Informatics

ATLAS: Automated Amortised Complexity Analysis of Self-Adjusting Data Structures

MASTERARBEIT

zur Erlangung des akademischen Grades

Master of Science

im Rahmen des Studiums

European Master's Program in Computational Logic

eingereicht von

Lorenz Leutgeb, BSc
Matrikelnummer 01127842

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Dipl.-Math. Dr.techn. Florian Zuleger

Wien, 1. Jänner 2021

Lorenz Leutgeb   Florian Zuleger

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

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

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

Informatics

ATLAS: Automated Amortised Complexity Analysis of Self-Adjusting Data Structures

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

European Master's Program in Computational Logic

by

Lorenz Leutgeb, BSc
Registration Number 01127842

to the Faculty of Informatics

at the TU Wien

Advisor: Dipl.-Math. Dr.techn. Florian Zuleger

Vienna, 1st January, 2021

Lorenz Leutgeb | Florian Zuleger

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

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

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

Erklärung zur Verfassung der Arbeit

Lorenz Leutgeb, BSc

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.

Wien, 1. Jänner 2021

Lorenz Leutgeb

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

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

Acknowledgements

Florian Zuleger and Georg Moser invited me to join their collaboration. We spent many hours discussing, debugging, researching. Without them, this thesis would not have been possible.
Thank you!

My family always supports me in every way possible.
Thank you!

This thesis concludes my studies in the European Master's Program in Computational Logic, and I look back on adventures in Dresden, Bolzano, and Canberra, full of gratefulness. What makes these places and memories special for me, is the excellent bunch of people that I share the experiences with.
Thank you!

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

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

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

Kurzfassung

Seit der Vorstellung der *amortisierten Analyse* durch Sleator und Tarjan wird an der systematischen Analyse des Verhaltens von selbst-organisierenden Datenstrukturen wie z.B. Splay-Bäumen gearbeitet. Solch Analyse erfordert komplizierte Potentialfunktionen, welche typischerweise logarithmische Ausdrücke enthalten.

Vielleicht deshalb, und trotz der jüngsten Fortschritte in der automatisierten Ressourcenanalyse, konnte dies bisher noch nicht automatisiert werden. In dieser Arbeit berichten wir von der ersten, vollständig automatisierten Analyse der amortisierten Komplexität von selbst-organisierenden Datenstrukturen.

Wir liefern folgende Beiträge:

1. Wir stellen eine neuartige amortisierte Ressourcenanalyse in Form eines Typsystems vor. Unsere Analyse ist im Stil der Physiker-Methode formuliert, und basiert auf dem Konzept des Potentials. Das Typsystem verwendet logarithmische Potentialfunktionen und ist das erste solche, das logarithmische amortisierte Komplexität behandelt.

2. Wir codieren die Suche nach konkreten Koeffizienten fuer Potentialfunktionen als Optimierungsproblem über einem passenden Gleichungssystem. Unsere Zielfunktion steuert die Suche in Richtung kleinerer Koeffizienten, sodass die inferierte amortisierte Komplexität minimiert wird.

3. Automatisierung wird durch ein lineares Gleichungssystem in Verbindung mit entsprechenden Hilfssätzen erreicht, die nichtlineare Eigenschaften des Logairthmus ausdrücken. Wir diskutieren unsere Entscheidungen, die eine Skalierung des Ansatzes ermöglichen.

4. Wir präsentieren unser Computerprogramm ATLAS zur automatischen Analyse und berichten über experimentelle Ergebnisse zu Splay-Bäumen, Splay-Halden und Pairing-Halden. Wir inferieren vollautomatisch Komplexitätsabschätzungen, welche denen der Literatur entsprechen, obwohl diese nur durch komplexe Beweise auf Papier erzielt werden konnte, und verbessern diese in einigen Fällen sogar.

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

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

Abstract

Being able to argue about the performance of self-adjusting data structures (such as splay trees) has been a main objective when Sleator and Tarjan introduced the notion of *amortised* complexity.

Analysing these data structures requires sophisticated potential functions, which typically contain logarithmic expressions. Possibly for these reasons, and despite the recent progress in automated resource analysis, they have so far eluded automation. In this thesis, we report on the first fully automated amortised complexity analysis of self-adjusting data structures.

We make the following contributions:

1. We introduce a novel amortised resource analysis couched in a type-and-effect system. Our analysis is formulated in terms of the physicist's method of amortised analysis, and is potential-based. The type system makes use of logarithmic potential functions and is the first such system to exhibit *logarithmic amortised complexity*.

2. We encode the search for concrete potential function coefficients as an optimisation problem over a suitable constraint system. Our target function steers the search towards coefficients that minimise the inferred amortised complexity.

3. Automation is achieved by using a linear constraint system in conjunction with suitable lemmata schemes that encapsulate the required non-linear facts about the logarithm. We discuss our choices that achieve a scalable analysis.

4. We present our tool ATLAS and report on experimental results for *splay trees*, *splay heaps* and *pairing heaps*. We completely automatically infer complexity estimates that match previous results (obtained by sophisticated pen-and-paper proofs), and in some cases even infer better complexity estimates than previously published.

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

Contents

Kurzfassung ix

Abstract xi

Contents xiii

1 Introduction 1
- 1.1 State of the Art and Related Work . . . 3
- 1.2 Contributions . . . 4
- 1.3 Outline . . . 8
- 1.4 The Physicist's Method of Amortised Analysis . . . 8

2 Preliminaries 11
- 2.1 Setting the Stage . . . 11
- 2.2 A Necessarily Simple and Sufficiently Complex Programming Language . . . 16
- 2.3 Motivating Example: Splay Trees . . . 17

3 A Type System for Analysis of Logarithmic Amortized Complexity 21
- 3.1 Resource Functions . . . 21
- 3.2 A Type System for Logarithmic Amortised Resource Analysis . . . 25
- 3.3 Example Analysis . . . 33

4 Automation 41
- 4.1 Linearisation and Expert Knowledge . . . 41
- 4.2 Type Inference . . . 45

5 Implementation 49
- 5.1 The Three Phases of ATLAS . . . 50
- 5.2 Optimisation . . . 55
- 5.3 Evaluation . . . 56

6 Conclusion 59

List of Figures 61

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

List of Tables 61

Bibliography 63

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

CHAPTER I

Introduction

Amortised analysis, as pioneered by Sleator and Tarjan [DR85; RE 85], is a method for the worst-case cost analysis of data structures. The innovation of amortised analysis is to not only consider the cost of performing a single operation on a data structure, but the cost of performing a sequence of operations. The methodology of amortised analysis allows one to assign a low (e.g. constant or logarithmic) amortised cost to a data structure operation even though the worst-case cost of a single operation might be high (e.g. linear, polynomial or worse).

The setup of amortised analysis guarantees that for a sequence of data structure operations the worst-case cost is indeed the number of data structure operations (i.e. the length of given sequence) times the amortised cost. In this way, amortised analysis provides a methodology for worst-case cost analysis.

Notably, the cost analysis of self-adjusting data structures, such as splay trees, has been a motivating example already in the initial proposal of amortised analysis [DR85; RE 85].

These data structures have the behaviour that a single data structure operation might be expensive (i.e. linear in the size of the tree) but the cost is guaranteed to „average out“ in a sequence of data structure operations (i.e. logarithmic in the size of the tree).

Analysing these data structures requires sophisticated potential functions, which typically contain logarithmic expressions. Possibly for these reasons, and despite the recent progress in automated complexity analysis, they have so far eluded automation.

In this thesis, we present the first automated amortised cost analysis of self-adjusting data structures, that is, of *splay trees*, *splay heaps* and *pairing heaps*, for which only results obtained by manual on pen-and-paper, or by using interactive theorem provers, have been reported.

We extend the line of work by Martin Hofmann and his collaborators on amortised analysis, where the search for suitable potential functions is encoded as a type system. This line of work has led to several successful tools for deriving accurate bounds on the resource usage of functional [JKM12a; S J+09; MLG15; MG16] and imperative [HR13; JZ14] programs, as well as term rewriting systems [MGM16; MG14; MG15; MS20]. The cited approaches employ a variety of potential functions: While initially

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

confined to inferring linear cost [MS03], the methods were subsequently extended to cover polynomial [JM10b], multivariate polynomial[JKM12a], and also exponential cost [HR13].

In this thesis, we present the first fully automated amortised cost analysis of self-adjusting data structures, that is, of *splay trees*, *splay heaps* and *pairing heaps*, We for the first time propose a type system that supports logarithmic potential functions (and at the same time enables a multivariate cost analysis).

Our analysis is coached in a simple first-order functional language just sufficiently rich to provide a full definition of our motivating examples (intially splay trees, but also splay heaps and pairing heaps). We employ a big-step semantics, following similar approaches in the literature. We note that this decision only supports the resource analysis of terminating programs. However, it is straightforward to provide a partial big-step semantics [JM10a] or a small-step semantics [MS20] to overcome this limitation. Furthermore, our type system is geared towards runtime as computation cost i.e. we assign a unit cost to each function call and zero cost to every other evaluation step). Again it would not be difficult to provide a parametric type system that supports other cost models. We consider both issues as complementary to our main agenda.

Our type system has been designed with the goal of automation. As in previous work on type-based amortised analysis, the type system infers constraints on unknown coefficients of template potential functions in a syntax directed way from the program under analysis. Suitable coefficients can then be found automatically by solving the collected constraints with a suitable constraint solver (i.e. an SMT solver that supports the theory of linear arithmetic). The derivation of constraints is straightforward for all syntactic constructs of our programming language. However, our automated analysis also requires a *weakening* rule, which supports the comparison of different potential functions. As our potential functions are logarithmic, we cannot directly encode the comparison between logarithmic expressions within the theory of linear arithmetic. Here we propose several ideas for *linearising* the required comparison of logarithmic expressions. The obtained linear constraints can be then be added to the constraint system. Our proposed linearisation makes use of

1. mathematical facts about the logarithm,
2. facts inferred from the program under analysis about the arguments of the logarithmic expressions — we call both these facts *expert knowledge* —, and
3. Farkas' Lemma (Lemma 5) for turning the universally-quantified premise of the weakening rule into an existentially-quantified statement that can be added to the constraint system.

Our work takes the seminal study of Schoenmakers [Sch92; B S93] as a starting point, who has for the first time formulated self-adjusting data structures in a functional setting and analysed the amortised cost of the obtained functional data structures. An important precursor of our work is the recent effort by Nipkow et al. [T N15; NB19], who has verified the amortised cost of these data structures with the interactive theorem prover Isabelle/HOL, which allows for a semi-automated verification (most of the calculations need some manual intervention though).

Achieving full automation required substantial implementation effort as the structural rules need to be applied carefully — as we learned during our experiments — in order to avoid a size explosion of

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

the generated constraint system. We evaluate and discuss our design choices that lead to a scalable implementation.

1.1 State of the Art and Related Work

To the best of our knowledge the established type-and-effect system for the analysis of logarithmic amortised complexity is novel and also the automated resource analysis of self-adjusting data structures like splay trees, is unparalleled in the literature.

The automated cost analysis of imperative, functional and object-oriented programs as well as of more abstract programming paradigms such as term rewriting systems and logic programming is an active research topic [C A+10; Bla+10; GZ10; E A+11; MV +12; MLG15; MGM16; A F17; J G+17].

(Constant) amortised cost analysis has been in particular pioneered by Martin Hofmann and his collaborators. Starting with seminal work on the static prediction of heap space usage [MS03; HR13], the approach has been generalised to (lazy) functional programming [S J+09; S J+10; JKM11; JKM12b; JKM12a] and rewriting [MG14; MG15].

A few papers also target the inference of exponential and logarithmic bounds [Alb+08; AEM11; CFG17; WWC17; KH20; SG20]. Some of the cited approaches are able to conduct an automated amortised analysis in the sense of Sleator and Tarjan: The work on type-based cost analysis by Martin Hofmann and his collaborators [MS03; S J+09; S J+10; J H11; JKM12a; MG14; MG15; JZ15a; JZ15b; S J+17; JAS17], which we discuss in more detail in Section 2.1.1, directly employs potential functions as proposed by Sleator and Tarjan [DR85; RE 85].

For imperative programs, a line of work infers cost bounds from lexicographic ranking functions using arguments that implicitly achieve an amortised analysis [SZV14; SZV15; SZV17; Fie+18] (for details we refer the reader to [SZV17]). The connection between ranking functions and amortised analysis has also been discussed in the context of term rewriting systems [MG14]. Proposals that incorporate amortised analysis within the recurrence relations approach to cost analysis have been discussed in [AG12; A F17]. To the best of our knowledge, none of the cited approaches is able to conduct a worst-case cost analysis for self-adjusting binary search trees such as splay trees. One notable exception is [T N15] where the correct amortised analysis of splay trees [DR85; RE 85] and other data structures is certified in Isabelle/HOL with some tactic support. However, it is not clear if the approach can be further automated.

Automation of amortised resource analysis has also been greatly influenced by Hofmann, yielding to sophisticated tools for the analysis of higher-order functional programs [JM10b; JM10a; J H11], as well as of object-oriented programs [HR13; BJH18]. We mention here the highly sophisticated analysis behind the RaML prototype developed in [JZ14; JZ15a; JZ15b; JAS17] and the RAJA tool [HR13].

We now overview alternatives to conducting amortised cost analysis by means of a type-and-effect system. The line of work [Zul+11; SZV14; SZV15; SZV17; Fie+18] has focused on identifying abstractions resp. abstract program models that can be used for the automated resource analysis of imperative programs. The goal has been to identify program models that are sufficiently rich to support the inference of precise bounds and sufficiently abstract to allow for a scalable analysis, employing the size-change abstraction [Zul+11], (lossy) vector-addition systems [SZV14] and difference-constraint systems [SZV15;

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

SZV17]. This work has led to the development of the tool LOOPUS, which performs amortised analysis for a class of programs that cannot be handled by related tools from the literature. Interestingly, LOOPUS infers worst-case costs from lexicographic ranking functions using arguments that implicitly achieve an amortised analysis (for details we refer the reader to [SZV17]). Another line of work has targeted the resource bound analysis of imperative and object-oriented programs through the extraction of recurrence relations from the program under analysis, whose closed-form solutions then allows one to infer upper bounds on resource usage [Alb+08; E A+11; AG12; A F17]. Amortised analysis with recurrence relations has been discussed for the tools COSTA [AG12] and CoFloCo [A F17]. Amortised analysis has also been employed in the resource analysis for rewriting [MS20] and non-strict function programs, in particular, if *lazy evaluation* is conceived, see [S J+17].

Sublinear bounds are typically not in the focus of these tools, but can be inferred by some tools. In the recurrence relations based approach to cost analysis [Alb+08; E A+11] refinements of linear ranking functions are combined with criteria for divide-and-conquer patterns; this allows their tool PUBS to recognise logarithmic bounds for some problems, but examples such as *mergesort* or *splaying* are beyond the scope of this approach. Logarithmic and exponential terms are integrated into the synthesis of ranking functions in [CFG17], making use of an insightful adaption of Farkas' and Handelman's lemmata. The approach is able to handle examples such as *mergesort*, but again not suitable to handle self-adjusting data structures. A type based approach to cost analysis for an ML-like language is presented in [WWC17], which uses the Master Theorem to handle divide-and-conquer-like recurrences. Very recently, support for the Master Theorem was also integrated for the analysis of rewriting systems [SG20], extending [MG16] on the modular resource analysis of rewriting to so-called logically constrained rewriting systems [FKN17]. The resulting approach also supports the fully automated analysis of *mergesort*.

We also mention the quest for abstract program models whose resource bound analysis problem is decidable, and for which the obtainable resource bounds can be precisely characterised. We list here the size-change abstraction, whose worst-case complexity has been completely characterised as polynomial (with rational coefficients) [CDZ14; Zul15], vector-addition systems [Brá+18; Zul20], for which polynomial complexity can be decided, and LOOP programs [BH19], for which multivariate polynomial bounds can be computed. We are not aware of similar results for programs models that induce logarithmic bounds.

1.2 Contributions

Summarising, we make the following contributions:

- We propose a new class of template potential functions suitable for logarithmic amortised analysis; these potential functions in particular include a variant of Schoenmakers' potential (a key building block for the analysis of the splay function) and logarithmic expressions. Based on these template potential functions, we present a type system for potential-based resource analysis capable of expressing logarithmic amortised costs, and prove its soundness.

- We encode the search for concrete potential function coefficients as an optimisation problem over a suitable constraint system. Our target function steers the search towards coefficients that

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

minimise the inferred amortised complexity. Our approach does not rely on manual annotations, it is a „push button" automation.

- We give the details of our implementation that enable an automated analysis. The main challenge consists in automating the calculations about the logarithmic potential functions. We achieve automation by using Farkas' Lemma (5) for the linear part of the calculations, and isolate monotonicity and a simple inequality between logarithmic expressions as the necessary non-linear facts that need to be added to the linear reasoning.

- We present our tool ATLAS and report on experimental results for *splay trees*, *splay heaps* and *pairing heaps*. We completely automatically infer complexity estimates that match previous results (obtained by sophisticated pen-and-paper proofs), and in some cases even infer better complexity estimates than previously published.

- We report on experimental results for three self-adjusting data structures, that is *splay trees*, *splay heaps* and *pairing heaps*, and automatically infer logarithmic amortised cost for their operations.

The theory presented in this thesis (mainly in Chapter 3) was developed in collaboration with Martin Hofmann, David Obwaller, Georg Moser, and Florian Zuleger. The software tool ATLAS (subject of Chapter 5) was developed solely by the author of this thesis, and can be considered the main contribution of the thesis.

To some extent, the theory and the tool implementing it were developed in parallel, with a synergistic effect. Improvements to the theory, caused by advancing the implementation, pointing out problematic cases, and asking the right questions, were part of this process. Chapter 4 gives some insight into the intersection of the two parts.

To provide sufficient context, this thesis also combines the contents of the following two publications, fruits of the collaborative effort mentioned above, in the style of an extended version.

1. Martin Hofmann et al. „Type-Based Analysis of Logarithmic Amortised Complexity". In: *Mathematical Structures Computer Science* (2021). to appear

2. Lorenz Leutgeb, Georg Moser, and Florian Zuleger. „ATLAS: Automated Amortised Complexity Analysis of Self-Adjusting Data Structures". In: *Computer Aided Verification - 33rd International Conference, CAV 2021, July 18-23, Online, 2021, Proceedings*. Lecture Notes in Computer Science. to appear. Springer, 2021

1.2.1 New Results for Amortised Complexity Analysis of Self-Adjusting Data Structures

We either improve the best known complexity bounds or provide new (alternative) proofs for known complexity bounds. In Table 1.1 we state the complexity bounds computed by ATLAS next to results from the literature. We match or improve the results from [Sch92; B S93; NB19]. To the best of our knowledge, the bounds for splay trees and splay heaps represent the state-of-the-art. We improve the

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

Function	ATLAS	[Sch92][1]	[NB19]
SplayTree			
splay	$^3/_2 \log(\lvert t \rvert)$	$^3/_2 \log(\lvert t \rvert) + 1$	$^3/_2 \log(\lvert t \rvert) + 1$
splay_max	$^3/_2 \log(\lvert t \rvert)$	—	$^3/_2 \log(\lvert t \rvert) + 1$
insert	$2 \log(\lvert t \rvert) + {^3/_2}$	$2 \log(\lvert t \rvert + 1) + O(1)$	$2 \log(\lvert t \rvert) + {^3/_2}$
delete	$^5/_2 \log(\lvert t \rvert) + 3$	$3 \log(\lvert t \rvert + 1) + O(1)$	$3 \log(\lvert t \rvert) + 2$
SplayHeap			
partition	$^3/_4 \log(\lvert t \rvert) + \log(\lvert t \rvert + 1)$	—	$2 \log(\lvert t \rvert + 1) + 1$
insert	$^3/_4 \log(\lvert t \rvert) + \log(\lvert t \rvert + 1) + {^3/_2}$	—	$3 \log(\lvert t \rvert + 2) + 1$
del_min	$\log(\lvert t \rvert)$	—	$2 \log(\lvert t \rvert + 1) + 1$
PairingHeap			
merge_pairs	$^3/_2 \log(\lvert h \rvert)$	—	$3 \log(\lvert h \rvert) + 4$
insert	$^1/_2 \log(\lvert h \rvert)$	—	$\log(\lvert h \rvert + 1) + 1$
merge	$^1/_2 \log(\lvert h_1 \rvert + \lvert h_2 \rvert) + 1$	$^1/_2 \log(\lvert h_1 \rvert + \lvert h_2 \rvert)$	$\log(\lvert h_1 \rvert + \lvert h_2 \rvert + 1) + 2$
del_min	$\log(\lvert h \rvert)$	$\log(\lvert h \rvert)$	$3 \log(\lvert h \rvert + 1) + 4$

Table 1.1: Amortised complexity bounds for splay trees (module name `SplayTree`, abbrev. `ST`), splay heaps (`SplayHeap`, `SH`) and pairing heaps (`PairingHeap`, `PH`).

bound for the `delete` function of splay trees and all bounds for the splay heap functions. For pairing heaps, Iacono [Iac00; IY16] has proven (using a more involved potential function) that `insert` and `merge` have constant amortised complexity, while the other data structure operations continue to have an amortised complexity of $k \log(|t|)$; while we leave an automated analysis based on Iacono's potential function for future work, we note that his coefficients k in the logarithmic terms are large, and that therefore the small coefficients in Table 1.1 are still of interest. We will detail below that we used a simpler potential function than [Sch92; BS93; NB19] to obtain our results. Hence, also the new proofs of the confirmed complexity bounds can be considered a contribution.

1.2.2 A New Approach for the Complexity Analysis of Data Structures

Establishing the prior results in Table 1.1 required considerable effort. Schoenmakers studied in his PhD thesis [Sch92] the best amortised complexity bounds that can be obtained using a parametrised potential function $\phi(t)$, where t is a binary tree, defined by $\phi(\texttt{leaf}) := 0$ and $\phi((l, d, r)) := \phi(l) + \beta \log_\alpha(|l|+|r|) + \phi(r)$, for real-valued parameters $\alpha, \beta > 0$. Carrying out a sophisticated optimisation with pen and paper, he concluded that the best bounds are obtained by setting $\alpha = \sqrt[3]{4}$ and $\beta =$

[1] [Sch92] uses a different cost metric, i.e. the numbers of arithmetic comparisons, whereas we and [NB19] count the number of (recursive) function applications. We adapted the results of [Sch92] to our cost metric to make the results easier to compare, i.e. the coefficients of the logarithmic terms are by a factor 2 smaller compared to [Sch92].

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

$\frac{1}{3}$ for splay trees, and by setting $\alpha = \sqrt{2}$ and $\beta = \frac{1}{2}$ for pairing heaps (splay heaps were proposed only some years later by Okasaki in [C O99]). Brinkop and Nipkow verify their complexity results for splay trees in the theorem prover Isabelle/HOL [NB19]. They note that manipulating the expressions corresponding to $\beta \log_\alpha(|t|)$ could only partly be automated.[2] For splay heaps, there is to the best of our knowledge no previous attempt to optimise the obtained complexity bounds, which might explain why our optimising analysis was able to improve all bounds. For pairing heaps, Brinkop and Nipkow did not use the optimal parameters reported by Schoenmakers — probably in order to avoid reasoning about polynomial inequalities —, which explains the worse complexity bounds. In contrast to the discussed approaches, we were able to verify and improve the previous results fully automatically. Our approach uses a variation of Schoenmakers' potential function, where we roughly fix $\alpha = 2$ and leave β as a parameter for the optimisation phase (see Section 2.1 for more details). Despite these choices, our approach was able to derive bounds that match and improve the previous results, which came as a surprise to us. Looking back at our experiments and interpreting the obtained results, we recognise that we might have been in luck with the particular choice of the potential function (because we can obtain the previous results despite fixing $\alpha = 2$). However, we would not have expected that an automated analysis is able to match and improve all previously reported coefficients, which shows the power of the optimisation phase.

We believe that our results suggest a new approach for the complexity analysis of data structures. So far, self-adjusting data structures had to be analysed manually. This is possibly due to the use of sophisticated potential functions, which may contain logarithmic expressions. Both features are challenging for automated reasoning. Our results suggest that the following alternative (see Sections 2.1 and 4.1 for more details):

1. Fix a parametrised potential function;
2. derive a (linear) constraint system over the function parameters from the AST of the program;
3. capture the required non-linear reasoning in lemmata, and use Farkas' Lemma (Lemma 5) to integrate the application of these lemmata into the constraint system (in our case two lemmata, one about an arithmetic property and one about the monotonicity of the logarithm, were sufficient for all of our benchmarks); and finally
4. find values for the parameters by an (optimising) constraint solver.

We believe that our approach will carry over to other data structures: one needs to adapt the potential functions and add suitable lemmata, but the overall setup will be the same. We compare the proposed methodology to program synthesis by sketching [Sol09], where the synthesis engineer communicates her main insights to the synthesis engine (in our case the potential functions plus suitable lemmata), and a constraint solver then fills in the details.

[2] Nipkow et al. [NB19] state „The proofs in this subsection require highly nonlinear arithmetic. Only some of the polynomial inequalities can be automated with Harrison's sum-of-squares method [Har07].“

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

1.3 Outline

The rest of this thesis is organised as follows:

In Chapter 2, to set the stage (Section 2.1) we review the key concepts underlying type-based amortised analysis (Sections 2.1.1 and 2.1.2) and present our ideas for their extension (Sections 2.1.1 and 2.1.2 respectively). We also present a necessarily simple but at the same time sufficiently complex programming languge (Section 2.2) to be used in the later chapters, and spell out the motivating example of splay trees in this programming language (Section 2.3).

