emptyset$ *that* $\mathbf{Types}^{\mathfrak{s}}_{\Phi}(\phi) = \mathbf{Types}^{\mathfrak{s}'}_{\Phi}(\phi)$.

Proof. Let $\mathcal{T} \in \mathbf{Types}^{\mathfrak{s}}_{\Phi}(\phi)$. Then there exists a heap $\mathfrak{h}$ such that $\mathcal{T} = \mathsf{type}_{\Phi}(\mathfrak{s}, \mathfrak{h})$ and $(\mathfrak{s}, \mathfrak{h}) \models_{\Phi} \phi$. Let $\mathfrak{h}'$ be a heap such that $(\mathfrak{s}, \mathfrak{h}) \cong (\mathfrak{s}', \mathfrak{h}',)$. Note that such an $\mathfrak{h}'$ exists because $\mathsf{aliasing}(\mathfrak{s}) = \mathsf{aliasing}(\mathfrak{s}')$. By Lemma 11.23, $\mathsf{type}_{\Phi}(\mathfrak{s}', \mathfrak{h}') = \mathcal{T}$.

Moreover, by Lemma 8.5, $(\mathfrak{s}', \mathfrak{h}') \models_{\Phi} \phi$, which yields $\mathsf{type}_{\Phi}(\mathfrak{s}', \mathfrak{h}') = \mathcal{T} \in \mathbf{Types}^{\mathfrak{s}'}_{\Phi}(\phi)$.

As $\mathcal{T}$ was arbitrary, it follows that $\mathbf{Types}^{\mathfrak{s}}_{\Phi}(\phi) \subseteq \mathbf{Types}^{\mathfrak{s}'}_{\Phi}(\phi)$; the other inclusion, $\mathbf{Types}^{\mathfrak{s}}_{\Phi}(\phi) \supseteq \mathbf{Types}^{\mathfrak{s}'}_{\Phi}(\phi)$, can then be shown by a symmetrical argument. □

Corollary 11.25. $\mathbf{Types}^{\mathfrak{s}}_{\Phi}(\phi) = \mathbf{Types}^{\mathsf{aliasing}(\mathfrak{s})}_{\Phi}(\phi)$ *for all* $\mathfrak{s} \in \mathbf{Stacks}$ *and all* $\phi \in \mathbf{SLID}^{\mathfrak{g}}_{\mathsf{btw}}$ *with* $\mathsf{locs}(\phi) = \emptyset$.

Corollary 11.26. $\mathbf{Types}^{\mathbf{x}}_{\Phi}(\phi) = \bigcup \left\{ \mathbf{Types}^{\Sigma}_{\Phi}(\phi) \mid \Sigma \in \mathbf{AC}^{\mathbf{x}} \right\}$ *for all* $\phi \in \mathbf{SLID}^{\mathfrak{g}}_{\mathsf{btw}}$ *with* $\mathsf{locs}(\phi) = \emptyset$.

Corollary 11.26 implies that we can express the $\mathbf{x}$-types of ϕ as a *finite* union—induced by the finitely many aliasing constraints $\mathbf{AC}^{\mathbf{x}}$—as opposed to an union over the *infinitely many* different stacks over $\mathbf{x}$. In particular, if we can compute $\mathbf{Types}^{\Sigma}_{\Phi}(\phi)$ for all $\Sigma \in \mathbf{AC}^{\mathbf{x}}$, we can compute $\mathbf{Types}^{\mathbf{x}}_{\Phi}(\phi)$. This is the approach we will take in Chapter 12. Before we get there, we lift several operations from models to types to facilitate computation at the level of types rather than models.

11.2.2 *Composing Types*

We will now show that the types of heaps can be *composed* by applying projection composition, $\bullet_{\mathbf{P}}$, to all members of the types.

THE COMPLETENESS OF $\bullet_{\mathbf{P}}$ ON THE DUSH FRAGMENT. We lift Corollary 11.13 from $\mathfrak{s}$-delimited forests and $\bullet_{\mathbf{F}}$ to DUSHs and $\bullet_{\mathbf{P}}$.

Lemma 11.27. *Let $\mathfrak{s}$ be a stack and let $\mathfrak{h}_1, \mathfrak{h}_2$ be heaps with $(\mathfrak{s}, \mathfrak{h}_1), (\mathfrak{s}, \mathfrak{h}_2) \in$* $\mathbf{Models}^{\mathfrak{g}}_{\Phi}$ *and $\mathfrak{h}_1 + \mathfrak{h}_2 \neq \bot$. If $\phi \in \mathsf{type}_{\Phi}(\mathfrak{s}, \mathfrak{h}_1 + \mathfrak{h}_2)$ is delimited then there exist DUSHs $\psi_i \in \mathsf{type}_{\Phi}(\mathfrak{s}, \mathfrak{h}_i)$, $1 \leq i \leq 2$, such that $\phi \in \psi_1 \bullet_{\mathbf{P}} \psi_2$.*

Proof. Let $\mathfrak{f} \in \mathsf{forests}_{\Phi}(\mathfrak{h}_1 + \mathfrak{h}_2)$ with $\phi = \mathsf{project}(\mathfrak{s}, \mathfrak{f})$. Since ϕ is delimited, $\mathfrak{f}$ is delimited by Lemma 11.15. By Corollary 11.13, there exist $\mathfrak{s}$-delimited forests $\mathfrak{f}_1$ and $\mathfrak{f}_2$ with $\mathfrak{f} \in \mathfrak{f}_1 \bullet_{\mathbf{F}} \mathfrak{f}_2$ and $\mathsf{heap}(\mathfrak{f}_i) = \mathfrak{h}_i$. Let $\mathfrak{f}_1, \mathfrak{f}_2$ be such forests and let $\psi_i \triangleq \mathsf{project}(\mathfrak{s}, \mathfrak{f}_i)$. It follows from Theorem 10.47 that $\mathsf{project}(\mathfrak{s}, \mathfrak{f}) \in \psi_1 \bullet_{\mathbf{P}} \psi_2$ and from Lemma 11.15 that ψ_1, ψ_2 are delimited. □

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Understanding the incompleteness of [KMZ19a]

In the extended version of [KMZ19a] available at [KMZ19b], we made a similar claim about *context decompositions* (cf. [KMZ19b, Lemma 33]), which, roughly speaking, correspond to DUSHs without existential quantifiers. Unfortunately, Lemma 11.27 fails to hold when disallowing (guarded) existentials in projections. That is because the lemma depends on Corollary 11.13, which only holds for the "full" DUSH fragment, including guarded existentials. This is the source of the incompleteness of the approach of [KMZ19a]. I illustrate this problem in the example immediately below this box.

Example 11.28 (Necessity to allow existentials in DUSHs). *Recall the forests* $\mathfrak{f}_1, \mathfrak{f}_2$ *from Example 10.30. Then* $\mathsf{project}(\mathfrak{s}, \mathfrak{f}_0) \in \mathsf{project}(\mathfrak{s}, \mathfrak{f}_1) \bullet_{\mathbf{P}} \mathsf{project}(\mathfrak{s}, \mathfrak{f}_2)$*, because it holds for*

$$\psi \triangleq \boxminus e_1.(\mathsf{p}_2(x_3, x_2, e_1) \mathrel{-\!\!\star} \mathsf{p}_1(x_1, x_2, x_3)) \star (\mathbf{emp} \mathrel{-\!\!\star} \mathsf{ptr}(x_2, e_1)) \\ \star (\mathsf{ptr}(x_2, e_1) \mathrel{-\!\!\star} \mathsf{p}_2(x_3, x_2, e_1))$$

that $\psi \in (\mathsf{project}(\mathfrak{s}, \mathfrak{f}_1) \bar{\star} \mathsf{project}(\mathfrak{s}, \mathfrak{f}_2))$ *and*

$$\psi \rhd \boxminus e_1.\,(\mathsf{p}_2(x_3, x_2, e_1) \mathrel{-\!\!\star} \mathsf{p}_1(x_1, x_2, x_3)) \star (\mathbf{emp} \mathrel{-\!\!\star} \mathsf{p}_2(x_3, x_2, e_1)) \\ \rhd \mathbf{emp} \mathrel{-\!\!\star} \mathsf{p}_1(x_1, x_2, x_3) = \mathsf{project}(\mathfrak{s}, \mathfrak{f}_0),$$

witnessing Lemma 11.27.

In contrast, there do not *exist forests* $\mathfrak{f}'_1$, $\mathfrak{f}'_2$ *such that* $\mathfrak{f}'_i \in \mathsf{forests}_\Phi(\mathfrak{h}_i)$*, neither* $\mathsf{project}(\mathfrak{s}, \mathfrak{f}'_1)$ *nor* $\mathsf{project}(\mathfrak{s}, \mathfrak{f}'_2)$ *contains existentials, and*

$$\mathsf{project}(\mathfrak{s}, \mathfrak{f}_0) \in \mathsf{project}(\mathfrak{s}, \mathfrak{f}'_1) \bullet_{\mathbf{P}} \mathsf{project}(\mathfrak{s}, \mathfrak{f}_2).$$

This illustrates that it is crucial to allow existential quantifiers in DUSHs—for the set of all DUSHs that do not contain existentials, Lemma 11.27 does not hold. This is the key distinction between the context decompositions of [KMZ19a] and the DUSHs I use in this thesis.

Theorem 11.29 (Compositionality of Φ-types). *Let* $\mathfrak{s}$ *be a stack and let* $\mathfrak{h}_1, \mathfrak{h}_2$ *be heaps with* $(\mathfrak{s}, \mathfrak{h}_1), (\mathfrak{s}, \mathfrak{h}_2) \in \mathbf{Models}^{\mathrm{g}}_\Phi$ *and* $\mathfrak{h}_1 + \mathfrak{h}_2 \neq \bot$*. Then*

$$\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}) = \{\phi \in \mathbf{DUSH}_\Phi \mid \textit{ex. } \psi_1 \in \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_1), \\ \psi_2 \in \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_2) \\ \textit{s.t. } \phi \in \psi_1 \bullet_{\mathbf{P}} \psi_2\}.$$

Proof. $\subseteq$ Let $\phi \in \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h})$. By definition, ϕ is delimited. There then exist by Lemma 11.27, $\psi_i \in \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_i)$ such that $\phi \in \psi_1 \bullet_{\mathbf{P}} \psi_2$.

$\supseteq$ Let ϕ be such that there exist for $1 \leq i \leq 2$ formulas $\psi_i \in \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_i)$ with $\phi \in \psi_1 \bullet_{\mathbf{P}} \psi_2$. By definition, there exist forests $\mathfrak{f}_i$ with $\psi_i = \mathsf{project}(\mathfrak{s}, \mathfrak{f}_i)$ and $\mathfrak{h}_i = \mathsf{heap}(\mathfrak{f}_i)$. Theorem 10.47 yields that there exist forests $\bar{\mathfrak{f}}_1, \bar{\mathfrak{f}}_2$ such that $\mathfrak{f}_1 \equiv_\alpha \bar{\mathfrak{f}}_1$, $\mathfrak{f}_2 \equiv_\alpha \bar{\mathfrak{f}}_2$,

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

$\mathfrak{f} \in \bar{\mathfrak{f}}_1 \bullet_{\mathbf{F}} \bar{\mathfrak{f}}_2$, and $\mathsf{project}(\mathfrak{s}, \mathfrak{f}) = \phi$. We know from Lemma 10.41 that $\mathsf{heap}(\bar{\mathfrak{f}}_1) = \mathfrak{h}_1$ and $\mathsf{heap}(\bar{\mathfrak{f}}_2) = \mathfrak{h}_2$, so Lemma 10.7 yields $\mathfrak{h}_1 + \mathfrak{h}_2 = \mathsf{heap}(\bar{\mathfrak{f}}_1 \uplus \bar{\mathfrak{f}}_2)$. It follows from Lemma 10.14 that $\mathfrak{h} = \mathsf{heap}(\mathfrak{f})$, i.e., $\mathfrak{f} \in \mathsf{forests}_\Phi(\mathfrak{h})$. Consequently, $\phi \in \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h})$. □

COMPOSING Φ-TYPES. In light of Theorem 11.29, we define a composition operation $\bullet$ on Φ-types as follows.

Definition 11.30 (Type composition). *Let $\mathcal{T}_1, \mathcal{T}_2$ be Φ-types. The* composition *of $\mathcal{T}_1$ and $\mathcal{T}_2$ is given by*

$$\mathcal{T}_1 \bullet \mathcal{T}_2 \triangleq \begin{cases} \bot, & \textit{if } \mathsf{alloced}(\mathcal{T}_1) \cap \mathsf{alloced}(\mathcal{T}_2) \neq \emptyset \\ \phi_1 \bullet_{\mathbf{P}} \phi_2, & \textit{otherwise.} \end{cases}$$

The composition of heaps $\mathfrak{h}_1 + \mathfrak{h}_2$ is only defined if it does not lead to double allocation. Mirroring this, $\bullet$ is undefined iff the disjoint-union operation $+$ is undefined on the underlying heaps (up to isomorphism).

Lemma 11.31 (Undefinedness of type composition). *Let $\mathfrak{s}$ be a stack and $\mathfrak{h}_1, \mathfrak{h}_2$ be models. Then $\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_1) \bullet \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_2) = \bot$ if and only if for all $\mathfrak{h}_2'$ with $(\mathfrak{s}, \mathfrak{h}_2) \cong (\mathfrak{s}, \mathfrak{h}_2')$, $\mathfrak{h}_1 + \mathfrak{h}_2' = \bot$.*

Proof. If $\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_1) \bullet \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_2) = \bot$, then $\mathsf{alloced}(\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_1)) \cap \mathsf{alloced}(\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_2)) \neq \emptyset$. By Lemma 11.22, we then also have

$$\mathsf{alloced}(\mathfrak{s}, \mathfrak{h}_1) \cap \mathsf{alloced}(\mathfrak{s}, \mathfrak{h}_2) \neq \emptyset.$$

Let x be a variable with $\mathfrak{s}(x) \in \mathrm{dom}(\mathfrak{h}_1) \cap \mathrm{dom}(\mathfrak{h}_2)$. By definition of isomorphism, it follows for all $\mathfrak{h}_2'$ with $(\mathfrak{s}, \mathfrak{h}_2) \cong (\mathfrak{s}, \mathfrak{h}_2')$ that also $\mathfrak{s}(x) \in \mathrm{dom}(\mathfrak{h}_1) \cap \mathrm{dom}(\mathfrak{h}_2')$. Consequently, $\mathfrak{h}_1 + \mathfrak{h}_2' = \bot$.

Conversely, if it holds for all $\mathfrak{h}_2$ with $(\mathfrak{s}, \mathfrak{h}_2) \cong (\mathfrak{s}, \mathfrak{h}_2')$ that $\mathfrak{h}_1 + \mathfrak{h}_2' = \bot$, there must by a variable x with $\mathfrak{s}(x) \in \mathfrak{h}_1 \cap \mathfrak{h}_2$—otherwise, there would exist an $\mathfrak{h}_2'$ such that $\mathfrak{h}_1 + \mathfrak{h}_2' \neq \bot$. By Lemma 11.22, it holds that $\mathsf{alloced}(\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_1)) \cap \mathsf{alloced}(\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_2)) \neq \emptyset$, implying $\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_1) \bullet \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_2) = \bot$. □

Furthermore, $\mathsf{type}_\Phi(\mathfrak{s}, \cdot)$ is a homomorphism from heaps and $+$ to types and $\bullet$.

Corollary 11.32 (Compositionality of type abstraction). *Let $\mathfrak{s}$ be a stack and let $\mathfrak{h}_1, \mathfrak{h}_2$ be heaps with $(\mathfrak{s}, \mathfrak{h}_1), (\mathfrak{s}, \mathfrak{h}_2) \in \mathbf{Models}_\Phi^{\mathfrak{g}}$ and $\mathfrak{h}_1 + \mathfrak{h}_2 \neq \bot$. Then $\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_1 + \mathfrak{h}_2) = \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_1) \bullet \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_2)$.*

Proof. An immediate consequence of Theorem 11.29. □

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

11.2.3 *Instantiating Variables in Φ-Types*

To compute the types of predicate calls $\mathsf{pred}(\mathbf{y})$ from the types of $\mathsf{pred}(\mathsf{fvars}(\mathsf{pred}))$, we need a way to rename the variables in the types of $\mathsf{pred}(\mathsf{fvars}(\mathsf{pred}))$ in a way that reflects the instantiation of the formal arguments $\mathsf{fvars}(\mathsf{pred})$ with the actual arguments $\mathbf{y}$. To this end, we first capture this instantiation at the level of stacks.

Definition 11.33 (Stack instantiation)**.** *Let $\mathfrak{s}, \mathfrak{s}'$ be stacks with $|\mathfrak{s}| \geq |\mathfrak{s}'|$, let $\mathbf{x}, \mathbf{y}$ be sequences of variables with $|\mathbf{x}| = |\mathbf{y}| = |\mathfrak{s}|$ such that $\mathbf{x}$ is a permutation of $\mathrm{dom}(\mathfrak{s})$ and $\mathbf{y}$ contains each variable in $\mathrm{dom}(\mathfrak{s}')$ at least once. Then $\mathfrak{s}'$ is the* $[\mathbf{x}/\mathbf{y}]$-instantiation *of $\mathfrak{s}$ iff it holds for all variables $x \in \mathrm{dom}(\mathfrak{s})$ that $\mathfrak{s}(x) = \mathfrak{s}'(x[\mathbf{x}/\mathbf{y}])$.*

Note in particular that if $\mathfrak{s}'$ is a $[\mathbf{x}/\mathbf{y}]$-instantiation of $\mathfrak{s}$ then $[x]^{\mathfrak{s}}_{=} = [x[\mathbf{x}/\mathbf{y}]]^{\mathfrak{s}'}_{=}$ for all $x \in \mathrm{dom}(\mathfrak{s})$, i.e., stack instantiation cannot merge equivalence classes.

Lemma 11.34. *If $(\mathfrak{s}, \mathfrak{h}) \models_{\Phi} \mathsf{pred}(\mathsf{fvars}(\mathsf{pred}))$ and $\mathfrak{s}'$ is a $[\mathsf{fvars}(\mathsf{pred})/\mathbf{y}]$-instantiation of $\mathfrak{s}$, then $(\mathfrak{s}', \mathfrak{h}) \models_{\Phi} \mathsf{pred}(\mathbf{y})$.*

Proof. By definition of stack instantiation, it holds that

$$\mathfrak{s}'(x[\mathsf{fvars}(\mathsf{pred})/\mathbf{y}]) = \mathfrak{s}(x) \text{ for all } x \in \mathsf{fvars}(\mathsf{pred}). \qquad (\dagger)$$

Using this identity, we obtain

$$\begin{aligned}
&(\mathfrak{s}, \mathfrak{h}) \models_{\Phi} \mathsf{pred}(\mathsf{fvars}(\mathsf{pred})) \\
\Longrightarrow\ &\mathfrak{h} \models_{\Phi} \mathsf{pred}(\mathsf{fvars}(\mathsf{pred}))[\mathrm{dom}(\mathfrak{s}) / \mathrm{img}(\mathfrak{s})] \\
\Longrightarrow\ &\mathfrak{h} \models_{\Phi} \mathsf{pred}(\mathbf{y})[\mathbf{y}/\mathfrak{s}'(\mathbf{y})] && \text{(by } (\dagger)) \\
\Longrightarrow\ &\mathfrak{h} \models_{\Phi} \mathsf{pred}(\mathbf{y})[\mathrm{dom}(\mathfrak{s}') / \mathrm{img}(\mathfrak{s}')] \\
\Longrightarrow\ &(\mathfrak{s}', \mathfrak{h}) \models_{\Phi} \mathsf{pred}(\mathbf{y}). && \square
\end{aligned}$$

We lift instantiation from stacks to types. Recall from Section 10.2 that the stack–forest projection always replaces a location l by the minimal stack variable that is interpreted by l. To guarantee that the result of *type instantiation* yields stack–forest projections, we include a normalization step via $[\cdot]^{\mathfrak{s}'}_{=}$ in Definition 11.35.

Definition 11.35 (Type instantiation)**.** *Let $\mathfrak{s}, \mathfrak{s}'$ be stacks and let $\mathbf{x}, \mathbf{y}$ be such that $\mathfrak{s}'$ is the $[\mathbf{x}/\mathbf{y}]$-instantiation of $\mathfrak{s}$. Moreover, let $\mathbf{y}' \triangleq [\mathbf{y}]^{\mathfrak{s}'}_{=}$ be the sequence obtained by replacing every variable in $\mathbf{y}$ by the maximal variable in its equivalence class. The* $[\mathbf{x}/\mathbf{y}]$-instantiation *of $\mathcal{T}$ is given by*

$$\mathcal{T}[\mathbf{x}/\mathbf{y}] \triangleq \{\phi[\mathbf{x}/\mathbf{y}'] \mid \phi \in \mathcal{T}\}.$$

Thanks to the normalization step via $[\cdot]^{\mathfrak{s}'}_{=}$, the following lemma holds.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Lemma 11.36. *Let $\mathfrak{s}, \mathfrak{s}'$ be stacks with and let $\mathbf{x}, \mathbf{y}$ be such that $\mathfrak{s}'$ is the $[\mathbf{x}/\mathbf{y}]$-instantiation of $\mathfrak{s}$. Then $\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h})[\mathbf{x}/\mathbf{y}] = \mathsf{type}_\Phi(\mathfrak{s}', \mathfrak{h})$.*

Proof. Let $\mathbf{y}' \triangleq [\mathbf{y}]^{\mathfrak{s}'}_{=}$. Let $\phi \in \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h})[\mathbf{x}/\mathbf{y}]$. Then there exists a formula $\phi' \in \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h})$ such that $\phi = \phi'[\mathbf{x}/\mathbf{y}']$. Let $\mathfrak{f} \in \mathsf{forests}_\Phi(\mathfrak{h})$ such that $\mathsf{project}(\mathfrak{s}, \mathfrak{f}) = \phi'$. Note that because stack instantiation does *not* merge equivalence classes of $\mathfrak{s}$, it holds that $\mathsf{project}(\mathfrak{s}', \mathfrak{f}) = \mathsf{project}(\mathfrak{s}, \mathfrak{f})[\mathbf{x}/\mathbf{y}'] = \phi'[\mathbf{x}/\mathbf{y}'] = \phi$. Thus, $\phi \in \mathsf{type}_\Phi(\mathfrak{s}', \mathfrak{h})$.

For the other direction, let $\phi \in \mathsf{type}_\Phi(\mathfrak{s}', \mathfrak{h})$. Let $\mathfrak{f} \in \mathsf{forests}_\Phi(\mathfrak{h})$ such that $\mathsf{project}(\mathfrak{s}', \mathfrak{f}) = \phi$ and define $\phi' \triangleq \mathsf{project}(\mathfrak{s}, \mathfrak{f})$. Observe that $\phi = \phi'[\mathbf{x}/\mathbf{y}']$, because no equivalence classes of $\mathfrak{s}$ are merged by $[\mathbf{x}/\mathbf{y}']$-instantiation. Thus, $\phi \in \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h})[\mathbf{x}/\mathbf{y}]$. □

11.2.4 *Forgetting Variables in Φ-Types*

In the following, we are concerned with removing a variable y from the formulas in a type $\mathcal{T} = \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h})$. Depending on $\mathfrak{s}$, removing y from an individual DUSH $\phi \in \mathcal{T}$ could mean either of two things:

- existentially quantifying over y, if y does not alias with any other free variable, i.e., if $[y]^{\mathfrak{s}}_{=} = \{y\}$,
- replacing y with the maximal variable that aliases with y, i.e., $\max([y]^{\mathfrak{s}}_{=} \setminus \{y\})$.

Formally, we define an operation for removing a variable y from a formula given stack-aliasing constraint $\Sigma \triangleq \mathsf{aliasing}(\mathfrak{s})$.

$$\mathsf{forget}_{\Sigma,y}(\phi) \triangleq \begin{cases} \phi, & \text{if } y \notin \mathsf{fvars}(\phi) \\ \exists y.\,\phi, & \text{if } y \in \mathsf{fvars}(\phi), [y]^{\Sigma}_{=} = \{y\} \\ \phi[y/\max([y]^{\Sigma}_{=} \setminus \{y\})], & \text{if } y \in \mathsf{fvars}(\phi), [y]^{\Sigma}_{=} \supsetneq \{y\}\,. \end{cases}$$

We can then define the removal of y from the free variables of a Φ-type by applying $\phi[y/]$ to all formulas $\phi \in \mathcal{T}$. As $\mathsf{forget}_{\Sigma,y}(\phi)$ may existentially quantify over a root variable of ϕ—in which case the resulting formula is no longer in the DUSH fragment—we intersect the result with $\mathbf{DUSH}_\Phi$.

Definition 11.37 (Forgetting a variable)**.** *Let $\mathcal{T}$ be a type, Σ a stack-aliasing constraint and $y \in$ **Var**. Then forgetting y in $\mathcal{T}$ w.r.t. Σ is defined by* $\mathsf{forget}_{\Sigma,y}(\mathcal{T}) \triangleq \left\{\mathsf{forget}_{\Sigma,y}(\phi) \mid \phi \in \mathcal{T}\right\} \cap \mathbf{DUSH}_\Phi$.

Removing a variable y from a stack and then computing the type is the same as computing the type and then forgetting the variable y, provided $\mathfrak{s}(y) \in \mathrm{dom}(\mathfrak{h})$. The requirement $\mathfrak{s}(y) \in \mathrm{dom}(\mathfrak{h})$ guarantees that it is sound to introduce a guarded existential (as opposed to an unguarded existential) when we remove y from the stack.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Lemma 11.38. *Let $\mathfrak{s}$ be a stack with $y \in \mathrm{dom}(\mathfrak{s})$ and let $\mathfrak{s}'$ be the restriction of $\mathfrak{s}$ to $\mathrm{dom}(\mathfrak{s}) \setminus \{y\}$. Then for all $\mathfrak{h}$ with $\mathfrak{s}(y) \in \mathrm{dom}(\mathfrak{h})$, $\mathsf{type}_\Phi(\mathfrak{s}', \mathfrak{h}) = \mathsf{forget}_{\mathsf{aliasing}(\mathfrak{s}),y}(\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}))$.*

Proof.

$$\begin{aligned}
&\mathsf{type}_\Phi(\mathfrak{s}', \mathfrak{h}) \\
&= \{\mathsf{project}(\mathfrak{s}', \mathfrak{f}) \mid \mathfrak{f} \in \mathsf{forests}_\Phi(\mathfrak{h})\} \cap \mathbf{DUSH}_\Phi \\
&= \left\{\mathsf{forget}_{\mathsf{aliasing}(\mathfrak{s}),y}(\mathsf{project}(\mathfrak{s}, \mathfrak{f})) \mid \mathfrak{f} \in \mathsf{forests}_\Phi(\mathfrak{h})\right\} \cap \mathbf{DUSH}_\Phi \\
&= \mathsf{forget}_{\mathsf{aliasing}(\mathfrak{s}),y}(\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}))
\end{aligned}$$

□

11.2.5 *Size of the Type Domain*

Lemma 11.39. *Let $\Phi \in \mathbf{ID}_{\mathsf{btw}}$ and let $\mathfrak{s}$ be a stack. Let $n \triangleq |\Phi| + |\mathfrak{s}|$. Then $|\mathbf{Types}^{\mathfrak{s}}(\Phi)| \in 2^{2^{\mathcal{O}(n^2 \log(n))}}$.*

Proof. Every Φ-type for stack $\mathfrak{s}$ is a subset of $\mathbf{DUSH}_\Phi^{\mathrm{dom}(\mathfrak{s})}$, which is of size $2^{\mathcal{O}(n^2 \log(n))}$ by Lemma 11.16. □

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.

12 USING TYPES TO DECIDE GUARDED SEPARATION LOGIC

In this chapter, I develop a decision procedure for guarded quantifier-free SL formulas $\phi \in \mathbf{SLID}^{g}_{btw}$, i.e., guarded **SLID** formulas in which inductive definitions are restricted to $\mathbf{ID}_{btw}$. The idea is to use Φ-types to decide satisfiability and entailment for $\mathbf{SLID}^{g}_{btw}$, similar to the way we used AMSs to decide the same reasoning problems for SSL in Part ii.

This involves two steps, as with AMS. (See also Section 3.4 for a discussion of the parallels between the two decision procedures.)

1. We need a *refinement theorem* for $\mathbf{SLID}^{g}_{btw}$ formulas and Φ-types. This justifies reducing satisfiability checking to checking nonemptiness of the set of Φ-types of the formula. I prove the refinement theorem in Section 12.1.
2. We need an algorithm for computing the set $\mathbf{Types}^{\mathbf{x}}_{\Phi}(\phi)$ for finite $\mathbf{x} \in \mathbf{Var}^{*}$ and $\phi \in \mathbf{SLID}^{g}_{btw}$. Actually, we need two algorithms: One for computing the types of the predicate calls of an SID, and one algorithm for computing the types of arbitrary $\mathbf{SLID}^{g}_{btw}$ formulas. I formulate these algorithms in Section 12.2 and Section 12.3, respectively.

In the remainder of this chapter, I would like to avoid dedicated reasoning about points-to assertions. For this reason I only handle *pointer-closed* SIDs, as defined in Definition 8.7, in this chapter.

12.1 THE REFINEMENT THEOREM FOR GUARDED FORMULAS

Our goal in this section is to show that models with the same type satisfy the same $\mathbf{SLID}^{g}_{btw}$ formulas that do not contain location terms. This is perhaps surprising, as types only contain formulas from the DUSH fragment, which is largely orthogonal to $\mathbf{SLID}^{g}_{btw}$. For example, $\mathbf{SLID}^{g}_{btw}$ formulas allow guarded negation and guarded septraction, but neither quantifiers nor unguarded magic wands, whereas DUSHs allow limited use of guarded quantifiers and unguarded magic wands, but neither Boolean structure nor septraction.

DECOMPOSING TYPES INTO SUB-TYPES. In Corollary 11.32, we saw that $\mathrm{type}_{\Phi}(\mathfrak{s}, \mathfrak{h}_1) \bullet \mathrm{type}_{\Phi}(\mathfrak{s}, \mathfrak{h}_2) = \mathrm{type}_{\Phi}(\mathfrak{s}, \mathfrak{h}_1 + \mathfrak{h}_2)$. For the refinement theorem, we need to reverse this result: Given $\mathrm{type}_{\Phi}(\mathfrak{s}, \mathfrak{h}) = \mathcal{T}_1 \bullet \mathcal{T}_2$, we need to find $\mathfrak{h}_1$ and $\mathfrak{h}_2$ with $\mathfrak{h} = \mathfrak{h}_1 + \mathfrak{h}_2$, $\mathrm{type}_{\Phi}(\mathfrak{s}, \mathfrak{h}_1) = \mathcal{T}_1$ and

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

$\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_2) = \mathcal{T}_2$. We need a couple of auxiliary lemmas before we can show this result in Lemma 12.7.

Lemma 12.1. *Let* $(\mathfrak{s}, \mathfrak{h}) \in \mathbf{Models}^{\mathfrak{g}}_{\Phi}$. *Then there exists a forest* $\mathfrak{f}$ *that is* $\mathfrak{s}$*-decomposed and* $\mathfrak{s}$*-delimited with* $\mathsf{project}(\mathfrak{s}, \mathfrak{f}) \in \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h})$.

Proof. Take an arbitrary forest $\bar{\mathfrak{f}}$ with $\mathsf{project}(\mathfrak{s}, \bar{\mathfrak{f}}) \in \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h})$. Such a forest exists by Lemma 11.18. By definition, $\bar{\mathfrak{f}}$ is $\mathfrak{s}$-delimited. Let $\mathfrak{f} \triangleq \mathsf{split}(\mathfrak{f}, \mathsf{img}(\mathfrak{s}))$ be the $\mathfrak{s}$-decomposition of $\bar{\mathfrak{f}}$. By Lemma 11.9, $\mathfrak{f}$ is $\mathfrak{s}$-delimited. By Lemma 10.12, $\mathfrak{f} \blacktriangleright^* \bar{\mathfrak{f}}$. Lemma 10.14 thus gives us that $\mathsf{heap}(\mathfrak{f}) = \mathfrak{h}$. Hence $\mathsf{project}(\mathfrak{s}, \mathfrak{f}) \in \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h})$. $\square$

Lemma 12.2. *Let* $\mathcal{T}_1, \mathcal{T}_2 \in \mathbf{Types}^{\mathfrak{s}}(\Phi)$ *be types with* $\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}) = \mathcal{T}_1 \bullet \mathcal{T}_2$. *Let* $\mathfrak{f}$ *be an* $\mathfrak{s}$*-decomposed* Φ*-forest with* $\mathsf{heap}(\mathfrak{f}) = \mathfrak{h}$ *and* $\mathsf{project}(\mathfrak{s}, \mathfrak{f}) \in \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h})$. *Then there exist* Φ*-forests* $\mathfrak{f}_1, \mathfrak{f}_2$ *such that* $\mathfrak{f} = \mathfrak{f}_1 \uplus \mathfrak{f}_2$ *and* $\mathsf{project}(\mathfrak{s}, \mathfrak{f}_i) \in \mathcal{T}_i$.

Proof. By definition of $\bullet$, there exist formulas $\psi_1 \in \mathcal{T}_1$, $\psi_2 \in \mathcal{T}_2$ with $\mathsf{project}(\mathfrak{s}, \mathfrak{f}) \in \psi_1 \bullet_{\mathbf{P}} \psi_2$. By definition, there exist Φ-forests $\mathfrak{f}'_1, \mathfrak{f}'_2$ with $\mathsf{project}(\mathfrak{s}, \mathfrak{f}'_i) = \psi_i$. Because $\mathcal{T}_1 \bullet \mathcal{T}_2$ is defined, we have $\mathsf{alloced}(\mathcal{T}_1) \cap \mathsf{alloced}(\mathcal{T}_2) = \emptyset$, allowing us to assume w.l.o.g. that $\mathfrak{f}'_1 \uplus \mathfrak{f}'_2 \neq \bot$. As a consequence of Lemma 10.46, there then exist forests $\mathfrak{f}_1, \mathfrak{f}_2$ with $\mathsf{project}(\mathfrak{s}, \mathfrak{f}_i) = \psi_i$ and $\mathfrak{f} \in \mathfrak{f}_1 \bullet_{\mathbf{F}} \mathfrak{f}_2$. Because $\mathfrak{f}$ is $\mathfrak{s}$-decomposed, this implies that zero $\blacktriangleright$-steps were taken by $\bullet_{\mathbf{F}}$, i.e., $\mathfrak{f} = \mathfrak{f}_1 \uplus \mathfrak{f}_2$. $\square$

Lemma 12.3. *Let* $\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}) = \mathcal{T}_1 \bullet \mathcal{T}_2$. *Then there exist* $\mathfrak{s}$*-decomposed,* $\mathfrak{s}$*-delimited* Φ*-forests* $\mathfrak{f}_1, \mathfrak{f}_2$ *such that* $\mathsf{project}(\mathfrak{s}, \mathfrak{f}_i) \in \mathcal{T}_i$ *and* $\mathsf{heap}(\mathfrak{f}_1 \uplus \mathfrak{f}_2) = \mathfrak{h}$.

Proof. Let $\mathfrak{f}$ be an $\mathfrak{s}$-decomposed, $\mathfrak{s}$-delimited forest with $\mathsf{project}(\mathfrak{s}, \mathfrak{f}) \in \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h})$; note that there exists at least one such forest by Lemma 12.1. By Lemma 12.2, there then exist forests $\mathfrak{f}_1, \mathfrak{f}_2$ with $\mathfrak{f} = \mathfrak{f}_1 \uplus \mathfrak{f}_2$ and $\mathsf{project}(\mathfrak{s}, \mathfrak{f}_i) \in \mathcal{T}_i$, $1 \leq i \leq 2$. $\square$

Definition 12.4 (Roots of a DUSH). *Let* $\phi = \exists\mathbf{e}. \bigstar_{1 \leq i \leq k} \forall \mathbf{a_i}. (\zeta_i \mathrel{-\!\!\star} \mathsf{pred}_i(\mathbf{z_i}))$ *be a DUSH. The roots of* ψ *are the set*

$$\mathsf{dushroots}_{\mathfrak{s}}(\phi) \triangleq \bigcup \left\{ [\mathsf{predroot}(\mathsf{pred}_i(\mathbf{z_i}))]^{\mathfrak{s}}_{=} \mid 1 \leq i \leq k \right\}.$$

Clearly, the roots of a forest are connected to the roots of a DUSH via the stack.

