DIPLOMARBEIT

First-Order Reasoning with Aggregates

ausgeführt am

Institute of
Logic and Computation
TU Wien

unter der Anleitung von

Univ.Prof. Dr.techn. Laura Kovacs

durch

Sophie Rain, BSc.
Matrikelnummer: 01425316
Peigarten 58
2053 Peigarten

Wien, am 10. Mai 2020

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 research grants ERC Starting Grant SYMCAR 639270 and the ERC Proof of Concept Grant SYMELS 842066 are partially supporting scientific activities that led to 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.

Abstract

Aggregates are widely used in software verification and especially important for verifying smart contracts that deal with transactions of crypto-currencies between accounts. The formal verification of smart contracts is a very active area of research, since the main blocker for the use of smart contracts in every-life is the absence of a security guarantee. Vulnerabilities may have tremendous consequences, as recent bugs have shown.

In general, aggregates require higher-order logics, but for certain use-cases it is possible to express in first-order logic both the allowed operations and the desired properties. This thesis addresses this challenge by suggesting encodings for aggregates of finite but arbitrary length. We have focused on sums and developed a novel way to express them using uninterpreted first-order logic. Within this work we explain the setting and the intended applications accurately.

The main idea is to encode the crypto-currency as coins and to assign every existing coin to some account. This is realized using a binary relation. The invariants ensuring the sum to be correct encode that every existing coin is owned by precisely one address. The transactions, such as transferring money, can easily be fitted into this setting. The experimental results using Vampire and Z3 are promising. Both are able to prove interesting properties.

We have proven that this translation from higher-order logic to first-order logic is sound and complete. The proof is given in the present thesis. Further, the translation itself is first-order expressible. In fact, we present an encoding that implements this translation. As such, we also provided a way to directly reason about sums of non-negative integers. We present a naive encoding and two equally expressive restrictions. While the first one is crucial for the comparison of two sums, the second one only decreases the search space. We also give a proof that we do not lose any generality by restricting the encodings. The experiments show that automated theorem provers can handle both of the restricted versions well, whereas they struggle with the naive encoding. The reason is that the naive encoding does not provide any information about the relation of the coins in the two compared sums. However, we can prove various results in terms of sums using the restricted encodings.

We briefly explain how to adapt the presented invariants in order to be applicable for other aggregates such as the minimum or the maximum. The idea of using coins to represent the balances of accounts can be adapted, also the binary relation expressing the ownership of coins can be reused.

This work is, to the best of our knowledge, the first to encode sums of finite but arbitrary length in first-order logic. It also enables automated theorem provers to reason about aggregates and hence takes a step towards automatic verification of smart contracts.

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.

Kurzfassung

Aggregate werden häufig für Software Verifikation verwendet und sind besonders wichtig für Smart Contracts, nachdem diese Transaktionen von Kryptowährungen zwischen Konten tätigen. Die formale Verifikation von Smart Contracts ist ein sehr aktives Forschungsgebiet, weil die größte Hürde für eine alltägliche Verwendung von Smart Contracts im Fehlen von garantierter Veranlagungssicherheit besteht. Unschärfen im Source Code können enorme Auswirkungen haben, wie jüngste Softwarefehler gezeigt haben.
Im Allgemeinen benötigen Aggregate Logik höherer Stufe, aber für gewisse Fälle ist es möglich die gewünschten Operationen und Eigenschaften in Logik erster Stufe auszudrücken. In dieser Diplomarbeit versucht man dies für Aggregate beliebiger aber endlicher Länge. Es wurde ein Weg gefunden Summen in allgemeiner Logik erster Stufe auszudrücken. Die Konzepte und angestrebten Anwendungen werden in der Arbeit sorgfältig erklärt.
Die Hauptidee ist es Kryptowährung in Münzen zu kodieren und diese Konten zuzuordnen. Die Invarianten stellen sicher, dass die Summe korrekt ist, indem sie kodieren, dass jede aktive Münze genau einem Konto zugeordnet ist. Auch die Transaktionen können leicht in dieses Konzept eingebettet werden. Die Experimente mit Vampire und Z3 sind vielversprechend. Beide können interessante Eigenschaften beweisen.
In dieser Arbeit wurde bewiesen, dass die Übersetzung von Logik höherer Stufe in Logik erster Stufe 'sound' und 'complete' ist. Der Beweis ist Teil der Arbeit. Des Weiteren wird gezeigt, wie die Übersetzung selbst in Logik erster Stufe ausgedrückt werden kann. So eine Kodierung wird ebenfalls präsentiert. Dadurch haben wir einen Weg gefunden direkt über Summen von natürlichen Zahlen Schlüsse zu ziehen. Es werden eine naive Kodierung und zwei eingeschänkte Versionen vorgestellt. Die erste der Einschränkungen ist wesentlich um zwei Summen vergleichen zu können, die andere verkleinert lediglich den Suchbereich. Es wird ebenfalls bewiesen, dass die Einschränkungen die Allgemeinheit der Kodierung nicht beschränken. Die Experimente zeigen, dass 'Theorem Prover' mit beiden eingeschränkten Versionen gut umgehen können, aber Probleme mit der naiven Version haben. Der Grund dafür ist, dass die naive Version keinerlei Information beinhaltet in welcher Relation die beiden Summen stehen. Nichtsdestotrotz können einige Eigenschaften von Summen in den eingeschränkten Kodierungen bewiesen werden.
Des Weiteren wird vorgestellt, wie man die Konzepte auf andere Aggregate wie das Minimum oder das Maximum übertragen kann. Die Idee Münzen zu verwenden und sie durch eine Relation Konten zuzuordnen kann aufgegriffen werden.
Nach unserem besten Wissen ist diese Arbeit die erste die Summen endlicher aber beliebiger Länge in Logik erster Stufe kodiert. Sie ermöglicht es automatischen 'Theorem Provern' über Aggregate zu schlussfolgern und trägt dadurch zur automatischen Verifikation von Smart Contracts bei.

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.
TU WIEN Bibliothek Your knowledge hub

Acknowledgement

First of all, I want to express my gratefulness to Laura Kovacs for her excellent supervision and the incredibly fast and uncomplicated review process. She always took the time for my questions and supported me wherever possible. Moreover, she enabled me to participate in events such as summer schools, workshops and networking dinners.

Further, I want to thank Mooly Sagiv and Neta Elad from the University of Tel Aviv for their inspirations, inputs and numerous interesting discussions. I am also grateful for making my stay in Tel Aviv possible which led to crucial proceedings in my work.

People, without whom I never would have got this far, are my mother Christa, my father Wilfried and my sister Hannah. From the first days of my studies they stood by me wherever they could, even though they may have never fully understood my choice of studying mathematics. Especially in the first semesters, I really needed them mentally. They smoothed out every obstacle, where always listening to my problems and shared my sorrows. I never really expressed my thankfulness and want to use this opportunity. Thank you for everything you have done for me, I appreciacte all of your efforts (and spendings).

The same holds for my boyfriend Constantin. As he was also studying at TU Wien, he could always comprehend my worries and time constraints. Nevertheless, I want to apologize for every time I put the progress of my studies first. Thank you for your never ending understanding and patience.

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe.

Wien, am 10. Mai 2020

Sophie Rain

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

Contents

1 Introduction **1**
1.1 Motivating Example . . . 1
1.2 Problem Statement . . . 2
1.3 List of Contributions . . . 3

2 Preliminaries **4**
2.1 Aggregates . . . 4
2.2 Smart Contracts and Deductive Verification . . . 8
2.3 Automated Theorem Proving in First-Order Logic . . . 9

3 Encoding Finite Sums of Arbitrary Length **11**
3.1 Encoding of Sum Properties in First-Order Logic . . . 12
3.2 Transitions and Impact on Sums . . . 14
3.3 Experimental Results of the Sum Encoding . . . 22

4 The Encoding is Sound and Complete **26**
4.1 Higher-Order and First-Order States . . . 26
4.2 Equivalent First-Order States . . . 26
4.3 Translating Higher-Order States to First-Order States . . . 29
4.4 Soundness and Completeness . . . 30

5 The Translation f is First-Order Expressible **32**
5.1 Naive First-Order Encoding . . . 32
5.2 Restricted, yet Equally Expressive Encodings . . . 36
5.3 Experimental Results of the f Encodings . . . 45

6 Related Work **50**

7 Conclusion and Future Work **54**

Bibliography **56**

List of Figures **58**

List of Tables **59**

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.
TU WIEN Bibliothek Your knowledge hub

1 Introduction

Aggregates are widely used in software verification and especially important for verifying smart contracts that deal with transactions of crypto-currencies between accounts. One important aggregate property is preserving the sum of coins in the accounts when transferring money. To define aggregates, that is values such as the mean or the sum of some set of numerical elements, it is required to use expressions such as $\sum$ or variables with indices up to an arbitrary n - at least if it is done in the usual way. While the behaviour of '+' or '$\cdot$' can be expressed naturally in first-order logic, it seems hard or even impossible to fully define aggregates of arbitrary length in first-order logic.
However, if just some properties of the aggregate are of interest, there are possibilities to encode these. Some are presented in this thesis.

As the motivation of this work comes from formal verification of smart contracts, it is also tailored to fit this use case. The value of being able to reason about the sum and other aggregates is explained in the following motivating example.

1.1 Motivating Example

We illustrate below the relevance of aggregates for verifying smart contracts.

In 2018, there was the case, as reported in [1], that an adversarial user increased her personal wealth by exploiting an at that time unknown vulnerability of a smart contract. She asked an address A to send 2^{255} BEC tokens each to two different addresses B_1 and B_2. A BEC token is the unit of the currency used. Since the total amount of tokens asked for was $\texttt{amount} = 2^{255} \cdot 2 = 2^{256}$ and the variable `amount` was a 256-bit integer this led to an overflow and hence to $\texttt{amount} = 0$.
Thus, the sanity check of this transaction passed and A's balance was decreased by 0, while each of the balances of B_1 and B_2 was increased by

$$2^{255} \approx 578\,960\,446\,186\,580\,977\,117\,854\,925\,043\,440\,000\,000\,000\,000\,000\,000\,000\,000\,000\,000\,000\,000\,000\,000\,000\,000\,000\,000\,000\,000\,000\,000\,000\,000\,000\,000\,000 \text{ tokens.}$$

This was only discovered because some warning was raised that stated an unreasonably big amount of tokens was transferred. The details and consequences of this incident can be found in [1].

In this example the sanity check could not detect the problem due to integer overflow. By formally verifying that the sum of balances behaved as expected, that is remains unchanged for transferring actions, this bug would have been found before deployment.

```
1          address bank;
2          uint bal(address);
3
4          procedure donate(address[] receivers, uint a){
5          for(i = 0; i < receivers.length; i++){
6          transferFrom(bank, receivers[i], a);
7          }
8          }
```

Figure 1.1: Simplfied `donate` procedure.

The present thesis aims to support such formal verification. In simplified terms, the task we aim to address is the following.

Property to be ensured.
We know that the sum of all balances equals a certain constant `sum`, that is

$$\sum_{A \in \texttt{address}} \texttt{bal}(A) = \texttt{sum}$$

and we then execute the procedure `donate` from Figure 1.1 in which a bank `bank` donates `a` tokens to each of the receivers in the array `receivers`.

After execution of lines 4-8 of Figure 1.1 it still has to be the case that

$$\sum_{A \in \texttt{Address}} \texttt{bal}(A) = \texttt{sum} .$$

1.2 Problem Statement

The goal of this thesis is to find a sound way to encode what it means for two finite sums to be the same in first-order logic. It can be assumed that the relation of the elements leading to these sums is known. In order to do so, there have to be first-order formulas expressing the most important properties of sums in advance. The application of deductive verification should be in focus. That is finding invariants that ensure a certain behaviour of the sum:

(P1) **How can properties about sums be encoded in first-order logic and what are suitable invariants?**

Further, this encoding should be extended to similar problems, such as comparing sums of numeric values that only differ slighty. Again, the main goal is to find invariants for corresponding transitions:

(P2) **How can an earlier developed encoding be extended to related problems?**

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 next objective is to run experiments using first-order theorem provers to validate the usability of the before developed concepts in (P1) and (P2):

(P3) **Can automated theorem provers handle the encodings of (P1) and (P2)?**

Lastly, other aggregates shall be considered:

(P4) **Can the developed encodings from (P1) and (P2) be applied for other aggregates as well?**

1.3 List of Contributions

This work brings the following contributions to the problems listed in Section 1.2.

(P1) We present a novel way to reason about sums and their properties. In Chapter 3 an implicit encoding including invariants is proposed, whereas in Chapter 5 an explicit encoding without invariants is introduced. Both of them are sound and complete relative to a translation function f. The proofs of these claims are given in Chapter 4 and Section 5.2, respectively.

(P2) The encodings presented to address (P1) can also express related problems. For example, formalizing that one sum is greater or equal than another sum or two sums differ a constant. These related problems are considered together with the equality of sums in Chapter 3 and Chapter 5.

(P3) The performance of first-order theorem provers given the encodings is promising, as discussed in Section 3.2 and Section 5.3. The SMT solver Z3 [11] could prove all of the standard tasks and the first-order theorem prover Vampire [10] few tasks.

(P4) In Chapter 7, a brief overview on possibilities to adapt the presented encodings for other aggregates is given.

The word 'we' is used to address the reader only. The contributions and ideas listed here are owned by the author of this thesis. Only the idea of encoding balances and sums by using their units (Chapter 3) resulted from joint work with Neta Elad (Tel Aviv University).

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

2 Preliminaries

In this chapter the most important concepts underlying this work are presented briefly. Generally, $\mathbb{N}$ is defined to be the set of natural numbers, $\mathbb{N}^+$ is the set of positive natural numbers, that is $\mathbb{N}^+ := \mathbb{N} \setminus \{0\}$, and $\mathbb{R}$ is the set of real numbers.

2.1 Aggregates

Intuitively, aggregates are summary values. That means, given any n-tuple x of numerical values, we expect an aggregate of x to be a scalar that aims to summarize x. For the concrete example of $x \in \mathbb{R}^n$, we expect an aggregate $Agg(x) \in \mathbb{R}$ to be a representative value of x, such as the mean ot the median of x.

The concept of aggregation functions is formalized in [2] in the following way. It is assumed that all variables on which the aggregation function depends have a common domain $\mathbb{I}$, that is the elements of the function's codomain and the components of the elements in its domain. In [2], $\mathbb{I} \subseteq \overline{\mathbb{R}} = [-\infty, +\infty]$ and $\mathbb{I}$ is non-empty, a setting which will be also considered in this thesis.

Definition 1 (Aggregation Functions on $\mathbb{I}^n$)**.** Let $n \in \mathbb{N}^+$. Then $Agg^{(n)} : \mathbb{I}^n \to \mathbb{I}$ is an aggregation function, if:

(1) $Agg^{(n)}$ is nondecreasing, that is for $x, y \in \mathbb{I}^n$:

$$\Big(\forall i \in [1, n] : \; x_i \geq y_i\Big) \;\Rightarrow\; Agg^{(n)}(x) \geq Agg^{(n)}(y) \; .$$

(2) $Agg^{(n)}$ satisfies the boundary condition:

$$\inf_{x \in \mathbb{I}^n} \left(Agg^{(n)}(x)\right) = \inf\left(\mathbb{I}\right) ,$$
$$\sup_{x \in \mathbb{I}^n} \left(Agg^{(n)}(x)\right) = \sup\left(\mathbb{I}\right) .$$

Following Definition 1, the mean is an aggregation function for any $n \in \mathbb{N}^+$ and any interval $\mathbb{I}$, whereas the sum only for certain domains $\mathbb{I}$, such as $\mathbb{I} = \mathbb{R}$. Instances of this claim are shown in Example 1. For $n = 1$, usually $Agg^{(1)}(x) := x$ is used.

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.

Example 1.

1. The mean $M^{(n)}$ of an n-tuple is an aggregate for $\mathbb{I} = [a, b]$, where $a, b \in \mathbb{R}$, $a < b$. To show $M^{(n)}$ is nondecreasing, let now $x, y \in [a, b]^n$, with $x_i \geq y_i$, $i \in \{1, ..., n\}$. We have
$$M^{(n)}(x) := \frac{1}{n}\sum_{i=1}^{n} x_i \geq \frac{1}{n}\sum_{i=1}^{n} y_i =: M^{(n)}(y) \ .$$
Lastly, consider $y, z \in [a, b]^n$, $y_i = a$, $z_i = b$ for every $i \in \{1, ..., n\}$ to show the boundary condition. It holds $a = M^{(n)}(y)$ and for every $x \in \mathbb{I}^n$, $x \neq y$ we have $x_i \geq y_i$ for every i. Since $M^{(n)}$ is nondecreasing, it follows $M^{(n)}(x) \geq M^{(n)}(y)$ and thus $\inf_{x\in[a,b]^n}\left(M^{(n)}(x)\right) = a = \inf\left(\mathbb{I}\right)$. Similar for z and the supremum.
This also shows that the codomain is indeed $\mathbb{I}$.

2. The sum $S^{(n)}$ of an n-tuple, $n > 1$, is not an aggregate for $\mathbb{I} = [a, b]$, where $a, b \in \mathbb{R}$, $a < b$. Consider $x \in [a, b]^n$, with $x_i = b$ for all i. Since $n > 1$, we know
$$S^{(n)}(x) := \sum_{i=1}^{n} x_i = \sum_{i=1}^{n} b \geq \sum_{i=1}^{2} b > b \ .$$
Thus the codomain of $S^{(n)}$ is not $I = [a, b]$ and hence $S^{(n)}$ is not an aggregation function for $\mathbb{I} = [a, b]$.

To address arbitrary n, the following definition of an extended aggregation function is given.

Definition 2 (Extended Aggregation Functions on $\mathbb{I}$)**.** We say
$$Agg : \bigcup_{n\in\mathbb{N}^+} \mathbb{I}^n \to \mathbb{I}$$
is an extended aggregation function, if for every $n \in \mathbb{N}$, the restriction $Agg^{(n)} := Agg_{|_{\mathbb{I}^n}}$ is an aggregation function.

Example 2. The mean
$$M : x \mapsto \frac{1}{|x|}\sum_{i=1}^{|x|} x_i$$
is an extended aggregation function for $I = [a, b]$, $a < b$. The function M assigns every element of $\bigcup_{n\in\mathbb{N}^+}[a, b]^n$ a value, thus the domain is correct. Since $M_{|_{[a,b]^n}} = M^{(n)}$ and $M^{(n)}$ is an aggregation function for every $n \in N^+$, the codomain of M is also $[a, b]$, thus M is an extended aggregation function.

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

By Definition 2, the following functions are among others extended aggregation functions.

- The arithmetic mean on any interval $\mathbb{I}$.
- The median on any $\mathbb{I}$.
- The sum on $\mathbb{I} = \mathbb{R}$.
- Minimum and maximum on any $\mathbb{I}$.
- The product on $\mathbb{I} = \mathbb{R}$.
- The projection onto the i-th element for every $\mathbb{I}$.

Note the length of the input tuple is not an extended aggregate for any $\mathbb{I}$.

As this work is motivated by smart contracts, we deal with addresses and their balances, as explained in Section 2.2, instead of tuples $x \in \mathbb{I}^n$. Hence, we have to adapt these definitions slightly in order to be useful. For us, any finite non-empty set $\mathcal{A}$ without any ordering can represent the addresses. As a consequence, the relevant domain is

$$\mathcal{D}^{(n)} := \bigcup_{\mathcal{A}\colon |\mathcal{A}|=n} \mathbb{I}^{\mathcal{A}} .$$

Additionally, we do not want aggregates to depend on the names of the addresses. Thus, we define an *adapted aggregation function* in the following way.

Definition 3 (Adapted Aggregation Function on $\mathbb{I}$)**.** Let $n \in \mathbb{N}^+$, $Ada^{(n)} : \mathcal{D}^{(n)} \to \mathbb{I}$. Then $Ada^{(n)}$ is an adapted aggregation function, if:

(1) $Ada^{(n)}$ is nondecreasing and 'commutative', that is for any $f \in \mathbb{I}^{\mathcal{A}}$, $f' \in \mathbb{I}^{\mathcal{A}'}$, with $\mathcal{A}$, $\mathcal{A}'$, both sets of length n, it holds

$$\Big(\exists\, \phi : \mathcal{A} \to \mathcal{A}' \text{ bijective, with } \forall A \in \mathcal{A} : f(A) \geq f'(\phi(A))\Big) \Rightarrow Ada^{(n)}(f) \geq Ada^{(n)}(f')$$

and

$$\Big(\exists\, \phi : \mathcal{A} \to \mathcal{A}' \text{ bijective, with } \forall A \in \mathcal{A} : f(A) = f'(\phi(A))\Big) \Rightarrow Ada^{(n)}(f) = Ada^{(n)}(f') \,.$$

(2) Ada satisfies the boundary condition analogue to Definition 1.

Extended adapted aggregation functions are defined analogously to Definition 2. These definitions lead to similar functions. The major difference is that projections are aggregation functions but not adapted aggregation functions.