Chapter 3 presents a type system for logarithmic amoritsed resource analysis. We first discuss resource functions in Section 3.1, then present the type system and its rules in Section 3.2, and finally apply it to analyse two programs in Section 3.3.

Chapter 4 bridges between the theory itself (Chapter 3) and its implementation (Chapter 5): A number of challenges at this intersection needed to be solved. We group them as follows: In Section 4.1 we address translation of non-linear properties of the logarithm into a workable linear constraint system as well as clarify the role of Farkas' Lemma. The steps required to go from type checking to type inference are covered in Section 4.2.

The implementation of the tool is described in Chapter 5. We report experimental results for splay trees, splay heaps and pairing heaps in Section 5.3.

We conclude in Chapter 6.

1.4 The Physicist's Method of Amortised Analysis

Before we elaborate any further, we revisit the seminal work [DR85; RE 85] introducing amortised analysis, since it is foundational for this thesis. Originally, amortised analysis was presented from two points of view, called the *the banker's view* and *the physicist's view*, respectively. In this section we focus on the latter, since it is the approach taken in this thesis.

In the physicist's view, amoritsed analysis revolves around the notion of a *potential function*. A potential function $\Phi(D)$ maps any datastructure configuration D into a number. We call $\Phi(D)$ the *potential* of D.

The idea is to use this concept of potential to reason about the *amortised cost* of an operation performed on the datastructure. The notion of *amortised cost* relates to the actual cost as follows: For an operation f with actual cost $c(f)$, the amortised cost of f is $a(f) \coloneqq c(f)(D) + \Phi(f(D)) - \Phi(D)$, i.e. the sum of the actual cost of performing f on D and the potential of the result, minus the potential of D.

The analogy at work is that of physical objects conserving potential energy in classical mechanics: Moving an object higher up (compared to some reference height) requires work, and will increase its gravitational potential energy. This energy is converted back to kinetic energy when the object moves back down. Operating on datastructures is thus analoguos to moving and up or down in space, depending on the characteristics of the potential function and the operation.

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

Another key aspect of amortised analysis is that it considers sequences of operations. We address this next. We use $\circ$ to denote composition of operations. Amortised cost generalises in a simple way:

$$a(f \circ g)(D) \coloneqq a(g)(D) + a(f)(g(D)) \tag{1.1}$$

Further, we use exponentiation f^n to denote repeated composition of an operation with itself. We set $\forall f \forall x. f^0(x) = x$.

$$a(\underbrace{f \circ \cdots \circ f}_{n \text{ times}})(D) = a(f^n)(D) \coloneqq \begin{cases} a(f)(D) & n = 1 \\ a(f)(D) + a(f^{n-1})(f(D)) & \text{otherwise} \end{cases} \tag{1.2}$$

With the convention that the empty composition is equivalent to its unit, identity, which does not incur any cost, we can express this recurrence as a sum.

$$a(f_n \circ \cdots \circ f_1)(D) = \sum_{i=1}^{n} c(f_i)(f_{i-1} \circ \cdots \circ f_1(D)) + \Phi(f_i \circ \cdots \circ f_1(D)) - \Phi(f_{i-1} \circ \cdots \circ f_1(D)) \tag{1.3}$$

Note that two of the three terms in the sum telescope. By exploiting this, we arrive at a more direct form that only talks about the potential before applying any operation, $\Phi(D)$, and the potential after all operations have been applied, $\Phi(f_n \circ \cdots \circ f_1(D))$.

$$a(f_n \circ \cdots \circ f_1)(D) = \Phi(f_n \circ \cdots \circ f_1(D)) - \Phi(D) + \sum_{i=1}^{n} c(f_i)(f_{i-1} \circ \cdots \circ f_1(D)) \tag{1.4}$$

We also use $c(f_n \circ \cdots \circ f_1) \coloneqq \sum_{i=1}^{n} c(f_i)(f_{i-1} \circ \cdots \circ f_1(D))$ to refer to the actual cost of applying operations f_1 through f_n.

To use amortised cost as an upper bound for actual cost, we impose two restrictions on Φ. Firstly, $\Phi(D) = 0$, which is to say that the initial potential is zero. This first condition is also intuitive in the sense that an empty datastructure stores no data and therefore no „fuel" for computation. And secondly, $\forall D. \Phi(D) \geq 0$ which avoids „borrowing" potential when there is none left. This gives an upper bound for the actual cost of repeated application of f:

$$\begin{aligned} a(f^n)(D) \geq c(f^n)(D) &= a(f_n \circ \cdots \circ f_1)(D) - \Phi(f_n \circ \cdots \circ f_1(D)) + \Phi(D) \\ &= a(f_n \circ \cdots \circ f_1)(D) - \Phi(f_n \circ \cdots \circ f_1(D)) \end{aligned}$$

With this machinery, cost analysis is reframed as the task of choosing Φ in such a way that the difference between amortised cost and actual cost is minimal.

Note that amortised analysis has a compositional character. Even though we analyse each operation only once, we arrive at cost for a sequence of operations. Amortised analysis allows to assign amortised cost that is logarithmic in the size of the input data structure to an operation, even though the operation analysed in isolation would yield higher, e.g. linear worst-case cost.

The challenge raised is to find a suitable Φ, with the goal of establishing bounds that are as tight as possible. In automated analysis, our goal is to develop methods that will find a definition for Φ that yields bounds for a given set of operations.

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

The above notion of amortised cost for sequences of operations does not adequately capture operations that take two or more datastructures as input (e.g. merging/union, difference). For a generalisation from sequences to trees, refer to [NB19, Sec. 3].

Considering statically typed functional programming languages, operations on the data structure $\mathcal{D}$ are functions $f : \mathcal{D} \to \mathcal{D}$. With partial application and polymorphism, examples would be `insert` $: \alpha \to \mathcal{D}\alpha \to \mathcal{D}\alpha$ and `delete` $: \alpha \to \mathcal{D}\alpha \to \mathcal{D}\alpha$. The approach taken in this thesis is to encode the Φ in the types. In later chapters, we will define some structure Q which characterises Φ, and thus, we get types of the form $f : \mathcal{D}\alpha|Q \to \mathcal{D}\alpha|Q'$.

Soundness of the modified type system will mean that $f : \mathcal{D}\alpha|Q \to \mathcal{D}\alpha|Q'$ implies $\Phi_Q(D) - \Phi_{Q'}(f(D)) \geq c(f)(D)$. Annotations Q and Q' are constrained by typing rules, and thus the task of choosing them is reframed as type inference.

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

CHAPTER 2

Preliminaries

In this chapter, we briefly present the state-of-the-art that our approach builds on. The goal is to highlight similarities between existing polynomial amortised analysis to our logarithmic analysis, and marking points for departure, such as cost-free typing. We also present our programming language.

2.1 Setting the Stage

Our analysis is formulated in terms of the physicist's method of amortised analysis in the style of Sleator and Tarjan [DR85; RE 85]. This method assigns a *potential* to data structures of interest and defines the *amortised cost* of an operation as the sum of the actual cost plus the change of the potential through execution of the operation, i.e. the central idea of an amortised analysis as formulated by Sleator and Tarjan is to choose a potential function ϕ such that

$$\phi(v) + a_f(v) = c_f(v) + \phi(f(v)) ,$$

holds for all inputs v to a function f, where a_f, c_f denote the amortised and total cost, respectively, of executing f. Hofmann et al. [MS03; JKM11; JKM12b; JKM12a; MG14; MG15] provide a generalisation of this idea to a set of potential functions ϕ, ψ, such that

$$\phi(v) \geqslant c_f(v) + \psi(f(v)) ,$$

holds for all inputs v. This allows one to read off an upper bound on the amortised cost of f, i.e. we have $a_f(v) \leqslant \phi(v) - \psi(v)$. We add that the above inequality indeed generalises the original formulation, which can be seen by setting $\phi(v) := a_f(v) + \psi(v)$.

In this thesis, we present a type-based resource analysis based on the idea of potential functions that can infer logarithmic amortised cost. Following previous work by Hofmann et al., we tackle two key problems in order to achieve a semi-automated logarithmic amortised analysis:

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

1) Automation is achieved by a *type-and-effect system* that uses *template potential functions*, i.e. functions of a fixed shape with indeterminate coefficients. Here, the key challenge is to identify templates that are suitable for logarithmic analysis and that are closed under the basic operations of the considered programming language.

2) In addition to the actual amortised analysis with costs, we employ *cost-free* analysis as a subroutine, setting the amortised a_f and actual costs c_f of all functions f to zero. This enables a *size analysis* of sorts, because the inequality $\phi(v) \geqslant \psi(f(v))$ bounds the size of the potential $\psi(f(v))$ in terms of the potential $\phi(v)$. The size analysis we conduct allows lifting the analysis of a subprogram to a larger context, which is crucial for achieving a *compositional analysis*.

2.1.1 Type-and-Effect System

To set the scene, we briefly review amortised analysis formulated as a type-and-effect system up to and including the multivariate polynomial analysis, see [SJ+10; JM10b; JM10a; JKM11; JKM12a; MG14; MG15; JAS17; SJ+17].

Polynomial Amortised Analysis

Suppose that we have types $\alpha, \beta, \gamma, \ldots$ representing sets of values. We write $[\![\alpha]\!]$ for the set of values represented by type α. Types may be constructed from base types such as Booleans and integers, denoted by Base, and by type formers such as list, tree, product, sum, etc. For each type α, we define a (possibly infinite) set of *basic potential functions* $\mathcal{BF}(\alpha)\colon [\![\alpha]\!] \to \mathbb{R}_0^+$. Thus, if $p \in \mathcal{BF}(\alpha)$ and $v \in [\![\alpha]\!]$ then $p(v) \in \mathbb{R}_0^+$. An *annotated type* is a pair of a type α and a function $Q : \mathcal{BF}(\alpha) \to \mathbb{R}_0^+$ providing a coefficient for each *basic potential function*. The function Q must be zero on all but finitely many basic potential functions. For each annotated type $\alpha|Q$, the *potential function* $\phi_Q : [\![\alpha]\!] \to \mathbb{R}_0^+$ is then given by

$$\phi_Q(v) \coloneqq \sum_{p \in \mathcal{BF}(\alpha)} Q(p) \cdot p(v) \ .$$

By introducing product types, one can regard functions with several arguments as unary functions, which allows for technically smooth formalisations, see [JM10b; JM10a; JH11]; the analyses in the cited papers are called *univariate* as the set of basic potential functions $\mathcal{BF}(\alpha)$ of a product type α is given directly. In the later *multivariate* versions of automated amortised analysis [JKM11; JKM12a; MG15] one takes a more fine-grained approach to products. Namely, one then sets (for arbitrary n)

$$\mathcal{BF}(\alpha_1 \times \cdots \times \alpha_n) \coloneqq \mathcal{BF}(\alpha_1) \times \cdots \times \mathcal{BF}(\alpha_n) \ ,$$

$$(p_1, \ldots, p_n)(v_1, \ldots, v_n) \coloneqq \prod_{i=1}^{n} p_i(v_i) \ .$$

Thus, the basic potential function for a product type is obtained as the multiplication of the basic potential functions of its constituents.[1]

[1] Suppose that for each type α there exists a distinguished element $u \in \mathcal{BF}(\alpha)$ with $u(a) = 1$ for all $a \in [\![\alpha]\!]$. Then, the multivariate product types contain all (linear combinations) of the basic potential functions, extending earlier univariate definitions of product types.

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

Automation

The idea behind this setup is that the basic potential functions $\mathcal{BF}(\alpha)$ are suitably chosen and fixed by the analysis designer, the coefficients $Q(p)$ for $p \in \mathcal{BF}(\alpha)$, however, are left indeterminate and will (automatically) be fixed during the analysis. For this, constraints over the unknown coefficients are collected in a syntax-directed way from the function under analysis and then solved by a suitable constraint solver. The type-and-effect system formalises this collection of constraints as typing rules, where for each construct of the considered programming language a typing rule is given that corresponds to constraints over the coefficients of the annotated types. Expressing the quest for suitable type annotations as a type-and-effect system allows one to compose typing judgements in a syntax-oriented way without the need for fixing additional intermediate results, which is often required by competing approaches. This syntax-directed approach to amortised analysis has been demonstrated to work well for datatypes like lists or trees whose basic potential functions are polynomials over the length of a list resp. the number of nodes of a tree. One of the reasons why this works well is, e.g., that functional programming languages typically include dedicated syntax for list construction and that polynomials are closed under addition by one (i.e. if $p(n)$ is a polynomial, so is $p(n+1)$), supporting the formulation of a suitable typing rule for list construction, see [JM10b; JM10a; J H11; JKM11; JKM12a]. The syntax-directed approach has been shown to generalise from lists and trees to general inductive data types, see [MG14; MG15; GM18].

Logarithmic Amortised Analysis

We now motivate the design choices of our type-and-effect system. The main objective of our approach is the automated analysis of data structures such as splay trees, which have *logarithmic* amortised cost. The amortised analysis of splay trees is tricky and requires choosing an adequate potential function: our work makes use of a variant of Schoenmakers' potential, $\mathsf{rk}(t)$ for a tree t, see [B S93; T N15], defined inductively by

$$\begin{aligned}\mathsf{rk}(\mathtt{leaf}) &\coloneqq 1,\\ \mathsf{rk}((l\,,\,d\,,\,r)) &\coloneqq \mathsf{rk}(l) + \log(|l|) + \log(|r|) + \mathsf{rk}(r)\ ,\end{aligned}$$

where l, r are the left resp. right child of the tree $(l\,,\,d\,,\,r)$, $|t|$ denotes the number of leaves of a tree t, and d is some data element that is ignored by the potential function. Besides Schoenmakers' potential we need to add further basic potential functions to our analysis. This is motivated as follows: Similar to the polynomial amortised analysis discussed above we want that the basic potential functions can express the construction of a tree, e.g., let us consider the function

$$f(x, d, y) \coloneqq (x\,,\,d\,,\,y),$$

which constructs the tree $(x\,,\,d\,,\,y)$ from some trees x, y and some data element d, and let us assume a constant cost $c_f(x, y) = 1$ for the function f. A type annotation for f is given by

$$\underbrace{\mathsf{rk}(x) + \log(|x|) + \mathsf{rk}(y) + \log(|y|) + 1}_{\phi(x,y)} \geqslant c_f(x, y) + \underbrace{\mathsf{rk}(f(x, d, y))}_{\psi(f(x,y))}\ ,$$

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

i.e. the potential $\phi(x, y)$ suffices to pay for the cost c_f of executing f and the potential of the result $\psi(f(x, y))$ (the correctness of this annotation can be established directly from the definition of Schoenmakers' potential). As mentioned above, the logarithmic expressions in $\phi(x, y)$, i.e. $\log(|x|) + \log(|y|) + 1$, specify the *amortised costs* of the operation.

We see that in order to express the potential $\phi(x, y)$ we also need the basic potential functions $\log(|t|)$ for a tree t. In fact, we will choose the slightly richer set of basic potential functions

$$p_{(a,b)}(t) = \log(a \cdot |t| + b) ,$$

where $a, b \in \mathbb{N}$ and t is a tree. We note that by setting $a = 0$ and $b = 2$ this choice allows us to represent the constant function u with $u(t) = 1$ for all trees t. We further note that this choice of potential functions is sufficiently rich to express that $p_{(a,b)}(t) = p_{(a,b+a)}(s)$ for trees s, t with $|t| = |s| + 1$, which is needed for precisely expressing the change of potential when a tree is extended by one node. Further, we define basic potential functions for products of trees by setting

$$p_{(a_1,\ldots,a_n,b)}(t_1, \ldots, t_n) = \log(a_1 \cdot |t_1| + \cdots + a_n \cdot |t_n| + b) ,$$

where $a_1, \ldots, a_n, b \in \mathbb{N}$ and $t_1, \ldots, t_n$ is a tuple of trees. This is sufficiently rich to state the equality $p_{(a_0,a_1,\ldots,a_n,b)}(x_1, x_1, \ldots, x_n) = p_{(a_0+a_1,\ldots,a_n,b)}(x_1, \ldots, x_n)$, which supports the formulation of a *sharing* rule, which in turn is needed for supporting the let-construct in functional programming; see [JKM11; JKM12a; MG15] for a more detailed exposition on the sharing rule and the let-construct.

2.1.2 Cost-Free Semantics

Polynomial Amortised Analysis

We begin by reviewing the cost-free semantics underlying previous work [JM10a; J H11; JKM11; JKM12a] on polynomial amortised analysis. Assume that we want to analyse the composed function call $g(f(\vec{x}), \vec{z})$ using already established analysis results for $f(\vec{x})$ and $g(y, \vec{z})$. Suppose we have already established that for all $\vec{x}$, y, $\vec{z}$ we have:

$$\phi_0(\vec{x}) \geqslant c_f(\vec{x}) + \beta(f(\vec{x})) \tag{2.1}$$

$$\phi_i(\vec{x}) \geqslant \phi'_i(f(\vec{x})) \quad \text{for all } i \ (0 < i \leqslant n) \tag{2.2}$$

$$\beta(y) + \gamma(\vec{z}) + \sum_{i=1}^{n} \phi'_i(y)\phi''_i(\vec{z}) \geqslant c_g(y, \vec{z}) + \psi(g(y, \vec{z})) , \tag{2.3}$$

where as in the multivariate case above, n is arbitrary and equations (2.1) and (2.3) assume cost, while equation (2.2) is *cost-free*. Then, we can conclude for all $\vec{x}$, $\vec{z}$ that

$$\underbrace{\phi_0(\vec{x}) + \gamma(\vec{z}) + \sum_{i=1}^{n} \phi_i(\vec{x})\phi''_i(\vec{z})}_{\phi(\vec{x},\vec{z})} \geqslant c_f(\vec{x}) + c_g(f(\vec{x}), \vec{z}) + \psi(g(f(\vec{x}), \vec{z})) ,$$

guaranteeing that the potential $\phi(\vec{x}, \vec{z})$ suffices to pay for the cost $c_f(\vec{x})$ of computing $f(\vec{x})$, the cost $c_g(f(\vec{x}), \vec{z})$ of computing $g(f(\vec{x}), \vec{z})$ and the potential $\psi(g(f(\vec{x}), \vec{z}))$ of the result $g(f(\vec{x}), \vec{z})$. We

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

note that the correctness of this inference hinges on the fact that we can multiply equation (2.2) with $\phi_i''(\vec{z})$ for $i = 1 \dots n$, using the monotonicity of the multiplication operation (note that potential functions are non-negative). We highlight that the multiplication argument works well with cost-free semantics, and enables lifting the resource analysis of $f(\vec{x})$ and $g(y, \vec{z})$ to the composed function call $g(f(\vec{x}), \vec{z})$.

Remark 1. *We point out that the above exposition of cost-free semantics in the context of polynomial amortised analysis differs from the motivation given in the literature [JM10a; J H11; JKM11; JKM12a], where cost-free semantics are motivated by the quest for* resource polymorphism*, which is the problem of computing (a representation of) all polynomial potential functions (up to a fixed maximal degree) for the program under analysis; this problem has been deemed of importance for the handling of non tail-recursive programs. We add that for the amortised cost analysis of inductively generated data-types, the cost-free semantics proved necessary even for handling basic data structure manipulations [MG14; MG15; MS20]. In our view, cost-free semantics incorporate a* size analysis *of sorts. We observe that equation* (2.2) *states that the potential of the result of the evaluation of $f(\vec{x})$ is bounded by the potential of the function arguments $\vec{x}$, without accounting for the costs of this evaluation. Thus, for suitably chosen potential functions ϕ_i, ϕ_i' can act as* norms *and capture the size of the result of the evaluation $f(\vec{x})$ in relation to the size of the argument. As stated above, a separated cost and size analysis enables a compositional analysis, an insight that we also exploit for logarithmic amortised analysis.*

Logarithmic Amortised Analysis

Similar to the polynomial case, we want to analyse the composed function call $g(f(\vec{x}), \vec{z})$ using already established analysis results for $f(\vec{x})$ and $g(y, \vec{z})$. However, now we extend the class of potential functions to sublinear functions. Assume that we have already established that

$$\phi_0(\vec{x}) \geqslant c_f(\vec{x}) + \beta(f(\vec{x})) \tag{2.4}$$

$$\log(\phi_i(\vec{x})) \geqslant \log(\phi_i'(\vec{x})) \qquad \text{for all } i \ (0 < i \leqslant n) \tag{2.5}$$

$$\beta(y) + \gamma(\vec{z}) + \sum_{i=1}^{n} \log(\phi_i'(y) + \phi_i''(\vec{z})) \geqslant c_g(y, \vec{z}) + \psi(g(y, \vec{z})) \ , \tag{2.6}$$

where equations (2.4) and (2.6) assume cost, while equation (2.5) is *cost-free*. Equations (2.4) and (2.5) represent the result of an analysis of $f(\vec{x})$ (note that these equations do not contain the parameters $\vec{z}$, which will however be needed for the analysis of $g(f(\vec{x}), \vec{z})$), and equation (2.6) the result of an analysis of $g(y, \vec{z})$. Then, we can conclude for all $\vec{x}, y, \vec{z}$ that

$$\underbrace{\phi_0(\vec{x}) + \gamma(\vec{z}) + \sum_{i=1}^{n} \log(\phi_i(\vec{x}) + \phi_i''(\vec{z}))}_{\phi(\vec{x},\vec{z})} \geqslant c_f(\vec{x}) + c_g(f(\vec{x}), \vec{z}) + \psi(g(f(\vec{x}), \vec{z})) \ ,$$

guaranteeing that the potential $\phi(\vec{x}, \vec{z})$ suffices to pay for the cost $c_f(\vec{x})$ of computing $f(\vec{x})$, the cost $c_g(f(\vec{x}), \vec{z})$ of computing $g(f(\vec{x}), \vec{z})$ and the potential $\psi(g(f(\vec{x}), \vec{z}))$ of the result $g(f(\vec{x}), \vec{z})$. Here, we crucially use monotonicity of the logarithm function, as formalised in Lemma 2. This reasoning allows us to lift isolated analyses of the functions $f(\vec{x})$ and $g(y, \vec{z})$ to the composed function call $g(f(\vec{x}), \vec{z})$; this is what is required for a compositional analysis!

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

Example 1. *We now illustrate the compositional reasoning on an example. We reconsider the function* $f(x, d, y) := (x, d, y)$*, which takes two trees* x, y *and some data element d and returns the tree* (x, d, y)*. Assume that we already have established that*

$$\psi(x) + \psi(y) + 1 \geqslant c_f(x, y) + \mathsf{rk}(f(x, d, y)) \tag{2.7}$$
$$\log(|x| + |y|) \geqslant \log(|f(x, d, y)|)\,, \tag{2.8}$$

where $\psi(u) = \mathsf{rk}(u) + \log(|u|)$*,* $c_f(x, y) = 1$*, and d is an arbitrary data element, which is not relevant for the cost analysis of* f*. We now want to analyse the composed function* $h(x, a, y, b, z) := f(f(x, a, y), b, z)$*. We will use the above reasoning, instantiating equations (2.4) and (2.5) with equations (2.7) and (2.8) for the inner function call* $f(x, a, y)$*, and equation (2.6) with the sum of equations (2.7) and (2.8) for the outer function call* $f(u, b, z)$*. As argued above, we can then conclude for all* x, y, z *that*

$$\begin{aligned}\psi(x) + \psi(y) + \psi(z) + \log(|x| + |y|) + \log(|x| + |y| + |z|) + 2 \geqslant \\ \geqslant c_f(x, a, y) + c_f(f(x, a, y), b, z) + \psi(f(f(x, y), z))\,,\end{aligned}$$

is a valid type annotation for $h(x, a, y, b, z) := f(f(x, a, y), b, z)$*; we have used equation (2.8) twice in this derivation, once as* $\log(|x| + |y|) \geqslant \log(|f(x, a, y)|)$ *and once lifted as* $\log(|x| + |y| + |z|) \geqslant \log(|f(x, a, y)| + |z|)$*. Kindly note that the above example appears in similar form as part of the analysis of the* `splay` *function described in Section 3.3.*

2.2 A Necessarily Simple and Sufficiently Complex Programming Language

In this section we introduce a first-order programming language that will be used throughout the later chapters. It is designed to be as simple as possible and comfortable, while still allowing to define all operations on splay trees (presented as defined in [TN15] below, and analysed in detail in Section 3.3.2) which are the primary motivating example.

2.2.1 Syntax

Consider following grammar in a BNF-like style that defines expressions e. Note that d corresponds to a function definition, while f is to be substituted by the name of a function definition and $x, x_1, \ldots, x_n$ are to be substituted by variable names.

Definition 1 (Syntax).

$$\circ ::= \texttt{<} \mid \texttt{>} \mid \texttt{=} \qquad \textit{Comparison}$$
$$d ::= f\, x_1\, \ldots\, x_n \texttt{ = } e \qquad \textit{Function Definition}$$
$$\begin{aligned} e ::= \; & \mid f\, x_1\, \ldots\, x_n && \textit{Function Application} \\ & \mid \texttt{true} \mid \texttt{false} \mid e_1 \circ e_2 && \mid \texttt{if}\; x\; \texttt{then}\; e_1\; \texttt{else}\; e_2 \\ & \mid (x_1, x_2, x_3) \mid \texttt{leaf} && \mid \texttt{match}\; x\; \texttt{with} \mid \texttt{leaf} \rightarrow e_1 \mid (x_1, x_2, x_3) \rightarrow e_2 \\ & \mid \texttt{let}\; x \texttt{ = } e_1\; \texttt{in}\; e_2 && \mid x \end{aligned}$$

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

We skip the standard definition of integer constants $n \in \mathbb{Z}$ as well as variable declarations, cf. [B P02].