Lemma 12.5. *Let* $\mathfrak{f}$ *be a* Φ*-forest. Then*

$$\mathsf{dushroots}_{\mathfrak{s}}(\mathsf{project}(\mathfrak{s}, \mathfrak{f})) = \{ x \mid \mathfrak{s}(x) \in \mathsf{roots}(\mathfrak{f}) \}.$$

Proof. Let $\phi \triangleq \mathsf{project}(\mathfrak{s}, \mathfrak{f})$. By definition of DUSHs, $\mathsf{roots}(\mathfrak{f}) \subseteq \mathsf{img}(\mathfrak{s})$. Every root $l \in \mathsf{roots}(\mathfrak{f})$ is therefore replaced by a variable in $\mathfrak{s}^{-1}(l)$ by stack–forest projection. Since $\mathsf{dushroots}_{\mathfrak{s}}(\phi)$ closes the set of roots under $[\cdot]^{\mathfrak{s}}_{=}$, we obtain that $\mathsf{dushroots}_{\mathfrak{s}}(\phi)$ contains *all* variables x with $\mathfrak{s}(x) \in \mathsf{roots}(\mathfrak{f})$. $\square$

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Lemma 12.6. *Let $\mathcal{T}$ be a Φ-type and $\psi \in \mathcal{T}$. There exists a formula $\psi' \in \mathcal{T}$ such that $\psi' \rhd^* \psi$ and $\mathsf{alloced}(\mathcal{T}) = \mathsf{dushroots}_{\mathfrak{s}}(\psi')$.*

Proof. Let $(\mathfrak{s},\mathfrak{h})$ be such that $\mathsf{type}_\Phi(\mathfrak{s},\mathfrak{h}) = \mathcal{T}$. In particular, it then holds that $\mathsf{alloced}(\mathfrak{s},\mathfrak{h}) = \mathsf{alloced}(\mathcal{T})$ (†). By definition of Φ-types, there then exists a Φ-forest $\mathfrak{f}$ with $\mathsf{heap}(\mathfrak{f}) = \mathfrak{h}$ and $\mathsf{project}(\mathfrak{s},\mathfrak{f}) = \psi$. Let $\bar{\mathfrak{f}} \triangleq \mathsf{split}(\mathfrak{f}, \mathsf{img}(\mathfrak{s}))$ be the $\mathfrak{s}$-decomposition of $\mathfrak{f}$ and write $\psi' \triangleq \mathsf{project}(\mathfrak{s},\bar{\mathfrak{f}})$. We show that ψ' has the desired properties.

First, $\bar{\mathfrak{f}} \blacktriangleright^* \mathfrak{f}$ by Lemma 10.12. Observe that $\mathfrak{f} \in \bar{\mathfrak{f}} \bullet_{\mathbf{F}} \emptyset$. Corollary 10.43 therefore guarantees that $\psi \in \psi' \bullet_{\mathbf{P}} \mathbf{emp}$ and thus $\psi' \rhd^* \psi$.

Second, by Lemma 10.14, $\mathsf{heap}(\bar{\mathfrak{f}}) = \mathsf{heap}(\mathfrak{f})$ and thus $\psi' \in \mathcal{T}$. Moreover, by Lemma 11.11, $\mathsf{alloced}(\mathfrak{s},\mathfrak{h}) = \{x \mid \mathfrak{s}(x) \in \mathsf{roots}(\bar{\mathfrak{f}})\}$. We combine this with (†) to derive $\mathsf{alloced}(\mathcal{T}) = \{x \mid \mathfrak{s}(x) \in \mathsf{roots}(\bar{\mathfrak{f}})\}$. Lemma 12.5 yields that $\mathsf{alloced}(\mathcal{T}) = \mathsf{dushroots}_{\mathfrak{s}}(\psi')$. □

Lemma 12.7 (Type decomposability)**.** *Let $(\mathfrak{s},\mathfrak{h}) \in \mathbf{Models}_\Phi^{\mathfrak{g}}$ and assume that $\mathsf{type}_\Phi(\mathfrak{s},\mathfrak{h}) = \mathcal{T}_1 \bullet \mathcal{T}_2$. Then there exist $\mathfrak{h}_1, \mathfrak{h}_2$ such that $\mathfrak{h} = \mathfrak{h}_1 + \mathfrak{h}_2$, $\mathcal{T}_1 = \mathsf{type}_\Phi(\mathfrak{s},\mathfrak{h}_1)$, and $\mathcal{T}_2 = \mathsf{type}_\Phi(\mathfrak{s},\mathfrak{h}_2)$.*

Proof. Let $\mathcal{T} \triangleq \mathsf{type}_\Phi(\mathfrak{s},\mathfrak{h})$. By Lemma 12.3, there exist $\mathfrak{s}$-decomposed, $\mathfrak{s}$-delimited forests $\mathfrak{f}_1, \mathfrak{f}_2$ with

1. $\mathsf{project}(\mathfrak{s},\mathfrak{f}_i) \in \mathcal{T}_i$, $1 \le i \le 2$,
2. $\mathsf{heap}(\mathfrak{f}_1 \uplus \mathfrak{f}_2) = \mathfrak{h}$ and thus $\mathsf{project}(\mathfrak{s},\mathfrak{f}_1 \uplus \mathfrak{f}_2) \in \mathcal{T}$.

Define $\mathfrak{h}_1 \triangleq \mathsf{heap}(\mathfrak{f}_1)$, $\mathfrak{h}_2 \triangleq \mathsf{heap}(\mathfrak{f}_2)$. Observe that $\mathfrak{h}_1 + \mathfrak{h}_2 = \mathfrak{h}$. Because the $\mathfrak{f}_i$ are $\mathfrak{s}$-delimited, we have $\mathsf{dangling}(\mathfrak{h}_i) \subseteq \mathsf{img}(\mathfrak{s})$ (♣). Further, we have

$$\mathsf{alloced}(\mathfrak{s},\mathfrak{h}_i) = \{x \mid \mathfrak{s}(x) \in \mathsf{roots}(\mathfrak{f}_i)\} = \mathsf{alloced}(\mathcal{T}_i), \qquad (†)$$

where the first equality follows from Lemma 11.11, and the second equality holds because the $\mathfrak{f}_i$ are decomposed and $\mathsf{project}(\mathfrak{s},\mathfrak{f}_i) \in \mathcal{T}_i$.

I will show that $\mathcal{T}_i = \mathsf{type}_\Phi(\mathfrak{s},\mathfrak{h}_i)$. I only prove $\mathcal{T}_1 \subseteq \mathsf{type}_\Phi(\mathfrak{s},\mathfrak{h}_1)$, as the other three inclusions can be proved analogously.

Let $\psi_1 \in \mathcal{T}_1$. By Lemma 12.6, there then exists a formula $\psi_1' \in \mathcal{T}_1$ such that $\psi_1' \rhd^* \psi_1$, and $\mathsf{alloced}(\mathcal{T}_1) = \mathsf{dushroots}_{\mathfrak{s}}(\psi_1')$. Combining this fact with (†) yields $\mathsf{alloced}(\mathfrak{s},\mathfrak{h}_1) = \mathsf{dushroots}_{\mathfrak{s}}(\psi_1')$ (‡).

Let $\psi_2' \triangleq \mathsf{project}(\mathfrak{s},\mathfrak{f}_2) \in \mathcal{T}_2$ for $\mathfrak{f}_2$ as above. By definition of type composition, $\bullet$, it follows that $\psi_1' \star \psi_2' \in \mathcal{T}$. Consequently, there exist Φ-forests $\mathfrak{g}', \mathfrak{g}_1', \mathfrak{g}_2'$ with $\mathfrak{g}' \in \mathsf{forests}_\Phi(\mathfrak{s},\mathfrak{h})$, $\mathfrak{g}' = \mathfrak{g}_1' \uplus \mathfrak{g}_2'$, $\mathsf{project}(\mathfrak{s},\mathfrak{g}_1') = \psi_1'$ and $\mathsf{project}(\mathfrak{s},\mathfrak{g}_2') = \psi_2'$.

Observe that (‡) implies that $\{x \mid \mathfrak{s}(x) \in \mathsf{roots}(\mathfrak{g}_1')\} = \mathsf{alloced}(\mathfrak{s},\mathfrak{h}_1)$. In particular, $\mathfrak{g}_1'$ is $\mathfrak{s}$-decomposed. Moreover, $\{x \mid \mathfrak{s}(x) \in \mathsf{roots}(\mathfrak{g}_2')\} = \mathsf{alloced}(\mathfrak{s},\mathfrak{h}_2)$: Because $\mathfrak{f}_2$ and $\mathfrak{g}_2'$ have identical stack–forest projections, it holds that

$$\{x \mid \mathfrak{s}(x) \in \mathsf{roots}(\mathfrak{g}_2')\} = \{x \mid \mathfrak{s}(x) \in \mathsf{roots}(\mathfrak{f}_2)\};$$

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

the identity then follows by (†).

Trivially, $\mathrm{dom}(\mathfrak{g}'_i) \supseteq \mathsf{roots}(\mathfrak{g}'_i)$, so

$$\{x \mid \mathfrak{s}(x) \in \mathrm{dom}(\mathfrak{g}'_1)\} \supseteq \mathsf{alloced}(\mathfrak{s}, \mathfrak{h}_1),$$
$$\{x \mid \mathfrak{s}(x) \in \mathrm{dom}(\mathfrak{g}'_2)\} \supseteq \mathsf{alloced}(\mathfrak{s}, \mathfrak{h}_2).$$

Since $\mathsf{alloced}(\mathfrak{s}, \mathfrak{h}) = \mathsf{alloced}(\mathfrak{s}, \mathfrak{h}_1) \cup \mathsf{alloced}(\mathfrak{s}, \mathfrak{h}_2)$, we have that every allocated location in $\mathrm{img}(\mathfrak{s})$ is contained in at least one of the sets $\mathrm{dom}(\mathfrak{g}_1)$ and $\mathrm{dom}(\mathfrak{g}_2)$. It follows that

$$\{x \mid \mathfrak{s}(x) \in \mathrm{dom}(\mathfrak{g}'_i)\} = \mathsf{alloced}(\mathfrak{s}, \mathfrak{h}_i). \qquad (\spadesuit)$$

If this weren't the case, there would be double allocation of a stack variable. We use this result to argue that $\mathfrak{h}_1 = \mathsf{heap}(\mathfrak{g}'_1)$.

Write $\{\mathfrak{t}_1, \ldots, \mathfrak{t}_k\} \triangleq \mathfrak{g}'_1$ and let $\mathfrak{h}_{1,j} \triangleq \mathsf{heap}(\mathfrak{t}_j)$. Because $\mathfrak{t}_j$ is $\mathfrak{s}$-delimited, $\mathsf{root}(\mathfrak{t}_j) \in \mathrm{img}(\mathfrak{s})$, so ($\spadesuit$) implies that $\mathsf{root}(\mathfrak{t}_j) \in \mathrm{dom}(\mathfrak{h}_1)$. Hence, $\mathfrak{h}_{1,j} \cap \mathfrak{h}_1 \neq \emptyset$. Assume $\mathfrak{h}_{1,j} \subsetneq \mathfrak{h}_1$. Because $\mathfrak{g}'_1$ is $\mathfrak{s}$-decomposed, there then exists a location $l \in \mathrm{dom}(\mathfrak{h}_{1,j}) \setminus \mathrm{img}(\mathfrak{s})$ such that $l \in \mathsf{dangling}(\mathfrak{h}_1)$. This contradicts ($\clubsuit$). Consequently, $\mathfrak{h}_{1,j} \subseteq \mathfrak{h}_1$. It follows that $\mathsf{heap}(\mathfrak{g}'_1) \supseteq \mathfrak{h}_1$.

A similar argument shows that for all $\mathfrak{t}' \in \mathfrak{g}'_2$, $\mathfrak{h}_1 \cap \mathsf{heap}(\mathfrak{t}') = \emptyset$: $\mathfrak{h}_1$ cannot contain the root of $\mathfrak{t}'$ without contradicting (‡), and thus cannot contain *any* of the locations in $\mathrm{dom}(\mathfrak{t}')$ without contradicting ($\clubsuit$). Consequently, $\mathsf{heap}(\mathfrak{g}'_1) \subseteq \mathfrak{h}_1$. By combining the two inequalities, we finally obtain $\mathsf{heap}(\mathfrak{g}'_1) = \mathfrak{h}_1$. Therefore, $\psi'_1 \in \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_1)$.

Since $\psi'_1 \rhd^* \psi_1$, in particular $\psi_1 \in \psi'_1 \bullet_{\mathbf{P}} \mathbf{emp}$. Lemma 10.46 then gives us a forest $\mathfrak{g}_1$ s.t. $\mathfrak{g}'_1 \blacktriangleright^* \mathfrak{g}_1$ and $\mathsf{project}(\mathfrak{s}, \mathfrak{g}_1) = \psi_1$. Because also $\mathsf{heap}(\mathfrak{g}'_1) = \mathsf{heap}(\mathfrak{g}_1)$ by Lemma 10.14, $\mathfrak{g}_1 \in \mathsf{forests}_\Phi(\mathfrak{h}_1)$ and $\mathsf{project}(\mathfrak{s}, \mathfrak{g}_1) \in \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_1)$. As $\mathfrak{g}_1$ was an arbitrary forest with projection in $\mathcal{T}_1$, $\mathcal{T}_1 \subseteq \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_1)$ as desired. □

MODELS WITH THE SAME TYPES SATISFY THE SAME GUARDED FORMULAS. Lemma 12.7 allows us to show the refinement theorem, Theorem 12.10, which states that guarded models with identical types satisfy the same guarded formulas.

We need two additional properties of types to prove the refinement theorem. First, observe that a model satisfies all the formulas in its type.

Lemma 12.8. *If* $\phi \in \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h})$ *then* $(\mathfrak{s}, \mathfrak{h}) \models_\Phi \phi$.

Proof. Since $\phi \in \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h})$, there exists a Φ-forest $\mathfrak{f}$ with $\mathsf{heap}(\mathfrak{f}) = \mathfrak{h}$ and $\mathsf{project}(\mathfrak{s}, \mathfrak{f}) = \mathfrak{h}$. By Lemma 10.29, $(\mathfrak{s}, \mathsf{heap}(\mathfrak{f})) \models_\Phi \mathsf{project}(\mathfrak{s}, \mathfrak{f})$. Substituting the previous identities, we obtain $(\mathfrak{s}, \mathfrak{h}) \models_\Phi \phi$. □

Second, if two models have the same type, then the first model is guarded iff the second model is guarded.

Lemma 12.9. *Let* $\mathfrak{s} \in \mathbf{Stacks}$, $\mathfrak{h}_1, \mathfrak{h}_2 \in \mathbf{Heaps}$ *and assume that (1)* $(\mathfrak{s}, \mathfrak{h}_1) \in \mathbf{Models}^{\mathfrak{g}}_\Phi$ *and (2)* $\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_1) = \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_2)$. *Then* $(\mathfrak{s}, \mathfrak{h}_2) \in \mathbf{Models}^{\mathfrak{g}}_\Phi$.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Proof. By Definition 8.8, $(\mathfrak{s}, \mathfrak{h}_1) \models_\Phi \mathsf{pred}_1(\mathbf{x_1}) \star \cdots \star \mathsf{pred}_k(\mathbf{x_k})$ for some choice of predicate calls. For $\psi \triangleq \bigstar_{1 \leq i \leq k}(\mathbf{emp} \mathrel{-\!\!\star} \mathsf{pred}_i(\mathbf{x_i}))$, it then holds that $\psi \in \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_1) = \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_2)$. Lemma 12.8 yields that $(\mathfrak{s}, \mathfrak{h}_2) \models_\Phi \psi$ holds. Therefore, $(\mathfrak{s}, \mathfrak{h}_2) \models_\Phi \mathsf{pred}_1(\mathbf{x_1}) \star \cdots \star \mathsf{pred}_k(\mathbf{x_k})$, implying $(\mathfrak{s}, \mathfrak{h}_2) \in \mathbf{Models}^{\mathrm{g}}_\Phi$. □

Theorem 12.10 (Refinement theorem)**.** *Let $\mathfrak{s}$ be a stack and $\mathfrak{h}_1, \mathfrak{h}_2$ be heaps. Let $\phi \in \mathbf{SLID}^{\mathrm{g}}_{\mathrm{btw}}$ with $\mathsf{locs}(\phi) = \emptyset$ and let $\Phi \in \mathbf{ID}_{\mathrm{btw}}$ be pointer-closed w.r.t. ϕ. Moreover, assume $\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_1) = \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_2)$. Then $(\mathfrak{s}, \mathfrak{h}_1) \models_\Phi \phi$ iff $(\mathfrak{s}, \mathfrak{h}_2) \models_\Phi \phi$.*

Proof. If $(\mathfrak{s}, \mathfrak{h}_1), (\mathfrak{s}, \mathfrak{h}_2) \notin \mathbf{Models}^{\mathrm{g}}_\Phi$, then $\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_1)$ is undefined and neither model satisfies ϕ by Lemma 8.9.

Assume $(\mathfrak{s}, \mathfrak{h}_1), (\mathfrak{s}, \mathfrak{h}_2) \in \mathbf{Models}^{\mathrm{g}}_\Phi$ We proceed by induction on the structure of ϕ. We only show that if $(\mathfrak{s}, \mathfrak{h}_1) \models_\Phi \phi$ then $(\mathfrak{s}, \mathfrak{h}_2) \models_\Phi \phi$, as the proof of the other direction is completely analogous.

CASE $\phi = \mathbf{emp}$. Assume that $(\mathfrak{s}, \mathfrak{h}_1) \models_\Phi \phi$. By the semantics of $\mathbf{emp}$, $\mathfrak{h}_1 = \emptyset$. Let $\mathfrak{f} = \emptyset$ be the empty forest. Then $\mathfrak{f} \in \mathsf{forests}_\Phi(\mathfrak{h}_1)$ and thus $\mathbf{emp} = \mathsf{project}(\mathfrak{s}, \mathfrak{f}) \in \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_1) = \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_2)$. Thus $(\mathfrak{s}, \mathfrak{h}_2) \models_\Phi \mathbf{emp}$ by Lemma 12.8.

CASES $\phi = x \approx y$, $\phi = x \not\approx y$. Trivial, because the models have the same stack.

CASE $\phi = x \mapsto \langle y_1, \ldots, y_k \rangle$. Assume that $(\mathfrak{s}, \mathfrak{h}_1) \models_\Phi \phi$. By assumption, Φ is pointer-closed, so $(\mathfrak{s}, \mathfrak{h}_1) \models_\Phi \mathsf{ptr}_k(x, y_1, \ldots, y_k)$. Let

$$\mathfrak{t} = \{\mathfrak{s}(x) \mapsto \langle \emptyset, \mathsf{ptr}_k(\mathfrak{s}(x), \mathfrak{s}(y_1), \ldots, \mathfrak{s}(y_k)) \Leftarrow \mathfrak{s}(x) \mapsto \langle \mathfrak{s}(y_1), \ldots, \mathfrak{s}(y_k) \rangle \rangle\}$$

be a tree and $\mathfrak{f} = \{\mathfrak{t}\}$. Observe that $\mathfrak{f} \in \mathsf{forests}_\Phi(\mathfrak{h}_1)$ and

$$\mathsf{ptr}_k(x, y_1, \ldots, y_k) = \mathsf{project}(\mathfrak{s}, \mathfrak{f}) \in \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_1) = \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_2).$$

Thus $(\mathfrak{s}, \mathfrak{h}_2) \models_\Phi \mathsf{ptr}_k(x, y_1, \ldots, y_k)$ by Lemma 12.8. By definition of ptr_k, we conclude that $(\mathfrak{s}, \mathfrak{h}_2) \models_\Phi x \mapsto \langle y_1, \ldots, y_k \rangle$.

CASE $\phi = \mathsf{pred}(z_1, \ldots, z_k)$. Assume that $(\mathfrak{s}, \mathfrak{h}_1) \models_\Phi \phi$. By Lemma 10.5, there exists a Φ-tree $\mathfrak{t}$ with $\mathsf{rootpred}(\mathfrak{t}) = \mathsf{pred}(\mathfrak{s}(z_1), \ldots, \mathfrak{s}(z_k))$, $\mathsf{allholepreds}(\mathfrak{t}) = \emptyset$, and $\mathsf{heap}(\{\mathfrak{t}\}) = \mathfrak{h}_1$. Let

$$\psi \triangleq \mathsf{pred}(\mathfrak{s}(z_1), \ldots, \mathfrak{s}(z_k))[\mathrm{dom}(\mathfrak{s}^{-1}_{\max}) / \mathrm{img}(\mathfrak{s}^{-1}_{\max})].$$

Note that $\psi = \mathsf{project}(\mathfrak{s}, \{\mathfrak{t}\})$ by definition of stack–forest projection and thus $\psi \in \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_1) = \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_2)$. By Lemma 12.8, $(\mathfrak{s}, \mathfrak{h}_2) \models_\Phi \psi$. Observe that while $\psi \neq \mathsf{pred}(\mathbf{z})$ is possible, we have by definition of $\mathfrak{s}^{-1}_{\max}$ that the parameters of the predicate call in ψ evaluate to the same locations as the parameters $\mathbf{z}$. Thus, $(\mathfrak{s}, \mathfrak{h}_2) \models_\Phi \mathsf{pred}(z_1, \ldots, z_k)$.

TU Bibliothek Your knowledge hub
Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.

CASE $\phi = \phi_1 \wedge \phi_2$. Assume that $(\mathfrak{s}, \mathfrak{h}_1) \models_\Phi \phi$. We then have that $(\mathfrak{s}, \mathfrak{h}_1) \models_\Phi \phi_1$ and $(\mathfrak{s}, \mathfrak{h}_1) \models_\Phi \phi_2$. By the induction hypotheses, $(\mathfrak{s}, \mathfrak{h}_2) \models_\Phi \phi_1$ and $(\mathfrak{s}, \mathfrak{h}_2) \models_\Phi \phi_2$ and thus $(\mathfrak{s}, \mathfrak{h}_2) \models_\Phi \phi_1 \wedge \phi_2$.

CASE $\phi = \phi_1 \vee \phi_2$. Assume that $(\mathfrak{s}, \mathfrak{h}_1) \models_\Phi \phi$. We then have that $(\mathfrak{s}, \mathfrak{h}_1) \models_\Phi \phi_1$ or $(\mathfrak{s}, \mathfrak{h}_1) \models_\Phi \phi_2$. Assume w.l.o.g. that $(\mathfrak{s}, \mathfrak{h}_1) \models_\Phi \phi_1$. By induction, $(\mathfrak{s}, \mathfrak{h}_2) \models_\Phi \phi_1$ and thus $(\mathfrak{s}, \mathfrak{h}_2) \models_\Phi \phi_1 \vee \phi_2$.

CASE $\phi = \phi_1 \wedge \neg\phi_2$. Assume that $(\mathfrak{s}, \mathfrak{h}_1) \models_\Phi \phi$. We then have that $(\mathfrak{s}, \mathfrak{h}_1) \models_\Phi \phi_1$ and $(\mathfrak{s}, \mathfrak{h}_2) \not\models_\Phi \phi_2$. By the induction hypotheses, $(\mathfrak{s}, \mathfrak{h}_2) \models_\Phi \phi_1$ $(\mathfrak{s}, \mathfrak{h}_2) \not\models_\Phi \phi_2$ and thus $(\mathfrak{s}, \mathfrak{h}_2) \models_\Phi \phi_1 \wedge \neg\phi_2$.

CASE $\phi = \phi_1 \star \phi_2$. Assume that $(\mathfrak{s}, \mathfrak{h}_1) \models_\Phi \phi$. By the semantics of $\star$, there exist $\mathfrak{h}_{1,1}$ and $\mathfrak{h}_{1,2}$ such that $(\mathfrak{s}, \mathfrak{h}_{1,i}) \models_\Phi \phi_i$ for $1 \leq i \leq 2$. Let $\mathcal{T}_i \triangleq \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_{1,i})$ and apply Lemma 12.7 to $(\mathfrak{s}, \mathfrak{h}_2)$, $\mathcal{T}_1$ and $\mathcal{T}_2$ to obtain models $\mathfrak{h}_{2,1}$ and $\mathfrak{h}_{2,2}$ with $\mathfrak{h}_2 = \mathfrak{h}_{2,1} + \mathfrak{h}_{2,2}$, $\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_{2,1}) = \mathcal{T}_1$, and $\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_{2,2}) = \mathcal{T}_2$.

Note that $(\mathfrak{s}, \mathfrak{h}_{1,i}) \in \mathbf{Models}^{\mathfrak{g}}_\Phi$ by Lemma 8.9. Lemma 12.9, applied once to $\mathfrak{h}_{1,1}$ and $\mathfrak{h}_{2,1}$, once to $\mathfrak{h}_{1,2}$ and $\mathfrak{h}_{2,2}$, then yields that also $(\mathfrak{s}, \mathfrak{h}_{2,i}) \in \mathbf{Models}^{\mathfrak{g}}_\Phi$.

We can therefore apply the induction hypothesis for both $\mathfrak{h}_{1,1}$, $\mathfrak{h}_{1,2}$, ϕ_1 and $\mathfrak{h}_{2,1}$, $\mathfrak{h}_{2,2}$, ϕ_2 to obtain that for $1 \leq i \leq 2$, $(\mathfrak{s}, \mathfrak{h}_{2,i}) \models_\Phi \phi_i$. As $\mathfrak{h}_{2,1} + \mathfrak{h}_{2,2} = \mathfrak{h}_2$, it follows by the semantics of $\star$ that $(\mathfrak{s}, \mathfrak{h}_2) \models_\Phi \phi$.

CASE $\phi = \phi_0 \wedge (\phi_1 \mathbin{-\!\!\circledast} \phi_2)$. Assume that $(\mathfrak{s}, \mathfrak{h}_1) \models_\Phi \phi$. Then there exists a heap $\mathfrak{h}_0$ with $(\mathfrak{s}, \mathfrak{h}_0) \models_\Phi \phi_1$ and $(\mathfrak{s}, \mathfrak{h}_1 + \mathfrak{h}_0) \models_\Phi \phi_2$.

Since $(\mathfrak{s}, \mathfrak{h}_1)$ and $(\mathfrak{s}, \mathfrak{h}_2)$ have the same type, $\mathsf{alloced}(\mathfrak{s}, \mathfrak{h}_1) = \mathsf{alloced}(\mathfrak{s}, \mathfrak{h}_2)$ holds. We can therefore assume w.l.o.g. that $\mathfrak{h}_2 + \mathfrak{h}_0$ is defined—if this is not the case, simply replace $\mathfrak{h}_0$ with a heap $\mathfrak{h}'_0$ such that $(\mathfrak{s}, \mathfrak{h}_0) \cong (\mathfrak{s}, \mathfrak{h}'_0)$ and both $\mathfrak{h}_1 + \mathfrak{h}'_0$ and $\mathfrak{h}_2 + \mathfrak{h}'_0$ are defined. By Lemma 8.5, $\mathfrak{h}_1 + \mathfrak{h}'_0 \models_\Phi \phi$.

By assumption $(\mathfrak{s}, \mathfrak{h}_1) \in \mathbf{Models}^{\mathfrak{g}}_\Phi$ and by Lemma 8.9, $(\mathfrak{s}, \mathfrak{h}_0) \in \mathbf{Models}^{\mathfrak{g}}_\Phi$. Thus, $(\mathfrak{s}, \mathfrak{h}_1 + \mathfrak{h}_0) \in \mathbf{Models}^{\mathfrak{g}}_\Phi$. Our assumptions and Corollary 11.32 thus yield

$$\begin{aligned}
\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_1 + \mathfrak{h}_0) &= \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_1) \bullet \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_0) \\
&= \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_2) \bullet \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_0) \\
&= \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_2 + \mathfrak{h}_0).
\end{aligned}$$

We apply the induction hypothesis for ϕ_0 and $\mathfrak{h}_1$ and $\mathfrak{h}_2$ to obtain $(\mathfrak{s}, \mathfrak{h}_2) \models_\Phi \phi_0$ Furthermore, we apply the induction hypotheses for formula ϕ_2 and models $(\mathfrak{s}, \mathfrak{h}_1 + \mathfrak{h}_0)$ and $(\mathfrak{s}, \mathfrak{h}_2 + \mathfrak{h}_0)$ to obtain $(\mathfrak{s}, \mathfrak{h}_2 + \mathfrak{h}_0) \models_\Phi \phi_2$. By the semantics of $\mathbin{-\!\!\circledast}$ and $\wedge$, we get $(\mathfrak{s}, \mathfrak{h}_2) \models_\Phi \phi_0 \wedge (\phi_1 \mathbin{-\!\!\circledast} \phi_2)$.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

CASE $\phi = \phi_0 \wedge (\phi_1 \twoheadrightarrow\!\!\star\, \phi_2)$. x Analogous to the case for guarded septraction, except that we must consider *arbitrary* models $\mathfrak{h}_0$ with $(\mathfrak{s}, \mathfrak{h}_0) \models_\Phi \phi_1$ and $(\mathfrak{s}, \mathfrak{h}_1 + \mathfrak{h}_0) \models_\Phi \phi_2$. Lemma 8.5 and Corollary 11.32, and the induction hypotheses can then be applied just like in the case for septraction. □

For Theorem 12.10 to hold, it is crucial that $\mathbf{SLID}^{\mathrm{g}}_{\mathrm{btw}}$ only contains quantifier-free formulas, as illustrated by the following example.

Example 12.11. *Recall the SID Φ_{ls} from Example 8.2. Moreover, let $(\mathfrak{s}, \mathfrak{h}_k)$, $k \in \mathbb{N}$, be a list of length k from x_1 to x_2. It then holds for all $i, j \geq 2$ that* $\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_i) = \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_j)$. *However,*

$$(\mathfrak{s}, \mathfrak{h}_2) \not\models_\Phi \exists \langle y_1, y_2 \rangle \,.\, \mathsf{lseg}(x_1, y_1) \star \mathsf{lseg}(y_1, y_2) \star \mathsf{lseg}(y_2, x_2),$$

whereas

$$(\mathfrak{s}, \mathfrak{h}_j) \models_\Phi \exists \langle y_1, y_2 \rangle \,.\, \mathsf{lseg}(x_1, y_1) \star \mathsf{lseg}(y_1, y_2) \star \mathsf{lseg}(y_2, x_2)$$

for all $j \geq 3$.

Theorem 12.10 immediately implies that if the type of a guarded model $(\mathfrak{s}, \mathfrak{h})$ is among the types of ϕ, then $(\mathfrak{s}, \mathfrak{h})$ is a model of ϕ.

Corollary 12.12. *Let $\phi \in \mathbf{SLID}^{\mathrm{g}}_{\mathrm{btw}}$ and let $(\mathfrak{s}, \mathfrak{h}) \in \mathbf{Models}^{\mathrm{g}}_\Phi$ be a guarded model such that* $\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}) \in \mathbf{Types}^{\mathfrak{s}}_\Phi(\phi)$. *Then $(\mathfrak{s}, \mathfrak{h}) \models_\Phi \phi$.*

Proof. Because $\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}) \in \mathbf{Types}^{\mathfrak{s}}_\Phi(\phi)$, there exists, by definition of $\mathbf{Types}^{\mathfrak{s}}_\Phi(\phi)$, a heap $\mathfrak{h}'$ such that $(\mathfrak{s}, \mathfrak{h}') \models_\Phi \phi$ and $\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}') = \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h})$. Because $(\mathfrak{s}, \mathfrak{h}) \in \mathbf{Models}^{\mathrm{g}}_\Phi$ by assumption, Theorem 12.10 then implies that $(\mathfrak{s}, \mathfrak{h}) \models_\Phi \phi$. □

12.2 COMPUTING THE TYPES OF PREDICATE CALLS

12.2.1 *Computing on Sets of Types*

In the remainder of this chapter, we will frequently compute on sets of types. To simplify notation, we lift $\bullet$, $\cdot[\cdot/\cdot]$, and forget from types to sets of types in a point-wise manner, i.e.,

$$\begin{aligned}
&\{\mathcal{T}_1, \ldots, \mathcal{T}_m\} \bullet \{\mathcal{T}'_1, \ldots, \mathcal{T}'_n\} \\
&\quad \triangleq \left\{\mathcal{T}_i \bullet \mathcal{T}'_j \mid 1 \leq i \leq m, 1 \leq j \leq n, \mathcal{T}_i \bullet \mathcal{T}'_j \neq \bot\right\}, \\
&\{\mathcal{T}_1, \ldots, \mathcal{T}_m\}[\mathbf{x}/\mathbf{y}] \triangleq \{\mathcal{T}_1[\mathbf{x}/\mathbf{y}], \ldots, \mathcal{T}_m[\mathbf{x}/\mathbf{y}]\} \text{ and} \\
&\mathsf{forget}_{\Sigma,y}(\{\mathcal{T}_1, \ldots, \mathcal{T}_m\}) \triangleq \left\{\mathsf{forget}_{\Sigma,y}(\mathcal{T}_1), \ldots, \mathsf{forget}_{\Sigma,y}(\mathcal{T}_m)\right\}
\end{aligned}$$

Note that the lifted $\bullet$ operator can be used to compute the types of $\phi_1 \star \phi_2$ from the types of ϕ_1 and ϕ_2. This is formalized in the following lemma.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Lemma 12.13 (Composition of type sets). *Let $\phi_1, \phi_2 \in \mathbf{SLID}^{g}_{btw}$. Then $\mathbf{Types}^{\Sigma}_{\Phi}(\phi_1 \star \phi_2) = \mathbf{Types}^{\Sigma}_{\Phi}(\phi_1) \bullet \mathbf{Types}^{\Sigma}_{\Phi}(\phi_2)$.*

Proof. We show each inclusion separately.

- Let $\mathcal{T} \in \mathbf{Types}^{\Sigma}_{\Phi}(\phi_1 \star \phi_2)$. Let $\mathfrak{h}$ be such that $(\mathfrak{s}, \mathfrak{h}) \models_{\Phi} \phi_1 \star \phi_2$ and $\mathcal{T} = \mathsf{type}_{\Phi}(\mathfrak{s}, \mathfrak{h})$. By Corollary 8.13, there then exist heaps $\mathfrak{h}_1, \mathfrak{h}_2$ such that $(\mathfrak{s}, \mathfrak{h}_1), (\mathfrak{s}, \mathfrak{h}_2) \in \mathbf{Models}^{g}_{\Phi}$, $(\mathfrak{s}, \mathfrak{h}_i) \models_{\Phi} \phi_i$ and $\mathfrak{h} = \mathfrak{h}_1 + \mathfrak{h}_2$. Corollary 11.32 yields $\mathcal{T} = \mathsf{type}_{\Phi}(\mathfrak{s}, \mathfrak{h}_1) \bullet \mathsf{type}_{\Phi}(\mathfrak{s}, \mathfrak{h}_2)$. Since $(\mathfrak{s}, \mathfrak{h}_i) \models_{\Phi} \phi_i$, it follows that $\mathsf{type}_{\Phi}(\mathfrak{s}, \mathfrak{h}_i) \in \mathbf{Types}^{\Sigma}_{\Phi}(\phi_i)$. This implies $\mathcal{T} \in \mathbf{Types}^{\Sigma}_{\Phi}(\phi_1) \bullet \mathbf{Types}^{\Sigma}_{\Phi}(\phi_2)$.

- Let $\mathcal{T} \in \mathbf{Types}^{\Sigma}_{\Phi}(\phi_1) \bullet \mathbf{Types}^{\Sigma}_{\Phi}(\phi_2)$. There exist $\mathcal{T}_1 \in \mathbf{Types}^{\Sigma}_{\Phi}(\phi_1)$ and $\mathcal{T}_2 \in \mathbf{Types}^{\Sigma}_{\Phi}(\phi_2)$ s.t. $\mathcal{T} = \mathcal{T}_1 \bullet \mathcal{T}_2$. As $\mathcal{T}_i \in \mathbf{Types}^{\Sigma}_{\Phi}(\phi_i)$, there are heaps $\mathfrak{h}_1, \mathfrak{h}_2$ such that $\mathsf{type}_{\Phi}(\mathfrak{s}, \mathfrak{h}_i) = \mathcal{T}_i$ and $(\mathfrak{s}, \mathfrak{h}_i) \models_{\Phi} \phi_i$. By the semantics of $\star$, $(\mathfrak{s}, \mathfrak{h}_1 + \mathfrak{h}_2) \models_{\Phi} \phi_1 \star \phi_2$. According to Lemma 8.9, $(\mathfrak{s}, \mathfrak{h}_1), (\mathfrak{s}, \mathfrak{h}_2) \in \mathbf{Models}^{g}_{\Phi}$. Further, it holds that $\mathcal{T} = \mathsf{type}_{\Phi}(\mathfrak{s}, \mathfrak{h}_1) \bullet \mathsf{type}_{\Phi}(\mathfrak{s}, \mathfrak{h}_2)$. It follows by Corollary 11.32 that $\mathcal{T} = \mathsf{type}_{\Phi}(\mathfrak{s}, \mathfrak{h}_1 + \mathfrak{h}_2)$. Therefore, $\mathcal{T} \in \mathbf{Types}^{\Sigma}_{\Phi}(\phi_1 \star \phi_2)$. □

12.2.2 *Interlude: Consequences of Establishment*

Recall from Section 8.3 that all SIDs in $\mathbf{ID}_{btw}$ have the *establishment* property. The algorithm for computing the Φ-types of predicates, which I present in Section 12.2.3, depends in nontrivial ways on this property.

First, we need that if (the location interpreting) an existentially-quantified variable is allocated in a recursive call, the variable must have been passed explicitly as parameter to the recursive call.