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. 1. The adapted mean

$$M^{(n)} : \bigcup_{\mathcal{A}:|\mathcal{A}|=n} [a,b]^{\mathcal{A}} \to [a,b]$$
$$f \mapsto \frac{1}{n}\sum_{A\in\mathcal{A}} f(A)\ ,$$

is also an adapted aggregation function for any $n \in \mathbb{N}^+$ and any interval $\mathbb{I} = [a,b]$, $a < b$.
To show $M^{(n)}$ is nondecreasing, let $f \in [a,b]^{\mathcal{A}}$, $g \in [a,b]^{\mathcal{A}'}$, with $|\mathcal{A}| = |\mathcal{A}'| = n$. Further let $\phi : \mathcal{A} \to \mathcal{A}'$ be a bijection such that $f(A) \geq g(\phi(A))$, for every $A \in \mathcal{A}$. Then we have

$$M^{(n)}(f) := \frac{1}{n}\sum_{A\in\mathcal{A}} f(A) \geq \frac{1}{n}\sum_{A\in\mathcal{A}} g(\phi(A)) = \frac{1}{n}\sum_{A'\in\mathcal{A}'} g(A') = M^{(n)}(g)\ .$$

For $f(A) = g(\phi(A))$, for every $A \in \mathcal{A}$, it follows $M^{(n)}(f) = M^{(n)}(g)$ immediately.
For the boundary condition consider an arbitrary $f : \mathcal{A} \to [a,b]$ in the domain. Then we know, since $|\mathcal{A}| = n$, we have

$$M^{(n)}(f) = \frac{1}{n}\sum_{A\in\mathcal{A}} f(A) \geq \frac{1}{n}\sum_{A\in\mathcal{A}} a = a = \inf[a,b]\ ,$$

thus $\inf_{f\in\mathcal{D}}\left(M^{(n)}(f)\right) \geq \inf[a,b]$. As $f : \mathcal{A} \to [a,b]$, $|\mathcal{A}| = n$ with $f(A) = a$, for every $A \in \mathcal{A}$, is an element of the domain with $M^{(n)}(f) = a$ it holds $\inf_{f\in\mathcal{D}}\left(M^{(n)}(f)\right) \geq \inf[a,b]$. The argumentation for the supremum is similar.
The function obtained by extending the domain of $M^{(n)}$ to $\mathcal{A}$ of arbitray finite length, is an extended adapted aggregation function. The argumentation is similar to Example 2.

2. The sum on the natural numbers $S^{(n)} : \bigcup_{\mathcal{A}:|\mathcal{A}|=n} \mathbb{N}^{\mathcal{A}} \to \mathbb{N}$ is an adapted aggregation function. The codomain $\mathbb{N}$ is ensured by the fact that $(\mathbb{N}, +)$ is a monoid, hence $\mathbb{N}$ is closed under addition. Further, $S^{(n)}$ is nondecreasing and commutative. This can be shown by proceeding in the same way as we did for the mean in (1.), since the factor $1/n$ and the domain do not effect the (in-)equalities.
The infimum of $S^{(n)}$ has to be greater than or equal to 0, as the codomain being $\mathbb{N}$ is already shown. Now consider $f : \mathcal{A} \to \mathbb{N}$, $|\mathcal{A}| = n$ with $f(A) = 0$ for every $A \in \mathcal{A}$, then f is in the domain and $S^{(n)}(f) = 0$. Thus the infimum property holds. For the supremum, assume $\sup_{f\in\mathcal{D}^{(n)}}\left(S^{(n)}(f)\right) = x$, for any $x \in \mathbb{R}$. Then let $\mathcal{A}$ be any set of size $n \in \mathbb{N}^+$. We construct a function $g : \mathcal{A} \to \mathbb{N}$ such that $g(A) = \lceil x \rceil + 1 \in \mathbb{N}$ for one $A \in \mathcal{A}$ and $g(B) = 0$ for every other $B \in \mathcal{A}$. Then $S^{(n)}(g) = \lceil x \rceil + 1 > x$. This contradicts the assumption that x was the supremum, hence $\sup_{f\in\mathcal{D}^{(n)}}\left(S^{(n)}(f)\right) = \infty = \sup\mathbb{N}$. Therefore, the sum $S^{(n)}$ of length n is an adapted aggregation function on $\mathbb{N}$, for any $n \in \mathbb{N}^+$ and thus also an extended adapted aggregation function.

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 aggregate sum is in the focus of this thesis as it is the most important one for the intended application, as explained below. Other aggregates such as minimum and maximum will be mentioned in Chapter 7.

2.2 Smart Contracts and Deductive Verification

In the last years, blockchain technology evolved drastically. A blockchain literally is a chain of blocks, similar to a linked list that is stored distributedly. Due to this and its architecture which uses hash functions and other cryptographic means, they are almost tamper-proof. These two reasons are why they are a good choice when trying to replace a trustworthy third party for example in financial affairs. Smart contracts are now decentralized computer programs executed on a blockchain-based system, as explained in [3]. Among other tasks, smart contracts automate financial transactions. That is, a smart contract is a contract written in a programming language called Solidity [4], performs transactions involving digital assets automatically, when the preconditions are met, and it is deployed on a blockchain and thus cannot be modified at a later point.

Since Solidity is a very expressive programming language, similar to Java Script, it requires software delevoplers with expertise to write secure and bug-free programs. This becomes even more crucial, because these programs will deal with tremendous amounts of money and cannot be fixed once deployed.
There were quite many cases were adversarial parties exploited bugs to increase their personal wealth as discussed in [5]. As these examples show, thorough reviews and software tests are not sufficient to find such vulnerabilites.
The lack of guaranteed error-free software is the main blocker of worldwide everyday use of smart contracts. Thus, the idea of new research projects is to find a way to only allow provably bug-free software to be deployed on blockchains. This is only possible by applying formal methods, as they can ensure correctness of programs in a mathematical gap-free way. Smart contracts are a good starting point for such verification efforts, because of the following reasons.

1. As mentioned before, the code is immutable and hence it is not possible to fix incorrect transactions.
2. It ensures fortunes not to get lost which is a strong motivator for the smart contracts society to support such efforts.
3. Smart contracts constist of rather small pieces of code compared to other industrial areas.

An overview of what has been done so far in terms of static analysis is provided in [6]. Further, one complete small-step semantics for Solidity-like languages is presented in [7].

The idea of using formal verification, more precisely deductive verification, is based on the following principle.

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 change of state a computer performs when executing a line of code is called transition. C.A.R Hoare was the first one to formalize this in 1969 by defining semantics of programs in [8]. He also described what happens to each of the variables when one command is executed depending on the inital value of the variable. That is, given a state that satisfies the precondition, then modify the state according to a programming command, which postconditions hold. This means he also considered what can be said about this new state. Such programming commands are for example assigning a value to a variable, entering a loop or checking the condition of an if-then-else block.

For the application of verifying programs the precondition of the first line of code specifies what kind of input is expected, whereas the postcondition of the last line states the expected output. Then, there has to be found preconditions and postconditions for each of the intermediate steps such that the postcondition of line n implies the precondition of line $n+1$. This task is particularly hard for loops, as they require loop invariants that are satisfied before and after performing transitions. Details on deductive verification can be found in [9].

However, today the challenges are to implement Hoare's ideas, automate the process of finding invariants and prove implications of post- and pre-conditions using automated theorem provers still remain. The formal verification approach we aim for in this thesis is now combining automated theorem provers, as explained in Section 2.3, and Hoare's pioneering work for automated deductive verification.

As mentioned before, finding invariants is one of the hardest tasks. Thus, having non-trivial generic invariants that always apply is a promising approach. It turns out that properties about aggegates of the parties' balances are good candidates. As such, especially the sum is very interesting. For most of the attacks that caused major damage the sum of the balances did not behave as it would have in a bug-free version. Hence, having the correct amount of total money as invariant would have found the vulnerabilities and could have prevented attacks.

Since the main actions in smart contracts are transferring money, modelling inflation by minting money or reducing money by claiming transaction fees, they are the most important transitions to preserve the invariants.

Hence, it is desired to find a way to encode properties of sums of arbitrary but finite length and to find invariants that ensure the correct impact of the sum when performing the transitions mentioned above.

2.3 Automated Theorem Proving in First-Order Logic

We use first-order logic with equality and sorts. Basic familiarity with first-order logic is assumed. Equality, denoted as '=', is an equivalence relation for which additionally

TU 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.

$x = y \rightarrow f(x) = f(y)$, for functions f, and $x = y \rightarrow \big(r(x) \leftrightarrow r(y)\big)$, for relations r, hold, for any x, y. Sorts are used to differentiate elements that have certain roles. That is, if someone wants to formalize relations or laws between different types of 'things', such as for example addresses and balances, then addresses can be modeled as a sort `Address` and balances can be modeled as a different sort `Int`. Semantically, having two elements x, y of different sorts means they cannot be the same, hence $x \neq y$.

Automated theorem provers in first-order logic receive a set of first-order input formulas and check whether the conjunction of these formulas is satisfiable. As every theorem can be equivalently reformulated to an unsatifiable set of formulas, proving a theorem is done by proving unsatisfiability. There are many different approaches on how to do this, but each of them uses a set of sound inference rules, called inference system. By applying an inference rule to some of the input formulas another first-order formula is obtained. This new formula is entailed by the input formulas. Thus, the goal is to reach the empty clause $\square$, also called `false` or bottom $\bot$. This formula is always unsatisfiable, since it is false independent of the interpretation.

The automated theorem provers used in this work are Vampire [10] and Z3 [11]. We will also refer to them as 'solvers' and 'provers'.

The first-order theorem prover Vampire uses a superposition calculus to saturate the set in input clauses up to redundancy, according to certain rules (selection function) [12], [13]. That means, the 'difficult' clauses that are implied by 'easier' ones are called redundant and are not considered for saturation. Satisfiability checking in first-order logic is undecidable, thus Vampire does not always yield '`sat`'of '`unsat`' but it may also yield '`unknown`' for example in case of time out. However, in practice it works well.

The Z3 theorem prover, is an SMT solver, where SMT stands for Satisfiability Modulo Theories [14]. We say a (usually quantifier-free) formula ϕ is satisfiable modulo a theory $\mathcal{T}$, if there exists an interpretation I that makes all theory axioms valid and satisfies ϕ. The theory $\mathcal{T}$ is defined by a signature and a set of axioms. That means when considering SMT, we are only interested in interpretations of a specific form. Well-known theories are for example the theory of linear integer arithmetic, the theory of equality or the theory of uninterpreted function symbols. SMT solvers implement specializied decision procedures for (the quantifier-free fragment of) some theories. The combination of the three theories mentioned above is called UFLIA and its quantifier-free fragment is decidable. Z3 has a decision procedure for this class of problems. However, in this work the full UFLIA theory is needed, which is undecidable. Thus, it may also occur that Z3 times out when trying to prove sets of formulas presented in this work.

Technically, the expectation is that Vampire deals with quantifiers better, whereas Z3 is better in tasks having more involved arithmetics. As the experimental results using theorem provers in this work may depend on this fact, both solvers are considered.

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.
TU WIEN Bibliothek Your knowledge hub

3 Encoding Finite Sums of Arbitrary Length

The work in this thesis is motivated by rigorous reasoning about smart contracts. Also, the names of the functions, sorts etc are oriented towards this one use case. However, all the theoretical and experimental results presented within this work can be applied to any setting where sums of non-negative integer arrays are compared. The work in this chapter is thus **not restricted to smart contracts**, but only explained and detailed for this use case.

In this chapter an approach to reason about finite sums of arbitrary length is presented by addressing the following two challenges.

[Ch1] Find a way to encode the higher-order term $\sum_{i\in\mathcal{I}} i$ in first-order logic, where $\mathcal{I} \subset \mathbb{N}$ is an arbitrary finite set.

[Ch2] Design the encoding in such a way that automated theorem provers can reason about it. In particular, given the encodings of $\mathcal{I}, \mathcal{J} \subset \mathbb{N}$, both finite, and specifying how $\mathcal{I}$ and $\mathcal{J}$ (and thus their encodings) are related, the solvers should be able to prove that their sums are related accordingly.

Setting to address challenge [Ch1].

The basis of the encoding (Sections 3.1, 3.2) which is tailored to fit the smart contract's constrains, is the following.

> *It is assumed that there exist finitely, but unboundedly, many addresses. The set of addresses is called $\mathcal{A}$. They can be imagined to be all the bank accounts in the world. Even though their number is arbitrary, within one encoding this number is fixed and will thus not change. That is, it is not possible to open or close bank accounts.*
>
> *Additionally, every address has a balance. The balances are non-negative integers. This means, it is not possible to owe some address money.*
>
> *We are now looking for a way to capture the 'total money in the world', that is the sum of the balances of all addresses. In order to do so, only one unique 'currency' is considered. The sum does not necessarily have to be available in an explicit way. However, we want to be able to compare it to another world.*

In this chapter, it is assumed that the other world is based on the previous one, but one time step ahead. This is why, it is referred to them as 'old-world' and 'new-world'. Typically, they differ by exactly one smart contract's transition, as explained in Section 2.2. The transactions considered in the Section 3.2 are transferring, minting and burning one unit of money. One unit of money is called a coin.
Finally, the relation between the 'old total money' and the 'new total money' is of interest.

The aim, for verifying smart contracts, is to make sure that while performing one transition no coin (dis-)appears accidentally.

Evaluation for challenge [Ch2].

The performance with respect to the second challenge [Ch2] is addressed in Section 3.3. There, the results of what the solvers Vampire (cite) and Z3 (cite) could (not) prove are presented. The results are relative to the encoding, the transition and the property to be ensured. For the cases, in which the solvers cannot prove the desired property, an intuitive explanation of failure is given.

3.1 Encoding of Sum Properties in First-Order Logic

In this section a very promising encoding of sums developed in this work is presented and explained.

The encoding is a set of first-order formulas with equality and multiple sorts, as defined in Section 2.3. The sorts used are called `Coin` and `Address`. The `Coin` sort is representing the money and one coin is one unit of money. The set of coins $\mathcal{C}$ is unbounded but countable. The sort `Address` is representing the bank accounts. The set of addresses $\mathcal{A}$ is finite. It is `Coins` $\equiv \mathcal{C}$ and `Address` $\equiv \mathcal{A}$ and the notations will be used interchangeably. For practical uses by automated provers, $\mathcal{C}$ and $\mathcal{A}$ are just arbitrary disjoint sets.

The main idea of the encoding is to assign every coin c, that has the predicate `active`, to precisely one address A. This means that address A owns that coin c. This is expressed through the binary `has-coin` predicate. Additionally, it shall also be the case that if an address A has the coin c, then c is `active`.

Figure 3.1 illustrates our setting. On a high level, there are two axes (left-right and up-down) and the picture is 'symmetric' in both ways. The left-right axis can be understood as the time. As explained in the introductory paragraph of this chapter, there is the 'old-world' on the left and the 'new-world' on the right. They are connected with the specification of one smart contract's transition and its expected impact on the total amount of money (Section 3.2). For the up-down axis, we have the 'macro-world' and the 'micro-world'. The total amount of money, that is the sum of balances, is handled in the 'macro-world', whereas the amount of money in each account is handled in the 'micro-world'.
In the higher-order setting, the micro-world consists of the balance function:

$$\texttt{bal} : \mathcal{A} \to \mathbb{N}\,,$$

which assigns to every address A the number of coins $\texttt{bal}(A)$ they own, that is its balance. In this encoding, the balance function only exists implicitly. Instead, there is the binary predicate $\texttt{has-coin} \subseteq \mathcal{A} \times \mathcal{C}$. It holds $(A, c) \in \texttt{has-coin}$, if the coin c is owned by address A. By doing so, the computationally complex sort `Integer` can be avoided. This advantage comes at the price of not having the balance per se at hand. The dark blue dashed line between `bal` and `has-coin` in Figure 3.1 represents the implicit connection of the encoding

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

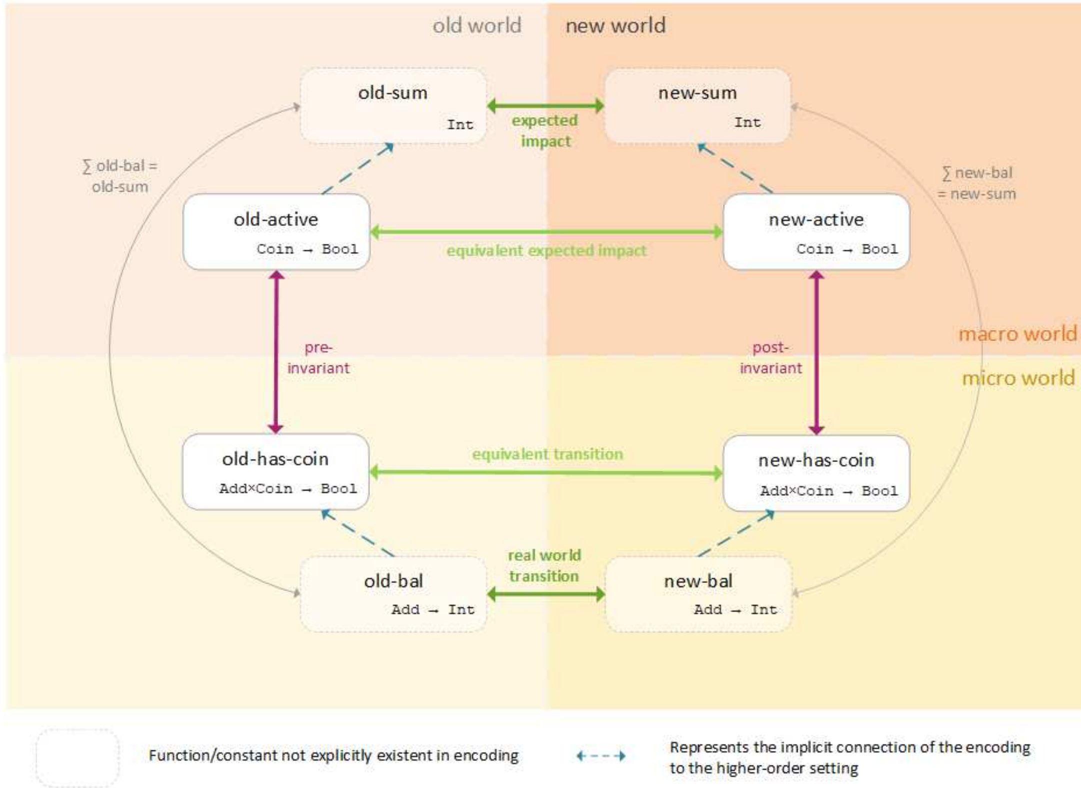

Figure 3.1: Representation of the sums encoding and the relation between the higher-order setting ('outer circle') and the first-order encoding ('inner circle').

to the real world. This connection will be formalized in Chapter 4.

For the macro-world the situation is similar. In the higher-order setting, there is the non-negative integer `sum` which is the total number of coins c owned by some address A, hence $\texttt{sum} = \sum_{A \in \mathcal{A}} \texttt{bal}(A)$. In this encoding a unary predicate $\texttt{active} \subseteq \mathcal{C}$ is used to capture all the coins c that are owned by some address A.
As a consequence, the relation between the micro-world and the macro-world, $\texttt{sum} = \sum_{A \in \mathcal{A}} \texttt{bal}(A)$ as depicted in Figure 3.1 with the grey curved lines, cannot be expressed. Instead, there has to be a relation between `active` and `has-coin` that ensures this very property. This relation is illustrated by the purple arrows in the figure. While the relations between the old and the new world highly depend on the transition, the relation between the macro and the micro world remains the same. Hence, in application of software verification they can be used as invariants. This means that no matter what sequence of actions are taken, it can be guaranteed that no coin is lost or won accidentally, provided the invariants hold. The invariants are formalized in the following way.

Definition 4 (Invariants)**.** A pair (`has-coin`, `active`), where $\texttt{has-coin} \subseteq \mathcal{A} \times \mathcal{C}$ and $\texttt{active} \subseteq \mathcal{C}$, *satisfies the invariants*, if and only if it satisfies the following three formulas.

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) Only active coins c can be in an address A, that is:

$$\forall c \in \mathcal{C} : \exists A \in \mathcal{A} : \texttt{has-coin}(A, c) \rightarrow \texttt{active}(c) \ . \qquad \textbf{(Inactive-Coins)}$$

ii) Every active coin c is in some address A, expressed as:

$$\forall c \in \mathcal{C} : \texttt{active}(c) \rightarrow \exists A \in \mathcal{A} : \texttt{has-coin}(A, c) \ . \qquad \textbf{(At-Least-One-Address)}$$

iii) Every coin c is in at most one address A, that is:

$$\forall c \in \mathcal{C},\ A, B \in \mathcal{A} : (\texttt{has-coin}(A, c) \land \texttt{has-coin}(B, c) \rightarrow A = B) \ . \qquad \textbf{(At-Most-One-Address)}$$

We write inv(has-coin, active) for

$$\textbf{Inactive-Coins} \land \textbf{At-Least-One-Address} \land \textbf{At-Most-One-Address} \ .$$

Theorem 1. *Given a balance function* $\mathtt{bal}$*, a non-negative integer* $\mathtt{sum}$*, a unary predicate* $\mathtt{active}$ *and a binary predicate* $\mathtt{has\text{-}coin}$*, as defined above, such that:*

$$|\mathtt{active}| = \mathtt{sum}$$

and for every address A*:*

$$\mathtt{bal}(A) = |\{c \in \mathcal{C} : (A, c) \in \mathtt{has\text{-}coin}\}| \ ,$$

then $\sum_{A \in \mathcal{A}} \mathtt{bal}(A) = \mathtt{sum}$*, if* $(\mathtt{has\text{-}coin}, \mathtt{active})$ *satisfies the invariants.*

Note that this claim precisely states the soundness of the encoding. However, the notion of completeness requires more involved formalization. This soundness claim, and thus Theorem 1, will be proven together with a completeness claim in Chapter 4.