In the definition of syntax above and semantics and typing rules below, expressions are given in *let-normal-form* for simplicity. On the other hand, examplary code will not be presented in let-normal-form for readability. A translation to let-normal-form is described in Chapter 5.

2.2.2 Semantics

To make the presentation more succinct, we assume only the following types: Boolean values $\mathsf{Bool} = \{\texttt{true}, \texttt{false}\}$, an abstract base type Base (abbrev. B), product types, and binary trees Tree (abbrev. T), whose internal nodes are labelled with elements $b : \mathsf{Base}$. We use lower-case Greek letters for the denotation of types. Elements $t : \mathsf{Tree}$ are defined by the following grammar which fixes notation.

$$t ::= \texttt{leaf} \mid \texttt{(}t_1\texttt{,}\ b\texttt{,}\ t_2\texttt{)}\ .$$

The size of a tree is the number of leaves: $|\texttt{leaf}| := 1, |\texttt{(}t\texttt{,}\ a\texttt{,}\ u\texttt{)}| := |t| + |u|$.

Furthermore, we omit binary operators, and only define essential comparisons. For our analysis, these are unimportant, as long as we assume that no actual costs are emitted.

A *typing context* is a mapping from variables $\mathcal{V}$ to types. Type contexts are denoted by upper-case Greek letters (usually Γ, Δ). A program P consists of a signature $\mathcal{F}$ together with a set of function definitions of the form $f(x_1, \ldots, x_n) = e$, where the x_i are variables and e an expression. A *substitution* or (*environment*) σ is a mapping from variables to values that respects types. Substitutions are denoted as sets of assignments: $\sigma = \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$. We write $\mathsf{dom}(\sigma)$ ($\mathsf{rg}(\sigma)$) to denote the domain (range) of σ. Let σ, τ be substitutions such that $\mathsf{dom}(\sigma) \cap \mathsf{dom}(\tau) = \varnothing$. Then we denote the (disjoint) union of σ and τ as $\sigma \uplus \tau$. We employ a simple cost-sensitive big-step semantics based on eager evaluation, whose rules are given in Figure 2.1. The judgement $\sigma \overset{\ell}{\vdash} e \Rightarrow v$ means that under environment σ, expression e is evaluated to value v in exactly ℓ steps. Here only rule applications emit (unit) costs.

2.3 Motivating Example: Splay Trees

Splay trees have been introduced by Sleator and Tarjan [DR85; RE 85] as self-adjusting binary search trees with strictly increasing inorder traversal. There is no explicit balancing condition. All operations rely on a tree rotating operation dubbed *splaying*; `splay a t` is performed by rotating element a to the root of tree t while keeping inorder traversal intact. If a is not contained in t, then the last element found before `leaf` is rotated to the tree. The complete definition is given in Figure 2.2. Based on splaying, searching is performed by splaying with the sought element and comparing to the root of the result. Similarly, the definition of insertion and deletion depends on splaying. As an example the definition of insertion and delete is given in Figure 2.3 and 2.4 respectively. See also [T N15] for full algorithmic, formally verified, descriptions.

All basic operations can be performed in $O(\log n)$ amortised runtime. The logarithmic amortised complexity is crucially achieved by local rotations of subtrees in the definition of `splay`. Amortised cost analysis of splaying has been provided for example by Sleator and Tarjan [DR85], Schoenmakers [B

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

$$\frac{}{\sigma \stackrel{0}{\vdash} \texttt{false} \Rightarrow \texttt{false}} \qquad \frac{}{\sigma \stackrel{0}{\vdash} \texttt{true} \Rightarrow \texttt{true}} \qquad \frac{}{\sigma \stackrel{0}{\vdash} \texttt{leaf} \Rightarrow \texttt{leaf}}$$

$$\frac{x_1\sigma = t \quad x_2\sigma = b \quad x_3\sigma = u}{\sigma \stackrel{0}{\vdash} \texttt{(}x_1\texttt{,}x_2\texttt{,}x_3\texttt{)} \Rightarrow \texttt{(}t\texttt{,}b\texttt{,}u\texttt{)}} \qquad \frac{x\sigma = v}{\sigma \stackrel{0}{\vdash} x \Rightarrow v} \qquad \frac{b \text{ is value of } x_1\sigma \circ x_2\sigma}{\sigma \stackrel{0}{\vdash} x_1 \circ x_2 \Rightarrow b}$$

$$\frac{f\ y_1\ \ldots\ y_k\ \texttt{=}\ e \in \mathsf{P} \quad \sigma' \stackrel{\ell}{\vdash} e \Rightarrow v}{\sigma \stackrel{\ell+1}{\vdash} f\ x_1\ \ldots\ x_k \Rightarrow v} \qquad \frac{\sigma \stackrel{\ell_1}{\vdash} e_1 \Rightarrow w \quad \sigma[x \mapsto w] \stackrel{\ell_2}{\vdash} e_2 \Rightarrow v}{\sigma \stackrel{\ell_1+\ell_2}{\vdash} \texttt{let}\ x\ \texttt{=}\ e_1\ \texttt{in}\ e_2 \Rightarrow v}$$

$$\frac{x\sigma = \texttt{leaf} \quad \sigma \stackrel{\ell}{\vdash} e_1 \Rightarrow v}{\sigma \stackrel{\ell}{\vdash} \begin{array}{l}\texttt{match}\ x\ \texttt{with}\ \texttt{|}\ \texttt{leaf} \to e_1 \\ \qquad\qquad\quad \texttt{|}\ \texttt{(}x_0\texttt{,}x_1\texttt{,}x_2\texttt{)} \to e_2\end{array} \Rightarrow v} \qquad \frac{x\sigma = \texttt{false} \quad \sigma \stackrel{\ell}{\vdash} e_2 \Rightarrow v}{\sigma \stackrel{\ell}{\vdash} \texttt{if}\ x\ \texttt{then}\ e_1\ \texttt{else}\ e_2 \Rightarrow v}$$

$$\frac{x\sigma = \texttt{(}t\texttt{,}a\texttt{,}u\texttt{)} \quad \sigma' \stackrel{\ell}{\vdash} e_2 \Rightarrow v}{\sigma \stackrel{\ell}{\vdash} \begin{array}{l}\texttt{match}\ x\ \texttt{with}\ \texttt{|}\ \texttt{leaf} \to e_1 \\ \qquad\qquad\quad \texttt{|}\ \texttt{(}x_0\texttt{,}x_1\texttt{,}x_2\texttt{)} \to e_2\end{array} \Rightarrow v} \qquad \frac{x\sigma = \texttt{true} \quad \sigma \stackrel{\ell}{\vdash} e_1 \Rightarrow v}{\sigma \stackrel{\ell}{\vdash} \texttt{if}\ x\ \texttt{then}\ e_1\ \texttt{else}\ e_2 \Rightarrow v}$$

Here $\sigma[x \mapsto w]$ denotes the update of the environment σ such that $\sigma[x \mapsto w](x) = w$ and the value of all other variables remains unchanged. Furthermore, in the second `match` rule, we set $\sigma' := \sigma \uplus \{x_0 \mapsto t, x_1 \mapsto a, x_2 \mapsto u\}$ and for function application we set $\sigma' := \{y_1 \mapsto x_1\sigma, \ldots, y_k \mapsto x_k\sigma\}$.

Figure 2.1: Big-Step Semantics

S93], Nipkow [TN15], Okasaki [CO99], among others. Below, we follow Nipkow's approach, where the actual cost of splaying is measured by counting the number of calls to `splay`: $\mathsf{B} \times \mathsf{T} \to \mathsf{T}$.

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

```
splay a t = match t with
  | (cl, c, cr) → if a == c
    then (cl, c, cr)
    else if a < c
      then match cl with
        | leaf →         (leaf, c, cr)
        | (bl, b, br) → if a == b
          then (bl, a, (br, c, cr))
          else if a < b
            then match bl with
              | leaf → (leaf, b, (br, c, cr))
              | bl1 →  match splay a bl1 with
                | leaf →         leaf
                | (al, a1, ar) → (al, a1, (ar, b, (br, c, cr)))
            else match br with
              | leaf → (bl, b, (leaf, c, cr))
              | br1 →  match splay a br1 with
                | leaf →         leaf
                | (al, a1, ar) → ((bl, b, al), a1, (ar, c, cr))
      else match cr with
        | leaf →         (cl, c, leaf)
        | (bl, b, br) → if a == b
          then ((cl, c, bl), a, br)
          else if a < b
            then match bl with
              | leaf → ((cl, c, leaf), b, br)
              | bl1 →  match splay a bl1 with
                | leaf →         leaf
                | (al, a1, ar) → ((cl, c, al), a1, (ar, b, br))
            else match br with
              | leaf → ((cl, c, bl), b, leaf)
              | br1 →  match splay a br1 with
                | leaf →         leaf
                | (al, a1, ar) → (((cl, c, bl), b, al), a1, ar)
```

Figure 2.2: Function Definition `SplayTree.splay`.

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

```
insert a t = match t with
  | leaf → (leaf, a, leaf)
  | t1 →   match splay a t1 with
    | leaf →       leaf
    | (l, a1, r) → if a == a1
      then (l, a, r)
      else if a < a1
        then (l, a, (leaf, a1, r))
        else ((l, a1, leaf), a, r)
```

Figure 2.3: Function Definition `SplayTree.insert`.

```
delete a t = match t with
  | t1 → match splay a t1 with
    | leaf →       leaf
    | (l, a1, r) → if a == a1
      then match l with
        | leaf → r
        | l1 →   match splay_max l1 with
          | leaf →        leaf
          | (ll1, m, _) → (ll1, m, r)
      else (l, a1, r)
```

Figure 2.4: Function Definition `SplayTree.delete`.

```
splay_max t = match t with
    | (l, b, r) → match r with
      | leaf →        (l, b, leaf)
      | (rl, c, rr) → match rr with
        | leaf →  ((l, b, rl), c, leaf)
        | rr1 →  match splay_max rr1 with
          | leaf →        leaf
          | (rrl1, x, xa) → (((l, b, rl), c, rrl1), x, xa)
```

Figure 2.5: Function Definition `SplayTree.splay_max`.

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

CHAPTER 3

A Type System for Analysis of Logarithmic Amortized Complexity

In this chapter, we will present a type system designed for logarithmic amortised worst-case complexity. It equips the programming language defined in Section 2.2 with a standard Hindley-Milner type inference, enriched by type annotations that capture potential.

We will first introduce resource functions, that measure the potential of a tree-based data structure, and then proceed with presenting the type system in Section 3.2. To conclude the chapter, we will apply it to our running example of splay trees in Section 3.3.

3.1 Resource Functions

In this section, we detail the basic potential functions employed and clarify the definition of potentials used.

Only trees are assigned non-zero potential. This is not a severe restriction as potentials for basic datatypes would only become essential if the construction of such types would emit actual costs. This is not the case in our context. Moreover, note that lists can be conceived as trees of particular shape. The potential $\Phi(t)$ of a tree t is given as a non-negative linear combination of basic functions, which essentially amount to „sums of logs", see Schoenmakers [B S93]. It suffices to specify the basic functions for the type of trees T. As already mentioned in Section 2.1, the *rank* $\mathsf{rk}(t)$ of a tree is defined as follows

$$\begin{aligned} \mathsf{rk}(\mathtt{leaf}) &:= 1 \\ \mathsf{rk}((t, a, u)) &:= \mathsf{rk}(t) + \log'(|t|) + \log'(|u|) + \mathsf{rk}(u)\ . \end{aligned}$$

We set $\log'(n) := \log_2(\max\{n, 1\})$, that is, the (binary) logarithm function is defined for all numbers. This is merely a technicality, introduced to ease the presentation as it simplifies the statement of

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

subsequent definitions. In the following, we will denote the modified logarithmic function, simply as log. Furthermore, recall that $|t|$ denotes the number of leaves in tree t. The definition of „rank“ is inspired by the definition of potential in [B S93; T N15], but subtly changed to suit it to our context.

Definition 2. *The* basic potential functions *of* Tree*, denoted $\mathcal{BF}$, are*

- $\mathsf{rk}(t)$*, and*
- $p_{(a,b)}(t) := \log(a \cdot |t| + b)$*, where a, b are natural numbers.*

Note that the constant function 1 is representable: $1 = \log(0 \cdot |t| + 2) = p_{(0,2)}$.

Following the recipe of the high-level description in Section 2.1, potentials or more generally *resource functions* become definable as linear combinations of basic potential functions.

Definition 3. *A* resource function $r\colon [\![\mathsf{T}]\!] \to \mathbb{R}_0^+$ *is a non-negative linear combination of basic potential functions, that is,*

$$r(t) := \sum_{i \in I} q_i \cdot p_i(t) ,$$

where $q_i \in \mathbb{R}_0^+$, $p_i \in \mathcal{BF}$ and $I := \{*\} \cup (\mathbb{N} \times \mathbb{N})$. The set of resource functions is denoted $\mathcal{RF}$.

A *resource annotation over* T, or simply *annotation*, is a sequence $Q = [q_*] \cup [(q_{(a,b)})_{a,b \in \mathbb{N}}]$ with $q_*, q_{(a,b)} \in \mathbb{Q}_0^+$ with all but finitely many of the coefficients $q_*, q_{(a,b)}$ equal to 0. It represents a (finite) linear combination of basic potential functions, that is, a resource function. The empty annotation, that is, the annotation where all coefficients are set to zero, is denoted as $\varnothing$.

Remark 2. *We use the convention that the sequence elements of resource annotations are denoted by the lower-case letter of the annotation, potentially with corresponding sub- or superscripts.*

Definition 4. *The* potential *of a tree t with respect to an annotation $Q = [q_*] \cup [(q_{(a,b)})_{a,b \in \mathbb{N}}]$, is defined as follows.*

$$\Phi(t|Q) := q_* \cdot \mathsf{rk}(t) + \sum_{a,b \in \mathbb{N}} q_{(a,b)} \cdot p_{(a,b)}(t) ,$$

Recall that $p_{(a,b)} = \log(a \cdot |t| + b)$ and that rk *is the rank function, defined above.*

Example 2. *Let t be a tree, then its potential could be defined as follows:* $\mathsf{rk}(t) + 3 \cdot \log(|t|) + 1$. *With respect to the above definition this potential becomes representable by setting* $q_* := 1, q_{(1,0)} := 3, q_{(0,2)} := 1$. *Thus,* $\Phi(t|Q) = \mathsf{rk}(t) + 3 \cdot \log(|t|) + 1$. □

We emphasise that the linear combination defined above is not independent. Consider, for example $\log(2|t| + 2) = \log(|t| + 1) + 1$.

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

3.1.1 Analysis of Products of Trees

We now lift the basic potential functions $p_{(a,b)}$ of a single tree to products of trees. As discussed in Section 2.1, we define the potential functions $p_{(a_1,\dots,a_m,b)}$ for a sequence of m trees $t_1, \dots, t_m$, by setting:

$$p_{(a_1,\dots,a_m,b)}(t_1, \dots, t_m) := \log(a_1 \cdot |t_1| + \cdots + a_m \cdot |t_m| + b) \ ,$$

where $a_1, \dots, a_m, b \in \mathbb{N}$. Equipped with this definition, we generalise annotations to sequences of trees. An annotation for a sequence of length m is a sequence

$$Q = [q_1, \dots, q_m] \cup [(q_{(a_m,\dots,a_n,b)})_{a_i,b\in\mathbb{N}}] \ ,$$

again vanishing almost everywhere. Note that an annotation of length 1 is simply an annotation as defined above, where the coefficient q_1 is set to equal the coefficient q_*. Based on this, the potential of a sequence of trees $t_1, \dots, t_m$ is defined as follows:

Definition 5. *Let $t_1, \dots, t_m$ be trees and let $Q = [q_1, \dots, q_m] \cup [(q_{(a_1,\dots,a_m,b)})_{a_i,b\in\mathbb{N}}]$ be an annotation of length m as above. We define*

$$\Phi(t_1, \dots, t_m|Q) := \sum_{i=1}^{m} q_i \cdot \mathsf{rk}(t_i) + \sum_{a_1,\dots,a_m,b\in\mathbb{N}} q_{(a_1,\dots,a_m,b)} \cdot p_{(a_1,\dots,a_m,b)}(t_1, \dots, t_m) \ ,$$

where $p_{(a_1,\dots,a_m,b)}(t_1, \dots, t_m) := \log(a_1 \cdot |t_1| + \cdots + a_m \cdot |t_m| + b)$ as defined above. We use $|Q|$ to denote the length of Q.

Note that for an empty sequence of trees, we have $\Phi(\epsilon|Q) = \sum_{b\in\mathbb{N}} q_b \log(b)$. Note that the rank function $\mathsf{rk}(t)$ amounts to the sum of the logarithms of the size of subtrees of t. In particular if the tree t simplifies to a list of length n, then $\mathsf{rk}(t) = (n+1) + \sum_{i=1}^{n} \log(i)$. Moreover, as $\sum_{i=1}^{n} \log(i) \in \Theta(n \log n)$, the above defined potential functions are sufficiently rich to express linear combinations of sub- and super-linear functions.

Let σ denote a substitution, let Γ denote a typing context and let $x_1 : \mathsf{T}, \dots, x_m : \mathsf{T}$ denote all tree types in Γ. A *resource annotation for* Γ or simply *annotation* is an annotation for the sequence of trees $x_1\sigma, \dots, x_m\sigma$. We define the *potential* of $\Gamma|Q$ with respect to σ as

$$\Phi(\sigma; \Gamma|Q) := \Phi(x_1\sigma, \dots, x_m\sigma|Q) \ .$$

Definition 6. *An* annotated signature *$\overline{\mathcal{F}}$ is a mapping from functions f to sets of pairs consisting of the annotation type for the arguments of f, $\alpha_1 \times \cdots \times \alpha_n|Q$ and the annotation type $\beta|Q'$ for the result:*

$$\overline{\mathcal{F}}(f) := \left\{ \alpha_1 \times \cdots \times \alpha_n|Q \to \beta|Q' : \begin{array}{l} f \text{ has } n \text{ arguments of which } m \text{ are trees,} \\ |Q| = m \text{ and } |Q'| = 1 \end{array} \right\}$$

Note that $m \leqslant n$ by definition.

We confuse the signature and the annotated signature and denote the latter simply as $\mathcal{F}$. Because the signature is usually clear from the context (one per program), instead of $\alpha_1 \times \cdots \times \alpha_n|Q \to \beta|Q' \in \mathcal{F}(f)$, we typically write $f : \alpha_1 \times \cdots \times \alpha_n|Q \to \beta|Q'$. As our analysis makes use of a *cost-free semantics* any function symbol is possibly equipped with a *cost-free* signature, independent of $\mathcal{F}$. The cost-free signature is denoted as $\mathcal{F}^{\mathrm{cf}}$.

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

Example 3. *Consider the function* `splay`*:* $\mathsf{B} \times \mathsf{T} \to \mathsf{T}$*. The induced annotated signature is given as* $\mathsf{B} \times \mathsf{T}|Q \to \mathsf{T}|Q'$*, where* $Q := [q_*] \cup [(q_{(a,b)})_{a,b \in \mathbb{N}}]$ *and* $Q' := [q'_*] \cup [(q'_{(a,b)})_{a,b \in \mathbb{N}}]$*. The logarithmic amortised cost of splaying is then expressible through the following setting:* $q_* := 1$*,* $q_{(1,0)} = 3$*,* $q_{(0,2)} = 1$*,* $q'_* := 1$*. All other coefficients are zero.*

This amounts to a potential of the arguments $\mathsf{rk}(t) + 3\log(|t|) + 1$*, while for the result we consider only its rank, that is, the annotation expresses* $3\log(|t|) + 1$ *as the logarithmic cost of splaying. The correctness of the induced logarithmic amortised costs for the zig-zig case of splaying is verified in Section 3.3 and is also automatically verified by our prototype.* □

Suppose $\Phi(t_1, \dots, t_n, u_1, u_2|Q)$ denotes an annotated sequence of length $n+2$. Suppose further $u_1 = u_2 =: u$ and we want to *share* the value u, that is, the corresponding function arguments appears multiple times in the body of the function definition. Then we make use of the operator $\curlyvee(Q)$ that adapts the potential suitably. The operator is also called *sharing operator* in analogy to [JKM12a, Lemma 6.6].

Lemma 1. *Let* $t_1, \dots, t_n, u_1, u_2$ *denote a sequence of trees of length* $n+2$ *with annotation* Q*. Then there exists a resource annotation* $\curlyvee(Q)$ *such that* $\Phi(t_1, \dots, t_n, u_1, u_2|Q) = \Phi(t_1, \dots, t_n, u|\curlyvee(Q))$*, if* $u_1 = u_2 = u$*.*

Proof. W.l.o.g. we assume $n = 0$. Thus, let $Q = [q_1, q_2] \cup [(q_{(a_1,a_2,b)})_{a_1,a_2,b \in \mathbb{N}}]$. By definition

$$\Phi(u_1, u_2|Q) = q_1 \cdot \mathsf{rk}(u_1) + q_2 \cdot \mathsf{rk}(u_2) + \sum_{a_1,a_2,b \in \mathbb{N}} q_{(a_1,a_2,b)} \cdot p_{(a_1,a_2,b)}(u_1, u_2) ,$$

where $p_{(a_1,a_2,b)}(u_1, u_2) = \log(a_1 \cdot |u_1| + a_2 \cdot |u_2| + b)$. By assumption $u = u_1 = u_2$. Thus, we obtain

$$\begin{aligned} \Phi(u, u|Q) &= q_1 \cdot \mathsf{rk}(u) + q_2 \cdot \mathsf{rk}(u) + \sum_{a_1,a_2,b \in \mathbb{N}} q_{(a_1,a_2,b)} \cdot p_{(a_1,a_2,b)}(u, u) \\ &= (q_1 + q_2)\,\mathsf{rk}(u) + \sum_{a_1+a_2,b \in \mathbb{N}} q_{(a_1+a_2,b)} \cdot p_{(a_1+a_2,b)}(u) \\ &= \Phi(u|\curlyvee(Q)) , \end{aligned}$$

for suitable defined annotation $\curlyvee(Q)$, whose definition can be directly read off from the above constraints. □

We emphasise that the definability of the sharing annotation $\curlyvee(Q)$ is based on the fact that the basic potential functions $p_{(a_1,\dots,a_m,b)}$ have been carefully chosen so that

$$p_{(a_0,a_1,a_2,\dots,a_m,b)}(x_1, x_1, \dots, x_m) = p_{(a_0+a_1,a_2,\dots,a_m,b)}(x_1, x_2, \dots, x_m) ,$$

holds, see Section 2.1.

Remark 3. *We observe that the proof-theoretic analogue of the sharing operation constitutes in a contraction rule, if the type system is conceived as a proof system.*

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

Let $Q = [q_*] \cup [(q_{(a,b)})_{a,b \in \mathbb{N}}]$ be an annotation and let $K \in \mathbb{Q}_0^+$. Then we define $Q' := Q + K$ as follows: $Q' = [q_*] \cup [(q'_{(a,b)})_{a,b \in \mathbb{N}}]$, where $q'_{(0,2)} := q_{(0,2)} + K$ and for all $(a, b) \neq (0, 2)$ $q'_{(a,b)} := q_{(a,b)}$. By definition the annotation coefficient $q_{(0,2)}$ is the coefficient of the basic potential function $p_{(0,2)}(t) = \log(0|t| + 2) = 1$, so the annotation $Q + K$, adds cost K to the potential induced by Q. Further, we define the multiplication of an annotation Q by a constant K, denoted as $K \cdot Q$ pointwise. Moreover, let $P = [p_*] \cup [(p_{(a,b)})_{a,b \in \mathbb{N}}]$ be another annotation. Then the addition $P + Q$ of annotations P, Q is similarly defined pointwise.

3.2 A Type System for Logarithmic Amortised Resource Analysis

In this section, we present the central contribution of this thesis. We delineate a novel type-and-effect system incorporating a potential-based amortised resource analysis capable of expressing *logarithmic* amortised costs. Soundness of the approach is established in Theorem 3.

Our potential-based amortised resource analysis is couched in a type system, given in Figure 3.1. If the type judgement $\Gamma|Q \vdash e : \alpha|Q'$ is derivable, then the worst-case cost of evaluating the expression e is bound from above by the difference between the potential $\Phi(\sigma; \Gamma|Q)$ before the execution and the potential $\Phi(v|Q')$ of the value v obtained through the evaluation of the expression e. The type system makes use of a *cost-free* semantics, which does not attribute any costs to the calculation. The cost-free typing judgement is denoted as $\Gamma|Q \vdash^{\text{cf}} e : \alpha|Q'$ and based on a cost-free variant of the application rule, denoted as $(\mathsf{app} : \mathsf{cf})$. The rule $(\mathsf{app} : \mathsf{cf})$ is defined as the rule (app), however, no costs are accounted for. W.r.t. the cost-free semantics, the *empty signature*, denoted as $\alpha_1 \times \cdots \times \alpha_n|\varnothing \to \beta|\varnothing$, is always admissible. We note that the cost-free signatures form a cone, as stated in the following remark:

Remark 4. *If $\alpha_1 \times \cdots \times \alpha_n|P \to \beta|P'$ and $\alpha_1 \times \cdots \times \alpha_n|Q \to \beta|Q'$ are both cost-free signatures for a function f, then any linear combination is admissable as cost-free signature of f. I.e. we can assume $\alpha_1 \times \cdots \times \alpha_n|K \cdot P + L \cdot Q \to \beta|K \cdot P' + L \cdot Q' \in \mathcal{F}^{cf}(f)$, where $K, L \in \mathbb{Q}_0^+$.*

Remark 5. *Principally the type system can be parameterised in the resource metric (see e.g.[JKM12a]). In this thesis, we focus on amortised and worst-case runtime complexity, symbolically measured through the number of function applications. It is straightforward to generalise this type system to other monotone cost models. W.r.t. non-monotone costs, like e.g. heap usage, we expect the type system can also be readily be adapted, but this is outside the scope of the thesis.*

We consider the typing rules in turn; recall the convention that sequence elements of annotations are denoted by the lower-case letter of the annotation. Further, note that sequence elements which do not occur in any constraint are set to zero. The variable rule (var) types a variable of unspecified type α. As no actual costs are required the annotation is unchanged. Similarly no resources are lost through the use of control operators. Hence the definition of the rules (cmp) and (ite) is straightforward.