Lemma 12.14 (Allocated existentials must be parameters). *Let $\Phi \in \mathbf{ID}_{btw}$ and $(\mathsf{pred}(\mathbf{x}) \Leftarrow \phi) \in \Phi$, $\phi = \exists \mathbf{e}.\, \phi'$, $\phi' = (a \mapsto \mathbf{b}) \star \mathsf{pred}_1(\mathbf{z_1}) \star \cdots \star \mathsf{pred}_k(\mathbf{z_k}) \star \Pi$, where Π is a pure constraint, and assume $(\mathfrak{s}, \mathfrak{h}) \models_{\Phi} \phi$. Moreover, let $\mathfrak{s}' \supseteq \mathfrak{s}$ and $\mathfrak{h}_0, \ldots, \mathfrak{h}_k$ be such that*

- $\mathrm{dom}(\mathfrak{s}') = \mathrm{dom}(\mathfrak{s}) \cup \mathbf{e}$,
- $(\mathfrak{s}', \mathfrak{h}) \models_{\Phi} \phi'$,
- $\mathfrak{h} = \mathfrak{h}_0 + \cdots + \mathfrak{h}_k$,
- $(\mathfrak{s}', \mathfrak{h}_0) \models_{\Phi} a \mapsto \mathbf{b}$, *and*
- $(\mathfrak{s}', \mathfrak{h}_i) \models_{\Phi} \mathsf{pred}_i(\mathbf{z_i})$.

Finally, let $e \in \mathbf{e}$ such that $\mathfrak{s}(e) \in \mathrm{dom}(\mathfrak{h}_i)$. Then there exists a variable $v \in \mathbf{z_i}$ such that $\mathfrak{s}'(v) = \mathfrak{s}(e)$.

Proof. If there were no such variable, then we could replace $\mathfrak{h}_i$ with $\mathfrak{h}_i'$ such that $\mathfrak{s}(e) \notin \mathrm{dom}(\mathfrak{h}_i')$ but $(\mathfrak{s}', \mathfrak{h}_i') \models_{\Phi} \mathsf{pred}_i(\mathbf{z_i})$. We would then have

$$(\mathfrak{s}', \mathfrak{h}_0 + \cdots \mathfrak{h}_{i-1} + \mathfrak{h}_i' + \mathfrak{h}_{i+1} \cdots + \mathfrak{h}_k) \models_{\Phi} \phi$$

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

with $\mathfrak{s}(e) \notin \mathrm{dom}(\mathfrak{h}_0 + \cdots \mathfrak{h}_{i-1} + \mathfrak{h}'_i + \mathfrak{h}_{i+1} \cdots + \mathfrak{h}_k)$, contradicting establishment. □

Moreover, we exploit that establishment limits the way that locations can be shared between sub-heaps. Let $\mathfrak{t}$ be a Φ-tree with root predicate $\mathsf{pred}(\mathbf{v})$. Assume we split off a "child tree" $\mathfrak{t}_{\mathsf{sub}}$ with root predicate $\mathsf{pred}_{\mathsf{sub}}(\mathbf{w})$ from $\mathfrak{t}$, i.e., a tree $\mathfrak{t}_{\mathsf{sub}}$ with $\mathsf{root}(\mathfrak{t}_{\mathsf{sub}}) \in \mathsf{succ}_{\mathfrak{t}}(\mathsf{root}(\mathfrak{t}))$. Then the only locations that are shared between $\mathfrak{t}$ and $\mathfrak{t}_{\mathsf{sub}}$—that is, the only locations that can occur in a points-to assertion in both $\mathfrak{t}_{\mathsf{sub}}$ and $\mathfrak{t} \setminus \mathfrak{t}_{\mathsf{sub}}$—are $\mathbf{v} \cup \mathbf{w}$. Put differently, all shared locations must occur explicitly as parameter of either the root of the "main" tree or the root of the sub-tree. This is captured for arbitrary sub-trees (as opposed to child trees) by the following lemma.

Lemma 12.15 (Locations shared between sub-trees). *Let $\mathfrak{t}$ be a Φ-tree with $\mathsf{rootpred}(\mathfrak{t}) = \mathsf{pred}(\mathbf{v})$ and $\mathsf{allholes}(\mathfrak{t}) = \emptyset$. Let $l \in \mathbf{Loc}$, let $\mathfrak{t}_{\mathsf{sub}}$ be the sub-tree of $\mathfrak{t}$ with root l and let $\mathfrak{t}_{\mathsf{rem}}$ be the remainder of $\mathfrak{t}$, i.e., $\mathsf{split}(\{\mathfrak{t}\}, \{l\}) = \{\mathfrak{t}_{\mathsf{sub}}, \mathfrak{t}_{\mathsf{rem}}\}$ and $\mathsf{root}(\mathfrak{t}_{\mathsf{sub}}) = l$. Assume $\mathsf{rootpred}(\mathfrak{t}_{\mathsf{sub}}) = \mathsf{pred}_{\mathsf{sub}}(\mathbf{w})$. Then $\mathsf{ptrlocs}(\mathfrak{t}_{\mathsf{sub}}) \cap \mathsf{ptrlocs}(\mathfrak{t}_{\mathsf{rem}}) \subseteq \mathbf{v} \cup \mathbf{w}$.*

Proof. Let $v \in \mathsf{ptrlocs}(\mathfrak{t}_{\mathsf{sub}}) \cap \mathsf{ptrlocs}(\mathfrak{t}_{\mathsf{rem}})$. As $v \in \mathsf{ptrlocs}(\mathfrak{t}_{\mathsf{sub}})$, it must either be allocated or dangling. We handle each case separately.

CASE v IS ALLOCATED IN $\mathfrak{t}_{\mathsf{sub}}$. Formally,

$$v \in \mathrm{dom}(\mathfrak{t}_{\mathsf{sub}}) = \mathrm{dom}(\mathsf{heap}(\mathfrak{t}_{\mathsf{sub}})).$$

Because we also have $v \in \mathsf{ptrlocs}(\mathfrak{t}_{\mathsf{rem}})$, v must be dangling in $\mathfrak{t}_{\mathsf{rem}}$, i.e.,

$$v \in \mathsf{dangling}(\mathsf{heap}(\mathfrak{t}_{\mathsf{rem}})) \subseteq \mathsf{dangling}(\mathsf{heap}(\mathfrak{t})) \cup \{l\}. \qquad (\dagger)$$

If $v = l$, then v is the root parameter of $\mathsf{pred}_{\mathsf{sub}}(\mathbf{w})$, so $v \in \mathbf{w} \subseteq \mathbf{v} \cup \mathbf{w}$.

If $v \neq l$, we recall that Lemma 10.22 guarantees that $\mathsf{heap}(\mathfrak{t}) \models_{\Phi} \mathsf{pred}(\mathbf{v})$. As all dangling locations of a model of $\mathsf{pred}(\mathbf{v})$ have to be included in $\mathbf{v}$ by Lemma 8.10, it follows from $(\dagger)$ that $v \in \mathbf{v} \subseteq \mathbf{v} \cup \mathbf{w}$.

CASE v IS DANGLING IN $\mathfrak{t}_{\mathsf{sub}}$. Formally, $v \in \mathsf{dangling}(\mathsf{heap}(\mathfrak{t}_{\mathsf{sub}}))$. As in the previous case, we obtain that $\mathsf{heap}(\mathfrak{t}_{\mathsf{sub}}) \models_{\Phi} \mathsf{pred}_{\mathsf{sub}}(\mathbf{w})$ by Lemma 10.22. Because all dangling locations of a model of $\mathsf{pred}_{\mathsf{sub}}(\mathbf{w})$ have to be included in $\mathbf{w}$ by Lemma 8.10, we have $v \in \mathbf{w} \subseteq \mathbf{v} \cup \mathbf{w}$. □

This has profound consequences on the type computation for predicate calls. Assume we would like to compute the $\mathbf{x}$-types of all predicates. Pick a rule

$$\mathsf{pred}(\mathsf{fvars}(\mathsf{pred})) \Leftarrow \exists \mathbf{y}.\, (a \mapsto \mathbf{b}) \star \mathsf{pred}_1(\mathbf{z_1}) \star \cdots \star \mathsf{pred}_k(\mathbf{z_k}).$$

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

We begin by fixing some stack $\mathfrak{s}$ with $\mathrm{dom}(\mathfrak{s}) = \mathbf{x} \cup \mathsf{fvars}(\mathsf{pred})$. We extend this stack to a stack $\mathfrak{s}'$ with $\mathrm{dom}(\mathfrak{s}') = \mathrm{dom}(\mathfrak{s}) \cup \mathbf{y}$ and are left with computing the types of

$$(a \mapsto \mathbf{b}) \star \mathsf{pred}_1(\mathbf{z_1}) \star \cdots \star \mathsf{pred}_k(\mathbf{z_k}).$$

At a first glance, this implies recursively computing the types of the calls, $\mathsf{pred}_i(\mathbf{z_i})$, w.r.t. the variables $\mathbf{x}' \triangleq \mathbf{x} \cup \mathsf{fvars}(\mathsf{pred}) \cup \mathbf{y}$. We attempt to do so by picking a rule of pred_i, say

$$\mathsf{pred}_i(\mathsf{fvars}(\mathsf{pred}_i)) \Leftarrow \exists \mathbf{y}'.\, (a' \mapsto \mathbf{b}') \star \mathsf{pred}'_1(\mathbf{z}'_1) \star \cdots \star \mathsf{pred}'_{k'}(\mathbf{z}'_{\mathbf{k}'}).$$

We start from a stack $\mathfrak{s}_i$ with $\mathrm{dom}(\mathfrak{s}_i) = \mathbf{x}' \cup \mathsf{fvars}(\mathsf{pred}_i)$, extend it to a stack $\mathfrak{s}'_i$ with $\mathrm{dom}(\mathfrak{s}'_i) = \mathbf{x}' \cup \mathsf{fvars}(\mathsf{pred}_i) \cup \mathbf{y}'$. We then have to compute the types of the pred'_j w.r.t. $\mathbf{x}'' \triangleq \mathbf{x}' \cup \mathsf{fvars}(\mathsf{pred}_i) \cup \mathbf{y}'$.

You see where this is going: the computation diverges as we have to enlarge the set $\mathbf{x}$ again and again.

We can avoid this divergence by using Lemma 12.15. Recall that every type is the type of a model; and that every model of a predicate call corresponds to a tree without holes by Lemma 10.5. Computing the types of a predicate pred thus corresponds to computing the types of all Φ-trees with root predicate pred and without holes. Computing the types of $\mathsf{pred}(\mathsf{fvars}(\mathsf{pred}))$ in terms of the types of the recursive calls $\mathsf{pred}_1(\mathbf{z_1}), \ldots, \mathsf{pred}_k(\mathbf{z_k})$ thus corresponds to composing the types of sub-trees. Lemma 12.15 shows that not all locations can be shared between the sub-trees, so we do not actually have to compute the types of pred_i w.r.t. the entire set of variables $\mathbf{x}'$. Likewise, we do not need to compute the types of pred'_j w.r.t. all variables in $\mathbf{x}''$. Instead, we restrict these sets of variables to those variables whose interpretation can actually occur in the models of pred_i and pred'_j. For example, we restrict $\mathbf{x}'$ to the variables

$$\{x \mid \mathfrak{s}'(x) \text{ can occur in the model of } \mathsf{pred}_i(\mathbf{z_i})\}.$$

The first step towards this restriction is the following lemma. Like Lemma 12.15, we show the lemma for arbitrary sub-trees, as the proof does not depend on $\mathfrak{t}_{\mathsf{sub}}$ being a "child tree" of $\mathfrak{t}$.

Lemma 12.16. *Let* $(\mathsf{pred}(\mathsf{fvars}(\mathsf{pred})) \Leftarrow \exists \mathbf{y}.\, \psi) \in \Phi$. *Let* $\mathbf{x}$ *be a set of variables and let* $\mathfrak{s}$ *be a stack with* $\mathrm{dom}(\mathfrak{s}) = \mathbf{x} \cup \mathsf{fvars}(\mathsf{pred})$, $\mathfrak{s}(\mathbf{x}) \supseteq \mathbf{v}$ *and* $\mathfrak{s}(\mathsf{fvars}(\mathsf{pred})) = \mathbf{v}$. *Assume* $\mathfrak{t}$ *is a* Φ*-tree with* $\mathsf{allholes}(\mathfrak{t}) = \emptyset$ *such that* $\mathsf{rule}_{\mathfrak{t}}(\mathsf{root}(\mathfrak{t}))$ *is an instance of the above rule and let* $\mathbf{u}$ *be the sequence of locations by which the existentially-quantified variables* $\mathbf{y}$ *were instantiated in the rule instance. Let* $\mathfrak{s}' \triangleq \mathfrak{s} \cup \{\mathbf{y} \mapsto \mathbf{u}\}$ *be the corresponding stack extension. Finally, let* $\mathfrak{t}_{\mathsf{sub}}$ *be a sub-tree of* $\mathfrak{t}$ *with* $\mathsf{rootpred}(\mathfrak{t}_{\mathsf{sub}}) = \mathsf{pred}_{\mathsf{sub}}(\mathbf{w})$. *Then* $\mathsf{ptrlocs}(\mathfrak{t}_{\mathsf{sub}}) \cap \mathrm{img}(\mathfrak{s}') \subseteq \mathfrak{s}'(\mathbf{x}) \cup \mathbf{w}$.

Proof. Let $\mathfrak{t}_{\mathsf{rem}}$ be the unique Φ-tree with $\mathsf{split}(\{\mathfrak{t}\}, \{l\}) = \{\mathfrak{t}_{\mathsf{sub}}, \mathfrak{t}_{\mathsf{rem}}\}$. By Lemma 12.15, $\mathsf{ptrlocs}(\mathfrak{t}_{\mathsf{sub}}) \cap \mathsf{ptrlocs}(\mathfrak{t}_{\mathsf{rem}}) \subseteq \mathbf{v} \cup \mathbf{w}$. By assumption, $\mathbf{v} \subseteq \mathfrak{s}(\mathbf{x}) = \mathfrak{s}'(\mathbf{x})$, so

$$\mathsf{ptrlocs}(\mathfrak{t}_{\mathsf{sub}}) \cap \mathsf{ptrlocs}(\mathfrak{t}_{\mathsf{rem}}) \subseteq \mathfrak{s}'(\mathbf{x}) \cup \mathbf{w}. \qquad (\dagger)$$

TU Bibliothek, Your knowledge hub, WIEN
Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.

By establishment, every variable in $\mathbf{y}$ is eventually allocated. Because $\mathsf{allholes}(\mathfrak{t}) = \emptyset$, we thus have $\mathfrak{s}(\mathbf{y}) \subseteq \mathrm{dom}(\mathfrak{t})$.

Let $\mathbf{y}' \triangleq \{y \in \mathbf{y} \mid \mathfrak{s}'(y) \notin \mathbf{w}\}$. It follows from Lemma 12.14 that $\mathfrak{s}'(\mathbf{y}') \subseteq \mathrm{dom}(\mathfrak{t}_{\mathsf{rem}})$ and thus, in particular, $\mathfrak{s}'(\mathbf{y}') \subseteq \mathsf{ptrlocs}(\mathfrak{t}_{\mathsf{rem}})$. Combining this with (†), we obtain $\mathsf{ptrlocs}(\mathfrak{t}_{\mathsf{sub}}) \cap \mathfrak{s}'(\mathbf{y}') = \emptyset$. As $\mathrm{img}(\mathfrak{s}') = \mathfrak{s}'(\mathbf{x}) \cup \mathfrak{s}'(\mathsf{fvars}(\mathsf{pred})) \cup \mathbf{w} \cup \mathfrak{s}'(\mathbf{y}')$ and $\mathfrak{s}'(\mathsf{fvars}(\mathsf{pred})) = \mathbf{v} \subseteq \mathfrak{s}'(\mathbf{x})$ by assumption, we conclude $\mathsf{ptrlocs}(\mathfrak{t}_{\mathsf{sub}}) \cap \mathrm{img}(\mathfrak{s}') \subseteq \mathfrak{s}'(\mathbf{x}) \cup \mathbf{w}$. □

Let us revisit the divergence argument outlined before Lemma 12.16. In light of Lemma 12.16, we can avoid divergence: we start from a stack of size $|\mathbf{x}| \cup |\mathsf{fvars}(\mathsf{pred})|$ and perform the recursive type computation for pred_i for a stack of size $|\mathbf{x}| \cup |\mathsf{fvars}(\mathsf{pred}_i)|$, because only locations in $\mathfrak{s}(\mathbf{x})$ and the locations passed to pred_i can occur in the model of pred_i.

Lemma 12.16 is not sufficient to obtain a practical algorithm, because it only allows computing the types of pred w.r.t. $\mathbf{x} \cup \mathsf{fvars}(\mathsf{pred})$ in terms of the types of pred_i w.r.t. $\mathbf{x} \cup \mathbf{z_i}$, where $\mathbf{z_i}$ are the actual arguments of the call to pred_i, consisting of variables from $\mathsf{fvars}(\mathsf{pred})$ and some subset of existentially-quantified variables of size at most $\mathsf{fvars}(\mathsf{pred}_i)$. In other words, even though we are computing the types of pred w.r.t. the formal arguments of pred, the recursion needs access to the types of pred_i w.r.t. the actual arguments of pred_i.

Because of this dependence on the actual arguments as opposed to the formal parameters, we might still have to consider many different stack domains during type computation.

To avoid this, we apply the following trick. We compute the types of pred_i w.r.t. $\mathbf{x} \cup \mathsf{fvars}(\mathsf{pred}_i)$ (assuming w.l.o.g. that $\mathsf{fvars}(\mathsf{pred}_i) \cap \mathsf{fvars}(\mathsf{pred}) = \emptyset$) and then simply rename the variables to the actual arguments via $\cdot[\mathsf{fvars}(\mathsf{pred}_i)/\mathbf{z_i}]$. This is captured in the following corollary, which is identical to Lemma 12.16 apart from the definition of the stack $\mathfrak{s}_{\mathsf{sub}}$ that implements the trick.

Corollary 12.17. *Let* $(\mathsf{pred}(\mathsf{fvars}(\mathsf{pred})) \Leftarrow \exists \mathbf{y}.\, (a \mapsto \mathbf{b}) \star \mathsf{pred}_1(\mathbf{z_1}) \star \cdots \star \mathsf{pred}_k(\mathbf{z_k})) \in \Phi$. *Let* $\mathfrak{s}$ *be a stack with* $\mathrm{dom}(\mathfrak{s}) = \mathbf{x} \cup \mathsf{fvars}(\mathsf{pred})$ *with* $\mathfrak{s}(\mathbf{x}) \supseteq \mathbf{v}$ *and* $\mathfrak{s}(\mathsf{fvars}(\mathsf{pred})) = \mathbf{v}$. *Further, let* $\mathfrak{t}$ *be a* Φ*-tree with* $\mathsf{allholes}(\mathfrak{t}) = \emptyset$ *such that* $\mathsf{rule}_{\mathfrak{t}}(\mathsf{root}(\mathfrak{t}))$ *is an instance of the above rule and let* $\mathbf{u}$ *be the sequence of locations by which the existentially-quantified variables* $\mathbf{y}$ *were instantiated in the rule instance. Let* $\mathfrak{t}_{\mathsf{sub}}$ *be a sub-tree of* $\mathfrak{t}$ *with* $\mathsf{rootpred}(\mathfrak{t}_{\mathsf{sub}}) = \mathsf{pred}_{\mathsf{sub}}(\mathbf{w})$. *Finally, define*

$$\mathfrak{s}' \triangleq \mathfrak{s} \cup \{\mathbf{y} \mapsto \mathbf{u}\} \qquad \mathfrak{s}_{\mathsf{sub}} \triangleq \mathfrak{s}' \cup \{\mathsf{fvars}(\mathsf{pred}_{\mathsf{sub}}) \mapsto \mathbf{w}\}\,.$$

Then $\mathsf{ptrlocs}(\mathfrak{t}_{\mathsf{sub}}) \cap \mathrm{img}(\mathfrak{s}') \subseteq \mathrm{img}(\mathfrak{s}_{\mathsf{sub}})$.

Proof. By Lemma 12.16, $\mathsf{ptrlocs}(\mathfrak{t}_{\mathsf{sub}}) \cap \mathrm{img}(\mathfrak{s}') \subseteq \mathfrak{s}'(\mathbf{x}) \cup \mathbf{w}$. By construction, $\mathrm{img}(\mathfrak{s}_{\mathsf{sub}}) = \mathfrak{s}'(\mathbf{x}) \cup \mathbf{w}$. □

It is thus possible to compute the types of pred w.r.t. $\mathbf{x} \cup \mathsf{fvars}(\mathsf{pred})$ in terms of the types of pred_i w.r.t. $\mathbf{x} \cup \mathsf{fvars}(\mathsf{pred}_i)$. Our next goal is to develop an algorithm that does exactly that.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

12.2.3 *A Fixed-Point Algorithm for Computing the Types of Predicates*

For the remainder of this section, we fix a pointer-closed SID $\Phi \in \mathbf{ID}_{\mathsf{btw}}$ and a finite set of variables $\mathbf{x}$, for which we assume w.l.o.g. that $\mathbf{x} \cap \mathsf{fvars}(\mathsf{pred}) = \emptyset$ for all $\mathsf{pred} \in \mathbf{Preds}(\Phi)$.

Our goal in this section is to compute for every predicate $\mathsf{pred} \in \mathbf{Preds}(\Phi)$ the set of all $\mathbf{x}$-types of pred. Specifically, we will compute the set

$$\mathbf{Types}^{\Sigma}_{\Phi}(\mathsf{pred}) \triangleq \mathbf{Types}^{\Sigma}_{\Phi}(\mathsf{pred}(\mathsf{fvars}(\mathsf{pred}))).$$

for all aliasing constraints $\Sigma \in \mathbf{AC}^{\mathsf{fvars}(\mathsf{pred}) \cup \mathbf{x}}$.

Once we have a way to compute these types, we can also compute the types of all $\mathbf{SLID}^{\mathsf{g}}_{\mathsf{btw}}$ formulas with free variables $\mathbf{x}$, as we will see in Section 12.3.

We compute $\mathbf{Types}^{\Sigma}_{\Phi}(\mathsf{pred})$ for all choices of Σ and pred by a simultaneous fixed-point computation. Specifically, we compute a (partial) function

$$p\colon \mathbf{Preds}(\Phi) \times \mathbf{AC} \rightharpoonup 2^{\mathbf{Types}(\Phi)}$$

that maps every predicate pred and every $\Sigma \in \mathbf{AC}^{\mathbf{x} \cup \mathsf{fvars}(\mathsf{pred})}$ to the types $\mathbf{Types}^{\Sigma}_{\Phi}(\mathsf{pred})$. We start off the fixed-point computation with $p(\mathsf{pred}, \Sigma) = \emptyset$ for all predicates pred and aliasing constraints Σ; each iteration adds to p some more types such that after each iteration, $p(\mathsf{pred}, \Sigma)$ is a subset of $\mathbf{Types}^{\Sigma}_{\Phi}(\mathsf{pred})$; and when we reach the fixed point, $p(\mathsf{pred}, \Sigma) = \mathbf{Types}^{\Sigma}_{\Phi}(\mathsf{pred})$ holds for all pred and Σ.

Each iteration of the fixed-point computation consists in applying the function $\mathsf{ptypes}^{\mathbf{x}}_{p}(\phi, \Sigma)$ defined in Fig. 12.1 to all rule bodies $\phi \in \mathbf{SH}^{\exists}$ of the SID Φ and all stack-aliasing constraints Σ. Here, p is the pre-fixed point from the previous iteration.

In Fig. 12.1, we use the following two auxiliary definitions.

- $\mathsf{ptrmodel}_{\Sigma}(a \mapsto \mathbf{b})$ denotes an arbitrary stack–heap pair $(\mathfrak{s}, \mathfrak{h})$ with $\mathsf{aliasing}(\mathfrak{s}) = \Sigma$ and $(\mathfrak{s}, \mathfrak{h}) \models_{\Phi} a \mapsto \mathbf{b}$.

- We lift stack instantiation (Definition 11.33) to stack-aliasing constraints: Let Σ be a stack-aliasing constraint, let $\mathbf{x}$ be a permutation of $\mathrm{dom}(\Sigma)$, and let $\mathbf{y} \in \mathbf{Var}^{|\mathrm{dom}(\Sigma)|}$. If there exist stacks $\mathfrak{s}, \mathfrak{s}'$ such that (1) $\mathsf{aliasing}(\mathfrak{s}) = \Sigma$ and (2) $\mathfrak{s}'$ is the $[\mathbf{x}/\mathbf{y}]$-instantiation of $\mathfrak{s}$, then we define $\Sigma[\mathbf{x}/\mathbf{y}] \triangleq \mathsf{aliasing}(\mathfrak{s}')$. Otherwise $\Sigma[\mathbf{x}/\mathbf{y}] \triangleq \bot$. Note that $|\Sigma| = |\Sigma[\mathbf{x}/\mathbf{y}]|$, reflecting that the instantiation does not merge equivalence classes of Σ.

Informally, the function $\mathsf{ptypes}^{\mathbf{x}}_{p}(\phi, \Sigma)$ works as follows.

- Recall that according to our semantics, (dis)equalities only hold in the empty heap. Consequently, if ϕ is a (dis)equality, we look up in the aliasing constraint whether the (dis)equality holds and return either the type of the empty model or no type.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

$$
\begin{aligned}
\mathsf{ptypes}_p^{\mathbf{x}}(x \approx y, \Sigma) &\triangleq \text{if } \langle x, y\rangle \in \Sigma \text{ then } \{\{\mathbf{emp}\}\} \text{ else } \emptyset \\
\mathsf{ptypes}_p^{\mathbf{x}}(x \not\approx y, \Sigma) &\triangleq \text{if } \langle x, y\rangle \notin \Sigma \text{ then } \{\{\mathbf{emp}\}\} \text{ else } \emptyset \\
\mathsf{ptypes}_p^{\mathbf{x}}(a \mapsto \mathbf{b}, \Sigma) &\triangleq \{\mathsf{type}_\Phi(\mathsf{ptrmodel}_\Sigma(a \mapsto \mathbf{b}))\} \\
\mathsf{ptypes}_p^{\mathbf{x}}(\mathsf{pred}(\mathbf{y}), \Sigma) &\triangleq \text{let } \mathbf{C} \triangleq \Big\{\Sigma' \in \mathbf{AC}^{\mathbf{x} \cup \mathsf{fvars}(\mathsf{pred})} \mid \\
&\qquad\qquad \Sigma'[\mathsf{fvars}(\mathsf{pred})/\mathbf{y}] = \Sigma|_{\mathbf{x}\cup\mathbf{y}} \Big\} \\
&\quad\ \text{in } \bigcup \{p(\mathsf{pred}, \Sigma')[\mathsf{fvars}(\mathsf{pred})/\mathbf{y}] \mid \Sigma' \in \mathbf{C}\} \\
\mathsf{ptypes}_p^{\mathbf{x}}(\phi_1 \star \phi_2, \Sigma) &\triangleq \mathsf{ptypes}_p^{\mathbf{x}}(\phi_1, \Sigma) \bullet \mathsf{ptypes}_p^{\mathbf{x}}(\phi_2, \Sigma) \\
\mathsf{ptypes}_p^{\mathbf{x}}(\exists y.\, \phi, \Sigma) &\triangleq \text{let } \mathbf{z} \triangleq \mathrm{dom}(\Sigma), \\
&\qquad \mathbf{C} \triangleq \Big\{\Sigma' \in \mathbf{AC}^{\mathbf{z} \cup \{y\}} \mid \Sigma'|_{\mathbf{z}} = \Sigma\Big\} \\
&\quad\ \text{in } \bigcup \Big\{\mathsf{forget}_{\Sigma', y}(\mathsf{ptypes}_p^{\mathbf{x}}(\phi, \Sigma')) \mid \Sigma' \in \mathbf{C}\Big\}
\end{aligned}
$$

Figure 12.1: Computing (a subset of) the Φ-types of existentially-quantified symbolic heap $\phi \in \mathbf{SH}^{\exists}$ for stacks with stack-aliasing constraint Σ under the assumption that p maps every predicate symbol pred and every stack-aliasing constraint to (a subset of) the types $\mathbf{Types}_\Phi^\Sigma(\mathsf{pred})$.

- If $\phi = a \mapsto \mathbf{b}$, there is up to isomorphism only one model with stack-aliasing constraint Σ that satisfies ϕ. We return the type of this model.
- If $\phi = \mathsf{pred}(\mathbf{y})$, we look up the types of $\mathsf{pred}(\mathsf{fvars}(\mathsf{pred}))$ in the pre-fixed point p and then rename the formal parameters $\mathsf{fvars}(\mathsf{pred})$ to the actual arguments $\mathbf{y}$.

  Crucially, we look up in p only types w.r.t. aliasing constraints over the variables $\mathbf{x} \cup \mathsf{fvars}(\mathsf{pred})$, which in general need not contain all variables in $\mathrm{dom}(\Sigma)$—in particular, if the predicate call occurs in scope of existential quantifiers.

  This restriction guarantees that the computation of $\mathsf{ptypes}_p^{\mathbf{x}}(\phi, \Sigma)$ does not diverge by considering larger and larger aliasing constraints in recursive calls. See Corollary 12.17 for a justification of the restriction.
- If $\phi = \phi_1 \star \phi_2$, we simply compose the types of the subformulas.
- If $\phi = \exists y.\, \phi'$, we consider all ways to extend the aliasing constraint Σ with y and recurse. Our treatment of predicate calls outlined above guarantees that this does not lead to divergence.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

We wrap ptypes in a fixed-point computation as follows.

$$\begin{aligned}
&\mathsf{unfold}_{\mathbf{x}} \colon (\mathbf{Preds}(\Phi) \times \mathbf{AC} \rightharpoonup 2^{\mathbf{Types}(\Phi)}) \\
&\qquad\qquad \to (\mathbf{Preds}(\Phi) \times \mathbf{AC} \rightharpoonup 2^{\mathbf{Types}(\Phi)}), \\
&\mathsf{unfold}_{\mathbf{x}}(p) = \lambda(\mathsf{pred}, \Sigma).\, p(\mathsf{pred}, \Sigma) \cup \bigcup_{(\mathsf{pred}(\mathbf{y}) \Leftarrow \phi) \in \Phi} \mathsf{ptypes}^{\mathbf{x}}_{p}(\Sigma, \phi), \\
&\mathsf{lfp}(\mathsf{unfold}_{\mathbf{x}}) \triangleq \lim_{n \in \mathbb{N}} \mathsf{unfold}^{n}_{\mathbf{x}}(\lambda(\mathsf{pred}', \Sigma').\, \emptyset).
\end{aligned}$$

CONVERGENCE. The fixed point will always be reached after finitely many iterations. To see this, we first recall our underlying ordering,

$$f \sqsubseteq g \triangleq \forall \mathsf{pred} \forall \Sigma.\, f(\mathsf{pred}, \Sigma) \subseteq g(\mathsf{pred}, \Sigma).$$

Note that

1. $\mathsf{unfold}^{n}_{\mathbf{x}}(\lambda(\mathsf{pred}', \Sigma').\, \emptyset) \sqsubseteq \mathsf{unfold}^{n+1}_{\mathbf{x}}(\lambda(\mathsf{pred}', \Sigma').\, \emptyset)$ for all n;
2. the domain of the computed functions is given by

   $$\left\{ \langle \mathsf{pred}, \Sigma \rangle \mid \mathsf{pred} \in \mathbf{Preds}(\Phi), \Sigma \in \mathbf{AC}^{\mathbf{x} \cup \mathsf{fvars}(\mathsf{pred})} \right\}$$

   and thus of a fixed, finite size; and
3. the image of the computed functions, $2^{\mathbf{Types}(\Phi)}$, is finite.

In other words, there are only finitely many functions that can be returned by an iteration of the function; and every iteration returns a larger function w.r.t. $\sqsubseteq$. Consequently, $\mathsf{lfp}(\mathsf{unfold}_{\mathbf{x}})$ is the least fixed point of $\mathsf{unfold}_{\mathbf{x}}$ and is reached after finitely many iterations.

CORRECTNESS. To show the correctness of the fixed-point computation, we proceed as follows.

1. We show $\mathsf{lfp}(\mathsf{unfold}_{\mathbf{x}})(\mathsf{pred}, \Sigma) \subseteq \mathbf{Types}^{\Sigma}_{\Phi}(\mathsf{pred})$ in Section 12.2.4.
2. We show $\mathsf{lfp}(\mathsf{unfold}_{\mathbf{x}})(\mathsf{pred}, \Sigma) \supseteq \mathbf{Types}^{\Sigma}_{\Phi}(\mathsf{pred})$ in Section 12.2.5.
3. We use these results and perform a complexity analysis in Section 12.2.6 to conclude that $\mathbf{Types}^{\Sigma}_{\Phi}(\mathsf{pred})$ is computable in double-exponential time.

12.2.4 *Soundness of the Type Computation*

We first show that $\mathsf{lfp}(\mathsf{unfold}_{\mathbf{x}})(\mathsf{pred}, \Sigma) \subseteq \mathbf{Types}^{\Sigma}_{\Phi}(\mathsf{pred})$. To show this, we first need to establish that $\mathsf{ptypes}^{\mathbf{x}}_{p}(\Sigma, \phi)$ is sound when ϕ is a rule of pred—i.e., that $\mathsf{ptypes}^{\mathbf{x}}_{p}(\Sigma, \phi) \subseteq \mathbf{Types}^{\Sigma}_{\Phi}(\phi)$—under the assumption that p maps every pair of predicate identifier and stack-aliasing constraint to a subset of the corresponding types. As SID rules are

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

guaranteed to be existentially-quantified symbolic heaps, it suffices to prove this result for arbitrary $\phi \in \mathbf{SH}^{\exists}$.

As preparation, we need a few simple lemmas characterizing the types of atomic formulas.

Lemma 12.18. *For all* Σ, $\mathbf{Types}^{\Sigma}_{\Phi}(\mathbf{emp}) = \{\{\mathbf{emp}\}\}$.

Proof. Let $(\mathfrak{s},\mathfrak{h})$ be a model with $\mathsf{type}_{\Phi}(\mathfrak{s},\mathfrak{h}) \in \mathbf{Types}^{\Sigma}_{\Phi}(\mathbf{emp})$ and $\mathsf{aliasing}(\mathfrak{s}) = \Sigma$. By definition, $(\mathfrak{s},\mathfrak{h}) \models_{\Phi} \mathbf{emp}$ and thus $\mathfrak{h} = \emptyset$, which in turn implies that $\mathsf{alloced}(\mathfrak{s},\mathfrak{h}) = \emptyset$. Moreover, $\mathbf{emp} \in \mathsf{type}_{\Phi}(\mathfrak{s},\mathfrak{h})$ because $\mathbf{emp} = \mathsf{project}(\mathfrak{s},\{\emptyset\})$ and $\emptyset \in \mathsf{forests}_{\Phi}(\mathfrak{h})$. Putting this together, we obtain $\mathsf{type}_{\Phi}(\mathfrak{s},\mathfrak{h}) = \langle \Sigma, F, \emptyset \rangle$ for some $F \supseteq \{\mathbf{emp}\}$.

Assume $\psi \in F$. By definition, there is a Φ-forest $\mathfrak{f} = \{\mathfrak{t}_1, \dots, \mathfrak{t}_k\}$ with $\mathsf{project}(\mathfrak{s},\mathfrak{f}) = \psi$ and $\mathsf{heap}(\mathfrak{f}) = \mathfrak{h}$. As $\mathfrak{h} = \emptyset$, we have $\mathsf{heap}(\mathfrak{f}) = \emptyset$ and thus $\mathsf{heap}(\mathfrak{t}_i) = \emptyset$ for all i. Hence, $k = 0$. By definition of stack–forest projection, we then have $\psi = \mathbf{emp}$. Therefore, $F \subseteq \{\mathbf{emp}\}$. Since $(\mathfrak{s},\mathfrak{h})$ was an arbitrary model of ϕ with $\mathsf{type}_{\Phi}(\mathfrak{s},\mathfrak{h}) \in \mathbf{Types}^{\Sigma}_{\Phi}(\mathbf{emp})$ and $\mathsf{aliasing}(\mathfrak{s}) = \Sigma$, we have

$$\mathbf{Types}^{\Sigma}_{\Phi}(\mathbf{emp}) = \{\{\mathbf{emp}\}\} = \mathsf{ptypes}^{\mathbf{x}}_{p}(\Sigma, \mathbf{emp}). \qquad \square$$

Lemma 12.19. *Let* Σ *be a stack-aliasing constraint and* $x, y \in \mathrm{dom}(\Sigma)$. *If* $(x,y) \in \Sigma$, *then* $\mathbf{Types}^{\Sigma}_{\Phi}(x \approx y) = \{\{\mathbf{emp}\}\}$ *and* $\mathbf{Types}^{\Sigma}_{\Phi}(x \not\approx y) = \emptyset$. *Otherwise,* $\mathbf{Types}^{\Sigma}_{\Phi}(x \not\approx y) = \{\{\mathbf{emp}\}\}$ *and* $\mathbf{Types}^{\Sigma}_{\Phi}(x \approx y) = \emptyset$.