3.2 Transitions and Impact on Sums

This section focusses on the relation between the old and the new world, that is the transition and its expected impact represented by both the lime and the dark green arrows in Figure 3.1. As mentioned before, these links illustrate how the world changes in one time step, where one time step is defined by taking one of the transactions listed.

For the higher-order setting (dark green arrows), this means they describe how the balances in the new world evolved from the old ones and what the expected impact on the total money is:
If the balance function `bal` changes in a certain way (transition) and the value `sum` changes

TU 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.

according to the expected impact of the transition on the total amount of money, then we can verify no coin (dis-)appeared accidentally. That means $\sum_{A\in\mathcal{A}} \texttt{bal}(A) = \texttt{sum}$ is still the case.

The lime arrows exist as the dark greens' equivalent first-order versions. They describe how the `active` and the `has-coin` predicate in the new world evolved from the old ones.

The **precise explanation** of the above is stated below.

- What we wish to reason about (higher-order setting):
  Provided $\sum_{A\in\mathcal{A}} \texttt{old-bal}(A) = \texttt{old-sum}$ and ($\texttt{old-bal} \mapsto \texttt{new-bal}$) then ($\texttt{old-sum} \mapsto \texttt{new-sum} \Leftrightarrow \sum_{A\in\mathcal{A}} \texttt{new-bal}(A) = \texttt{new-sum}$) .

- What we are reasoning about instead (first-order encoding):
  Provided (`old-has-coin`, `old-active`) satisfies the invariants and

  $$(\texttt{old-has-coin} \mapsto \texttt{new-has-coin}) \,,$$

  then

  $$(\texttt{old-active} \mapsto \texttt{new-active} \;\Leftrightarrow\; (\texttt{new-has-coin}, \texttt{new-active})$$

  satisfies the invariants).

Note that for the motivation of software verification only the direction '⇒' is of interest, since the goal of the proof is independent from the transition which is crucial for invariants. The nature of an invariant is to remain unchanged while ensuring the correctness of a piece of code, as explained in Section 2.2. However, the other direction '⇐' is addressed in Chapter 5.

Considered Transitions

In this work the three most important actions for smart contracts are considered as transitions. They are transferring money from an address A to another one B, 'minting' money, which is used to model inflation, as described in Section 2.2, and burning money. For simplicity and due to the fact that only one direction of the equivalence above is of interest, the transition and its expected impact on the macro world are defined as one entity.

Transition `transfer`(A_1,A_2)

The first transition considered is transferring money. Mathematically speaking, transferring a coin from address A_1 to address A_2 means decreasing A_1's balance by one and increasing A_2's balance by one. Every other value stays unchanged. The expected impact

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

on the total amount of money is that it remains unchanged.

Remark. In this work it is not possible to have negative balance for any address. Hence, before taking the transition, it has to be ensured that address A_1 has enough money. Here, this issue is solved by not transferring money at all in this case. For software verification, there has to be found a way to address this problem, depending on how the software should behave and what other conditions and properties the input has. E.g. the initial state may fulfill certain properties, such as 'Every address has at least n coins.', which can be used to deduce that the case of not having enough coins cannot happen.

The transition is encoded by formulating formulas that constrain the new world, given the old world. For $\texttt{transfer}(A_1,A_2)$ there are four formulas. The first one:

$$A_1 \neq A_2 \ , \tag{T1}$$

is ensuring that the orderer is distinct from the beneficiary. Secondly, in case A_1 does not have any coins, nothing changes:

$$\begin{aligned}&(\forall c \in \mathcal{C} : \neg\texttt{old-has-coin}(A_1, c) \vee \neg\texttt{old-active}(c))\\ \rightarrow \quad &(\forall c \in \mathcal{C}, A \in \mathcal{A} : \texttt{new-has-coin}(A, c) \leftrightarrow \texttt{old-has-coin}(A, c)) \ .\end{aligned} \tag{T2}$$

Finally, if A_1 has a coin, then one coin c that belonged to A_1 before moves to A_2. Every other coin-address pair (B, d) remains unchanged. 'Every other' here means $d \neq c \vee (B \neq A_1 \wedge B \neq A_2)$. The formula is:

$$\begin{aligned}&\exists c \in \mathcal{C} : \texttt{old-has-coin}(A_1, c) \wedge \texttt{old-active}(c)\\ \rightarrow \quad &\exists c \in \mathcal{C} :\\ &\qquad\qquad \texttt{old-has-coin}(A_1, c) \wedge \neg\texttt{new-has-coin}(A_1, c)\\ &\qquad \wedge \quad \neg\texttt{old-has-coin}(A_2, c) \wedge \texttt{new-has-coin}(A_2, c)\\ &\qquad \wedge \quad \forall B \in \mathcal{A}, d \in \mathcal{C} :\\ &\qquad\qquad\quad d \neq c \vee (B \neq A_1 \wedge B \neq A_2)\\ &\qquad\qquad\quad \rightarrow \quad (\texttt{old-has-coin}(B, d) \leftrightarrow \texttt{new-has-coin}(B, d) \ .\end{aligned} \tag{T3}$$

The expectation is, that no coin changes its activity status, this means

$$\forall c \in \mathcal{C} : \texttt{old-active}(c) \leftrightarrow \texttt{new-active}(c) \ . \tag{T4}$$

The transition $\texttt{transfer}(A_1, A_2)$ is thus defined as T1 $\wedge$ T2 $\wedge$ T3 $\wedge$ T4 and we have the following result.

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.

Lemma 1 (Soundness of $\texttt{transfer}(A_1, A_2)$).
Given the balance functions $\textit{old-bal}, \textit{new-bal} : \mathcal{A} \to \mathbb{N}$, the natural numbers $\textit{old-sum}$, $\textit{new-sum}$, unary predicates $\textit{old-active}, \textit{new-active} \subseteq \mathcal{C}$ and binary predicates $\textit{old-has-coin}, \textit{new-has-coin} \subseteq \mathcal{A} \times \mathcal{C}$.
As in Theorem 1, let

$$|\textit{old-active}| = \textit{oldsum}, \ |\textit{new-active}| = \textit{newsum} \tag{1}$$

and for every address A

$$\begin{aligned} \textit{old-bal}(A) &= |\{c \in \mathcal{C} : (A, c) \in \textit{old-has-coin}\}| \ , \\ \textit{new-bal}(A) &= |\{c \in \mathcal{C} : (A, c) \in \textit{new-has-coin}\}| \ . \end{aligned} \tag{2}$$

Additionally, let $A_1, A_2 \in \mathcal{A}$ and $\textit{old-bal}(A_1) > 0$ and $\textit{transfer}(A_1, A_2)$.
Then, $\textit{new-bal}(A_1) = \textit{old-bal}(A_1) - 1$, $\textit{new-bal}(A_2) = \textit{old-bal}(A_2) + 1$ and for all other addresses $B \in \mathcal{A}$, $B \notin \{A_1, A_2\}$ it holds $\textit{new-bal}(B) = \textit{old-bal}(B)$. Also $\textit{old-sum} = \textit{new-sum}$.

Proof. Firstly, show $\texttt{old-sum} = \texttt{new-sum}$. Because of (T4), we know that $\texttt{old-active} = \texttt{new-active}$, hence $\texttt{old-sum} = |\texttt{old-active}| = |\texttt{new-active}| = \texttt{new-sum}$ by using (1).

The proof of $B \in \mathcal{A}$, $B \notin \{A_1, A_2\}$ implies $\texttt{new-bal}(B) = \texttt{old-bal}(B)$ works similarly. Since $\texttt{old-bal}(A_1) > 0$, it is the case that $\exists c \in \mathcal{C} : \texttt{old-has-coin}(A_1, c)$. Hence the right hand side of (T3) holds as well. Especially the last subformula of the conjunction. For $B \notin \{A_1, A_2\}$ it is also true that $d \neq c \vee (B \neq A_1 \wedge B \neq A_2)$, thus we know $\{d \in \mathcal{C} : \texttt{old-has-coin}(B, d)\} = \{d \in \mathcal{C} : \texttt{new-has-coin}(B, d)\}$. And therefore,

$$\begin{aligned} \texttt{old-bal}(B) &= |\{d \in \mathcal{C} : \texttt{old-has-coin}(B, d)\}| \\ &= |\{d \in \mathcal{C} : \texttt{new-has-coin}(B, d)\}| = \texttt{new-bal}(B) \ , \end{aligned}$$

by using (2).

It remains to show that both $\texttt{new-bal}(A_1) = \texttt{old-bal}(A_1) - 1$ holds and also $\texttt{new-bal}(A_2) = \texttt{old-bal}(A_2) + 1$ holds. As argued above, the right hand side of (T3) holds. As a consequence there is a coin c such that both

$$c \in \texttt{old-has-coin}(A_1, .) \backslash \texttt{new-has-coin}(A_1, .)$$

and

$$c \in \texttt{new-has-coin}(A_2, .) \backslash \texttt{old-has-coin}(A_2, .) \ .$$

Now consider another coin $d \neq c$. For this coin especially $d \neq c \vee (B \neq A_1 \wedge B \neq A_2)$. Hence,

$$\{d \in \mathcal{C} \backslash \{c\} : \texttt{old-has-coin}(A_1, d)\} = \{d \in \mathcal{C} \backslash \{c\} : \texttt{new-has-coin}(A_1, d)\}$$

and

$$\{d \in \mathcal{C} \backslash \{c\} : \texttt{old-has-coin}(A_2, d)\} = \{d \in \mathcal{C} \backslash \{c\} : \texttt{new-has-coin}(A_2, d)\} \ .$$

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

Combining this, it follows

$$\{d \in \mathcal{C} : \texttt{old-has-coin}(A_1, d)\} \backslash \{c\} = \{d \in \mathcal{C} : \texttt{new-has-coin}(A_1, d)\}$$

and

$$\{d \in \mathcal{C} : \texttt{old-has-coin}(A_2, d)\} = \{d \in \mathcal{C} : \texttt{new-has-coin}(A_2, d)\} \backslash \{c\} \; .$$

Therefore, both

$$\begin{aligned} \texttt{old-bal}(A_1) - 1 &= |\{d \in \mathcal{C} : \texttt{old-has-coin}(A_1, d)\} \backslash \{c\}| \\ &= |\{d \in \mathcal{C} : \texttt{new-has-coin}(A_1, d)\}| = \texttt{new-bal}(A_1) \end{aligned}$$

and

$$\begin{aligned} \texttt{old-bal}(A_2) &= |\{d \in \mathcal{C} : \texttt{old-has-coin}(A_2, d)\}| \\ &= |\{d \in \mathcal{C} : \texttt{new-has-coin}(A_2, d)\} \backslash \{c\}| = \texttt{new-bal}(A_2) - 1 \; . \end{aligned}$$

This concludes the proof of this lemma. □

Transitions $\texttt{throw}(c, A)$ and $\texttt{catch}(c, A)$

In the case when the $\texttt{transfer}(A_1, A_2)$ action shall be considered as two subsequent steps, namely taking the coin c from A_1 and giving it to A_2, then also the transition has to be split up into a $\texttt{throw}(c, A_1)$ and a $\texttt{catch}(c, A_2)$ transition. Note that performing only one of them is not a valid transition and will result in the invariants being violated.

The first part, $\texttt{throw}(c,A)$ models the action of 'freeing' a coin without deleting it. The formulas defining it are as follows.
For A to 'free' c, it has to be the case that A had c. Hence:

$$\texttt{old-has-coin}(A, c) \land \forall B \in \mathcal{A} : \neg\texttt{new-has-coin}(B, c) \; . \tag{Th1}$$

'Freeing' a coin means it does not belong to A any longer, but it also does not belong to any other address. That is:

$$\begin{aligned} &\forall B \in \mathcal{A}, d \in \mathcal{C} : \\ &\quad d \neq c \lor B \neq A \\ &\quad \rightarrow \quad \big(\texttt{old-has-coin}(B, d) \leftrightarrow \texttt{new-has-coin}(B, d)\big) \; . \end{aligned} \tag{Th2}$$

Since coin c is not deleted only transferred, we do not want c to become inactive, this is why:

$$\texttt{old-active}(c) \land \texttt{new-active}(c) \; . \tag{Th3}$$

TU Bibliothek, WIEN 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.

The activity status of all the other coins also does not change:

$$\forall d \in \mathcal{C} : \texttt{old-active}(d) \leftrightarrow \texttt{new-active}(d) \; . \tag{Th4}$$

The transition $\texttt{throw}(c, A)$ is thus defined as Th1 $\wedge$ Th2 $\wedge$ Th3 $\wedge$ Th4.

Note that after performing `throw` the invariants are *not* satisfied. This is good, since the transferring is not complete yet and the program is not allowed to stop at this point.

It has to be considered that `throw` and `catch` require one time step each. Thus, the result of `throw` (`new-has-coin` and `new-active`) is the starting point of `catch` (`old-has-coin` and `old-active`). The formulas for A 'catching' the 'free' coin c are as follows.
The coin c did not belong to any address A and is now assigned to A:

$$\texttt{new-has-coin}(A, c) \;\wedge\; \forall B \in \mathcal{A} : \neg\texttt{old-has-coin}(B, c) \; . \tag{C1}$$

For all the other coins their assignment does not change:

$$\begin{aligned} &\forall d \in \mathcal{C}, B \in \mathcal{A} : \\ &\quad d \neq c \vee B \neq A \\ &\quad \rightarrow (\texttt{new-has-coin}(B, d) \leftrightarrow \texttt{old-has-coin}(B, d)) \; . \end{aligned} \tag{C2}$$

Since coin c is not minted, only transferred, we do not want c to have been inactive, this is why:

$$\texttt{old-active}(c) \wedge \texttt{new-active}(c) \; . \tag{C3}$$

All the other coins' activity status should not change:

$$\forall d \in \mathcal{C} : \texttt{old-active}(d) \leftrightarrow \texttt{new-active}(d) \; . \tag{C4}$$

With this, we define $\texttt{catch}(c, A)$ as C1 $\wedge$ C2 $\wedge$ C3 $\wedge$ C4.

Calling the intermediate state's predicates and functions `int-act`, `int-sum`, `int-has-coin` and `int-bal`, we get the following result.

Lemma 2 (Soundness of $\texttt{throw}(c, A_1)$, then $\texttt{catch}(c, A_2)$)**.**
Given the old, new and intermediate predicates and functions, as in Lemma 1. Additionally, let $A_1, A_2 \in \mathcal{A}$, $c \in \mathcal{C}$ and $\texttt{throw}(c, A_1)$ (with renaming the 'output' predicates from `new-`

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

to $\texttt{int-}$*) and* $\texttt{catch}(c, A_2)$ *(with renaming the 'input' predicates from* $\texttt{old-}$ *to* $\texttt{int-}$*).*
Then, $\texttt{new-bal}(A_1) = \texttt{old-bal}(A_1) - 1$*,* $\texttt{new-bal}(A_2) = \texttt{old-bal}(A_2) + 1$ *and for all other addresses* $B \in \mathcal{A}$*,* $B \notin \{A_1, A_2\}$ *it holds* $\texttt{new-bal}(B) = \texttt{old-bal}(B)$*. Also* $\texttt{old-sum} = \texttt{new-sum}$*.*

The proof of Lemma 2 is very similar to the proof of Lemma 1 and will therefore be skipped.

Transition $\texttt{mint}(c, A)$

The next transition $\texttt{mint}$ expresses that new, previously inexistent, money is created, that is minted, and that one address specified will own the new money. All the other balances stay the same. The total amount of money should increase by one, when one coin c is minted. The transition $\texttt{mint}(c, A)$ activates the coin c and A will own c. More precisely, this means the following.

The coin c has to be inactive before and is activated now:

$$\neg\texttt{old-active}(c) \wedge \texttt{new-active}(c) \tag{M1}$$

and the address A owns the new coin c:

$$\texttt{new-has-coin}(A, c) \wedge \forall B \in \mathcal{A} : \neg\texttt{old-has-coin}(B, c)\ . \tag{M2}$$

Everything else stays the same, hence:

$$\begin{aligned} &\forall d \in \mathcal{C}, B \in \mathcal{A} : \\ &\quad (d \neq c \vee B \neq A) \rightarrow \\ &\quad (\texttt{new-active}(d) \leftrightarrow \texttt{old-active}(d)) \wedge (\texttt{new-has-coin}(B, d) \leftrightarrow \texttt{old-has-coin}(B, d)) \end{aligned} \tag{M3}$$

The transition $\texttt{mint}(c, A)$ is now defined as M1 $\wedge$ M2 $\wedge$ M3.

Lemma 3 (Soundness of $\texttt{mint}(c, A)$)**.**
Given the balance functions $\texttt{old-bal}$*,* $\texttt{new-bal} : \mathcal{A} \rightarrow \mathbb{N}$*, the Natural Numbers* $\texttt{old-sum}$*,* $\texttt{new-sum}$*, unary predicates* $\texttt{old-active}$*,* $\texttt{new-active} \subseteq \mathcal{C}$ *and binary predicates* $\texttt{old-has-coin}$*,* $\texttt{new-has-coin} \subseteq \mathcal{A} \times \mathcal{C}$ *as in Lemma 1.*
Additionally, let $A \in \mathcal{A}$*,* $c \in \mathcal{C}$ *and* $\texttt{mint}(c, A)$*.*
Then, $\texttt{new-bal}(A) = \texttt{old-bal}(A) + 1$ *and for all other addresses* $B \in \mathcal{A}$*,* $B \neq A$ *it holds* $\texttt{new-bal}(B) = \texttt{old-bal}(B)$*. Also* $\texttt{new-sum} = \texttt{old-sum} + 1$*.*

Again, the proof of Lemma 3 is similar to the one of Lemma 1 and will thus be skipped.

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

Transition $\texttt{burn}(c, A)$

The last transition considered handles the planned decrease of money, such as burning money. It is in a way the inverse of `mint`. This means we take an existing coin and label it inactive. The address that used to have this coin does not have it any more and therefore its balance decreased by one.
In order to strictly decrease the money, money had to be there in the first place. This is why, for the 'burn' transition, there is an axiom required. It is

$$\exists c \in \mathcal{C} : \texttt{old-active}(c) \; . \tag{There-are-Coins}$$

Remark. The property (**There-are-Coins**) is mentioned as an axiom here, because it is not really part of the transition. It does not describe the connection between the old and the new world.

The coin c to be burned has to be active before and not active afterwards, that is:

$$\texttt{old-active}(c) \wedge \neg\texttt{new-active}(c) \tag{B1}$$

and the address A that used to own c does not have it any more:

$$\texttt{old-has-coin}(A, c) \wedge \forall B \in \mathcal{A} : \neg\texttt{new-has-coin}(B, c) \; . \tag{B2}$$

Everything else stays the same, thus:

$$\begin{aligned} &\forall d \in \mathcal{C}, A \in \mathcal{A} : \\ &\quad d \neq c \vee B \neq A \\ &\quad (\texttt{new-active}(d) \leftrightarrow \texttt{old-active}(d)) \wedge (\texttt{new-has-coin}(B, d) \leftrightarrow \texttt{old-has-coin}(B, d)) \; . \end{aligned} \tag{B3}$$

The transition $\texttt{burn}(c, A)$ is now defined as B1 $\wedge$ B2 $\wedge$ B3.

Lemma 4 (Soundness of $\texttt{burn}(c, A)$)**.**
Given the balance functions $\texttt{old-bal}$, $\texttt{new-bal} : \mathcal{A} \to \mathbb{N}$, *the Natural Numbers* $\texttt{old-sum}$, $\texttt{new-sum}$, *unary predicates* $\texttt{old-active}$, $\texttt{new-active} \subseteq \mathcal{C}$ *and binary predicates* $\texttt{old-has-coin}$, $\texttt{new-has-coin} \subseteq \mathcal{A} \times \mathcal{C}$ *as in Lemma 1.*
Additionally, let $A \in \mathcal{A}$, $c \in \mathcal{C}$, $\texttt{old-sum} > 0$ *and* $\texttt{burn}(c, A)$.
Then, $\texttt{new-bal}(A) = \texttt{old-bal}(A) - 1$ *and for all other addresses* $B \in \mathcal{A}$, $B \neq A$ *it holds* $\texttt{new-bal}(B) = \texttt{old-bal}(B)$. *Also* $\texttt{new-sum} = \texttt{old-sum} - 1$.

Note that $\texttt{old-sum} > 0$ is equivalent to (**There-are-Coins**). The proof of Lemma 4 is also similar to the one of Lemma 1 and will thus be skipped.