As exemplary constructor rules, we have rule (leaf) for the empty tree and rule (node) for the node constructor. Both rules define suitable constraints on the resource annotations to guarantee that the potential of the values is correctly represented.

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

The application rule (app) represents the application of a rule given in P. Each application emits actual cost 1, which is indicated in the subtraction of 1. In its simplest form, that is, for the factor $K = 0$, the rule allows to directly read off the required annotation from the set of signatures $\mathcal{F}$. For arbitrary $K \in \mathbb{Q}_0^+$, the rule allows to combine some signature with cost with a cost-free signature. We note that Remark 4 would in fact allow us to add any positive linear combination of cost-free signatures; however, for performance reasons we refrain from doing so.

In the pattern matching rule (match) the potential freed through the destruction of the tree construction is added to the annotation R, which is used in the right premise of the rule. Note that $|R| = m+2$, where m equals the number of tree types in the type context Γ.

The constraints expressed in the typing rules (let : T) and (let : gen), guarantee that the potential provided through annotation Q is distributed among the call to e_1 and e_2, that is, this rule takes care of function composition. The numbers m, k, respectively, denote the number of tree types in Γ, Δ. Due to the sharing rule — discussed in a moment — we can assume w.l.o.g. that each variable in e_1 and e_2 occurs at most once.

First, consider the rule (let : gen), that is, the expression e_1 evaluates to a value w of arbitrary type $\alpha \neq \mathsf{Tree}$. In this case the resulting value w cannot carry any potential. This is indicated through the empty annotation $\varnothing$ in the typing judgement $\Gamma|P \vdash e_1 : \alpha|\varnothing$. Similarly, in the judgement $\Delta, x:\alpha|R \vdash e_2 : \beta|Q'$ for the expression e_2, all available potential prior to the execution of e_2 stems from the potential embodied in the type context Δ w.r.t. annotation Q. This is enforced by the corresponding constraints. Suppose for $\vec{a} \neq \vec{0}$ and $\vec{b} \neq \vec{0}$, $q_{(\vec{a},\vec{b},c)}$ would be non-zero. Then the corresponding shared potential between the contexts Γ and Δ w.r.t. Q is discarded by the rule, as there is no possibility this potential is attached to the result type α.

Second, consider the more involved rule (let : T). To explain this rule, we momentarily assume that in Q no potential is shared, that is, $q_{(\vec{a},\vec{b},c)} = 0$, whenever $\vec{a} \neq \vec{0}, \vec{b} \neq \vec{0}$. In this sub-case the rule can be simplified as follows:

$$\frac{\begin{array}{lll} p_i = q_i & & r_j = q_{m+j} \\ p_{(\vec{a},c)} = q_{(\vec{a},\vec{0},c)} & r_{(\vec{b},0,c)} = q_{(\vec{0},\vec{b},c)} \quad (\vec{b} \neq \vec{0}) & r_{k+1} = p'_* \\ \Gamma|P \vdash e_1 : \mathsf{T}|P' & \Delta, x:\mathsf{T}|R \vdash e_2 : \beta|Q' & r_{(\vec{0},a,c)} = p'_{(a,c)} \end{array}}{\Gamma, \Delta|Q \vdash \mathtt{let}\ x\ \mathtt{=}\ e_1\ \mathtt{in}\ e_2 : \beta|Q'} \ (\mathsf{let} : \mathsf{T})$$

Again the potential in Γ, Δ (w.r.t. annotation Q) is distributed for the typing of the expressions e_1, e_2, respectively, which is governed by the constraints on the annotations. The simplified rule is obtained, as the assumption that no shared potential exists, makes almost all constraints vacuous. In particular, the cost-free derivation $\Gamma|P^{(\vec{b},d,e)} \vdash^{\mathrm{cf}} e_1 : \mathsf{T}|P'^{(\vec{b},d,e)}$ is not required.

Finally, consider the most involved sub-case, where shared potentials are possible. Contrary to the simplified rules discussed above, such shared potential cannot be split between the type contexts Γ and Δ, respectively. Thus, the full rule necessarily employs the *cost-free semantics*. Consequently, the premise $\Gamma|P^{(\vec{b},d,e)} \vdash^{\mathrm{cf}} e_1 : \alpha|P'^{(\vec{b},d,e)}$ expresses that for all non-zero vectors $\vec{b}$ and arbitrary indices d, e, the potentials $\Phi(\Gamma|P^{(\vec{b},d,e)})$ suffices to cover the potential $\Phi(\alpha|P'^{(\vec{b},d,e)})$, if no extra costs are

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

emitted (compare Section 2.1). Intuitively this represents that the values do not increase during the evaluation of e_1 to value w.

At last, the type system makes use of structural rules, like the *sharing* rule (share) and the weakening rules $(\mathsf{w} : \mathsf{var})$ and (w). The *sharing* rule employs the sharing operator, defined in Lemma 1. Note that the variables x, y introduced in the assumption of the typing rule are fresh variables, that do not occur in Γ. Similarly, the rule (shift) allows to shift the potential before and after evaluation of the expression e by a constant K.

Note that the weakening rules embody changes in the potential of the type context of expressions considered. This amounts to the comparison on logarithmic expressions, principally a non-trivial task that cannot be directly represented as constraints in the type system. Instead, the rule (w) employs a symbolic potential expressions for these comparisons, replacing actual values for tree by variables. Let Γ denote a type context containing the type declarations $x_1 : \mathsf{T}, \ldots, x_m : \mathsf{T}$ and let Q be an annotation of length m. Then the *symbolic potential*, denoted as $\Phi(\Gamma|Q)$, is defined as follows.

$$\Phi(x_1, \ldots, x_m|Q) := \sum_{i=1}^{m} q_i \cdot \mathsf{rk}(x_i) + \sum_{a_1,\ldots,a_m,b\in\mathbb{N}} q_{(a_1,\ldots,a_m,b)} \cdot p_{(a_1,\ldots,a_m,b)}(x_1, \ldots, x_m) \ ,$$

where $p_{(a_1,\ldots,a_m,b)}(x_1, \ldots, x_m) = \log(a_1 \cdot |x_1| + \cdots + a_m \cdot |x_m| + b)$. In order to actually solve these constraints over symbolic potentials, we have to *linearise* the underlying comparisons of logarithmic expressions. This is taken up again in Section 4.1.

Definition 7. *A program* P *is called* well-typed *if for any rule* $f(x_1, \ldots, x_k) = e \in \mathsf{P}$ *and any annotated signature* $\alpha_1 \times \cdots \times \alpha_k|Q \to \beta|Q' \in \mathcal{F}(f)$*, we have* $x_1 : \alpha_1, \ldots, x_k : \alpha_k|Q \vdash e : \beta|Q'$*. A program* P *is called* cost-free *well-typed, if the cost-free typing relation is employed.*

Before we state and prove the soundness of the presented type-and-effect system, we establish the following auxiliary result, employed in the correct assessment of the transfer of potential in the case of function composition, see Figure 3.1. See also the high-level description provided in Section 2.1.

Lemma 2. *Assume* $\sum_i q_i \log a_i \geqslant q \log b$ *for some rational numbers* $a_i, b > 0$ *and* $q_i \geqslant q$*. Then,* $\sum_i q_i \log(a_i + c) \geqslant q \log(b + c)$ *for all* $c \geqslant 1$.

Proof. W.l.o.g. we can assume $q = 1$ and $q_i \geqslant 1$, as otherwise we simply divide the assumed inequality by q. Further, observe that the assumption $\sum_i q_i \log a_i \geqslant q \log b$ is equivalent to

$$\prod_i a_i^{q_i} \geqslant b \ . \tag{3.1}$$

First, we prove that

$$(x + y)^r \geqslant x^r + y^r \quad r \geqslant 1 \quad x, y \geqslant 0 \ . \tag{3.2}$$

This is proved as follows. Fix some $x \geqslant 0$ and consider $(x + y)^r$ and $x^r + y^r$ as functions in y. It is then sufficient to observe that $(x + y)^r \geqslant x^r + y^r$ for $y = 0$ and that ${}^{d}/_{dy}(x + y)^r \geqslant {}^{d}/_{dy}(x^r + y^r)$ (the derivatives with regard to y) for all $y \geqslant 0$. Indeed, we have ${}^{d}/_{dy}(x + y)^r = r(x + y)^{r-1}$ and

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

${}^{d}/_{dy}(x^r + y^r) = ry^{r-1}$. Because of $r \geqslant 1$ and $x \geqslant 0$, we can thus deduce that ${}^{d}/_{dy}(x+y)^r \geqslant {}^{d}/_{dy}(x^r + y^r)$ for all $y \geqslant 0$.

Now we consider some $c \geqslant 1$. Combining (3.1) and (3.2), we get

$$\prod_i (a_i + c)^{q_i} \geqslant \prod_i (a_i^{q_i} + c^{q_i}) \geqslant \prod_i a_i^{q_i} + \prod_i c^{q_i} \geqslant b + c\,,$$

where we have used that $i \geq 1$, and that $q_i \geqslant 1$ and $c \geqslant 1$ imply $\prod_i c^{q_i} \geqslant c$. By taking the logarithm on both sides of the inequality we obtain the claim. □

Finally, we obtain the following soundness result, which roughly states that if a program P terminates, then the difference in potential has paid its execution costs.[1]

Theorem 3 (Soundness Theorem). *Let* P *be well-typed and let* σ *be a substitution. Suppose* $\Gamma|Q \vdash e\colon\alpha|Q'$ *and* $\sigma \overset{\ell}{\vdash} e \Rightarrow v$. *Then* $\Phi(\sigma;\Gamma|Q) - \Phi(v|Q') \geqslant \ell$. *Further, if* $\Gamma|Q \vdash^{cf} e\colon\alpha|Q'$, *then* $\Phi(\sigma;\Gamma|Q) \geqslant \Phi(v|Q')$.

Proof. The proof embodies the high-level description given in Section 2.1. It proceeds by main induction on $\Pi\colon \sigma \overset{\ell}{\vdash} e \Rightarrow v$ and by side induction on $\Xi\colon \Gamma|Q \vdash e\colon\alpha|Q'$, where the latter is employed in the context of the weakening rules. We consider only a few cases of interest. For example, for a case not covered: the variable rule (var) types a variable of unspecified type α. As no actual costs are required the annotation is unchanged and the theorem follows trivially.

Case. Π derives $\sigma \overset{0}{\vdash} \texttt{leaf} \Rightarrow \texttt{leaf}$. Then Ξ consists of a single application of the rule (leaf):

$$\frac{\forall c \geqslant 2\; q_{(c)} = \sum_{a+b=c} q'_{(a,b)} \qquad K = q'_*}{\varnothing|Q + K \vdash \texttt{leaf}\colon \mathsf{T}|Q'}\ (\mathsf{leaf})\,.$$

By assumption $Q = [(q_{(c)})_{c\in\mathbb{N}}]$ is an annotation for the empty sequence of trees. On the other hand $Q' = [(q'_{(a,b)})_{a,b\in\mathbb{N}}]$ is an annotation of length 1. Note that $\mathsf{rk}(\texttt{leaf}) = 1$ by definition. Thus, we obtain:

$$\begin{aligned}
\Phi(\epsilon|Q + K) &= K + \sum_c q_{(c)} \cdot \log(c) \\
&= K + \sum_{c\geq 2} q_{(c)} \cdot \log(c) \\
&= q'_* + \sum_{a+b\geq 2} q'_{(a,b)} \cdot \log(a+b) \\
&= q'_* + \sum_{a,b} q'_{(a,b)} \cdot \log(a+b) \\
&= q'_*\,\mathsf{rk}(\texttt{leaf}) + \sum_{a,b} q'_{(a,b)} p_{(a,b)}(\texttt{leaf}) = \Phi(\texttt{leaf}|Q')\,.
\end{aligned}$$

[1] A stated, soundness assumes termination of P, but our analysis is not restricted to terminating programs. In order to avoid the assumption the soundness theorem would have to be formulated w.r.t. to a partial big-step or a small step semantics, see [JM10a; MS20]. We consider this outside the scope of this thesis.

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

Case. Suppose Π has the following from:

$$\frac{x_1\sigma = t \quad x_2\sigma = b \quad x_3\sigma = u}{\sigma \overset{0}{\vdash} (x_1, x_2, x_3) \Rightarrow (t, b, u)} \; .$$

W.l.o.g. Ξ consists of a single application of the rule (node):

$$\frac{q_1 = q_2 = q'_* \quad q_{(1,0,0)} = q_{(0,1,0)} = q'_* \quad q_{(a,a,b)} = q'_{(a,b)}}{x_1 \colon \mathsf{T}, x_2 \colon \mathsf{B}, x_3 \colon \mathsf{T} | Q \vdash (x_1, x_2, x_3) \colon \mathsf{T} | Q'} \;(\mathsf{node})$$

By definition, we have $Q = [q_1, q_2] \cup [(q_{(a_1,a_2,b)})_{a_i,b \in \mathbb{N}}]$ and $Q' = [q'_*] \cup [(q'_{(a',b')})_{a',b' \in \mathbb{N}}]$. We set $\Gamma \coloneqq x_1 \colon \mathsf{T}, x_2 \colon \mathsf{B}, x_3 \colon \mathsf{T}$ as well as $x_1\sigma = u$, $x_2\sigma = b$, and $x_3\sigma = v$. Thus $\Phi(\sigma; \Gamma|Q) = \Phi(u, v|Q)$ and we obtain:

$$\begin{aligned}
\Phi(u, v|Q) &= q_1 \cdot \mathsf{rk}(u) + q_2 \cdot \mathsf{rk}(v) + \sum_{a_1,a_2,b} q_{(a_1,a_2,b)} \cdot \log(a_1 \cdot |u| + a_2 \cdot |v| + b) \\
&\geqslant q'_* \cdot \mathsf{rk}(u) + q'_* \cdot \mathsf{rk}(v) + q_{(1,0,0)} \cdot \log(|u|) + q_{(0,1,0)} \cdot \log(|v|) + \\
&\quad + \sum_{a,b} q_{(a,a,b)} \cdot \log(a \cdot |u| + a \cdot |v| + b) \\
&= q'_* \cdot (\mathsf{rk}(u) + \mathsf{rk}(v) + \log(|u|) + \log(|v|)) + \\
&\quad + \sum_{a,b} q'_{(a,b)} \cdot \log(a \cdot (|u| + |v|) + b) \\
&= q'_* \cdot \mathsf{rk}((u, b, v)) + \sum_{a,b} q'_{(a,b)} \cdot p_{(a,b)}((u, b, v)) = \Phi((u, b, v)|Q') \; .
\end{aligned}$$

Case. Suppose $\sigma \overset{\ell}{\vdash} e \Rightarrow v$ and let the last rule in Ξ be of the following form:

$$\frac{\Gamma|Q \vdash e \colon \alpha|Q'}{\Gamma|Q + K \vdash e \colon \alpha|Q' + K} \; ,$$

where $K \geqslant 0$. By SIH, we have that $\Phi(\sigma; \Gamma|Q) - \Phi(v|Q') \geqslant \ell$, from which we immediately obtain:

$$\Phi(\sigma; \Gamma|Q) + K - \Phi(v|Q') - K = \Phi(\sigma; \Gamma|Q) - \Phi(v|Q') \geqslant \ell \; .$$

Case. Consider the first (match) rule, where Π ends as follows:

$$\frac{x\sigma = \mathtt{leaf} \quad \sigma \overset{\ell}{\vdash} e_1 \Rightarrow v}{\sigma \overset{\ell}{\vdash} \mathtt{match}\ x\ \mathtt{with} |\ \mathtt{leaf} \rightarrow e_1 |\ (x_0, x_1, x_2) \rightarrow e_2 \Rightarrow v} \; .$$

W.l.o.g. we may assume that Ξ ends with the related application of the (match) rule:

$$\frac{\begin{array}{ll} r_{(\vec{a},a,a,b)} = q_{(\vec{a},a,b)} & r_{m+1} = r_{m+2} = q_{m+1} \\ p_{(\vec{a},c)} = \sum_{a+b=c} q_{(\vec{a},a,b)} & r_{(\vec{0},1,0,0)} = r_{(\vec{0},0,1,0)} = q_{m+1} \\ \Gamma|P + q_{m+1} \vdash e_1 \colon \alpha|Q' & \Gamma, x_1 \colon \mathsf{T}, x_2 \colon \mathsf{B}, x_3 \colon \mathsf{T}|R \vdash e_2 \colon \alpha|Q' \quad q_i = r_i = p_i \end{array}}{\Gamma, x \colon \mathsf{T}|Q \vdash \mathtt{match}\ x\ \mathtt{with}\ |\ \mathtt{leaf} \rightarrow e_1 |\ (x_1, x_2, x_3) \rightarrow e_2 \colon \alpha|Q'} \; .$$

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

Let Q be an annotation of length $m+1$ while Q' is of length 1. Thus annotations P, R have lengths m, $m+2$, respectively. We write $\vec{t} := t_1, \ldots, t_n$ for the substitution instances of the variables in Γ. Further $x\sigma = \texttt{leaf}$, where the latter equality follows from the assumption on Π. By definition and the constraints given in the rule, we obtain:

$$\begin{aligned}
\Phi(\sigma;\Gamma, x{:}\,\mathsf{T}|Q) &= \sum_i q_i \,\mathsf{rk}(t_i) + q_{m+1}\,\mathsf{rk}(\texttt{leaf}) + \sum_{\vec{a},a,c} q_{(\vec{a},a,c)} \log(\vec{a}|\vec{t}| + a|\texttt{leaf}| + c) \\
&= \sum_i q_i \,\mathsf{rk}(t_i) + q_{m+1}(\mathsf{rk}(\texttt{leaf})) + \sum_{\vec{a},a,c} q_{(\vec{a},a,c)} \log(\vec{a}|\vec{t}| + a + c) \\
&= \Phi(\sigma;\Gamma|P) + q_{m+1} \ .
\end{aligned}$$

Thus $\Phi(\sigma;\Gamma, x{:}\,\mathsf{T}|Q) = \Phi(\sigma;\Gamma|P + q_{m+1})$ and the theorem follows by an application of MIH.

Now, consider the second (match) rule, that is, Π ends as follows:

$$\frac{x\sigma = \texttt{(}t\texttt{,}a\texttt{,}u\texttt{)} \qquad \sigma' \overset{\ell}{\vdash} e_2 \Rightarrow v}{\sigma \overset{\ell}{\vdash} \texttt{match}\ x\ \texttt{with|}\ \texttt{leaf}\ \texttt{->}\ e_1 \texttt{|}\ \texttt{(}x_0\texttt{,}x_1\texttt{,}x_2\texttt{)}\ \texttt{->}\ e_2 \Rightarrow v} \ .$$

As above, we may assume that Ξ ends with the related application of the (match) rule. In this subcase, the assumption on Π yields $t := x\sigma = \texttt{(}u\texttt{,}\ b\texttt{,}\ v\texttt{)}$. By definition and the constraints given in the rule, we obtain:

$$\begin{aligned}
\Phi(\sigma;\Gamma, x{:}\,\mathsf{T}|Q) &= \sum_i q_i \,\mathsf{rk}(t_i) + q_{m+1}\,\mathsf{rk}(\texttt{(}u\texttt{,}\ b\texttt{,}\ v\texttt{)}) + \\
&\quad + \sum_{\vec{a},a,c} q_{(\vec{a},a,c)} \log(\vec{a}|\vec{t}| + a|\texttt{(}u\texttt{,}\ b\texttt{,}\ v\texttt{)}| + c) \\
&= \sum_i q_i \,\mathsf{rk}(t_i) + q_{m+1}(\mathsf{rk}(u) + \log(|u|) + \log(|v|) + \mathsf{rk}(v)) + \\
&\quad + \sum_{\vec{a},a,c} q_{(\vec{a},a,c)} \log(\vec{a}|\vec{t}| + a(|u| + |v|) + c) \\
&= \Phi(\sigma;\Gamma, x_1{:}\,\mathsf{T}, x_2{:}\,\mathsf{B}, x_3{:}\,\mathsf{T}|R) \ ,
\end{aligned}$$

where we write $\vec{a}|\vec{t}|$ as shorthand to denote componentwise multiplication.

Thus $\Phi(\sigma;\Gamma, x{:}\,\mathsf{T}|Q) = \Phi(\sigma;\Gamma, x_1{:}\,\mathsf{T}, x_2{:}\,\mathsf{B}, x_3{:}\,\mathsf{T}|R)$ and the theorem follows by an application of MIH.

Case. Consider the (let) rule, that is, Π ends in the following rule:

$$\frac{\sigma \overset{\ell_1}{\vdash} e_1 \Rightarrow w \qquad \sigma[x \mapsto w] \overset{\ell_2}{\vdash} e_2 \Rightarrow v}{\sigma \overset{\ell_1+\ell_2}{\vdash} \texttt{let}\ x\ \texttt{=}\ e_1\ \texttt{in}\ e_2 \Rightarrow v} \ ,$$

where $\ell = \ell_1 + \ell_2$. First, we consider the sub-case, where the type of e_1 is an arbitrary type α but not of type Tree. I.e. we assume that Ξ ends in the following application of the $(\mathsf{let}:\mathsf{gen})$-rule

$$\frac{\begin{array}{lll} p_i = q_i \quad p_{(\vec{a},c)} = q_{(\vec{a},\vec{0},c)} & q_{(\vec{0},\vec{b},c)} = r_{(\vec{b},c)} \ (\vec{b} \neq \vec{0}) & r_j = q_{m+j} \\ \Gamma|P \vdash e_1{:}\,\alpha|\varnothing & \Delta, x{:}\,\alpha|R \vdash e_2{:}\,\beta|Q' & \alpha \neq \mathsf{T} \end{array}}{\Gamma, \Delta|Q \vdash \texttt{let}\ x\ \texttt{=}\ e_1\ \texttt{in}\ e_2{:}\,\beta|Q'} \ (\mathsf{let}:\mathsf{gen}) \ .$$

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

Recall that $\vec{a} = a_1, \ldots, a_n$, $\vec{b} = b_1, \ldots, b_m$, $i \in \{1, \ldots, m\}$, $j \in \{1, \ldots, k\}$ and a_i, b_j, a, b, c, d, e are natural numbers. Further, the annotations Q, P, R are of length $m + k$, m and k, respectively, while the corresponding resulting annotations Q', P' and R', are of length 1.

By definition and due to the constraints expressed in the typing rule, we have:

$$
\begin{aligned}
\Phi(\sigma; \Gamma, \Delta | Q) &= \sum_i q_i \operatorname{rk}(t_i) + \sum_j q_{m+j} \operatorname{rk}(u_j) + \sum_{\vec{a}, \vec{b}, c} q_{(\vec{a}, \vec{b}, c)} \log(\vec{a}|\vec{t}| + \vec{b}|\vec{u}| + c) \\
\Phi(\sigma; \Gamma | P) &= \sum_i q_i \operatorname{rk}(t_i) + \sum_{\vec{a}, c} q_{(\vec{a}, \vec{0}, c)} \log(\vec{a}|\vec{t}| + c) \\
\Phi(w | \varnothing) &= 0 \\
\Phi(\sigma; \Delta, x\colon \alpha | R) &= \sum_j q_{m+j} \operatorname{rk}(u_j) + r_{k+1} \operatorname{rk}(w) + \sum_{\vec{b}, a, c} q_{(\vec{0}, \vec{b}, c)} \log(\vec{b}|\vec{u}| + c) \ ,
\end{aligned}
$$

where we set $\vec{t} \coloneqq t_1, \ldots, t_m$ and $\vec{u} \coloneqq u_1, \ldots, u_k$, denoting the substitution instances of the variables in Γ, Δ, respectively. Thus, we obtain

$$
\Phi(\sigma; \Gamma, \Delta | Q) \geqslant \Phi(\sigma; \Gamma | P) + \Phi(\sigma; \Delta, x\colon \alpha | R) \ .
$$

By main induction hypothesis, we conclude that

$$
\Phi(\sigma; \Gamma | P) - \Phi(w | \varnothing) \geqslant \ell_1 \text{ and } \Phi(\sigma; \Delta, x\colon \alpha | R) - \Phi(v | Q') \geqslant \ell_2 \ ,
$$

from which the sub-case follows.