Proof. We only consider the case $x \approx y$, as the argument for $x \not\approx y$ is completely analogous. If $\langle x, y\rangle \in \Sigma$, our semantics of equalities gives us that $\mathbf{Types}^{\Sigma}_{\Phi}(\mathbf{emp}) = \mathbf{Types}^{\Sigma}_{\Phi}(x \approx y)$. The claim then follows from Lemma 12.18. If $\langle x, y\rangle \notin \Sigma$, it holds for all $\mathfrak{s}$ with $\mathsf{aliasing}(\mathfrak{s}) = \Sigma$ that $\mathfrak{s}(x) \neq \mathfrak{s}(y)$. The semantics of $x \approx y$ then give us for all heaps $\mathfrak{h}$ that $(\mathfrak{s},\mathfrak{h}) \not\models_{\Phi} x \approx y$. Consequently, $\mathbf{Types}^{\Sigma}_{\Phi}(x \approx y) = \emptyset$. $\square$

Lemma 12.20. *Let* Σ *be a stack-aliasing constraint, let* $a \in \mathrm{dom}(\Sigma)$, *and let* $\mathbf{b} \in \mathbf{Var}^{*}$ *with* $\mathbf{b} \subseteq \mathrm{dom}(\Sigma)$. *Then*

$$\mathbf{Types}^{\Sigma}_{\Phi}(a \mapsto \mathbf{b}) = \{\mathsf{type}_{\Phi}(\mathsf{ptrmodel}_{\Sigma}(a \mapsto \mathbf{b}))\}\,.$$

Proof. "$\supseteq$" Let $(\mathfrak{s},\mathfrak{h}) \triangleq \mathsf{ptrmodel}_{\Sigma}(a \mapsto \mathbf{b})$ and $\mathcal{T} \triangleq \mathsf{type}_{\Phi}(\mathfrak{s},\mathfrak{h})$. By definition, $(\mathfrak{s},\mathfrak{h}) \models_{\Phi} a \mapsto \mathbf{b}$. Hence, $\mathcal{T} \in \mathbf{Types}^{\Sigma}_{\Phi}(a \mapsto \mathbf{b})$.

"$\subseteq$" Let $(\mathfrak{s},\mathfrak{h})$ be an arbitrary model with $\mathsf{type}_{\Phi}(\mathfrak{s},\mathfrak{h}) \in \mathbf{Types}^{\Sigma}_{\Phi}(a \mapsto \mathbf{b})$ and $\mathsf{aliasing}(\mathfrak{s}) = \Sigma$. By definition, $(\mathfrak{s},\mathfrak{h}) \models_{\Phi} a \mapsto \mathbf{b}$ and thus $\mathfrak{h} = \{\mathfrak{s}(a) \mapsto \mathfrak{s}(\mathbf{b})\}$ by the semantics of points-to assertions. Consequently, $(\mathfrak{s},\mathfrak{h}) \cong \mathsf{ptrmodel}_{\Sigma}(a \mapsto \mathbf{b})$ and

$$\mathsf{type}_{\Phi}(\mathfrak{s},\mathfrak{h}) = \mathsf{type}_{\Phi}(\mathsf{ptrmodel}_{\Sigma}(a \mapsto \mathbf{b})).$$

Since $(\mathfrak{s},\mathfrak{h})$ was an arbitrary model of ϕ with $\mathsf{type}_{\Phi}(\mathfrak{s},\mathfrak{h}) \in \mathbf{Types}^{\Sigma}_{\Phi}(a \mapsto \mathbf{b})$ and $\mathsf{aliasing}(\mathfrak{s}) = \Sigma$, we have $\mathbf{Types}^{\Sigma}_{\Phi}(a \mapsto \mathbf{b}) \subseteq \{\mathsf{type}_{\Phi}(\mathsf{ptrmodel}_{\Sigma}(a \mapsto \mathbf{b}))\}$. $\square$

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

We are now ready to prove the "relative soundness" of ptypes for arbitrary symbolic heaps: if the parameter p (which is the pre-fixed point from the previous iteration) maps predicates and aliasing constraints to a subset of the corresponding types, the same property holds for the function computed in the next iteration (which consists in evaluating ptypes for all predicates and aliasing constraints).

Lemma 12.21. *Let* $\phi \in \mathbf{SH}^{\exists}$ *and* $\Sigma \in \mathbf{AC}$ *with* $\mathsf{dom}(\Sigma) \supseteq \mathsf{fvars}(\phi)$. *Moreover, let*

$$p\colon \mathbf{Preds}(\Phi) \times \mathbf{AC} \rightharpoonup 2^{\mathbf{Types}(\Phi)}$$

be such that for all $\mathsf{pred} \in \mathbf{Preds}(\Phi)$ *and all* $\Sigma' \in \mathbf{AC}^{\mathbf{x} \cup \mathsf{fvars}(\mathsf{pred})}$, *it holds that* $p(\mathsf{pred}, \Sigma') \subseteq \mathbf{Types}_{\Phi}^{\Sigma'}(\mathsf{pred})$. *Then* $\mathsf{ptypes}_p^{\mathbf{x}}(\phi, \Sigma) \subseteq \mathbf{Types}_{\Phi}^{\Sigma}(\phi)$.

Proof. We proceed by induction on the structure of ϕ.

CASES $\phi = x \approx y$, $\phi = x \not\approx y$. The claim follows from Lemma 12.19.

CASE $\phi = a \mapsto \mathbf{b}$. The claim follows from Lemma 12.20.

CASE $\phi = \mathsf{pred}(\mathbf{y})$. Let $\mathcal{T} \in \mathsf{ptypes}_p^{\mathbf{x}}(\Sigma, \phi)$. By definition of ptypes, there is a stack-aliasing constraint $\Sigma' \in \mathbf{AC}^{\mathbf{x} \cup \mathsf{fvars}(\mathsf{pred})}$ such that $\Sigma'[\mathsf{fvars}(\mathsf{pred})/\mathbf{y}] = \Sigma|_{\mathbf{x} \cup \mathbf{y}}$ and $\mathcal{T} \in p(\mathsf{pred}, \Sigma')[\mathsf{fvars}(\mathsf{pred})/\mathbf{y}]$.

By assumption, $p(\mathsf{pred}, \Sigma') \subseteq \mathbf{Types}_{\Phi}^{\Sigma'}(\mathsf{pred})$, so there exists a $\mathcal{T}' \in \mathbf{Types}_{\Phi}^{\Sigma'}(\mathsf{pred})$ such that $\mathcal{T} \in \mathcal{T}'[\mathsf{fvars}(\mathsf{pred})/\mathbf{y}]$. By definition of $\mathbf{Types}_{\Phi}^{\Sigma'}(\mathsf{pred})$, there is model $(\mathfrak{s}', \mathfrak{h})$ with $\mathsf{type}_{\Phi}(\mathfrak{s}', \mathfrak{h}) = \mathcal{T}'$ and $(\mathfrak{s}', \mathfrak{h}) \models_{\Phi} \mathsf{pred}(\mathsf{fvars}(\mathsf{pred}))$.

Note that because $\Sigma'[\mathsf{fvars}(\mathsf{pred})/\mathbf{y}] \neq \bot$, we know that there is a stack $\mathfrak{s}$ such that $\mathfrak{s}$ is the $[\mathsf{fvars}(\mathsf{pred})/\mathbf{y}]$-instantiation of $\mathfrak{s}'$ and $\mathsf{aliasing}(\mathfrak{s}) = \Sigma'[\mathsf{fvars}(\mathsf{pred})/\mathbf{y}] = \Sigma|_{\mathbf{x} \cup \mathbf{y}}$.

By Lemma 11.34, $(\mathfrak{s}, \mathfrak{h}) \models_{\Phi} \mathsf{pred}(\mathbf{y})$ (†).

By Lemma 11.36, $\mathcal{T} = \mathsf{type}_{\Phi}(\mathfrak{s}', \mathfrak{h})[\mathsf{fvars}(\mathsf{pred})/\mathbf{y}] = \mathsf{type}_{\Phi}(\mathfrak{s}, \mathfrak{h})$. Together with (†), this implies that $\mathcal{T} \in \mathbf{Types}_{\Phi}^{\Sigma|_{\mathbf{x} \cup \mathbf{y}}}(\mathsf{pred}(\mathbf{y}))$. Consequently, it also holds that $\mathcal{T} \in \mathbf{Types}_{\Phi}^{\Sigma}(\mathsf{pred}(\mathbf{y}))$.[1]

CASE $\phi = \phi_1 \star \phi_2$. Let $\mathcal{T} \in \mathsf{ptypes}_p^{\mathbf{x}}(\Sigma, \phi_1 \star \phi_2)$. By definition, $\mathcal{T} \in \mathsf{ptypes}_p^{\mathbf{x}}(\Sigma, \phi_1) \bullet \mathsf{ptypes}_p^{\mathbf{x}}(\Sigma, \phi_2)$. From the induction hypotheses for ϕ_1 and ϕ_2, we obtain

$$\mathcal{T} \in \mathbf{Types}_{\Phi}^{\Sigma}(\phi_1) \bullet \mathbf{Types}_{\Phi}^{\Sigma}(\phi_2).$$

By Lemma 12.13, $\mathcal{T} \in \mathbf{Types}_{\Phi}^{\Sigma}(\phi_1 \star \phi_2)$.

1 This, in fact, only holds up to normalization of variable names. It might be the case that there exists a variable z with $\max{[z]_{=}^{\Sigma}} > \max{[z]_{=}^{\Sigma|_{\mathbf{x} \cup \mathbf{y}}}}$, in which case some variable names in the formulas in $\mathcal{T}$ might be different than the variable names used in $\mathbf{Types}_{\Phi}^{\Sigma}(\mathsf{pred}(\mathbf{y}))$. This can be avoided easily by a normalization step similar to Definition 11.35, which I have not included in Fig. 12.1 to reduce clutter.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

CASE $\phi = \exists y.\, \phi$. Let $\mathcal{T} \in \mathsf{ptypes}^{\mathbf{x}}_p(\Sigma, \exists y.\, \phi)$. By definition, there exist a stack-aliasing constraint $\Sigma' \in \mathbf{AC}^{\mathrm{dom}(\Sigma)\cup\{y\}}$ with $\Sigma'|_{\mathrm{dom}(\Sigma)} = \Sigma$ and a type $\mathcal{T}' \in \mathsf{ptypes}^{\mathbf{x}}_p(\phi, \Sigma')$ such that $\mathcal{T} = \mathsf{forget}_{\Sigma',y}(\mathcal{T}')$. By the induction hypothesis, we have that $\mathcal{T}' \in \mathbf{Types}^{\Sigma'}_{\Phi}(\phi)$. Consequently, there exists a model $(\mathfrak{s}, \mathfrak{h})$ with $(\mathfrak{s}, \mathfrak{h}) \models_{\Phi} \phi$ and $\mathsf{type}_{\Phi}(\mathfrak{s}, \mathfrak{h}) = \mathcal{T}'$.

Let $\mathfrak{s}'$ be the restriction of $\mathfrak{s}$ to $\mathrm{dom}(\mathfrak{s}) \setminus \{y\}$. Note that establishment guarantees that $\mathfrak{s}(y) \in \mathsf{locs}(\mathfrak{h})$. We can thus apply Lemma 11.38 and obtain

$$\mathcal{T} = \mathsf{type}_{\Phi}(\mathfrak{s}', \mathfrak{h}) = \mathsf{forget}_{\Sigma',y}(\mathsf{type}_{\Phi}(\mathfrak{s}, \mathfrak{h})) = \mathsf{forget}_{\Sigma',y}(\mathcal{T}').$$

Moreover, because $(\mathfrak{s}, \mathfrak{h}) \models_{\Phi} \phi$, we have by Lemma 8.3 that $(\mathfrak{s}', \mathfrak{h}) \models_{\Phi} \exists y.\, \phi$. Consequently, $\mathcal{T} \in \mathbf{Types}^{\Sigma}_{\Phi}(\exists y.\, \phi)$. ☐

Theorem 12.22. $\mathsf{lfp}(\mathsf{unfold}_{\mathbf{x}})(\mathsf{pred}, \Sigma) \subseteq \mathbf{Types}^{\Sigma}_{\Phi}(\mathsf{pred})$.

Proof. A straightforward induction on top of Lemma 12.21. ☐

12.2.5 *Completeness of the Type Computation*

Our next goal is to prove that $\mathsf{lfp}(\mathsf{unfold}_{\mathbf{x}})(\mathsf{pred}, \Sigma) \supseteq \mathbf{Types}^{\Sigma}_{\Phi}(\mathsf{pred})$. The main challenge is to show the completeness of our treatment of predicate calls: we need to show that ptypes discovers all types even though $\mathsf{ptypes}^{\mathbf{x}}_p(\mathsf{pred}(\mathbf{y}), \Sigma)$ restricts the stack-aliasing constraint Σ to $\mathbf{x} \cup \mathbf{y}$. To this end, we will use the consequences of the establishment property derived in Section 12.2.2.

Before we continue, recall that throughout this section, we assume $\mathbf{x} \cap \bigcup_{\mathsf{pred} \in \mathbf{Preds}(\Phi)} \mathsf{fvars}(\mathsf{pred}) = \emptyset$.

We will show that the fixed-point algorithm discovers for all predicates pred and all $\Sigma \in \mathbf{AC}^{\mathbf{x}\cup\mathsf{fvars}(\mathsf{pred})}$ all the types

$$\begin{aligned}\{\mathsf{type}_{\Phi}(\mathfrak{s}, \mathsf{heap}(\mathfrak{t})) \mid\ & \mathsf{aliasing}(\mathfrak{s}) = \Sigma,\\ & \mathsf{rootpred}(\mathfrak{t}) = \mathsf{pred}(\mathfrak{s}(\mathsf{fvars}(\mathsf{pred}))),\\ & \mathsf{allholes}(\mathfrak{t}) = \emptyset\}.\end{aligned}$$

This set contains all Σ-types of $\mathsf{pred}(\mathsf{fvars}(\mathsf{pred}))$ because every model corresponds to at least one Φ-tree by Lemma 10.5.

Lemma 12.23. *Let* $\mathfrak{s}$ *be a stack with* $\mathrm{dom}(\mathfrak{s}) = \mathbf{x} \cup \mathsf{fvars}(\mathsf{pred})$ *and* $\mathfrak{s}(\mathsf{fvars}(\mathsf{pred})) = \mathbf{v}$ *and let* $\mathfrak{t}$ *be a* Φ*-tree with* $\mathsf{rootpred}(\mathfrak{t}) = \mathsf{pred}(\mathbf{v})$ *and* $\mathsf{allholes}(\mathfrak{t}) = \emptyset$*. Then*

$$\mathsf{type}_{\Phi}(\mathfrak{s}, \mathsf{heap}(\mathfrak{t})) \in \mathsf{unfold}^{\mathsf{height}(\mathfrak{t})+1}_{\mathbf{x}}(\lambda(\mathsf{pred}', \Sigma').\, \emptyset)(\mathsf{pred}, \mathsf{aliasing}(\mathfrak{s})).$$

Proof. Define $\mathfrak{h} \triangleq \mathsf{heap}(\mathfrak{t})$, $r \triangleq \mathsf{root}(\mathfrak{t})$, and $\Sigma \triangleq \mathsf{aliasing}(\mathfrak{s})$. We prove the claim by induction on $\mathsf{height}(\mathfrak{t})$.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Induction base: $\mathsf{height}(\mathfrak{t}) = 0$. Because $\mathsf{allholes}(\mathfrak{t}) = \emptyset$, there exists a non-recursive rule[2]

$$(\mathsf{pred}(\mathsf{fvars}(\mathsf{pred})) \Leftarrow (y \mapsto \mathbf{z}) \star \Pi) \in \Phi$$

such that r is labeled with an instance of this rule, i.e., such that there exist a location a, a sequence of locations $\mathbf{b}$ and a (trivial) pure constraint Π' such that

$$\begin{aligned}
&\mathsf{rule}_{\mathfrak{t}}(r) = \mathsf{pred}(\mathbf{v}) \Leftarrow (a \mapsto \mathbf{b}) \star \Pi', \\
&((y \mapsto \mathbf{z}) \star \Pi)[\mathsf{fvars}(\mathsf{pred})/\mathbf{v}] = (a \mapsto \mathbf{b}) \star \Pi' \text{ and} \\
&\mathfrak{h} = \{a \mapsto \mathbf{b}\}.
\end{aligned}$$

By construction, we then have $(\mathfrak{s}, \mathfrak{h}) \models_\Phi y \mapsto \mathbf{z}$, and thus

$$(\mathfrak{s}, \mathfrak{h}) \cong \mathsf{ptrmodel}_\Sigma(y \mapsto \mathbf{z}).$$

Consequently,

$$\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}) \in \mathsf{ptypes}_p^{\mathbf{x}}(y \mapsto \mathbf{z}, \Sigma)$$

for $p = \lambda(\mathsf{pred}', \Sigma').\emptyset$. Then,

$$\begin{aligned}
&\mathsf{unfold}_{\mathbf{x}}^{\mathsf{height}(\mathfrak{t})+1}(\lambda(\mathsf{pred}', \Sigma').\emptyset)(\mathsf{pred}, \Sigma) \\
=&\mathsf{unfold}_{\mathbf{x}}(\lambda(\mathsf{pred}', \Sigma').\emptyset)(\mathsf{pred}, \Sigma) \\
=&\bigcup_{(\mathsf{pred}(\mathsf{fvars}(\mathsf{pred}))\Leftarrow\phi)\in\Phi} \mathsf{ptypes}^{\mathbf{x}}_{\lambda(\mathsf{pred}',\Sigma').\emptyset}(\Sigma, \phi) \\
\supseteq&\mathsf{ptypes}_p^{\mathbf{x}}((y \mapsto \mathbf{z}) \star \Pi, \Sigma) \\
=&\mathsf{ptypes}_p^{\mathbf{x}}(y \mapsto \mathbf{z}, \Sigma) \\
\supseteq&\{\mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h})\}.
\end{aligned}$$

Induction step: $\mathsf{height}(\mathfrak{t}) \geq 1$. There is a recursive rule $(\mathsf{pred}(\mathbf{x}) \Leftarrow \phi) \in \Phi$, for some $\phi = \exists \mathbf{e}.\, \phi'$, $\phi' = (y \mapsto \mathbf{z}) \star \mathsf{pred}_1(\mathbf{z_1}) \star \cdots \star \mathsf{pred}_k(\mathbf{z_k}) \star \Pi$, Π pure, such that

1. $(\mathfrak{s}, \mathfrak{h}) \models_\Phi \phi$, and
2. $\mathsf{rule}_{\mathfrak{t}}(r) = \mathsf{pred}(\mathbf{v}) \Leftarrow \phi'[\mathsf{fvars}(\mathsf{pred}) \cdot \mathbf{e}/\mathbf{v} \cdot \mathbf{m}]$ for some $\mathbf{m} \in \mathbf{Loc}^*$, i.e., the root of $\mathfrak{t}$ is labeled with an instance of the rule.

Let $\mathfrak{s}' \supseteq \mathfrak{s}$ be a stack with $\mathrm{dom}(\mathfrak{s}') = \mathrm{dom}(\mathfrak{s}) \cup \mathbf{e}$ and $(\mathfrak{s}', \mathfrak{h}) \models_\Phi \phi'$; such a stack exists by Lemma 8.3. (Assuming w.l.o.g. that $\mathbf{e} \cap \mathrm{dom}(\mathfrak{s}) = \emptyset$.) By the semantics of $\star$ and Corollary 8.13, there exist $\mathfrak{h}_0, \ldots, \mathfrak{h}_k$ such that the models $(\mathfrak{s}, \mathfrak{h}_i)$ are guarded, $\mathfrak{h} = \mathfrak{h}_0 + \cdots + \mathfrak{h}_k$, $(\mathfrak{s}', \mathfrak{h}_0) \models_\Phi y \mapsto \mathbf{z}$, and $(\mathfrak{s}', \mathfrak{h}_i) \models_\Phi \mathsf{pred}_i(\mathbf{z_i})$ for $i \geq 1$.

2 Recall from Section 8.3 that we assume w.l.o.g. that non-recursive rules do not contain existentials.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

We define

$$\Sigma' \triangleq \mathsf{aliasing}(\mathfrak{s}')$$
$$\mathcal{T}_0 \triangleq \mathsf{type}_\Phi(\mathsf{ptrmodel}_{\Sigma'}(y \mapsto \mathbf{z}))$$
$$\mathcal{T}_i \triangleq \mathsf{type}_\Phi(\mathfrak{s}', \mathfrak{h}_i), i \geq 1$$
$$\langle s_1, \ldots, s_k \rangle \triangleq \mathsf{succ}_\mathfrak{t}(t)$$

Let $\mathfrak{t}_i$ be the sub-tree of $\mathfrak{t}$ with root s_i and let $\mathbf{v_i}$ be a sequence of locations such that $\mathsf{rootpred}(\mathfrak{t}_i) = \mathsf{pred}_i(\mathbf{v_i})$.

For $1 \leq i \leq k$, let $\mathfrak{s}_i \triangleq \mathfrak{s} \cup \{\mathsf{fvars}(\mathsf{pred}_i) \mapsto \mathfrak{s}'(\mathbf{z_i})\}$. Define $\mathcal{T}_i' \triangleq \mathsf{type}_\Phi(\mathfrak{s}_i, \mathfrak{h}_i)$ and $\Sigma_i \triangleq \mathsf{aliasing}(\mathfrak{s}_i)$. Corollary 12.17 then gives us that $\mathsf{ptrlocs}(\mathfrak{t}_i) \cap \mathrm{img}(\mathfrak{s}') \subseteq \mathrm{img}(\mathfrak{s}_i) = \mathfrak{s}'(\mathbf{x}) \cup \mathfrak{s}'(\mathbf{z_i})$. By construction, this is equivalent to

$$\mathsf{ptrlocs}(\mathfrak{t}_i) \cap \mathrm{img}(\mathfrak{s}') \subseteq \mathfrak{s}'(\mathbf{x}) \cup \mathfrak{s}'(\mathbf{z_i}). \tag{$\dagger$}$$

Let $\mathfrak{s}''$ be the restriction of $\mathfrak{s}'$ to $\mathbf{x} \cup \mathbf{z_i}$.

Observe that $\mathfrak{s}''$ is the $[\mathsf{fvars}(\mathsf{pred}_i)/\mathbf{z_i}]$-instantiation of $\mathfrak{s}_i$. It follows by Lemma 11.36 that $\mathcal{T}_i'[\mathsf{fvars}(\mathsf{pred}_i)/\mathbf{z_i}] = \mathsf{type}_\Phi(\mathfrak{s}'', \mathfrak{h}_i)$. Moreover, because only the locations in $\mathrm{img}(\mathfrak{s})$ can actually occur in $\mathfrak{h}_i$ by ($\dagger$), we have $\mathsf{type}_\Phi(\mathfrak{s}'', \mathfrak{h}_i) = \mathsf{type}_\Phi(\mathfrak{s}, \mathfrak{h}_i)$. Putting this together, we obtain

$$\mathcal{T}_i = \mathcal{T}_i'[\mathsf{fvars}(\mathsf{pred}_i)/\mathbf{z_i}]. \tag{$\ddagger$}$$

Finally, note that $\mathsf{height}(\mathfrak{t}_i) < \mathsf{height}(\mathfrak{t})$. Consequently, we can apply the induction hypothesis to obtain that

$$\begin{aligned} \mathcal{T}_i' &= \mathsf{type}_\Phi(\mathfrak{s}_i, \mathfrak{h}_i) \\ &\in \mathsf{unfold}_\mathbf{x}^{\mathsf{height}(\mathfrak{t}_i)+1}(\lambda(\mathsf{pred}', \Sigma').\emptyset)(\mathsf{pred}_i, \mathsf{aliasing}(\mathfrak{s}_i)) \\ &\sqsubseteq \mathsf{unfold}_\mathbf{x}^{\mathsf{height}(\mathfrak{t})}(\lambda(\mathsf{pred}', \Sigma').\emptyset)(\mathsf{pred}_i, \Sigma_i). \end{aligned} \tag{$\clubsuit$}$$

We are now in a position to prove the claim. Let

$$p \triangleq \mathsf{unfold}_\mathbf{x}^{\mathsf{height}(\mathfrak{t})}(\lambda(\mathsf{pred}', \Sigma').\emptyset) \text{ and}$$
$$\Sigma_i \triangleq \mathsf{aliasing}(\mathfrak{s}_i).$$

Then,

$$\begin{aligned} &\mathsf{unfold}_\mathbf{x}^{\mathsf{height}(\mathfrak{t})+1}(\lambda(\mathsf{pred}', \Sigma').\emptyset)(\mathsf{pred}, \Sigma) \\ &= \bigcup_{(\mathsf{pred}(\mathsf{fvars}(\mathsf{pred})) \Leftarrow \phi) \in \Phi} \mathsf{ptypes}_p^\mathbf{x}(\Sigma, \phi) \\ &\supseteq \mathsf{ptypes}_p^\mathbf{x}(\Sigma, \exists \mathbf{e}.\, \phi') \\ &\supseteq \mathsf{forget}_{\Sigma', \mathbf{e}}(\mathsf{ptypes}_p^\mathbf{x}(\phi', \Sigma')) \\ &= \mathsf{forget}_{\Sigma', \mathbf{e}}(\mathsf{ptypes}_p^\mathbf{x}(y \mapsto \mathbf{z}, \Sigma') \bullet \mathsf{ptypes}_p^\mathbf{x}(\mathsf{pred}_1(\mathbf{z_1}), \Sigma') \\ &\qquad \bullet \cdots \bullet \mathsf{ptypes}_p^\mathbf{x}(\mathsf{pred}_1(\mathbf{z_k}), \Sigma')) \end{aligned}$$

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

$$
\begin{aligned}
=&\mathsf{forget}_{\Sigma',\mathbf{e}}(\{\mathcal{T}_0\} \bullet \mathsf{ptypes}_p^{\mathbf{x}}(\mathsf{pred}_1(\mathbf{z_1}),\Sigma') \\
&\qquad \bullet \cdots \bullet \mathsf{ptypes}_p^{\mathbf{x}}(\mathsf{pred}_1(\mathbf{z_k}),\Sigma')) \\
\supseteq&\mathsf{forget}_{\Sigma',\mathbf{e}}(\{\mathcal{T}_0\} \bullet p(\mathsf{pred}_1,\Sigma_1)[\mathsf{fvars}(\mathsf{pred}_1)/\mathbf{z_1}] \\
&\qquad \bullet \cdots \bullet p(\mathsf{pred}_k,\Sigma_k)[\mathsf{fvars}(\mathsf{pred}_k)/\mathbf{z_k}]) \\
\supseteq&\mathsf{forget}_{\Sigma',\mathbf{e}}(\{\mathcal{T}_0\} \bullet \{\mathcal{T}_1'\}[\mathsf{fvars}(\mathsf{pred}_1)/\mathbf{z_1}]) \\
&\qquad \bullet \cdots \bullet \{\mathcal{T}_k'\}[\mathsf{fvars}(\mathsf{pred}_k)/\mathbf{z_k}] && \text{(by } (\clubsuit)) \\
=&\mathsf{forget}_{\Sigma',\mathbf{e}}(\{\mathcal{T}_0 \bullet \mathcal{T}_1 \cdots \bullet \mathcal{T}_k\}) && \text{(by } (\ddagger)) \\
=&\{\mathsf{forget}_{\Sigma',\mathbf{e}}(\mathsf{type}_\Phi(\mathfrak{s}',\mathfrak{h}))\} && \text{(by Corollary 11.32)} \\
=&\{\mathsf{type}_\Phi(\mathfrak{s},\mathfrak{h})\}\,. && \text{(by Lemma 11.38, as } \mathfrak{s}(\mathbf{e}) \subseteq \mathrm{dom}(\mathfrak{h}))
\end{aligned}
$$

Put differently, $\mathsf{type}_\Phi(\mathfrak{s},\mathfrak{h}) \in \mathsf{unfold}_{\mathbf{x}}^{\mathsf{height}(\mathfrak{t})+1}(\lambda(\mathsf{pred}',\Sigma').\emptyset)(\mathsf{pred},\Sigma)$.
$\square$

Completeness then follows by exploiting the one-to-one correspondence between Φ-trees without holes and the models of a predicate.

Theorem 12.24. *Let* $\mathsf{pred} \in \mathbf{Preds}(\Phi)$ *with* $\mathsf{fvars}(\mathsf{pred}) = \langle z_1,\ldots,z_k\rangle$. *Let* $\mathbf{y} \triangleq \langle y_1,\ldots,y_k\rangle \subseteq \mathbf{x}$. *Assume* $(\mathfrak{s},\mathfrak{h}) \models_\Phi \mathsf{pred}(\mathbf{y})$ *and let* $\mathfrak{s}' \triangleq \mathfrak{s} \cup \{z_1 \mapsto \mathfrak{s}(y_1),\ldots,z_k \mapsto \mathfrak{s}(y_k)\}$. *Then*

$$\mathsf{type}_\Phi(\mathfrak{s},\mathfrak{h}) \in \big(\mathsf{lfp}(\mathsf{unfold}_{\mathbf{x}})(\mathsf{pred},\mathsf{aliasing}(\mathfrak{s}'))\big)[\mathsf{fvars}(\mathsf{pred})/\mathbf{y}].$$

Proof. By Lemma 10.5, there exists a Φ-tree $\mathfrak{t}$ with $\mathsf{rootpred}(\mathfrak{t}) = \mathsf{pred}(\mathfrak{s}(\mathbf{y}))$, $\mathsf{allholes}(\mathfrak{t}) = \emptyset$ and $\mathsf{heap}(\{\mathfrak{t}\}) = \mathfrak{h}$. We apply Lemma 12.23 to $\mathfrak{t}$, obtaining

$$\mathsf{type}_\Phi(\mathfrak{s}',\mathsf{heap}(\mathfrak{t})) \in \mathsf{unfold}_{\mathbf{x}}^{\mathsf{height}(\mathfrak{t})+1}(\lambda(\mathsf{pred}',\Sigma').\emptyset)(\mathsf{pred},\mathsf{aliasing}(\mathfrak{s}')).$$

Recalling that $\mathsf{lfp}(\mathsf{unfold}_{\mathbf{x}}) = \lim_{n\in\mathbb{N}} \mathsf{unfold}_{\mathbf{x}}^n(\lambda(\mathsf{pred}',\Sigma').\emptyset)$ and $\mathfrak{h} = \mathsf{heap}(\mathfrak{t})$, we get

$$\mathsf{type}_\Phi(\mathfrak{s}',\mathfrak{h}) \in \mathsf{lfp}(\mathsf{unfold}_{\mathbf{x}})(\mathsf{pred},\mathsf{aliasing}(\mathfrak{s}')). \qquad (\dagger)$$

Note that $\mathfrak{s}$ is the $[\mathsf{fvars}(\mathsf{pred})/\mathbf{y}]$-instantiation of $\mathfrak{s}'$, because $\mathbf{y} \subseteq \mathbf{x}$ by assumption. By Lemma 11.36, we then have

$$\mathsf{type}_\Phi(\mathfrak{s}',\mathfrak{h})[\mathsf{fvars}(\mathsf{pred})/\mathbf{y}] = \mathsf{type}_\Phi(\mathfrak{s},\mathfrak{h}),$$

where the latter equality follows because $\mathbf{y} \subseteq \mathbf{x}$ by assumption. Combining this identity with $(\dagger)$, we conclude

$$\mathsf{type}_\Phi(\mathfrak{s},\mathfrak{h}) \in \big(\mathsf{lfp}(\mathsf{unfold}_{\mathbf{x}})(\mathsf{pred},\mathsf{aliasing}(\mathfrak{s}'))\big)[\mathsf{fvars}(\mathsf{pred})/\mathbf{y}]. \qquad \square$$

12.2.6 *Correctness and Complexity of the Fixed-Point Computation*

Together, the results of Sections 12.2.4 and 12.2.5 imply that the fixed point $\mathsf{lfp}(\mathsf{unfold}_{\mathbf{x}})$ contains all and only the Φ-types of all predicates and stack-aliasing constraints.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Theorem 12.25. *For all* $\mathsf{pred} \in \mathbf{Preds}(\Phi)$ *and* $\Sigma \in \mathbf{AC}^{\mathbf{x} \cup \mathsf{fvars}(\mathsf{pred})}$, *it holds that* $\mathsf{lfp}(\mathsf{unfold}_{\mathbf{x}})(\mathsf{pred}, \Sigma) = \mathbf{Types}^{\Sigma}_{\Phi}(\mathsf{pred})$.

Proof. Immediate from Theorems 12.22 and 12.24. □

To derive the double-exponential complexity bound, we first investigate the complexity of computing the types of points-to assertions.

Intuitively, the models of points-to assertions correspond to Φ-trees of size 1. To compute the types of a points-to assertion $a \mapsto \mathbf{b}$, we thus systematically enumerate all such trees and check for each tree whether the points-to assertion in the tree node is identical to $\mathfrak{s}(a) \mapsto \mathfrak{s}(\mathbf{b})$.

Lemma 12.26. *Let* Σ *be a stack-aliasing constraint, let* $a \in \mathrm{dom}(\Sigma)$, *and let* $\mathbf{b} \in \mathbf{Var}^*$ *with* $\mathbf{b} \subseteq \mathrm{dom}(\Sigma)$. *Let* $n \triangleq |\Phi| + |\mathrm{dom}(\Sigma)|$. *Then* $\mathsf{type}_{\Phi}(\mathsf{ptrmodel}_{\Sigma}(a \mapsto \mathbf{b}))$ *is computable in* $2^{\mathcal{O}(n \log(n))}$.

Proof. Let $(\mathfrak{s}, \mathfrak{h}) = \mathsf{ptrmodel}_{\Sigma}(a \mapsto \mathbf{b})$. Let

$$k \triangleq \max_{\mathsf{pred} \in \mathbf{Preds}(\Phi)} |\mathsf{fvars}(\mathsf{pred})| - 1.$$

We set $\mathcal{L} \triangleq \mathrm{img}(\mathfrak{s}) \cup \{l_1, \ldots, l_k\}$, assuming w.l.o.g. that $\mathrm{img}(\mathfrak{s}) \cap \{l_1, \ldots, l_k\} = \emptyset$. Clearly, $|\mathcal{L}| \in \mathcal{O}(n)$. Now define

$$\begin{aligned} \mathbf{R} \triangleq \{ &\mathsf{pred}(\mathbf{l}) \Leftarrow (v \mapsto \mathbf{w}) \star \phi \in \mathbf{RuleInst}(\Phi) \mid \mathbf{l} \in \mathcal{L}^*, \\ &\mathsf{predroot}(\mathsf{pred}(\mathbf{l})) \in \mathrm{img}(\mathfrak{s}), \mathfrak{s}(a) = v, \mathfrak{s}(\mathbf{b}) = \mathbf{w} \} \end{aligned}$$

Recall that in every predicate call in every DUSH, at least the root parameter is a stack variable. By "padding" the set $\mathcal{L}$ with $l_1, \ldots, l_k$, we have ensured that it is possible for all other locations to be pairwise different non-stack variables, i.e., universally or existentially-quantified variables. $\mathbf{R}$ thus contains rule instances for every possible assignment of stack- and non-stack locations to parameters.

Then $\mathsf{type}_{\Phi}(\mathfrak{s}, \mathfrak{h})$ is given by

$$\{\mathsf{project}(\mathfrak{s}, \{\{a \mapsto \langle \emptyset, \mathcal{R} \rangle\}\}) \mid \mathcal{R} \in \mathbf{R}\}.$$

The complexity bound follows immediately from the fact that the Φ contains at most n predicates and $|\mathcal{L}| \in \mathcal{O}(n)$, which imply that $|\mathbf{R}| \leq n \cdot n^n \in 2^{\mathcal{O}(n \log(n))}$. □

Because (1) $\left|\mathbf{Types}^{\Sigma}(\Phi)\right| \in 2^{2^{\mathcal{O}(n^2 \log(n))}}$, and (2) there are only exponentially many different stack-aliasing constraint Σ, the fixed-point algorithm terminates after doubly-exponentially many iterations. The double-exponential complexity bound then follows by examining the time spent in each iteration.

Theorem 12.27 (Complexity of the fixed-point computation)**.** *Let* $n \triangleq |\Phi| + |\mathbf{x}|$. *Then* $\mathsf{lfp}(\mathsf{unfold}_{\mathbf{x}})$ *can be computed in* $2^{2^{\mathcal{O}(n^2 \log(n))}}$.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Proof. By Lemma 11.39, it holds that $|\mathbf{Types}^{\mathfrak{s}}(\Phi)| \in 2^{2^{\mathcal{O}(n^2 \log(n))}}$ for all $\mathfrak{s}$ with $\mathrm{dom}(\mathfrak{s}) = \mathbf{x}$. The number of predicates of Φ is bounded by n and the number of different stack-aliasing constraints over $\mathbf{x}$ is given by the $|\mathbf{x}|$-th Bell number, bounded by $n^n \in \mathcal{O}(2^{n\log(n)})$. Consequently, the number of functions with signature $\mathbf{Preds}(\Phi) \times \mathbf{AC} \rightharpoonup 2^{\mathbf{Types}(\Phi)}$ is bounded by

$$n \cdot \mathcal{O}(2^{n\log(n)}) \cdot 2^{2^{\mathcal{O}(n^2 \log(n))}} = 2^{2^{\mathcal{O}(n^2 \log(n))+2}} = 2^{2^{\mathcal{O}(n^2 \log(n))}}.$$

Since every iteration of the fixed-point computation discovers at least one new type, the computation terminates after at most $2^{2^{\mathcal{O}(n^2 \log(n))}}$ many iterations. Each iteration consists in computing

- for every predicate $\mathsf{pred} \in \mathbf{Preds}(\Phi)$ (of which there are at most n),
- for every stack-aliasing constraint $\Sigma \in \mathbf{AC}^{\mathbf{x}}$ (of which there are at most $\mathcal{O}(2^{n\log(n)})$),
- for every rule $\mathsf{pred}(\mathsf{fvars}(\mathsf{pred})) \Leftarrow \phi \in \Phi$ (of which there are at most n),

the function $\mathsf{ptypes}^{\mathbf{x}}_p(\phi, \Sigma)$, where p is the pre-fixed point from the previous iteration.