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

Other Proof Goals

So far, we were interested to conclude that $\sum_{A \in \mathcal{A}} \texttt{bal}(A) = \texttt{sum}$. The encoding was designed in such a way that this was the main focus. However, it is also possible to handle properties like $\sum_{A \in \mathcal{A}} \texttt{bal}(A) \leq \texttt{sum}$ and $\sum_{A \in \mathcal{A}} \texttt{bal}(A) \geq \texttt{sum}$.
In order to do so, the invariants have to be weakened. For the first property $\sum_{A \in \mathcal{A}} \texttt{bal}(A) \leq \texttt{sum}$ the invariant to be dropped is (**At-Least-One-Address**). This means it is possible for an active coin to not be owned by any address, but is still necessary that only active coins can be owned and that a coin cannot be owned by more than one address.

For $\sum_{A \in \mathcal{A}} \texttt{bal}(A) \geq \texttt{sum}$, the (**Inactive-Coins**) invariant has to be dropped. Hence, there it is possible that inactive coins are owned by at most one address, as long as every active coin is also owned by precisely one address.
Unfortunately, the nice intuition with exactly the active coins to be physically located at some address does not apply any more. The most convenient part is though, that all the transitions can be taken over identically from the initial $\sum_{A \in \mathcal{A}} \texttt{bal}(A) = \texttt{sum}$ setting.

A probably more interesting property is to make sure that the total money in the world does not drop below a certain *constant* number t. There are only a few adaptions to be made in order to use the encoding presented as follows.

1. Both `old-sum` and `new-sum` are replaced by t, since we want to obtain both $\sum_{A \in \mathcal{A}} \texttt{old-bal}(A) \geq t$ and $\sum_{A \in \mathcal{A}} \texttt{new-bal}(A) \geq t$.
2. Likewise, `old-active` and `new-active` are replaced by one mutual `active`.
3. This means there is no update of `active` in transitions. However, constraints about coins' activity status can still be expressed.
4. The (**Inactive-Coins**) invariant has to be removed.
5. The transition `burn` may be problematic. One can add a constraint that there has to exist an inactive coin c and an address A, such that address A had it before performing the action.

This means to make sure that the total money stays greater equal $t = |\texttt{active}|$, we have to show that for the initial set of active coins every element is still owned by some address.

In order to reason about $\sum_{A \in \mathcal{A}} \texttt{old-bal}(A) \leq t$ and $\sum_{A \in \mathcal{A}} \texttt{new-bal}(A) \leq t$ the equivalent adaptions of the encoding have to be made. Especially, the invariant to be dropped is (**At-Least-One-Address**).

3.3 Experimental Results of the Sum Encoding

In this section the performance of the encoding regarding challenge [Ch2] is presented. The specification of [Ch2] to our precise setting is stated subsequently.

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

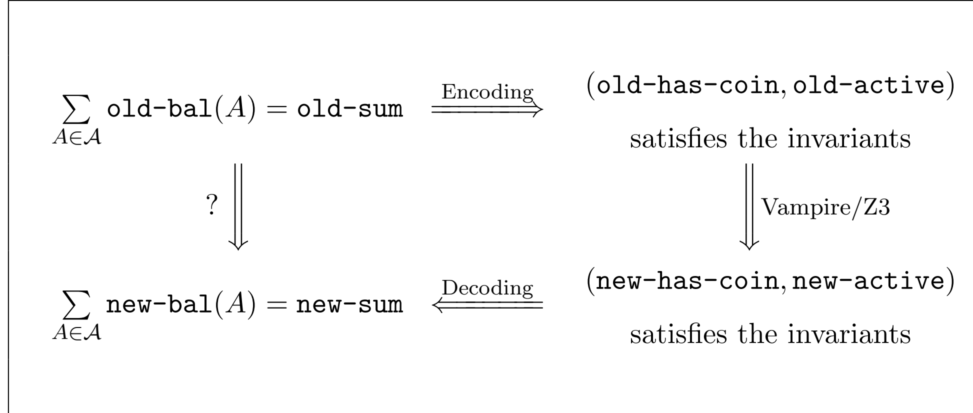

Table 3.1: Overview of which part is considered by theorem provers.

[Ch2'] Given encodings (`old-has-coin`, `old-active`) and (`new-has-coin`, `new-active`) of `old-bal` and `new-bal`, respectively. Assume that the transitions between `old-bal` and `new-bal` (and hence also (`old-has-coin`, `old-active`) and (`new-has-coin`, `new-active`)) are known. Then the solvers Vampire and Z3 can prove that the sums over `old-bal` and `new-bal` are related accordingly.

In Table 3.1 the experimental setting for our main proof goal is illustrated. The question mark represents the reasoning goal, which is reached by the detour of encoding, using an automated theorem prover - such as Vampire or Z3 - and decoding.
The task of proving has been mentioned briefly in the beginning of Section 3.2. Namely, given that

1. (`old-has-coin`, `old-active`) satisfies the invariants and
2. a transition to obtain (`new-has-coin`, `new-active`),

then prove that also (`new-has-coin`, `new-active`) satisfies the invariants.
Equivalently, the goal to prove is that the conjunction

$$\begin{aligned} & \texttt{inv}(\texttt{old-has-coin}, \texttt{old-active}) \\ \wedge\quad & \texttt{transfer}(A_1, A_2) \\ \wedge\quad & \neg\texttt{inv}(\texttt{old-has-coin}, \texttt{old-active}) \end{aligned} \qquad \text{(Goal)}$$

is unsatisfiable.

For the other proof goals mentioned in Section 3.2, the encoding is similar to (Goal). In Table 3.2, the main property to be proven is referred to as $\sum_{A\in\mathcal{A}} \texttt{bal}(A) = \texttt{sum}$, and the others as $\sum_{A\in\mathcal{A}} \texttt{bal}(A) \geq \texttt{sum}$ and $\sum_{A\in\mathcal{A}} \texttt{bal}(A) \geq t$. For every property, the four transitions presented in Section 3.2 are considered. For $\sum_{A\in\mathcal{A}} \texttt{bal}(A) = \texttt{sum}$, there 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

two additional transitions listed to give an example of possible extensions of the encoding. They are called 'mintN' and 'mintN-helper'. They encode the transaction of transferring N coins, for an arbitrary unfixed $N \in \mathbb{N}^+$. This is done by replacing the 'old- ' and 'new-' prefixes by an additional input value (integer), similarly to [27].
The transitions 'mintN' and 'mintN-helper' only differ by a helping lemma added in 'mintN-helper'. It states the instance of the induction principle for this precise application.

Property	Transition	Z3	Vampire default	Vampire casc	Vampire avatar off
$\sum_{A\in\mathcal{A}} \texttt{bal}(A) = \texttt{sum}$	transfer	0.01	0.073	0.805	104.510
	throw-catch	0.01	0.071	0.204	0.188
	mint	0.01	0.044	0.083	0.036
	burn	0.01	0.049	0.073	0.041
	mintN-helper	0.06	X	7.593	X
	mintN	X	X	X	X
$\sum_{A\in\mathcal{A}} \texttt{bal}(A) \geq \texttt{sum}$	transfer	0.01	0.067	0.780	2.634
	throw-catch	0.01	0.119	1.142	X
	mint	0.01	0.050	0.063	0.037
	burn	0.01	0.037	0.072	0.037
$\sum_{A\in\mathcal{A}} \texttt{bal}(A) \geq t$	transfer	0.01	0.057	0.758	2.177
	throw-catch	0.01	0.073	0.785	152.096
	mint	0.01	0.046	0.112	0.052
	burn	0.01	0.047	0.128	0.054

Table 3.2: Experimental results using Vampire and Z3. Time (in seconds) is given when the solver proved the (unsat) goal. Otherwise, timeout is indicated by 'X'.

Table 3.2 contains the results of Vampire (in different operation modes) and Z3 proving the properties mentioned, relative to the transition. If the prover was able to prove it unsatisfiable, then the time necessary to do so is listed. Otherwise, a time out or exceeded memory limit is indicated by an 'X' in the corresponding cell. Time out limit for Z3 was one hour, whereas for Vampire five minutes.

Remark. The transitions are implemented as defined in Section 3.2. In particular, 'transfer' is the only one considering the case that the transaction is not performable, while the others assume the existance of the necessary coins as precondition.

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

Both Vampire and Z3 could derive unsatisfiable for every standard transition and every property. The different modes of Vampire are listed, because they show an interesting complexity property of 'transfer' and 'throw-catch'. This is, the invariant (**Inactive-Coins**) is the hardest part to prove for 'transfer', while the fact that both active and inactive coins can be transferred for $\sum_{A \in \mathcal{A}} \texttt{bal}(A) \geq \texttt{sum}$ and $\sum_{A \in \mathcal{A}} \texttt{bal}(A) \geq t$ does not affect the running time for 'transfer', but makes 'throw-catch' much more complex. For $\sum_{A \in \mathcal{A}} \texttt{bal}(A) \geq t$ 'throw-catch' is even harder, since there are two activity functions `old-active` and `new-active` in contrast to $\sum_{A \in \mathcal{A}} \texttt{bal}(A) \geq \texttt{sum}$.

The transition 'mintN' shows what cannot be done (yet), in terms of automation. Performing actions for an arbitrary number of coins is problematic. It requires the provers to apply inductive reasoning which requires higher-order reasoning. This is the case, since it is all-quantified over all kinds of relations defined on an inductive datatype. However, providing the instance of induction needed as a lemma in the input (as it is done in 'mintN-helper'), both Vampire and Z3 find a proof.

TU Bibliothek, WIEN 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.

4 The Encoding is Sound and Complete

The aim of this chapter is to formalize the encoding of Section 3.1 and its relation to the higher-order setting. In particular, there are no transitions considered and we thus only have one macro and one micro world to reason about. The crucial part is the soundness and completeness result. At first, the necessary definitions are stated.

4.1 Higher-Order and First-Order States

As explained in Chapter 3, the higher-order setting is based on the two disjoint sets $\mathbb{N}$ and the finite set of addresses $\mathcal{A}$. Let us now define a higher-order state.

Definition 5 (Definition of Higher-Order State)**.** Let $\texttt{bal} \in \mathbb{N}^{\mathcal{A}}$ and $\texttt{sum} \in \mathbb{N}$, then the pair $(\texttt{bal}, \texttt{sum})$ is called a *higher-order state*.
The set of all higher-order states is called $\mathcal{H} := \mathbb{N}^{\mathcal{A}} \times \mathbb{N}$.

Definition 6 (Valid Higher-Order States)**.** A higher-order state $(\texttt{bal}, \texttt{sum})$ is called *valid*, if $\sum_{A \in \mathcal{A}} \texttt{bal}(A) = \texttt{sum}$.

The first-order setting has been explained informally in Chapter 3 already. It uses the two disjoint sets $\mathcal{C}$ which is the countable set of coins and the set of addresses $\mathcal{A}$.

Definition 7 (First-Order States)**.** Let $\texttt{has-coin} \subseteq \mathcal{A} \times \mathcal{C}$ and $\texttt{active} \subseteq \mathcal{C}$ then the pair $(\texttt{has-coin}, \texttt{active})$ is called a *first-order state*.
The set of all first-order states is called $\mathcal{F} := 2^{\mathcal{A} \times \mathcal{C}} \times 2^{\mathcal{C}}$.

4.2 Equivalent First-Order States

Having the basic principles at hand, we can now start defining what it means for two first-order states $(\texttt{has-coin}_1, \texttt{active}_1)$ and $(\texttt{has-coin}_2, \texttt{active}_2)$ to be equivalent. We start by defining some sets relevant for a relation $\sim$ on $\mathcal{F}$.

Definition 8. Given a state $s := (\texttt{active}, \texttt{has-coin})$. For an address $A \in \mathcal{A}$, we define:

$$C_A := \{c \in \mathcal{C} : \texttt{has-coin}(A, c)\}$$

to be the set of all coins c the address A has. Further, we define three types of error coins according to the invariants from Section 3.1:

i) $M_{\text{Inact}} := \{c \in \mathcal{C} : \neg\texttt{active}(c) \wedge \exists A \in \mathcal{A} : \texttt{has-coin}(A, c)\}$,

ii) $M_{\text{Least}} := \{c \in \mathcal{C} : \texttt{active}(c) \wedge \forall A \in \mathcal{A} : \neg\texttt{has-coin}(A, c)\}$ and

iii) $M_{\text{Most}} := \{c \in \mathcal{C} : \exists A, B \in \mathcal{A} : A \neq B \wedge \texttt{has-coin}(A, c) \wedge \texttt{has-coin}(B, c)\}$.

We define one type of error pairs:

$$M_{\text{Pairs}} := \{(A, c) \in \mathcal{A} \times \mathcal{C} : \texttt{has-coin}(A, c) \wedge \exists B \in \mathcal{A} : A \neq B \wedge \texttt{has-coin}(B, c)\} .$$

to refine the number of mistakes caused by the violation of (**At-Most-One-Address**). The *total number of mistakes* m_s of the first-order state s is now defined as:

$$m_s := |M_{\text{Inact}}| + |M_{\text{Least}}| + |M_{\text{Pairs}}| - |M_{\text{Most}}| .$$

Note that the transitions presented in Section 3.2 preserve the size of the mistake sets M_{Inact}, M_{Least}, M_{Pairs} and M_{Most}.
Given Definition 8, the following properties hold.

Lemma 5. *Let $s \in \mathcal{F}$. Then $|M_{Most}| \leq |M_{Pairs}|$ holds. Further, $|M_{Most}| = |M_{Pairs}|$ if and only if $M_{Most} = \emptyset$.*

Proof. Assume $M_{\text{Most}} = \emptyset$, that is there exists no coin c such that two distinct addresses A and B have it, which is equivalent to there is no pair (A, c) such that both A has c and a distinct B has c. This means precisely $M_{\text{Pairs}} = \emptyset$. Therefore $|M_{\text{Most}}| = |M_{\text{Pairs}}|$.

Now assume $c \in M_{\text{Most}}$. Let $A, B \in \mathcal{A}$, $A \neq B$ such that both $\texttt{has-coin}(A, c)$ and $\texttt{has-coin}(B, c)$. This implies both $(A, c), (B, c) \in M_{\text{Pairs}}$. The very same argument works for every other coin in M_{Most}. On the other hand, if a pair $(A, c) \in M_{\text{Pairs}}$, then by definition also $c \in M_{\text{Most}}$. Thus we have $|M_{\text{Most}}| < |M_{\text{Pairs}}|$.

□

Lemma 6. *Given a first-order state $s = (\texttt{has-coin}, \texttt{active}) \in \mathcal{F}$, such that $m_s = 0$, then $\bigcup_{A \in \mathcal{A}} C_A = \texttt{active}$.*

Proof. The first step is to prove $\bigcup_{A \in \mathcal{A}} C_A \subseteq \texttt{active}$.
Let $c \in \bigcup_{A \in \mathcal{A}} C_A$. That is, there exists an $A \in \mathcal{A}$ such that $\texttt{has-coin}(A, c)$. We know $m_s = 0$ which implies $M_{\text{Inact}} = \emptyset$ by applying Lemma 5, hence also $\texttt{active}(c)$.

To show $\texttt{active} \subseteq \bigcup_{A \in \mathcal{A}} C_A$, we consider an arbitrary $c \in \texttt{active}$. We know that $m_s = 0$, hence $c \notin M_{\text{Least}}$, by Lemma 5. In particular, there has to exist an $A \in \mathcal{A}$ such that $(A, c) \in \texttt{has-coin}$. Hence, $c \in \bigcup_{A \in \mathcal{A}} C_A$.

Combining the results, we obtain $\bigcup_{A \in \mathcal{A}} C_A = \texttt{active}$.

□

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

Lemma 7. *Given* $s = (\texttt{has-coin}, \texttt{active}) \in \mathcal{F}$, *such that* $m_s = 0$, *then* $|\texttt{active}| = \sum_{A \in \mathcal{A}} |C_A|$.

Proof. Assume there exist $A, B \in \mathcal{A}$, $A \neq B$ and $c \in \mathcal{C}$ such that both $c \in C_A$ and $c \in C_B$. That is the C_A are not pairwise disjoint. Then we have both $(A, c), (B, c) \in \texttt{has-coin}$ and thus $c \in M_{\text{Most}}$. Lemma 5 tells us that in this case $|M_{\text{Most}}| < |M_{\text{Pairs}}|$ which contradicts $m_s = 0$. By applying Lemma 6 we get $|\texttt{active}| = |\bigcup_{A \in \mathcal{A}} C_A| = \sum_{A \in \mathcal{A}} |C_A|$. □

Lemma 8. *Given* $s = (\texttt{has-coin}, \texttt{active}) \in \mathcal{F}$, *such that at least one of* $|M_{Least}| = 0$ *and* $|M_{Inact}| + |M_{Pairs}| - |M_{Most}| = 0$ *holds. Then*

$$\sum_{A \in \mathcal{A}} |C_A| - |\texttt{active}| = |M_{Inact}| + |M_{Pairs}| - |M_{Most}| - |M_{Least}| \,.$$

Proof. First, assume $|M_{\text{Inact}}| + |M_{\text{Pairs}}| - |M_{\text{Most}}| = 0$. Consider

$$M_{\text{Least}} = \{c \in \mathcal{C} : \texttt{active}(c) \land \forall A \in \mathcal{A} : \neg \texttt{has-coin}(A, c)\} \subseteq \texttt{active} \,.$$

Then the state $s' := (\texttt{has-coin}, \ \texttt{active} \setminus M_{\text{Least}})$ has $m'_s = 0$, since $M'_{\text{Least}} = \emptyset$ and the other mistake sets have not changed. They have been $\emptyset$, due to Lemma 5.
This means we can apply Lemma 7 and obtain $\sum_{A \in \mathcal{A}} |C'_A| = |\texttt{active}'|$. Therefore,

$$\sum_{A \in \mathcal{A}} |C_A| = \sum_{A \in \mathcal{A}} |C'_A| = |\texttt{active}'| = |\texttt{active}| - |M_{\text{Least}}|$$

and hence $\sum_{A \in \mathcal{A}} |C_A| - |\texttt{active}| = |M_{\text{Inact}}| + |M_{\text{Pairs}}| - |M_{\text{Most}}| - |M_{\text{Least}}|$.

Now assume $|M_{\text{Least}}| = 0$. We have $M_{\text{Inact}} \cap \texttt{active} = \emptyset$ and $M_{\text{Pairs}} \subseteq \texttt{has-coin}$. Consider the state $s' := (\texttt{has-coin} \setminus M_{\text{Pairs}}, \ \texttt{active} \cup M_{\text{Inact}})$. It satisfies $M'_{\text{Inact}} = \emptyset$ and $M'_{\text{Pairs}} = \emptyset$, and according to Lemma 5 also $M'_{\text{Most}} = \emptyset$. However, we also have $M'_{\text{Least}} = M_{\text{Most}}$. This is the case because $M_{\text{Least}} = \emptyset$ and precisely the coins that more than one address had before - which had to be active - are now not assigned to a sinlge one any more. Thus $s'' := (\texttt{has-coin} \setminus M_{\text{Pairs}}, (\texttt{active} \cup M_{\text{Inact}}) \setminus M_{\text{Most}})$ has $m''_s = 0$.
Applying Lemma 7 to s'', results in $\sum_{A \in \mathcal{A}} |C''_A| = |\texttt{active}''|$. Since we have $\sum_{A \in \mathcal{A}} |C''_A| = \sum_{A \in \mathcal{A}} |C_A| - |M_{\text{Pairs}}|$, by definition of M_{Pairs} and C_A, it follows:

$$\sum_{A \in \mathcal{A}} |C_A| - |M_{\text{Pairs}}| = \sum_{A \in \mathcal{A}} |C''_A| = |\texttt{active}''| = |\texttt{active}| + |M_{\text{Inact}}| - |M_{\text{Most}}| \,.$$

Hence, $\sum_{A \in \mathcal{A}} |C_A| - |\texttt{active}| = |M_{\text{Inact}}| + |M_{\text{Pairs}}| - |M_{\text{Most}}| - |M_{\text{Least}}|$ which conludes the proof. □

Having the necessary definitions and their properties at hand, we are now ready to define the relation on $\mathcal{F}$.

Definition 9 (Relation $\sim$)**.**
Let $s_1 = (\texttt{has-coin}_1, \texttt{active}_1)$, $s_2 = (\texttt{has-coin}_2, \texttt{active}_2) \in \mathcal{F}$ two first-order states. We say $s_1 \sim s_2 :\Leftrightarrow$

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. $|\texttt{active}_1| = |\texttt{active}_2|$,
2. $\forall A \in \mathcal{A}: \ |C_{1,A}| = |C_{2,A}|$,
3. $|M_{1,\text{Least}}| = |M_{2,\text{Least}}|$,
4. $|M_{1,\text{Inact}}| + |M_{1,\text{Pairs}}| - |M_{1,\text{Most}}| = |M_{2,\text{Inact}}| + |M_{2,\text{Pairs}}| - |M_{2,\text{Most}}|$.

The most important property of $\sim$ is stated below.

Lemma 9. *The relation $\sim$ is an equivalence relation on $\mathcal{F}$.*

Proof.