Second, we consider the more involved sub-case, where e_1 is of Tree type. Thus, w.l.o.g. Ξ ends in the following application of the $(\mathsf{let} : \mathsf{T})$-rule.

$$
\frac{
\begin{gathered}
p_i = q_i \quad p_{(\vec{a}, c)} = q_{(\vec{a}, \vec{0}, c)} \quad r_j = q_{m+j} \quad r_{k+1} = p'_{*} \quad r_{(\vec{0}, d, e)} = p'_{(d,e)} \\
\forall \vec{b} \neq \vec{0} \left(r_{(\vec{b}, 0, 0)} = q_{(\vec{0}, \vec{b}, 0)} \right) \\
\forall \vec{b} \neq \vec{0}, \vec{a} \neq \vec{0} \lor c \neq 0 \left(q_{(\vec{a}, \vec{b}, c)} = \textstyle\sum_{(d,e)} p^{(\vec{b}, d, e)}_{(\vec{a}, c)} \right) \\
\forall \vec{b} \neq \vec{0}, d \neq 0 \lor e \neq 0 \left(r_{(\vec{b}, d, e)} = p'^{(\vec{b}, d, e)}_{(d,e)} \land \forall (d', e') \neq (d, e) \left(p'^{(\vec{b}, d, e)}_{(d', e')} = 0 \right) \land \right. \\
\left. \land \textstyle\sum_{(\vec{a}, c)} p^{(\vec{b}, d, e)}_{(\vec{a}, c)} \geq p'^{(\vec{b}, d, e)}_{(d,e)} \land \forall \vec{a} \neq \vec{0} \lor c \neq 0 \left(p^{(\vec{b}, d, e)}_{(\vec{a}, c)} \neq 0 \rightarrow p'^{(\vec{b}, d, e)}_{(d,e)} \leqslant p^{(\vec{b}, d, e)}_{(\vec{a}, c)} \right) \right) \\
\Gamma | P \vdash e_1 \colon \mathsf{T} | P' \qquad \forall \vec{b} \neq \vec{0}, d \neq 0 \lor e \neq 0 \left(\Gamma | P^{(\vec{b}, d, e)} \vdash^{\mathrm{cf}} e_1 \colon \mathsf{T} | P'^{(\vec{b}, d, e)} \right) \\
\Delta, x \colon \mathsf{T} | R \vdash e_2 \colon \beta | Q'
\end{gathered}
}{
\Gamma, \Delta | Q \vdash \mathtt{let}\ x\ \mathtt{=}\ e_1\ \mathtt{in}\ e_2 \colon \beta | Q'
} \ (\mathsf{let} : \mathsf{T}) \ ,
$$

where the annotations Q, P, R, Q', P' and the sequences $\vec{a}$, $\vec{b}$ are as above. Further, for each sequence $\vec{b} \neq \vec{0}$, $P^{(\vec{b}, u, v)}$ and $P'^{(\vec{b}, u, v)}$ denote annotations of length m. By definition and due to the constraints

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

expressed in the typing rule, we have for all $\vec{b} \neq \vec{0}$:

$$\begin{aligned}
\Phi(\sigma; \Gamma, \Delta | Q) &= \sum_i q_i \operatorname{rk}(t_i) + \sum_j q_j \operatorname{rk}(u_j) + \sum_{\vec{a} \neq \vec{0} \vee \vec{b} \neq \vec{0} \vee c \neq 0} q_{(\vec{a},\vec{b},c)} \log(\vec{a}|\vec{t}| + \vec{b}|\vec{u}| + c) \\
\Phi(\sigma; \Gamma | P) &= \sum_i q_i \operatorname{rk}(t_i) + \sum_{\vec{a} \neq \vec{0} \vee c \neq 0} q_{(\vec{a},\vec{0},c)} \log(\vec{a}|\vec{t}| \\
\Phi(w | P') &= r_{k+1} \operatorname{rk}(w) + \sum_{a \neq 0 \vee c \neq 0} r_{(\vec{0},a,c)} \log(a|w| + c) \\
\Phi(\sigma; \Gamma | P^{(\vec{b},d,e)}) &= \sum_{\vec{a} \neq \vec{0} \vee c \neq 0} p^{(\vec{b},d,e)}_{(\vec{a},c)} \log(\vec{a}|\vec{t}| + c) \\
\Phi(w | P'^{(\vec{b},d,e)}) &= p'^{(\vec{b},d,e)}_{(d,e)} \log(d|w| + e) \\
\Phi(\sigma; \Delta, x{:}\mathsf{T} | R) &= \sum_j q_j \operatorname{rk}(u_j) + r_{k+1} \operatorname{rk}(w) + \\
&\quad + \sum_{\vec{a} \neq \vec{0} \vee d \neq 0 \vee e \neq 0} r_{(\vec{b},d,e)} \log(\vec{b}|\vec{u}| + d|w| + e) \ ,
\end{aligned}$$

where we set $\vec{t} \coloneqq t_1, \ldots, t_m$ and $\vec{u} \coloneqq u_1, \ldots, u_k$, denoting the substitution instances of the variables in Γ, Δ, respectively.

By main induction hypothesis, we conclude that $\Phi(\sigma; \Gamma | P) - \Phi(w | P') \geqslant \ell_1$. Further, for all $\vec{b} \neq \vec{0}$, $d \neq 0 \vee e \neq 0$, we have, due to the cost-free typing constraints $\Phi(\sigma; \Gamma | P^{(\vec{b},d,e)}) \geqslant \Phi(w | P'^{(\vec{b},d,e)})$. The latter yields more succinctly (for all $\vec{b} \neq \vec{0}$, $d \neq 0 \vee e \neq 0$) that

$$\sum_{\vec{a},c} p^{(\vec{b},d,e)}_{(\vec{a},c)} \log(\vec{a}|\vec{t}| + c) \geqslant p'^{(\vec{b},d,e)}_{(d,e)} \log(d|w| + e) \ . \tag{3.3}$$

A third application of MIH yields that $\Phi(\sigma; \Delta, x{:}\mathsf{T} | R) - \Phi(v | Q') \geqslant \ell_2$. Due to the conditions $\sum_{(a,c)} p^{(\vec{b},d,e)}_{(\vec{a},c)} \geq p'^{(\vec{b},d,e)}_{(d,e)}$, for all $(d', e') \neq (d, e)$, $p'^{(\vec{b},d,e)}_{(d',e')} = 0$ and for all $\vec{a}, c$

$$\left(p^{(\vec{b},d,e)}_{(\vec{a},c)} \neq 0 \rightarrow p'^{(\vec{b},d,e)}_{(d,e)} \leqslant p^{(\vec{b},d,e)}_{(\vec{a},c)} \right) \ ,$$

we can apply Lemma 2 to Equation (3.3) and obtain

$$\sum_{\vec{a} \neq \vec{0} \vee c \neq 0} p^{(\vec{b},d,e)}_{(\vec{a},c)} \log(\vec{a}|\vec{t}| + \vec{b}|\vec{u}| + c) \geqslant p'^{(\vec{b},d,e)}_{(d,e)} \log(\vec{b}|\vec{u}| + d|w| + e) \ .$$

Due to the condition $\left(q_{(\vec{a},\vec{b},c)} = \sum_{(d,e)} p^{(\vec{b},d,e)}_{(\vec{a},c)} \right)$ for all $\vec{b} \neq \vec{0}, \vec{a} \neq \vec{0} \vee c \neq 0$, we can sum those equations for all $d \neq 0 \vee e \neq 0$ and obtain (for all $\vec{b} \neq \vec{0}$, $d \neq 0 \vee e \neq 0$) that

$$\sum_{\vec{a} \neq \vec{0} \vee c \neq 0} q_{(\vec{a},\vec{b},c)} \log(\vec{a}|\vec{t}| + \vec{b}|\vec{u}| + c) \geqslant \sum_{d \neq 0 \vee e \neq 0} r_{(\vec{b},d,e)} \log(\vec{b}|\vec{u}| + d|w| + e) \ .$$

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

We can combine the above fact to conclude the case.

Case. Finally, we consider the application rules (app) and (app : cf). As the cost-free variant only differs in sofar that costs are not counted by (app : cf), it suffices to consider the rule (app). Let $f(x_1,\ldots,x_k) = e \in \mathsf{P}$ and let Π derives $\sigma \stackrel{\ell+1}{\vdash} f(x_1,\ldots,x_k) \Rightarrow v$. We consider the costed typing $x_1\colon\alpha_1,\ldots,x_k\colon\alpha_k|(P + K\cdot Q) + 1 \vdash^{\mathrm{cf}} f(x_1,\ldots,x_k)\colon\alpha|P' - 1 + K\cdot Q'$, where $K \in \mathbb{Q}_0^+$. Set $\Gamma :=$ $x_1\colon\alpha_1,\ldots,x_k\colon\alpha_k$. As P is well-typed, we have

$$\Gamma|P \vdash e\colon\beta|P' \qquad \text{and} \qquad \Gamma|Q \vdash^{\mathrm{cf}} e\colon\beta|Q'\ .$$

We can apply MIH w.r.t. the evaluation Π' of $\sigma \stackrel{\ell}{\vdash} e \Rightarrow v$ to conclude $\Phi(\sigma;\Gamma|P) - \Phi(v|P') \geqslant \ell$ as well as $\Phi(\sigma;\Gamma|Q) \geqslant \Phi(v|Q')$. By monotonicity of addition and multiplication:

$$\begin{aligned}\Phi(\sigma;\Gamma|P + K\cdot Q) &= \Phi(\sigma;\Gamma|P) + K\cdot\Phi(\sigma;\Gamma|Q)\\ &\geqslant (\Phi(v|P') + \ell) + K\cdot\Phi(v|Q') = \Phi(v|P' + K\cdot Q') + \ell\ .\end{aligned}$$

Thus

$$\begin{aligned}&\Phi(\sigma;\Gamma|P + K\cdot Q) - \Phi(v|P' - 1 + K\cdot Q') =\\ &= (\Phi(\sigma;\Gamma|P + K\cdot Q) - \Phi(v|P' + K\cdot Q')) + 1 \geqslant \ell + 1\ .\end{aligned}$$

From this, the case follows, which completes the proof of the soundness theorem. □

Remark 6. *We note that the basic resource functions can be generalised to additionally represent linear functions in the size of the arguments. The above soundness theorem is not affected by this generalisation.*

In the next section, we exemplify the use of the proposed type-and-effect system, see Figure 3.1, on the motivating example.

3.3 Example Analysis

In this section we apply the proposed type-and-effect system to obtain an analysis of the amortised costs of the zig-zig case of *splaying*, for type annotations that are fixed a priori. As a preparatory step, also to emphasise the need for the cost-free semantics, we make precise the informal account of compositional reasoning given in Section 2.1.2.

3.3.1 Let-Normal-Form

We consider the expression $t' :=$ `(al,a,(ar,b,(br,c,cr)))`, which becomes the following expression e in let-normal form:

```
1 let t''' = (br, c, cr) in (e')
```

where e' abbreviates

```
1 let t'' = (ar, b, t''') in (al, a, t'')
```


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

The expression e is typable with the following derivation. We ignore expressions of base type to increase readability.

$$\dfrac{br\colon\mathsf{T}, cr\colon\mathsf{T}|Q_1 \vdash (br,c,cr)\colon\mathsf{T}|Q'_1 \qquad \dfrac{\dfrac{\begin{array}{l} q^3_1 = q^3_2 = q'^3_* \\ q^3_{(1,0,0)} = q^3_{(0,1,0)} = q'^3_* \\ q^3_{(1,1,0)} = q'^3_{(1,0)} \end{array}}{ar\colon\mathsf{T}, t'''\colon\mathsf{T}|Q_3 \vdash (ar,b,t''')\colon\mathsf{T}|Q'_3}\ (3.4)}{ar\colon\mathsf{T}, al\colon\mathsf{T}, t'''\colon\mathsf{T}|Q_2 \vdash e'\colon\mathsf{T}|Q'}}{br\colon\mathsf{T}, cr\colon\mathsf{T}, ar\colon\mathsf{T}, al\colon\mathsf{T}|Q \vdash e\colon\mathsf{T}|Q'}\ (*)$$

Here, we employ the derivability of the following type judgement (3.4) by a single application of (node), w.r.t. the annotation Q_4, Q' given below.

$$al\colon\mathsf{Tree}, t''\colon\mathsf{Tree}|Q_4 \vdash (al,a,t'')\colon\mathsf{Tree}|Q'\ . \tag{3.4}$$

It is not difficult to check, that the above derivation indeed proves well-typedness of the expression e w.r.t. the below given type annotations.

$$\begin{aligned}
Q\colon\ & q_1 = q_2 = q_3 = q_4 = 1; q_{(1,1,1,0,0)} = 1; q_{(1,1,0,0,0)} = 1; q_{(0,0,0,0,2)} = 1\ , \\
& q_{(1,0,0,0,0)} = q_{(0,1,0,0,0)} = q_{(0,0,1,0,0)} = q_{(0,0,0,1,0)} = 1\ , \\
Q'\colon\ & q'_* = 1, q'_{(0,2)} = 1\ , \\
Q_1\colon\ & q^1_1 = q_1 = 1; q^1_2 = q_2 = 1; q^1_{(0,0,2)} = q_{(0,0,0,0,2)} = 1\ , \\
& q^1_{(1,1,0)} = q_{(1,1,0,0,0)}; q^1_{(1,0,0)} = q_{(1,0,0,0,0)} = 1; q^1_{(0,1,0)} = q_{(0,1,0,0,0)} = 1\ , \\
Q'_1\colon\ & q'^1_* = 1; q'^1_{(1,0)} = 1; q'^1_{(0,2)} = 1\ , \\
P^{(1,0,1,0)}\colon\ & p^{(1,0,1,0)}_{(1,1,0)} = q_{(1,1,1,0,0)} = 1\ , \\
P'^{(1,0,1,0)}\colon\ & p'^{(1,0,1,0)}_{(1,0)} = 1\ , \\
Q_2\colon\ & q^2_1 = q_3 = 1; q^2_2 = q_4 = 1; q^2_3 = q'^1_* = 1;\ , \\
& q^2_{(1,0,0,0)} = q_{(0,0,1,0,0)} = 1; q^2_{(0,1,0,0)} = q_{(0,0,0,1,0)} = 1;\ , \\
& q^2_{(0,0,1,0)} = q'^1_{(1,0)} = 1; q^2_{(0,0,0,2)} = q'^1_{(0,2)} = 1; q^2_{(1,0,1,0)} = p'^{(1,0,1,0)}_{(1,0)} = 1\ , \\
Q_3\colon\ & q^3_1 = q^2_1 = 1; q^3_2 = q^2_3 = 1; q^3_{(0,0,2)} = q^2_{(0,0,0,2)} = 1 \\
& q^3_{(1,0,0)} = q^2{}_{(1,0,0,0)} = 1; q^3_{(0,1,0)} = q^2{}_{(0,0,1,0)} = 1; q^3_{(1,1,0)} = q^2{}_{(1,0,1,0)} = 1\ , \\
Q'_3\colon\ & q'^3_* = 1, q'^3_{(1,0)} = 1, q'^3_{(0,2)} = 1 \\
Q_4\colon\ & q^4_1 = q^2_2 = 1; q^4_{(0,0,2)} = q'^3_{(0,2)} = 1; q^4_{(1,0,0)} = q^2_{(0,1,0,0)} = 1; q^4_{(0,1,0)} = q'^3_{(1,0)} = 1
\end{aligned}$$

In the inference marked with $(*)$, we employ the (almost trivial) correctness of the following *cost-free* typing derivation for $br\colon\mathsf{T}, cr\colon\mathsf{T}|P^{(1,0,1,0)} \vdash^{\mathsf{cf}} (br,c,cr)\colon\mathsf{T}|P'^{(1,0,1,0)}$. (For instantiation of the rule $(\mathsf{let}:\mathsf{T})$ note $\vec{b} = (1,0)$.)

$$\dfrac{p^{(1,0,1,0)}_{(1,1,0)} = p'^{(1,0,1,0)}_{(1,0)}}{br\colon\mathsf{Tree}, cr\colon\mathsf{Tree}|P^{(1,0,1,0)} \vdash^{\mathsf{cf}} (br,c,cr)\colon\mathsf{Tree}|P'^{(1,0,1,0)}}\ . \tag{3.5}$$

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

For all $\vec{b} \neq (0)$, $\vec{b} \neq (1)$ and arbitrary d, e, we set $P^{(\vec{b},d,e)} = P'^{(\vec{b},d,e)} := \varnothing$. Our prototype fully automatically checks correctness of the above given annotations.

We emphasise that the involved (let)-rule, employed in step $(*)$ cannot be avoided. In particular, the additional cost-free derivation (3.5) is essential. Observe the annotation marked in red in the calculation above. Eventually these amount to a shared potential employed in step $(*)$. The cost-free semantics allows us to exploit this shared potential, which otherwise would have to be discarded.

To wit, assume momentarily the rule (let) would not make use of cost-free reasoning, similar to the simplified (let)-rule, that we have used in the explanations on page 26. Then the shared potential represented by the coefficient $q_{(1,1,1,0,0)} \in Q$ is discarded by the rule. However, this potential is then missing, if we attempt to type the judgement

$$ar\colon \mathsf{Tree}, al\colon \mathsf{Tree}, t'''\colon \mathsf{Tree}|R \vdash e'\colon \mathsf{Tree}|Q' ,$$

where R is defined as Q_2, except that $r_{(1,0,1,0)} = 0$. Thus, this attempt fails. (Note that the corresponding coefficient of Q_2, marked in red, is non-zero.)

Remark 7. *To some extent this is in contrast to the use of cost-free semantics in the literature [JH11; JKM12a; MG15; JAS17; MS20]. While cost-free semantics appear as an add-on in these works, essential only if we want to capture non tail-recursive programs, cost-free semantics is essential in our context — it is already required for the representation of simple values.*

3.3.2 Splay Trees

In this subsection, we exemplify the use of the type system presented in the last section on the function `splay`, see Figure 2.2. Our amortised analysis of splaying yields that the amortised cost of `splay a t` is bound by $3\log(|t|)+1$, where the actual cost counts the number of recursive calls to `splay`, see [DR85; BS93; TN15]. To verify this amortised cost, we derive

$$a\colon \mathsf{Base}, t\colon \mathsf{Tree}|Q \vdash e\colon \mathsf{Tree}|Q' , \tag{3.6}$$

where the expression e is the definition of `splay` given in Figure 2.2 and the annotations Q and Q' are as follows:

$$Q\colon q_1 = 1, q_{(1,0)} = 3, q_{(0,2)} = 1 ,$$
$$Q'\colon q'_* = 1 .$$

Remark that the amortised cost of splaying is represented by the coeficients $q_{(1,0)}$ and $q_{(0,2)}$, expressing in sum $3\log(|t|) + 1$. Note, further that the coefficient q_1, q'_*, represent Schoenmakers' potential, that is, $\mathsf{rk}(t)$ and $\mathsf{rk}($`splay a t`$)$, respectively.

We restrict to the zig-zig case which amounts to `t = ((bl,b,br),c,cr)` together with the recursive call `splay a bl = (al,a',ar)` and side conditions $a < b < c$. Thus `splay a t` yields `(al,a',(ar,b,(br,c,cr)))`. Recall that a need not occur in t, in this case, the last element a' before a leaf was found, is rotated to the root. Our prototype checks correctness of these annotations automatically.

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

Let e_1 denote the subexpression of the definition of splaying, starting in program line 4. On the other hand let e_2 denote the subexpression defined from line 5 to 15 and let e_3 denote the program code within e_2 starting in line 8. Finally the expression in lines 11 and 12, expands to e_4 as follows, if we remove part of the syntactic sugar:

```
1  let x = splay a bl in (
2    match x with
3      | leaf -> leaf
4      | (al, a', ar) -> t'
5  )
```

Below, we show a simplified derivation of (3.6), where we have focused only on a particular path in the derivation tree, suited to the considered zig-zig case of the definition of splaying. Omission of premises or rules is indicated by double lines in the inference step. Again we make crucial use of the cost-free semantics in this derivation. The corresponding inference step is marked with (∗) and the employed shared potentials are marked in red. To ease presentation we take the liberty of removing types of tree variables from the context.

We abbreviate $\Gamma := a{:}\mathsf{B}, b{:}\mathsf{B}, c{:}\mathsf{B}$, $\Delta := b{:}\mathsf{B}, c{:}\mathsf{B}$. In addition to the original signature of splaying, $\mathsf{B} \times \mathsf{T}|Q \to \mathsf{T}|Q'$, we use the following annotations, induced by constraints in the type system, see Figure 3.1. As in Section 3.3.1, we mark annotations that require cost-free derivations in the $(\mathsf{let} : \mathsf{T})$ rule in red.

$$
\begin{aligned}
Q_1 &: q^1_1 = q^1_2 = q_1 = 1, q^1_{(1,1,0)} = q_{(1,0)} = 3, q^1_{(1,0,0)} = q^1_{(0,1,0)} = q_1 = 1, q^1_{(0,0,2)} = q_{(0,2)} = 1\ ,\\
Q_2 &: q^2_1 = q^2_2 = q^2_3 = 1, q^2_{(0,0,0,2)} = 1, q^2_{(1,1,1,0)} = q^1_{(1,1,0)} = 3, q^2_{(0,1,1,0)} = q^1_{(1,0,0)} = 1,\\
&\quad q^2_{(1,0,0,0)} = q^1_{(0,1,0)} = 1, q^2_{(0,1,0,0)} = q^2_{(0,0,1,0)} = q^1_1 = 1\ ,\\
Q_3 &: q^3_1 = q^3_2 = q^3_3 = 1, q^3_{(0,0,0,2)} = 2,\\
&\quad q^3_{(0,1,0,0)} = 3, q^3_{(1,0,0,0)} = q^3_{(0,0,1,0)} = q^3_{(1,0,1,0)} = \color{red}{q^3_{(1,1,1,0)}} = 1\ .
\end{aligned}
$$

In the step marked with the rule (w) in the derivation above, a *weakening* step is applied, which amounts to the following inequality:

$$
\Phi(\Gamma, cr{:}\mathsf{T}, bl{:}\mathsf{T}, br{:}\mathsf{T}|Q_2) \geqslant \Phi(\Gamma, cr{:}\mathsf{T}, bl{:}\mathsf{T}, br{:}\mathsf{T}|Q_3)\ .
$$

We emphasise that this step can neither be avoided, nor easily moved to the axioms of the derivation. To verify the correctness of *weakening* through a direct comparison. Let σ be a substitution. Then,

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

we have

$$\begin{aligned}\Phi(\sigma; cr: \mathsf{T}, bl: \mathsf{T}, br: \mathsf{T}|Q_2) &= 1 + \mathsf{rk}(cr) + \mathsf{rk}(bl) + \mathsf{rk}(br) + 3\log(|cr| + |bl| + |br|) + \\ &\quad + \log(|bl| + |br|) + \log(|cr|) + \log(|bl|) + \log(|br|) \\ &= 1 + \mathsf{rk}(cr) + \mathsf{rk}(bl) + \mathsf{rk}(br) + 2\log(|t|) + \log(|t|) + \\ &\quad + \log(|bl| + |br|) + \log(|cr|) + \log(|bl|) + \log(|br|) \qquad (3.7) \\ &\geqslant 1 + \mathsf{rk}(cr) + \mathsf{rk}(bl) + \mathsf{rk}(br) + \log(|bl|) + \\ &\quad + \log(|br| + |cr|) + 2 + \log(|bl| + |br| + |cr|) + \\ &\quad + \log(|bl| + |br|) + \log(|cr|) + \log(|bl|) + \log(|br|) \qquad (3.8) \\ &\geqslant \mathsf{rk}(bl) + 1 + 3\log(|bl|) + \mathsf{rk}(cr) + \mathsf{rk}(br) + \log(|br|) + \\ &\quad + \log(|cr|) + \log(|br| + |cr|) + \\ &\quad + \log(|bl| + |br| + |cr|) + 1 \\ &= \Phi(\sigma; cr: \mathsf{T}, bl: \mathsf{T}, br: \mathsf{T}|Q_3)\ .\end{aligned}$$

Note that as we have $|t| = |((bl, b, br), c, cr)| = |bl| + |br| + |cr|$, we use

$$2\log(|t|) \geqslant \log(|bl|) + \log(|br| + |cr|) + 2\ ,$$

to go from (3.7) to (3.8). The general form, Lemma 4, is presented in Chapter 4. Furthermore, we have only used monotonicity of log and formal simplifications.