Consequently, each iteration consists of at most $n \cdot \mathcal{O}(2^{n\log(n)}) \cdot n = \mathcal{O}(2^{n\log(n)})$ many invocations of calls of the form $\mathsf{ptypes}^{\mathbf{x}}_p(\phi, \Sigma)$, where ϕ is a rule body. There are additional recursive calls, but at most $|\phi| \leq n$ for each rule body ϕ, so the total number of invocations of ptypes remains in $\mathcal{O}(2^{n\log(n)})$.

Now observe that:

1. If ϕ is a (dis-)equality, the evaluation of $\mathsf{ptypes}^{\mathbf{x}}_p(\phi, \Sigma)$ takes constant time.
2. The evaluation of $\mathsf{ptypes}^{\mathbf{x}}_p(\phi, \Sigma)$ for points-to assertions can be done in time $\mathcal{O}(2^{n\log(n)})$ by Lemma 12.26.
3. The evaluation of the three operations $\bullet$, $\cdot[\cdot/\cdot]$, and $\mathsf{forget}_{\cdot,\cdot}(\cdot)$ each takes time polynomial in the size of the types to which the operation is applied. For $\cdot[\cdot/\cdot]$, and $\mathsf{forget}_{\cdot,\cdot}(\cdot)$, this is trivial. For the composition operation, $\bullet$, the polynomial bound follows from the facts that (1) we apply merge to polynomially many formulas (namely to all pairs of formulas in the types that are composed), (2) the number of $\triangleright$ steps that can be applied is bounded by the length of the formula and (3) each $\triangleright$ step can be computed in polynomial time.
4. As Lemma 11.16 bounds the size of each type by $2^{\mathcal{O}(n^2 \log(n))}$, this yields a bound of $2^{poly(n)}$ for each of the $\cdot[\cdot/\cdot]$, $\mathsf{forget}_{\cdot,\cdot}(\cdot)$, and $\bullet$ operations.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

$$
\begin{aligned}
\mathsf{types}(\mathbf{emp}, \Sigma) &\triangleq \{\{\mathbf{emp}\}\}\\
\mathsf{types}(x \approx y, \Sigma) &\triangleq \text{if } \langle x, y\rangle \in \Sigma \text{ then } \{\{\mathbf{emp}\}\} \text{ else } \emptyset\\
\mathsf{types}(x \not\approx y, \Sigma) &\triangleq \text{if } \langle x, y\rangle \notin \Sigma \text{ then } \{\{\mathbf{emp}\}\} \text{ else } \emptyset\\
\mathsf{types}(a \mapsto \mathbf{b}, \Sigma) &\triangleq \{\mathsf{type}_{\Phi}(\mathsf{ptrmodel}_{\Sigma}(a \mapsto \mathbf{b}))\}\\
\mathsf{types}(\mathsf{pred}(\mathbf{y}), \Sigma) &\triangleq \text{let } \Sigma' \in \mathbf{AC}^{\mathsf{dom}(\Sigma)\cup\mathsf{fvars}(\mathsf{pred})} \text{ be the unique}\\
&\qquad \text{aliasing constraint with } \Sigma = \Sigma'[\mathsf{fvars}(\mathsf{pred})/\mathbf{y}]\\
&\qquad \text{in } \left(\mathsf{lfp}(\mathsf{unfold}_{\mathbf{x}})(\mathsf{pred}, \Sigma')\right)[\mathsf{fvars}(\mathsf{pred})/\mathbf{y}]\\
\mathsf{types}(\phi_1 \star \phi_2, \Sigma) &\triangleq \mathsf{types}(\phi_1, \Sigma) \bullet \mathsf{types}(\phi_2, \Sigma)\\
\mathsf{types}(\phi_1 \wedge \phi_2, \Sigma) &\triangleq \mathsf{types}(\phi_1, \Sigma) \cap \mathsf{types}(\phi_2, \Sigma)\\
\mathsf{types}(\phi_1 \vee \phi_2, \Sigma) &\triangleq \mathsf{types}(\phi_1, \Sigma) \cup \mathsf{types}(\phi_2, \Sigma)\\
\mathsf{types}(\phi_1 \wedge \neg\phi_2, \Sigma) &\triangleq \mathsf{types}(\phi_1, \Sigma) \setminus \mathsf{types}(\phi_2, \Sigma)\\
\mathsf{types}(\phi_0 \wedge (\phi_1 \mathbin{-\!\circledast} \phi_2), \Sigma) &\triangleq\\
&\{\mathcal{T} \in \mathsf{types}(\phi_0, \Sigma) \mid \exists \mathcal{T}' \in \mathsf{types}(\phi_1, \Sigma).\, \mathcal{T} \bullet \mathcal{T}' \in \mathsf{types}(\phi_2, \Sigma)\}\\
\mathsf{types}(\phi_0 \wedge (\phi_1 \mathbin{-\!\star} \phi_2), \Sigma) &\triangleq\\
&\{\mathcal{T} \in \mathsf{types}(\phi_0, \Sigma) \mid \forall \mathcal{T}' \in \mathsf{types}(\phi_1, \Sigma).\, \mathcal{T} \bullet \mathcal{T}' \in \mathsf{types}(\phi_2, \Sigma)\}
\end{aligned}
$$

Figure 12.2: Computing the Φ-types of quantifier-free guarded formula $\phi \in \mathbf{SLID}^{\mathsf{g}}_{\mathsf{btw}}$ for stacks with stack-aliasing constraint Σ.

5. Again by Lemma 11.39, the number of types to which each function is applied is bounded by $2^{2^{\mathcal{O}(n^2 \log(n))}}$.

We thus obtain that the evaluation of $\mathsf{ptypes}^{\mathbf{x}}_{p}(\phi, \Sigma)$ for a fixed formula ϕ, not taking into account the evaluation time of recursive calls, takes at most time

$$2^{\mathcal{O}(poly(n))} \cdot 2^{2^{\mathcal{O}(n^2 \log(n))}} = 2^{2^{\mathcal{O}(n^2 \log(n))}}.$$

The total run time of the fixed-point computation is thus bounded by

$$\underbrace{\mathcal{O}(2^{n \log(n)})}_{\text{number of calls to ptypes}} \cdot \underbrace{2^{2^{\mathcal{O}(n^2 \log(n))}}}_{\text{evaluation time per call}} = 2^{2^{\mathcal{O}(n^2 \log(n))}}. \qquad \square$$

12.3 COMPUTING THE TYPES OF GUARDED FORMULAS

Now that we know how to compute the types of predicate calls, we are ready to define a function $\mathsf{types}(\phi, \Sigma)$ that computes the types of arbitrary $\mathbf{SLID}^{\mathsf{g}}_{\mathsf{btw}}$ formulas ϕ—i.e., quantifier-free guarded formulas—for fixed stack-aliasing constraint Σ. I define the function types in Fig. 12.2.

Theorem 12.28 (Correctness of type computation)**.** *Let* $\phi \in \mathbf{SLID}^{\mathsf{g}}_{\mathsf{btw}}$ *with* $\mathsf{fvars}(\phi) = \mathbf{x}$ *and* $\mathsf{locs}(\phi) = \emptyset$. *Let* $\Sigma \in \mathbf{AC}^{\mathbf{x}}$. *Then* $\mathbf{Types}^{\Sigma}_{\Phi}(\phi) = \mathsf{types}(\phi, \Sigma)$.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Wien Bibliothek Your knowledge hub

Proof. We proceed by induction on ϕ.

CASE $\phi = \mathbf{emp}$. The claim follows from Lemma 12.18.

CASE $\phi = x \approx y$.] The claim follows from Lemma 12.19.

CASE $\phi = x \not\approx y$. The claim follows from Lemma 12.19.

CASE $\phi = a \mapsto \mathbf{b}$. The claim follows from Lemma 12.20.

CASE $\phi = \mathsf{pred}(\mathbf{y})$. Let $\mathcal{T} \in \mathbf{Types}^{\Sigma}_{\Phi}(\phi)$. Let $(\mathfrak{s}, \mathfrak{h})$ be a model with $\mathsf{aliasing}(\mathfrak{s}) = \Sigma$, $(\mathfrak{s}, \mathfrak{h}) \models_{\Phi} \mathsf{pred}(\mathbf{y})$ and $\mathcal{T} = \mathsf{type}_{\Phi}(\mathfrak{s}, \mathfrak{h})$. Such a model must exist by the definition of $\mathbf{Types}^{\Sigma}_{\Phi}(\cdot)$.

Let $\mathfrak{s}' \triangleq \mathfrak{s} \cup \{\mathsf{fvars}(\mathsf{pred}) \mapsto \mathfrak{s}(\mathbf{y})\}$. It is easy to see that $\mathfrak{s}$ is a $[\mathsf{fvars}(\mathsf{pred})/\mathbf{y}]$-instantiation of $\mathfrak{s}'$.

Let $\Sigma' \triangleq \mathsf{aliasing}(\mathfrak{s}')$ and $\mathcal{T}' \triangleq \mathsf{type}_{\Phi}(\mathfrak{s}', \mathfrak{h})$. It follows from Lemma 11.36 that $\mathcal{T}'[\mathsf{fvars}(\mathsf{pred})/\mathbf{y}] = \mathsf{type}_{\Phi}(\mathfrak{s}, \mathfrak{h}) = \mathcal{T}$ (†). Since $(\mathfrak{s}, \mathfrak{h}) \models_{\Phi} \mathsf{pred}(\mathbf{y})$, we obtain $(\mathfrak{s}', \mathfrak{h}) \models_{\Phi} \mathsf{pred}(\mathsf{fvars}(\mathsf{pred}))$ and thus $\mathcal{T}' \in \mathbf{Types}^{\Sigma'}_{\Phi}(\mathsf{pred}(\mathsf{fvars}(\mathsf{pred})))$.

Now observe that:

1. By construction, Σ' is the unique stack-aliasing constraint with $\Sigma = \Sigma'[\mathsf{fvars}(\mathsf{pred})/\mathbf{y}]$.
2. By Theorem 12.25,
$$\mathsf{lfp}(\mathsf{unfold}_{\mathbf{x}})(\mathsf{pred}, \Sigma') = \mathbf{Types}^{\Sigma'}_{\Phi}(\mathsf{pred}(\mathsf{fvars}(\mathsf{pred}))),$$
so $\mathcal{T}' \in \mathsf{lfp}(\mathsf{unfold}_{\mathbf{x}})(\mathsf{pred}, \Sigma')$.
3. $\mathcal{T} = \mathcal{T}'[\mathsf{fvars}(\mathsf{pred})/\mathbf{y}]$ by (†).

Consequently, $\mathcal{T} \in \mathsf{types}(\mathsf{pred}(\mathbf{y}), \Sigma)$.

Conversely, let $\mathcal{T} \in \mathsf{types}(\mathsf{pred}(\mathbf{y}), \Sigma)$. Write Σ' for the unique stack-aliasing constraint with $\Sigma = \Sigma'[\mathsf{fvars}(\mathsf{pred})/\mathbf{y}]$. Then $\mathcal{T} \in (\mathsf{lfp}(\mathsf{unfold}_{\mathbf{x}})(\mathsf{pred}, \Sigma'))[\mathsf{fvars}(\mathsf{pred})/\mathbf{y}]$. By Theorem 12.25, this guarantees that

$$\mathcal{T} \in \mathbf{Types}^{\Sigma'}_{\Phi}(\mathsf{pred}(\mathsf{fvars}(\mathsf{pred})))[\mathsf{fvars}(\mathsf{pred})/\mathbf{y}].$$

Since $\langle y_i, z_i \rangle \in \Sigma'$, where y_i and z_i denote the i-th elements of $\mathbf{y}$ and $\mathbf{z}$ and $1 \leq i \leq |\mathbf{y}|$, it follows that

$$\mathcal{T} \in \mathbf{Types}^{\Sigma'}_{\Phi}(\mathsf{pred}(\mathbf{y}))[\mathsf{fvars}(\mathsf{pred})/\mathbf{y}].$$

After instantiating $\mathsf{fvars}(\mathsf{pred})$ with $\mathbf{y}$, we are left with the original stack-aliasing constraint Σ, so it holds that

$$\mathbf{Types}^{\Sigma'}_{\Phi}(\mathsf{pred}(\mathbf{y}))[\mathsf{fvars}(\mathsf{pred})/\mathbf{y}] = \mathbf{Types}^{\Sigma}_{\Phi}(\mathsf{pred}(\mathbf{y})).$$

Thus, $\mathcal{T} \in \mathbf{Types}^{\Sigma}_{\Phi}(\mathsf{pred}(\mathbf{y}))$.

TU Bibliothek Your knowledge hub WIEN
Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.

CASE $\phi = \phi_1 \star \phi_2$.

$$\begin{aligned}
&\mathbf{Types}^{\Sigma}_{\Phi}(\phi_1 \star \phi_2) \\
=&\mathbf{Types}^{\Sigma}_{\Phi}(\phi_1) \bullet \mathbf{Types}^{\Sigma}_{\Phi}(\phi_2) && \text{(by Lemma 12.13)} \\
=&\mathsf{types}(\phi_1, \Sigma) \bullet \mathsf{types}(\phi_2, \Sigma) && \text{(by the induction hypotheses)} \\
=&\mathsf{types}(\phi_1 \star \phi_2, \Sigma).
\end{aligned}$$

CASE $\phi = \phi_1 \wedge \phi_2$. Let $\mathcal{T} \in \mathbf{Types}^{\Sigma}_{\Phi}(\phi)$. Let $(\mathfrak{s}, \mathfrak{h})$ be a model with $\mathsf{aliasing}(\mathfrak{s}) = \Sigma$, $(\mathfrak{s}, \mathfrak{h}) \models_{\Phi} \phi_1 \wedge \phi_2$ and $\mathcal{T} = \mathsf{type}_{\Phi}(\mathfrak{s}, \mathfrak{h})$. Such a model must exist by the definition of $\mathbf{Types}^{\Sigma}_{\Phi}(\cdot)$. Then $(\mathfrak{s}, \mathfrak{h}) \models_{\Phi} \phi_1$ and $(\mathfrak{s}, \mathfrak{h}) \models_{\Phi} \phi_2$ by the semantics of $\wedge$. Consequently, $\mathcal{T} \in \mathbf{Types}^{\Sigma}_{\Phi}(\phi_1)$ and $\mathcal{T} \in \mathbf{Types}^{\Sigma}_{\Phi}(\phi_2)$, i.e.,

$$\mathcal{T} \in \mathbf{Types}^{\Sigma}_{\Phi}(\phi_1) \cap \mathbf{Types}^{\Sigma}_{\Phi}(\phi_2).$$

By the induction hypotheses, $\mathbf{Types}^{\Sigma}_{\Phi}(\phi_i) = \mathsf{types}(\phi_i, \Sigma)$, $1 \leq i \leq 2$. Consequently, $\mathcal{T} \in \mathsf{types}(\phi_1, \Sigma) \cap \mathsf{types}(\phi_2, \Sigma) = \mathsf{types}(\phi, \Sigma)$.

Conversely, let $\mathcal{T} \in \mathsf{types}(\phi, \Sigma) = \mathsf{types}(\phi_1, \Sigma) \cap \mathsf{types}(\phi_2, \Sigma)$. By the induction hypotheses, we then have $\mathcal{T} \in \mathbf{Types}^{\Sigma}_{\Phi}(\phi_1)$ and $\mathcal{T} \in \mathbf{Types}^{\Sigma}_{\Phi}(\phi_2)$. Let $(\mathfrak{s}, \mathfrak{h})$ be a model with $\mathsf{aliasing}(\mathfrak{s}) = \Sigma$, $(\mathfrak{s}, \mathfrak{h}) \models_{\Phi} \phi_1$ and $\mathcal{T} = \mathsf{type}_{\Phi}(\mathfrak{s}, \mathfrak{h})$. Because $\mathcal{T} \in \mathbf{Types}^{\Sigma}_{\Phi}(\phi_2)$ and $(\mathfrak{s}, \mathfrak{h}) \in \mathbf{Models}^{\mathfrak{g}}_{\Phi}$ (by Lemma 8.9), it follows from Corollary 12.12 that $(\mathfrak{s}, \mathfrak{h}) \models_{\Phi} \phi_2$. Hence, $(\mathfrak{s}, \mathfrak{h}) \models_{\Phi} \phi_1 \wedge \phi_2$, implying $\mathcal{T} \in \mathbf{Types}^{\Sigma}_{\Phi}(\phi)$.

CASE $\phi = \phi_1 \vee \phi_2$. Analogously.

CASE $\phi = \phi_1 \wedge \neg\phi_2$. Analogously.

CASE $\phi = \phi_0 \wedge (\phi_1 \mathrel{-\!\!\circledast} \phi_2)$. Let $\mathcal{T} \in \mathbf{Types}^{\Sigma}_{\Phi}(\phi)$. Let $(\mathfrak{s}, \mathfrak{h})$ be a model with $\mathsf{aliasing}(\mathfrak{s}) = \Sigma$, $(\mathfrak{s}, \mathfrak{h}) \models_{\Phi} \phi_0$ and $(\mathfrak{s}, \mathfrak{h}) \models_{\Phi} \phi_1 \mathrel{-\!\!\circledast} \phi_2$ and $\mathcal{T} = \mathsf{type}_{\Phi}(\mathfrak{s}, \mathfrak{h})$. Such a model must exist by the definition of $\mathbf{Types}^{\Sigma}_{\Phi}(\cdot)$. By the induction hypothesis, $\mathcal{T} \in \mathsf{types}(\phi_0, \Sigma)$ (†). By the semantics of $\mathrel{-\!\!\circledast}$, there exists a heap $\mathfrak{h}_1$ with $(\mathfrak{s}, \mathfrak{h}_1) \models_{\Phi} \phi_1$ and $(\mathfrak{s}, \mathfrak{h} + \mathfrak{h}_1) \models_{\Phi} \phi_2$. Let $\mathcal{T}_1 \triangleq \mathsf{type}_{\Phi}(\mathfrak{s}, \mathfrak{h}_1)$ and $\mathcal{T}_2 \triangleq \mathsf{type}_{\Phi}(\mathfrak{s}, \mathfrak{h} + \mathfrak{h}_2)$. By Corollary 11.32, $\mathcal{T}_2 = \mathcal{T} \bullet \mathcal{T}_1$. Combining this with (†), we derive that $\mathcal{T}$ is in the set

$$\{\mathcal{T}' \in \mathsf{types}(\phi_0, \Sigma) \mid \exists \mathcal{T}'' \in \mathbf{Types}^{\Sigma}_{\Phi}(\phi_1).\ \mathcal{T}' \bullet \mathcal{T}'' \in \mathbf{Types}^{\Sigma}_{\Phi}(\phi_2)\}.$$

By the induction hypotheses for ϕ_1 and ϕ_2, we thus have that $\mathcal{T}$ is in the set

$$\begin{aligned}
&\{\mathcal{T}' \in \mathsf{types}(\phi_0, \Sigma) \mid \exists \mathcal{T}'' \in \mathsf{types}(\phi_1, \Sigma).\ \mathcal{T}' \bullet \mathcal{T}'' \in \mathsf{types}(\phi_2, \Sigma)\} \\
=&\mathsf{types}(\phi_1 \mathrel{-\!\!\circledast} \phi_2, \Sigma).
\end{aligned}$$

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Conversely, let $\mathcal{T} \in \mathsf{types}(\phi_0 \wedge (\phi_1 \circledast \phi_2), \Sigma)$. By the induction hypotheses for ϕ_0, ϕ_1 and ϕ_2, this implies that $\mathcal{T}$ is in the set

$$\{\mathcal{T}' \in \mathbf{Types}^{\Sigma}_{\Phi}(\phi_0) \mid \exists \mathcal{T}'' \in \mathbf{Types}^{\Sigma}_{\Phi}(\phi_1). \\ \mathcal{T}' \bullet \mathcal{T}'' \in \mathbf{Types}^{\Sigma}_{\Phi}(\phi_2)\}.$$

In particular, $\mathcal{T} \in \mathbf{Types}^{\Sigma}_{\Phi}(\phi_0)$ (†). Let $\mathcal{T}_1 \in \mathbf{Types}^{\Sigma}_{\Phi}(\phi_1)$ such that $\mathcal{T} \bullet \mathcal{T}_1 \in \mathbf{Types}^{\Sigma}_{\Phi}(\phi_2)$, i.e., a witness for the existential in the above set. Moreover, let $\mathfrak{h}$ be a heap such that $\mathsf{type}_{\Phi}(\mathfrak{s}, \mathfrak{h}) = \mathcal{T}$. Further, let $\mathfrak{h}_1$ be a heap such that $\mathsf{type}_{\Phi}(\mathfrak{s}, \mathfrak{h}_1) = \mathcal{T}_1$ and $(\mathfrak{s}, \mathfrak{h}_1) \models_{\Phi} \phi_1$. Such a heap exists by definition of $\mathbf{Types}^{\Sigma}_{\Phi}(\phi_1)$. Assume w.l.o.g. that $\mathfrak{h} + \mathfrak{h}_1 \neq \bot$—otherwise, replace $\mathfrak{h}_1$ with an isomorphic heap that has this property.

Corollary 11.32 yields $\mathsf{type}_{\Phi}(\mathfrak{s}, \mathfrak{h} + \mathfrak{h}_1) = \mathcal{T} \bullet \mathcal{T}_1 \in \mathbf{Types}^{\Sigma}_{\Phi}(\phi_2)$. Because $\phi_0, \phi_1 \in \mathbf{SLID}^{\mathsf{g}}_{\mathsf{btw}}$, $(\mathfrak{s}, \mathfrak{h}) \in \mathbf{Models}^{\mathsf{g}}_{\Phi}$ and $(\mathfrak{s}, \mathfrak{h}_1) \in \mathbf{Models}^{\mathsf{g}}_{\Phi}$ by Lemma 8.9. Therefore also $(\mathfrak{s}, \mathfrak{h} + \mathfrak{h}_1) \in \mathbf{Models}^{\mathsf{g}}_{\Phi}$. Corollary 12.12 then gives us that $(\mathfrak{s}, \mathfrak{h} + \mathfrak{h}_1) \models_{\Phi} \phi_2$. Therefore, $(\mathfrak{s}, \mathfrak{h}) \models_{\Phi} \phi_1 \circledast \phi_2$, implying $\mathcal{T} \in \mathbf{Types}^{\Sigma}_{\Phi}(\phi_1 \circledast \phi_2)$. Combining this with (†), we conclude that $\mathcal{T} \in \mathbf{Types}^{\Sigma}_{\Phi}(\phi_0 \wedge (\phi_1 \circledast \phi_2))$.

CASE $\phi = \phi_0 \wedge (\phi_1 \mathrel{-\!\!\star} \phi_2)$. Analogously. □

Unsurprisingly, the asymptotic complexity of $\mathsf{types}(\phi, \Sigma)$ coincides with the asymptotic complexity of the fixed-point computation.

Theorem 12.29 (Complexity of type computation)**.** *Let $\phi \in \mathbf{SLID}^{\mathsf{g}}_{\mathsf{btw}}$ and let $\Sigma \in \mathbf{AC}^{\mathsf{fvars}(\phi)}$. Let $n \triangleq |\Phi| + |\phi|$. Then $\mathsf{types}(\phi, \Sigma)$ can be computed in $2^{2^{\mathcal{O}(n^2 \log(n))}}$.*

Proof. Recall from Lemma 11.39 that $\left|\mathbf{Types}^{\Sigma}(\Phi)\right| \in 2^{2^{\mathcal{O}(n^2 \log(n))}}$ (†).

The evaluation of $\mathsf{types}(\phi, \Sigma)$ consists in evaluating at most $|\phi| \leq n$ invocations of the form $\mathsf{types}(\cdot, \Sigma)$. Each of these invocations can be evaluated in time at most $2^{2^{\mathcal{O}(n^2 \log(n))}}$:

- For **emp** and (dis-)equalities, this is trivial.
- For points-to assertions, this follows from Lemma 12.26.
- For predicate calls, this follows from Theorem 12.27.
- For $\phi_1 \star \phi_2$, this follows from the facts that (1) $\bullet$ is applied to at most $2^{2^{\mathcal{O}(n^2 \log(n))}} \cdot 2^{2^{\mathcal{O}(n^2 \log(n))}} = 2^{2^{\mathcal{O}(n^2 \log(n))}}$ many types by (†) and (2) the composition $\mathcal{T}_1 \bullet \mathcal{T}_2$ takes time at most $2^{poly(n)}$, as argued in the proof of Theorem 12.27.
- For septraction and the magic wand, (†) also implies that we must compute at most $2^{2^{\mathcal{O}(n^2 \log(n))}} \cdot 2^{2^{\mathcal{O}(n^2 \log(n))}}$ many composition operations, so the bound follows as for $\star$.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

- For $\wedge$, $\vee$, and $\neg$, the bound follows immediately from ($\dagger$).

The overall complexity is thus bounded by $n \cdot 2^{2^{\mathcal{O}(n^2 \log(n))}} = 2^{2^{\mathcal{O}(n^2 \log(n))}}$. □

Now that we know that $\mathsf{types}(\phi, \Sigma)$ computes $\mathbf{Types}^{\Sigma}_{\Phi}(\phi)$ in double-exponential time, we are ready to prove that the satisfiability problem for guarded quantifier-free formulas with SIDs from $\mathbf{ID}_{\mathsf{btw}}$ is decidable in double-exponential time.

Theorem 12.30 (Decidability of $\mathbf{SLID}^{\mathsf{g}}_{\mathsf{btw}}$)**.** *Let $\phi \in \mathbf{SLID}^{\mathsf{g}}_{\mathsf{btw}}$. Let $n \triangleq |\Phi| + |\phi|$. It is decidable in time $2^{2^{\mathcal{O}(n^2 \log(n))}}$ whether ϕ is satisfiable.*

Proof. Let $\mathbf{x} \triangleq \mathsf{fvars}(\phi)$. Note that $|\mathbf{x}| \leq n$. The formula ϕ is satisfiable if and only if there exists a model $(\mathfrak{s}, \mathfrak{h})$ with $(\mathfrak{s}, \mathfrak{h}) \models_{\Phi} \phi$. By definition,

$$\mathbf{Types}^{\mathbf{x}}_{\Phi}(\phi) = \{\mathsf{type}_{\Phi}(\mathfrak{s}, \mathfrak{h}) \mid \mathrm{dom}(\mathfrak{s}) = \mathbf{x}, \mathfrak{h} \in \mathbf{Heaps}, (\mathfrak{s}, \mathfrak{h}) \models_{\Phi} \phi\}\,.$$

Because every model of ϕ is guarded by Lemma 8.9, it follows that ϕ is satisfiable if and only if $\mathbf{Types}^{\mathbf{x}}_{\Phi}(\phi) \neq \emptyset$.

By Corollary 11.26, $\mathbf{Types}^{\mathbf{x}}_{\Phi}(\phi) = \bigcup \{\mathbf{Types}^{\Sigma}_{\Phi}(\phi) \mid \Sigma \in \mathbf{AC}^{\mathbf{x}}\}$. By Theorem 12.28, we then obtain

$$\mathbf{Types}^{\mathbf{x}}_{\Phi}(\phi) = \bigcup \{\mathsf{types}(\Sigma, \phi) \mid \Sigma \in \mathbf{AC}^{\mathbf{x}}\}\,.$$

We use Theorem 12.29 and the observation that

$$|\mathbf{AC}^{\mathbf{x}}| \leq n^n \in \mathcal{O}(2^{n \log(n)})$$

to conclude that we can compute the finite set $\mathbf{Types}^{\mathbf{x}}_{\Phi}(\phi) \neq \emptyset$ (and thus also check whether this set is nonempty) in time

$$\mathcal{O}(2^{n \log(n)}) \cdot 2^{2^{\mathcal{O}(n^2 \log(n))}} = 2^{2^{\mathcal{O}(n^2 \log(n))}}.$$ □

Since the entailment query $\phi \models_{\Phi} \psi$ is equivalent to checking the unsatisfiability of $\phi \wedge \neg\psi$, and the negation in $\phi \wedge \neg\psi$ is guarded, we obtain an entailment checker with the same complexity for free.

Corollary 12.31 (Decidability of entailment for $\mathbf{SLID}^{\mathsf{g}}_{\mathsf{btw}}$)**.** *Let $\phi, \psi \in \mathbf{SLID}^{\mathsf{g}}_{\mathsf{btw}}$ and $n \triangleq |\Phi| + |\phi| + |\psi|$. The entailment problem $\phi \models_{\Phi} \psi$ is decidable in time $2^{2^{\mathcal{O}(n^2 \log(n))}}$.*

Proof. If $\phi, \psi \in \mathbf{SLID}^{\mathsf{g}}_{\mathsf{btw}}$, then $\phi \wedge \neg\psi \in \mathbf{SLID}^{\mathsf{g}}_{\mathsf{btw}}$. The entailment $\phi \models_{\Phi} \psi$ is valid iff $\phi \wedge \neg\psi$ is unsatisfiable. Since 2ExpTime is closed under complementation, the claim follows from Theorem 12.30. □

In other words, we have developed a sound and complete entailment checker for guarded formulas in the bounded-treewidth fragment, fixing the incompleteness issues of [KMZ19a], while extending the result beyond symbolic heaps and retaining the same asymptotic complexity as in [KMZ19a].

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Wien Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.

13 BEYOND GUARDED SEPARATION LOGIC: UNDECIDABILITY PROOFS

In this chapter, I write $\mathbf{SLID}_{\mathsf{btw}}(\cdot_1, \ldots, \cdot_k)$ for the restriction of $\mathbf{SLID}^{\mathsf{qf}}_{\mathsf{btw}}$ to formulas built from atomic predicates τ (as defined in Fig. 8.1) and the additional symbols and operators $\cdot_1, \ldots, \cdot_k$. For example, $\mathbf{SLID}_{\mathsf{btw}}(\wedge, \star, \mathsf{t})$ is the SL in which formulas are built from atomic predicates τ, additional predicate t and binary operators $\star$, $\wedge$.

In this chapter, I explicitly need t as built-in atom rather than derived formula. As usual, t holds in all models, i.e., $(\mathfrak{s}, \mathfrak{h}) \models_{\Phi} \mathsf{t}$ for all choices of $\mathfrak{s}$ and $\mathfrak{h}$.

13.1 UNDECIDABILITY OF UNGUARDED SLID

In this section, I justify the use of guarded negation, magic wand, and septraction in $\mathbf{SLID}^{\mathsf{g}}_{\mathsf{btw}}$ by proving that allowing any of these three operators to be used without guards leads to an undecidable logic. Together with the decidability result from Chapter 12, this yields an almost tight delineation between decidability and undecidability of separation logics that allow arbitrary $\mathbf{ID}_{\mathsf{btw}}$ SIDs.[1]

CONTEXT-FREE GRAMMARS. The undecidability results are based on an encoding of the language-intersection problem for context-free grammars.

Definition 13.1 (Context-free grammar)**.** *A* context-free grammar *(CFG) is a 4-tuple $\mathbf{G} = \langle \mathbf{N}, \mathbf{T}, \mathbf{R}, \mathbf{S} \rangle$, where $\mathbf{N}$ is a finite set of* nonterminals*; $\mathbf{T}$ is a finite set of terminals, disjoint from $\mathbf{N}$; $\mathbf{R} \subseteq \mathbf{N} \times (\mathbf{N}^2 \cup \mathbf{T})$ is a finite set of* production rules*; and $\mathbf{S} \in \mathbf{N}$ is the* start symbol*. $\mathbf{CFG}$ is the set of all CFGs.*

We often denote production rules $\langle a, b \rangle$ by $a \to b$ to improve readability. We assume w.l.o.g. that CFGs are in Chomsky normal form. Further, we only consider CFGs that do not accept the empty word. Under these assumptions, rules are either of the form $N \to AB$, $A, B \in \mathbf{N}$, or of the form $N \to a$, $a \in \mathbf{T}$.

Definition 13.2. *Let $\mathbf{G} = \langle \mathbf{N}, \mathbf{T}, \mathbf{R}, \mathbf{S} \rangle \in \mathbf{CFG}$ and let $v, w \in \mathbf{N} \cup \mathbf{T}^*$. We write $v \Rightarrow w$ if there exist $u_1, u_2 \in \mathbf{N} \cup \mathbf{T}^*$, $\langle a, b \rangle \in \mathbf{R}$ such that $v = u_1 \cdot a \cdot u_2$ and $w = u_1 \cdot b \cdot u_2$. We write $\Rightarrow^+$ for the transitive closure of $\Rightarrow$. The* language *of $\mathbf{G}$ is given by $\mathcal{L}(\mathbf{G}) \triangleq \{ w \in \mathbf{T}^* \mid \mathbf{S} \Rightarrow^+ w \}$.*

1 The ideas presented in this section are to a large extent due to Florian Zuleger; in particular, Zuleger proposed the SID encoding of the CFG intersection problem.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

$$\mathsf{letter}_i(a) \Leftarrow a \mapsto \underbrace{\langle \mathsf{nil}, \ldots, \mathsf{nil} \rangle}_{\text{length } i},\ 1 \leq i \leq n$$

$$N(x_1, x_2, x_3) \Leftarrow \exists l, r, m.\, (x_1 \mapsto \langle l, r \rangle) \star A(l, x_2, m) \star B(r, m, x_3),$$
$$j \in \{1, 2\}, (N \to AB) \in \mathbf{R}_j$$

$$N(x_1, x_2, x_3) \Leftarrow \exists a.\, (x_1 \mapsto \langle x_3, a \rangle) \star \mathsf{letter}_k(a) \star x_1 \approx x_2,$$
$$j \in \{1, 2\}, (N \to a_k) \in \mathbf{R}_j$$

$$\mathsf{word}(x, y) \Leftarrow \exists a.\, (x \mapsto \langle y, a \rangle) \star \mathsf{letter}_i(a),$$
$$1 \leq i \leq n$$

$$\mathsf{word}(x, y) \Leftarrow \exists \langle n, a \rangle.\, (x \mapsto \langle n, a \rangle) \star \mathsf{letter}_i(a) \star \mathsf{word}(n, y),$$
$$1 \leq i \leq n$$

Figure 13.1: The SID Φ that encodes the derivations of the context-free grammars $\mathbf{G}_1 = \langle \mathbf{N}_1, \mathbf{T}, \mathbf{R}_1, \mathbf{S}_1 \rangle$ and $\mathbf{G}_2 = \langle \mathbf{N}_2, \mathbf{T}, \mathbf{R}_2, \mathbf{S}_2 \rangle$.

In the following, we exploit the following classic result.

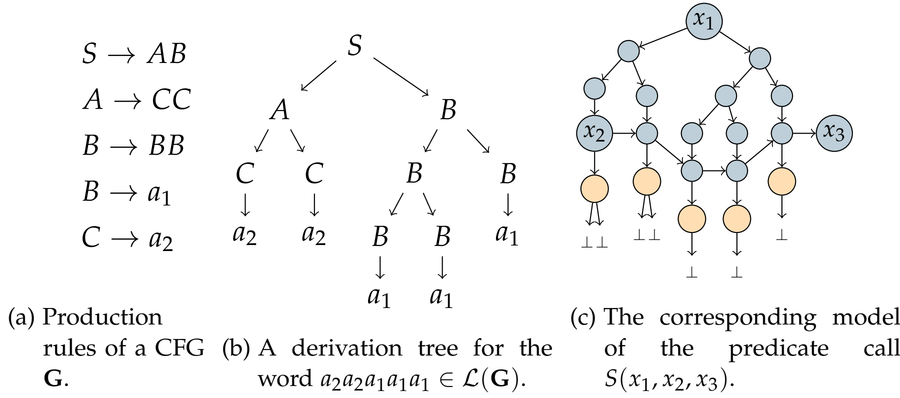
Theorem 13.3 (Undecidability of language intersection)**.** *Let* $\mathbf{G}_1, \mathbf{G}_2 \in$ **CFG**. *It is undecidable whether* $\mathcal{L}(\mathbf{G}_1) \cap \mathcal{L}(\mathbf{G}_2) \neq \emptyset$.

It is easy to see that this result continues to hold under our assumption that CFGs do not accept the empty word.