- Reflexivity of $\sim$.
  Let $s = (\texttt{has-coin}, \texttt{active}) \in \mathcal{F}$, then clearly $|\texttt{active}| = |\texttt{active}|$, for all $A \in \mathcal{A}$ we have $|C_A| = |C_A|$ and also for all $x \in \{\text{Inact}, \text{Least}, \text{Most}, \text{Pairs}\}$ it is the case that $|M_x| = |M_x|$.
- Symmetry of $\sim$.
  Let $s_1, s_2 \in \mathcal{F}$ such that $s_1 \sim s_2$, then due to symmetry of $=$ also $s_2 \sim s_1$ holds.
- Transitivity of $\sim$.
  Let $s_1, s_2, s_3 \in \mathcal{F}$, such that $s_1 \sim s_2$ and $s_2 \sim s_3$ then due to the transitivity of $=$ also $s_1 \sim s_3$ holds.

□

Finally, the notion of a valid class of first-order states can be expressed.

Definition 10 (Valid Class of First-Order States)**.** Let $[s]_\sim \in \mathcal{F}/_\sim$ be a class of first-order states. It is called *valid* if for one (or equivalently for every) $s \in [s]_\sim$ it holds $m_s = 0$.

Remark. A first-order state s is valid if and only if it satisfies the invariants from Section 3.1. This is the case since precisely for states $s \in \mathcal{F}$ satisfying the invariants all of the sets M_{Inact}, M_{Least}, M_{Most} and M_{Pairs} are empty, hence $m_s = 0$.

4.3 Translating Higher-Order States to First-Order States

We are now looking for a way to unambiguously map states in higher-order logic to states in first-order logic. This connection is indicated in Figure 3.1 by the blue dashed arrows. Furthermore, it has been mentioned as 'Encoding' and 'Decoding' in Table 3.1. As already discussed, in the first-order setting we will consider equivalence classes of states, that is $\mathcal{F}/_\sim$.
Thus, we want to have an injective function $f : \mathcal{H} \to \mathcal{F}/_\sim$. As shown in Section 4.4, the function we are looking for is the following.

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

Theorem 2. *Let the function* $f : \mathcal{H} \to \mathcal{F}/_\sim \quad (\mathit{bal}, \mathit{sum}) \mapsto [(\mathit{has\text{-}coin}, \mathit{active})]_\sim$*, such that it satisfies the following conditions:*

1. $\mathit{sum} = |\mathit{active}|$.
2. *For every* $A \in \mathcal{A}$ *it holds* $\mathit{bal}(A) = C_A$.
3. *At least one of* $|M_{Least}| = 0$ *and* $|M_{Inact}| + |M_{Pairs}| - |M_{Most}| = 0$ *holds.*

Here $(\mathit{has\text{-}coin}, \mathit{active}) \in [(\mathit{has\text{-}coin}, \mathit{active})]_\sim$ *arbitrary.*
The function f *is well-defined and injective.*

Proof. To show f is well defined, we have to make sure that if one element of a class satisfies the properties above, then all the others do as well. Further, we have to show that if two distinct first states s_1 and s_2 satisfy the function definition properties, then they are members of the same class $s_2 \in [s_1]_\sim$ and lastly, that there exists only one function satisfying the constraints.

The first part is ensured by the definition of $\sim$. The classes are constructed in such a way that precisely the four crucial values for the definition of f are the same for every class member.

For the second part we consider two distinct first-order states s_1 and s_2, both satisfying the properties 1, 2 and 3. This implies that both $|\mathtt{active}_1| = |\mathtt{active}_2|$ and $|C_{1,A}| = |C_{2,A}|$ for every A. Applying Lemma 8, we know $|M_{1,\text{Inact}}| + |M_{1,\text{Pairs}}| - |M_{1,\text{Most}}| - |M_{1,\text{Least}}| = |M_{2,\text{Inact}}| + |M_{2,\text{Pairs}}| - |M_{2,\text{Most}}| - |M_{2,\text{Least}}|$. Then, by Lemma 5, it cannot be the case that $M_{1,\text{Least}} = 0$ and $M_{2,\text{Least}} \neq 0$ and vice versa, but one summand has to be zero. This implies that both $M_{1,\text{Least}} = M_{2,\text{Least}}$ and $|M_{1,\text{Inact}}| + |M_{1,\text{Pairs}}| - |M_{1,\text{Most}}| = |M_{2,\text{Inact}}| + |M_{2,\text{Pairs}}| - |M_{2,\text{Most}}|$. Therefore, $s_1 \in [s_2]$.

To show there exists only one such f, we assume the existance of another function g fulfilling the same constraints. Let $h \in \mathcal{H}$ arbitrary and $(\mathtt{has\text{-}coin}_f, \mathtt{active}_f) \in f(h)$, $(\mathtt{has\text{-}coin}_g, \mathtt{active}_g) \in g(h)$. Then $|\mathtt{active}_f| = \mathtt{sum} = |\mathtt{active}_g|$, for all $A \in \mathcal{A}$ $|C_{f,A}| = \mathtt{bal}(A) = |C_{g,A}|$ and with the same argument as before $M_{f,\text{Least}} = M_{g,\text{Least}}$ and $|M_{f,\text{Inact}}| + |M_{f,\text{Pairs}}| - |M_{f,\text{Most}}| = |M_{g,\text{Inact}}| + |M_{g,\text{Pairs}}| - |M_{g,\text{Most}}|$. Hence, $(\mathtt{has\text{-}coin}_g, \mathtt{active}_g) \in [(\mathtt{has\text{-}coin}_f, \mathtt{active}_f)]_\sim$. But this means $f(h) = g(h)$ for every $h \in \mathcal{H}$, hence $f = g$.

For injectivity, assume two higher-order states $h_1 = (\mathtt{bal}_1, \mathtt{sum}_1)$ and $h_2 = (\mathtt{bal}_2, \mathtt{sum}_2)$ such that $f(h_1) = f(h_2)$. Now, we know $\mathtt{sum}_1 = |\mathtt{active}_1| = |\mathtt{active}_2| = \mathtt{sum}_2$ and for arbitrary $A \in \mathcal{A}$ we know $\mathtt{bal}_1(A) = |C_{1,A}| = |C_{2,A}| = \mathtt{bal}_2(A)$. Therefore, we have $h_1 = h_2$. □

4.4 Soundness and Completeness

Using the definitions and lemmas from Section 4.3, our next result follows naturally.

Theorem 3. *Given* $s \in \mathcal{F}$, $h \in \mathcal{H}$, *then* $s \in f(h)$ *implies*

$$\sum_{A \in \mathcal{A}} \mathit{bal}(A) - \mathit{sum} = |M_{Inact}| + |M_{Pairs}| - |M_{Most}| - |M_{Least}| \, .$$

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

Proof. Let $h = (\texttt{bal}, \texttt{sum}) \in \mathcal{H}$ and $s = (\texttt{has-coin}, \texttt{active}) \in f(h)$. For s we know at least one of $|M_{\text{Least}}| = 0$ and $|M_{\text{Inact}}| + |M_{\text{Pairs}}| - |M_{\text{Most}}| = 0$ and hence using Lemma 8 it follows $\sum_{A \in \mathcal{A}} \texttt{bal}(A) - \texttt{sum} = \sum_{A \in \mathcal{A}} |C_A| - |\texttt{active}| = |M_{\text{Inact}}| + |M_{\text{Pairs}}| - |M_{\text{Most}}| - |M_{\text{Least}}|$. This proves the theorem.

□

A special instance of Theorem 3 is now our soundness and completeness result.

Corollary 1 (Soundness and Completeness of the Encoding)**.**
Let $h \in \mathcal{H}$ a higher-order state, then

$$h \textit{ valid} \quad \Leftrightarrow \quad \forall s \in f(h) : s \textit{ valid} .$$

That is for $h = (\texttt{bal}, sum)$ and $s = (\texttt{has-coin}, \texttt{bal}) \in f(h)$ it holds that

$$\sum_{A \in \mathcal{A}} \texttt{bal}(A) = sum \quad \Leftrightarrow \quad (\texttt{has-coin}, \texttt{active}) \textit{ satisfies the invariants} .$$

Proof. Given h and s as specified and let $\sum_{A \in \mathcal{A}} \texttt{bal}(A) = \texttt{sum}$. Then according to Theorem 1, $|M_{\text{Inact}}| + |M_{\text{Pairs}}| - |M_{\text{Most}}| - |M_{\text{Least}}| = 0$. Since by definition of f, one of $|M_{\text{Least}}| = 0$ and $|M_{\text{Inact}}| + |M_{\text{Pairs}}| - |M_{\text{Most}}| = 0$ had to be the case already, it follows that the other has to be zero as well. Thus, s is valid and satisfies the invariants.

Let now $s \in f(h)$ be valid (satisfy the invariants), then $m_s := |M_{\text{Inact}}| + |M_{\text{Least}}| + |M_{\text{Pairs}}| - |M_{\text{Most}}| = 0$. Applying Lemma 5, we know that each the summand has to be zero. This implies that $|M_{\text{Inact}}| + |M_{\text{Pairs}}| - |M_{\text{Most}}| - |M_{\text{Least}}| = 0$ and hence $\sum_{A \in \mathcal{A}} \texttt{bal}(A) = \texttt{sum}$. □

Note: For other properties over sum, such as being greater or lower equal some number, the mistakes counter has to be adapted and one of the invariants has to be dropped, depending on the property to be proven.

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.
TU Bibliothek WIEN Your knowledge hub

5 The Translation f is First-Order Expressible

The goal of this chapter is to make the function f from Theorem 2 (Chapter 4) expressible in first-order logic. In order to do so, the sets of higher-order and first-order states $\mathcal{H}$ and $\mathcal{F}$, respectively, are used as before, but in this chapter only valid states are considered. This means the former invariants as in Definition 4 (Chapter 3) are now axioms and thus $\sum_{A \in \mathcal{A}} \texttt{bal}(A) = \texttt{sum}$ is given and does not have to be shown. Hence, we are now addressing the orthogonal problem to the problem from Chapter 3. However, we aim for the encoding to be in such a way that automated theorem provers can reason about it as well.
The general idea is to add different counters on coins to be able to express some of the cardinality constraints from Chapter 4. Unfortunately, this requires the use of integers or an inductive datatype. In this work, integers are preferred, since linear integer arithmetic is supported in most theorem provers.
While in Chapter 3 we aimed to prove the invariants are preserved given a certain transition, in this chapter we have the invariants as a precondition and want to prove the impact on the sum, given the relation of the balances, as mentioned in the 'precise explanation' of Section 3.2. As a consequence, the motivation of software verification is not in our focus here.

In Section 5.1, a naive encoding of f is presented. Subsequently, in Section 5.2, the previous encoding is restricted in two steps, such that it does not affect expressiveness. The experimental results of the theorem provers' performance on the different encodings are discussed in Section 5.3.

5.1 Naive First-Order Encoding

In this section a very natural approach for encoding f is used. As mentioned above, we want to express some of the cardinalities from Chapter 4 by defining counters of coins. Since many of them are only necessary to handle the number of mistakes, only a few remain to be addressed.
Generally, the functions and predicates `bal`, `sum`, `has-coin` and `active` are as defined in Chapters 3-4.

Definition 11 (Index Function)**.** For every $A \in \mathcal{A}$ let $\texttt{ind}_A : \mathcal{C} \to \mathbb{N}^+$ be an injective function. Further, $\texttt{ind}_A$ shall be surjective on the interval $[1, \texttt{bal}(A)]$. Then $\texttt{ind}_A(c)$ is the index of a coin c with respect to the address A.
The function $\texttt{ind} : \mathcal{A} \times \mathcal{C} \to \mathbb{N}^+$, where $\texttt{ind}(A, c) := \texttt{ind}_A(c)$, is called the *index function*.

We aim at ensuring that every address A has precisely the first $\texttt{bal}(A)$ coins. That means exactly the coins $c \in \mathcal{C}$ with $\texttt{ind}_A(c) \in [1, \texttt{bal}(A)]$. For this set of coins to have the right cardinality, $\texttt{ind}_A$ has to be injective and surjective at least on that interval.

Definition 12 (Count Function)**.** Let $\texttt{count} : \mathcal{C} \to \mathbb{N}^+$ be an injective function. Also, it shall be surjective on the interval $[1, \texttt{sum}]$. Then, for every coin $c \in \mathcal{C}$, $\texttt{count}(c)$ is called the *number* of c.

Similarly to the index function, the count function aims to enumerate the active coins. Hence, we want that exactly every active coin c has a number $\texttt{count}(c) \in [1, \texttt{sum}]$. Further, `count` has to be injective and surjective at least on that interval to provide the desired property.

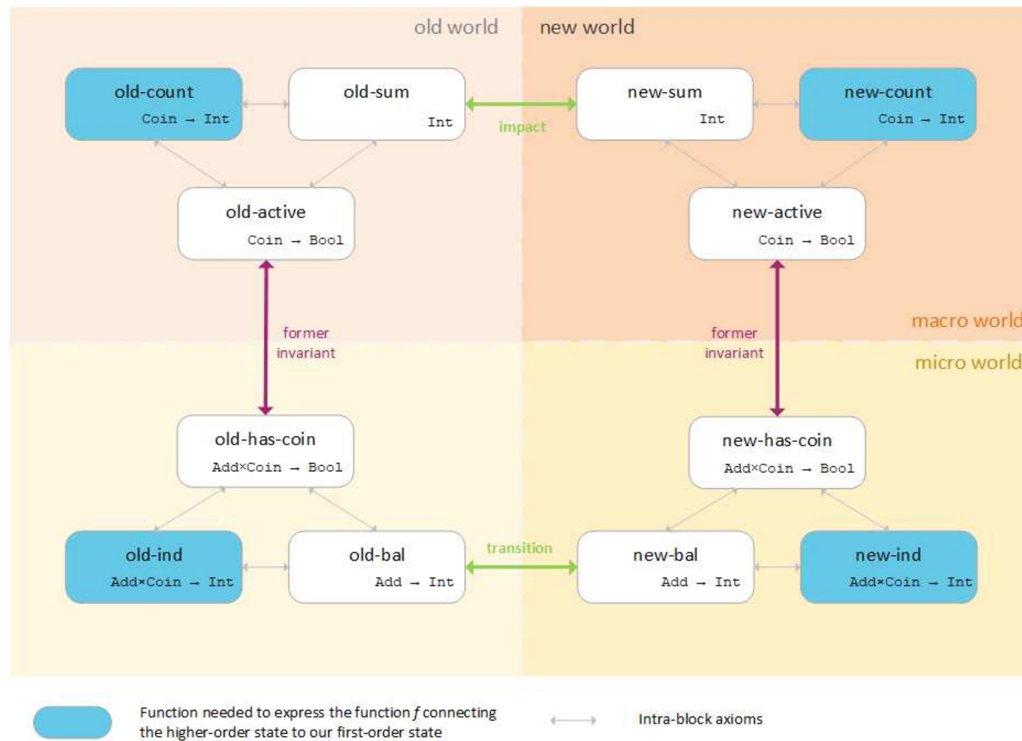

Figure 5.1: Representation of the naive encoding to make f explicit.

As in Chapter 3.1, we also have two of such settings in parallel which is illustrated in Figure 5.1. Again, we use the `old-` and `new-` prefixes to distinguish them. If some property is meant to apply for both, we will use the prefix `z-`. Both shall fulfill the following axioms.

Axioms in the Naive Encoding

The following list of axioms is structured according to Figure 5.1. The axioms corresponding to one quarter, called block, are presented together.

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

Intra-block axioms for the macro world.
These are all the axioms regarding sum, the count function count and their relation to active. They are represented by the grey arrows between the respective bubbles in Figure 5.1. The axioms are:

$$\texttt{z-sum} \geq 0 \tag{M1}$$
$$\forall c \in \mathcal{C} : \texttt{z-count}(c) > 0 \tag{M2}$$
$$\forall c_1, c_2 \in \mathcal{C} : \big(\texttt{z-count}(c_1) = \texttt{z-count}(c_2) \to c_1 = c_2\big) \tag{M3}$$
$$\forall n \in \mathbb{Z} : \big(0 < n \land n \leq \texttt{z-sum} \to \exists c \in \mathcal{C} : \texttt{z-count}(c) = n\big) \tag{M4}$$
$$\forall c \in \mathcal{C} : \big(\texttt{z-active}(c) \leftrightarrow \texttt{z-count}(c) \leq \texttt{z-sum}\big) \tag{M5}$$

The axioms (M1) - (M5) formalize, that the sum is non-negative (M1), the codomain of count is $\mathbb{N}^+$ (M2), count is injective (M3) and partly surjective (M4). A coin is active if and only if its number is within a certain interval (M5).

Intra-block axioms for the micro world.
The axioms listed here effect bal, the index function ind and their relation to has-coin. They are:

$$\forall A \in \mathcal{A} : \texttt{z-bal}(A) \geq 0 \tag{m1}$$
$$\forall c \in \mathcal{C}, A \in \mathcal{A} : \texttt{z-ind}(A, c) > 0 \tag{m2}$$
$$\forall A \in \mathcal{A}, c_1, c_2 \in \mathcal{C} : \big(\texttt{z-ind}(A, c_1) = \texttt{z-ind}(A, c_2) \to c_1 = c_2\big) \tag{m3}$$
$$\forall A \in \mathcal{A}, n \in \mathbb{Z} : \big(0 < n \land n \leq \texttt{z-bal}(A) \to \exists c \in \mathcal{C} : \texttt{z-ind}(A, c) = n\big) \tag{m4}$$
$$\forall A \in \mathcal{A}, c \in \mathcal{C} : \big(\texttt{z-has-coin}(A, c) \leftrightarrow \texttt{z-ind}(A, c) \leq \texttt{z-bal}(A)\big) \tag{m5}$$

The axioms (m1)-(m5) state that the balances are non-negative (m1), the indicies are in $\mathbb{N}^+$ (m2), for every $A \in \mathcal{A}$ the $\texttt{ind}_A$ function is injective (m3) and partly surjective (m4). The address A has precisely the coins c with index $\texttt{ind}_A(c) \leq \texttt{bal}(A)$ (m5).

Inter-block axioms connecting the micro and the macro world.
The axioms between the micro and the macro world are exactly the former invariants from Chapter 3.1. Now they have the role of inter-block axioms, ensuring $\sum_{A \in \mathcal{A}} \texttt{bal}(A) = \texttt{sum}$. They are represented by the purple arrows in Figure 5.1 and are stated again for the sake of completeness:

$$\forall c \in \mathcal{C} : \ \big(\exists A \in \mathcal{A} : \texttt{z-has-coin}(A, c)\big) \leftrightarrow \texttt{z-active}(c) \tag{I1}$$
$$\forall A_1, A_2 \in \mathcal{A}, c \in \mathcal{C} : \texttt{z-has-coin}(A_1, c) \land \texttt{z-has-coin}(A_2, c) \to A_1 = A_2 \tag{I2}$$

With this list of axioms, we have the following result.

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

Lemma 10. *Given a higher-order state* $h = (\mathtt{bal}, \mathtt{sum}) \in \mathcal{H}$, *a first-order state* $s = (\mathtt{has\text{-}coin}, \mathtt{active}) \in \mathcal{F}$, $\mathtt{ind}$ *and* $\mathtt{count}$ *such that (M1)-(M5), (m1)-(m5) and (I1)-(I2) hold.*
Then $|\mathtt{active}| = \mathtt{sum}$ *and for every* $A \in \mathcal{A}$ *it holds* $|\{c \in \mathcal{C} : \mathtt{has\text{-}coin}(A, c)\}| = \mathtt{bal}(A)$.
In particular, we have $\sum_{A \in \mathcal{A}} \mathtt{bal}(A) = \mathtt{sum}$.

Proof. Let s and h be as specified and all the axioms be satisfied. At first, we show $|\mathtt{active}| = \mathtt{sum}$. For every coin $c \in \mathcal{C}$ we know it is active if and only if $\mathtt{count}(c) \in [1, \mathtt{sum}]$ (M5). Further, $\mathtt{count}$ is injective (M3), thus $|\mathtt{active}| = \mathtt{sum}$ and for every integer $n \in [1, \mathtt{sum}]$ there exists a coin c such that $\mathtt{count}(c) = n$ (M4). Hence $|\mathtt{active}| = \mathtt{sum}$.

Fix an arbitrary $A \in \mathcal{A}$. The same argumentation as before, using (m3)-(m5) instead of (M3)-(M5), leads to $|\{c \in \mathcal{C} : \mathtt{has\text{-}coin}(A, c)\}| = \mathtt{bal}(A)$.

We know now that $|\mathtt{active}| = \mathtt{sum}$, for every $A \in \mathcal{A}$ it holds $|\{c \in \mathcal{C} : \mathtt{has\text{-}coin}(A, c)\}| = \mathtt{bal}(A)$ and $m_s = 0$, since s satisfies $I1$ and $I2$. In particular, $[s]_\sim$ is in the image set of f. Hence, $s \in f(h)$. By Corollary 1, as $s = (\mathtt{has\text{-}coin}, \mathtt{active})$ satisfies the invariants, it follows $\sum_{A \in \mathcal{A}} \mathtt{bal}(A) = \mathtt{sum}$.