Further, we verify the use of the ($\mathsf{let} : \mathsf{T}$)-rule, marked with ($*$) in the proof. Consider the following annotation Q_4:

$$\begin{aligned} Q_4: \ & q^4_1 = q^3_1; q^4_2 = q^3_3; q^4_3 = q'_*; q^4_{(1,0,0,0)} = q^3_{(1,0,0,0)}; q^4_{(0,1,0,0)} = q^3_{(0,0,1,0)}; \\ & q^4_{(1,1,0,0)} = q^3_{(1,0,1,0)}; q^4_{(1,1,1,0)} = p'^{(1,1,1,0)}_{(1,0)} = 1 \\ P^{(1,1,1,0)}: \ & p^{(1,1,1,0)}_{(1,0)} = q^3_{(1,1,1,0)} = 1 \\ P'^{(1,1,1,0)}: \ & p'^{(1,1,1,0)}_{(1,0)} = 1 \end{aligned}$$

To see that Q_4 is consistent with the constraints on resource annotations in the ($\mathsf{let} : \mathsf{T}$)-rule, we first note that

$$Q + 1: \ q = q^3_2 = 1, q_{(1,0)} = q^3_{(0,1,0,0)} = 3; q_{(0,2)} = q^3_{(0,0,0,2)}\ .$$

Hence the constraints on the annotations for the left typing tree in the ($\mathsf{let} : \mathsf{T}$)-rule amount to the following:

$$q = q^3_2 = 1 \quad q_{(1,0)} = q^3_{(0,1,0,0)} = 3 \quad q_{(0,2)} = q_{(0,0,0,2)} = 2 \quad q'_* = q^4_3 = 1\ ,$$

which are fulfilled. Further, the right typing tree yields the constraints:

$$\begin{aligned} & q^4_1 = q^3_1 = 1 \quad q^4_2 = q^3_3 = 1 \quad q^4_{(1,0,0,0)} = q^3_{(1,0,0,0)} = 1 \quad q^4_{(0,1,0,0)} = q^3_{(0,0,1,0)} = 1 \\ & q^4_{(1,1,0,0)} = q^3_{(1,0,1,0)} = 1\ , \end{aligned}$$

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

which are also fulfilled. Hence, it remains to check the correctness of the constraints for the actual cost-free derivation. First, note that for the vector $\vec{b} = (1,1)$, the cost-free derivation needs to be checked w.r.t. the annotation pair $P^{(1,1,1,0)} = [p^{(1,1,1,0)}_{(1,0)}]$ and $P'^{(1,1,1,0)} = [p'^{(1,1,1,0)}_{(1,0)}]$. Second, the various constraints in the rule $(\mathsf{let}:\mathsf{T})$ simplify to the inequality $p^{(1,1,1,0)}_{(1,0)} \geqslant p'^{(1,1,1,0)}_{(1,0)}$, which holds. Third, the actual cost-free type derivation reads as follows:

$$a : \mathsf{Base}, bl : \mathsf{Tree}|P^{(1,1,1,0)} \vdash \texttt{splay a bl} : \mathsf{Tree}|P'^{(1,1,1,0)} \ . \tag{3.9}$$

The typing judgement (3.9) is derivable if the following cost-free signatures are employed for splaying:

$$\texttt{splay} : \mathsf{Tree}|P \to^{\mathrm{cf}} \mathsf{Tree}|P' \qquad \mathsf{Tree}|\varnothing \to^{\mathrm{cf}} \mathsf{Tree}|\varnothing \ ,$$

where $P = [p_{(1,0)}]$, $P' = [p'_{(1,0)}]$, with $p_{(1,0)} = p'_{(1,0)} := 1$. Recall that $\varnothing$ denotes the empty annotation, where all coefficients are set to zero. By definition, $P = P^{(1,1,1,0)}$ and $P' = P'^{(1,1,1,0)}$. Informally, this cost-free signature is admissible, as the following equality holds:

$$\Phi(\sigma; a : \mathsf{B}, bl : \mathsf{T}|P) = \log(|bl|) = \log(|(al, a', ar)|) = \Phi((al\,,\, a'\,,\, ar) : \mathsf{T}|P') \ .$$

Recall that we have $\texttt{splay a bl} = (al\,,\, a'\,,\, ar)$ for the recursive call and that $|bl| = |(al\,,\, a'\,,\, ar)|$. As depicted below, the type derivation of (3.9) proceeds similarly to the derivation above in conjunction with the analysis in Subsection 3.3.1.

As indicated the cost-free derivation also requires the use of the full version of the rule $(\mathsf{let}:\mathsf{T})$, as marked by $(*)$. In particular, the informal argument on the size of the argument and the result of splaying is built into the type system. We use the following annotations:

$$\begin{aligned}
P &: p_{(1,0)} = 1 & P' &: p'_{(1,0)} = 1 \\
P_1 &: p^1_{(1,1,0)} = p_{(1,0)} = 1 \\
P_2 &: p^2_{(1,1,1,0)} = p^1_{(1,1,0)} = 1 \\
P_3 &: p^3_{(1,1,1,0)} = p'_{(1,0)} = 1 \\
P_4 &: p^4_{(1,1,1,1,0)} = p^3_{(1,1,1,0)} = 1
\end{aligned}$$

Finally, one further application of the (match)-rule, yields the desired derivation for suitable Q_5. See also the previous subsection. Note that one further application of the *weakening* rule is required here.

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

$$\frac{\forall c \geqslant 2\; q_{(c)} = \sum_{a+b=c} q'_{(a,b)} \qquad k = q'_*}{\varnothing | Q + k \vdash \texttt{leaf} : \mathsf{T} | Q'}\ (\mathsf{leaf}) \qquad \frac{\Gamma | Q \vdash e : \alpha | Q' \quad K \geqslant 0}{\Gamma | Q + K \vdash e : \alpha | Q' + K}\ (\mathsf{shift})$$

$$\frac{\Gamma | R \vdash e : \beta | Q' \quad r_i = q_i \quad r_{(\vec{a},b)} = q_{(\vec{a},0,b)}}{\Gamma, x : \alpha | Q \vdash e : \beta | Q'}\ (\mathsf{w:var}) \qquad \frac{\Gamma, x : \alpha, y : \alpha | Q \vdash e[x,y] : \beta | Q'}{\Gamma, z : \alpha | \curlyvee(Q) \vdash e[z,z] : \beta | Q'}\ (\mathsf{share})$$

$$\frac{q_1 = q_2 = q'_* \quad q_{(1,0,0)} = q_{(0,1,0)} = q'_* \quad q_{(a,a,c)} = q'_{(a,c)}}{x_1 : \mathsf{T}, x_2 : \mathsf{B}, x_3 : \mathsf{T} | Q \vdash \texttt{(}x_1\texttt{,}x_2\texttt{,}x_3\texttt{)} : \mathsf{T} | Q'}\ (\mathsf{node})$$

$$\frac{\circ \in \{<,>,=\}}{x_1 : \alpha, x_2 : \alpha | Q \vdash x_1 \circ x_2 : \mathsf{Bool} | Q}\ (\mathsf{cmp}) \qquad \frac{\Gamma | Q \vdash e_1 : \alpha | Q' \quad \Gamma | Q \vdash e_2 : \alpha | Q'}{\Gamma, x : \mathsf{Bool} | Q \vdash \texttt{if}\ x\ \texttt{then}\ e_1\ \texttt{else}\ e_2 : \alpha | Q'}\ (\mathsf{ite})$$

$$\begin{array}{ll} r_{(\vec{a},a,a,b)} = q_{(\vec{a},a,b)} & r_{m+1} = r_{m+2} = q_{m+1} \\ p_{(\vec{a},c)} = \sum_{a+b=c} q_{(\vec{a},a,b)} & r_{(\vec{0},1,0,0)} = r_{(\vec{0},0,1,0)} = q_{m+1} \end{array}$$
$$\frac{\Gamma | P + q_{m+1} \vdash e_1 : \alpha | Q' \qquad \Gamma, x_1 : \mathsf{T}, x_2 : \mathsf{B}, x_3 : \mathsf{T} | R \vdash e_2 : \alpha | Q' \qquad q_i = r_i = p_i}{\Gamma, x : \mathsf{T} | Q \vdash \texttt{match}\ x\ \texttt{with}\ \texttt{|}\ \texttt{leaf}\ \texttt{->}\ e_1 \texttt{|}\ \texttt{(}x_1\texttt{,}x_2\texttt{,}x_3\texttt{)}\ \texttt{->}\ e_2 : \alpha | Q'}\ (\mathsf{match})$$

$$p_i = q_i \quad p_{(\vec{a},c)} = q_{(\vec{a},\vec{0},c)} \quad r_j = q_{m+j} \quad r_{k+1} = p'_* \quad r_{(\vec{0},d,e)} = p'_{(d,e)}$$
$$\forall \vec{b} \neq \vec{0} \left(r_{(\vec{b},0,0)} = q_{(\vec{0},\vec{b},0)} \right)$$
$$\forall \vec{b} \neq \vec{0}, \vec{a} \neq \vec{0} \vee c \neq 0 \left(q_{(\vec{a},\vec{b},c)} = \sum_{(d,e)} p^{(\vec{b},d,e)}_{(\vec{a},c)} \right)$$
$$\forall \vec{b} \neq \vec{0}, d \neq 0 \vee e \neq 0 \left(r_{(\vec{b},d,e)} = p'^{(\vec{b},d,e)}_{(d,e)} \wedge \forall (d',e') \neq (d,e) \left(p'^{(\vec{b},d,e)}_{(d',e')} = 0 \right) \wedge \right.$$
$$\left. \wedge \sum_{(\vec{a},c)} p^{(\vec{b},d,e)}_{(\vec{a},c)} \geqslant p'^{(\vec{b},d,e)}_{(d,e)} \wedge \forall \vec{a} \neq \vec{0} \vee c \neq 0 \left(p^{(\vec{b},d,e)}_{(\vec{a},c)} \neq 0 \rightarrow p'^{(\vec{b},d,e)}_{(d,e)} \leqslant p^{(\vec{b},d,e)}_{(\vec{a},c)} \right) \right)$$
$$\frac{\Gamma | P \vdash e_1 : \mathsf{T} | P' \qquad \forall \vec{b} \neq \vec{0}, d \neq 0 \vee e \neq 0 \left(\Gamma | P^{(\vec{b},d,e)} \vdash^{\mathrm{cf}} e_1 : \mathsf{T} | P'^{(\vec{b},d,e)} \right) \qquad \Delta, x : \mathsf{T} | R \vdash e_2 : \beta | Q'}{\Gamma, \Delta | Q \vdash \texttt{let}\ x\ \texttt{=}\ e_1\ \texttt{in}\ e_2 : \beta | Q'}\ (\mathsf{let:T})$$

$$\frac{\begin{array}{lll} p_i = q_i \quad p_{(\vec{a},c)} = q_{(\vec{a},\vec{0},c)} & q_{(\vec{0},\vec{b},c)} = r_{(\vec{b},c)} \ (\vec{b} \neq \vec{0}) & r_j = q_{m+j} \\ \Gamma | P \vdash e_1 : \alpha | \varnothing & \Delta, x : \alpha | R \vdash e_2 : \beta | Q' & \alpha \neq \mathsf{T} \end{array}}{\Gamma, \Delta | Q \vdash \texttt{let}\ x\ \texttt{=}\ e_1\ \texttt{in}\ e_2 : \beta | Q'}\ (\mathsf{let:gen})$$

$$\frac{\Gamma | P \vdash e : \alpha | P' \quad \Phi(\Gamma | P) \leqslant \Phi(\Gamma | Q) \quad \Phi(\Gamma | P') \geqslant \Phi(\Gamma | Q')}{\Gamma | Q \vdash e : \alpha | Q'}\ (\mathsf{w}) \qquad \frac{x \text{ a variable}}{x : \alpha | Q \vdash x : \alpha | Q}\ (\mathsf{var})$$

$$\frac{\alpha_1 \times \cdots \times \alpha_n | P \rightarrow \beta | P' \in \mathcal{F}(f) \quad \alpha_1 \times \cdots \times \alpha_n | Q \rightarrow \beta | Q' \in \mathcal{F}^{\mathrm{cf}}(f) \quad K \in \mathbb{Q}^+_0}{x_1 : \alpha_1, \ldots, x_n : \alpha_n | P + K \cdot Q \vdash f(x_1, \ldots, x_n) : \beta | (P' + K \cdot Q') - 1}\ (\mathsf{app})$$

To ease notation, we set $\vec{a} := a_1, \ldots, a_m$, $\vec{b} := b_1, \ldots, b_k$ for vectors of indices $a_i, b_j \in \mathbb{N}$. Further, $i \in \{1, \ldots, m\}$, $j \in \{1, \ldots, k\}$ and $a, b, c, d, e \in \mathbb{N}$. Sequence elements of annotations, which are not constrained, are set to zero.

Figure 3.1: Type System for Logarithmic Amortised Resource Analysis

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

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\texttt{splay}\colon \mathsf{T}|Q \to \mathsf{T}|Q'}{a\colon \mathsf{B}, bl|Q \vdash \texttt{splay a bl}\colon \mathsf{T}|Q'-1} \quad \dfrac{\Delta, cr, br, al, a'\colon \mathsf{B}, ar|Q_5 \vdash t'\colon \mathsf{T}|Q'}{\Delta, cr, br, x|Q_4 \vdash \texttt{match}\ x\ \texttt{with}\ |\,(al, a', ar)\ \to t'\colon \mathsf{T}|Q'}}{\Gamma, cr, bl, br|Q_3 \vdash e_4\colon \mathsf{T}|Q'}\ (*)}{\Gamma, cr, bl, br|Q_3 \vdash e_3\colon \mathsf{T}|Q'}}{\Gamma, cr, bl, br|Q_2 \vdash e_3\colon \mathsf{T}|Q'}\ (\mathsf{w})}{a\colon \mathsf{B}, b\colon \mathsf{B}, cl, cr|Q_1 \vdash \texttt{match}\ cl\ \texttt{with}\ |\,\texttt{leaf} \to (cl, c, cr)\,|\,(bl, b, br) \to e_3\colon \mathsf{T}|Q'}}{a\colon \mathsf{B}, cl, c\colon \mathsf{B}, cr|Q_1 \vdash \texttt{if}\ a = c\ \texttt{then}\ (cl, c, cr)\ \texttt{else}\ e_2\colon \mathsf{T}|Q'}}{a\colon \mathsf{B}, t|Q \vdash \texttt{match}\ t\ \texttt{with}\,|\,\texttt{leaf} \to \texttt{leaf}\,|\,(cl, c, cr) \to e_1\colon \mathsf{T}|Q'}$$

Figure 3.2: Partial Typing Derivation for `splay`, focusing on the zig-zig Case.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\texttt{splay}\colon \mathsf{T}|\varnothing \to \mathsf{T}|\varnothing}{a\colon \mathsf{B}, bl|\varnothing \vdash \texttt{splay a bl}\colon \mathsf{T}|\varnothing} \quad \dfrac{\Delta, cr, br, al, a'\colon \mathsf{B}, ar|P_4 \vdash t'\colon \mathsf{T}|P'}{\Delta, cr, br, x|P_3 \vdash \texttt{match}\ x\ \texttt{with}\ |\,(al, a', ar)\ \to t'\colon \mathsf{T}|P'}}{\Gamma, cr, bl, br|P_2 \vdash e_4\colon \mathsf{T}|P'}\ (*)}{\Gamma, cr, bl, br|P_2 \vdash e_3\colon \mathsf{T}|P'}}{a\colon \mathsf{B}, b\colon \mathsf{B}, cl, cr|P_1 \vdash \texttt{match}\ cl\ \texttt{with}\ |\,\texttt{leaf} \to (cl, c, cr)\,|\,(bl, b, br) \to e_3\colon \mathsf{T}|P'}}{a\colon \mathsf{B}, cl, c\colon \mathsf{B}, cr|P_1 \vdash \texttt{if}\ a = c\ \texttt{then}\ (cl, c, cr)\ \texttt{else}\ e_2\colon \mathsf{T}|P'}}{a\colon \mathsf{B}, t|P \vdash \texttt{match}\ t\ \texttt{with}\,|\,\texttt{leaf} \to \texttt{leaf}\,|\,(cl, c, cr) \to e_1\colon \mathsf{T}|P'}$$

Figure 3.3: Cost-Free Derivation for `splay`, focusing on the zig-zig Case.

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

CHAPTER 4

Automation

The type system presented in Chapter 3 is a sound, theoretical basis. In Section 2.1.1 we mentioned how inferring coefficients allows analysis. However, development of a tool that performs fully automated analysis, comes with additional challenges.

In this chapter, we want to highlight the challenges and how they were met.

4.1 Linearisation and Expert Knowledge

In the context of the presented type system (see Figure 3.1) an obvious challenge is the requirement to compare potentials symbolically (see Section 3.2) rather than to compare annotations directly. This is in contrast to results on resource analysis for constant amortised costs, see e.g. [S J+09; S J+10; JKM12a; JAS17; S J+17].

Comparison between logarithmic expressions, constitutes a first major challenge, as such a comparison cannot be directly encoded as a *linear* constraint problem.

To achieve such *linearisation*, we make use of the following:

(i) a subtle and surprisingly effective variant of Schoenmakers' potential (not covered below, refer to Section 3.1);

(ii) mathematical facts about the logarithm function — like Lemma 4 below — referred to as *expert knowledge*; and finally

(iii) Farkas' Lemma (Lemma 5) for turning the universally-quantified premise of the weakening rule into an existentially-quantified statement that can be added to the constraint system.

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

Furthermore, the presence of logarithmic basic functions seems to necessitate the embodiment of non-linear arithmetic. In particular, as we need to make use of basic laws of the log functions, as the following one. A variant of the below fact has already been observed by Okasaki, see [C O99].

Lemma 4 (`l2xy`)**.** *Let* $x, y \geqslant 1$*. Then* $2 + \log(x) + \log(y) \leqslant 2\log(x+y)$*.*

Proof. We observe

$$(x+y)^2 - 4xy = (x-y)^2 \geqslant 0\ .$$

Hence $(x+y)^2 \geqslant 4xy$ and from the monotonicity of log we conclude $\log(xy) \leqslant \log(\frac{(x+y)^2}{4})$. By elementary laws of log we obtain:

$$\log\left(\frac{(x+y)^2}{4}\right) = \log\left(\left(\frac{x+y}{2}\right)^2\right) = 2\log(x+y) - 2\ ,$$

from which the lemma follows as $\log(xy) = \log(x) + \log(y)$. □

We remark that our automated analysis shows that Lemma 4 is not only crucial in the analysis of splaying, but also for the other data structures we have investigated.

A refined and efficient approach which targets linear constraints is achievable as follows. All logarithmic terms, that is, terms of the form $\log(.)$ are replaced by new variables, focusing on finitely many. For the latter, we exploit the condition that in resource annotations only finitely many coefficients are non-zero. Consider the following inequality as prototypical example. Validity of the constraint ought to incorporate the monotonicity of log.

$$a_1 \log(|t|) + a_2 \log(|cr|) \geqslant b_1 \log(|t|) + b_2 \log(|cr|)\ , \tag{4.1}$$

where we assume $t = (cl\,,\ c\,,\ cr)$ for some value c and thus $|t| \geqslant |cr|$, see Section 3.3.2. Replacing $\log(|t|)$, $\log(|cr|)$ with new unknowns x, y, respectively, we represent (4.1) as follows:

$$\forall x, y \geqslant 0.\ a_1 x + a_2 y \geqslant b_1 x + b_2 y\ , \tag{4.2}$$

Here we keep the side-condition $x \geqslant y$ and observe that the unknowns x, y can be assumed to be non-negative, as they represent values of the log function. Thus properties like e.g. monotonicity of log, as well as properties like Lemma 4 above, can be expressed as inequalities over the introduced unknowns. E.g., the inequality $x \geqslant y$ above represents the axiom of monotonicity $\log(|t|) \geqslant \log(|cr|)$. All such obtained inequalities are collected as *expert or prior knowledge*. This entails that (4.2) is equivalent to the following existential constraint satisfaction problem:

$$\exists c, d.\ a_1 \geqslant b_1 + c \wedge a_2 \geqslant d \wedge b_2 \leqslant c + d\ . \tag{4.3}$$

We seek to systematise the derivation of inequalities such as (4.3) from expert knowledge. For that, we assume that the gathered prior knowledge is represented by a system of inequalities as $A\vec{x} \leqslant \vec{b}, \vec{x} \geqslant 0$, where A denotes a matrix with as many rows as we have prior knowledge, $\vec{b}$ a column vector and $\vec{x}$ the column vector of unknowns of suitable length; $\vec{x} \geqslant 0$ because log evaluates to non-negative values.

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

Below we discuss a general method for the derivation of inequalities such as (4.3) based on the affine form of Farkas' Lemma. First, we state the variant of Farkas' Lemma that we use in this article, see [Sch99]. Note that $\vec{u}$ and $\vec{f}$ denote column vectors of suitable length.

Lemma 5 (Farkas' Lemma)**.** *Suppose $A\vec{x} \leqslant \vec{b}, \vec{x} \geqslant 0$ is solvable. Then for all vectors $\vec{u}$ and scalars λ, the following assertions are equivalent.*

$$\forall \vec{x} \geqslant 0.\, A\vec{x} \leqslant \vec{b} \Rightarrow \vec{u}^T \vec{x} \leqslant \lambda \tag{4.4}$$

$$\exists \vec{f} \geqslant 0.\, \vec{u}^T \leqslant \vec{f}^T A \wedge \vec{f}^T \vec{b} \leqslant \lambda \tag{4.5}$$

Proof. It is easy to see that from (4.5), we obtain (4.4). Assume (4.5). Assume further that $A\vec{x} \leqslant \vec{b}$ for some column vector $\vec{x}$. Then we have

$$\vec{u}^T \vec{x} \leqslant \vec{f}^T A\vec{x} \leqslant \vec{f}^T \vec{b} \leqslant \lambda \ .$$

Note that for this direction the assumption that $A\vec{x} \leqslant \vec{b}, \vec{x} \geqslant 0$ is solvable is not required.

With respect to the opposite direction, we assume (4.4). By assumption, $A\vec{x} \leqslant \vec{b}, \vec{x} \geqslant 0$ is solvable. Hence, maximisation of $\vec{u}^T \vec{x}$ under the side condition $A\vec{x} \leqslant \vec{b}, \vec{x} \geqslant 0$ is feasible. Let w denote the maximal value. Due to (4.4), we have $w \leqslant \lambda$.

Now, consider the dual asymmetric linear program to minimise $\vec{y}^T \vec{b}$ under side condition $\vec{y}^T A = \vec{u}^T$ and $\vec{y} \geqslant 0$. Due to the Dualisation Theorem, the dual problem is also solvable with the same solution

$$\vec{y}^T \vec{b} = \vec{u}^T \vec{x} = w \ .$$

We define $\vec{f} := \vec{y}$, which attains the optimal value w, such that $\vec{f}^T A = \vec{u}^T$ and $\vec{f} \geqslant 0$ such that $\vec{f}^T \vec{b} = w \leqslant \lambda$. This yields (4.5). □

Second, we discuss a method for the derivation of inequalities such as (4.3) based on Farkas' Lemma. Our goal is to automatically discharge symbolic constraints such as $\Phi(\Gamma|P) \leqslant \Phi(\Gamma|Q)$ — as well as $\Phi(\Gamma|P') \geqslant \Phi(\Gamma|Q')$ — as required by the *weakening* rule (w) (see Section 3.2).

According to the above discussion we can represent the inequality $\Phi(\Gamma|P) \leqslant \Phi(\Gamma|Q)$ by

$$\vec{p}^T \vec{x} + c_p \leqslant \vec{q}^T \vec{x} + c_q \ ,$$

where $\vec{x}$ is a finite vector of variables representing the base potential functions, $\vec{p}$ and $\vec{q}$ are column vectors representing the unknown coefficients of the non-constant potential functions, and c_p and c_q are the coefficients of the constant potential functions. We assume the expert knowledge is given by the constraints $A\vec{x} \leqslant \vec{b}, \vec{x} \geqslant 0$. We now want to derive conditions for $\vec{p}$, $\vec{q}$, c_p, and c_q such that we can guarantee

$$\forall \vec{x} \geqslant 0.\, A\vec{x} \leqslant \vec{b} \Rightarrow \vec{p}^T \vec{x} + c_p \leqslant \vec{q}^T \vec{x} + c_q \ . \tag{4.6}$$

By Farkas' Lemma it is sufficient to find coefficients $\vec{f} \geqslant 0$ such that

$$\vec{p}^T \leqslant \vec{f}^T A + \vec{q}^T \wedge \vec{f}^T \vec{b} + c_p \leqslant c_q \ . \tag{4.7}$$

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

Hence, we can ensure (4.6) by (4.7) using the new unknowns $\vec{f}$.

We illustrate Equation (4.7) on the above example. We have $A = (-1\ 1)$, $b = 0$, $\vec{p} = (b_1\ b_2)^T$, $\vec{q} = (a_1\ a_2)^T$ as well as $c_p = c_q = 0$. Then, the inequality $fb + c_p \leqslant c_q$ simplifies to $0 \leqslant 0$ and can in the following be omitted. With the new unknown $f \geqslant 0$ we have

$$(b_1\ b_2) \leqslant f(-1\ 1) + (a_1\ a_2)\ ,$$

which we can rewrite to

$$b_1 \leqslant -f + a_1 \wedge b_2 \leqslant f + a_2\ ,$$

easily seen to be equivalent to Equation (4.3).

Thus, the validity of constraints incorporating the monotonicity of log becomes expressible in a systematic way. Further, the symbolic constraints enforced by the *weakening* rule can be discharged systematically and become expressible as existential constraint satisfaction problems. Note that the incorporation of Farkas' Lemma in the above form subsumes the well-known practice of coefficient comparison for the synthesis of polynomial interpretations [E C+05], ranking functions [AA04] or resource annotations in the context of constant amortised costs [JKM12a].

The incorporation of Farkas' Lemma with suitable expert knowledge is already essential for *type checking*, whenever (w) needs to be discharged.

ATLAS incorporates two facts into the expert knowledge: Lemma 5 and the monotonicity of the logarithm (see Chapter 5). We found these two facts to be sufficient for handling our benchmarks, i.e. expert knowledge of form (ii) and (iii) was not needed. (We note though that we have played with adding a dedicated size analysis (ii), which interestingly increased the solver performance, despite generating a large constraint system).

Further, the following variant of Farkas' Lemma, lies at the heart of an effective transformation of comparisons demanded by (w) into a linear constraint problem.

The lemma allows the assumption of *expert knowledge* through the assumption $A\vec{x} \leqslant \vec{b}$ for all $\vec{x} \geqslant 0$. The, thus formalised expert knowledge is a clear point of departure for additional information.

Further, we emphasise that Lemma 5 subsumes the well-known practise of coefficient comparison for the synthesis of polynomial interpretations [E C+05], ranking functions [AA04] or resource annotations in the context of constant amortised costs [JKM12a].

We indicate how ATLAS may be used to solve the constraints generated for the example in Section 2.1. We recall the crucial application of the *weakening* step between annotations Q_2 and Q_3. This weakening step can be automatically discharged using the monotonicity of logs and Lemma 4. (More precisely, ATLAS employs the mode `w{mono l2xy}` see, Chapter 5.) For example, ATLAS is able to verify the validity of the following concrete constants:

$$\begin{aligned}
Q_2\colon\ & q^2_1 = q^2_2 = q^2_3 = 1; q^2_{(1,0,0,0)} = 1; q^2_{(0,1,0,0)} = 1; \\
& q^2_{(1,1,1,0)} = 3, q^2_{(0,1,1,0)} = 1; q^2_{(0,0,0,2)} = 1; \\
Q_3\colon\ & q^3_1 = q^3_2 = q^3_3 = 1; q^3_{(0,1,0,0)} = 3; q^3_{(1,0,0,0)} = q^3_{(0,0,1,0)} = 1; \\
& q^3_{(1,0,1,0)} = q^3_{(1,1,1,0)} = 1; q^3_{(0,0,0,2)} = 2\ .
\end{aligned}$$

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

In the next section, we briefly detail our implementation of the established logarithmic amortised resource analysis, based on the observations in this section.

4.2 Type Inference

Second, we reckon to what these ideas are sufficient for type checking and detail further challenges to fully automated type inference and the main design choice in ATLAS, to overcome these challenges. Finally, we indicate how our tool solves the gathered constraints induced by the type system for the motivating example and we remark on challenges posed by our benchmarking code base for *splay heaps* and *pairing heaps*.

Further, we automate the application of structural rules like *sharing* or *weakening* We emphasise that it is not sufficient to include all weakening steps into the axioms of the typing rules. This is in contrast to the situation of earlier work by Hofmann et al.,e.g. [MS03; JKM11; JKM12b; HR13; JAS17; S J+17], which could rely on so-called algorithmic typing rules.

Concretely, they came about by a novel

(i) *optimisation layer*;

(ii) a careful control of the *structural rules*;

(iii) the generalisation of user-defined *proof tactics* into an overall strategy of type inference; and

(iv) provision of an automated amortised analysis in the sense of Sleator and Tarjan.

In the sequel of the section, we will discuss these stepping stones towards full automation in more details.

4.2.1 Optimisation Layer

We add an optimisation layer to the setup, in order to support *type inference*. This allows for the inference of (optimal) type annotations based on user-defined type annotations. For example, assume the user-provided type annotation $\mathsf{rk}(t) + 3\log(|t|) + 1 \to \mathsf{rk}(\mathtt{splay}(t))$ can in principle be checked automatically. Then — instead of checking this annotation — ATLAS automatically *optimises* the signature, by minimising the deduced coefficients. (In Chapter 5 we discuss how this optimisation step is performed.) That is, ATLAS reports the following annotation

$$\mathtt{splay}\colon {}^{1}\!/_{2}\,\mathsf{rk}(t) + {}^{3}\!/_{2}\log(|t|) \to {}^{1}\!/_{2}\,\mathsf{rk}(\mathtt{splay}(t)) \ ,$$

which yields the *optimal* amortised cost of splaying of ${}^{3}\!/_{2}\log(|t|)$. Optimality here means that no better bound has been obtained by earlier pen-and-paper verification methods (compare the discussion in Chapter 1).

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

```
1 (match (* t *) leaf
2   (match (* cl *) ?
3     (w{l2xy} (let:tree:cf (* s *)
4       app (* splay_eq a bl *)
5       (match leaf
6         (let:tree:cf node (let:tree:cf node (w{mono} node))))))))
```

Figure 4.1: Tactic that matches the zig-zig case of `splay` as shown in Fig. 2.2.

4.2.2 Structural Rules

We observed that an unchecked application of the structural rules, that is of the *sharing* and the *weakening* rule, quickly leads to an explosion of the size of the constraint system and thus to de-facto unsolvable problems. To wit, an earlier version of our implementation ran continuously for *24/7* without being able to infer a type for the complete definition of the function `splay`.

The type-and-effect system proposed by Hofmann et al. is in principle *linear*, that is, variables occur at most once in the function body. For example, this is employed in the definition of the (let)-rule, see Section 2.1. However, a *sharing* rule is admissible, that allows to treat multiple occurrences of variables. Occurrences of non-linear variables are suitably renamed apart and the carried potential is shared among the variants. The number of variables strongly influences the size of the constraint problem. Hence, eager application of the sharing rule proved infeasible. Instead, we restricted its application to individual program traces. For the considered benchmark examples, this removed the need for sharing altogether.

With respect to *weakening*, however, a refinement of the employed weakening steps proved essential. I.e. we make use of different levels of *granularity* of this automation, ranging from a simple coefficient comparison (indicated in the tactics as `w`) to full endowment of the methodology discussed above (`w{mono l2xy}`); see the detailed discussion in Chapter 5.

The structural rules can in principle be applied at every AST node of the program under analysis. However, they introduce additional variables and constraints and for performance reasons it is better to apply them sparingly. For the *sharing* rule we proceed as follows: We recall that the sharing rule allows us to assume that the type system is linear. In particular, we can assume that every variable occurs exactly once in the type context, which is exploited in the definition of the *let* rules. However, such an eager application of the sharing rule would directly yield to a size explosion in the number of constraints, as the generation of each fresh variables requires the generation of exponentially many annotations. Hence, we only apply sharing only when strictly necessary. In this way the typing context can be kept small. Similar to the sharing rule (share), variable weakening (w : var) is employed only when required. This in turn reduces the number of constraints generated. For the *weakening* rule, we employ our novel methods for symbolically comparing logarithmic expressions, which we discussed in Section 4.1. Because of our use of Farkas' Lemma, weakening introduces new unknown coefficients. For performance reasons, we need to control the size of the resulting constraint system and rely on the user to insert applications of the weakening rule. We note that the weakening rule may need to be applied in the middle of a type derivation, see for example the typing derivation for our motivating example in Figure 3.2. This contrasts to the literature where the weakening rule can typically be incorporated

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

into the axioms of the type system and thus dispensed with. Perhaps a similar approach is possible in the context of logarithmic amortised resource analysis. For now we have not been able to verify this.

4.2.3 Proof Tactics

In combination with our optimisation framework, tactics allow to signifcantly improve type annotations. To wit, ATLAS can be invoked with user-defined resource annotations for the function `splay`, representing its „standard" amortised complexity (e.g. copied from Okasaki's book [C O99]) and an easily definable tactic. For example, employing the tactic for the zig-zig case depicted in Figure 4.1. Then, ATLAS automatically derives the improved bound reported above. Still, for full automation, tactics are clearly not sufficient. In order to obtain *type inference* in general, we developed a generalisation of all the tactics that proved useful on our benchmark and incorporated this proof search strategy into the type inference algorithm. Using this, the aforementioned (unsuccessful) week-long quest for a type inference of splaying can now be successfully answered (with the best known result) in minutes.

4.2.4 Automated Amortised Analysis

In Section 2.1, we provided a high-level introduction into the potential method and remarked that Sleator and Tarjan's original formulation is re-obtained, if the corresponding potential functions are defined such that $\phi(v) := a_f(v) + \psi(x)$, see page 11. Formally this can be achieved by careful control of the annotated signatures of the functions studied. Suppose, we are interested in an amortised analysis of pairing heaps — in the original sense of Sleator and Tarjan. For that, it suffices to control the annotated type of the result the functions defined over pairing heaps, that is, we add the additional constraint that the type annotations for the results are equal. We now discuss how we can extract amortised complexities in the sense of Sleator and Tarjan from our approach. Suppose, we are interested in an amortised analysis of splay heaps. Then, it suffices to equate the right-hand sides of the annotated signatures of the splay heap functions. That is, we set `del_min`: $\mathsf{B} \times \mathsf{T}|Q_1 \to \mathsf{T}|Q'$, `insert`: $\mathsf{B} \times \mathsf{T}|Q_2 \to \mathsf{T}|Q'$ and `partition`: $\mathsf{T}|Q_3 \to \mathsf{T}|Q'$ for some unknown resource annotations Q_1, Q_2, Q_3, Q'. Note that we use the same annotation Q' for all signatures. We can then obtain a potential function from the annotation Q' in the sense of Sleator and Tarjan and deduce $Q_i - Q'$ as an upper bound on the amortised complexity of the respective function. In Chapter 5, we discuss how to automatically optimise $Q_i - Q'$ in order to minimise the amortised complexity bound. Thus, we can (by soundness) derive the amortised cost for *pairing heaps* by utilising the above definition. This automatic minimisation is the second major contribution of our work. Our results suggest a new approach for the complexity analysis of data structures. On the one hand, we obtain novel insights into the automated worst-case runtime complexity analysis of involved programs. On the other hand, we provide a proof-of-concept of a computer-aided analysis of amortised complexities of data-structures that so far have only be analysed manually. For example, our approach allows the automated verification of certificates in program code, stating the (expected) amortised complexity. Most often these comments only refer to the expectation of the programmer, but have not been verified in any way, let alone in a formally verifiable one.

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

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

CHAPTER 5

Implementation

Based on the principal approach, delineated in Section 2.1, we have implemented the logarithmic amortised resource analysis detailed above. In this chapter, we briefly indicate the corresponding design choices and heuristics used.

Our tool ATLAS implements the type system partly presented in Figure 3.1. Its core is the syntax-directed application of typing rules.

It operates in three phases:

1.) *Preprocessing*, ATLAS parses and normalises the input program;

2.) *Generation of the Constraint System*, ATLAS extracts constraints from the normalised program according to the typing rules (see Figure 3.1); and

3.) *Solving*, the derived constraint system is handed to an optimising constraint solver and the solver output is converted into a type annotation.

In terms of overall resource requirements, the bottleneck of the system is phase three. Preprocessing is both simple and fast. While the code implementing constraint generation might be complex, its execution is fast. All of the underlying complexity is shifted into the third phase. On modern machines with multiple gibibytes of main memory, ATLAS is constrained by the CPU, and not by the available memory.

In the remainder of this section, we first detail these phases of ATLAS. We then go into more details of the second phase. Finally, we elaborate the optimisation function which is the key enabler of type inference.

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

5.1 The Three Phases of ATLAS

5.1.1 Preprocessing

The parser used in the first phase is generated with ANTLR[1] and transformation of the syntax is implemented in Java. The preprocessing performs two tasks:

(i) Transformation of the input program into *let-normal-form*, which is the form of program input required by our type system.

(ii) The *unsharing* conversion creates explicit copies for variables that are used multiple times. Making multiple uses of a variables explicit is required by the (let)-rule of the type system.

In order to satisfy the requirement of the (let)-rule, it is actually sufficient to track variable usage on the level of program paths. It turns out that in our benchmarks variables are only used multiple times in different branches of an if-statement, for which no unsharing conversion is needed. Hence, we do not discuss the unsharing conversion further in this thesis and refer the interested reader to related approaches [J H11; JKM12b; JKM12a; JAS17] for more details.

Let-Normal-Form Conversion

Since programs are not usually written in the restricted syntax demanded by the type system, the input program is first converted to let-normal-form. The conversion is performed recursively and rewrites composed expressions into simple expressions, where each operator is only applied to a variable or a constant. This conversion is achieved by introducing additional let-constructs. We exemplify let-normal-form conversion on a code snippet in Figure 5.1.

Unsharing

The unsharing operation introduces an explicit *share* node before a let-expression whenever its subexpressions have a shared variable, as shown in Figure 5.2. It introduces fresh variables and renames occurrences of the shared variable. Note that this is a departure from the type system and makes the application of the (share) rule syntax-directed. We give an example for unsharing in Figure 5.2. We note that unsharing is only required when variables can be used multiple times on the same program path However, in none of our benchmarks variables are shared, so this step is not of relevance for the presented results.

5.1.2 Generation of the Constraint System

After preprocessing, we apply the typing rules. Each rule application generates a set of constraints, which are collected over multiple passes over the syntax tree. Importantly, the application of all typing rules, except for the weakening rule, which we discuss in further detail below, is *syntax-directed*: This means that each node of the AST of the input program dictates which typing rule is to be applied.

[1] See antlr.org.

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

```
1  LNF[if a < a'
2   then (l,a,(leaf,a',r))
3   else ((l,a',leaf),a,r)]
```

(a) An if-then-else expression before translation to let-normal-form.

```
1  let x1 = a < a' in if x1
2   then LNF[(l,a,(leaf,a',r))]
3   else LNF[((l,a',leaf),a,r)]
```

(b) Recursive translation to let-normal-form

```
1  let x1 = a < a' in if x1
2    then let x2 = leaf in let x3 = (x2, a', r) in (l, a, x3)
3    else let x4 = leaf in let x5 = (l, a', x4) in (x5, a, r)
```

(c) Completed translation to let-normal-form.

Figure 5.1: Example of translation to let-normal-form.

```
1  g (f y) y
```

(a) Before unsharing. Note that `y` is shared.

```
1  share y ≡ y1 ≡ y2 in
2    let x = f y1 in
3      g x y2
```

(b) After unsharing. Potential of `y` is carried by `y1` and `y2`.

Figure 5.2: Example of unsharing a function call.

The weakening rule could in principle be applied at each AST node, giving the constraint solver more freedom to find a solution. This degree of freedom needs to be controlled by the tool designer. In addition, recall that the suggested implementation of the weakening rule (see Section 4.1) is to be parameterised by the expert knowledge, fed into the weakening rule. In our experiments we noticed that the weakening rule has to be applied sparingly in order to avoid an explosion of the resulting constraint system.

We summarise the degrees of freedom available to the tool designer, which can be specified as parameters to ATLAS on source level.

1.) The selected template potential functions, i.e. the family of indices $\vec{a}, b$ for which coefficients $q_{(\vec{a},b)}$ are generated (we assume not explicitly generated are set to zero).

2.) The number of annotated signatures (with costs and without costs) for each function.

3.) The policy for applying the (parameterised) weakening rule.

We now discuss our choices for the aforementioned degrees of freedom.

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

Potential Function Templates

For each node in the AST of the considered input program, where n variables of type Tree are currently in context, we create the coefficients $q_1, \ldots, q_n$ for the rank functions and the coefficients $q_{(\vec{a},b)}$ for the logarithmic terms, where $\vec{a} \in \{0,1\}^n$ and $b \in \{0,2\}$. This choice turned out to be sufficient in our experiments.

Our potential-based method employs linear combinations of basic potential functions $\mathcal{BF}$, see Definition 2. In order to fix the cardinality of the set of resource functions to be considered, we restrict the coefficients of the potential functions $p_{(a_1,\ldots,a_m,b)}$. For the non-constant part, we demand that $a_i \in \{0,1\}$, while the coefficients b, representing the constant part are restricted to $\{0,1,2\}$. This restriction to a relative small set of basic potential functions suitably controls the number of constraints generated for each inference rule in the type system.

Number of Function Signatures

We fix the number of annotations for each function f to (i) One indeterminate type annotation representing a function call with costs; (ii) one indeterminate cost-free type annotation to represent a zero-cost call; and (iii) one fixed cost-free annotation with the empty annotation that doesn't carry any potential. These restrictions were sufficient to handle our benchmarks. A larger, potentially infinite set of type annotations is conceivable, as long as well-typedness is respected, see Definition 7. As noted in the context of the analysis of constant amortised complexity an enlarged set of type annotations may be even required to handle non-tail recursive programs, see [JKM12a; JAS17].

Weakening

In a *weakening* step, we need to discharge symbolic comparisons of form

$$\Phi(\Gamma|P) \leqslant \Phi(\Gamma|Q) \ .$$

In the implementation, we slightly simplify the constraints in Equation 4.7 by setting $\vec{b} = \vec{0}$. and treating c_p and c_q like the non-constant functions, moving them into matrix A, thus obtaining $\vec{p}^T \leqslant \vec{f}^T A + \vec{q}^T \leqslant \vec{0}$. As indicated in Section 4.1, we employ Farkas' Lemma to derive constraints for the weakening rule. For context $\Gamma = t_1, \ldots, t_n$, we introduce variables $x_{(\vec{a},b)}$ where $\vec{a} \in \{0,1\}^n, b \in \{0,2\}$, which represent the potential functions $p_{(\vec{a},b)} = \log(a_1|t_1| + \ldots + a_n|t_n| + b)$. Next, we explain how the monotonicity of log and Lemma 4 can be used to derive inequalities on the variables $x_{(\vec{a},b)}$; these inequalities can be used to instantiate matrix A in Farkas' Lemma as stated in Section 4.1.

Monotonicity We observe that if $a_1 \leqslant a'_1, \ldots, a_n \leqslant a'_n$ and $b \leqslant b'$, then

$$p_{(\vec{a},b)} = \log(a_1|t_1| + \ldots + a_n|t_n| + b) \leqslant \log(a'_1|t_1| + \ldots + a'_n|t_n| + b') = p_{(\vec{a}',b')} \ .$$

From this observation we obtain the lattice shown in Figure 5.3. A path from $x_{(\vec{a}',b')}$ to $x_{(\vec{a},b)}$ signifies

$$\begin{array}{lll} x_{(\vec{a},b)} & & \leqslant \ x_{(\vec{a}',b')} \\ x_{(\vec{a},b)} & -x_{(\vec{a}',b')} & \leqslant \ 0 \end{array}$$

which we represent by a row with coefficients 1 and -1 in the corresponding columns of matrix A.

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

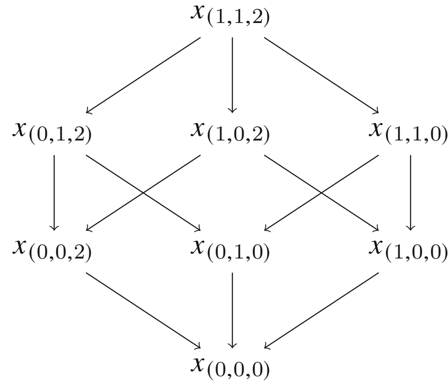

Figure 5.3: Monotonicity Lattice for $|Q| = 2$.

Mathematical facts, like Lemma 4 For an annotated context of length 2, Lemma 4 can be stated by the inequality

$$\begin{array}{llllll} 2x_{(0,0,2)} & +x_{(0,1,0)} & +x_{(1,0,0)} & & \leqslant & 2x_{(1,1,0)} \\ 2x_{(0,0,2)} & +x_{(0,1,0)} & +x_{(1,0,0)} & -2x_{(1,1,0)} & \leqslant & 0 \end{array}$$

we add a corresponding row with coefficients $2, 1, 1, -2$ to the matrix A. Likewise, for contexts of length > 2, we add, for each subset of 2 variables, a row with coefficients $2, 1, 1, -2$, setting the coefficients of all other variables to 0.

Sparse Expert Knowledge Matrix We observe for both kinds of constraints that matrix A is sparse. We exploit this in our implementation and only store non-zero coefficients.

Parametrisation of Weakening Each applications of the weakening rule is parametrised by the matrix A. In our tool, we instantiate A with either the constraints for

(i) monotonicity, shortly referenced as `w{mono}`;

(ii) Lemma 4 (`w{l2xy}`);

(iii) both (`w{mono l2xy}`); or

(iv) none of the constraints (`w`).

In the last case, Farkas' Lemma is not needed because weakening defaults to point-wise comparison of the coefficients $p_{(\vec{a},b)}$, which can be implemented more directly. Each time we apply weakening, we need to choose how to instantiate matrix A. Our experiments demonstrate that we need to apply monotonicity and Lemma 4 sparingly in order to avoid blowing up the constraint system.

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

Tactics and Automation

ATLAS supports manually applying the weakening rule — for this the user has to provide a tactic — and a fully automated mode.

The first goal of the implementation was to verify the annotations obtained from manual proofs. We developed a system that allows the user to give the input program and provide tactics to guide the proof derivation, fixing the decision on when to apply non-syntax-directed rules such as (w). Furthermore we allow the user to fix coefficients by giving explicit annotations. The implementation follows the tactics and derives constraints for the given program, which are finally sent to the SMT solver for obtaining a solution or proving unsatisfiability. A tactic is given as a text file that essentially contains a tree of rule names that indicates how rules should be applied to the syntax tree of the input program. The basic node is a rule name, e.g. `match` for the (match) rule. Depending on the number of sub-expressions a node may have several children that direct how the sub-expressions should be proved. Some special nodes exist that allow the user to control the behaviour of the prover. The user can place a `?` node to indicate a hole in the proof, place a sub-tree under a `!` node to fix the left- and right-hand-side annotations of the nested proof, or a `_` node to use a heuristic for choosing a rule. Figure 4.1 shows the basic syntax of a tactics file. Additionally the user can name sub-trees for reference in the result of the analysis and include ML-style comments in the tactics text.

Our tool also features a fully automated mode. However, the automated analysis is currently very memory intensive at the moment and is subject to ongoing performance optimisations.

Naive Automation

Our first attempt to automation applied the weakening rule everywhere instantiated with the full amount of available expert knowledge. This approach did not scale.

Manual Mode via Tactics

A tactic is given as a text file that contains a tree of rule names corresponding to the AST nodes of the input program, into which the user can insert applications of the weakening rule, parametrised by the expert knowledge which should be applied. A simple tactic is depicted in Figure 4.1. Tactics are distributed with ATLAS, see [Lor21a]. The user can name sub-trees for reference in the result of the analysis and include ML-style comments in the tactics text. We provide two special commands that allow the user to directly deal with a whole branch of the input program: The question mark (?) allows partial proofs; no constraints will be created for the part of the program thus marked. The underscore (_) switches to the naive automation of ATLAS and will apply the weakening rule with full expert knowledge everywhere. Both, ? and _, were invaluable when developing and debugging the automated mode. We note that the manual mode still achieves solving times that are by a magnitude faster than the automated mode, which may be of interest to a user willing to hand-optimise solving times.

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

Automated Mode

For automation, we extracted common patterns from the tactics we developed manually: Weakening with mode `w{mono}` is applied before (var) and (leaf), `w{mono l2xy}` is applied only before (app). Further, for AST subtrees that construct trees, i.e. which only consist of (node), (var) and (leaf) rule applications, we apply `w{mono}` for each inner node, and `w{l2xy}` for each outermost node. For all other cases, no weakening is applied. This approach is sufficient to cover all benchmarks, with further improvements possible.

5.1.3 Solving

For solving the generated constraint system, we rely on the Z3 SMT solver, see [MB08]. We employ Z3's Java bindings, load Z3 as a shared library, and exchange constraints for solutions. ATLAS forwards user-supplied configuration to Z3, which allows for flexible tuning of solver parameters. We also record Z3's statistics, most importantly memory usage. During the implementation of ATLAS, Z3's feature to extract unsatisfiable cores has proven valuable. It supplied us with many counterexamples, often directly pinpointing bugs in our implementation. The tool exports constraint systems in SMT-LIB format to the file system. This way, solutions could be cross-checked by re-computing them with other SMT solvers that support minimisation, such as OptiMathSAT [ST15].

5.2 Optimisation

Given an annotated function $f : \alpha_1 \times \cdots \times \alpha_n | Q \to \beta | Q'$, we want to find values for the coefficients of the resource annotations Q and Q' that minimise $\Phi(\Gamma|Q) - \Phi(\Gamma|Q')$, since this difference is an upper bound on the amortised cost of f, see Section 4.2.4 . However, as with weakening, we cannot directly express such a minimisation, and again resort to linearisation: We choose an optimisation function that directly maps from Q and Q' to $\mathbb{Q}$. Our optimisation function combines four measures, three of which involve a difference between coefficients of Q and Q', and a fourth one that only involves coefficients from Q in order to minimise the absolute values of the discovered coefficients. We first present these measures for the special case of $|Q| = 1$.

The first measure $d_1(Q, Q') := q_* - q'_*$ reflects our goal of preserving the coefficient for rk; note that for $d_1(Q, Q') \neq 0$, the resulting complexity bound would be super-logarithmic. The second measure $d_2(Q, Q') := \sum_{(a,b)} (q_{(a,b)} - q'_{(a,b)}) \cdot w(a, b)$ reflects the goal of achieving logarithmic bounds that are as small as possible. Weights are defined to penalise more complex terms, and to exclude constants. (Recall that 1 is representable as $\log(0 + 2)$.) We set

$$w(a, b) := \begin{cases} 0, & \text{for } (a, b) = (0, 2), \\ (a + (b + 1)^2)^2, & \text{otherwise.} \end{cases}$$

The third measure $d_3(Q, Q') := q_{(0,2)} - q'_{(0,2)}$ reflects the goal of minimising constant cost. Lastly, we set $d_4(Q, Q') := \sum_{(a,b)} q_{(a,b)}$ in order to obtain small absolute numbers. The last measure does not influence bounds on the amortised cost, but leads to more beautiful solutions. These measures are then composed to the linear objective function $\min \sum_{i=1}^{4} d_i(Q, Q') \cdot w_i$. In our implementation,

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

we set $w_i = [16127, 997, 97, 2]$; these weights are chosen (almost) arbitrary, we only noticed that w_1 must be sufficiently large to guarantee its priority.

For $|Q| > 1$, we set $d_1 := \sum_{i=1}^{|Q|} q_i - q'_*$ and $d_2(Q, Q') := \sum_{(a,a,\dots,b)} (q_{(a,a,\dots,b)} - q'_{(a,b)}) \cdot w(a, b)$. The required changes for d_3 and d_4 are straight-forward. In our benchmarks, there is only one function (`merge` of pairing heaps) that requires this minimisation function.

Our main results have already been stated in Table 1.1 in Chapter 1. We illustrate in Table 5.1a that the naive automation does not terminate within 24h for the core operations of the three considered data structures, whereas the improved automated mode produces optimised results within minutes. Here, „Selective“ means that limited expert knowledge is chosen by the automated mode, „all“ means that monotonicity and Lemma 4 are used. Timeouts are denoted by „t/o“. Naive automation does not support selection of expert knowledge for weakening, thus resulting in no answer, denoted „n/a“. In Table 5.1b, we compare the (improved) automated mode with the manual mode, and report on the sizes of the resulting constraint system and on the resources required to produce the same results. Observe that even though our automated mode achieves reasonable solving times, there is still a significant gap between the manually crafted tactics and the automated mode, which invites future work.

5.3 Evaluation

We first describe the benchmark functions employed to evaluate ATLAS and then detail this experimental evaluation, already depicted in Table 1.1.

5.3.1 Automated Analysis of Splaying et al.

Splay Trees

Introduced by Sleator and Tarjan [DR85; RE 85], *splay trees* are self-adjusting binary search trees with strictly increasing in-order traversal, but without an explicit balancing condition. Based on splaying, searching is performed by splaying with the sought element and comparing to the root of the result. Similarly, insertion and deletion are based on splaying. Above we used the zig-zig case of splaying, depicted in Figure 2.2 as motivating code example. While the pen-and-paper analysis of this case is the most involved, type inference for this case alone did not directly yield the desired automation of the complete definition. Rather, full automation required substantial implementation effort, as detailed in Chapter 5. As already emphasised, it came as a suprise to us that our tool ATLAS is able match up and partly improve upon the sophisticated optimisations perfomed by Schoemakers [Sch92; B S93]. This seems to be evidence of the versatility of the employed potential functions. Further, we leverage the sophistication of our optimisation layer in conjunction with the current power of state-of-the-art constraint solvers, like Z3 [MB08].

Splay Heaps

To overcome deficiencies of splay trees when implemented functionally, Okasaki introduced *splay heaps*. Splay heaps are defined similarly to splay trees and its (manual) amortised cost analysis follows similar pattern than the one for splay trees. Due to the similarity in the definitions between splay heaps

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

Function	Proof (w)	automated (naive)		automated (improved)		manual	
ST.splay (zig-zig)	Selective	n/a		7718	18S	2552	<1S
	All	11792	45S	9984	19S	2864	<1S
ST.splay	Selective	n/a		42095	8M1S	19111	12S
	All	68103	t/o 24H	54377	14M19S	23323	1M27S
SH.partition	Selective	n/a		33729	7M9S	15213	6S
	All	51995	t/o 24H	43549	15M2S	16829	10S
PH.merge_pairs	Selective	n/a		25860	1M3S	6414	<1S
	All	43515	t/o 24H	34918	13M41S	6558	<1S

(a) Comparison of the number of constraints generated and time taken for the type inference of the core operation of each benchmark plus the zig-zig case of splay.

Module	automated			manual		
	Assertions	Time	Memory	Assertions	Time	Memory
ST	54794	24M17S	3204	24677	43S	280
SH	37911	7M35S	1482	17877	12S	237
PH	29493	3M42S	760	7987	1S	29

(b) Number of assertions, solving time and maximum memory usage (in mebibytes) for the combined analysis of functions per-module.

Table 5.1: Experimental Results

and splay trees, extension of our experimental results in this direction did not pose any problems. Notably, however, ATLAS derives fully automatically the best (or slight improvements of) known complexity bounds on the amortised complexity for the functions studied. We also remark that typical assumptions made in pen-and-paper proofs are automatically discharged by our approach. To wit, Shoenmakers [Sch92; B S93] as well as Nipkow and Brinkop [NB19] make use of the (obvious) fact that the size of the resulting tree t' or heap h' equals the size of the input. As discussed, this information is captured through a cost-free derivation, see Section 2.1.

Pairing Heaps

These are another implementation of heaps, which are represented as binary trees, subject to the invariant that they are either `leaf`, or the right child is `leaf`, respectively. The left child is conceivable as list of pairing heaps. Schoenmakers and Nipkow et al. provide a (semi-)manual analysis of pairing heaps, that ATLAS can verify or even improve fully-automatically. We note that we analyse a single function `merge_pairs`, whereas [NB19] breaks down the analysis and studies two functions `pass_1` and `pass_2` with `merge_pairs` = `pass_2` ∘ `pass_1`. All definitions can be found at [Lor21b].

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

5.3.2 Experimental Results

Our main results have already been stated in Table 1.1 of Section 1.2. Table 5.1a compares the differences between the „naive automation" and our actual automation („automated mode"), see Section 5.1. Within the latter, we distinguish between a „selective" and a „full" mode. The „selective" mode is as described in Section 5.1.2. The „full" mode employs weakening for the same rule applications as the „selective" mode, but always with option `w{mono l2xy}`. The same applies to the „full" manual mode. The naive automation does not support selection of expert knowledge. Thus the „selective" option is not available, denoted as „n/a". Timeouts are denoted by „t/o". As depicted in the table, the naive automation does not terminate within 24h for the core operations of the three considered data structures, whereas the improved automated mode produces optimised results within minutes. In Table 5.1b, we compare the (improved) automated mode with the manual mode, and report on the sizes of the resulting constraint system and on the resources required to produce the same results. Observe that even though our automated mode achieves reasonable solving times, there is still a significant gap between the manually crafted tactics and the automated mode, which invites future work.

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

CHAPTER 6

Conclusion

We have presented an amortised resource analysis using the potential method. Potential functions take the shape of „sums of logarithms". The method is rendered in a type-and-effect system. Our type system has been carefully designed with the goal of automation, crucially invoking Farkas' Lemma for the linear part of the calculations and adding necessary facts about the logarithm.

Our contribution is novel, in the sense that this is the first approach to automation of a *logarithmic* amortised complexity analysis. In particular, our system automatically infers competitive results for the logarithmic amortised cost of multiple operations on various self-balancing data structures such as splay trees, splay heaps and pairing heaps.

As our potential functions are logarithmic, we cannot directly encode the comparison between logarithmic expressions within the theory of linear arithmetic. This however is vital for e.g. expressing Schhoenmakers' and Nipkow's (manual) analysis [B S93; T N15] in our type-and-effect system. In order to overcome this algorithmic challenge, we proposed several ideas for the *linearisation* of the induced constraint satisfaction problem.

These efforts can be readily extended by expanding upon the *expert knowledge* currently employed, e.g. via incorporation of the results of a static analysis performed in a pre-processing step.

Immediate future work is concerned with replacing the „sum of logarithms" potential function in order to analyse skew heaps and Fibonacci heaps [Sch92]. In particular, the potential function for skew heaps, which counts „right heavy" nodes, is interesting, because this function is used as a building block by Iacono in his improved analysis of pairing heaps [Iac00; IY16]. Further, we envision to extend our analysis to related probabilistic settings such as the analysis of priority queues [GM98] and skip lists [Pug90].

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

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

List of Figures

2.1 Big-Step Semantics 18
2.2 Function Definition `SplayTree.splay`. 19
2.3 Function Definition `SplayTree.insert`. 20
2.4 Function Definition `SplayTree.delete`. 20
2.5 Function Definition `SplayTree.splay_max`. 20

3.1 Type System for Logarithmic Amortised Resource Analysis 39
3.2 Partial Typing Derivation for `splay`, focusing on the zig-zig Case. 40
3.3 Cost-Free Derivation for `splay`, focusing on the zig-zig Case. 40

4.1 Tactic that matches the zig-zig case of `splay` as shown in Fig. 2.2. 46

5.1 Example of translation to let-normal-form. 51
5.2 Example of unsharing a function call. 51
5.3 Monotonicity Lattice for $|Q| = 2$. 53

List of Tables

1.1 Amortised complexity bounds for splay trees (module name `SplayTree`, abbrev. `ST`), splay heaps (`SplayHeap`, `SH`) and pairing heaps (`PairingHeap`, `PH`). 6

5.1 Experimental Results 57

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

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

Bibliography

[A F17] A. Flores-Montoya. „Cost Analysis of Programs Based on the Refinement of Cost Relations“. PhD thesis. Darmstadt University of Technology, Germany, 2017. URL: http://tuprints.ulb.tu-darmstadt.de/6746/.

[AA04] A. Podelski and A. Rybalchenko. „A Complete Method for the Synthesis of Linear Ranking Functions“. In: *Proc. 5th VMCAI*. Vol. 2937. LNCS. 2004, pp. 239–251.

[AEM11] M. Avanzini, N. Eguchi, and G. Moser. „A Path Order for Rewrite Systems that Compute Exponential Time Functions“. In: *Proceedings of the 22nd RTA*. Vol. 10. LIPIcs. 2011, pp. 123–138. DOI: 10.4230/LIPIcs.RTA.2011.123.

[AG12] D. E. Alonso-Blas and S. Genaim. „On the Limits of the Classical Approach to Cost Analysis“. In: *Proc. 19th SAS*. Ed. by A. Miné and D. Schmidt. Vol. 7460. LNCS. Springer, 2012, pp. 405–421. DOI: 10.1007/978-3-642-33125-1_27.

[Alb+08] E. Albert et al. „Automatic Inference of Upper Bounds for Recurrence Relations in Cost Analysis“. In: *Proc. 15th SAS*. Vol. 5079. 2008, pp. 221–237. DOI: 10.1007/978-3-540-69166-2_15.

[B P02] B. Pierce. *Types and programming languages*. MIT Press, 2002.

[B S93] B. Schoenmakers. „A Systematic Analysis of Splaying“. In: *IPL* 45.1 (1993), pp. 41–50.

[BH19] A. M. Ben-Amram and G. W. Hamilton. „Tight Worst-Case Bounds for Polynomial Loop Programs“. In: *Proc. 22nd FOSSACS*. Ed. by M. Bojanczyk and A. Simpson. Vol. 11425. LNCS. Springer, 2019, pp. 80–97. DOI: 10.1007/978-3-030-17127-8_5.

[BJH18] S. Bauer, S. Jost, and M. Hofmann. „Decidable Inequalities over Infinite Trees“. In: *Proc. 22nd LPAR*. Ed. by G. Barthe, G. Sutcliffe, and M. Veanes. Vol. 57. EPiC Series in Computing. EasyChair, 2018, pp. 111–130. URL: https://easychair.org/publications/paper/SSpj.

[Bla+10] R. Blanc et al. „ABC: Algebraic Bound Computation for Loops“. In: *Proc. 16th LPAR*. Vol. 6355. LNCS. 2010, pp. 103–118. DOI: 10.1007/978-3-642-17511-4_7.

[Brá+18] T. Brázdil et al. „Efficient Algorithms for Asymptotic Bounds on Termination Time in VASS“. In: *Proc. 33rd LICS*. Ed. by A. Dawar and E. Grädel. ACM, 2018, pp. 185–194. DOI: 10.1145/3209108.3209191.

[C A+10] C. Alias et al. „Multi-dimensional Rankings, Program Termination, and Complexity Bounds of Flowchart Programs“. In: *Proc. 17th SAS*. Vol. 6337. LNCS. 2010, pp. 117–133.

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

[C O99] C. Okasaki. *Purely functional data structures*. Cambridge University Press, 1999.

[CDZ14] T. Colcombet, L. Daviaud, and F. Zuleger. „Size-Change Abstraction and Max-Plus Automata“. In: *Proc. 39th MFCS*. Ed. by E. Csuhaj-Varjú, M. Dietzfelbinger, and Z. Ésik. Vol. 8634. LNCS. Springer, 2014, pp. 208–219. DOI: 10.1007/978-3-662-44522-8_18.

[CFG17] K. Chatterjee, H. Fu, and A. K. Goharshady. „Non-polynomial Worst-Case Analysis of Recursive Programs“. In: *Proc. 29th CAV*. Vol. 10427. LNCS. 2017, pp. 41–63. DOI: 10.1007/978-3-319-63390-9_3.

[DR85] D. Sleator and R. Tarjan. „Self-Adjusting Binary Search Trees“. In: *JACM* 32.3 (1985), pp. 652–686. DOI: 10.1145/3828.3835.

[E A+11] E. Albert et al. „Closed-Form Upper Bounds in Static Cost Analysis“. In: *JAR* 46.2 (2011).

[E C+05] E. Contejean et al. „Mechanically proving termination using polynomial interpretations“. In: *JAR* 34.4 (2005), pp. 325–363.

[Fie+18] T. Fiedor et al. „From Shapes to Amortized Complexity“. In: *Proc. 19th VMCAI*. Ed. by I. Dillig and J. Palsberg. Vol. 10747. LNCS. Springer, 2018, pp. 205–225. DOI: 10.1007/978-3-319-73721-8_10.

[FKN17] C. Fuhs, C. Kop, and N. Nishida. „Verifying Procedural Programs via Constrained Rewriting Induction“. In: *TOCL* 18.2 (2017), 14:1–14:50. DOI: 10.1145/3060143.

[GM18] G. Moser and M. Schneckenreither. „Automated Amortised Resource Analysis for Term Rewrite Systems“. In: *Proc. 14th FLOPS*. Vol. 10818. LNCS. 2018, pp. 214–229. DOI: 10.1007/978-3-319-90686-7.

[GM98] A. Gambin and A. Malinowski. „Randomized Meldable Priority Queues“. In: *SOFSEM '98: Theory and Practice of Informatics, 25th Conference on Current Trends in Theory and Practice of Informatics, Jasná, Slovakia, November 21-27, 1998, Proceedings*. Ed. by B. Rovan. Vol. 1521. Lecture Notes in Computer Science. Springer, 1998, pp. 344–349. DOI: 10.1007/3-540-49477-4_26.

[GZ10] S. Gulwani and F. Zuleger. „The reachability-bound problem“. In: *PLDI*. Ed. by B. G. Zorn and A. Aiken. ACM, 2010, pp. 292–304. DOI: 10.1145/1806596.1806630.

[Har07] J. Harrison. „Verifying Nonlinear Real Formulas Via Sums of Squares“. In: *Theorem Proving in Higher Order Logics, 20th International Conference, TPHOLs 2007, Kaiserslautern, Germany, September 10-13, 2007, Proceedings*. Ed. by K. Schneider and J. Brandt. Vol. 4732. Lecture Notes in Computer Science. Springer, 2007, pp. 102–118. DOI: 10.1007/978-3-540-74591-4_9.

[HR13] M. Hofmann and D. Rodriguez. „Automatic Type Inference for Amortised Heap-Space Analysis“. In: *Proc. 22nd ESOP*. Ed. by M. Felleisen and P. Gardner. Vol. 7792. LNCS. Springer, 2013, pp. 593–613. DOI: 10.1007/978-3-642-37036-6_32.

[Iac00] J. Iacono. „Improved Upper Bounds for Pairing Heaps“. In: *Algorithm Theory - SWAT 2000, 7th Scandinavian Workshop on Algorithm Theory, Bergen, Norway, July 5-7, 2000, Proceedings*. Ed. by M. M. Halldórsson. Vol. 1851. Lecture Notes in Computer Science. Springer, 2000, pp. 32–45. DOI: 10.1007/3-540-44985-X_5.

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

[IY16] J. Iacono and M. V. Yagnatinsky. „A Linear Potential Function for Pairing Heaps“. In: *Combinatorial Optimization and Applications - 10th International Conference, COCOA 2016, Hong Kong, China, December 16-18, 2016, Proceedings*. Ed. by T.-H. H. Chan, M. Li, and L. Wang. Vol. 10043. Lecture Notes in Computer Science. Springer, 2016, pp. 489–504. DOI: 10.1007/978-3-319-48749-6_36.

[J G+17] J. Giesl et al. „Analyzing Program Termination and Complexity Automatically with AProVE“. In: *JAR* 58.1 (2017), pp. 3–31. DOI: 10.1007/s10817-016-9388-y.

[J H11] J. Hoffmann. „Types with Potential: Polynomial Resource Bounds via Automatic Amortized Analysis“. PhD thesis. Ludwig-Maximilians-Universiät München, 2011.

[JAS17] J. Hoffmann, A. Das, and S-C. Weng. „Towards automatic resource bound analysis for OCaml“. In: *Proc. 44th POPL*. ACM, 2017, pp. 359–373. DOI: 10.1145/3009837.

[JKM11] J. Hoffmann, K. Aehlig, and M. Hofmann. „Multivariate Amortized Resource Analysis“. In: *Proc. 38th POPL*. ACM, 2011, pp. 357–370.

[JKM12a] J. Hoffmann, K. Aehlig, and M. Hofmann. „Multivariate amortized resource analysis“. In: *TOPLAS* 34.3 (2012), p. 14.

[JKM12b] J. Hoffmann, K. Aehlig, and M. Hofmann. „Resource Aware ML“. In: *Proc. 24th CAV*. Vol. 7358. LNCS. 2012, pp. 781–786.

[JM10a] J. Hoffmann and M. Hofmann. „Amortized Resource Analysis with Polymorphic Recursion and Partial Big-Step Operational Semantics“. In: *Proc. 8th APLAS*. Vol. 6461. LNCS. 2010, pp. 172–187.

[JM10b] J. Hoffmann and M. Hofmann. „Amortized Resource Analysis with Polynomial Potential“. In: *Proc. 19th ESOP*. Vol. 6012. LNCS. 2010, pp. 287–306.

[JZ14] J. Hoffmann and Z. Sho. „Type-Based Amortized Resource Analysis with Integers and Arrays“. In: *Proc. 12th FLOPS*. Vol. 8475. LNCS. 2014, pp. 152–168.

[JZ15a] J. Hoffmann and Z. Shao. „Automatic Static Cost Analysis for Parallel Programs“. In: *Proc. 24th ESOP*. Vol. 9032. LNCS. 2015, pp. 132–157.

[JZ15b] J. Hoffmann and Z. Shao. „Type-based amortized resource analysis with integers and arrays“. In: *JFP* 25 (2015). DOI: 10.1017/S0956796815000192.

[KH20] D. M. Kahn and J. Hoffmann. „Exponential Automatic Amortized Resource Analysis“. In: *Proc. 23rd FOSSACS*. Ed. by J. Goubault-Larrecq and B. König. Vol. 12077. LNCS. Springer, 2020, pp. 359–380. DOI: 10.1007/978-3-030-45231-5_19.

[LGF21] Lorenz Leutgeb, Georg Moser, and Florian Zuleger. „ATLAS: Automated Amortised Complexity Analysis of Self-Adjusting Data Structures“. In: *Computer Aided Verification - 33rd International Conference, CAV 2021, July 18-23, Online, 2021, Proceedings*. Lecture Notes in Computer Science. to appear. Springer, 2021.

[Lor21a] Lorenz Leutgeb. *ATLAS: Automated Amortised Complexity Analysis of Self-Adjusting Data Structures*. Software. 2021. DOI: 10.5281/zenodo.4724917.

[Lor21b] Lorenz Leutgeb. *ATLAS: Examples*. Dataset. 2021. DOI: 10.5281/zenodo.4880499.

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

[Mar+21] Martin Hofmann et al. „Type-Based Analysis of Logarithmic Amortised Complexity". In: *Mathematical Structures Computer Science* (2021). to appear.

[MB08] L. M. de Moura and N. 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 J. Rehof. Vol. 4963. Lecture Notes in Computer Science. Springer, 2008, pp. 337–340. DOI: 10.1007/978-3-540-78800-3_24.

[MG14] M. Hofmann and G. Moser. „Amortised Resource Analysis and Typed Polynomial Interpretations". In: *Proc. of Joint 25th RTA and 12th TLCA*. Vol. 8560. LNCS. 2014, pp. 272–286.

[MG15] M. Hofmann and G. Moser. „Multivariate Amortised Resource Analysis for Term Rewrite Systems". In: *Proc. 13th TLCA*. Vol. 38. LIPIcs. 2015, pp. 241–256. DOI: 10.4230/LIPIcs.TLCA.2015.241.

[MG16] M. Avanzini and G. Moser. „A Combination Framework for Complexity". In: *IC* 248 (2016), pp. 22–55. DOI: 10.1016/j.ic.2015.12.007.

[MGM16] M. Avanzini, G. Moser, and M. Schaper. „TcT: Tyrolean Complexity Tool". In: *Proc. 22nd TACAS*. Vol. 9636. LNCS. 2016, pp. 407–423. DOI: 10.1007/978-3-662-49674-9_24.

[MLG15] M. Avanzini, U. D. Lago, and G. Moser. „Analysing the complexity of functional programs: higher-order meets first-order". In: *Proc. 20th ICFP*. ACM, 2015, pp. 152–164. DOI: 10.1145/2784731.2784753.

[MS03] M. Hofmann and S. Jost. „Static prediction of heap space usage for first-order functional programs". In: *Proc. 30th POPL*. ACM, 2003, pp. 185–197.

[MS20] G. Moser and M. Schneckenreither. „Automated amortised resource analysis for term rewrite systems". In: *Sci. Comput. Program.* 185 (2020). DOI: 10.1016/j.scico.2019.102306.

[MV +12] M.V. Hermenegildo et al. „An overview of Ciao and its design philosophy". In: *TPLP* 12.1-2 (2012), pp. 219–252.

[NB19] T. Nipkow and H. Brinkop. „Amortized Complexity Verified". In: *JAR* 62.3 (2019), pp. 367–391. DOI: 10.1007/s10817-018-9459-3.

[Pug90] W. Pugh. „Skip Lists: A Probabilistic Alternative to Balanced Trees". In: *Commun. ACM* 33.6 (1990), pp. 668–676. DOI: 10.1145/78973.78977.

[RE 85] R.E. Tarjan. „Amortized Computational Complexity". In: *SIAM J. Alg. Disc. Meth* 6.2 (1985), pp. 306–318.

[S J+09] S. Jost et al. „"Carbon Credits" for Resource-Bounded Computations Using Amortised Analysis". In: *Proc. 2nd FM*. Vol. 5850. LNCS. 2009, pp. 354–369.

[S J+10] S. Jost et al. „Static determination of quantitative resource usage for higher-order programs". In: *Proc. 37th POPL*. ACM, 2010, pp. 223–236.

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

[S J+17] S. Jost et al. „Type-Based Cost Analysis for Lazy Functional Languages“. In: *JAR* 59.1 (2017), pp. 87–120. DOI: 10.1007/s10817-016-9398-9.

[Sch92] B. Schoenmakers. „Data Structures and Amortized Complexity in a Functional Setting“. PhD thesis. Eindhoven University of Technology, 1992.

[Sch99] A. Schrijver. *Theory of linear and integer programming*. Wiley, 1999. ISBN: 978-0-471-98232-6.

[SG20] S. Winkler and G. Moser. „Runtime Complexity Analysis of Logically Constrained Rewriting“. In: *Proc. LOPSTR 2020*. 2020.

[Sol09] A. Solar-Lezama. „The Sketching Approach to Program Synthesis“. In: *Programming Languages and Systems, 7th Asian Symposium, APLAS 2009, Seoul, Korea, December 14-16, 2009. Proceedings*. Ed. by Z. Hu. Vol. 5904. Lecture Notes in Computer Science. Springer, 2009, pp. 4–13. DOI: 10.1007/978-3-642-10672-9_3.

[ST15] R. Sebastiani and P. Trentin. „OptiMathSAT: A Tool for Optimization Modulo Theories“. In: *CAV*. 2015, pp. 447–454.

[SZV14] M. Sinn, F. Zuleger, and H. Veith. „A Simple and Scalable Static Analysis for Bound Analysis and Amortized Complexity Analysis“. In: *Proc. 26th CAV*. Vol. 8559. LNCS. 2014, pp. 745–761.

[SZV15] M. Sinn, F. Zuleger, and H. Veith. „Difference Constraints: An adequate Abstraction for Complexity Analysis of Imperative Programs“. In: *FMCAD*. Ed. by R. Kaivola and T. Wahl. IEEE, 2015, pp. 144–151.

[SZV17] M. Sinn, F. Zuleger, and H. Veith. „Complexity and Resource Bound Analysis of Imperative Programs Using Difference Constraints“. In: *JAR* 59.1 (2017), pp. 3–45. DOI: 10.1007/s10817-016-9402-4.

[T N15] T. Nipkow. „Amortized Complexity Verified“. In: *Proc. 6th ITP*. Vol. 9236. LNCS. 2015, pp. 310–324. DOI: 10.1007/978-3-319-22102-1_21.

[WWC17] P. Wang, D. Wang, and A. Chlipala. „TiML: A Functional Language for Practical Complexity Analysis with Invariants“. In: *Proc. ACM Program. Lang.* 1.OOPSLA (2017). DOI: 10.1145/3133903.

[Zul+11] F. Zuleger et al. „Bound Analysis of Imperative Programs with the Size-Change Abstraction“. In: *Proc. 18th SAS*. Ed. by E. Yahav. Vol. 6887. LNCS. Springer, 2011, pp. 280–297. DOI: 10.1007/978-3-642-23702-7_22.

[Zul15] F. Zuleger. „Asymptotically Precise Ranking Functions for Deterministic Size-Change Systems“. In: *Proc. 10th CSR*. Ed. by L. D. Beklemishev and D. V. Musatov. Vol. 9139. LNCS. Springer, 2015, pp. 426–442. DOI: 10.1007/978-3-319-20297-6_27.

[Zul20] F. Zuleger. „The Polynomial Complexity of Vector Addition Systems with States“. In: *Proc. 23rd FOSSACS*. Ed. by J. Goubault-Larrecq and B. König. Vol. 12077. LNCS. Springer, 2020, pp. 622–641. DOI: 10.1007/978-3-030-45231-5_32.

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