ENCODING CFGS AS SIDS. Let $\mathbf{T} = \{a_1, \ldots, a_n\}$ and for $1 \leq i \leq 2$, let $\mathbf{G}_i = \langle \mathbf{N}_i, \mathbf{T}, \mathbf{R}_i, \mathbf{S}_i \rangle$ be a context-free grammar. Assume w.l.o.g. that $\mathbf{N}_1 \cap \mathbf{N}_2 = \emptyset$. Consider the SID Φ defined in Fig. 13.1. The predicates N, $N \in \mathbf{N}_1 \cup \mathbf{N}_2$, and letter_i, $1 \leq i \leq n$, encode the derivations of the grammars $\mathbf{G}_1, \mathbf{G}_2$ as trees with linked leaves (TLL), similar to the SID in Fig. 3.2 on p. 27. The predicate word is an auxiliary predicate that overapproximates the lists of linked leaves that the TLLs may contain; we will need it later. Every word in $\mathcal{L}(\mathbf{G}_i)$ corresponds to at least one model $(\mathfrak{s}, \mathfrak{h})$ with $(\mathfrak{s}, \mathfrak{h}) \models_\Phi \mathbf{S}_i(x_1, x_2, x_3)$; and every model $(\mathfrak{s}, \mathfrak{h})$ with $(\mathfrak{s}, \mathfrak{h}) \models_\Phi \mathbf{S}_i(x_1, x_2, x_3)$ corresponds to a *derivation tree* and a word in $\mathcal{L}(\mathbf{G}_i)$, where the inner nodes of the TLL correspond to the derivation tree and the linked list of leaves correspond to the word in $\mathcal{L}(\mathbf{G}_i)$.

Example 13.4. *I illustrate the encoding in Fig. 13.2. Figure 13.2a shows the rules* $\mathbf{R}$ *of a simple CFG* $\mathbf{G} = \langle \{S, A, B, C\}, \{a_1, a_2\}, \mathbf{R}, S \rangle$. *The derivation tree in Figure 13.2b proves that the word* $a_2 a_2 a_1 a_1 a_1$ *is in the language of the grammar,* $\mathcal{L}(\mathbf{G})$. *In Fig. 13.2c, we show the stack–heap model that encodes the aforementioned derivation tree. Every nonterminal is translated to a node in a binary tree (blue). The leaves of the tree are linked. They each have a successor that encodes a terminal symbol of the derivation (orange): The node contains* k *pointers to* nil *(denoted as* $\perp$ *in the figure) to represent terminal* a_k. *The list of linked leaves and orange nodes together form the* induced word *of the model as defined later in Definition 13.7.*

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN



(a) Production rules of a CFG $\mathbf{G}$.

(b) A derivation tree for the word $a_2a_2a_1a_1a_1 \in \mathcal{L}(\mathbf{G})$.

(c) The corresponding model of the predicate call $S(x_1, x_2, x_3)$.

Figure 13.2: Encoding a derivation of a context-free grammar as a stack–heap model.

To show the correctness of the encoding, we need the *induced words* of the models of Φ, which correspond to the *induced letters* of the list of linked leaves in the models.

Definition 13.5 (Induced letters). *Let* $\mathbf{G} = \langle \mathbf{N}, \mathbf{T}, \mathbf{R}, \mathbf{S} \rangle$ *and let* Φ *be the corresponding SID encoding. Let* $(\mathfrak{s}, \mathfrak{h}) \models_\Phi \mathsf{word}(x, y)$ *and let* $j_1, \ldots, j_m \in \{1, \ldots, n\}$ *be such that*

$$
\begin{aligned}
(\mathfrak{s}, \mathfrak{h}) \models_\Phi \exists n_1, \ldots, n_{m-1}, b_1, \ldots, b_m.\, &((x \mapsto \langle n_1, b_1 \rangle) \star \mathsf{letter}_{j_1}(b_1)) \\
&\star ((n_1 \mapsto \langle n_2, b_2 \rangle) \star \mathsf{letter}_{j_2}(b_2)) \\
&\star \cdots \star ((n_{m-1} \mapsto \langle y, b_m \rangle) \star \mathsf{letter}_{j_m}(b_m)).
\end{aligned}
$$

We define the induced letters *of* $(\mathfrak{s}, \mathfrak{h})$ *and* x, y *as* $\mathsf{letters}(\mathfrak{s}, \mathfrak{h}, x, y) \triangleq a_{j_1} a_{j_2} \cdots a_{j_m}$.

Every model that satisfies $N(x_1, x_2, x_3)$ contains a sub-heap that satisfies the word predicate.

Lemma 13.6. *Let* $\mathbf{G}_1 = \langle \mathbf{N}_1, \mathbf{T}, \mathbf{R}_1, \mathbf{S}_1 \rangle$, $\mathbf{G}_2 = \langle \mathbf{N}_2, \mathbf{T}, \mathbf{R}_2, \mathbf{S}_2 \rangle$ *and let* Φ *be the corresponding SID encoding. Let* $x_1, x_2, x_3 \in \mathbf{Var}$, $N \in \mathbf{N}_1 \cup \mathbf{N}_2$ *and let* $(\mathfrak{s}, \mathfrak{h}) \models_\Phi N(x_1, x_2, x_3) \star \mathsf{t}$. *Then there exists a unique heap* $\mathfrak{h}_w \subseteq \mathfrak{h}$ *with* $(\mathfrak{s}, \mathfrak{h}_w) \models_\Phi \mathsf{word}(x_2, x_3)$.

Proof. A straightforward induction shows that the models of the predicate call $N(x_1, x_2, x_3)$ are trees with linked leaves with root $\mathfrak{s}(x_1)$, leftmost leaf $\mathfrak{s}(x_2)$, and successor of rightmost leaf $\mathfrak{s}(x_3)$. We pick as $\mathfrak{h}_w$ the heap that contains $\mathfrak{s}(x_2)$ as well as all locations reachable from $\mathfrak{s}(x_2)$ in $\mathfrak{h}$. This gives us precisely the list from $\mathfrak{s}(x_2)$ to $\mathfrak{s}(x_3)$. Moreover, every leaf satisfies a formula of the form $\exists a. (y \mapsto \langle z, a \rangle) \star \mathsf{letter}_k(a)$. Consequently, $(\mathfrak{s}, \mathfrak{h}_w) \models_\Phi \mathsf{word}(x_2, x_3)$. □

Lemma 13.6 ensures that the following is well defined.

Definition 13.7 (Induced word). *Let* $\mathbf{G} = \langle \mathbf{N}, \mathbf{T}, \mathbf{R}, \mathbf{S} \rangle$ *and let* Φ *be the corresponding SID encoding. Let* $x_1, x_2, x_3 \in \mathbf{Var}$, $N \in \mathbf{N}$ *and let* $(\mathfrak{s}, \mathfrak{h}) \models_\Phi$

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

$N(x_1, x_2, x_3)$. *Let* $\mathfrak{h}_w \subseteq \mathfrak{h}$ *be the unique heap with* $(\mathfrak{s}, \mathfrak{h}_w) \models_\Phi \mathsf{word}(x_2, x_3)$. *We define the* induced word *of* $(\mathfrak{s}, \mathfrak{h})$ *and* N *as* $\mathsf{wordof}_N(\mathfrak{s}, \mathfrak{h}, x_2, x_3) \triangleq \mathsf{letters}(\mathfrak{s}, \mathfrak{h}_w, x_2, x_3)$.

Every word $w \in \mathcal{L}(\mathbf{G})$ is the induced word of a model of the corresponding SID encoding.

Lemma 13.8 (Completeness of the encoding)**.** *Let* $\mathbf{G} = \langle \mathbf{N}, \mathbf{T}, \mathbf{R}, \mathbf{S} \rangle$ *and let* Φ *be the corresponding SID encoding. Let* $x_1, x_2, x_3 \in \mathbf{Var}$ *and let* $w \in \mathcal{L}(\mathbf{G})$. *Then there exists a model* $(\mathfrak{s}, \mathfrak{h})$ *with* $(\mathfrak{s}, \mathfrak{h}) \models_\Phi \mathbf{S}(x_1, x_2, x_3)$ *and* $\mathsf{wordof}_{\mathbf{S}}(\mathfrak{s}, \mathfrak{h}, x_2, x_3) = w$.

Proof. We show the stronger claim that for all $x_1, x_2, x_3 \in \mathbf{Var}$, $w \in \mathcal{L}(\mathbf{G})$, and $N \in \mathbf{N}$, if $N \Rightarrow^+ w$ then there exists a model $(\mathfrak{s}, \mathfrak{h})$ with $(\mathfrak{s}, \mathfrak{h}) \models_\Phi N(x_1, x_2, x_3)$ and $\mathsf{wordof}_N(\mathfrak{s}, \mathfrak{h}, x_2, x_3) = w$. We proceed by mathematical induction on the number m of $\Rightarrow$ steps in a (minimal-length) derivation $N \Rightarrow^+ w$.

If $m = 1$, $w = a_i$ for some $1 \leq i \leq n$ and there exists a rule $N \to a_i$. We let $(\mathfrak{s}, \mathfrak{h})$ be a model such that $(\mathfrak{s}, \mathfrak{h}) \models_\Phi \exists a. (x_1 \mapsto \langle x_3, a \rangle) \star \mathsf{letter}_i(a) \star x_1 \approx x_2$. Note that this is a rule of the predicate N, so it holds that $(\mathfrak{s}, \mathfrak{h}) \models_\Phi N(x_1, x_2, x_3)$. Moreover, $\mathsf{wordof}_N(\mathfrak{s}, \mathfrak{h}, x_2, x_3) = a_i$.

If $m > 1$, there exists a rule $N \to AB$ such that $N \Rightarrow AB \Rightarrow^+ w$. This implies that there exists words w_A, w_B with $w = w_A \cdot w_B$, $A \Rightarrow^+ w_A$ and $B \Rightarrow^+ w_B$.

Observe that both of these sub-derivations consist of strictly fewer than m steps. Let $l, m, r \in \mathbf{Var}$. By the induction hypotheses, there exist stacks $\mathfrak{s}_1, \mathfrak{s}_2$ and heaps $\mathfrak{h}_1, \mathfrak{h}_2$ such that

- $(\mathfrak{s}_1, \mathfrak{h}_1) \models_\Phi A(l, x_2, m)$ and $\mathsf{wordof}_A(\mathfrak{s}_1, \mathfrak{h}_1, x_2, m) = w_A$.
- $(\mathfrak{s}_2, \mathfrak{h}_2) \models_\Phi A(r, m, x_3)$ and $\mathsf{wordof}_B(\mathfrak{s}_2, \mathfrak{h}_2, m, x_3) = w_B$.

Let $v \in \mathbf{Loc}$. Assume w.l.o.g. that (1) $\mathrm{dom}(\mathfrak{s}_1) \cap \mathrm{dom}(\mathfrak{s}_2) = m$, (2) $\mathfrak{s}_1(m) = \mathfrak{s}_2(m)$, (3) and $\mathfrak{h}_1 + \mathfrak{h}_2 \neq \bot$, and (4) $l \notin \mathsf{locs}(\mathfrak{h}_1 + \mathfrak{h}_2)$—if this is not the case, simply replace $(\mathfrak{s}_1, \mathfrak{h}_1)$ and $(\mathfrak{s}_2, \mathfrak{h}_2)$ with appropriate isomorphic models.

Let $\mathfrak{s} \triangleq \mathfrak{s}_1 \cup \mathfrak{s}_2 \cup \{x_1 \mapsto l\}$ and $\mathfrak{h} \triangleq \mathfrak{h}_1 \cup \mathfrak{h}_2 \cup \{v \mapsto \langle \mathfrak{s}(l), \mathfrak{s}(r) \rangle\}$. We obtain $(\mathfrak{s}, \mathfrak{h}) \models_\Phi (x_1 \mapsto \langle l, r \rangle) \star A(l, x_2, m) \star B(r, m, x_3)$ and thus also $(\mathfrak{s}, \mathfrak{h}) \models_\Phi \exists \langle l, r, m \rangle . (x_1 \mapsto \langle l, r \rangle) \star A(l, x_2, m) \star B(r, m, x_3)$. By definition of Φ, we conclude $(\mathfrak{s}, \mathfrak{h}) \models_\Phi N(x_1, x_2, x_3)$. Furthermore, observe that

$$\begin{aligned}\mathsf{wordof}_N(\mathfrak{s}, \mathfrak{h}, x_2, x_3) =& \mathsf{wordof}_A(\mathfrak{s}, \mathfrak{h}_1, x_2, m) \cdot \mathsf{wordof}_N(\mathfrak{s}, \mathfrak{h}_2, m, x_3) \\ =& w_A \cdot w_B = w.\end{aligned}$$

□

Likewise, every induced word of a model of the corresponding SID encoding is in $\mathcal{L}(\mathbf{G})$.

Lemma 13.9 (Soundness of the encoding)**.** *Let* $\mathbf{G} = \langle \mathbf{N}, \mathbf{T}, \mathbf{R}, \mathbf{S} \rangle$ *and let* Φ *be the corresponding SID encoding. Let* $x_1, x_2, x_3 \in \mathbf{Var}$ *and let* $(\mathfrak{s}, \mathfrak{h})$ *be a model with* $(\mathfrak{s}, \mathfrak{h}) \models_\Phi \mathbf{S}(x_1, x_2, x_3)$. *Then* $\mathsf{wordof}_{\mathbf{S}}(\mathfrak{s}, \mathfrak{h}, x_2, x_3) \in \mathcal{L}(\mathbf{G})$.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Proof. We show the stronger claim that for all $x_1, x_2, x_3 \in \mathbf{Var}$, all models $(\mathfrak{s}, \mathfrak{h})$, and all $N \in \mathbf{N}$, if $(\mathfrak{s}, \mathfrak{h}) \models_\Phi N(x_1, x_2, x_3)$ then $N \Rightarrow^+ \mathsf{wordof}_N(\mathfrak{s}, \mathfrak{h}, x_2, x_3)$. Observe that $\mathfrak{h}$ is a tree overlaid with a linked list. We proceed by mathematical induction on the height h of the tree in $\mathfrak{h}$.

If $h = 0$, there exists a rule $(N(x_1, x_2, x_3) \Leftarrow \exists a.\,(x_1 \mapsto \langle x_3, a\rangle) \star \mathsf{letter}_k(a) \star x_1 \approx x_2) \in \Phi$ such that $(\mathfrak{s}, \mathfrak{h}) \models_\Phi \exists a.\,(x_1 \mapsto \langle x_3, a\rangle) \star \mathsf{letter}_k(a) \star x_1 \approx x_2$. Then $\mathsf{wordof}_N(\mathfrak{s}, \mathfrak{h}, x_2, x_3) = a_k$. By definition of Φ, this implies that $N \to a_k \in \mathbf{R}_1 \cup \mathbf{R}_2$. Consequently, $N \Rightarrow a_k$ and therefore $N \Rightarrow^+ a_k = \mathsf{wordof}_N(\mathfrak{s}, \mathfrak{h}, x_2, x_3)$.

If $h > 0$, there exists a rule $(N(x_1, x_2, x_3) \Leftarrow \psi) \in \Phi$, for $\psi = \exists l, r, m.\,(x_1 \mapsto \langle l, r\rangle) \star A(l, x_2, m) \star B(r, m, x_3)$, such that $(\mathfrak{s}, \mathfrak{h}) \models_\Phi \psi$. Recall that by definition of Φ, $N \to AB \in \mathbf{R}_1 \cup \mathbf{R}_2$ $(\dagger)$.

By the semantics of $\exists$ and $\star$ there then are a stack $\mathfrak{s}'$ with $\mathrm{dom}(\mathfrak{s}') = \mathrm{dom}(\mathfrak{s}) \cup \{l, r, m\}$ and heaps $\mathfrak{h}_0, \mathfrak{h}_A, \mathfrak{h}_B$ such that $\mathfrak{h} = \mathfrak{h}_0 + \mathfrak{h}_A + \mathfrak{h}_B$, $(\mathfrak{s}', \mathfrak{h}_0) \models_\Phi (x_1 \mapsto \langle l, r\rangle)$, $(\mathfrak{s}', \mathfrak{h}_A) \models_\Phi A(l, x_2, m)$, and $(\mathfrak{s}', \mathfrak{h}_B) \models_\Phi B(r, m, x_3)$. Note that the height of the trees in $\mathfrak{h}_A$ and $\mathfrak{h}_B$ is at most $h - 1$, so we can apply the induction hypotheses for these models to obtain

- $A \Rightarrow^+ \mathsf{wordof}_A(\mathfrak{s}', \mathfrak{h}_1, x_2, m)$,
- $B \Rightarrow^+ \mathsf{wordof}_B(\mathfrak{s}', \mathfrak{h}_2, m, x_3)$.

Together with $(\dagger)$, we derive

$$\begin{aligned} N &\Rightarrow AB \\ &\Rightarrow^+ \mathsf{wordof}_A(\mathfrak{s}', \mathfrak{h}_1, x_2, m) \cdot \mathsf{wordof}_B(\mathfrak{s}', \mathfrak{h}_2, m, x_3) \\ &= \mathsf{wordof}_N(\mathfrak{s}', \mathfrak{h}, x_2, x_3) \\ &= \mathsf{wordof}_N(\mathfrak{s}, \mathfrak{h}, x_2, x_3). \end{aligned}$$ □

We are ready to prove the undecidability results. First, we show that adding t (true) to the logic with binary operators $\wedge$ and $\star$ (i.e., the fragment of $\mathbf{SLID}^{\mathsf{g}}_{\mathsf{btw}}$ without disjunction or any of the guarded operators) immediately leads to undecidability.

Theorem 13.10. *The satisfiability problems of* $\mathbf{SLID}_{\mathsf{btw}}(\wedge, \star, \mathsf{t})$ *is undecidable.*

Proof. Let $\mathbf{G}_1 = \langle \mathbf{N}_1, \mathbf{T}, \mathbf{R}_1, \mathbf{S}_1 \rangle, \mathbf{G}_2 = \langle \mathbf{N}_2, \mathbf{T}, \mathbf{R}_2, \mathbf{S}_2 \rangle \in \mathbf{CFG}$ and let Φ be the corresponding SID encoding. I claim that $\phi \triangleq (\mathbf{S}_1(a, x, y) \star \mathsf{t}) \wedge (\mathbf{S}_2(b, x, y) \star \mathsf{t})$ is satisfiable iff $\mathcal{L}(\mathbf{G}_1) \cap \mathcal{L}(\mathbf{G}_2) \neq \emptyset$. We prove the implications separately.

If ϕ is satisfiable, there exists a model $(\mathfrak{s}, \mathfrak{h})$ with $(\mathfrak{s}, \mathfrak{h}) \models_\Phi \phi$. Let $\mathfrak{h}_{w_1}, \mathfrak{h}_{w_2} \subseteq \mathfrak{h}$ be such that $\mathsf{wordof}_{\mathbf{S}_i}(\mathfrak{s}, \mathfrak{h}, x, y) = \mathsf{letters}(\mathfrak{s}, \mathfrak{h}_{w_i}, x, y)$; such heaps exist by Lemma 13.6.

Observe that both $(\mathfrak{s}, \mathfrak{h}_{w_1}) \models_\Phi \mathsf{word}(x, y)$ and $(\mathfrak{s}, \mathfrak{h}_{w_2}) \models_\Phi \mathsf{word}(x, y)$. This implies that both $\mathfrak{h}_{w_1}$ and $\mathfrak{h}_{w_2}$ contain $\mathfrak{s}(x)$ and all locations of the heap $\mathfrak{h}$ that are reachable (in $\mathfrak{h}$) from $\mathfrak{s}(x)$ without going through $\mathfrak{s}(y)$. Consequently, $\mathfrak{h}_{w_1} = \mathfrak{h}_{w_2}$, which implies $w \triangleq \mathsf{wordof}_{\mathbf{S}_2}(\mathfrak{s}, \mathfrak{h}, x, y) =$

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

$\mathsf{wordof}_{\mathbf{S}_1}(\mathfrak{s},\mathfrak{h},x,y)$. By Lemma 13.9, it follows that $w \in \mathcal{L}(\mathbf{G}_1)$ and $w \in \mathcal{L}(\mathbf{G}_2)$, i.e., $w \in \mathcal{L}(\mathbf{G}_1) \cap \mathcal{L}(\mathbf{G}_2)$.

Conversely, assume $\mathcal{L}(\mathbf{G}_1) \cap \mathcal{L}(\mathbf{G}_2) \neq \emptyset$ and let $w \in \mathcal{L}(\mathbf{G}_1) \cap \mathcal{L}(\mathbf{G}_2)$. By Lemma 13.8, there exist models $(\mathfrak{s},\mathfrak{h}_1),(\mathfrak{s},\mathfrak{h}_2)$ with $(\mathfrak{s},\mathfrak{h}_1) \models_\Phi \mathbf{S}_1(a,x,y)$ and $(\mathfrak{s},\mathfrak{h}_2) \models_\Phi \mathbf{S}_2(b,x,y)$. Let $\mathfrak{h}_{w_1} \subseteq \mathfrak{h}_1, \mathfrak{h}_{w_2} \subseteq \mathfrak{h}_2$ be the unique heaps with $\mathsf{wordof}_{\mathbf{S}_1}(\mathfrak{s},\mathfrak{h}_1,x,y) = \mathsf{letters}(\mathfrak{s},\mathfrak{h}_{w_1},x,y) = w = \mathsf{letters}(\mathfrak{s},\mathfrak{h}_{w_2},x,y) = \mathsf{wordof}_{\mathbf{S}_2}(\mathfrak{s},\mathfrak{h}_2,x,y)$.

Observe that $(\mathfrak{s},\mathfrak{h}_{w_1}) \cong (\mathfrak{s},\mathfrak{h}_{w_2})$. Consequently, we can replace $\mathfrak{h}_2$ with an isomorphic heap that contains $\mathfrak{h}_{w_1}$ (as opposed to $\mathfrak{h}_{w_2}$) as sub-heap and is otherwise disjoint from $\mathfrak{h}_1$, i.e., there exists a heap $\mathfrak{h}_2'$ such that $\mathfrak{h}_2 \cong \mathfrak{h}_2'$, $\mathsf{locs}(\mathfrak{h}_2') \cap \mathsf{locs}(\mathfrak{h}_1) = \mathsf{locs}(\mathfrak{h}_{w_1})$, and $\mathsf{wordof}_{\mathbf{S}_2}(\mathfrak{s},\mathfrak{h}_2',x,y) = \mathsf{letters}(\mathfrak{s},\mathfrak{h}_{w_1},x,y) = w$. Note in particular that $(\mathfrak{s},\mathfrak{h}_2') \models_\Phi \mathbf{S}_2(b,x,y)$, because isomorphic models satisfy the same formulas. Now let $\mathfrak{h} \triangleq \mathfrak{h}_1 \cup \mathfrak{h}_2'$ be the (non-disjoint) union of $\mathfrak{h}_1$ and $\mathfrak{h}_2'$. Since $\mathfrak{h}_1 \subseteq \mathfrak{h}$ and $(\mathfrak{s},\mathfrak{h}_1) \models_\Phi \mathbf{S}_1(a,x,y)$, we have $(\mathfrak{s},\mathfrak{h}) \models_\Phi \mathbf{S}_1(a,x,y) \star \mathsf{t}$; and similarly, since $\mathfrak{h}_2' \subseteq \mathfrak{h}$ and $(\mathfrak{s},\mathfrak{h}_2') \models_\Phi \mathbf{S}_2(a,x,y)$, we have that $(\mathfrak{s},\mathfrak{h}) \models_\Phi \mathbf{S}_2(b,x,y)$. Consequently, $(\mathfrak{s},\mathfrak{h}) \models_\Phi \phi$. □

Corollary 13.11. *The satisfiability problem of* $\mathbf{SLID}_{\mathsf{btw}}(\wedge,\star,\neg)$ *is undecidable.*

Proof. this follows directly from the undecidability of $\mathbf{SLID}_{\mathsf{btw}}(\wedge,\star,\mathsf{t})$ (Theorem 13.10), because t is definable in $\mathbf{SLID}_{\mathsf{btw}}(\wedge,\star,\neg)$; for example $\mathsf{t} \triangleq \neg(\mathbf{emp} \wedge \neg\mathbf{emp})$. □

Corollary 13.12. *The satisfiability problem of* $\mathbf{SLID}_{\mathsf{btw}}(\wedge,\star,-\!\!\star)$ *is undecidable.*

Proof. Follows directly from the undecidability of $\mathbf{SLID}_{\mathsf{btw}}(\wedge,\star,\mathsf{t})$ (Theorem 13.10), because t is definable in $\mathbf{SLID}_{\mathsf{btw}}(\wedge,\star,-\!\!\star)$; for example $\mathsf{t} \triangleq (x \not\approx x) -\!\!\star \mathbf{emp}$. □

Our final undecidability proof concerns unguarded septractions. We need one more auxiliary result before we can prove this result.

Lemma 13.13. *Let* $\mathbf{G}_2 = \langle \mathbf{N}_2, \mathbf{T}, \mathbf{R}_2, \mathbf{S}_2 \rangle$, *let* Φ *be the corresponding SID encoding,* $\mathsf{word}_2(x,y) \triangleq (\mathsf{word}(x,y) \circledast \mathbf{S}_2(a,x,y)) \circledast \mathbf{S}_2(a,x,y)$, *and let* $(\mathfrak{s},\mathfrak{h})$ *be a model.* $(\mathfrak{s},\mathfrak{h}) \models_\Phi \mathsf{word}_2(x,y)$ *iff* $(\mathfrak{s},\mathfrak{h}) \models_\Phi \mathsf{word}(x,y)$ *and* $\mathsf{letters}(\mathfrak{s},\mathfrak{h},x,y) \in \mathcal{L}(\mathbf{G}_2)$.

Proof. Let $(\mathfrak{s},\mathfrak{h}) \models_\Phi \mathsf{word}_2(x,y)$. By the semantics of $\circledast$, there exists a heap $\mathfrak{h}_1$ with $(\mathfrak{s},\mathfrak{h}_1) \models_\Phi \mathsf{word}(x,y) \circledast \mathbf{S}_2(a,x,y)$ such that $(\mathfrak{s},\mathfrak{h}+\mathfrak{h}_1) \models_\Phi \mathbf{S}_2(a,x,y)$. Observe that $\mathfrak{h}_1$ contains precisely the inner nodes of the model $(\mathfrak{s},\mathfrak{h}+\mathfrak{h}_1)$, i.e., everything *except* the part of the model that induces the word. Consequently, $\mathfrak{h}$ is the part of the model that induces the word, i.e., $\mathsf{wordof}_{\mathbf{S}_2}(\mathfrak{s},\mathfrak{h}+\mathfrak{h}_1,x,y) = \mathsf{letters}(\mathfrak{s},\mathfrak{h},x,y)$ and $(\mathfrak{s},\mathfrak{h}) \models_\Phi \mathsf{word}(x,y)$. Lemma 13.9 then yields $\mathsf{letters}(\mathfrak{s},\mathfrak{h},x,y) \in \mathcal{L}(\mathbf{G}_2)$.

Conversely, let $(\mathfrak{s},\mathfrak{h})$ be such that $w \triangleq \mathsf{letters}(\mathfrak{s},\mathfrak{h},x,y) \in \mathcal{L}(\mathbf{G}_2)$. As a consequence of Lemma 13.8, there exists a heap $\mathfrak{h}_1$ with $(\mathfrak{s},\mathfrak{h}+$

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

$\mathfrak{h}_1) \models_\Phi \mathbf{S}_2(a,x,y)$. Because $(\mathfrak{s},\mathfrak{h}) \models_\Phi \mathsf{word}(x,y)$ by assumption, we have by the semantics of $\circledast$ that $(\mathfrak{s},\mathfrak{h}_1) \models_\Phi \mathsf{word}(x,y) \circledast \mathbf{S}_2(a,x,y)$. Because $(\mathfrak{s},\mathfrak{h}+\mathfrak{h}_1) \models_\Phi \mathbf{S}_2(a,x,y)$, we obtain by the semantics of $\circledast$ that $(\mathfrak{s},\mathfrak{h}) \models_\Phi (\mathsf{word}(x,y) \circledast \mathbf{S}_2(a,x,y)) \circledast \mathbf{S}_2(a,x,y)$. □

Theorem 13.14. *The satisfiability problem of* $\mathbf{SLID}_{\mathsf{btw}}(\circledast)$ *is undecidable.*

Proof. I claim that $\psi \triangleq \mathsf{word}_2(x,y) \circledast \mathbf{S}_1(a,x,y)$ is satisfiable iff $\mathcal{L}(\mathbf{G}_1) \cap \mathcal{L}(\mathbf{G}_2) \neq \emptyset$. Intuitively, this holds because ψ is satisfiable iff it is possible to replace the "word part" of a model of $\mathbf{S}_1(a,x,y)$ with the "word part" of a model of $\mathbf{S}_2(b,x,y)$. Let us formalize this intuition.

Assume ψ is satisfiable and let $(\mathfrak{s},\mathfrak{h})$ be such that $(\mathfrak{s},\mathfrak{h}) \models_\Phi \psi$. By the semantics of $\circledast$, we have that there exists a heap $\mathfrak{h}_0 \subseteq \mathfrak{h}$ with $\mathfrak{h}_0 \models_\Phi \mathsf{word}_2(x,y)$ and $(\mathfrak{s},\mathfrak{h}+\mathfrak{h}_0) \models_\Phi \mathbf{S}_1(a,x,y)$. As $\mathsf{letters}(\mathfrak{s},\mathfrak{h}_0,x,y) \in \mathcal{L}(\mathbf{G}_2)$ by Lemma 13.13, we have in particular that $(\mathfrak{s},\mathfrak{h}_0) \models_\Phi \mathsf{word}(x,y)$. It follows that $\mathfrak{h}_0$ is the unique sub-heap of $\mathfrak{h}+\mathfrak{h}_0$ with $\mathsf{wordof}_{\mathbf{S}_1}(\mathfrak{s},\mathfrak{h}+\mathfrak{h}_0) = \mathsf{letters}(\mathfrak{s},\mathfrak{h}_0,x,y)$. By Lemma 13.9, $\mathsf{letters}(\mathfrak{s},\mathfrak{h}_0,x,y) \in \mathcal{L}(\mathbf{G}_1)$. Together with Lemma 13.13, we thus have that $\mathsf{letters}(\mathfrak{s},\mathfrak{h}_0,x,y) \in \mathcal{L}(\mathbf{G}_1) \cap \mathcal{L}(\mathbf{G}_2)$.

Conversely, assume there exists a word $w \in \mathcal{L}(\mathbf{G}_1) \cap \mathcal{L}(\mathbf{G}_2)$. By Lemma 13.8, there exist heaps $\mathfrak{h},\mathfrak{h}_0,\mathfrak{h}',\mathfrak{h}'_0$ with

$$(\mathfrak{s},\mathfrak{h}) \models_\Phi \mathbf{S}_1(a,x,y),$$
$$\mathsf{wordof}_{\mathbf{S}_1}(\mathfrak{s},\mathfrak{h},x,y) = \mathsf{letters}(\mathfrak{s},\mathfrak{h}_0,x,y),$$
$$(\mathfrak{s},\mathfrak{h}') \models_\Phi \mathbf{S}_2(a,x,y), \text{ and}$$
$$\mathsf{wordof}_{\mathbf{S}_2}(\mathfrak{s},\mathfrak{h}',x,y) = \mathsf{letters}(\mathfrak{s},\mathfrak{h}'_0,x,y).$$

Because $\mathsf{letters}(\mathfrak{s},\mathfrak{h}_0,x,y) = \mathsf{letters}(\mathfrak{s},\mathfrak{h}'_0,x,y)$, it holds that $\mathfrak{h}_0 \cong \mathfrak{h}'_0$, so we can assume w.l.o.g. that $\mathfrak{h}_0 = \mathfrak{h}'_0$—if the models are not isomorphic, simply replace $\mathfrak{h}'$ with an appropriate isomorphic heap to establish this property. Let $\mathfrak{h}_2 \subseteq \mathfrak{h}'$ be the sub-heap of $\mathfrak{h}'$ with $\mathfrak{h}_2 + \mathfrak{h}_0 = \mathfrak{h}'$. By Lemma 13.13, $(\mathfrak{s},\mathfrak{h}_0) \models_\Phi \mathsf{word}_2(x,y)$. Consequently, $(\mathfrak{s},\mathfrak{h}_2) \models_\Phi \psi$, i.e., ψ is satisfiable. □

13.2 DISCUSSION

We now have a good picture of the decidability landscape for separation logics built on top of $\mathbf{ID}_{\mathsf{btw}}$ SIDs:

- As we saw in Chapter 12, we can decide $\mathbf{SLID}^{\mathsf{g}}_{\mathsf{btw}}$, in which any combination of the operators $\star$, $\wedge$ and $\vee$ and guarded occurrences of the operators $\neg$, $\mathrel{-\!\!\star}$, and $\circledast$ are allowed.

- As we saw in Section 13.1, all extensions of $\mathbf{SLID}^{\mathsf{g}}_{\mathsf{btw}}$ in which one of the guards is removed are undecidable.

It is, of course, also conceivable to remove some of the operators from $\mathbf{SLID}^{\mathsf{g}}_{\mathsf{btw}}$ prior to removing the guards. I did this to some extent for all

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

of the undecidability proofs in Section 13.1, but leave open the question whether there are restrictions of $\mathbf{SLID}_{\mathsf{btw}}(\wedge, \star, \mathsf{t})$, $\mathbf{SLID}_{\mathsf{btw}}(\wedge, \star, \neg)$ and $\mathbf{SLID}_{\mathsf{btw}}(\wedge, \star, \mathrel{-\!\!\star})$ that remain undecidable.

DECIDABILITY AND DANGLING POINTERS. What is the fundamental difference between the logics studied in Section 13.1 and the guarded separation logic $\mathbf{SLID}^{\mathsf{g}}_{\mathsf{btw}}$ that explains the undecidability of the former and the decidability of the latter?

If you examine the undecidability proofs, you will see that all of them rely on the possibility to decompose the heap into parts with unboundedly many dangling pointers. Specifically, by splitting the CFG encoding into the "non-word" part of the derivation and the induced word w introduces $|w|$ dangling pointers.

Conversely, the number of dangling pointers in the models of $\mathbf{SLID}^{\mathsf{g}}_{\mathsf{btw}}$ formulas is always bounded by the number of free variables of the formula—a simple consequence of Lemma 8.12.

I conjecture that generalizations of $\mathbf{SLID}_{\mathsf{btw}}$ that retain this property remain decidable. Such generalizations would still have the property that all models have bounded treewidth. Other interesting questions include: Is it possible to change the restrictions of the $\mathbf{SLID}_{\mathsf{btw}}$ fragment to capture exactly all MSO-definable models of bounded treewidth? And are there, perhaps, even decidable SLs that admit models of unbounded treewidth? Both of these questions remain open at the time of writing.

STRONG-SEPARATION SEMANTICS. Another research direction would be to give $\star$ the strong-separation semantics from Part ii and investigate whether we can remove the guardedness restrictions under this semantics. This would rule out the introduction of an unbounded number of dangling pointers and thus invalidate the undecidability proofs from Section 13.1. It makes sense to try and adapt the "garbage-chunk" idea from Part ii, but because of the possible ambiguity in the SIDs in $\mathbf{ID}_{\mathsf{btw}}$, there is *no* analog of decomposing the "non-garbage part" of the model in a unique way, i.e., no unique way to split a model into positive chunks and negative chunks.

DATA CONSTRAINTS. In my opinion, the most interesting type of extension would be to add some way to combine the shape constraints imposed by the $\mathbf{ID}_{\mathsf{btw}}$ predicates with data constraints. Not much is known about the decidability of SLs that combine user-defined predicates with data constraints; see Chapter 14. One approach would be to augment $\mathbf{ID}_{\mathsf{btw}}$ predicates with a variant of the data constraints from Part ii. I refrain from speculating about the decidability status of such an extension.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Part IV

CONCLUSION

I discuss related work and conclude.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.

14 RELATED WORK

In Parts ii and iii, we developed decision procedures for several fragments of separation logic. In particular:

1. a PSpace decision procedure for entailment in the separation logic **SSL**,
2. a coNP decision procedure for entailment in the separation logic $\mathbf{SSL}^{+}_{\mathsf{data}}$, and
3. a 2ExpTime decision procedure for entailment in the separation logic $\mathbf{SLID}^{\mathsf{g}}_{\mathsf{btw}}$.

In this chapter, I compare this result to (un-)decidability results from the literature. I begin with "shape-only" logics (such as $\mathbf{SLID}^{\mathsf{g}}_{\mathsf{btw}}$) in Section 14.1 and continue with logics that mix reasoning about shape and data (such as $\mathbf{SSL}_{\mathsf{data}}$) in Section 14.2.

Given the extent of the separation-logic literature in particular and the literature about "heap logics" in general, this chapter is necessarily incomplete. For this reason, I will provide a deliberately incomplete overview.

14.1 LOGICS FOR REASONING ABOUT THE SHAPE OF THE HEAP

SHAPE ANALYSIS. Much work on separation logic falls under the purview of *shape analysis*; see for example [DOY06; Ber+07; Cal+07; Yan+08; Cal+11; DES13; Le+14]. Other logic-based approaches to shape analysis have been proposed in the literature. I consider this work beyond the scope of this thesis, but recommend the influential work by Sagiv et al. [SRW02] as a starting point.