□

Lemma 10 states that the function f restricted to valid higher-order states is first-order expressible using our encoding. It also proves soundness of our encoding to reason about sums.

Relation between the Old World and the New World

So far, we have only considered one micro world together with one macro world. Thus, in terms of Figure 5.1, the lime arrows between the old world and the new world have not been discussed yet. Compared to Chapter 3, the previously implicit higher-order relations are made explicit. As we can reason about `bal` and `sum` directly, there is no need of equivalent first-order transition and impact any more. Therefore, the dark green arrows from Figure 3.1 are now redundant and are thus dropped.
The transition formula expresses the relation of `old-bal` and `new-bal`, whereas the expected impact formula closes the dependency cycle in the macro world. The setting in this chapter is the following.

Reasoning goal of Chapter 5.
Given the axioms (M1)-(M5), (m1)-(m5) and (I1)-(I2) hold, then

$$\text{the transition formula holds} \Rightarrow \text{the impact formula holds}\,.$$

For now, the most basic transition is encoded below as an example. This is, for every account, its balance is unchanged.

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

'By changing nothing, nothing changes.'

$$\forall A \in \mathcal{A} : \texttt{old-bal}(A) = \texttt{new-bal}(A) \qquad \text{(transition)}$$
$$\texttt{old-sum} = \texttt{new-sum} \qquad \text{(impact)}$$

Even though, this naive encoding indeed expresses f in first-order logic, it is very unlikely that any theorem prover will ever be able to prove a claim in this setting. The major problem is that it requires higher-order reasoning to link the functions `old-ind` and `new-ind`, `old-count` and `new-count`, respectively.
In the next section a promising simplification is presented addressing the usability of our encoding in automated theorem proving.

5.2 Restricted, yet Equally Expressive Encodings

The crucial point in this restricted version is the fact that we do not need two different index and count functions. This claim is explained and proved in this section. In Figure 5.2 the new setting is illustrated. Despite having only one mutual `ind` and one mutual `count`, the picture has not changed compared to Figure 5.1.

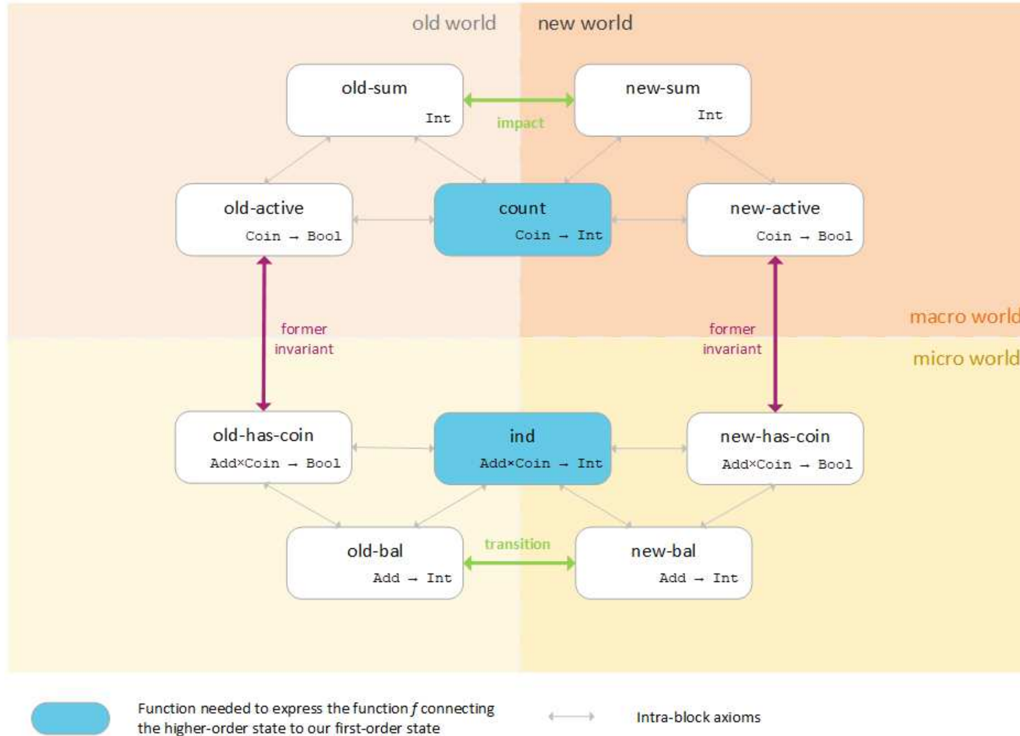

Figure 5.2: The restricted version of the encoding to make f explicit.

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 will show that this is possible without losing any generality of the setting, by having the following list of axioms.

Axioms in the Restricted Encoding

The axioms are very similar to the ones in Section 5.1.

Intra-block axioms for the macro world.
Now writing `count` instead of `z-count`, we have:

$$\texttt{z-sum} \geq 0 \tag{M1$'$}$$
$$\forall c \in \mathcal{C} : \texttt{count}(c) > 0 \tag{M2$'$}$$
$$\forall c_1, c_2 \in \mathcal{C} : \big(\texttt{count}(c_1) = \texttt{count}(c_2) \rightarrow c_1 = c_2\big) \tag{M3$'$}$$
$$\forall n \in \mathbb{Z} : \big(0 < n \wedge n \leq \texttt{z-sum} \rightarrow \exists c \in \mathcal{C} : \texttt{count}(c) = n\big) \tag{M4$'$}$$
$$\forall c \in \mathcal{C} : \big(\texttt{z-active}(c) \leftrightarrow \texttt{count}(c) \leq \texttt{z-sum}\big) \tag{M5$'$}$$

Intra-block axioms for the micro world.
We use `ind` instead of `z-ind` and state:

$$\forall A \in \mathcal{A} : \texttt{z-bal}(A) \geq 0 \tag{m1$'$}$$
$$\forall c \in \mathcal{C}, A \in \mathcal{A} : \texttt{ind}(A, c) > 0 \tag{m2$'$}$$
$$\forall A \in \mathcal{A}, c_1, c_2 \in \mathcal{C} : \big(\texttt{ind}(A, c_1) = \texttt{ind}(A, c_2) \rightarrow c_1 = c_2\big) \tag{m3$'$}$$
$$\forall A \in \mathcal{A}, n \in \mathbb{Z} : \big(0 < n \wedge n \leq \texttt{z-bal}(A) \rightarrow \exists c \in \mathcal{C} : \texttt{ind}(A, c) = n\big) \tag{m4$'$}$$
$$\forall A \in \mathcal{A}, c \in \mathcal{C} : \big(\texttt{z-has-coin}(A, c) \leftrightarrow \texttt{ind}(A, c) \leq \texttt{z-bal}(A)\big) \tag{m5$'$}$$

Inter-block axioms, transition and impact.
The inter-block axioms (I1)-(I2), the transition and its impact are not affected by the restriction.

Impact of the Restriction

Having two separate count and index functions leads to the freedom of compairing any two first-order states. That means, given the two valid higher-order states $h_o = (\texttt{old-bal}, \texttt{old-sum})$, $h_n = (\texttt{new-bal}, \texttt{new-sum}) \in \mathcal{H}$, we can reason about any two $s_o = (\texttt{old-has-coin}, \texttt{old-active}) \in f(h_o)$, $s_n = (\texttt{new-has-coin}, \texttt{new-active}) \in f(h_n)$. For automated provers this freedom is problematic, as it leaves them without any contraints about the transition. More precisely, provers have the need to have a stronger relation between the old and the new world. In the case of the setting from Chapter 3, the transition is made precise on coin level even. Thus the solver knows for every coin, which address it is assigned to in the new world, relative to the old world.

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

Hence what we lose when considering only one count and one index function is the possibility to change the affiliation of coins unnecessarily, while the total number of coins in each address stays as specified in the transition. As shown subsequently, the discussed constraints lead to the situation that fixing one of s_o, s_n, the other one has to be as similar as possible, while satisfying the cardinality restrictions.

The claim that we do not lose any generality of the encoding from Section 5.1 is now made precise and is proven. By 'no loss of generality' and 'equally expressive' we mean:

> *Every relation between two valid higher-order states h_o, h_n that could be expressed in Section 5.1 can still be expressed.*

For the practical use with theorem provers, this is relevant, since it ensures that independent of the transition, all the axioms are satisfied even if the two worlds have the same count and index functions. Hence, giving the prover the axioms, the transition and the negated impact formula as input, if it yields `unsat`, this is due to the negated impact.

We need the following lemmas to prove the claim.

Lemma 11.
Given two valid higher-order states $h_x = (\texttt{x-bal}, \texttt{x-sum})$, $h_y = (\texttt{y-bal}, \texttt{y-sum}) \in \mathcal{H}$ such that $\texttt{x-sum} \leq \texttt{y-sum}$. Further, let $s_x = (\texttt{x-has-coin}, \texttt{x-active}) \in f(h_x)$. Then there exists $s_y = (\texttt{y-has-coin}, \texttt{y-active}) \in f(h_y)$ satisfying the following properties:

(1) $\texttt{x-active} \subseteq \texttt{y-active}$.

(2) For $A \in \mathcal{A}$ with $\texttt{x-bal}(A) \leq \texttt{y-bal}$ it holds $\texttt{x-has-coin}(A, c) \Rightarrow \texttt{y-has-coin}(A, c)$, for every coin $c \in \mathcal{C}$.

(3) For $A \in \mathcal{A}$ with $\texttt{x-bal}(A) \geq \texttt{y-bal}$ it holds $\texttt{y-has-coin}(A, c) \Rightarrow \texttt{x-has-coin}(A, c)$, for every coin $c \in \mathcal{C}$.

Proof.
For h_y valid, recall that

$$f(h_y) := \{(\texttt{has-coin}, \texttt{active}) \in \mathcal{F} : \ m = 0, \\ |\texttt{active}| = \texttt{sum} = \sum_{A \in \mathcal{A}} \texttt{y-bal}(A), \forall A \in \mathcal{A} : |\texttt{has-coin}(A, .)| = \texttt{y-bal}(A)\}.$$

To fulfill property (1) of Lemma 11, construct `y-active` by taking $\texttt{x-active} \subseteq \mathcal{C}$ and add $\texttt{y-sum} - \texttt{y-sum}$ many coins from $\mathcal{C} \setminus \texttt{x-active}$ to get `y-active`.

For conditions (2) and (3) of Lemma 11, to every A exactly $\texttt{y-bal}(A)$ coins $c \in \texttt{y-active}$ need to be assigned. Which ones does not matter as long as the same coin is not used twice ($m_y = 0$). As also $m_x = 0$, there will not be any coin taken twice by overtaking the coin's assignment as far as possible. To be precise, property (3) has to be considered first to ensure a correct assignment.
For (3), we consider $A \in \mathcal{A}$ with $\texttt{x-bal}(A) \geq \texttt{y-bal}(A)$. In this case, it is known which

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

coins can be assigned to A. They are the ones such that $\texttt{x-has-coin}(A,c)$. 'Assign' means to set $\texttt{y-has-coin}$(A,c) to be true. Hence, for each such address A after doing so $\texttt{x-bal}(A) - \texttt{y-bal}(A)$ many coins are $\texttt{y-active}$ but not assigned to any address yet.
In the other case, $\texttt{x-bal}(A) \leq \texttt{y-bal}(A)$ which means (2), we only know a part of the coins to assign to A, namely the ones with $\texttt{x-has-coin}(A,c)$. Hence, there are still $\texttt{y-bal}(A) - \texttt{x-bal}(A)$ coins missing to be assigned to A.

Now consider the following coins are available. 'Available' here means $c \in \texttt{y-active}$ and we have not specified for which A it holds $\texttt{y-has-coin}(A,c)$ yet.

Coins	*Number of Coins*
$\texttt{y-active} \setminus \texttt{x-active}$	$\texttt{y-sum} - \texttt{x-sum}$
$c \in \mathcal{C} : \exists A \in \mathcal{A} :$ $\texttt{x-has-coin}(A,c) \land \neg\texttt{y-has-coin}(A,c)$	$\sum_{A\in\mathcal{A},\texttt{x-bal}(A)>\texttt{y-bal}(A)} \texttt{x-bal}(A) - \texttt{y-bal}(A)$

Table 5.1: Coins not considered in $\texttt{y-has-coin}$.

Note that the set of coins in the second row of Table 5.1 is a subset of $\texttt{x-active}$ and thus also of $\texttt{y-active}$.
Also, the following amount of coins is still needed to be considered in $\texttt{y-has-coin}$.
For every $A \in \mathcal{A}$ with $\texttt{y-bal}(A) > \texttt{x-bal}(A)$ we need $\texttt{y-bal}(A) - \texttt{x-bal}(A)$ many coins to be assigned to it. This yields

$$\sum_{\substack{A\in\mathcal{A},\\ \texttt{y-bal}(A)>\texttt{x-bal}(A)}} \texttt{y-bal}(A) - \texttt{x-bal}(A)$$

many coins.

Hence, the difference is

$$\begin{aligned}
\texttt{y-sum} - \texttt{x-sum} \quad &+ \sum_{\substack{A\in\mathcal{A},\\ \texttt{x-bal}(A)>\texttt{y-bal}(A)}} \texttt{x-bal}(A) - \texttt{y-bal}(A) \\
&- \sum_{\substack{A\in\mathcal{A},\\ \texttt{y-bal}(A)>\texttt{x-bal}(A)}} \texttt{y-bal}(A) - \texttt{x-bal}(A) \\
\overset{\text{states_valid}}{=} \left(\sum_{A\in\mathcal{A}} \texttt{y-bal}(A) - \sum_{A\in\mathcal{A}} \texttt{x-bal}(A) \right) &+ \sum_{\substack{A\in\mathcal{A},\\ \texttt{x-bal}(A)>\texttt{y-bal}(A)}} \texttt{x-bal}(A) - \texttt{y-bal}(A) \\
&- \sum_{\substack{A\in\mathcal{A},\\ \texttt{y-bal}(A)>\texttt{x-bal}(A)}} \texttt{y-bal}(A) - \texttt{x-bal}(A)
\end{aligned}$$

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 Bibliothek WIEN Your knowledge hub

$$\overset{\mathcal{A}\text{ finite}}{=} \sum_{A\in\mathcal{A}} \texttt{y-bal}(A) - \texttt{x-bal}(A) - \sum_{\substack{A\in\mathcal{A},\\ \texttt{x-bal}(A)>\texttt{y-bal}(A)}} \texttt{y-bal}(A) - \texttt{x-bal}(A) - \sum_{\substack{A\in\mathcal{A},\\ \texttt{y-bal}(A)>\texttt{x-bal}(A)}} \texttt{y-bal}(A) - \texttt{x-bal}(A)$$

$$= \sum_{\substack{A\in\mathcal{A},\\ \texttt{x-bal}(A)=\texttt{y-bal}(A)}} \texttt{y-bal}(A) - \texttt{x-bal}(A) = 0\,.$$

Hence, the remaining coins c needed to be fixed to belong to A with respect to $\texttt{y-has-coin}$ can be taken from the available ones, by using each of them exactly once. Since, as just shown, the number of available coins and the number of coins needed is the same, $m_y = 0$ can be guaranteed.

Thus, there exists $s_y = (\texttt{y-active}, \texttt{y-has-coin}) \in f(h_y)$ satisfying (1), (2) and (3).

$\square$

Lemma 12. *Given $h_x, h_y \in \mathcal{H}$ valid, $s_x \in f(h_x)$ as in Lemma 11. Then there is an $s_y \in f(h_y)$ as specified in Lemma 11, such that there exists a function $\texttt{count} : \mathcal{C} \to \mathbb{N}^+$ with:*

(i) $\texttt{count}$ *is injective,*

(ii) $\texttt{count}|_{\texttt{y-active}} : \texttt{y-active} \to [1, \texttt{y-sum}]$ *is a bijection and also*

(iii) $\texttt{count}|_{\texttt{x-active}} : \texttt{x-active} \to [1, \texttt{x-sum}]$ *is a bijection.*

And for every $A \in \mathcal{A}$ there exists a function $\texttt{ind}_A : \mathcal{C} \to \mathbb{N}^+$ with:

(iv) $\texttt{ind}_A$ *is injective,*

(v) $\texttt{ind}_A|_{C_{y,A}} : C_{y,A} \to [1, \texttt{y-bal}(A)]$ *is a bijection and also*

(vi) $\texttt{ind}_A|_{C_{x,A}} : C_{x,A} \to [1, \texttt{x-bal}(A)]$ *is a bijection.*

Proof. At first, we fix an arbitray bijective function

$$\texttt{count}' : \texttt{x-active} \subseteq \mathcal{C} \to [1, \texttt{x-sum}]\,.$$

This is possible, because $s_x \in f(h_x)$ and therefore $|\texttt{x-active}| = \texttt{x-sum}$.
Similarly, for every $A \in \mathcal{A}$, we fix a bijection

$$ind'_A : C_{x,A} \to [1, \texttt{x-bal}(A)]\,.$$

This is possible since $= |C_{x,A}| = |\{c \in \mathcal{C} : \texttt{x-has-coin}(A, c)\}| = \texttt{x-bal}(A)$ for every A, by definition of f.

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, since h_x is valid and thus also s_x is valid, precisely every coin in $\texttt{x-active}$ together with exactly one A (the A with $\texttt{x-has-coin}(A,c)$) has got assigned a value via $\texttt{ind}'$. That is, it results from h_x being valid that $c \in \texttt{x-active}$ if and only if there exists an $A \in \mathcal{A}$ with $\texttt{x-has-coin}(A,c)$.

In order to extend $\texttt{ind}'$ and $\texttt{count}'$ to their full scopes, $f(h_y)$ has to be considered. It is desired to find and fix $s_y = (\texttt{y-active}, \texttt{y-has-coin}) \in f(h_y)$ such that it satisfies the following properties.

(α) For all coins $c \in \mathcal{C}$ it holds $\texttt{x-active}(c)$ implies $\texttt{y-active}(c)$. That means we do not "lose" any active coins. As a consequence, for all $c \in \mathcal{C}$ if there exists $A \in \mathcal{A}$ such that $\texttt{x-has-coin}(A,c)$ then it also exists $A \in \mathcal{A}$ such that $\texttt{y-has-coin}(A,c)$. However, these As are not necessarily the same.

(β) Whenever $\texttt{x-bal}(A) \leq \texttt{y-bal}(A)$ then $\texttt{x-has-coin}(A,c) \rightarrow \texttt{y-has-coin}(A,c)$ and whenever $\texttt{y-bal}(A) \leq \texttt{x-bal}(A)$ then $\texttt{y-has-coin}(A,c) \rightarrow \texttt{x-has-coin}(A,c)$.

(γ) For every A such that $\texttt{y-bal}(A) \leq \texttt{x-bal}(A)$ there are precisely $\texttt{x-bal}(A) - \texttt{y-bal}(A)$ many distinct coins such that $\texttt{x-has-coin}(A,c) \wedge \neg\texttt{y-has-coin}(A,c)$. These shall be exactly the coins $c \in \texttt{x-active} \subseteq \texttt{y-active}$ with $\texttt{ind}'(A,c) \in (\texttt{y-bal}(A), \texttt{x-bal}(A)]$.

We now use Lemma 11, to show that such an element exists. Let $s_y \in f(h_y)$ fulfill (1)-(3) of Lemma 11. The properties (α) and (β) of s_y follow directly. For (γ), by (3) of Lemma 11, we know that there exist precisely $\texttt{x-bal}(A) - \texttt{y-bal}(A)$ coins such that $\texttt{x-has-coin}(A,c) \wedge \neg\texttt{y-has-coin}(A,c)$, since by definition of f the cardinalities are as required. These coins c satisfy $c \in \texttt{x-active}$, since $\texttt{x-has-coin}(A,c)$ and s_x is valid. From (1) of Lemma 11 it follows that also $c \in \texttt{y-active}$. Finally, we can assume without loss of generality that these coins are precisley the ones with $\texttt{ind}'(A,c) \in (\texttt{y-bal}(A), \texttt{x-bal}(A)]$. If this were not the case, then $\texttt{ind}'_A$ was replaced by another bijection from $\{c \in \mathcal{C} : \texttt{x-has-coin}(A,c)\}$ onto $[1, \texttt{x-bal}(A)]$ satisfying this constraint.

We can now extend $\texttt{count}'$ and $\texttt{ind}'$ to their full scopes.

Let $\texttt{count} : \mathcal{C} \rightarrow N^+$ be an injective extension of $\texttt{count}'$, with $\texttt{count}(c) \in (\texttt{x-sum}, \texttt{y-sum}]$ precisely for coins $c \in \texttt{y-active} \setminus \texttt{x-active}$. This is possible, since by (α)

$$|\texttt{y-active} \setminus \texttt{x-active}| = \texttt{y-sum} - \texttt{x-sum} .$$

This function thus fulfills the properties $(i) - (iii)$ of Lemma 12.

For $A \in \mathcal{A}$ with $\texttt{x-bal} \geq \texttt{y-bal}$, let $\texttt{ind}_A : \mathcal{C} \rightarrow \mathbb{N}^+$ any injective extension of $\texttt{ind}''_A$. Property (v) is ensured automatically by (γ). For the other addresses, that is A with $\texttt{y-bal}(A) \geq \texttt{x-bal}(A)$, (v) has to be ensured separately. Hence, in this case let $\texttt{ind}_A : \mathcal{C} \rightarrow \mathbb{N}^+$ an injective extension of $\texttt{ind}''_A$ with $\texttt{ind}_A(c) \in (\texttt{x-bal}(A), \texttt{y-bal}(A)]$ if and only if $\texttt{x-has-coin}(A,c)$ and $\neg\texttt{y-has-coin}(A,c)$. This is possible, since by (β)

$$|\{c \in \mathcal{C} : \texttt{y-has-coin}(A,c)\} \setminus \{c \in \mathcal{C} : \texttt{x-has-coin}(A,c)\}| = \texttt{y-bal}(A) - \texttt{x-bal}(A) .$$

Further, (iv) and (vi) also hold for any $A \in \mathcal{A}$ by definitions of $\texttt{ind}_A$ and $\texttt{ind}''_A$.

□

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 Bibliothek WIEN Your knowledge hub

Having these two lemmas at hand we can now state and prove the following result.

Theorem 4. *Given any two valid states*

$$h_o = (\textit{old-bal}, \textit{old-sum}),\ h_n = (\textit{new-bal}, \textit{new-sum}) \in \mathcal{H}\ .$$

There exist a count function $\textit{count} : \mathcal{C} \to \mathbb{N}^+$ *and an index function* $\textit{ind} : \mathcal{A} \times \mathcal{C} \to \mathbb{N}^+$ *as defined in Definition 11 and 12, such that there are*

$$s_o = (\textit{old-active}, \textit{old-has-coin}) \in f(h_o),\ s_n = (\textit{new-active}, \textit{new-has-coin}) \in f(h_n)$$

with

$$\forall c \in \mathcal{C} : \big(\textit{z-active}(c) \leftrightarrow \textit{count}(c) \leq \textit{z-sum}\big) \quad \textit{and} \tag{M5'}$$

$$\forall A \in \mathcal{A}, c \in \mathcal{C} : \big(\textit{z-has-coin}(A, c) \leftrightarrow \textit{ind}(A, c) \leq \textit{z-bal}(A)\big)\ . \tag{m5'}$$

Note that the Definitions 11 and 12 ensure (M1′)-(M4′), (m1′)-(m4′) respectively. The fact that we are considering valid states ensures (I1)-(I2) and (M5′) and (m5′) are stated explicitly. Thus, we prove that we can express any two valid higher-order states h_o and h_n with a mutual count and a mutual index function. In particular, any relation between h_o, h_n that could be expressed in Section 5.1 can still be expressed.

Proof.
The idea of the proof is the following. At first, a pair of counting functions $(\texttt{count}, \texttt{ind})$ is constructed based on h_o and h_n. This is done by chosing specific elements of $f(h_o)$ and $f(h_n)$. Secondly, it is shown that the previously constructed functions count and ind have all the necessary properties.

Given h_o and h_n, we give the one of them with the lower sum the prefix 'x-'. That is, $h_x \in \{h_o, h_n\}$ such that $\texttt{x-sum} = \min\{\texttt{old-sum}, \texttt{new-sum}\}$. The other state gets the prefix 'y-' from now on. Also elements in $f(h_x)$ and $f(h_y)$ will be called accordingly.

Now take an arbitrary element $s_x = (\texttt{x-active}, \texttt{x-has-coin}) \in f(h_x)$. By applying Lemma 12, we can fix a state $s_y = (\texttt{y-has-coin}, \texttt{y-active}) \in f(h_y)$ such that it satisfies (1)-(3) of Lemma 11 and there exist functions $\texttt{count} : \mathcal{C} \to \mathbb{N}^+$, $\texttt{ind}_A\mathcal{C} \to \mathbb{N}^+$ fulfilling $(i) - (vi)$ of Lemma 12. Subsequently, we fix count and $\texttt{ind} : \mathcal{A} \times \mathcal{C} \to \mathbb{N}^+$, with $\texttt{ind}(A, c) := \texttt{ind}_A(c)$ for every $A \in \mathcal{A}$, $c \in \mathcal{C}$.
This finishes the first step, the functions count and ind are defined. That means the functions are constructed based on the specific elements $s_o \in f(h_o)$ and $s_n \in f(h_n)$, but they do not depend on them in any way. They are only functions assigning naturals to coins, pairs of addresses and coins to naturals respectively. We now show count and ind are actually instances of the desired functions described in the theorem.

Injectivity and partial surjectivity constraints from Definition 11 and Definition 12 follow from Lemma 12. Thus, what remains to be shown is that there exists $s_o \in f(h_o)$ 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

$s_n \in f(h_n)$ for which (M5′) and (m5′) hold.
Therefore, consider the $s_x = (\texttt{x-active}, \texttt{x-has-coin})$, $s_y = (\texttt{y-active}, \texttt{y-has-coin})$ from the proof of Lemma 12.

To show (M5′), let $c \in \mathcal{C}$. Then by (*ii*), (*iii*) respectively, in Lemma 12 we have $\texttt{z-active}(c)$ if and only if $\texttt{count}(c) \leq \texttt{z-sum}$.

Similarly for (m5′), let $A \in \mathcal{A}$, $c \in \mathcal{C}$. Then by (*v*), (*vi*) respectively, of Lemma 12 we have $\texttt{z-has-coin}(A, c)$ if and only if $\texttt{ind}(A, c) \leq \texttt{z-bal}(A)$

This concludes the second part and thus proves Theorem 4.

□

Technically speaking, the setting introduced in this section is not a proper encoding of f any more, since not every $s \in f(h)$ is considered. Nevertheless, it provides a sound and complete way of encoding sums, where completeness is given by the following result.

Corollary 2. *Let $h_o = (\texttt{old-bal}, \texttt{old-sum})$, $h_n = (\texttt{new-bal}, \texttt{new-sum}) \in \mathcal{H}$ be any higher-order states, then i) and ii) are equivalent:*

i) The axioms (M1')-(M5'), (m1')-(m2') and (I1)-(I2) are satisfiable for both h_o and h_n at the same time.

ii) $\sum_{A \in \mathcal{A}} \texttt{x-bal}(A) = \texttt{x-sum}$ and $\sum_{A \in \mathcal{A}} \texttt{y-bal}(A) = \texttt{y-sum}$.

Proof. *i*) ⇒ *ii*). From *i*) follows the existance of $s_o, s_n \in \mathcal{F}$ as specified in Lemma 10. From there, we know both $\sum_{A \in \mathcal{A}} \texttt{old-bal}(A) = \texttt{old-sum}$ and $\sum_{A \in \mathcal{A}} \texttt{new-bal}(A) = \texttt{new-sum}$.

ii) ⇒ *i*). Having *ii*), precisely means that h_o and h_n are valid. Thus, we can apply Theorem 4 to conclude the axioms are satisfiable for both h_o and h_o at the same time. □

Note that this result is not specific to the encoding in this section. By using $\texttt{old-ind} = \texttt{ind} = \texttt{new-ind}$ and $\texttt{old-count} = \texttt{count} = \texttt{new-count}$, the same property holds for the naive encoding from Section 5.1.

Theoretically, first-order theorem provers could be able to prove various properties in this encoding. That means for many tasks there is no higher-order reasoning is required. However, provers can only prove very basic properties as shown in Section 5.3. One further adaption for efficiency is presented in the next subsection.

Remark:
It is not possible to transfer this encoding with its results to the software verification setting from Chapter 3. One can consider the reasoning goal from the former chapter with respect to the encoding from this chapter. It would result in:

> **Reasoning goal from Chapter 3, given the encoding from Chapter 5.**
> Given the axioms (I1)-(I2) only for h_o, (M1)-(M5), (m1)-(m5) and the transition formula, then
>
> $$\text{the impact formula holds} \Rightarrow \text{(I1)-(I2) for } h_n \text{ hold}.$$

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

This is problematic, because this statement is not valid. That means (I1)-(I2) are not invariants in this encoding. Hence, this cannot be transferred to the software verification setting. A simple counterexample to the above is the following model:

$$\mathcal{A} = \{A_0, A_1\} \quad \mathcal{C} = \{c_0, c_1\}$$

$$\begin{array}{llll}
\texttt{old-sum} = 1 & \texttt{old-bal}(A_0) = 0 & \texttt{ind}(A_0, c_0) = 1 & \texttt{old-active} = \{c_0\} \\
\texttt{new-sum} = 2 & \texttt{old-bal}(A_1) = 1 & \texttt{ind}(A_0, c_1) = 2 & \texttt{new-active} = \{c_0, c_1\} \\
\texttt{count}(c_0) = 1 & \texttt{new-bal}(A_0) = 1 & \texttt{ind}(A_1, c_0) = 1 & \texttt{old-has-coin} = \{(A_1, c_0)\} \\
\texttt{count}(c_1) = 2 & \texttt{new-bal}(A_1) = 1 & \texttt{ind}(A_1, c_1) = 2 & \texttt{new-has-coin} = \{(A_1, c_0), (A_0, c_0)\}
\end{array}$$

Further Restriction for Efficiency

With the result from Theorem 4, we now aim to further increase efficiency. Studying the intra-block axioms from the previous encoding, one can see that `z-active` and `z-has-coin` are only shortcuts for properties between `z-bal` and `ind`, `z-sum` and `count` respectively. Thus, dropping these relations has neither impact on how the encoding works nor on its expressiveness, it only decreases the search space and hence the encoding's complexity.

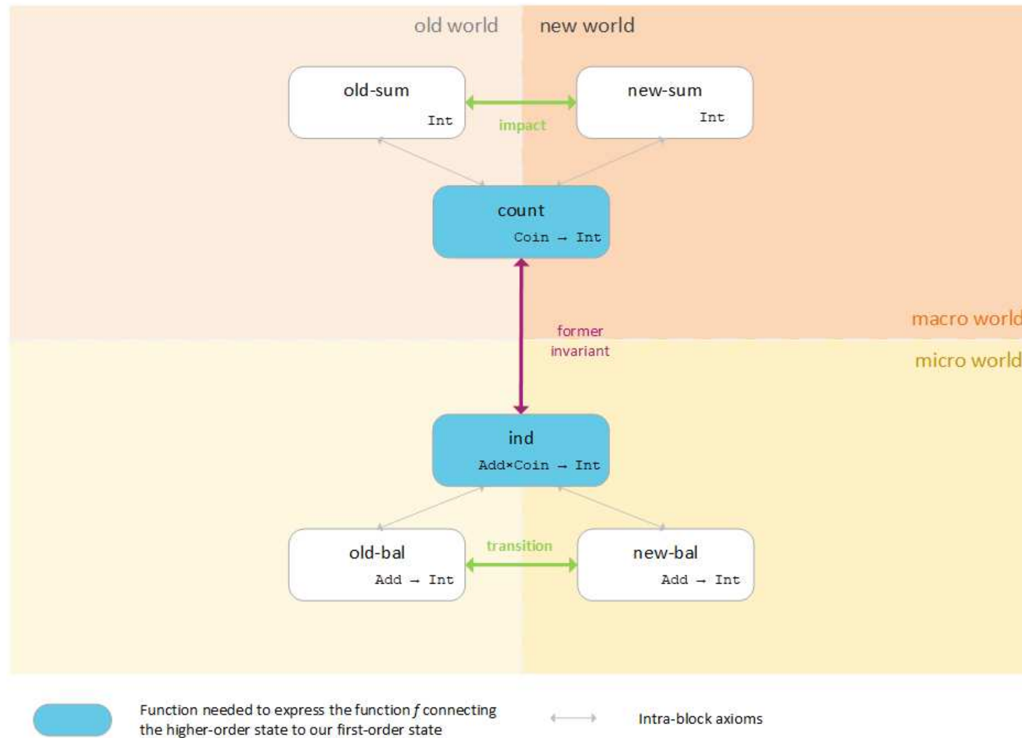

Figure 5.3: Encoding without `active` and `has-coin`.

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

In Figure 5.3, this further simplified setting is illustrated. The adapted axioms are the following.

Axioms in the Further Restricted Encoding

As mentioned above, `z-has-coin` and `z-active` are replaced by equivalent expressions, all the other axioms remain unchanged.

Intra-block axioms for the macro world.

$$\texttt{z-sum} \geq 0 \tag{M1''}$$
$$\forall c \in \mathcal{C} : \texttt{count}(c) > 0 \tag{M2''}$$
$$\forall c_1, c_2 \in \mathcal{C} : \big(\texttt{count}(c_1) = \texttt{count}(c_2) \to c_1 = c_2\big) \tag{M3''}$$
$$\forall n \in \mathbb{Z} : \big(0 < n \wedge n \leq \texttt{x-sum} \to \exists c \in \mathcal{C} : \texttt{count}(c) = n\big) \tag{M4''}$$

All the intra-block axioms listed here are unchanged, only the fifth axioms does not apply any more.

Intra-block axioms for the micro world.

$$\forall A \in \mathcal{A} : \texttt{z-bal}(A) \geq 0 \tag{m1''}$$
$$\forall c \in \mathcal{C}, A \in \mathcal{A} : \texttt{ind}(A, c) > 0 \tag{m2''}$$
$$\forall A \in \mathcal{A}, c_1, c_2 \in \mathcal{C} : \big(\texttt{ind}(A, c_1) = \texttt{ind}(A, c_2) \to c_1 = c_2\big) \tag{m3''}$$
$$\forall A \in \mathcal{A}, n \in \mathbb{Z} : \big(0 < n \wedge n \leq \texttt{z-bal}(A) \to \exists c \in \mathcal{C} : \texttt{z-ind}(A, c) = n\big) \tag{m4''}$$

The changes made are analog to the macro world.

Inter-block axioms connecting the micro and the macro world.
In the former invariants, the dropped relations are replaced by their definitions.

$$\forall c \in \mathcal{C} : (\exists A \in \mathcal{A} : \texttt{ind}(A, c) \leq \texttt{z-bal}(A)) \leftrightarrow \texttt{count}(c) \leq \texttt{z-sum} \tag{I1''}$$
$$\forall A_1, A_2 \in \mathcal{A}, c \in \mathcal{C} : \texttt{ind}(A_1, c) \leq \texttt{z-bal}(A_1) \wedge \texttt{ind}(A_2, c) \leq \texttt{z-bal}(A_2) \to A_1 = A_2 \tag{I2''}$$

Transition and expected impact.
Transition and impact are not affected and can be formulated as before.

5.3 Experimental Results of the f Encodings

In this section, the behaviour of the solvers Vampire and Z3 on the three different encodings is discussed. In Table 5.2 they are referred to as 'naive' for the naive encoding from Section 5.1, 'one count' for the restricted version from Section 5.2 and 'no relations' for the further simplified encoding without the relations `has-coin` and `active`.

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.

Each of the three entries is further split up into the three subversions 'surjective', 'partly surjective' and 'instances of surjectivity'. They only differ by the implementation of the axioms (M4) and (m4) which is assuring the surjectivity of the count and index function in a certain range. The 'surjective' versions have the axioms (M4) and (m4) implemented as they were presented in this work. The 'partly surjective' versions have (M4) and (m4) only in weakened format. That is surjectivity on $[\texttt{old-sum}, \texttt{new-sum}]$, $[\texttt{new-sum}, \texttt{old-sum}]$ respectively and the analogue for `bal`. Finally, the 'instances of surjectivity' only have assured that there exist coins $c \in \mathcal{C}$ such that $\texttt{count}(c) = \texttt{old-sum}$, $\texttt{count}(c) = \texttt{new-sum}$, $\texttt{ind}(A, c) = \texttt{old-bal}(A)$ and $\texttt{ind}(A, c) = \texttt{new-bal}(A)$, for every A.
For the last two ones, the encoding is no longer complete. That means, it may be the case, depending on the transition and impact, that (axioms $\wedge$ transition $\rightarrow$ impact) is no longer proveable, as the axioms have been weakened to much to imply the property we were aiming for.

The tasks listed in Table 5.2 are a transition formula together with the expected impact formula. Although, as mentioned before, this chapter does not apply for software verification, the chosen tasks are similar to the ones from Chapter 3, since also this encoding is tailored to address additive integer tasks, such as adding and subtracting certain numbers from balances. For simplicity, they are named similarly as before. They are the following.

- *'no changes'.* As in Section 5.1 in 'By changing nothing, nothing changes.'
- *'mint'.* One balance is modified by 1. That is:

$$\begin{aligned}&\texttt{new-bal}(A_0) = \texttt{old-bal}(A_0) + 1 \wedge \\ &\forall A \in \mathcal{A} : A \neq A_0 \rightarrow \texttt{old-bal}(A) = \texttt{new-bal}(A)\end{aligned} \tag{transition}$$

$$\texttt{new-sum} = \texttt{old-sum} + 1 \tag{impact}$$

- *'transfer'.* One balance is increased by 1, another one decreased by 1. That is:

$$\begin{aligned}&\texttt{new-bal}(A_0) = \texttt{old-bal}(A_0) + 1 \wedge \\ &\texttt{old-bal}(A_1) = \texttt{new-bal}(A_1) + 1 \wedge \\ &\forall A \in \mathcal{A} : (A \neq A_0 \wedge A \neq A_1) \rightarrow \texttt{old-bal}(A) = \texttt{new-bal}(A)\end{aligned} \tag{transition}$$

$$\texttt{new-sum} = \texttt{old-sum} \tag{impact}$$

- *'swap'.* The balances of two addresses are swapped. That is:

$$\begin{aligned}&\texttt{new-bal}(A_0) = \texttt{old-bal}(A_1) \wedge \\ &\texttt{new-bal}(A_1) = \texttt{old-bal}(A_0) \wedge \\ &\forall A \in \mathcal{A} : (A \neq A_0 \wedge A \neq A_1) \rightarrow \texttt{old-bal}(A) = \texttt{new-bal}(A)\end{aligned} \tag{transition}$$

$$\texttt{new-sum} = \texttt{old-sum} \tag{impact}$$

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.

Note that swapping balances is not an additive modification, but it is listed as it is interesting to see what we can do on other problem types.
In the results of Chapter 3, there were inductive tasks discussed. This is not done here due to the following. For the restricted encoding 'one count' it is not even possible to model all triples of higher-order states with one count and one index function and hence it and 'no relation' cannot be used. Thus, it would be necessary to use the naive encoding in which, as shown below, we cannot even prove very basic transitions. The possibility of increasing a balance by N at once is discussed below and in Chapter 7.

Results

As claimed in Section 5.1, the provers cannot solve a single task for 'naive'. It requires reasoning about all kinds of (`old-count`, `new-count`)- and (`old-ind`, `new-ind`)-pairs. The weakened versions are even satisfiable. The reason is that whenever one of the low numbers does not have a preimage in either of these functions, in the analog of the other world that number can have a preimage. In this way the actual number of coins in an address can be distinct without the balance function recognizing it.

In the restricted version 'one count', both Vampire and Z3 were able to prove the 'no changes task' in all of the surjectivity versions. That shows that it was crucial to unify the count and the index function for this encoding to be useful. Also 'mint' and 'transfer' could be proven by Z3. It is interesting that the partly surjectivity was harder than the full axioms version in the one case, but vice-versa for the other. The reason may be that for 'mint' the sum changes, whereas for 'transfer' it does not.

Lastly, for the 'no relation' encoding, one can see the decreased search space in the time elapsed for the solved tasks. Other than that, the results do not differ much from 'one count'. The only significant improvement is that the Z3 could prove 'transfer' with the full axioms (M4) and (m4).

In general, the weakest version 'instances of surjectivity' of each encoding does not seem to be helpful in Table 5.2. The time elapsed is indeed lower than for the others, but not a single task that was impossible to be solved in other versions, could be proved.
However, while the solvers time out when trying to prove harder additive tasks (e.g. increasing one balance by five, while another one decreases by two) in the stronger versions, Z3 can prove them using the weakest versions, as shown in Table 5.3. One has to be aware, though, that the instances of (M4) and (m4) have to be adapted manually in advance.

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

Version\Task		no changes		mint		transfer		swap	
naive	surjective	Z3	X	Z3	X	Z3	X	Z3	X
		Vampire	X	Vampire	X	Vampire	X	Vampire	X
	partly surj.	sat		sat		sat		sat	
	instances	sat		sat		sat		sat	
one count	surjective	Z3	0.03	Z3	0.22	Z3	X	Z3	X
		Vampire	0.771	Vampire	X	Vampire	X	Vampire	X
	partly surj.	Z3	0.02	Z3	X	Z3	0.20	Z3	X
		Vampire	0.824	Vampire	X	Vampire	X	Vampire	X
	instances	Z3	0.01	Z3	0.02	Z3	0.05	sat	
		Vampire	0.486	Vampire	X	Vampire	X		
no relations	surjective	Z3	0.03	Z3	0.14	Z3	0.73	Z3	X
		Vampire	0.117	Vampire	X	Vampire	X	Vampire	X
	partly surj.	Z3	0.02	Z3	X	Z3	0.44	Z3	X
		Vampire	0.098	Vampire	X	Vampire	X	Vampire	X
	instances	Z3	0.02	Z3	0.02	Z3	0.08	sat	
		Vampire	0.082	Vampire	X	Vampire	X		

Table 5.2: Experimental results using Vampire and Z3. Time (in seconds) is given when the solver proved the (unsat) goal, timeout (after 60 seconds) is indicated by 'X'. The weakened incomplete versions that are satisfiable, are indicated by 'sat', independent from whether the solver could prove it.