FIRST-ORDER LOGIC (FO) AND SMT. In FO, transitive closure cannot be axiomatized. Consequently, it is impossible to reason about the reachability in unbounded data structures [Lev+09]. This impossibility result can be avoided by working in fragments such as EPR [Itz+13] or by translating from a logic that has a small-model property [LQ08]. The latter approach was used for several SL dialects [PR13; PWZ13; PWZ14a; KJW18a], including $\mathbf{SSL}^{+}_{\mathsf{data}}$ (cf. Chapter 7). I will further discuss this work in Section 14.2.

MONADIC SECOND-ORDER LOGIC (MSO). When viewing the heap as a directed graph, as we have done throughout this thesis, it is quite

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

natural to use MSO over graphs to reason about the heap [MS01]. Unlike in FO, unbounded reachability is expressible in MSO. Moreover, MSO over graphs of bounded treewidth is decidable [Cou90], albeit with nonelementary complexity [SM73]; if quantifier alternation is bounded, the complexity is elementary. It is also possible to automate reasoning about some structures of unbounded treewidth by combining MSO with a fragment of FO with counting quantifiers [KVZ16].

Separation logic itself has a (monadic) second-order "flavor". The separating conjunction $\phi \star \psi$ holds in a heap if there exists a set of locations—i.e., a monadic relation—such that ϕ holds in the sub-heap corresponding to the set of locations and ψ holds in the remainder of the heap. As such, it is not surprising that reductions of separation logics to MSO are possible [IRS13]. In fact, there is a close relationship between *graph grammars* in the MSO world and systems of inductive definitions in SLID [DP08; Jan17]. Note, however, that not all separation logics are second-order logics: it is, for example, possible to translate SL without inductive predicates to first-order logic [EIP19b].

SYMBOLIC-HEAP SEPARATION LOGIC. The symbolic-heap fragment goes back to the `Smallfoot` [BCO05a] tool, which implemented symbolic execution based on symbolic heaps [BCO05b]. The original symbolic-heap fragment had a built-in singly-linked list predicate and no support for other data structures [BCO04]. Entailment is this fragment was later shown to be decidable in PTIME in the quantifier-free case and in coNP for existentially-quantified formulas [Coo+11].

There is a large body of work on extensions of the symbolic-heap fragment with user-defined inductive definitions, i.e., the logic **SLID** studied in Part iii. If arbitrary (existentially-quantified) symbolic heaps can be used in the inductive definitions, satisfiability and model checking are EXPTIME-complete [Bro+14; Bro+16] and entailment is undecidable [Ant+14].

Various fragments have been proposed, including:

- The logic $\mathbf{SLID}_{\mathrm{btw}}$ that we studied in Part iii. The results in this thesis together with a recent hardness proof [EIP20] imply that $\mathbf{SLID}_{\mathrm{btw}}$ is 2EXPTIME-complete.

- An EXPTIME-complete separation logic without dis-equalities, but with inductive definitions that are general enough to model doubly-linked lists and trees [IRV14]. This logic is a fragment of $\mathbf{SLID}_{\mathrm{btw}}$.

- The fragment with monadic definitions and so-called *implicit existentials* [TK15]. This fragment allows data structures with an arbitrary number of dangling pointers (via implicit existentials), but only a single "structural parameter" that can be used to build the structure. This is sufficient for defining singly-linked lists and trees, but not for doubly-linked structures. The logic was

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

proved decidable, but no complexity bound was given. This logic is not a fragment of $\mathbf{SLID}_{\mathrm{btw}}$, because $\mathbf{SLID}_{\mathrm{btw}}$ does not admit implicit existentials. When implicit existentials are disallowed, we obtain a fragment of $\mathbf{SLID}_{\mathrm{btw}}$

- Separation logic with *cone inductive definitions* [TNK19] is a restriction of $\mathbf{SLID}_{\mathrm{btw}}$. In this logic, existentially-quantified variables in inductive rules must be allocated immediately in a recursive call, thus strengthening the establishment property. The *inductive wands* used in [TNK19] can be viewed as the quantifier-free special case of the DUSH fragment we saw in Chapter 11. The proof-theoretic decision procedure in [TNK19] runs in 2NExpTime.

Besides the aforementioned complete decision procedures, various incomplete methods for proving and dis-proving SLID entailments have been proposed and implemented, for example in `Cyclist`, `Spen`, and `Songbird` [BGP12; BG15; DES13; Ene+14; Ene+17; Ta+18].

Tools for semi-automated verification, such as `Verifast` [Jac+11], also work with (undecidable) assertion languages that are related to SLID.

BEYOND SYMBOLIC HEAPS. The propositional fragment of separation logic without *any* inductive definitions, i.e., with no spatial atoms besides points-to assertions, was shown to be PSpace-complete in [CYO01]. It was later reduced to first-order logic [CGH05] and implemented in the SMT solver `Cvc4` [Rey+16].

Adding the singly-linked list predicate to this basic logic already leads to undecidability [DLM18]—assuming the standard "weak" semantics of the separating conjunction. This result was our main motivation to investigate strong-separation logic, which—as we saw in Part ii—remains decidable in PSpace even when list and tree predicates are added to the logic.

Similarly, an extension of propositional SL with an exists–forall quantifier prefix also leads to undecidability [EIP19a] (despite earlier claims to the contrary [RIS17]).

More generally, extensions with even very limited forms of quantification lead to undecidability or non-elementary complexity very quickly [BDL12; DD15b; DD16; Dem+17; EIP19b].

The decidability status of SSL with quantifiers is open at the time of writing.

BEYOND INDUCTION. In a sense, Part iii shows "how far we can go" with inductive definitions. While some further generalization of $\mathbf{ID}_{\mathrm{btw}}$ is likely possible without sacrificing decidability, it is worth pointing out that the inductive approach has inherent limitations. Not every data structure can be described inductively. This is especially true in a concurrent setting. Krishna et al. have recently proposed

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

flows as an alternative to inductive definitions for circumventing these limitations [KSW18; KSW20].

14.2 LOGICS FOR REASONING ABOUT SHAPE AND CONTENT

SATISFIABILITY MODULO THEORIES. Several heap logics rely on a reduction to SMT [LQ08; MPQ11; PR13; PWZ13; PWZ14a]. Among these, [PR13; PWZ13; PWZ14a] are most closely related to $\mathbf{SSL}^{+}_{\mathsf{data}}$.

Navarro Pérez et al. [PR13] only support the list-segment predicate, not trees. Moreover, they do not have any support for constraining the data stored inside a data structure (as opposed to the data stored at individual locations, which can easily be handled thanks to the reduction to SMT). They can, however, reason about the length of list segments, which is not supported by $\mathbf{SSL}^{+}_{\mathsf{data}}$.

Piskac et al. [PWZ13; PWZ14a] propose the logics `Grass` and `Grit`. Combined, these logics are quite similar to $\mathbf{SSL}^{+}_{\mathsf{data}}$. There are several subtle distinctions. First, $\mathbf{SSL}^{+}_{\mathsf{data}}$ supports tree segments as opposed to complete binary trees. Second, the SMT encoding of trees for $\mathbf{SSL}^{+}_{\mathsf{data}}$ does not require ghost parent pointers. Third, the encoding of $\mathbf{SSL}^{+}_{\mathsf{data}}$ does not need the "heavy machinery" of local theory extensions to deal with data constraints. Fourth, `Grass` and `Grit` can express certain properties that cannot be expressed in $\mathbf{SSL}^{+}_{\mathsf{data}}$. In particular, $\mathbf{SSL}^{+}_{\mathsf{data}}$ can only express that certain individual elements must be stored inside a data structure; it cannot be used to reason about the complete set of elements stored in a data structure.

DATA CONSTRAINTS IN RECURSIVE DEFINITIONS. A key difference between $\mathbf{SSL}_{\mathsf{data}}$ and most other SLs with data is that data constraints in $\mathbf{SSL}_{\mathsf{data}}$ are orthogonal to shape constraints: we do not allow data constraints inside inductive definitions. Instead, we allow annotating predicate calls with data predicates (see Section 5.2). A similar approach can be found in the `Strand` logic [MPQ11]. `Strand` is undecidable in general. While `Strand` is not a separation logic, the decidable fragment identified in [MPQ11] has a similar expressiveness to $\mathbf{SSL}^{+}_{\mathsf{data}}$, but involves an encoding to both MSO (for the shape constraints) and SMT (for the data constraints).

There is an extensive body of work on allowing data constraints within user-defined inductive definitions. This was done in the logic `Dryad` [Qiu+13], but also in variants of SLID [Chi+12; TLC16; Le+17; Ta+18]. In all of these logics, the entailment problem is either known to be undecidable or not known to be decidable.

If we restrict inductive data structures to lists, several decidable variants have been proposed. For example, lists with ordered data can be handled by [BBL09; Bro13]. The separation logic in [GCW16] supports user-defined list predicates with a limited form of arithmetic constraints that have a similar expressiveness to our data predicates

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

when $\mathcal{T}_{\mathbf{Data}}$ is instantiated with linear integer arithmetic. Finally, the formalism in [GCW19] can be used to reason about set constraints (not supported by $\mathbf{SSL}_{\mathrm{data}}$) as well as limited arithmetic constraints.

POINTER ARITHMETIC. Another way to go "beyond shape" is to support reasoning about pointer arithmetic. This extension is studied for SL without inductive definitions in [BK18]. More pragmatic work in this direction (without decidability or complexity results) can, for example, be found in [Cal+06; Hol+16].

14.3 TOOL SUPPORT

As mentioned before, separation logics have been used in a wide range of program analysis and program verification tools. I will give a brief, deliberately incomplete overview of this work and how it relates to the results presented in this thesis.

A significant number of tools for deductive program verification are based on separation logic or closely related formalisms. These tools include `GRASShopper` [PWZ14b] (for a custom intermediate language), `Viper` [MSS16] (for a custom intermediate language, to which multiple high-level languages have been translated), and `Verifast` [Jac+11] (for C and Java programs). In such tools, the code has to be annotated with preconditions, postconditions and loop invariants written in an assertion language that is a variant of separation logic. Program correctness is then proved automatically from these annotations. Such (semi-)automated proofs require provers for the assertion language. Exploiting the decision procedures developed in Parts ii and iii of this thesis, both **SSL** and $\mathbf{SLID}^{\mathrm{g}}_{\mathrm{btw}}$ could be used as assertion language in deductive verification tools. In the case of concurrent separation logics, **SSL** and $\mathbf{SLID}^{\mathrm{g}}_{\mathrm{btw}}$ could be used as underlying sequential base logics.

Separation logics have also been very successfully employed for finding bugs (as opposed to proving their absence). Arguably the most prominent tool that takes this approach is Facebook's `Infer` [CD11; Cal+15], which is based in part on separation logic. `Infer` relies on a heuristic prover for bi-abduction, i.e., a prover that computes for inputs of the form $\phi \star [?] \models \psi \star [?]$ formulas ϕ', ψ' such that the entailment $\phi \star \phi' \models \psi \star \psi'$ is valid. Thanks to their compositional nature, the decision procedures for **SSL** and $\mathbf{SLID}^{\mathrm{g}}_{\mathrm{btw}}$ can be extended to bi-abduction and could thus, at least in principle, be used as underlying prover in `Infer`. This would, for example, enable `Infer` to reason about all data structures definable in $\mathbf{SLID}^{\mathrm{g}}_{\mathrm{btw}}$—albeit at a high computational cost.

SSL also opens up new possibilities for tool development: thanks to the decidability of the magic wand in **SSL**, implementing weakest-precondition calculi for separation logic as developed in [IO01; Rey02] becomes feasible.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.

15 CONCLUSION

This thesis has been a deep dive into automated reasoning for separation logic.

First, I proposed a new separation logic, *strong-separation logic with lists, trees, and data*, $\mathbf{SSL}_{\mathsf{data}}$, that combined a nonstandard semantics of the separating conjunction with a novel approach for constraining the data stored within data structures. $\mathbf{SSL}_{\mathsf{data}}$ is expressive enough to enable a combined shape–value analysis about important data structures such as binary search trees and max heaps. I showed that despite this expressiveness, the full propositional fragment including the magic wand and negation, but without data constraints, is decidable in PSpace; and that entailment in the positive fragment of $\mathbf{SSL}_{\mathsf{data}}$ (subsuming the symbolic-heap fragment) is decidable in coNP. $\mathbf{SSL}_{\mathsf{data}}$ is thus more tractable than many other logics that combine reasoning about shape and data [MPQ11; Chi+12; Qiu+13; Le+17].

Second, I designed a novel decision procedure for a large fragment of *separation logic with inductive definitions*, $\mathbf{SLID}_{\mathsf{btw}}$ [IRS13]. $\mathbf{SLID}_{\mathsf{btw}}$ is expressive enough to enable reasoning about intricate data structures such as trees with linked leaves. I developed the first direct decision procedure for this logic. This decision procedure avoids a blowup of multiple exponentials compared to the only previously known decision procedure [IRS13]; it still takes double-exponential time in the worst case, but a recent 2ExpTime-hardness proof shows that this is unavoidable [EIP20].

In Chapter 2, I argued that research into fragments of separation logic is to a large extent about finding the right trade-off between expressiveness and tractability. Whether or not I struck the right balance with the work in this thesis is debatable (and can ultimately only be revealed by an extensive evaluation). At any rate, the results in this thesis show that it is possible to design expressive fragments of separation logic without incurring super-exponential complexity ($\mathbf{SSL}_{\mathsf{data}}$) or relying on reductions that blow up the size of the input by many exponentials ($\mathbf{SLID}_{\mathsf{btw}}$).

IMPLEMENTATION. In this thesis, I have focused almost exclusively on theoretical results. While I hope I succeeded in conveying interesting theoretical insights, I also hope that some of these insights can be put into practice—i.e., that the decision procedures developed in this thesis find their way into mature tools.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Given the abysmal worst-case performance of all decision procedures presented here, this may seem like a pointless endeavor. I am more optimistic. First, pretty much every decision procedure implemented in SMT solvers has a seemingly prohibitive worst-case performance. Nevertheless, SMT solvers can handle very large inputs in practice. Second, I have already implemented special cases of the decision procedures I presented in this thesis, which resulted in two open-source[1] prototypes with encouraging performance.

HARRSH [Kat+18] implements the heap automaton framework introduced in [Jan+17], including decision procedures for all the robustness properties studied in that paper. It also implements the Φ-*profile* abstraction for entailment checking, which we developed in [KMZ19a; KMZ19b] and which is the precursor of the Φ-*type* abstraction presented in Part iii. For the reasons explained in Part iii, `Harrsh` cannot handle the entire $\mathbf{SLID}_{\mathrm{btw}}$ fragment. For this reason, I chose not to present an evaluation of the tool in this thesis. The tool can, however, handle a large fragment of $\mathbf{SLID}_{\mathrm{btw}}$ by an approach that is closely related to the one in Part iii; and it fares fairly well in practice despite its double-exponential asymptotic complexity, as was revealed at the competition *SL-COMP'19* [Sig+19].

`Harrsh` could thus serve as a starting point for implementing an entailment checker based on the ϕ-types of Part iii.

SLOTH is a satisfiability and entailment checker for $\mathbf{SL}^{*}_{\mathrm{data}}$ formulas [KJW18a]. `Sloth` works by encoding $\mathbf{SL}^{*}_{\mathrm{data}}$ to SMT using an encoding that is very similar to the SMT encoding of $\mathbf{SSL}^{+}_{\mathrm{data}}$ presented in Part ii. An initial evaluation[2] provided evidence that solving queries involving, for example, binary search trees is feasible.

`Sloth` could thus serve as a starting point for an implementation of the decision procedures for $\mathbf{SSL}^{+}_{\mathrm{data}}$ and $\mathbf{SSL}$.

While I cannot provide direct evidence of the practical merit of the decision procedures as formulated in Parts ii and iii, I believe that the performance of `Harrsh` and `Sloth` merits further work on tool development for both $\mathbf{SLID}^{\mathrm{g}}_{\mathrm{btw}}$ and $\mathbf{SSL}_{\mathrm{data}}$.

FUTURE WORK. Besides developing solvers for the logics $\mathbf{SLID}^{\mathrm{g}}_{\mathrm{btw}}$ and (fragments of) $\mathbf{SSL}_{\mathrm{data}}$, there are several other promising avenues for further practical research.

SYMBOLIC EXECUTION. Abstract memory states and extended abstract memory states as introduced in Part ii are suitable as

1 see https://github.com/katelaan/

2 Not formally published, but available in the informal proceedings of the First Workshop on Automated Deduction for Separation Logics.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Wien Bibliothek Your knowledge hub

abstract domains for a symbolic-execution engine. It is straightforward to formulate the standard symbolic-execution rules for separation logic [BCO05b] in terms of AMS as opposed to symbolic heaps. It would be very interesting to compare the scalability and precision of this approach against the approach based on symbolic heaps.

WEAKEST PRECONDITIONS. Weakest-precondition calculi for separation logic make use of the magic wand [IO01; Rey02], which has made it difficult to automate them [App14]. Thanks to the decidability of the magic wand in **SSL**, it would be feasible to implement a program verifier based on weakest preconditions that uses an **SSL** solver as backend.

BI-ABDUCTION. Thanks to their compositional nature, both the decision procedures for **SSL** and for $\mathbf{SLID}^{g}_{btw}$ can be extended to the abduction, frame inference, and bi-abduction problems as studied, e.g., in [Cal+11; GKO11; LSQ18]. Whether these extensions are feasible in practice can only be revealed by an implementation.

If I were to add another year to my PhD, I would start working through the above list.

At the same time, many interesting theoretical research questions about **SSL** and **SLID** remain unanswered. For example, one could look into:

- extending the PSPACE decision procedure for **SSL** to $\mathbf{SSL}_{data}$ (perhaps under some mild semantic assumptions on the data predicates);
- adding quantifiers to **SSL**;
- adding data predicates to (fragments of) $\mathbf{SLID}_{btw}$;
- studying an unguarded extension of $\mathbf{SLID}^{g}_{btw}$ assuming strong-separation semantics, either with arbitrary $\mathbf{ID}_{btw}$ SIDs or with further SID restrictions, for example to “deterministic” SIDs as defined in [Bro+16];
- capturing monadic-second order logic over (connected) graphs of bounded treewidth by an extension of $\mathbf{ID}_{btw}$.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.

INDEX

abduction, 219
abstract edge relation, 60
abstract magic wand, 72
abstract memory state, 58, 61
abstraction, 57
  for satisfiability checking, 27
  of $\mathbf{SLID}^{g}_{\mathrm{btw}}$ formulas, 164
  of **SSL** formulas, 64
access path ordering, 33
acyclicity, 32
adjoint, 46
allocated location, 19
allocated variables, 19
  of a Φ-type, 164
  of an AMS, 62
α-equivalence, 151
AMS, *see* abstract memory state
AMS composition, 67
AMS computation, 70
  correctness of, 75
arity, 110
array theory, 91
automated reasoning, 4, 17

background theory, 39
bi-abduction, 215, 219
binary data predicate, 53
binary tree
  SID defining, 112
binary-tree segment
  in SSL, 40
`Boolector`, 91
bound-lifting, 75
bounded treewidth, 26, 115
BTW, *see* bounded treewidth

CFG, *see* context-free grammar
chunk, 58
chunk size, 73
compatibility
  of AMSs, 67
  of SMT models, 103

complexity
  of Φ-type computation, 198
  of $\mathbf{SLID}^{g}_{\mathrm{btw}}$, 199
  of $\mathbf{SSL}^{+}_{\mathrm{data}}$, 90
  PSpace-hardness of **SSL**, 79
  PSpace-membership of **SSL**, 80
composition
  of AMSs, 67
  of Φ-types, 166, 168
compositional abstraction, 27
compositional verification, 12
compositionality, 28
  for $\mathbf{SLID}^{g}_{\mathrm{btw}}$, 168
  for **SSL**, 67
concolic execution, 12
cone inductive definitions, 213
connected component, 36
connected list segment, 35
connectivity, 116
context-free grammar, 201
correctness
  of AMS computation, 75
  of Φ-type computation, 195
corresponding SMT model, 93
`Cvc4`, 213
cycle, 35
cyclic data structures, 45
`Cyclist`, 213

`Dafny`, 4
dangling locations, 19
dangling pointers, 23
  in $\mathbf{SLID}_{\mathrm{btw}}$, 118
data predicate, 21, 24, 53
  restrictions on, 55
data structure
  in SSL, 40
data theory, 39
data variable, 43
decidability
  of $\mathbf{SLID}^{g}_{\mathrm{btw}}$, 199

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

decomposability
  into chunks, 59
  of AMS, 68
  of Φ-types, 175
delimited unfolded symbolic heap, 162
depth
  of a tree, 33
derivability
  between projections, 149
  between Φ-forests, 134
derivation tree
  of context-free grammar, 202
directed graph, 31
  induced by Φ-tree, 130
directed indexed graph, 31
directed tree
  with holes, 35
  with root and terminal, 33
disequality, 13, 110
doubly-linked list, 41
`Dryad`, 214
DUSH, 162

empty-heap predicate, 13
entailment problem, 9, 17
  of **SLID**, 115
  of **SSL**, 49
equality, 13, 110
establishment, 116, 180
existential data predicate, 53

false, 42
field, 40
field identifier, 25
first-order logic, 211
fixed-point computation, 184
flow, 214
forests, 131
  Φ-forest, 131
  derivation between, 134
  of a heap, 157
  projection, 126, 140
  union of, 133
forget
  a variable of a Φ-type, 170
frame rule, 10
frame inference, 219
frame problem, 10
frame rule
  of separation logic, 12
free variables
  of predicate, 112
  of SL formula, 43

garbage-chunk count, 62
garbage-free AMS, 64
generalized modus ponens, 138
graph grammar, 212
graphs of bounded treewidth, 115
`Grass`, 214
`GRASShopper`, 215
`Grit`, 214
guarded magic wand, 110
guarded model, 117
guarded negation
  in **SLID**, 110
  in **SSL**, 42
guarded separation logic, 110
guarded septraction, 110
guardedness, 119
guess-and-check procedure, 83

`Harrsh`, 218
heap, 14
heap automaton, 109
heap interpretation, 83
Hoare logic, 8
Hoare triple, 8
hole
  in **SSL**, 41
  of Φ-tree, 124, 130
hole predicate, 124, 130
homomorphism
  of AMS abstraction, 67, 72
  of forest projection, 155
  of Φ-type abstraction, 168
hyperedges
  of an AMS, 62

implicit existential, 212
indexed entailment, 49
induced AMS, 63
induced graph
  of a heap, 31

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

induced heap
  of Φ-forest, 131
induced hyperedge, 62
induced letters, 203
inductive definition, 112
inductive wand, 157, 213
`Infer`, 215
interface, 158
isomorphism
  in **SLID**, 115
  in **SSL**, 47
  of models, 19

join point, 60

l-split, 133
labeled location, 19
language intersection
  of context-free languages, 202
lasso, 35
length
  of a list, 33
  of a path, 32
list segment
  in SSL, 40
  user-defined, 112
local allocation, 116
local analysis, 12
local references, 116
local theory extension, 214
location
  in formulas, 111
location variable, 43

magic wand, 13
model, 15
model checking
  for SSL, 80
  for $\mathbf{SSL}^{+}_{\mathtt{data}}$, 90
model correspondence
  of SMT model, 93
model-size bound, 87
modular analysis, 12
modus ponens, 138
monadic second-order logic, 26, 115, 211
MSO, *see* monadic second-order logic

negation
  guarded, 110
negative chunk, 60
negative-allocation constraint, 62
node signature, 40
nodes
  of an AMS, 62
nonterminal, 201
null pointer, 14

parameter
  of a predicate, 112
partial unfolding tree, 124
path
  in directed graph, 32
per-field allocation, 83
Φ-forest, 131
Φ-tree, 123, 130
Φ-type composition, 168
pointer arithmetic, 15
pointer-closed, 117
points-to assertion, 13
  intuitionistic, 42
  of SSL, 41
positive chunk, 60
positive model, 50
positive SSL, 41
precise, 44, 89, 113
predecessor, 32
predicate identifier, 110
production rule, 201
progress, 116
proof tree, 9
pure, 13
pure constraint, 111

QBF, *see* quantified Boolean formulas
quantified Boolean formulas, 79

re-scoping, 148
reachability
  in directed graphs, 32
realizability size
  of AMS, 84
  of $\mathbf{SSL}_{\mathtt{data}}$ formula, 86
referenced location, 19
referenced variables, 19
refined refinement theorem, 73

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

refinement, 27
refinement theorem
  of Φ-types, 177
  of AMS, 69
  of **SSL**, 65
related work, 211
rewriting equivalence, 137
robustness property, 109
root
  of directed tree, 33
  of predicate, 116
rule
  of SID, 112
rule instance, 129
rule of consequence, 9

𝔰-decomposition, 160
𝔰-delimited, 159
𝔰-types, 164
satisfiability problem, 17
  of SLID, 115
semantics
  of $\mathbf{SL_{base}}$, 15
  of **SLID**, 113
  of **SSL**, 44
  of $\mathbf{SSL}_{\mathtt{data}}$, 55
separating connective, 13
separating conjunction, 3, 11
separating implication, 13
separation algebra, 12, 44
separation logic, 3, 11
  guarded, 110
  of bounded treewidth, 115
  quantifier-free, 110
separation logic competition, 18, 218
separation logic with inductive definitions, 21, 109
septraction, 13, 42, 110
sequence rule, 9
shape analysis, 12, 211
singly-linked list, 35
sink, 32
sink sequence, 33
size
  of SID, 112
  of an AMS, 62
  of SSL formula, 43
size formula, 52
SL-COMP, 18, 218
SL-projection
  of stack–forests pairs, 142
SLID, *see* separation logic with inductive definitions
`Sloth`, 99
$\mathbf{SL}^{*}_{\mathtt{data}}$, 83, 99
small-model property, 83
  of $\mathbf{SSL}^{+}$, 86
  of $\mathbf{SSL}^{+}_{\mathtt{data}}$, 87
`Smallfoot`, 26
SMT, 39, 214
SMT solver, 4
`Songbird`, 213
source, 32
spatial, 13
spatial formula, 42
`Spen`, 213
SSL, *see* strong-separation logic
stably infinite, 91
stack, 14
stack instantiation, 169
stack inverse, 20
stack–forest projection, 142
stack–heap pair, 14
stack-aliasing constraint, 20
stack-choice function, 20
`Strand`, 214
strong separation, 22
strong-separation logic, 21, 22, 39
  semantics, 43
  syntax, 41
successor, 32
symbolic execution, 12, 218
symbolic heap, 42, 110
symbolic-heap fragment, 17
symmetry closure, 36
syntax
  of $\mathbf{SL_{base}}$, 13
  of **SLID**, 110
  of **SSL**, 41
  of $\mathbf{SSL}_{\mathtt{data}}$, 55
system of inductive definitions, 112

terminal, 33, 201
theory of arrays, 91

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

TLL, *see* tree with linked leaves
tools, 215
transitive closure, 211
tree
  Φ-trees, 129
  directed tree, 33
tree projection, 138
tree with linked leaves, 26, 202
treewidth, 115
true
  in SLID, 111
  in **SSL**, 42
type, 163
type instantiation, 169
type composition, 166, 168
type computation
  completeness, 192
  fixed-point computation, 184
  of $\mathbf{SLID}^{g}_{btw}$ formula, 195
  soundness, 186

unary data predicate, 53
undecidability
  of $\mathbf{SLID}_{btw}(\wedge, \star, t)$, 205
  of $\mathbf{SLID}_{btw}(\wedge, \star, \neg)$, 206
  of $\mathbf{SLID}_{btw}(\wedge, \star, -\!\star)$, 206
  of $\mathbf{SLID}_{btw}(-\!\circledast)$, 207
unfolded symbolic heap, 157
  delimited, 162
unfolding
  of predicates, 123
unfolding tree, 123
union
  of forests, 133
unique footprint property
  of SSL, 89
universal data predicate, 53
USH, 157

validity
  of Hoare triple, 9
variable instantiation, 112
`Verifast`, 213
verification condition, 4, 24
`Viper`, 215

weak-separation logic, 50
weaken, 51
weakest preconditions, 24, 219
weakest-precondition calculus, 14
witness
  of data predicate, 86

x-types, 164

Z3, 91

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.

[Ant+14] Timos Antonopoulos, Nikos Gorogiannis, Christoph Haase, Max I. Kanovich, and Joël Ouaknine. "Foundations for Decision Problems in Separation Logic with General Inductive Predicates." In: *Foundations of Software Science and Computation Structures - 17th International Conference, FOSSACS 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014, Proceedings*. Ed. by Anca Muscholl. Vol. 8412. Lecture Notes in Computer Science. Springer, 2014, pp. 411–425 (cit. on pp. 22, 26, 112, 113, 115, 212).

[App14] Andrew W. Appel. *Program Logics - for Certified Compilers*. Cambridge University Press, 2014 (cit. on pp. 3, 14, 24, 219).

[BBL09] Kshitij Bansal, Rémi Brochenin, and Étienne Lozes. "Beyond Shapes: Lists with Ordered Data." In: *Foundations of Software Science and Computational Structures, 12th International Conference, FOSSACS 2009, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, York, UK, March 22-29, 2009. Proceedings*. Ed. by Luca de Alfaro. Vol. 5504. Lecture Notes in Computer Science. Springer, 2009, pp. 425–439 (cit. on p. 214).

[BCS15] Vince Bárány, Balder ten Cate, and Luc Segoufin. "Guarded Negation." In: *J. ACM* 62.3 (2015), 22:1–22:26 (cit. on p. 42).

[Bar+09] Clark W. Barrett, Roberto Sebastiani, Sanjit A. Seshia, and Cesare Tinelli. "Satisfiability Modulo Theories." In: *Handbook of Satisfiability*. Ed. by Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh. Vol. 185. Frontiers in Artificial Intelligence and Applications. IOS Press, 2009, pp. 825–885 (cit. on p. 39).

[Ber+07] Josh Berdine, Cristiano Calcagno, Byron Cook, Dino Distefano, Peter W. O'Hearn, Thomas Wies, and Hongseok Yang. "Shape Analysis for Composite Data Structures." In: *Computer Aided Verification, 19th International Conference, CAV 2007, Berlin, Germany, July 3-7, 2007, Proceedings*. Ed. by Werner Damm and Holger Hermanns. Vol. 4590. Lecture Notes in Computer Science. Springer, 2007, pp. 178–192 (cit. on p. 211).

[BCO04] Josh Berdine, Cristiano Calcagno, and Peter W. O'Hearn. "A Decidable Fragment of Separation Logic." In: *FSTTCS 2004: Foundations of Software Technology and Theoretical Computer Science, 24th International Conference, Chennai, India, December 16-18, 2004, Proceedings*. Ed. by Kamal Lodaya and Meena Mahajan. Vol. 3328. Lecture Notes in Computer Science. Springer, 2004, pp. 97–109 (cit. on pp. 17, 21, 23, 26, 44, 111, 212).

[BCO05a] Josh Berdine, Cristiano Calcagno, and Peter W. O'Hearn. "Smallfoot: Modular Automatic Assertion Checking with Separation Logic." In: *Formal Methods for Components and Objects, 4th International Symposium, FMCO 2005, Amsterdam, The Netherlands, November 1-4, 2005, Revised Lectures*. Ed. by Frank S. de Boer,

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Marcello M. Bonsangue, Susanne Graf, and Willem P. de Roever. Vol. 4111. Lecture Notes in Computer Science. Springer, 2005, pp. 115–137 (cit. on pp. 26, 212).

[BCO05b] Josh Berdine, Cristiano Calcagno, and Peter W. O'Hearn. "Symbolic Execution with Separation Logic." In: *Programming Languages and Systems, Third Asian Symposium, APLAS 2005, Tsukuba, Japan, November 2-5, 2005, Proceedings*. Ed. by Kwangkeun Yi. Vol. 3780. Lecture Notes in Computer Science. Springer, 2005, pp. 52–68 (cit. on pp. 12, 17, 21, 23, 24, 26, 212, 219).

[BB14] Armin Biere and Roderick Bloem, eds. *Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings*. Vol. 8559. Lecture Notes in Computer Science. Springer, 2014.

[BH15] Stefan Blom and Marieke Huisman. "Witnessing the elimination of magic wands." In: *Int. J. Softw. Tools Technol. Transf.* 17.6 (2015), pp. 757–781 (cit. on p. 14).

[Bob+11] Mihaela Gheorghiu Bobaru, Klaus Havelund, Gerard J. Holzmann, and Rajeev Joshi, eds. *NASA Formal Methods - Third International Symposium, NFM 2011, Pasadena, CA, USA, April 18-20, 2011. Proceedings*. Vol. 6617. Lecture Notes in Computer Science. Springer, 2011.

[BM07] Aaron R. Bradley and Zohar Manna. *The calculus of computation - decision procedures with applications to verification*. Springer, 2007 (cit. on p. 91).

[Bro13] Rémi Brochenin. "Separation logic : expressiveness, complexity, temporal extension. (Logique de séparation : expressivité, complexité, extension temporelle)." PhD thesis. École normale supérieure de Cachan, France, 2013 (cit. on p. 214).

[BDL12] Rémi Brochenin, Stéphane Demri, and Étienne Lozes. "On the almighty wand." In: *Inf. Comput.* 211 (2012), pp. 106–137 (cit. on pp. 13, 14, 17, 213).

[BDP11] James Brotherston, Dino Distefano, and Rasmus Lerchedahl Petersen. "Automated Cyclic Entailment Proofs in Separation Logic." In: *Automated Deduction - CADE-23 - 23rd International Conference on Automated Deduction, Wroclaw, Poland, July 31 - August 5, 2011. Proceedings*. Ed. by Nikolaj Bjørner and Viorica Sofronie-Stokkermans. Vol. 6803. Lecture Notes in Computer Science. Springer, 2011, pp. 131–146 (cit. on pp. 17, 22).

[Bro+14] James Brotherston, Carsten Fuhs, Juan Antonio Navarro Pérez, and Nikos Gorogiannis. "A decision procedure for satisfiability in separation logic with inductive predicates." In: *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14 - 18, 2014*. Ed. by Thomas A. Henzinger and Dale Miller. ACM, 2014, 25:1–25:10 (cit. on pp. 114, 212).

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[BG15] James Brotherston and Nikos Gorogiannis. "Disproving Inductive Entailments in Separation Logic via Base Pair Approximation." In: *Automated Reasoning with Analytic Tableaux and Related Methods - 24th International Conference, TABLEAUX 2015, Wrocław, Poland, September 21-24, 2015. Proceedings*. Ed. by Hans de Nivelle. Vol. 9323. Lecture Notes in Computer Science. Springer, 2015, pp. 287–303 (cit. on p. 213).

[Bro+16] James Brotherston, Nikos Gorogiannis, Max I. Kanovich, and Reuben Rowe. "Model checking for symbolic-heap separation logic with inductive predicates." In: *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*. Ed. by Rastislav Bodík and Rupak Majumdar. ACM, 2016, pp. 84–96 (cit. on pp. 212, 219).

[BGP12] James Brotherston, Nikos Gorogiannis, and Rasmus Lerchedahl Petersen. "A Generic Cyclic Theorem Prover." In: *Programming Languages and Systems - 10th Asian Symposium, APLAS 2012, Kyoto, Japan, December 11-13, 2012. Proceedings*. Ed. by Ranjit Jhala and Atsushi Igarashi. Vol. 7705. Lecture Notes in Computer Science. Springer, 2012, pp. 350–367 (cit. on p. 213).

[BK18] James Brotherston and Max I. Kanovich. "On the Complexity of Pointer Arithmetic in Separation Logic." In: *Programming Languages and Systems - 16th Asian Symposium, APLAS 2018, Wellington, New Zealand, December 2-6, 2018, Proceedings*. Ed. by Sukyoung Ryu. Vol. 11275. Lecture Notes in Computer Science. Springer, 2018, pp. 329–349 (cit. on p. 215).

[CD11] Cristiano Calcagno and Dino Distefano. "Infer: An Automatic Program Verifier for Memory Safety of C Programs." In: *NASA Formal Methods - Third International Symposium, NFM 2011, Pasadena, CA, USA, April 18-20, 2011. Proceedings*. Ed. by Mihaela Gheorghiu Bobaru, Klaus Havelund, Gerard J. Holzmann, and Rajeev Joshi. Vol. 6617. Lecture Notes in Computer Science. Springer, 2011, pp. 459–465 (cit. on p. 215).

[Cal+15] Cristiano Calcagno, Dino Distefano, Jérémy Dubreil, Dominik Gabi, Pieter Hooimeijer, Martino Luca, Peter W. O'Hearn, Irene Papakonstantinou, Jim Purbrick, and Dulma Rodriguez. "Moving Fast with Software Verification." In: *NASA Formal Methods - 7th International Symposium, NFM 2015, Pasadena, CA, USA, April 27-29, 2015, Proceedings*. Ed. by Klaus Havelund, Gerard J. Holzmann, and Rajeev Joshi. Vol. 9058. Lecture Notes in Computer Science. Springer, 2015, pp. 3–11 (cit. on pp. 3, 24, 215).