This fact shows the problem with adding an arbitrary number N to one account. The solvers cannot do more than an update of one for the stronger versions, while the weakest one cannot even express arbitrary N instances of surjectivity and is thus satisfiable for any set of instances. The experimental results on adding N to one balance indeed led to time out for both Vampire and Z3.

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

Task		Time
Transition	Impact	
$\texttt{new-bal}(A_0) = \texttt{old-bal}(A_0) + 3$ $\texttt{new-bal}(A_1) = \texttt{old-bal}(A_1) - 3$	$\texttt{new-sum} = \texttt{old-sum}$	0.09
$\texttt{new-bal}(A_0) = \texttt{old-bal}(A_0) + 5$ $\texttt{new-bal}(A_1) = \texttt{old-bal}(A_1) - 2$	$\texttt{new-sum} = \texttt{old-sum} + 3$	0.84
$\texttt{new-bal}(A_0) = \texttt{old-bal}(A_0) + 5$ $\texttt{new-bal}(A_1) = \texttt{old-bal}(A_1) - 2$ $\texttt{new-bal}(A_2) = \texttt{old-bal}(A_2) - 3$	$\texttt{new-sum} = \texttt{old-sum}$	2.22

Table 5.3: Harder tasks for 'no relation' in the 'instances of surjectivity' version. Proven by Z3 in the time listed (in seconds).

As 'swap' is a different kind of problem, it does not come as a surprise that neither Z3 nor Vampire can prove it. Anyway, there are also other ways to strengthen the relation between the index and the count functions tailored to fit swaps or permutations, their generalization. For such cases, it comes naturally to ask for the index functions to be permutations of each other as well. That is, $\texttt{old-ind}(A, c) = \texttt{new-ind}(\pi(A), c)$, where $\pi : \mathcal{A} \to \mathcal{A}$ a bijection. The count function can remain mutual. Of course, this is neither a complete encoding, since not every pair of higher-order states can satisfy this, nor is it very sophisticated. Still, it shows the potential of this approach. Both Vampire and Z3 could prove the generalization, that is permutations have the same sum, (Vampire in 0.217 seconds, Z3 in 1.22 seconds) and Vampire could also prove the special case of one swap (in 0.766 seconds).

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.
TU WIEN Bibliothek Your knowledge hub

6 Related Work

We do not aim to overview each automated theorem proving approach in general, but we focus only on the formal verification of smart contracts. Further, work related to our approach that may help to extend our work is discussed. We also present what has been done to reason about sums prior to this thesis.

Formal verification for smart contracts.
It the literature there are various methods and tools aiming to verify certain properties of (specific) smart contracts or to find bugs automatically. Following the categorization in [6], Oyente [15] is a static analysis tool for automated bug finding. It operates on bytecode and supports common security properties such as transaction order dependency, reentrancy and time-stamp dependency. As it is meant to find bugs but not to verify code, it is not crucial for the reasoning to be sound nor complete. Further, these security properties are very different from what we considered in this thesis and lack a semantic characterization. Extensions such as SASC [16] or Majan [17] have been proposed. They add specific patterns and Majan also considers multiple invocations. However, they are still bug-finding techniques that do not aim for soundness or completeness in contrast to our approach.
There is also a lot of research on methods that formally guarantee their results. One of them is ZEUS [18]. It uses symbolic model checking to analyze contracts written in Solidity, by translating the source code to LLVM bitcode via an abstract intermediate language. This approach aims to ensure generic properties that follow from the analysis, whereas we want to prove specific properties that cannot be inferred from code analysis.
Another approach is to use the higher-order theorem prover Isabelle [19]. It can be used to manually prove safety properties of smart contracts. In [20] this is done by over-approximating the original semantics in a sound way. In our work the goal is to prove properties automatically.
Apart from these static analysis strategies, it has also been proposed to monitor predefined security properties dynamically, not statically as we propose. In [21] an efficient online algorithm is presented which discovers executions that are often sources of bugs. Another monitoring tool is proposed in [22]. It surveilles incoming transactions. If a transition is considered dangerous a counter-action is taken, but there is no guarantee that it resolves the problem.

A very recent work (2020) on '*Fantastic bugs and how to find them*' (quote by Mooly Sagiv, alluding to the movie 'Fantastic beasts and how to find them') [23] states that the major challenge to verify smart contracts is expressing the desired properties and not the acutal verification step itself. In [23], some invariants considered useful are presented and the community is encouraged to share invariants as the verification of smart contracts becomes less costly, once a certain number of invariants is available. They claim that they

can be used for many different software versions and platforms.
The work in [23] mentions three steps to verify smart contracts. First, the desired properties have to be specified informally. Then, secondly, they have to be translated to a formal language such as higher-order or first-order logic. And lastly, the code has to be checked. The present thesis addresses the second step.
Further, some informally specified invariants are presented in [23]. One of them is to ensure that 'aggregated values over collections are correctly maintained', this is called 'Aggregated Ledger Integrity'. Others are 'Bounded Supply', that is there cannot be infinite minting, or 'Robustness' which means there shall not be a radical value change for small input changes. It is also mentioned in [23] that these properties can be formalized by using aggregates such as sums. Thus, the development of first-order invariants addressed in our work, fills the gap of formalizing those properties in a language automated theorem provers can handle and hence [23] emphasizes the relevance of our work.

Prior work on aggregates.
The main result the present thesis is based on is from [27]. Even though it does not address smart contracts nor aggregates specifically, it shows a way to prove that two permuted arrays that only differ in one swap of adjacent components have the same sum.
The aim of [27] is to verify relational properties in imperative programs. This was reduced to a validity problem in trace logic. The semantics of programs and the relational properties are encoded in trace logic, which is an instance of many-sorted first-order logic with equality. First-order theorem proving is used to reason about the resulting formulas. Relational properties are properties between different exeutions of programs. That is, if the input in two traces is related in a specific way, the ouputs are related accordingly. The security properties non-interference and robustness have relational character.
The motivating example used in [27], is an easy imparative program in which the sum of an array a is computed in a loop over an integer i by updating a variable hw by $hw + a[i]$. The aim is to show that given a trace t_1 and a trace t_2 with arrays a_1, a_2, respectively, such that there exists an index k with $a_1[k] = a_2[k+1]$ and $a_1[k+1] = a_2[k]$, whereas for every other index $i \notin \{k, k+1\}$ it holds $a_1[i] = a_2[i]$, then the values hw_1, hw_2 are equal at the time of termination.
Since in trace logic not only single transitions but entire programs are considered, each variable and function symbol is modeled by an additional integer-valued parameter n. Let x be a variable in a program, then for the corresponding variable in trace logic, $x(n)$ stands for the value x has after executing line n of the program. For loops another parameter m is added to model the state of the variable x in line n in the m-th iteration of the loop.
The present thesis extends this reasoning about sums to 'arrays' (balance functions in our case) that are related differently. In Chapter 5, we have indeed proven that arbitrary permutations have the same sum using Z3. In Chapter 3 the `mintN` transition uses the idea of adding one more parameter to the relations to generalize the transition `mint`, as it was done in [27]. It is an open challenge to fit the encodings from Chapter 3 and Chapter 5 into the trace logic framework. This combination could improve the performance in proving more involved properties of programs. However, this may lead to a tremendous increase of computational complexity.

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 [26] an SMT module for aggregates is presented. It is based on FO(Agg) which is an extension of first-order logic and aims to address aggregates such as cardinalities, sum and maximum. Note that cardinalities are not considered as aggregates in our work as defined in Section 1.
Low complexity algorithms for satisfiablilty checking are proposed. Further, the presented SMT module can be used in all DPLL-based SAT, SMT and ASP solvers. However, even though the logic FO(Agg) can express aggregates, there is no hint on how to express common properties in this logic. That is, the SMT module does not ensure that the aggregates behave as expected, it only provides a way to express functions that have sets or similar constructs as inputs. While this approach works for the examples presented, it is crucial for the smart contract's case to precisely specify how the aggregates behave and reason about first- and higher-order properties with theories, which is not supported in [26] but handled in this thesis.

Related approaches for future work.
In [25] a methodology for deductive verification based on effectively propositional logic (EPR) is presented. EPR is also known as Bernays-Schönfinkel-Ramsey class and is a decidable fragment of first-order logic. This methodology is used to verify different variants of the distributed protocol Paxos. They proceed as follows. At first the protocols are modeled in full first-order logic. Subsequently, the first-order formulas are translated to EPR. This translation is also checked mechanically. Lastly, the now decidable properties are checked. In fact, extended EPR is used. That means not only formulas with $\exists^*\forall^*$ prefixes are allowed, but also stratified function symbols and quantifier alternation. On a high level a formula ϕ is stratified, if for every pair of sorts $\mathcal{A}$, $\mathcal{B}$ it is not the case that there is both a $\forall a \in \mathcal{A}\ \exists b \in \mathcal{B}$ expression as well as a $\forall b \in \mathcal{B}\ \exists a \in \mathcal{A}$ expression. Note that a function $f : \mathcal{A} \to \mathcal{B}$ implies a hidden quantifier alternation $\forall a \in \mathcal{A}\ \exists b \in \mathcal{B}$.
The main idea is for every pair of sorts $\mathcal{A}$, $\mathcal{B}$ violating stratification to rewrite $\exists b \in \mathcal{B} : \phi$ expressions in the scope of $\forall a \in \mathcal{A}$ by a new relation r and weaken the derived equivalence between them such that the most important properties are maintained but simultaneously the $\forall\exists$ alternation is removed.
While describing the first step in [25], it is explicitly mentioned as a hurdle that the invariants to prove usually include higher-order quantification but that one can partly express higher-order concepts in first-order logic.
Since, to the best of our knowledge, the present work is the first to encode sums of arbitrary length in first-order logic with their full semantics (Chapter 5), the properties verified in [25] do not include reasoning about sums. In particular, in terms of the cited work, the first step of the described method is done already. However, it is not obvious how to translate it to extended EPR, respectively whether it is possible at all. That is our encodings of Chapter 5 are in an undecidable fragment of first-order logic. It is not clear yet whether it is possible to express at least some of the properties in a decidable fragment such as EPR. We assume it is not possible to translate the full encoding to a decidable fragment because of its bijective behaviour.

In [24], a decision algortihm for reasoning about cardinalities in first-order logic is proposed. In order to do so, the syntax of first-order logic is extended to sets. The suggested

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

logic BAPA (Boolean Algebra with Presburger Arithmetic) supports set union, set intersection, set complement, set cardinalities and linear reasoning about integers. As such, this work is interesting for our approach, since, as mentioned before, a crucial point is to correctly reason about cardinalities of the coins.
Nevertheless, we cannot apply the results, since we use uninterpreted function and relation symbols we do not know how to remove, as they are crucial for the encoding, and which are not supported in BAPA.

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.
TU Bibliothek WIEN Your knowledge hub

7 Conclusion and Future Work

In this thesis we present invariants in first-order logic to ensure properties of sums of finite but arbitrary length. This encoding enables the formal verification approaches in the area of smart contracts to reason about various important security properties and may thus prohibit vulnerable contracts to be deployed. As a further consequence, it ensures assets to be securely stored. We also showed that the theorem provers Vampire and Z3 can handle our encoding together with transitions tailored to the smart contracts use case. This provides automated verification and is hence convenient for the user. The proposed encodings are proved sound and complete relative to a translation function.

The second major contribution is the first sound and complete encoding of finite sums in first-order logic. That is, we have developed axioms that ensure a non-negative integer to be the sum of arbitrary many non-negative integers if and only if our axioms hold. Our experimental results can be integrated within software verification tools, such as the one presented in [27]. Unfortunately, swapping the roles of the axioms and the relations between two sums does not lead directly to explicit invariants for sums. It is an interesting direction of future research to adapt our approach accordingly. Generally, all the claims made haven been proven thorougly in a clear and structured mathematical manner as part of this thesis.

For the specific use case of permutations, we achieved a generalization in Section 5.2 of what was presented as motivating example in [27]. This first result was obtained by making slight changes to the restricted f encoding. Using a helper axiom, it could be shown that any pair of permutations has the same sum. It should be possible to adapt or further restrict the index and count function to address the special needs of permutations without using helper axioms. For the example in [27], our generalization will unfortunately not apply directly, since the solver cannot conclude that the stepwisely calculated expression hw happens to be the sum. Since permutations are bijections, it is unlikely that such an adaption will be expressed in a decidable fragment of first-order logic. We will work on this challenge in the near future.

Another aggregate that behaves similarly to the sum it the mean. In order to reason that the mean of two sets or function values is the same, our encodings for sums can directly be used, since the number of values is fixed. In oder to reason about means, non-linear arithmetic is required, which makes the task even harder.

To address the aggregation function minimum, our idea for both the invariants as well as the 'axiomatization' can be adapted. The relations `active` and `has-coin` remain. But now, in order to assure every address has at least m coins, it has to be the case that every address has every active coin and maybe more. Using this approach, the nice intuition with every coin is physically owned by someone does not apply any longer. Also the transitions

have to be changed accordingly while not affecting the number of mistakes. Note that mistakes are defined correspondingly in this context.
For the maximum, a very similar modification can be made. The difference is that here no address shall have any coin that is not active. It is allowed though, to not have a coin which is active.
Probably one of the most difficult aggregates to reason about in first-order logic is the product. It could be handled similarly to our encoding proposed for sums. In particular, the product and its factors could be specified using prime factors. One has to be aware that this suggestion assumes the availability of an encoding to enumerate prime factors in first-order logic. Even then, the value of such an encoding is limited, since only multiplication or division of primes can be modeled.

There are various interesting directions of future research. First, towards software verification, one can extend our work to perform many transitions in one encoding. That could be done by adding further parameters as in [27]. It is also appealing to do more reseach on other aggregates. Some ideas where mentioned above.
Further, it could also be useful to try different versions of our encodings. For example natural numbers can be considered instead of integers. A big part of the arithmetic provided by integers is not used which may spoil the solvers. Thus, it could help to use a simple inductive datatype instead.
Finally, it is an open question, which high-level properties of sums are undecidable in first-order logic. We assume that every sound and complete encoding is undecidable, as we always encountered cycles [25]. That is we could not avoid to have quantifier alternations of both forms $\forall A \in \mathcal{A}\ \exists c \in \mathcal{C}$ and $\forall c \in \mathcal{C}\ \exists A \in \mathcal{A}$.

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

Bibliography

[1] **PeckShield Inc.**
New batchOverflow Bug in Multiple ERC20 Smart Contracts (CVE-2018–10299) *https://medium.com/@peckshield/alert-new-batchoverflow-bug-in-multiple-erc20-smart-contracts-cve-2018-10299-511067db6536*, accessed May 3rd, 2020.

[2] **Michael Grabisch**, Jean-Luc Marichal, Radko Mesiar, *et al.*
Monograph: Aggregation Functions. *Symposium on Intelligent Systems and Informatics*, pages 1-7, 2008.

[3] **Matthew N. O. Sadiku**, Kelechi G. Eze, Sarhan M. Musa.
Smart Contracts: A Primer. *Journal of Scientific and Engineering Research*, pages 538-541, 2018.

[4] **Ethereum Revision 4c1e821e**.
Solidity Documentation *https://solidity.readthedocs.io/en/latest/*, accessed May 5th, 2020.

[5] **Nicola Atzei**, Massimo Bartoletti and Tiziana Cimoli.
A Survey of Attacks on Ethereum Smart Contracts.
International Conference on Principles of Security and Trust, pages 164-186, 2017.

[6] **Ilya Grishchenko**, Matteo Maffei, Clara Schneidewind.
Foundations and Tools for the Static Analysis of Ethereum Smart Contracts. *International Conference on Computer Aided Verification*, pages 51-78, 2018.

[7] **Ilya Grishchenko**, Matteo Maffei, Clara Schneidewind.
A Semantic Framework for the Security Analysis of Ethereum Smart Contracts. *International Conference on Principles of Security and Trust*, pages 243-269, 2018.

[8] **Charles A. R. Hoare.**
An Axiomatic Basis for Computer Programming. *Communications of the Association for Computing Machinery*, pages 576-580, 1969.

[9] **Reiner Hähnle**, Marieke Huisman.
Deductive Software Verification: From Pen-and-Paper Proofs to Industrial Tools. *Computing and Software Science*, pages 345-373, 2019.

[10] **Laura Kovacs** and Andrei Voronkov.
First-Order Theorem Proving and VAMPIRE. *International Conference on Computer Aided Verification*, pages 1-35, 2013.

[11] **Nicolaj Bjørner**, Leonardo de Moura.
Z3: An Efficient SMT Solver. *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337-340, 2008.

[12] **Leo Bachmair**, Harald Ganzinger, David A. McAllester, *et al.*
Resolution Theorem Proving. *Handbook of Automated Reasoning (in 2 volumes)*, pages 19-99, 2001.

[13] **Robert Nieuwenhuis** and Albert Rubio.
Paramodulation-Based Theorem Proving, *Handbook of Automated Reasoning (in 2 volumes)*, pages 371-444, 2001.

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

[14] **Cesare Tinelli**.
A DPLL-based Calculus for Ground Satisfiability Modulo Theories. *European Conference on Logics in Artificial Intelligence*, pages 23-26, 2002.

[15] **Loi Luu**, Duc-Hiep Chu, Hrishi Olickel, *et al.*
Making Smart Contracts Smarter. *Conference on Computer and Communications Security*, pages 254-269, 2016.

[16] **Ence Zhou**, Song Hua, Bingfeng Pi, *et al.*
Security Assurance for Smart Contract. *New Technologies, Mobility and Security*, pages 1-5, 2018.

[17] **Ivica Nikolic**, Aashish Kolluri, Ilya Sergey, *et al.*
Finding the Greedy, Prodigal and Suicidal Contracts at Scale. *arXiv preprint arXiv:1802.06038*, 2018.

[18] **Sukrit Kalra**, Seep Goel, Mohan Dhawan, *et al.*
ZEUS: Analyzing Safety of Smart Contracts. *Network and Distributed System Security Symposium*, pages 1-12, 2018.

[19] **Tobias Nipkow**.
Interactive Proof: Introduction to Isabelle/HOL. *NATO Science for Peace and Security Series, Volume 33: Software Safety and Security*, pages 254-285, 2012.

[20] **Yoichi Hirai**.
Defining the Etheruem Virtual Machine for Interactive Theorem Provers. *International Conference on Financial Cryptography and Data Security*, pages 520-535, 2017.

[21] **Shelly Grossmann**, Ittai Abraham, Guy Golan-Gueta, *et al.*
Online Detection of Effectively Callback Free Objects with Applications to Smart Contracts. *Association for Computing Machinery on Programming Languages*, pages 48:1-48:28, 2017.

[22] **Thomas Cook**, Alex Latham, Jae Hyung Lee.
Dappguard: Active Monitoring and Defense for Solidity Smart Contracts. *MIT Student Project on https://courses.csail.mit.edu/6.857/2017/project/23.pdf*, 2017.

[23] **Thomas Bernardi**, Nurit Dor, Anastasia Fedotov.
WIP: Finding Bugs Automatically in Smart Contracts with Parameterized Invariants. On *https://www.coinresear.ch/*, 2020.

[24] **Viktor Kuncak**, Huu Hai Nguyen and Martin Rinard.
An Algorithm for Deciding BAPA: Boolean Algebra with Presburger Arithmetic. *International Conference on Automated Deduction*,pages 260-277, 2005.

[25] **Oded Padon**, Guiliano Losa, Mooly Sagiv et al.
Paxos made EPR: Decidable Reasoning about Distributed Protocols. *Association for Computing Machinery on Programming Languages*, pages 1-31, 2017.

[26] **Broes De Cat** and Marc Denecker.
DPLL(Agg): an Efficient SMT Module for Aggregates. *Workshop on Logic and Search*, 2010.

[27] **Gilles Barth**, Renate Eilers, Pamina Georgiou et al.
Verifying Relational Properties using Trace Logic. *Formal Methods in Comppputer-Aided Design*, pages 170-178, 2019.

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 Figures

1.1 Simplfied `donate` procedure. 2

3.1 Representation of the sums encoding and the relation between the higher-order setting ('outer circle') and the first-order encoding ('inner circle'). . . 13

5.1 Representation of the naive encoding to make f explicit. 33
5.2 The restricted version of the encoding to make f explicit. 36
5.3 Encoding without `active` and `has-coin`. 44

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

3.1 Overview of which part is considered by theorem provers. 23
3.2 Experimental results using Vampire and Z3. Time (in seconds) is given when the solver proved the (unsat) goal. Otherwise, timeout is indicated by 'X'. 24

5.1 Coins not considered in `y-has-coin`. 39
5.2 Experimental results using Vampire and Z3. Time (in seconds) is given when the solver proved the (unsat) goal, timeout (after 60 seconds) is indicated by 'X'. The weakened incomplete versions that are satisfiable, are indicated by 'sat', independent from whether the solver could prove it. 48
5.3 Harder tasks for 'no relation' in the 'instances of surjectivity' version. Proven by Z3 in the time listed (in seconds). 49

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