[Cal+06] Cristiano Calcagno, Dino Distefano, Peter W. O'Hearn, and Hongseok Yang. "Beyond Reachability: Shape Abstraction in the Presence of Pointer Arithmetic." In: *Static Analysis, 13th International Symposium, SAS 2006, Seoul, Korea, August 29-31, 2006, Proceedings*. Ed. by Kwangkeun Yi. Vol. 4134. Lecture Notes in Computer Science. Springer, 2006, pp. 182–203 (cit. on pp. 15, 215).

[Cal+07] Cristiano Calcagno, Dino Distefano, Peter W. O'Hearn, and Hongseok Yang. "Footprint Analysis: A Shape Analysis That Discovers Preconditions." In: *Static Analysis, 14th International Symposium, SAS 2007, Kongens Lyngby, Denmark, August 22-24, 2007, Proceedings*. Ed. by Hanne Riis Nielson and Gilberto Filé.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Vol. 4634. Lecture Notes in Computer Science. Springer, 2007, pp. 402–418 (cit. on p. 211).

[Cal+11] Cristiano Calcagno, Dino Distefano, Peter W. O'Hearn, and Hongseok Yang. "Compositional Shape Analysis by Means of Bi-Abduction." In: *J. ACM* 58.6 (2011), 26:1–26:66 (cit. on pp. 3, 12, 18, 24, 111, 211, 219).

[CGH05] Cristiano Calcagno, Philippa Gardner, and Matthew Hague. "From Separation Logic to First-Order Logic." In: *Foundations of Software Science and Computational Structures, 8th International Conference, FOSSACS 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2005, Edinburgh, UK, April 4-8, 2005, Proceedings*. Ed. by Vladimiro Sassone. Vol. 3441. Lecture Notes in Computer Science. Springer, 2005, pp. 395–409 (cit. on p. 213).

[COY07] Cristiano Calcagno, Peter W. O'Hearn, and Hongseok Yang. "Local Action and Abstract Separation Logic." In: *22nd IEEE Symposium on Logic in Computer Science (LICS 2007), 10-12 July 2007, Wroclaw, Poland, Proceedings*. IEEE Computer Society, 2007, pp. 366–378 (cit. on pp. 12, 16, 23, 44, 89, 113).

[CYO01] Cristiano Calcagno, Hongseok Yang, and Peter W. O'Hearn. "Computability and Complexity Results for a Spatial Assertion Language for Data Structures." In: *The Second Asian Workshop on Programming Languages and Systems, APLAS'01, Korea Advanced Institute of Science and Technology, Daejeon, Korea, December 17-18, 2001, Proceedings*. 2001, pp. 289–300 (cit. on pp. 17, 24, 26, 79, 213).

[CCA17] Qinxiang Cao, Santiago Cuellar, and Andrew W. Appel. "Bringing Order to the Separation Logic Jungle." In: *Programming Languages and Systems - 15th Asian Symposium, APLAS 2017, Suzhou, China, November 27-29, 2017, Proceedings*. Ed. by Bor-Yuh Evan Chang. Vol. 10695. Lecture Notes in Computer Science. Springer, 2017, pp. 190–211 (cit. on pp. 12, 44).

[Chi+12] Wei-Ngan Chin, Cristina David, Huu Hai Nguyen, and Shengchao Qin. "Automated verification of shape, size and bag properties via user-defined predicates in separation logic." In: *Sci. Comput. Program.* 77.9 (2012), pp. 1006–1036 (cit. on pp. 18, 214, 217).

[Coo+11] Byron Cook, Christoph Haase, Joël Ouaknine, Matthew J. Parkinson, and James Worrell. "Tractable Reasoning in a Fragment of Separation Logic." In: *CONCUR 2011 - Concurrency Theory - 22nd International Conference, CONCUR 2011, Aachen, Germany, September 6-9, 2011. Proceedings*. Ed. by Joost-Pieter Katoen and Barbara König. Vol. 6901. Lecture Notes in Computer Science. Springer, 2011, pp. 235–249 (cit. on pp. 17, 23, 26, 212).

[Cou90] Bruno Courcelle. "The Monadic Second-Order Logic of Graphs. I. Recognizable Sets of Finite Graphs." In: *Inf. Comput.* 85.1 (1990), pp. 12–75 (cit. on pp. 26, 212).

[DD14] Stéphane Demri and Morgan Deters. "Expressive completeness of separation logic with two variables and no separating conjunction." In: *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14 - 18, 2014*. Ed. by Thomas A. Henzinger and Dale Miller. ACM, 2014, 37:1–37:10 (cit. on p. 53).

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[DD15a] Stéphane Demri and Morgan Deters. "Separation logics and modalities: a survey." In: *Journal of Applied Non-Classical Logics* 25.1 (2015), pp. 50–99 (cit. on p. 11).

[DD15b] Stéphane Demri and Morgan Deters. "Two-Variable Separation Logic and Its Inner Circle." In: *ACM Trans. Comput. Log.* 16.2 (2015), 15:1–15:36 (cit. on pp. 53, 213).

[DD16] Stéphane Demri and Morgan Deters. "Expressive Completeness of Separation Logic with Two Variables and No Separating Conjunction." In: *ACM Trans. Comput. Log.* 17.2 (2016), 12:1–12:44 (cit. on p. 213).

[Dem+17] Stéphane Demri, Didier Galmiche, Dominique Larchey-Wendling, and Daniel Méry. "Separation Logic with One Quantified Variable." In: *Theory Comput. Syst.* 61.2 (2017), pp. 371–461 (cit. on p. 213).

[DLM18] Stéphane Demri, Étienne Lozes, and Alessio Mansutti. "The Effects of Adding Reachability Predicates in Propositional Separation Logic." In: *Foundations of Software Science and Computation Structures - 21st International Conference, FOSSACS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings*. Ed. by Christel Baier and Ugo Dal Lago. Vol. 10803. Lecture Notes in Computer Science. Springer, 2018, pp. 476–493 (cit. on pp. 21, 23, 24, 37, 45, 53, 213).

[Die16] Reinhard Diestel. *Graph Theory, 5th Edition*. Vol. 173. Graduate texts in mathematics. Springer, 2016 (cit. on p. 115).

[DOY06] Dino Distefano, Peter W. O'Hearn, and Hongseok Yang. "A Local Shape Analysis Based on Separation Logic." In: *Tools and Algorithms for the Construction and Analysis of Systems, 12th International Conference, TACAS 2006 Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2006, Vienna, Austria, March 25 - April 2, 2006, Proceedings*. Ed. by Holger Hermanns and Jens Palsberg. Vol. 3920. Lecture Notes in Computer Science. Springer, 2006, pp. 287–302 (cit. on pp. 3, 12, 211).

[DP08] Mike Dodds and Detlef Plump. "From Hyperedge Replacement to Separation Logic and Back." In: *ECEASST* 16 (2008) (cit. on p. 212).

[DES13] Cezara Dragoi, Constantin Enea, and Mihaela Sighireanu. "Local Shape Analysis for Overlaid Data Structures." In: *Static Analysis - 20th International Symposium, SAS 2013, Seattle, WA, USA, June 20-22, 2013. Proceedings*. Ed. by Francesco Logozzo and Manuel Fähndrich. Vol. 7935. Lecture Notes in Computer Science. Springer, 2013, pp. 150–171 (cit. on pp. 211, 213).

[EIP19a] Mnacho Echenim, Radu Iosif, and Nicolas Peltier. "Prenex Separation Logic with One Selector Field." In: *Automated Reasoning with Analytic Tableaux and Related Methods - 28th International Conference, TABLEAUX 2019, London, UK, September 3-5, 2019, Proceedings*. Ed. by Serenella Cerrito and Andrei Popescu. Vol. 11714. Lecture Notes in Computer Science. Springer, 2019, pp. 409–427 (cit. on p. 213).

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[EIP19b] Mnacho Echenim, Radu Iosif, and Nicolas Peltier. "The Bernays-Schönfinkel-Ramsey Class of Separation Logic on Arbitrary Domains." In: *Foundations of Software Science and Computation Structures - 22nd International Conference, FOSSACS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings*. Ed. by Mikolaj Boja'nczyk and Alex Simpson. Vol. 11425. Lecture Notes in Computer Science. Springer, 2019, pp. 242–259 (cit. on pp. 53, 212, 213).

[EIP20] Mnacho Echenim, Radu Iosif, and Nicolas Peltier. "Entailment Checking in Separation Logic with Inductive Definitions is 2-EXPTIME hard." In: *LPAR 2020: 23rd International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Alicante, Spain, May 22-27, 2020*. Ed. by Elvira Albert and Laura Kovács. Vol. 73. EPiC Series in Computing. EasyChair, 2020, pp. 191–211 (cit. on pp. 4, 22, 27, 212, 217).

[Ene+14] Constantin Enea, Ondrej Lengál, Mihaela Sighireanu, and Tomás Vojnar. "Compositional Entailment Checking for a Fragment of Separation Logic." In: *Programming Languages and Systems - 12th Asian Symposium, APLAS 2014, Singapore, November 17-19, 2014, Proceedings*. Ed. by Jacques Garrigue. Vol. 8858. Lecture Notes in Computer Science. Springer, 2014, pp. 314–333 (cit. on pp. 17, 22, 213).

[Ene+17] Constantin Enea, Ondrej Lengál, Mihaela Sighireanu, and Tomás Vojnar. "SPEN: A Solver for Separation Logic." In: *NASA Formal Methods - 9th International Symposium, NFM 2017, Moffett Field, CA, USA, May 16-18, 2017, Proceedings*. Ed. by Clark W. Barrett, Misty Davies, and Temesghen Kahsai. Vol. 10227. Lecture Notes in Computer Science. 2017, pp. 302–309 (cit. on pp. 17, 213).

[EM19] Gidon Ernst and Toby Murray. "SecCSL: Security Concurrent Separation Logic." In: *Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part II*. Ed. by Isil Dillig and Serdar Tasiran. Vol. 11562. Lecture Notes in Computer Science. Springer, 2019, pp. 208–230 (cit. on p. 3).

[FP15] Xinyu Feng and Sungwoo Park, eds. *Programming Languages and Systems - 13th Asian Symposium, APLAS 2015, Pohang, South Korea, November 30 - December 2, 2015, Proceedings*. Vol. 9458. Lecture Notes in Computer Science. Springer, 2015.

[GCW19] Chong Gao, Taolue Chen, and Zhilin Wu. "Separation Logic with Linearly Compositional Inductive Predicates and Set Data Constraints." In: *SOFSEM 2019: Theory and Practice of Computer Science - 45th International Conference on Current Trends in Theory and Practice of Computer Science, Nový Smokovec, Slovakia, January 27-30, 2019, Proceedings*. Ed. by Barbara Catania, Rastislav Královic, Jerzy R. Nawrocki, and Giovanni Pighizzini. Vol. 11376. Lecture Notes in Computer Science. Springer, 2019, pp. 206–220 (cit. on p. 215).

[GKO11] Nikos Gorogiannis, Max I. Kanovich, and Peter W. O'Hearn. "The Complexity of Abduction for Separated Heap Abstractions." In: *Static Analysis - 18th International Symposium, SAS 2011, Venice, Italy, September 14-16, 2011. Proceedings*.

TU Bibliothek. Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar. The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.

Ed. by Eran Yahav. Vol. 6887. Lecture Notes in Computer Science. Springer, 2011, pp. 25–42 (cit. on p. 219).

[Got+07] Alexey Gotsman, Josh Berdine, Byron Cook, Noam Rinetzky, and Mooly Sagiv. "Local Reasoning for Storable Locks and Threads." In: *Programming Languages and Systems, 5th Asian Symposium, APLAS 2007, Singapore, November 29-December 1, 2007, Proceedings*. Ed. by Zhong Shao. Vol. 4807. Lecture Notes in Computer Science. Springer, 2007, pp. 19–37 (cit. on p. 11).

[GCW16] Xincai Gu, Taolue Chen, and Zhilin Wu. "A Complete Decision Procedure for Linearly Compositional Separation Logic with Data Constraints." In: *Automated Reasoning - 8th International Joint Conference, IJCAR 2016, Coimbra, Portugal, June 27 - July 2, 2016, Proceedings*. Ed. by Nicola Olivetti and Ashish Tiwari. Vol. 9706. Lecture Notes in Computer Science. Springer, 2016, pp. 532–549 (cit. on pp. 26, 214).

[HM14] Thomas A. Henzinger and Dale Miller, eds. *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14 - 18, 2014*. ACM, 2014.

[Hol+16] Lukás Holík, Michal Kotoun, Petr Peringer, Veronika Soková, Marek Trtík, and Tomás Vojnar. "Predator Shape Analysis Tool Suite." In: *Hardware and Software: Verification and Testing - 12th International Haifa Verification Conference, HVC 2016, Haifa, Israel, November 14-17, 2016, Proceedings*. Ed. by Roderick Bloem and Eli Arbel. Vol. 10028. Lecture Notes in Computer Science. 2016, pp. 202–209 (cit. on p. 215).

[IRS13] Radu Iosif, Adam Rogalewicz, and Jirí Simácek. "The Tree Width of Separation Logic with Recursive Definitions." In: *Automated Deduction - CADE-24 - 24th International Conference on Automated Deduction, Lake Placid, NY, USA, June 9-14, 2013. Proceedings*. Ed. by Maria Paola Bonacina. Vol. 7898. Lecture Notes in Computer Science. Springer, 2013, pp. 21–38 (cit. on pp. v, viii, 4, 5, 17, 18, 22, 23, 26, 27, 41, 107, 109, 112, 113, 115, 123, 129, 212, 217).

[IRV14] Radu Iosif, Adam Rogalewicz, and Tomás Vojnar. "Deciding Entailments in Inductive Separation Logic with Tree Automata." In: *Automated Technology for Verification and Analysis - 12th International Symposium, ATVA 2014, Sydney, NSW, Australia, November 3-7, 2014, Proceedings*. Ed. by Franck Cassez and Jean-François Raskin. Vol. 8837. Lecture Notes in Computer Science. Springer, 2014, pp. 201–218 (cit. on pp. 17, 22, 23, 26, 115, 123, 129, 212).

[IO01] Samin S. Ishtiaq and Peter W. O'Hearn. "BI as an Assertion Language for Mutable Data Structures." In: *Conference Record of POPL 2001: The 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, London, UK, January 17-19, 2001*. Ed. by Chris Hankin and Dave Schmidt. ACM, 2001, pp. 14–26 (cit. on pp. 13, 14, 24, 25, 215, 219).

[Itz+13] Shachar Itzhaky, Anindya Banerjee, Neil Immerman, Aleksandar Nanevski, and Mooly Sagiv. "Effectively-Propositional Reasoning about Reachability in Linked Data Structures." In: *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*. Ed.

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

by Natasha Sharygina and Helmut Veith. Vol. 8044. Lecture Notes in Computer Science. Springer, 2013, pp. 756–772 (cit. on p. 211).

[Jac+11] Bart Jacobs, Jan Smans, Pieter Philippaerts, Frédéric Vogels, Willem Penninckx, and Frank Piessens. "VeriFast: A Powerful, Sound, Predictable, Fast Verifier for C and Java." In: *NASA Formal Methods - Third International Symposium, NFM 2011, Pasadena, CA, USA, April 18-20, 2011. Proceedings*. Ed. by Mihaela Gheorghiu Bobaru, Klaus Havelund, Gerard J. Holzmann, and Rajeev Joshi. Vol. 6617. Lecture Notes in Computer Science. Springer, 2011, pp. 41–55 (cit. on pp. 3, 12, 213, 215).

[Jan17] Christina Jansen. "Static Analysis of Pointer Programs - Linking Graph Grammars and Separation Logic." PhD thesis. RWTH Aachen University, Germany, 2017 (cit. on p. 212).

[Jan+17] Christina Jansen, Jens Katelaan, Christoph Matheja, Thomas Noll, and Florian Zuleger. "Unified Reasoning About Robustness Properties of Symbolic-Heap Separation Logic." In: *Programming Languages and Systems - 26th European Symposium on Programming, ESOP 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings*. Ed. by Hongseok Yang. Vol. 10201. Lecture Notes in Computer Science. Springer, 2017, pp. 611–638 (cit. on pp. 5, 107, 109, 123, 129, 218).

[Jun+18] Ralf Jung, Robbert Krebbers, Jacques-Henri Jourdan, Ales Bizjak, Lars Birkedal, and Derek Dreyer. "Iris from the ground up: A modular foundation for higher-order concurrent separation logic." In: *J. Funct. Program.* 28 (2018), e20 (cit. on p. 12).

[KJW18a] Jens Katelaan, Dejan Jovanović, and Georg Weissenbacher. "A Separation Logic with Data: Small Models and Automation." In: *Automated Reasoning - 9th International Joint Conference, IJCAR 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings*. Ed. by Didier Galmiche, Stephan Schulz, and Roberto Sebastiani. Vol. 10900. Lecture Notes in Computer Science. Springer, 2018, pp. 455–471 (cit. on pp. 5, 37, 39, 83, 211, 218).

[KJW18b] Jens Katelaan, Dejan Jovanović, and Georg Weissenbacher. *Sloth: Separation Logic and Theories via Small Models*. Informal proceedings of the First Workshop on Automated Deduction for Separation Logics (ADSL). 2018 (cit. on p. 99).

[Kat+18] Jens Katelaan, Christoph Matheja, Thomas Noll, and Florian Zuleger. "Harrsh: A Tool for Unified Reasoning about Symbolic-Heap Separation Logic." In: *Proceedings of the 13th International Workshop on the Implementation of Logics (IWIL)*. 2018 (cit. on pp. 5, 107, 109, 218).

[KMZ19a] Jens Katelaan, Christoph Matheja, and Florian Zuleger. "Effective Entailment Checking for Separation Logic with Inductive Definitions." In: *Tools and Algorithms for the Construction and Analysis of Systems - 25th International Conference, TACAS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings, Part II*. Ed. by Tomás Vojnar and Lijun Zhang. Vol. 11428. Lecture Notes in

TU WIEN Bibliothek Your knowledge hub
Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.

Computer Science. Springer, 2019, pp. 319–336 (cit. on pp. 5, 107, 109, 167, 199, 218).

[KMZ19b] Jens Katelaan, Christoph Matheja, and Florian Zuleger. *Supplementary material*. The webpage https://github.com/katelaan/harrsh provides access to the extended version of our paper Effective Entailment Checking for Separation Logic with Inductive Definitions. 2019 (cit. on pp. 167, 218).

[KVZ16] Tomer Kotek, Helmut Veith, and Florian Zuleger. "Monadic Second Order Finite Satisfiability and Unbounded Tree-Width." In: *25th EACSL Annual Conference on Computer Science Logic, CSL 2016, August 29 - September 1, 2016, Marseille, France*. Ed. by Jean-Marc Talbot and Laurent Regnier. Vol. 62. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016, 13:1–13:20 (cit. on p. 212).

[KTB17] Robbert Krebbers, Amin Timany, and Lars Birkedal. "Interactive proofs in higher-order concurrent separation logic." In: *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017*. Ed. by Giuseppe Castagna and Andrew D. Gordon. ACM, 2017, pp. 205–217 (cit. on p. 12).

[KSW20] S. Krishna, A.J. Summers, and T. Wies. "Local Reasoning for Global Graph Properties." In: 2020 (cit. on p. 214).

[KSW18] Siddharth Krishna, Dennis E. Shasha, and Thomas Wies. "Go with the flow: compositional abstractions for concurrent data structures." In: *PACMPL* 2.POPL (2018), 37:1–37:31 (cit. on pp. 3, 214).

[LQ08] Shuvendu K. Lahiri and Shaz Qadeer. "Back to the future: revisiting precise program verification using SMT solvers." In: *Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2008, San Francisco, California, USA, January 7-12, 2008*. Ed. by George C. Necula and Philip Wadler. ACM, 2008, pp. 171–182 (cit. on pp. 211, 214).

[Le+14] Quang Loc Le, Cristian Gherghina, Shengchao Qin, and Wei-Ngan Chin. "Shape Analysis via Second-Order Bi-Abduction." In: *Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings*. Ed. by Armin Biere and Roderick Bloem. Vol. 8559. Lecture Notes in Computer Science. Springer, 2014, pp. 52–68 (cit. on p. 211).

[LSQ18] Quang Loc Le, Jun Sun, and Shengchao Qin. "Frame Inference for Inductive Entailment Proofs in Separation Logic." In: *Tools and Algorithms for the Construction and Analysis of Systems - 24th International Conference, TACAS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings, Part I*. Ed. by Dirk Beyer and Marieke Huisman. Vol. 10805. Lecture Notes in Computer Science. Springer, 2018, pp. 41–60 (cit. on p. 219).

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[Le+17] Quang Loc Le, Makoto Tatsuta, Jun Sun, and Wei-Ngan Chin. "A Decidable Fragment in Separation Logic with Inductive Predicates and Arithmetic." In: *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part II*. Ed. by Rupak Majumdar and Viktor Kuncak. Vol. 10427. Lecture Notes in Computer Science. Springer, 2017, pp. 495–517 (cit. on pp. 214, 217).

[Lei10] K. Rustan M. Leino. "Dafny: An Automatic Program Verifier for Functional Correctness." In: *Logic for Programming, Artificial Intelligence, and Reasoning - 16th International Conference, LPAR-16, Dakar, Senegal, April 25-May 1, 2010, Revised Selected Papers*. Ed. by Edmund M. Clarke and Andrei Voronkov. Vol. 6355. Lecture Notes in Computer Science. Springer, 2010, pp. 348–370 (cit. on p. 4).

[Lev+09] Tal Lev-Ami, Neil Immerman, Thomas W. Reps, Mooly Sagiv, Siddharth Srivastava, and Greta Yorsh. "Simulating reachability using first-order logic with applications to verification of linked data structures." In: *Logical Methods in Computer Science* 5.2 (2009) (cit. on p. 211).

[MPQ11] P. Madhusudan, Gennaro Parlato, and Xiaokang Qiu. "Decidable logics combining heap structures and data." In: *Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011, Austin, TX, USA, January 26-28, 2011*. Ed. by Thomas Ball and Mooly Sagiv. ACM, 2011, pp. 611–622 (cit. on pp. 214, 217).

[Mat20] Christoph Matheja. "Automated reasoning and randomization in separation logic." PhD thesis. Aachen: RWTH Aachen University, 2020 (cit. on p. 18).

[MJN15] Christoph Matheja, Christina Jansen, and Thomas Noll. "Tree-Like Grammars and Separation Logic." In: *Programming Languages and Systems - 13th Asian Symposium, APLAS 2015, Pohang, South Korea, November 30 - December 2, 2015, Proceedings*. Ed. by Xinyu Feng and Sungwoo Park. Vol. 9458. Lecture Notes in Computer Science. Springer, 2015, pp. 90–108 (cit. on pp. 17, 23).

[McC62] John McCarthy. "Towards a Mathematical Science of Computation." In: *Information Processing, Proceedings of the 2nd IFIP Congress 1962, Munich, Germany, August 27 - September 1, 1962*. North-Holland, 1962, pp. 21–28 (cit. on p. 91).

[MH69] John McCarthy and Patrick J. Hayes. "Some Philosophical Problems from the Standpoint of Artificial Intelligence." In: *Machine Intelligence*. Edinburgh University Press, 1969, pp. 463–502 (cit. on p. 10).

[MS01] Anders Møller and Michael I. Schwartzbach. "The Pointer Assertion Logic Engine." In: *Proceedings of the 2001 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), Snowbird, Utah, USA, June 20-22, 2001*. Ed. by Michael Burke and Mary Lou Soffa. ACM, 2001, pp. 221–231 (cit. on pp. 41, 212).

[MB08] Leonardo Mendonça de Moura and Nikolaj Bjørner. "Z3: An Efficient SMT Solver." In: *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*. Ed. by C. R. Ramakrishnan and Jakob Rehof.

TU WIEN Bibliothek Your knowledge hub
Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.

Vol. 4963. Lecture Notes in Computer Science. Springer, 2008, pp. 337–340 (cit. on pp. 4, 91).

[MB09] Leonardo Mendonça de Moura and Nikolaj Bjørner. "Generalized, efficient array decision procedures." In: *Proceedings of 9th International Conference on Formal Methods in Computer-Aided Design, FMCAD 2009, 15-18 November 2009, Austin, Texas, USA*. IEEE, 2009, pp. 45–52 (cit. on p. 91).

[MSS16] Peter Müller, Malte Schwerhoff, and Alexander J. Summers. "Viper: A Verification Infrastructure for Permission-Based Reasoning." In: *Verification, Model Checking, and Abstract Interpretation - 17th International Conference, VMCAI 2016, St. Petersburg, FL, USA, January 17-19, 2016. Proceedings*. Ed. by Barbara Jobstmann and K. Rustan M. Leino. Vol. 9583. Lecture Notes in Computer Science. Springer, 2016, pp. 41–62 (cit. on p. 215).

[OHe07] Peter W. O'Hearn. "Resources, concurrency, and local reasoning." In: *Theor. Comput. Sci.* 375.1-3 (2007), pp. 271–307 (cit. on p. 3).

[OHe19] Peter W. O'Hearn. "Separation logic." In: *Commun. ACM* 62.2 (2019), pp. 86–95 (cit. on pp. 3, 12, 18, 50).

[PMZ20] Jens Pagel, Christoph Matheja, and Florian Zuleger. "Complete Entailment Checking for Separation Logic with Inductive Definitions." In: *To be published. Preprint available at CoRR* abs/2002.01202 (2020). arXiv: 2002.01202 (cit. on pp. 5, 107, 109).

[PZ20a] Jens Pagel and Florian Zuleger. "Beyond Symbolic Heaps: Deciding Separation Logic With Inductive Definitions." In: *LPAR 2020: 23rd International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Alicante, Spain, May 22-27, 2020*. Ed. by Elvira Albert and Laura Kovács. Vol. 73. EPiC Series in Computing. EasyChair, 2020, pp. 390–408 (cit. on pp. 5, 107, 109).

[PZ20b] Jens Pagel and Florian Zuleger. "Strong-Separation Logic." In: *Submitted. Preprint available at CoRR* abs/2001.06235 (2020). arXiv: 2001.06235 (cit. on pp. 5, 23, 37, 39, 58).

[Par10] Matthew J. Parkinson. "The Next 700 Separation Logics - (Invited Paper)." In: *Verified Software: Theories, Tools, Experiments, Third International Conference, VSTTE 2010, Edinburgh, UK, August 16-19, 2010. Proceedings*. Ed. by Gary T. Leavens, Peter W. O'Hearn, and Sriram K. Rajamani. Vol. 6217. Lecture Notes in Computer Science. Springer, 2010, pp. 169–182 (cit. on pp. 3, 11).

[PR13] Juan Antonio Navarro Pérez and Andrey Rybalchenko. "Separation Logic Modulo Theories." In: *Programming Languages and Systems - 11th Asian Symposium, APLAS 2013, Melbourne, VIC, Australia, December 9-11, 2013. Proceedings*. Ed. by Chung-chieh Shan. Vol. 8301. Lecture Notes in Computer Science. Springer, 2013, pp. 90–106 (cit. on pp. 17, 26, 211, 214).

[Pha+19a] Long H. Pham, Quang Loc Le, Quoc-Sang Phan, and Jun Sun. "Concolic Testing Heap-Manipulating Programs." In: *Formal Methods - The Next 30 Years - Third World Congress, FM 2019, Porto, Portugal, October 7-11, 2019, Proceedings*. Ed. by Maurice H. ter Beek, Annabelle McIver, and José N. Oliveira. Vol. 11800. Lecture Notes in Computer Science. Springer, 2019, pp. 442–461 (cit. on pp. 3, 12).

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[Pha+19b] Long H. Pham, Quang Loc Le, Quoc-Sang Phan, Jun Sun, and Shengchao Qin. "Enhancing Symbolic Execution of Heap-Based Programs with Separation Logic for Test Input Generation." In: *Automated Technology for Verification and Analysis - 17th International Symposium, ATVA 2019, Taipei, Taiwan, October 28-31, 2019, Proceedings*. Ed. by Yu-Fang Chen, Chih-Hong Cheng, and Javier Esparza. Vol. 11781. Lecture Notes in Computer Science. Springer, 2019, pp. 209–227 (cit. on p. 12).

[PWZ13] Ruzica Piskac, Thomas Wies, and Damien Zufferey. "Automating Separation Logic Using SMT." In: *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*. Ed. by Natasha Sharygina and Helmut Veith. Vol. 8044. Lecture Notes in Computer Science. Springer, 2013, pp. 773–789 (cit. on pp. 15, 17, 26, 93, 211, 214).

[PWZ14a] Ruzica Piskac, Thomas Wies, and Damien Zufferey. "Automating Separation Logic with Trees and Data." In: *Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings*. Ed. by Armin Biere and Roderick Bloem. Vol. 8559. Lecture Notes in Computer Science. Springer, 2014, pp. 711–728 (cit. on pp. 17, 26, 211, 214).

[PWZ14b] Ruzica Piskac, Thomas Wies, and Damien Zufferey. "GRASShopper - Complete Heap Verification with Mixed Specifications." In: *Tools and Algorithms for the Construction and Analysis of Systems - 20th International Conference, TACAS 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014. Proceedings*. Ed. by Erika Ábrahám and Klaus Havelund. Vol. 8413. Lecture Notes in Computer Science. Springer, 2014, pp. 124–139 (cit. on pp. 3, 12, 18, 215).

[PNB13] Mathias Preiner, Aina Niemetz, and Armin Biere. "Lemmas on Demand for Lambdas." In: *Proceedings of the Second International Workshop on Design and Implementation of Formal Tools and Systems, Portland, OR, USA, October 19, 2013*. Ed. by Malay K. Ganai and Alper Sen. Vol. 1130. CEUR Workshop Proceedings. CEUR-WS.org, 2013 (cit. on p. 91).

[Qiu+13] Xiaokang Qiu, Pranav Garg, Andrei Stefanescu, and P. Madhusudan. "Natural proofs for structure, data, and separation." In: *ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '13, Seattle, WA, USA, June 16-19, 2013*. Ed. by Hans-Juergen Boehm and Cormac Flanagan. ACM, 2013, pp. 231–242 (cit. on pp. 18, 214, 217).

[Rab69] Michael O Rabin. "Decidability of second-order theories and automata on infinite trees." In: *Transactions of the American Mathematical Society* (1969), pp. 1–35 (cit. on p. 26).

[RIS17] Andrew Reynolds, Radu Iosif, and Cristina Serban. "Reasoning in the Bernays-Schönfinkel-Ramsey Fragment of Separation Logic." In: *Verification, Model Checking, and Abstract Interpretation - 18th International Conference, VMCAI 2017, Paris, France, January 15-17, 2017, Proceedings*. Ed. by Ahmed Bouajjani and David Monniaux. Vol. 10145. Lecture Notes in Computer Science. Springer, 2017, pp. 462–482 (cit. on p. 213).

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[Rey+16] Andrew Reynolds, Radu Iosif, Cristina Serban, and Tim King. "A Decision Procedure for Separation Logic in SMT." In: *Automated Technology for Verification and Analysis - 14th International Symposium, ATVA 2016, Chiba, Japan, October 17-20, 2016, Proceedings*. Ed. by Cyrille Artho, Axel Legay, and Doron Peled. Vol. 9938. Lecture Notes in Computer Science. 2016, pp. 244–261 (cit. on p. 213).

[Rey02] John C. Reynolds. "Separation Logic: A Logic for Shared Mutable Data Structures." In: *17th IEEE Symposium on Logic in Computer Science (LICS 2002), 22-25 July 2002, Copenhagen, Denmark, Proceedings*. IEEE Computer Society, 2002, pp. 55–74 (cit. on pp. 3, 11, 14, 15, 18, 24, 25, 50, 111, 113, 136, 215, 219).

[Rey11] John C. Reynolds. "Making Program Logics Intelligible." In: *5th IEEE International Symposium on Theoretical Aspects of Software Engineering, TASE 2011, Xi'an, China, 29-31 August 2011*. Ed. by Zhenhua Duan and C.-H. Luke Ong. IEEE Computer Society, 2011, pp. 3–4 (cit. on p. 3).

[SRW02] Shmuel Sagiv, Thomas W. Reps, and Reinhard Wilhelm. "Parametric shape analysis via 3-valued logic." In: *ACM Trans. Program. Lang. Syst.* 24.3 (2002), pp. 217–298 (cit. on p. 211).

[SS15] Malte Schwerhoff and Alexander J. Summers. "Lightweight Support for Magic Wands in an Automatic Verifier." In: *29th European Conference on Object-Oriented Programming, ECOOP 2015, July 5-10, 2015, Prague, Czech Republic*. Ed. by John Tang Boyland. Vol. 37. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015, pp. 614–638 (cit. on p. 14).

[SI18] Cristina Serban and Radu Iosif. "An Entailment Checker for Separation Logic with Inductive Definitions." In: *ECEASST* 76 (2018) (cit. on pp. 17, 22).

[SV13] Natasha Sharygina and Helmut Veith, eds. *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*. Vol. 8044. Lecture Notes in Computer Science. Springer, 2013.

[Sig+19] Mihaela Sighireanu et al. "SL-COMP: Competition of Solvers for Separation Logic." In: *Tools and Algorithms for the Construction and Analysis of Systems - 25 Years of TACAS: TOOLympics, Held as Part of ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings, Part III*. Ed. by Dirk Beyer, Marieke Huisman, Fabrice Kordon, and Bernhard Steffen. Vol. 11429. Lecture Notes in Computer Science. Springer, 2019, pp. 116–132 (cit. on pp. 17, 18, 22, 26, 27, 218).

[SM73] Larry J. Stockmeyer and Albert R. Meyer. "Word Problems Requiring Exponential Time: Preliminary Report." In: *Proceedings of the 5th Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1973, Austin, Texas, USA*. Ed. by Alfred V. Aho, Allan Borodin, Robert L. Constable, Robert W. Floyd, Michael A. Harrison, Richard M. Karp, and H. Raymond Strong. ACM, 1973, pp. 1–9 (cit. on p. 212).

[Ta+18] Quang-Trung Ta, Ton Chanh Le, Siau-Cheng Khoo, and Wei-Ngan Chin. "Automated lemma synthesis in symbolic-heap separation logic." In: *PACMPL* 2.POPL (2018), 9:1–9:29 (cit. on pp. 17, 22, 213, 214).

TU WIEN Bibliothek Your knowledge hub
Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.

[TK15] Makoto Tatsuta and Daisuke Kimura. "Separation Logic with Monadic Inductive Definitions and Implicit Existentials." In: *Programming Languages and Systems - 13th Asian Symposium, APLAS 2015, Pohang, South Korea, November 30 - December 2, 2015, Proceedings*. Ed. by Xinyu Feng and Sungwoo Park. Vol. 9458. Lecture Notes in Computer Science. Springer, 2015, pp. 69–89 (cit. on pp. 17, 22, 23, 26, 212).

[TLC16] Makoto Tatsuta, Quang Loc Le, and Wei-Ngan Chin. "Decision Procedure for Separation Logic with Inductive Definitions and Presburger Arithmetic." In: *Programming Languages and Systems - 14th Asian Symposium, APLAS 2016, Hanoi, Vietnam, November 21-23, 2016, Proceedings*. Ed. by Atsushi Igarashi. Vol. 10017. Lecture Notes in Computer Science. 2016, pp. 423–443 (cit. on p. 214).

[TNK19] Makoto Tatsuta, Koji Nakazawa, and Daisuke Kimura. "Completeness of Cyclic Proofs for Symbolic Heaps with Inductive Definitions." In: *Programming Languages and Systems - 17th Asian Symposium, APLAS 2019, Nusa Dua, Bali, Indonesia, December 1-4, 2019, Proceedings*. Ed. by Anthony Widjaja Lin. Vol. 11893. Lecture Notes in Computer Science. Springer, 2019, pp. 367–387 (cit. on pp. 17, 22, 23, 26, 157, 213).

[TBR14] Aditya V. Thakur, Jason Breck, and Thomas W. Reps. "Satisfiability modulo abstraction for separation logic with linked lists." In: *2014 International Symposium on Model Checking of Software, SPIN 2014, Proceedings, San Jose, CA, USA, July 21-23, 2014*. Ed. by Neha Rungta and Oksana Tkachuk. ACM, 2014, pp. 58–67 (cit. on p. 13).

[Yan01] Hongseok Yang. "Local Reasoning for Stateful Programs." AAI3023240. PhD thesis. Champaign, IL, USA: University of Illinois at Urbana-Champaign, 2001 (cit. on p. 25).

[Yan+08] Hongseok Yang, Oukseh Lee, Josh Berdine, Cristiano Calcagno, Byron Cook, Dino Distefano, and Peter W. O'Hearn. "Scalable Shape Analysis for Systems Code." In: *Computer Aided Verification, 20th International Conference, CAV 2008, Princeton, NJ, USA, July 7-14, 2008, Proceedings*. Ed. by Aarti Gupta and Sharad Malik. Vol. 5123. Lecture Notes in Computer Science. Springer, 2008, pp. 385–398 (cit. on pp. 3, 211).

Die approbierte gedruckte Originalversion dieser Dissertation ist an der TU Wien Bibliothek verfügbar.
The approved original version of this doctoral thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub