אוניברסיטת תל-אביב

תזה ללימודי מוסמך

הסקה מסדר ראשון על סכומים לא חסומים עם יישומים למאפייני בטיחות של אסימוני ERC-20

מגיש: נטע אלעד

מנחים: פרופ׳ מולי שגיב ופרופ׳ שרון שוהם בוכבינדר

חיבור זה הוגש כחלק מהדרישות לקבלת תואר ״מוסמך אוניברסיטה״ – M.Sc. באוניברסיטת תל-אביב

ביה״ס למדעי המחשב ע״ש בלווטניק
הפקולטה למדעים מדויקים ע״ש ריימונד ובברלי סאקלר

שישה באוקטובר, 2020

תקציר

אסימוני ERC 20-מספקים מנשק פשוט לתפעול מטבעות קריפטוגרפיים (crypto-currencies). המנשק תומך בפעולות כגון העברת אסימונים בין כתובות, הטבעת אסימונים חדשים ובחינת היתרות של כל כתובת, כמו גם בחינת סך היצע האסימונים. המנשק עצמו פשוט וקטן יחסית, ועל-כן מהווה מטרה אידיאלית לאימות פורמלי.

עם זאת, הסקה לגבי קוד שנכתב באמצעות מנשק זה איננה טריוויאלית: מספר הכתובות איננו חסום, וביסוס אינווריאנטות גלובליות – כמו שימור סכום היתרות על-ידי פעולות כמו העברה – דורש הסקה מסדרים גבוהים.

אנו מציגים שתי גרסאות של לוגיקה מסדר ראשון מרובת-טיפוסים להסקה לגבי תכניות אשר משתמשות במנשק זה. הראשונה מביניהן היא "לוגיקת סכום" (Sum Logic, SL) אשר מציגה קבועי סכום שמייצגים את הסכומים הבלתי-חסומים של פונקציות יתרה לא מפורשות (uninterpreted). בלוגיקה זו אנו יכולים לתאר ישירות אינווריאנטות לגבי סכום, ויחסיהם עם היתרות.

הגרסה השנייה היא "מטבעות מפורשים עבור סכימה משתמעת" (Explicit Coins for Implicit Summations, ECIS) אשר מקודדת את אסימוני ה-ERC-20 מפורשות. הסקה לגבי סכומים בלוגיקה זו נעשית באמצעות דרישת אקסיומות נוספות אשר משמרות את מערכת היחסים בין הסכומים והיתרות הפרטניות.

Tel Aviv University

Master's Thesis

First-Order Reasoning about Unbounded Sums with Applications to Safety Properties of ERC-20 Tokens

Author:
Neta Elad

Supervisors:
Prof. Mooly Sagiv
& Prof. Sharon Shoham Buchbinder

This thesis was submitted as part of the requirements for receiving a M.Sc. degree from Tel Aviv University

Blavatnik School of Computer Sciense
Faculty of Exact Sciences

October 6, 2020

TEL AVIV UNIVERSITY

Abstract

Faculty of Exact Sciences
Blavatnik School of Computer Sciense

Master's Thesis

First-Order Reasoning about Unbounded Sums with Applications to Safety Properties of ERC-20 Tokens

by Neta Elad

ERC-20[1] tokens provide a simple interface (API) to manipulate crypto-currencies. They support operations like transferring tokens between addresses, minting tokens, and examining the balance of a specific address as well as the total supply of tokens. The interface itself is relatively small and simple, which makes it a perfect target for formal verification.

However, reasoning about code written against this interface is non-trivial: the number of addresses is potentially unbounded, and establishing global invariants like preservation of the sum of the balances by operations like transfer requires higher order reasoning.

We present two variants of many-sorted, first-order logic for reasoning about programs that use this interface. The first is Sum Logic (SL), which introduces sum constants that represent the unbounded sum of some uninterpreted balance function. In SL we can directly state global invariants about sums and their relation to the balances.

The second variant is Explicit Coins for Implicit Summations (ECIS) which encodes the ERC-20 tokens explicitly. Reasoning about sums in ECIS is done by requiring additional axioms to maintain the relationship between the sums and the individual balances.

[1] ERC stands for "Ethereum Request for Comment", and 20 is the proposal identifier within the Ethereum community

Acknowledgements

I would like express my gratitude to all of those who contributed to this work, and helped me during the process of producing it.

First and foremost, to my wonderful supervisors; to Prof. Mooly Sagiv, who guided me closely with great insight and patience; and to Prof. Sharon Shoham Buchbinder, who helped me from afar, and has been an important part of my master's studies throughout.

To Prof. Laura Kovacs and to Sophie Rain from TU Wien with whom I had the pleasure to work. Our interesting discussions were inspiring, and in no small way led to the results achieved in this thesis.

To Prof. Neil Immerman from UMass Amherst, whose expertise and vast knowledge proved invaluable time and time again. I consider myself very lucky for having the opportunity to get to know him and and to learn from him.

Finally, I'd like to thank my office mates in Tel Aviv University. They made the university into a home, and provided helpful remarks for this work, as well as much needed chit-chat breaks.

Table of Contents

Abstract i

Acknowledgements ii

Table of Contents iii

List of Figures v

List of Tables vi

List of Abbreviations vii

1 Introduction 1
- 1.1 Overview . . . 2
- 1.2 Preliminaries . . . 4
  - 1.2.1 Many-Sorted, First-Order Logic . . . 4
  - 1.2.2 Presburger Arithmetic . . . 6
  - 1.2.3 EPR in Many-Sorted Logic . . . 7
  - 1.2.4 2-Counter Machines . . . 7

2 The Sum Logic (SL) 8
- 2.1 Syntax & Semantics . . . 8
  - 2.1.1 Syntax . . . 8
  - 2.1.2 Semantics . . . 8
  - 2.1.3 Encoding ERC-20 in SL . . . 9
- 2.2 A Decidable Fragment of SL . . . 9
  - 2.2.1 Distinct Models . . . 10
  - 2.2.2 Small Models . . . 14
  - 2.2.3 Presburger Reduction . . . 20
    - Outline . . . 20
    - Defining the Transformations . . . 20
- 2.3 Undecidable Sum Logics . . . 29
  - 2.3.1 Outline . . . 29
  - 2.3.2 Formalization . . . 32

3 Theory of Explicit Coins for Implicit Summations (ECIS) 34
- 3.1 Syntax & Semantics . . . 34
  - 3.1.1 Syntax . . . 34
  - 3.1.2 Semantics . . . 34
  - 3.1.3 Axioms . . . 35
- 3.2 Encoding in a Limited Fragment of ECIS . . . 38
  - 3.2.1 Axioms in ECIS ∩ EPR . . . 38
  - 3.2.2 Transitions in ECIS ∩ EPR . . . 38
    - `transferFrom`(x_1, x_2) Transition . . . 38

`throw`(x_1) and `catch`(x_2) Transitions . . . 39
`transfer`(x, c) Transition . . . 40
`burn`(c) Transition . . . 40
3.3 Undecidabiliy Result in ECIS . . . 41
3.3.1 General Idea . . . 41
3.3.2 Encoding Ordering . . . 41
Global Ordering . . . 41
Per-Address, "Local" Ordering . . . 42
3.3.3 Encoding Market Size . . . 42
3.3.4 Encoding Balances . . . 43
3.3.5 Expressing SL using ECIS ∪ Nat . . . 43

4 Conclusions **46**
4.1 Related Work . . . 46
4.2 Future Research . . . 47

References **48**

List of Figures

1.1 Simplified `donate` procedure in ERC-20 2
1.2 Lower-level `donate` procedure 3

2.1 Transition System of a 2-Counter Machine in SL 30

List of Tables

1.1 ERC-20 Token Standard Interface 2

2.1 ERC-20 Token Standard interface in Sum Logic 9
2.2 Transition System of a 2-Counter Machine, Array View 31

List of Abbreviations

ECIS	**E**xplicit **C**oins (for) **I**mplicit **S**ummations
EPR	**E**ffectively **PR**opositional (Logic)
ERC	**E**thereum **R**equest (for) **C**omments
FO	**F**irst **O**rder
PA	**P**resburger **A**rithmetic
PC	**P**rogram **C**ounter
SL	**S**um **L**ogic

Chapter 1

Introduction

Deductive verification [1] is a technique for mathematically proving that all executions of a given program satisfy a correctness condition. We are interested in the special case of formal verification where the correctness condition is expressed by some logical formula ("verification condition").

This thesis addresses the challenge of expressing the verification condition in a way that is amenable to existing automatic techniques. In particular, we are interested in known decidable logics, in which to formalize those verification conditions.

Whereas previous works discuss extending first-order logic with aggregates as counting quantifiers or generalized quantifiers [2, 3, 4], in this thesis we restrict ourselves to the special case of sums over an uninterpreted functions. These sums are encoded using integer constants — and thus are not truly aggregate operators — and we show that their properties are first-order expressible. We prove that this manner of encoding results in decidable fragments of first-order logic, under certain conditions that we lay out.

Our examples are motivated by programs from the domain of smart contracts, and specifically the usage of the ERC-20 Token Standard Interface [5] to manipulate balances. We show two complementary ways to formally describe this interface and related verification conditions.

The first approach — Sum Logic, or SL for short — is an extension of Presburger arithmetic, with additional uninterpreted functions ("balances") and a semantic requirement that relates the sum over the balances with integer constants.

We show that for such an extension with at least 3 uninterpreted functions (and their associated sums) this logic becomes undecidable, thus giving us an upper-bound for the practicality of this approach.

However, for a restricted fragment with 1 uninterpreted function (and no explicit arithmetic) we prove that it is decidable. This encourages us to think that there exist additional decidable fragments of this logic.

The second framework for verifying ERC-20 programs is by explicitly encoding the tokens (which we dub coins) and expressing the higher-level properties using axioms over them. This is formalized using a theory of two-sorted, first-order logic with uninterpreted relations which we call "Explicit Coins for Implicit Summations" (ECIS).

EPR — Essentially Propositional Logic — is a fragment of first-order logic which is known to be decidable [6], but still quite expressible. We examine the intersection of this fragment with ECIS and show that several interesting programs and verification conditions can still be expressed, even in this limited logic.

Finally, we show that when we extend the expressiveness of ECIS, by adding uninterpreted functions — but without the semantic requirement we had in SL — we get a variant of first-order logic which is as expressible as SL, and hence undecidable.

Function	Arguments	Result
totalSupply	-	`uint`
balanceOf	`address`	`uint`
allowance	`address`, `address`	`uint`
transfer	`address`, `uint`	`boolean`
approve	`address`, `uint`	`boolean`
transferFrom	`address`, `address`, `uint`	`boolean`

TABLE 1.1: ERC-20 Token Standard Interface

```
donor:      address
receivers:  address[]
amount:     uint
---------------------------------------------------------------------

for each receiver in receivers:
    transferFrom(donor, receiver, amount)
---------------------------------------------------------------------
# Post-conditions
assert totalSupply() == totalSupply'()
for each receiver in receivers:
    assert balanceOf'(receiver) >= amount
```

FIGURE 1.1: Simplified `donate` procedure in ERC-20

It is important to note that throughout this thesis we regard sums as a special type of integer constants, and not as a general-purpose, higher-order operation. Moreover, the decidability results in this work further restrict the logics that we discuss. The fact that we can express these sums using many-sorted first-order logic — as in Section 3.3 — is therefore not counter-intuitive, and in complete agreement with previous results [2], that show logics with aggregates (or counting in particular) are more expressive than first-order logic.

1.1 Overview

The ERC-20 Token Standard interface consists of three data-types: `address`, `uint` and `boolean`; and six functions: "totalSupply", "balanceOf", "allowance", "transfer", "approve" and "transferFrom" (see Table 1.1).

The functions that return a `boolean` indicate success/failure, and are considered transitions, whereas the functions that return a `uint` represent the current state. Figure 1.1 depicts an example procedure for donations. We use primed versions of the API functions in the post-conditions to denote the state after the transition.

We focus on the "totalSupply", "balanceOf", "transfer" and "transferFrom" functions, and on verification conditions that arise in programs that use this interface. Specifically, we consider formulas that assert some state of the sum and the balances, and the relation between the states before and after a transition.

We show how to encode these operations and conditions in the two variants of first-order logic that we describe in Chapters 2 and 3. When encoding, we sometimes consider a lower-level description of the API (see Figure 1.2). Note that in the lower-level description, "balance" is used as a mutable map, which differs from the API "balanceOf" function.

```
donor:      address
receivers:  address[]
amount:     uint
balance:    address -> uint

---

for each receiver in receivers:
    # simplified low-level "transferFrom"
    balance[receiver] += amount
    balance[donor] -= amount

---

# Post-conditions
assert totalSupply() == totalSupply'()
for each receiver in receivers:
    assert balance'[receiver] >= amount
```

FIGURE 1.2: Lower-level `donate` procedure

In Section 2.1 we use a more direct approach, where uninterpreted functions encode the balances. We use multiple functions to represent different, unrelated "markets", or a single market before and after a transition occurs.

In addition, we have integer constants that we semantically require to be equal to the sum of the uninterpreted function over the unbounded *Address* space. This is akin to a higher-order operation, although of a very basic form.

By proving a non-trivial *small model property*, we are able to reduce this extended first-order logic back to Presburger arithmetic. The reduction outlines a decision procedure for SL. We prove this decidability condition in Section 2.2, and show a specific fragment that holds that condition.

In Section 2.3 we present an encoding of the halting problem for 2-counter machines in a fragment of SL, thus proving SL to be undecidable in some cases. The proof shows that three uninterpreted functions in SL are enough to make it undecidable, but also has an interesting corollary that any two-sorted extension to Presburger arithmetic with two uninterpreted functions, and a *size operation* is undecidable.

Section 3.1 shows a complementary way to express the higher-order setting of unbounded sums, and builds it up from a theory of discrete *Coin* elements. By using several axioms, specifically crafted for this type of programs, we can describe and maintain important properties of the code.

We explore the expressiveness of the decidable intersection of ECIS and EPR in Section 3.2, and show what kind of operations and verification conditions can be encoded in it.

Section 3.3 extends ECIS with a *Nat* sort, constants thereof, and uninterpreted functions, all of which in order to encode sums and balances. In contrast with SL, ECIS does not impose a semantic requirement that the sums and balances associate correctly. Instead, we rely on the axioms of the theory to express this requirement. We explain how this extension is as expressible as SL, and thus undecidable for the same cases as SL.

Finally, Chapter 4 summarizes the results presented here, surveys related work in this field, and concludes with implications and possible future research.

1.2 Preliminaries

1.2.1 Many-Sorted, First-Order Logic

Throughout this thesis we use standard many-sorted, first-order logic.

Definition 1.2.1. A sorted first-order vocabulary Σ consists of sorted constant symbols $c_1, \ldots, c_l$, sorted function symbols $f_1^{r_1}, \ldots, f_m^{r_m}$, and sorted relation symbols $R_1^{\tilde{r}_1}, \ldots, R_n^{\tilde{r}_n}$, where $r_1, \ldots, r_m, \tilde{r}_1, \ldots, \tilde{r}_n$ indicate the arity of the function and relation symbols respectively.

Definition 1.2.2. Given a vocabulary Σ, a term t over Σ is defined inductively:

$$
\begin{array}{rcll}
\langle t \rangle & \triangleq & x : \boldsymbol{s} & \text{where } x \text{ is a free variable of sort } \boldsymbol{s} \\
& | & c_i & \text{where } c_i \text{ is a constant symbol} \\
& | & f_j^{r_j}(t_1, \ldots, t_{r_j}) & \text{where } f_j \text{ is a function symbol} \\
& & & \quad \text{and } t_1, \ldots, t_{r_j} \text{ are terms}
\end{array}
$$

A term with no free variables is called a *ground term.*

Definition 1.2.3. Given a vocabulary Σ, a formula φ over Σ is defined inductively:

$$
\begin{array}{rcll}
\langle \varphi \rangle & \triangleq & t_1 \approx t_2 & \text{where } t_1, t_2 \text{ are terms} \\
& | & R_k^{\tilde{r}_k}(t_1, \ldots, t_{\tilde{r}_k}) & \text{where } R_k \text{ is a relation symbol} \\
& & & \quad \text{and } t_1, \ldots, t_{\tilde{r}_k} \text{ are terms} \\
& | & \neg \varphi' & \text{where } \varphi' \text{ is a formula} \\
& | & \varphi_1 \vee \varphi_2 & \text{where } \varphi_1, \varphi_2 \text{ are formulas} \\
& | & \varphi_1 \wedge \varphi_2 & \text{where } \varphi_1, \varphi_2 \text{ are formulas} \\
& | & \varphi_1 \rightarrow \varphi_2 & \text{where } \varphi_1, \varphi_2 \text{ are formulas} \\
& | & \varphi_1 \leftrightarrow \varphi_2 & \text{where } \varphi_1, \varphi_2 \text{ are formulas} \\
& | & \forall x : \boldsymbol{s}. \varphi' & \text{where } \varphi' \text{ is a formula} \\
& | & \exists x : \boldsymbol{s}. \varphi' & \text{where } \varphi' \text{ is a formula}
\end{array}
$$

A formula with no free variables is called a *sentence* or a *closed formula.*

We always assume terms admit to sort restrictions, that formulas are syntactically valid, and that the arguments to relations and function agree with their arity. We omit arity superscripts from functions and relations, and sort markers from quantifiers, when they are clear from context.

Definition 1.2.4. A structure $\mathcal{A}$ for a vocabulary Σ is a pair $(\mathcal{D}, \mathcal{I})$ where $\mathcal{D}$ (the *domain* of the structure) maps each sort $\boldsymbol{s}$ of Σ to some set $\mathcal{D}(\boldsymbol{s})$; and $\mathcal{I}$ (the *interpretation* of the structure) maps each symbol in Σ to a corresponding object:

1. Each constant symbol c_i of sort $\boldsymbol{s}$ to an element $\mathcal{I}(c_i) \in \mathcal{D}(\boldsymbol{s})$.
2. Each function symbol $f_j^{r_j}$ of sort $(\boldsymbol{s_1}, \ldots, \boldsymbol{s_{r_j}}) \rightarrow \boldsymbol{s}$ to a function $\mathcal{I}(f_j^{r_j}) \in \mathcal{D}(\boldsymbol{s})^{\mathcal{D}(\boldsymbol{s_1}) \times \cdots \times \mathcal{D}(\boldsymbol{s_{r_j}})}$.
3. Each relation symbol $R_k^{\tilde{r}_k}$ of sort $(\boldsymbol{s_1}, \ldots, \boldsymbol{s_{\tilde{r}_k}})$ to a relation $\mathcal{I}(R_k^{\tilde{r}_k}) \subseteq \mathcal{D}(\boldsymbol{s_1}) \times \cdots \times \mathcal{D}(\boldsymbol{s_{\tilde{r}_k}})$.

Definition 1.2.5. We denote the set of all structures for a vocabulary Σ as $\mathrm{STRUCT}[\Sigma]$.

Notation. For any symbol in the vocabulary we sometimes write its interpretation in some structure with a superscript. E.g. for a structure $\mathcal{A} = (\mathcal{D}, \mathcal{I})$, $c_i^{\mathcal{A}} = \mathcal{I}(c_i)$.

Definition 1.2.6. We extend the definition of an interpretation $\mathcal{I}$ to all ground terms inductively:

$$\mathcal{I}(f_j^{r_j}(t_1,\ldots,t_{r_j})) \triangleq \Big[\mathcal{I}(f_j^{r_j})\Big]\big(\mathcal{I}(t_1),\ldots,\mathcal{I}(t_{r_j})\big)$$

Definition 1.2.7. Given a structure $\mathcal{A} = (\mathcal{D},\mathcal{I})$ for a vocabulary Σ, an assignment Δ maps any free variable x of sort $\boldsymbol{s}$ into an element $\Delta(x) \in \mathcal{D}(\boldsymbol{s})$.

Definition 1.2.8. Given a structure $\mathcal{A} = (\mathcal{D},\mathcal{I})$ for a vocabulary Σ, a partial assignment Δ maps some free variables $x_1,\ldots,x_r$ of sorts $\boldsymbol{s_1},\ldots,\boldsymbol{s_r}$ into elements $\Delta(x_i) \in \mathcal{D}(\boldsymbol{s_i})$.

We can explicitly write the partial assignment as

$$[\Delta(x_1)/x_1,\ldots,\Delta(x_r)/x_r]$$

Notation. We use the same notation $[e_1'/e_1,\ldots,e_i'/e_i,\ldots]$ for syntactical substitutions, where e_i, e_i' are arbitrary expressions, and we replace each occurrence of e_i with e_i'.

Definition 1.2.9. We can override an assignment Δ with a partial assignment $\Delta' = [\alpha_1/x_1,\ldots,\alpha_r/x_r]$:

$$\big[\Delta\Delta'\big](x) \triangleq \begin{cases} \alpha_i & \text{if } x = x_i \in \{x_1,\ldots,x_r\} \\ \Delta(x) & \text{otherwise} \end{cases}$$

Definition 1.2.10. Given an assignment Δ, we extend the definition of an interpretation $\mathcal{I}$ to all terms inductively:

$$\begin{array}{lcl} \mathcal{I}_\Delta(x) & \triangleq & \Delta(x) \\ \mathcal{I}_\Delta(c_i) & \triangleq & \mathcal{I}(c_i) \\ \mathcal{I}_\Delta\Big(f_j^{r_j}(t_1,\ldots,t_{r_j})\Big) & \triangleq & \Big[\mathcal{I}(f_j^{r_j})\Big]\big(\mathcal{I}_\Delta(t_1),\ldots,\mathcal{I}_\Delta(t_{r_j})\big) \end{array}$$

Definition 1.2.11. Let φ be some formula over a vocabulary Σ. Given a structure $\mathcal{A} \in \mathrm{STRUCT}[\Sigma]$ and an assignment Δ, we define when $\mathcal{A},\Delta$ satisfy φ ($\mathcal{A},\Delta \vDash \varphi$) inductively:

$$\begin{array}{lcl} \mathcal{A},\Delta \vDash t_1 \approx t_2 & \triangleq & \mathcal{I}_\Delta(t_1) = \mathcal{I}_\Delta(t_2) \\ \mathcal{A},\Delta \vDash R_k^{\tilde{r}_k}(t_1,\ldots,t_{\tilde{r}_k}) & \triangleq & (\mathcal{I}_\Delta(t_1),\ldots,\mathcal{I}_\Delta(t_{\tilde{r}_k})) \in \mathcal{I}(R_k^{\tilde{r}_k}) \\ \mathcal{A},\Delta \vDash \neg\varphi' & \triangleq & \mathcal{A},\Delta \nvDash \varphi' \\ \mathcal{A},\Delta \vDash \varphi_1 \square \varphi_2 & \triangleq & \mathcal{A},\Delta \vDash \varphi_1 \quad \square \quad \mathcal{A},\Delta \vDash \varphi_2 \\ & & \qquad \text{where } \square \in \{\vee,\wedge,\rightarrow,\leftrightarrow\} \\ \mathcal{A},\Delta \vDash \forall x : \boldsymbol{s}.\varphi' & \triangleq & \mathcal{A},\Delta[\alpha/x] \vDash \varphi' \\ & & \qquad \text{for all } \alpha \in \mathcal{D}(\boldsymbol{s}) \\ \mathcal{A},\Delta \vDash \exists x : \boldsymbol{s}.\varphi' & \triangleq & \mathcal{A},\Delta[\alpha/x] \vDash \varphi' \\ & & \qquad \text{for some } \alpha \in \mathcal{D}(\boldsymbol{s}) \end{array}$$

We call $(\mathcal{A},\Delta)$ a model for φ.

Notation. We denote $\mathcal{A} \vDash \varphi$ if for all assignments Δ, $\mathcal{A},\Delta \vDash \varphi$, and we reserve this notation to closed formulas (*sentences*). In this case, we simply call $\mathcal{A}$ a model for the sentence φ.

Notation. We write $\mathcal{A} \vDash \varphi[\alpha_1/x_1,\ldots,\alpha_r/x_r]$ if for any assignment Δ,

$$\mathcal{A},\Delta[\alpha_1/x_1,\ldots,\alpha_r/x_r] \vDash \varphi$$

Definition 1.2.12. A formula φ is said to be satisfiable if there exist some structure $\mathcal{A}$, assignment Δ such that $\mathcal{A}, \Delta \vDash \varphi$.

Definition 1.2.13. A fragment FRAG over a sorted first-order vocabulary Σ is the set of all formulas φ over any vocabulary Σ' that is a subset of Σ. Moreover, FRAG may impose additional syntactic or semantic restrictions over the formulas that are within it.

Definition 1.2.14. The satisfiability problem for a fragment FRAG of first-order logic with vocabulary Σ is determining whether any given formula $\varphi \in$ FRAG is satisfiable.

We say that a fragment of first-order logic is decidable if its satisfiability problem is decidable.

Definition 1.2.15. Two formulas φ, ψ over vocabularies Σ, Σ' (respectively) are said to be equisatisfiable if φ is satisfiable $\iff$ ψ is satisfiable. I.e.,

$$\exists \mathcal{A} \in \text{STRUCT}[\Sigma], \Delta \text{ s.t. } \mathcal{A}, \Delta \vDash \varphi \iff \exists \mathcal{A}' \in \text{STRUCT}\left[\Sigma'\right], \Delta' \text{ s.t. } \mathcal{A}', \Delta' \vDash \psi.$$

Definition 1.2.16. Two formulas φ, ψ over some vocabulary Σ are said to be equivalent if for any structure $\mathcal{A} \in \text{STRUCT}[\Sigma]$, any assignment Δ,

$$\mathcal{A}, \Delta \vDash \varphi \iff \mathcal{A}, \Delta \vDash \psi$$

Definition 1.2.17. A formula φ is said to be in *prenex normal form* when it is written as a string of quantifiers, called the *prefix*, followed by a quantifier-free formula, called the *matrix*.

Claim 1.2.18. Every first-order formula has an equivalent formula in prenex normal form. Moreover, the translation is computable [7].

1.2.2 Presburger Arithmetic

Presburger arithmetic (PA) is the decidable [8] first-order theory of natural numbers with addition. I.e. it has the vocabulary $\Sigma_{\text{Presburger}} = \left(0, 1, c_1, \ldots, c_l, +^2\right)$, and all of the constants $0, 1, c_i$ are of *Nat* sort.

Definition 1.2.19. A structure $\mathcal{A} = (\mathcal{D}, \mathcal{I}) \in \text{STRUCT}[\Sigma_{\text{Presburger}}]$ is called a *Standard Model of Arithmetic* when $\mathcal{D}(Nat) = \mathbb{N}$ and $\mathcal{I}(+^2)$ is interpreted naturally.

The vocabulary can be extended to have a total order relation so we get $\Sigma^*_{\text{Presburger}} = \left(0, 1, +^2, <^2\right)$, where $<^2$ is interpreted naturally for Standard Models of Arithmetic. We usually write the symbols $+^2, <^2$ in infix notation: $t_1 + t_2, t_1 < t_2$.

Remark. We include zero in the naturals: $0 \in \mathbb{N}$. We write $\mathbb{N}^+$ to explicitly exclude zero.

Claim 1.2.20. Given a formula over $\Sigma^*_{\text{Presburger}}$, it is possible to decide if it is satisfiable in the theory of Presburger Arithmetic. Moreover, if it is decidable, we can construct a *Standard Model of Arithmetic* for it [8].

Put differently, if PA is the set of all formulas in the theory of Presburger Arithmetic, then $\varphi \in \text{PA} \iff \varphi$ has a Standard Model of Arithmetic.

1.2.3 EPR in Many-Sorted Logic

Definition 1.2.21. Let ψ be some first-order formula over vocabulary Σ in prenex normal form. We define the *sort dependency graph* for it as a directed graph with the sorts of the vocabulary as the vertices, and the following edges:

1. For each function symbol $f_j^{r_j}$ that appears in φ of sort $\left(\boldsymbol{s_1}, \ldots, \boldsymbol{s_{r_j}}\right) \to \boldsymbol{s}$ we have r_j edges $\boldsymbol{s_i} \to \boldsymbol{s}$.

2. For each existential quantifier in the prefix of sort $\boldsymbol{s}$ and each universal quantifier of sort $\boldsymbol{s'}$ that appears *before* it, we have an edge $\boldsymbol{s'} \to \boldsymbol{s}$.

Definition 1.2.22. A first-order formula φ is said to be in EPR[2] if there exists some equivalent formula ψ in prenex normal form, such that ψ's sort dependency graph has no cycles.

Claim 1.2.23. The fragment of EPR is decidable [6].

1.2.4 2-Counter Machines

A 2-counter machine is an abstract machine that has two general-purpose registers, as well as a program counter (sometimes called state register).

These machines are simple enough that we can encode their entire transition system in a quantifier-free formula over Presburger arithmetic with 6 free variables (one for the content of each register before and after the transition).

However, since 2-counter machines are Turing equivalent [9], the halting problem for them is undecidable. We use this fact to prove undecidability results in this thesis.

[2] Also known as Bernays–Schönfinkel or Bernays–Schönfinkel-Ramsey class

Chapter 2

The Sum Logic (SL)

2.1 Syntax & Semantics

In its most general form, SL is an extension of Presburger arithmetic, adding a new *Address* sort, and uninterpreted functions between the *Address* sort and the *Nat* sort, as well as a *Nat* constant for each of those functions, which represent the associated sum of each function, taken over the entire domain of the *Address* sort.

2.1.1 Syntax

Definition 2.1.1 (Sum vocabulary)**.** A vocabulary

$$\Sigma_{+,<}^{l,m,n} = \left(a_1, \ldots, a_l, b_1^1, \ldots, b_m^1, c_1, \ldots, c_n, s_1, \ldots, s_m, 0, 1, +^2, <^2\right)$$

where

- We have in mind two sorts: *Address* and *Nat*.
- The constants $a_1, \ldots, a_l$ are of *Address* sort.
- $b_1^1, \ldots, b_m^1$ are unary function symbols from *Address* to *Nat* ("balances").
- The constants $c_1, \ldots, c_n, s_1, \ldots, s_m$ and $0, 1$ are of *Nat* sort.
- $+^2$ is a binary function in *Nat*.
- $<^2$ is a binary relation in *Nat*.

The pair $\left(b_j^1, s_j\right)$ is called the j^{th} market, and we call the constant s_j the *associated sum* of the function b_j^1 (for any $j \in [1, m]$). Alternatively, we use the symbol s_b to represent the associated sum of the function b.

Remark. Unless stated otherwise, we always restrict ourselves to *universal sentences* over Sum vocabularies, with quantification only over the *Address* sort.

Notation. When the cardinalities of the vocabulary are clear from context, as well as the available binary functions $+, <$, we simply denote the vocabulary as Σ.

Conversely, when we want to explicitly state that some binary function is not available (in some fragment of SL), we denote it as $\Sigma_{\not{+},\not{<}}^{l,m,n}$.

2.1.2 Semantics

Definition 2.1.2 (Sum structure)**.** Let Σ be a Sum vocabulary. We usually write a structure $\mathcal{A} = (\mathcal{D}, \mathcal{I}) \in \text{STRUCT}[\Sigma]$ as a tuple

$$\mathcal{A} = \left(A, \mathbb{N}, a_1^{\mathcal{A}}, \ldots, a_l^{\mathcal{A}}, b_1^{\mathcal{A}}, \ldots, b_m^{\mathcal{A}}, c_1^{\mathcal{A}}, \ldots, c_n^{\mathcal{A}}, s_1^{\mathcal{A}}, \ldots, s_m^{\mathcal{A}}, 0, 1, +, <\right)$$

Function	Encoding
totalSupply	s or s'
balanceOf(a)	$b(a)$ or $b'(a)$
transfer(a, v)	$b'(a) \approx b(a) + v$
transferFrom(f, t, v)	$b'(t) \approx b(t) + v \wedge b(f) \approx b'(f) + v$

TABLE 2.1: ERC-20 Token Standard interface in Sum Logic

where $A = \mathcal{D}(Address)$ is some *finite*[3] set (possibly empty if $l = 0$); $a_i^{\mathcal{A}} = \mathcal{I}(a_i) \in A$; $b_j^{\mathcal{A}} = \mathcal{I}(b_j^1) \in \mathbb{N}^A$; and $c_k^{\mathcal{A}} = \mathcal{I}(c_k), s_j^{\mathcal{A}} = \mathcal{I}(s_j) \in \mathbb{N}$.

We always assume that $\mathcal{D}(Nat) = \mathbb{N}$, and that $0, 1, +^2$ and $<^2$ are interpreted naturally. For brevity, we omit them when describing Sum structures.

Definition 2.1.3 (Sum model)**.** Let φ be a first-order formula over a Sum vocabulary Σ. A Sum model for φ is a Sum structure $\mathcal{A} \in \text{STRUCT}[\Sigma]$ such that $\mathcal{A} \vDash \varphi$. In addition, we require that for each $j \in [1, m]$,

$$s_j^{\mathcal{A}} = \sum_{a \in A} b_j^{\mathcal{A}}(a). \qquad \text{(Sum property)}$$

We will denote $\mathcal{A} \vDash_{\text{SL}} \varphi$ to mean that $\mathcal{A} \vDash \varphi$ (as a many-sorted, first-order formula), and that $\mathcal{A}$ holds the Sum property. When it is clear from context, we will omit the SL subscript.

2.1.3 Encoding ERC-20 in SL

Given a sum vocabulary of $\Sigma^{l,2,n}$ we can trivially encode the ERC-20 operations, simply by formalizing the balance manipulations before/after a transition. We denote the balance functions as b, b' and their associated sums as s, s'. See Table 2.1 for the encoding of ERC-20.

2.2 A Decidable Fragment of SL

The general method of proving decidability for a fragment of SL is by showing a reduction to Presburger arithmetic. We do this by encoding the Sum Logic extensions with regular Presburger constructs. The crux of the proof relies on two properties:

1. Distinctness: the *Address* constants a_i represent distinct elements in the domain $\mathcal{D}(Address)$. This restriction is somewhat unnatural, but we show that for each vocabulary and formula that has a model, there exists an equisatisfiable formula over a different vocabulary that has a *distinct* model.

2. Smallness: given a formula φ, if it is satisfiable, then there exists a model where $|\mathcal{D}(Address)| \leq \kappa(|\varphi|)$, where $|\varphi|$ is the length of φ, and $\kappa(\cdot)$ is some computable function. This property only holds for some fragments of SL.

[3] In fact, we need only to require that the set of addresses with non-zero balances $\{\alpha \in \mathcal{D}(Address) \mid \forall j. b_j^{\mathcal{A}}(\alpha) > 0\}$ be finite. Except for addresses that are referred by an *Address* constant, we can always discard all zero-balance addresses from a model. Thus, we might as well limit ourselves to finite sets of addresses.

2.2.1 Distinct Models

Observation 2.2.1. For any set X and any partition P thereof, it holds that $|P| \leq |X|$.

Definition 2.2.2 (Partition-induced function)**.** Let P be a partition of a finite set X of size l. $P = \{A_1, \ldots, A_{l'}\}$ where $l' \leq l$.

We define the partition-induced function $f_P(x)$ (for any $x \in X$) as the index i such that $A_i \in P$ and $x \in A_i$.

For short, we denote $f_P(x)$ as $P(x)$.

Definition 2.2.3 (Function-induced equivalence class)**.** Let f be some function over some set X. We define the function-induced equivalence class for each $x \in X$ as

$$[x]_f \triangleq \big\{x' \in X \mid f(x') = f(x)\big\}.$$

Definition 2.2.4 (Function-induced partition)**.** Let f be some function defined over some set X. We define the function-induced partition P_f as

$$P_f \triangleq \{[x]_f \mid x \in X\}.$$

Definition 2.2.5 (Partitioning Sum terms by P)**.** Let t be some term over a Sum vocabulary $\Sigma = \Sigma^{l,m,n}$ (with l *Address* constants) and let P be some partition of $\{a_1, \ldots, a_l\}$.

We define a transformation $\tau_P(t)$ inductively as a term over a Sum vocabulary $\Sigma_P = \Sigma^{l',m,n}$ with $l' = |P| \leq l$ *Address* constants:

$$\tau_P(t) \triangleq \begin{cases} a_{P(a_i)} & \text{if } t = a_i \\ x_i & \text{if } t = x_i \text{ of sort } \mathit{Address} \\ s_j & \text{if } t = s_j \\ b_j(\tau_P(t_1)) & \text{if } t = b_j(t_1) \text{ where } t_1 \text{ is some } a_i \text{ or } x_i \\ \tau_P(t_1) + \tau_P(t_2) & \text{if } t = t_1 + t_2 \end{cases}$$

Definition 2.2.6 (Partitioning a Sum formula by P)**.** We naturally extend the terms transformation τ_P to formulas.

Observation 2.2.7. For any Sum vocabulary Σ, $\Sigma_P \subseteq \Sigma$, since $l' \leq l$. Therefore, for any formula φ in some fragment FRAG of SL, $\tau_P(\varphi) \in$ FRAG as well.

Definition 2.2.8 (Distinct Structures)**.** A Sum structure $\mathcal{A}$ is considered *distinct* when $\left|\{a_1^{\mathcal{A}}, \ldots, a_l^{\mathcal{A}}\}\right| = l$. I.e. the l *Address* constants represent l distinct elements in $\mathcal{D}(\mathit{Address})$.

Theorem 2.2.9. Let φ be some Sum formula over Σ. φ has a model $\iff$ there exists some partition P of $\{a_1, \ldots, a_l\}$ such that $\tau_P(\varphi)$ has a *distinct* model.

Proof of **Theorem 2.2.9**

Part 1: If φ has a model, then there exists some partition P such that $\tau_P(\varphi)$ has a *distinct* model ($\Rightarrow$)

Let $\mathcal{A}$ be some model of φ and let f be the mapping from $\{a_1, \ldots, a_l\}$ to A, i.e

$$f(a_i) \triangleq a_i^{\mathcal{A}}$$

Let P be the partition (of size l') induced by f and we construct a distinct model

$$A' = \left(A, a'_1, \ldots, a'_{l'}, b_1^{\mathcal{A}}, \ldots, b_m^{\mathcal{A}}, c_1^{\mathcal{A}}, \ldots, c_n^{\mathcal{A}}, s_1^{\mathcal{A}}, \ldots, s_m^{\mathcal{A}}\right)$$

for $\tau_P(\varphi)$, where A, $b_1^{\mathcal{A}}, \ldots, b_m^{\mathcal{A}}$, $c_1^{\mathcal{A}}, \ldots, c_n^{\mathcal{A}}$, and $s_1^{\mathcal{A}}, \ldots, s_m^{\mathcal{A}}$ are taken from $\mathcal{A}$.

For every $i' \in [1, l']$, $a'_{i'}$ is defined as $a'_{i'} = a_i^{\mathcal{A}}$ for some $i \in [1, l]$ such that $P(a_i) = i'$.

Remark. The choice of i is unimportant, since for any two indices i_1, i_2, if $P(a_{i_1}) = i' = P(a_{i_2})$ then by definition of P, $a_{i_1}^{\mathcal{A}} = a_{i_2}^{\mathcal{A}}$.

Observation 2.2.10. $\mathcal{A}'$ is distinct and holds the sum property.

Claim 2.2.11. For any closed term t over Σ, $\mathcal{I}(t) = \mathcal{I}'(\tau_P(t))$ (i.e. the interpretation of t in $\mathcal{A}$ equals to the interpretation of $\tau_P(t)$ in $\mathcal{A}'$).

Proof. Since $\tau_P(t) = t$ for all terms except terms containing a_i, and since $\mathcal{A}'$ is identical to $\mathcal{A}$ except for *Address* constants, we only need to consider this kind of terms.

Moreover, since τ_P is defined inductively, it suffices to prove the claim for the basis terms a_i.

Let $t = a_i$ for some $i \in [1, l]$, and let $i' = P(a_i)$:

$$\begin{aligned}
\mathcal{I}'(\tau_P(t)) &= \mathcal{I}'(\tau_P(a_i)) \\
&= \mathcal{I}'(a_{i'}) \\
&= a'_{i'} \\
&= a_i^{\mathcal{A}} \\
&= \mathcal{I}(a_i) = \mathcal{I}(t)
\end{aligned}$$

□

Claim 2.2.12. For any term t with free variables $x_1, \ldots, x_r$ (of sort *Address*), for all $\alpha_1, \ldots, \alpha_r \in A$, for any assignment Δ, let $\Delta' = \Delta[\alpha_1/x_1, \ldots, \alpha_r/x_r]$, and therefore $\mathcal{I}_{\Delta'}(t) = \mathcal{I}'_{\Delta'}(\tau_P(t))$.

Proof. Identical to the proof of Claim 2.2.11. □

Claim 2.2.13. Let ξ be a sub-formula of φ, therefore:

1. If ξ is a closed formula then $\mathcal{A} \vDash \xi \iff \mathcal{A}' \vDash \tau_P(\xi)$
2. If ξ is a formula with free variables $x_1, \ldots, x_r$ then for any $\alpha_1, \ldots, \alpha_r \in A$, $\mathcal{A} \vDash \xi[\alpha_1/x_1, \ldots, \alpha_r/x_r] \iff \mathcal{A}' \vDash \tau_P(\xi)[\alpha_1/x_1, \ldots, \alpha_r/x_r]$

Since φ is a closed formula, and since $\mathcal{A} \vDash \varphi$, it holds that $\mathcal{A}' \vDash \tau_P(\varphi)$ and therefore $\mathcal{A}'$ is a distinct model for $\tau_P(\varphi)$.

Proof of Claim 2.2.13. Let us consider the following cases:

Case 1.1: $\xi = t_1 \approx t_2$ without free variables

Follows from Claim 2.2.11.

Case 1.2: $\xi = t_1 \approx t_2$ with free variables $x_1, \ldots, x_r$

Follows from Claim 2.2.12.

Case 1.3: $\xi = \neg\zeta$ without free variables

ζ is also a closed sub-formula of φ and from the induction hypothesis:

$$\begin{aligned}\mathcal{A} \vDash \xi &\iff \mathcal{A} \nvDash \zeta \\ &\iff \mathcal{A}' \nvDash \tau_P(\zeta) \\ &\iff \mathcal{A}' \vDash \tau_P(\xi)\end{aligned}$$

Case 1.4: $\xi = \neg\zeta$ with free variables $x_1, \ldots, x_r$

ζ is also a sub-formula of φ with free variables $x_1, \ldots, x_r$ and from the induction hypothesis, for any $\alpha_1, \ldots, \alpha_r \in A$:

$$\begin{aligned}\mathcal{A} \vDash &\xi[\alpha_1/x_1, \ldots, \alpha_r/x_r] \\ &\iff \mathcal{A} \nvDash \zeta[\alpha_1/x_1, \ldots, \alpha_r/x_r] \\ &\iff \mathcal{A}' \nvDash \tau_P(\zeta)[\alpha_1/x_1, \ldots, \alpha_r/x_r] \\ &\iff \mathcal{A}' \vDash \tau_P(\xi)[\alpha_1/x_1, \ldots, \alpha_r/x_r]\end{aligned}$$

Case 1.5: $\xi = \zeta_1 \vee \zeta_2$ without free variables

ζ_1, ζ_2 are also closed sub-formulas of φ, and from the induction hypothesis:

$$\begin{aligned}\mathcal{A} \vDash \xi &\iff \mathcal{A} \vDash \zeta_1 \text{ or } \mathcal{A} \vDash \zeta_2 \\ &\iff \mathcal{A}' \vDash \tau_P(\zeta_1) \text{ or } \mathcal{A}' \vDash \tau_P(\zeta_2) \\ &\iff \mathcal{A}' \vDash \tau_P(\xi)\end{aligned}$$

Case 1.6: $\xi = \zeta_1 \vee \zeta_2$ with free variables $x_1, \ldots, x_r$

ζ_1, ζ_2 are also sub-formulas of φ with (at most) free variables $x_1, \ldots, x_r$, and from the induction hypothesis, for any $\alpha_1, \ldots, \alpha_r$:

$$\begin{aligned}\mathcal{A} \vDash &\xi[\alpha_1/x_1, \ldots, \alpha_r/x_r] \\ &\iff \mathcal{A} \vDash \zeta_1[\alpha_1/x_1, \ldots, \alpha_r/x_r] \\ &\quad \text{or } \mathcal{A} \vDash \zeta_2[\alpha_1/x_1, \ldots, \alpha_r/x_r] \\ &\iff \mathcal{A}' \vDash \tau_P(\zeta_1)[\alpha_1/x_1, \ldots, \alpha_r/x_r] \\ &\quad \text{or } \mathcal{A}' \vDash \tau_P(\zeta_2)[\alpha_1/x_1, \ldots, \alpha_r/x_r] \\ &\iff \mathcal{A}' \vDash \tau_P(\xi)[\alpha_1/x_1, \ldots, \alpha_r/x_r]\end{aligned}$$

Case 1.7: $\xi = \forall x.\zeta$ without free variables

ζ is a sub-formula of φ with (at most) one free variable x. From the induction hypothesis:

$$\begin{aligned}\mathcal{A} \vDash \xi &\iff \text{For any } \alpha \in A. \mathcal{A} \vDash \zeta[\alpha/x] \\ &\iff \text{For any } \alpha \in A. \mathcal{A}' \vDash \tau_P(\zeta)[\alpha/x] \\ &\iff \mathcal{A} \vDash \tau_P(\xi)\end{aligned}$$

Case 1.8: $\xi = \forall x.\zeta$ with free variables $x_1, \ldots, x_r$

ζ is a sub-formula of φ with (at most) $m+1$ free variables $x, x_1, \ldots, x_r$. From the induction hypothesis, for any $\alpha_1, \ldots, \alpha_r \in A$:

$$\begin{aligned}
&\mathcal{A} \vDash \xi[\alpha_1/x_1, \ldots, \alpha_r/x_r] \\
&\quad \iff \text{For any } \alpha \in A. \mathcal{A} \vDash \zeta[\alpha/x, \alpha_1/x_1, \ldots, \alpha_r/x_r] \\
&\quad \iff \text{For any } \alpha \in A. \mathcal{A}' \vDash \tau_P(\zeta)[\alpha/x, \alpha_1/x_1, \ldots, \alpha_r/x_r] \\
&\quad \iff \mathcal{A}' \vDash \tau_P(\xi)[\alpha_1/x_1, \ldots, \alpha_r/x_r]
\end{aligned}$$

□

Part 2: If there exists some partition P such that $\tau_P(\varphi)$ has a *distinct* model, then φ has a model ($\Leftarrow$)

Let $\mathcal{A}'$ be some model for $\tau_P(\varphi)$ and we construct a model

$$\mathcal{A} = \left(A, a_1^{\mathcal{A}}, \ldots, a_l^{\mathcal{A}}, b_m^{\mathcal{A}}, \ldots, b_m^{\mathcal{A}}, c_n^{\mathcal{A}}, \ldots, c_n^{\mathcal{A}}, s_m^{\mathcal{A}}, \ldots, s_m^{\mathcal{A}}\right)$$

for φ, where A, $b_m^{\mathcal{A}}, \ldots, b_m^{\mathcal{A}}$, $c_n^{\mathcal{A}}, \ldots, c_n^{\mathcal{A}}$, and $s_m^{\mathcal{A}}, \ldots, s_m^{\mathcal{A}}$ are taken from $\mathcal{A}'$.

For every $i \in [1, l]$, $a_i^{\mathcal{A}}$ is defined as: $a_i^{\mathcal{A}} = a'_{P(a_i)}$.

Observation 2.2.14. $\mathcal{A}$ is a Sum structure, and holds the sum property.

Claim 2.2.15. For any closed term t, $\mathcal{I}(t) = \mathcal{I}'(\tau_P(t))$

Proof. Similarly to Claim 2.2.11, we only need to consider $t = a_i$, and we get:

$$\begin{aligned}
\mathcal{I}(t) &= \mathcal{I}(a_i) \\
&= a_i^{\mathcal{A}} \\
&= a'_{P(a_i)} \\
&= \mathcal{I}'(a_{P(a_i)}) = \mathcal{I}'(\tau_P(t))
\end{aligned}$$

□

Claim 2.2.16. For any term t with free variables $x_1, \ldots, x_r$, for any assignment Δ, and for any $\alpha_1, \ldots, \alpha_r \in A$, we define $\Delta' = \Delta[\alpha_1/x_1, \ldots, \alpha_r/x_r]$, and $\mathcal{I}_{\Delta'}(t[\alpha_1/x_1, \ldots, \alpha_r/x_r]) = \mathcal{I}'_{\Delta'}(\tau_P(t)[\alpha_1/x_1, \ldots, \alpha_r/x_r])$.

Proof. Identical to the proof of Claim 2.2.15. □

Claim 2.2.17. Let ξ be a sub-formula of φ, therefore:

1. If ξ is a closed formula, then $\mathcal{A}' \vDash \tau_P(\xi) \iff \mathcal{A} \vDash \xi$.
2. If ξ is a formula with free variables $x_1, \ldots, x_r$ then for every $\alpha_1, \ldots, \alpha_r \in A$, $\mathcal{A}' \vDash \tau_P(\xi)[\alpha_1/x_1, \ldots, \alpha_r/x_r] \iff \mathcal{A} \vDash \xi[\alpha_1/x_1, \ldots, \alpha_r/x_r]$

Since φ is a closed formula, and since $\mathcal{A} \vDash \tau_P(\varphi)$ we get that $\mathcal{A}' \vDash \varphi$.

Proof of Claim 2.2.17. In the same vain of Claim 2.2.13, this follows from Claims 2.2.15 and 2.2.16. □

Q.E.D. Theorem 2.2.9.

2.2.2 Small Models

Definition 2.2.18 (Small model property)**.** Let FRAG be some fragment of SL over vocabulary $\Sigma = \Sigma^{l,m,n}_{+,<}$. FRAG is said to have *small models* if there exists some computable function $\kappa_\Sigma(\cdot)$, such that for any Sum formula $\varphi \in \text{FRAG}$, φ has a model $\iff$ φ has a *small* model $\mathcal{A} = (\mathcal{D}, \mathcal{I})$ where $|\mathcal{D}(Address)| \leq \kappa_\Sigma(|\varphi|)$.

Moreover, φ has a *distinct* model $\iff$ it has a *distinct*, *small* model (with regards to $\kappa_\Sigma(|\varphi|)$).

$\kappa_\Sigma(\cdot)$ is called the bound function of FRAG, and when the vocabulary is clear from context we simply write $\kappa(\cdot)$.

Theorem 2.2.19. For any l, n, the fragment of Sum formulas over the Sum vocabulary $\Sigma^{l,1,n}_{\not{+},\not{<}} = \left(a_1, \ldots, a_l, b^1, c_1, \ldots, c_n, s_b, 0, 1\right)$ holds the *small model property* with bound function $\kappa(x) = l + x + 1$.

I.e. we have a single uninterpreted function (and its associated sum), no plus and no order relation.

Proof of **Theorem 2.2.19**

Let there be some universal, closed formula φ over $\Sigma = \Sigma^{l,1,n}_{\not{+},\not{<}}$ and let there be some minimal structure $\mathcal{A} \in \text{STRUCT}[\Sigma]$ such that $\mathcal{A} \vDash_{\text{SL}} \varphi$ (i.e. $\mathcal{A}$ is a Sum model for φ).

We denote the (finite) size of $\mathcal{A}$ as $z \triangleq |A|$, and we assume towards contradiction that $z \geq l + |\varphi| + 1$ (as our bound function is $\kappa(x) = l + x + 1$). We construct a smaller model $\mathcal{A}'$ for φ. Thus contradicting the minimality of $\mathcal{A}$, and proving our desired claim.

We write out the given model $\mathcal{A} = \left(A, a_1^{\mathcal{A}}, \ldots, a_l^{\mathcal{A}}, b^{\mathcal{A}}, c_1^{\mathcal{A}}, \ldots, c_n^{\mathcal{A}}, s^{\mathcal{A}}\right)$

We know that $|A| = z > l$, and therefore the set

$$S \triangleq A \setminus \left\{a_1^{\mathcal{A}}, \ldots, a_l^{\mathcal{A}}\right\}$$

is not empty. Let us define

$$\alpha^* \triangleq \arg\min_{\alpha \in S} \left\{b^{\mathcal{A}}(\alpha)\right\}$$

and $b^* \triangleq b^{\mathcal{A}}(\alpha^*)$.

We construct the following *smaller* structure $\mathcal{A}' = \left(A', a_1^{\mathcal{A}'}, \ldots, a_l^{\mathcal{A}'}, b^{\mathcal{A}'}, c_1^{\mathcal{A}'}, \ldots, c_n^{\mathcal{A}'}, s^{\mathcal{A}'}\right)$ where

$$A' \triangleq A \setminus \{\alpha^*\} \tag{2.1}$$

$$a_i^{\mathcal{A}'} \triangleq a_i^{\mathcal{A}} \tag{2.2}$$

$$b^{\mathcal{A}'} \triangleq b^{\mathcal{A}} \text{ projected on } A' \tag{2.3}$$

$$s^{\mathcal{A}'} \triangleq s^{\mathcal{A}} - b^* \tag{2.4}$$

and we postpone defining $c_k^{\mathcal{A}'}$ for now. We observe that:

Observation 2.2.20. If $\mathcal{A}$ is a *distinct* model, then so is $\mathcal{A}'$.

Firstly, we prove the following claim:

Claim 2.2.21. $\mathcal{A}'$ holds the Sum property.

Proof. Since $\mathcal{A}$ holds the Sum property for $s^{\mathcal{A}}$:

$$\begin{aligned} s^{\mathcal{A}'} &= s^{\mathcal{A}} - b^* \\ &= \sum_{\alpha \in A} b^{\mathcal{A}}(\alpha) - b^* \\ &= \left(\sum_{\alpha \in A \setminus \{\alpha^*\}} b^{\mathcal{A}}(\alpha) \right) + \underbrace{b^{\mathcal{A}}(\alpha^*)}_{=b^*} - b^* \\ &= \sum_{\alpha \in A'} b^{\mathcal{A}'}(\alpha) \end{aligned}$$

□

The definition for $c_k^{\mathcal{A}'}$ depends on b^*. If $b^* = 0$ then simply $c_k^{\mathcal{A}'} = c_k^{\mathcal{A}}$. In this case, we prove the following:

Lemma 2.2.22. For any term t, assignment Δ,

$$\mathcal{I}_\Delta(t) = \mathcal{I}'_\Delta(t)$$

Proof. Since $b^* = 0$, we get that $s^{\mathcal{A}'} = s^{\mathcal{A}}$ and therefore the interpretations of $\mathcal{A}$ and $\mathcal{A}'$ are identical — $\mathcal{I} = \mathcal{I}'$. □

Corollary 2.2.22.1. Since the domain of $\mathcal{A}'$ is a strict subset of the domain of $\mathcal{A}$, for any formula ξ, $\mathcal{A} \vDash \xi \Rightarrow \mathcal{A}' \vDash \xi$, and in particular $\mathcal{A}'$ is also a model for φ.

In the case that $b^* > 0$, we define

$$c_k^{\mathcal{A}'} \triangleq \begin{cases} s^{\mathcal{A}} - b^* & \text{if } c_k^{\mathcal{A}} = s^{\mathcal{A}} \\ c_k^{\mathcal{A}} + 1 & \text{if } c_k^{\mathcal{A}} \geq s^{\mathcal{A}} - b^* \text{ and } c_k^{\mathcal{A}} \neq s^{\mathcal{A}} \\ c_k^{\mathcal{A}} & \text{otherwise} \end{cases}$$

and the proof is more involved. We firstly make the following observations:

Observation 2.2.23. For any $k \in [1, n]$,

$$c_k^{\mathcal{A}} = s^{\mathcal{A}} \iff c_k^{\mathcal{A}'} = s^{\mathcal{A}'}$$

Observation 2.2.24. For any $k_1, k_2 \in [1, n]$,

$$c_{k_1}^{\mathcal{A}} = c_{k_2}^{\mathcal{A}} \iff c_{k_1}^{\mathcal{A}'} = c_{k_2}^{\mathcal{A}'}$$

The central claim we need to prove is:

Claim 2.2.25. Let ξ be a sub-formula of φ,

1. If ξ is a closed, quantifier-free formula then

$$\mathcal{A} \vDash \xi \iff \mathcal{A}' \vDash \xi$$

2. If ξ is a quantifier-free formula with free variables $x_1, \ldots, x_r$, then for every $\alpha_1, \ldots, \alpha_r \in A'$,

$$\mathcal{A} \vDash \xi[\alpha_1/x_1, \ldots, \alpha_r/x_r] \iff \mathcal{A}' \vDash \xi[\alpha_1/x_1, \ldots, \alpha_r/x_r]$$

3. If ξ is a closed, universally quantified formula then

$$A \vDash \xi \Rightarrow \mathcal{A}' \vDash \xi$$

4. If ξ is a universally quantified formula with free variables $x_1, \ldots, x_r$, then for every $\alpha_1, \ldots, \alpha_r \in A'$,

$$\mathcal{A} \vDash \xi[\alpha_1/x_1, \ldots, \alpha_r/x_r] \Rightarrow \mathcal{A}' \vDash \xi[\alpha_1/x_1, \ldots, \alpha_r/x_r]$$

Corollary 2.2.25.1. $\mathcal{A}' \vDash \varphi$.

Proof. Since φ is a closed, universally quantified sub-formula of itself, and since it is given that $\mathcal{A} \vDash \varphi$, we get from Claim 2.2.25 that $\mathcal{A}' \vDash \varphi$. □

In order to prove Claim 2.2.25 we firstly need to prove the following two lemmas:

Lemma 2.2.26. For any $\alpha \in A'$,

$$b^{\mathcal{A}}(\alpha) = b^{\mathcal{A}'}(\alpha) < s^{\mathcal{A}'} < s^{\mathcal{A}}$$

Proof. First, since $b^{\mathcal{A}'}$ is defined to be a projection of $b^{\mathcal{A}}$ on a subset of its domain $A' \subseteq A$ it is obvious that $b^{\mathcal{A}}(\alpha) = b^{\mathcal{A}'}(\alpha)$ for any $\alpha \in A'$.

Also, since $s^{\mathcal{A}'} = s^{\mathcal{A}} - b^*$ and we know that $b^* > 0$, it is clear that $s^{\mathcal{A}'} < s^{\mathcal{A}}$.

What remains to prove is that for any $\alpha \in A'$, $b^{\mathcal{A}'}(\alpha) < s^{\mathcal{A}'}$. A' has at least $l + |\varphi| + 1$ elements, and therefore $A' \setminus \left\{a_1^{\mathcal{A}'}, \ldots, a_l^{\mathcal{A}'}\right\}$ has at least 2 elements. Let us denote them: α_1, α_2.

For both of these elements,

$$b^{\mathcal{A}'}(\alpha_1), b^{\mathcal{A}'}(\alpha_2) > 0$$

since otherwise they would have been chosen as α^* — contradicting b^*'s minimality. For any element α, since $\mathcal{A}'$ holds the Sum property,

$$\begin{aligned} s^{\mathcal{A}'} &= \sum_{\alpha' \in A'} b^{\mathcal{A}'}(\alpha') \\ &= b^{\mathcal{A}'}(\alpha) + \sum_{\alpha' \in A' \setminus \{\alpha\}} b^{\mathcal{A}'}(\alpha') \end{aligned}$$

We can re-arrange and get that

$$b^{\mathcal{A}'}(\alpha) = s^{\mathcal{A}'} - \sum_{\alpha' \in A' \setminus \{\alpha\}} b^{\mathcal{A}'}(\alpha')$$

and since $A' \setminus \{\alpha\}$ contains either α_1 or α_2, it must be that

$$\sum_{\alpha' \in A' \setminus \{\alpha\}} b^{\mathcal{A}'}(\alpha') > 0$$

and therefore $b^{\mathcal{A}'}(\alpha) < s^{\mathcal{A}'}$. □

Lemma 2.2.27.

$$|\varphi| < s^{\mathcal{A}'} < s^{\mathcal{A}}$$

Proof. Let us examine the set $S \triangleq A' \setminus \left\{a_1^{\mathcal{A}'}, \ldots, a_l^{\mathcal{A}'}\right\}$. It has at least $|\varphi|+1$ elements.

For any $\alpha \in S$, $b^{\mathcal{A}'}(\alpha) > 0$, otherwise it would have been chosen as α^* and we'd have $b^* = 0$ — which contradicts b^*'s minimality.

Therefore, on the one hand,

$$\sum_{\alpha \in S} b^{\mathcal{A}'}(\alpha) \geq |S| \geq |\varphi| + 1 > |\varphi|$$

And, on the other hand, since $S \subseteq A'$, we know that

$$\sum_{\alpha \in S} b^{\mathcal{A}'}(\alpha) \leq \sum_{\alpha \in A'} b^{\mathcal{A}'}(\alpha) = s^{\mathcal{A}'}$$

And combining the two results we get that $|\varphi| < s^{\mathcal{A}'}$, and since $b^* > 0$, $s^{\mathcal{A}'} < s^{\mathcal{A}}$. □

Proof of Claim 2.2.25. We prove the claim using structural induction.

Step 1: $\xi = t_1 \approx t_2$ without free variables

ξ is a closed, quantifier-free formula, so we prove that $\mathcal{A} \vDash \xi \iff \mathcal{A}' \vDash \xi$. We consider the following cases:

Case 1.1: $t_1 = t_2$

Tautology.

Case 1.2: $t_1 = s, t_2 = \text{numeral}$

Since ξ is a sub-formula of φ, $|\xi| \leq |\varphi|$, and therefore the numeral is less than $|\varphi|$.

However, $s^{\mathcal{A}}, s^{\mathcal{A}'} > |\varphi|$ from Lemma 2.2.27 and therefore $\mathcal{A}, \mathcal{A}' \nvDash \xi$.

Case 1.3: $t_1 = s, t_2 = c_k$

From Observation 2.2.23 we know that $s^{\mathcal{A}} = c_k^{\mathcal{A}} \iff s^{\mathcal{A}'} = c_k^{\mathcal{A}'}$ and therefore $\mathcal{A} \vDash \xi \iff \mathcal{A}' \vDash \xi$.

Case 1.4: $t_1 = s, t_2 = b(a_i)$

From Lemma 2.2.26 we know that for any $\alpha \in A'$, $b^{\mathcal{A}'}(\alpha) = b^{\mathcal{A}}(\alpha) < s^{\mathcal{A}'} < s^{\mathcal{A}}$ and in particular for $\alpha = a_i^{\mathcal{A}} = a_i^{\mathcal{A}'}$, $\mathcal{A}, \mathcal{A}' \nvDash \xi$.

Case 1.5: $t_1 = c_k, t_2 = \text{numeral}$

If $c_k^{\mathcal{A}} = c_k^{\mathcal{A}'}$ then trivially $\mathcal{A} \vDash \xi \iff \mathcal{A}' \vDash \xi$.

Otherwise, $c_k^{\mathcal{A}}, c_k^{\mathcal{A}'} \geq s^{\mathcal{A}'}$. However, since ξ is a sub-formula of φ, the numeral is less than $|\varphi|$, and $s^{\mathcal{A}'} > |\varphi|$, from Lemma 2.2.27. Therefore, $\mathcal{A}, \mathcal{A}' \nvDash \xi$.

Case 1.6: $t_1 = c_{k_1}, t_2 = c_{k_2}$

Trivial, from Observation 2.2.24.

Case 1.7: $t_1 = c_k, t_2 = b(a_i)$

If $c_k^{\mathcal{A}} = c_k^{\mathcal{A}'}$ then from Lemma 2.2.26, $\mathcal{A} \vDash \xi \iff \mathcal{A}' \vDash \xi$.

Otherwise, $c_k^{\mathcal{A}}, c_k^{\mathcal{A}'} \geq s^{\mathcal{A}'}$. However, from Lemma 2.2.26 we know that for any $a \in A'$ (and in particular for $a_j^{\mathcal{A}}$), $b^{\mathcal{A}'}(a) = b^{\mathcal{A}}(a) < s^{\mathcal{A}'} \leq c_k^{\mathcal{A}}, c_k^{\mathcal{A}'}$. Therefore, $\mathcal{A}, \mathcal{A}' \nvDash \xi$.

Case 1.8: $t_1 = a_{i_1}, t_2 = a_{i_2}$

Trivial, since the interpretation of the *Address* constants is identical in $\mathcal{A}$, $\mathcal{A}'$.

Any other case is symmetrical to one of the cases above.

Step 2: $\xi = t_1 \approx t_2$ with free variables $x_1, \ldots, x_r$

ξ is a quantifier-free formula, so we prove that for any $\alpha_1, \ldots, \alpha_r \in A'$

$$\mathcal{A} \vDash \xi[\alpha_1/x_1, \ldots, \alpha_r/x_r] \iff \mathcal{A}' \vDash \xi[\alpha_1/x_1, \ldots, \alpha_r/x_r]$$

We consider the following cases:

Case 2.1: $t_1 = t_2$

Tautology.

Case 2.2: $t_1 = s, t_2 = b(x)$

From Lemma 2.2.26 we know that for any $\alpha \in A'$, $b^{\mathcal{A}'}(\alpha) = b^{\mathcal{A}}(\alpha) < s^{\mathcal{A}'} < s^{\mathcal{A}}$ and in particular $\mathcal{A}, \mathcal{A}' \nvDash \xi[\alpha/x]$.

Case 2.3: $t_1 = b(x), t_2 = \text{numeral}$

Trivial, since from Lemma 2.2.26, for every $\alpha \in A'$, $b^{\mathcal{A}}(\alpha) = b^{\mathcal{A}'}(\alpha)$.

Case 2.4: $t_1 = b(x), t_2 = c_k$

Let there be $\alpha \in A'$, we separate into the following cases:

1. If $c_k^{\mathcal{A}} = c_k^{\mathcal{A}'}$:

   From Lemma 2.2.26 we get

$$\begin{aligned} \mathcal{A} \vDash \xi[\alpha/x] &\iff \mathcal{A} \vDash b(\alpha) \approx c_k \\ &\iff b^{\mathcal{A}}(\alpha) = c_k^{\mathcal{A}} \\ &\iff b^{\mathcal{A}'}(\alpha) = c_k^{\mathcal{A}'} \\ &\iff \mathcal{A}' \vDash b(\alpha) \approx c_k \\ &\iff \mathcal{A}' \vDash \xi[\alpha/x] \end{aligned}$$

2. Otherwise, $c_k^{\mathcal{A}} \geq s^{\mathcal{A}'}$ and $c_k^{\mathcal{A}'} \geq s^{\mathcal{A}'}$. From Lemma 2.2.26 we get $b^{\mathcal{A}'}(a) = b^{\mathcal{A}}(a) < s^{\mathcal{A}'} < s^{\mathcal{A}}$ and therefore $\mathcal{A} \nvDash \xi[\alpha/x]$ and $\mathcal{A}' \nvDash \xi[\alpha/x]$.

Case 2.5: $t_1 = b(x), t_2 = b(a_i)$

Trivial from Lemma 2.2.26.

Case 2.6: $t_1 = b(x_1), t_2 = b(x_2)$

Trivial from Lemma 2.2.26.

Case 2.7: $t_1 = a_i, t_2 = x$

Trivial, since the interpretation of the address constants is identical in $\mathcal{A}$ and $\mathcal{A}'$.

Case 2.8: $t_1 = x_1, t_2 = x_2$

Trivially holds for any $a \in A'$.

Any other case is symmetrical to one of the cases above.

Step 3: $\xi = \neg\zeta$ without free variables

Since φ is a universal formula we can assume it is in prenex form, and therefore, ζ is a closed, quantifier-free formula, shorter than ξ and from the induction hypothesis, $\mathcal{A} \vDash \zeta \iff \mathcal{A}' \vDash \zeta$, and therefore

$$\begin{aligned} \mathcal{A} \vDash \xi &\iff \mathcal{A} \nvDash \zeta \\ &\iff \mathcal{A}' \nvDash \zeta \\ &\iff \mathcal{A}' \vDash \xi \end{aligned}$$

Step 4: $\xi = \neg\zeta$ with free variables $x_1, \ldots, x_r$

Similarly to the closed formula case, ζ is a quantifier-free formula with free variables $x_1, \ldots, x_r$ and the claim holds from the induction hypothesis.

Step 5: $\xi = \zeta_1 \vee \zeta_2$ without free variables

Similarly to the negation formula case, ζ_1, ζ_2 are closed, quantifier-free formulas and the claim holds from the induction hypothesis.

Step 6: $\xi = \zeta_1 \vee \zeta_2$ with free variables $x_1, \ldots, x_r$

Similarly to the closed formula case, and the claim holds from the induction hypothesis.

Step 7: $\xi = \forall v.\zeta$ without free variables

Since ξ is a universal formula, we need to show that if $\mathcal{A} \vDash \xi$ then $\mathcal{A}' \vDash \xi$ (but not vice versa).

ζ is a universally quantified formula with (at most) one free variable x. If $\mathcal{A} \vDash \xi$ then for every $\alpha \in A$,

$$\mathcal{A} \vDash \zeta[\alpha/x]$$

and in particular for any $\alpha \in A' \subsetneq A$.

ζ is shorter than ξ and therefore the induction hypothesis holds:

$$\mathcal{A}' \vDash \zeta[\alpha/x]$$

for any $\alpha \in A'$, and therefore $\mathcal{A}' \vDash \xi$.

Step 8: $\xi = \forall v.\zeta$ with free variables $x_1, \ldots, x_r$

Let there be $\alpha_1, \ldots, \alpha_r \in A'$. If

$$\mathcal{A} \vDash \xi[\alpha_1/x_1, \ldots, \alpha_r/x_r]$$

then for every $\alpha \in A$,

$$\mathcal{A} \vDash \zeta[\alpha/x, \alpha_1/x_1, \ldots, \alpha_r/x_r]$$

and in particular for every $\alpha \in A' \subsetneq A$. From the induction hypothesis for ζ we get:

$$\mathcal{A}' \vDash \zeta[\alpha/x, \alpha_1/x_1, \ldots, \alpha_r/x_r]$$

which is true for any $a \in A'$, and therefore,

$$\mathcal{A}' \vDash \xi[\alpha_1/x_1, \ldots, \alpha_r/x_r]$$

□

Q.E.D. Theorem 2.2.19.

Remark. An attempt to extend the proof above for a fragment of SL with two balance functions falls apart in a few places, but most importantly Lemma 2.2.26 does not hold between different balance functions. I.e. if we have two balance functions we cannot be sure that $\forall x.b_1(x) \leq s_2$ (or vice versa).

This lemma in central for constructing a smaller model (as part of the proof by contradiction), and it is unclear how to adapt the proof for this extended fragment of SL.

Moreover, the fact that the proof does not work when there are comparisons of balances and unrelated sums is not accidental. We see in Section 2.3 that these comparisons are essential for proving our undecidability result in SL.

However, we believe that some restricted fragment of SL with dual balances can be decidable — e.g. if we disallow this kind of comparisons — and still be able to express interesting properties and transitions that are beyond the scope of the single function fragment. As we discuss in Section 4.2, the question is not only how to prove decidability for a fragment, but also how to find a fragment that is surprisingly interesting, albeit of restricted expressiveness.

2.2.3 Presburger Reduction

Outline

For showing decidability of a fragment of SL, we describe a Turing reduction to pure Presburger arithmetic. We start by defining a transformation $\tau(\cdot)$ of formulas in SL into formulas in Presburger arithmetic.

In addition, we define an auxiliary formula $\eta(\varphi)$, in order to set up some global constraints on the Presburger model we are looking for. By relying on the properties of *distinctness* and *small models* (as defined in the previous sub-sections) we get the following result:

Theorem 2.2.28 (Presburger reduction)**.** A Sum formula φ has a *distinct*, *small* Sum model $\iff$ $\varphi' \triangleq \tau(\varphi) \wedge \eta(\varphi)$ has a Standard Model of Arithmetic.

Corollary 2.2.28.1. Let FRAG be some fragment of SL that holds the *small model property*, with bound function $\kappa(\cdot)$. Therefore, FRAG is decidable.

Proof of Corollary 2.2.28.1. Let φ be some formula in FRAG. φ has a Sum model $\iff$ for some partition P of $\{a_1, \ldots, a_l\}$, $\tau_P(\varphi)$ has a *distinct* Sum model.

All of the formulas $\tau_P(\varphi)$ are in FRAG, therefore they have a *distinct* Sum model $\iff$ they have a *distinct*, *small* Sum model.

Combining with Theorem 2.2.28, we get that for any P, $\varphi_P = \tau_P(\varphi)$ has a *distinct* Sum model $\iff$ $\tau(\varphi_P) \wedge \eta(\varphi_P)$ has a model (in Presburger arithmetic).

We get the following decision procedure, using the Presburger arithmetic decision procedure as an oracle:

- For each possible partition P of $\{a_1, \ldots, a_l\}$ let $\varphi_P = \tau_P(\varphi)$.
- Check for each P if $\tau(\varphi_P) \wedge \eta(\varphi_P)$ has a model (using any decision procedure for Presburger arithmetic).
- If any of the Presburger queries found a model, then for some partition P, the formula φ_P has a *distinct* Sum model, and therefore φ has Sum model.
- Otherwise, there is no *distinct* Sum model for any partition P, and therefore there is no Sum model for φ.

□

Remark. The decision procedure described requires B_l Presburger queries, where B_l is Bell's number for all possible partitions of a set of size l. This number is huge, even for small values of l, but those queries can be done in parallel.

Defining the Transformations

The transformation of formulas from SL to PA works by explicitly writing out sums as additions and universal quantifiers as conjunctions. Since we're dealing with a fragment of SL that has some bound function $\kappa(\cdot)$, we know that for given formula φ, there is a model with at most $\kappa(|\varphi|)$ elements of *Address* sort.

Moreover, we use $\tilde{\kappa} \triangleq \max\{\kappa(|\varphi|), l\}$ as the upper bound (where l is the amount of *Address* constants). Since we're looking for distinct models, it is obvious that we need at least l distinct elements.

For each balance function b^1_j we have $\tilde{\kappa}$ constants $b_{1,j}, \ldots, b_{\tilde{\kappa},j}$.

In addition we have $\tilde{\kappa}$ indicator constants $a_1, \ldots, a_{\tilde{\kappa}}$, to mark if an *Address* element is "active". An inactive element has all zero balances, and is skipped over in universal quantifiers.

Any *Address* constant a_i or *Address* variable x is handled in two ways, depending on the context they appear in:

- If they are compared, we replace the comparison with $\top$ or $\bot$; we know statically if the comparison holds, since the *Address* constants are distinct and every universal quantifier is written out as a conjunction.
- Otherwise, they must be used in some balance function b^1_j, and then they are substituted with the corresponding $b_{i,j}$ or $b_{x,j}$ (which will be determined once the universal quantifiers are unrolled).

The integral constants $c_1, \ldots, c_n$ are simply copied over.
In summary:

Definition 2.2.29 (Corresponding Presburger vocabulary)**.** Given the Sum vocabulary $\Sigma^{l,m,n}$ and a bound $\tilde{\kappa} \geq l$, we define the *corresponding Presburger vocabulary* as $\Sigma^{l,m,n}_{\mathrm{Pres}(\tilde{\kappa})} = \mathrm{Pres}(\Sigma^{l,m,n}, \tilde{\kappa}) \triangleq \left(a_1, \ldots, a_{\tilde{\kappa}}, b_{1,1}, \ldots, b_{\tilde{\kappa},m}, c_1, \ldots, c_n, 0, 1, +^2, <^2\right)$.

Firstly, we define the simpler auxiliary formula $\eta(\varphi)$ in three parts:

Definition 2.2.30. We require that inactive *Address* elements have zero balances -

$$\eta_1(\varphi) = \bigwedge_{i=1}^{\tilde{\kappa}} \left[(a_i \approx 0) \to \left(\bigwedge_{j=1}^{m} b_{i,j} \approx 0 \right) \right]$$

Definition 2.2.31. And that elements referred by *Address* constants be active -

$$\eta_2(\varphi) = \bigwedge_{i=1}^{l} a_i \not\approx 0$$

Definition 2.2.32. Finally, we require that the active elements are a continuous sequence starting at 1. Or, put differently, once an indicator is zero, all indicators following it are also zero:

$$\eta_3(\varphi) = \bigwedge_{i=1}^{\tilde{\kappa}} \left[a_i \approx 0 \to \left(\bigwedge_{i'=i}^{\tilde{\kappa}} a_{i'} \approx 0 \right) \right]$$

The complete auxiliary formula is then $\eta(\varphi) = \eta_1(\varphi) \wedge \eta_2(\varphi) \wedge \eta_3(\varphi)$.

In order to define $\tau(\varphi)$, we firstly define the transformation for terms, and then build up the complete transformation, using several substitutions:

Definition 2.2.33. We define the terms transformation inductively, and we substitute balances and *Address* terms (constants or variables) with placeholders (marked with *), which are further substituted:

$$\tau_0(t) \triangleq \begin{cases} a^*_i & \text{if } t = a_i \\ x^* & \text{if } t = x \text{ for some free variable} \\ b_{1,j} + \cdots + b_{\tilde{\kappa},j} & \text{if } t = s_j \\ b^*_j(\tau_0(t_1)) & \text{if } t = b_j(t_1) \text{ where } t_1 \in \{a_i, x\} \\ \tau_0(t_1) + \tau_0(t_2) & \text{if } t = t_1 + t_2 \end{cases}$$

Definition 2.2.34. Next we define the transformation for formulas, replacing only variable placeholders:

$$\tau_1(\xi) \triangleq \begin{cases} \tau_0(t_1) \approx \tau_0(t_2) & \text{if } \xi = t_1 \approx t_2 \\ \tau_0(t_1) < \tau_0(t_2) & \text{if } \xi = t_1 < t_2 \\ \neg\tau_1(\zeta) & \text{if } \xi = \neg\zeta \\ \tau_1(\zeta_1) \wedge \tau_1(\zeta_2) & \text{if } \xi = \zeta_1 \wedge \zeta_2 \\ \bigwedge_{i=1}^{\tilde{\kappa}} (a_i \approx 0 \vee \tau_1(\zeta)[a_i^*/x^*]) & \text{if } \xi = \forall x.\zeta \end{cases}$$

We can see that for any formula ξ containing arbitrary terms, $\tau_1(\xi)$ only has a_i^* and b_j^* placeholders (but no x^* ones).

Definition 2.2.35. Now we define a substitution σ_1 that removes *Address* comparisons by evaluating them:

$$\sigma_1 \triangleq [\top/(a_i^* \approx a_i^*)][\bot/(a_i^* \approx a_{i'}^*)]$$

where $i, i' \in [1, \tilde{\kappa}]$.

Note. We first replace comparisons where $a_i^* \approx a_i^*$, which is equivalent to **true** ($\top$). Then any remaining comparison must be where $i \neq i'$, and therefore equivalent to **false** ($\bot$).

Definition 2.2.36. Finally, we're left with placeholders inside balance functions, which we substitute by the corresponding balance constant:

$$\sigma_2 \triangleq \left[b_{i,j}/b_j^*(a_i^*)\right]$$

where $i \in [1, \tilde{\kappa}], j \in [1, m]$.

Definition 2.2.37. The complete transformation is then:

$$\tau(\varphi) \triangleq \tau_1(\varphi)\sigma_1\sigma_2$$

Given the above definition, let us recall Theorem 2.2.28:

Theorem. A Sum formula φ has a *distinct*, *small* Sum model $\iff$ $\varphi' \triangleq \tau(\varphi) \wedge \eta(\varphi)$ has a model.

***Proof of* Theorem 2.2.28**

We first define congruence between structures in Sum Logic and structures in the corresponding Presburger vocabulary, and we prove a general theorem about them. We use that congruence theorem to prove that a formula φ has a *small*, *distinct* Sum model $\iff$ φ' has a Presburger model.

Part 1: Congruence Lemmas

Definition 2.2.38. Given a Sum vocabulary $\Sigma^{l,m,n}$, a bound $\tilde{\kappa} \geq l$ and a formula φ over $\Sigma^{l,m,n}$. Two structures, $\mathcal{A} \in \text{STRUCT}\left[\Sigma^{l,m,n}\right]$ and $\mathcal{A}' \in \text{STRUCT}\left[\text{Pres}(\Sigma^{l,m,n}, \tilde{\kappa})\right]$ are said to be congruent if the following conditions hold:

(1) $\mathcal{A}$ holds the Sum property.

(2) $\mathcal{A}'$ satisfies $\eta(\varphi)$.

(3) $z \triangleq |A| \leq \tilde{\kappa}$, and we write out $A = \{\alpha_1, \ldots, \alpha_z\}$.

(4) For any $i \in [1, l]$, $a_i^{\mathcal{A}} = \alpha_i$.

(5) For any $j \in [1,m]$, for any $i \in [1,z]$, $b_{i,j}^{\mathcal{A}'} = b_j^{\mathcal{A}}(\alpha_i)$, and for any $i > z$, $b_{i,j}^{\mathcal{A}'} = 0$.

(6) For any $i \in [1,z]$, $a_i^{\mathcal{A}'} > 0$ and for any $i > z$, $a_i^{\mathcal{A}'} = 0$.

(7) $\mathcal{A}$ is distinct, and in particular $l \leq z$.

Lemma 2.2.39. Let $\mathcal{A}, \mathcal{A}'$ be two congruent structures for Sum vocabulary Σ, bound $\tilde{\kappa}$ and formula φ. For any ground term t of sort *Nat* over Σ,

$$\mathcal{I}'(\tau_0(t)\sigma_2) = \mathcal{I}(t)$$

Proof. We prove the lemma using structural induction over all possible ground terms:
Step 1.1: $t = s_j$ for any $j \in [1,m]$

From Congruence Condition 2.2.38(1) $\mathcal{A}$ holds the sum property, and therefore:

$$\mathcal{I}(s_j) = s_j^{\mathcal{A}} = \sum_{\alpha \in A} b_j^{\mathcal{A}}(\alpha) = \sum_{i=1}^{z} b_j^{\mathcal{A}}(\alpha_i)$$

From Congruence Condition 2.2.38(5), for any $i \in [1,z]$, $b_{i,j}^{\mathcal{A}'} = b_j^{\mathcal{A}}(\alpha_i)$, and for any $i \in [z+1, \tilde{\kappa}]$, $b_{i,j}^{\mathcal{A}'} = 0$, therefore we can write the sum above as

$$\mathcal{I}(s_j) = \cdots = \sum_{i=1}^{\tilde{\kappa}} b_{i,j}^{\mathcal{A}'}$$

From the definition of τ_0 we get:

$$\mathcal{I}'(\tau_0(s_j)\sigma_2) = \mathcal{I}'([b_{1,j} + \cdots + b_{\tilde{\kappa},j}]\sigma_2) = \sum_{i=1}^{\tilde{\kappa}} b_{i,j}^{\mathcal{A}'}$$

Since we have no placeholders, σ_2 has no effect, and we get the same expression as for $\mathcal{I}(t)$.
Step 1.2: $t = b_j(a_i)$ where $i \in [1,l], j \in [1,m]$

From Congruence Condition 2.2.38(4), $a_i^{\mathcal{A}} = \alpha_i$, and we get:

$$\mathcal{I}(t) = b_j^{\mathcal{A}}(\alpha_i)$$

From the definition of τ_0 and σ_2 we get:

$$\tau_0(t)\sigma_2 = \left[b_j^*(a_i^*)\right]\sigma_2 = b_{i,j}$$

And therefore, since $\mathcal{A}$ is distinct, $i \leq l \leq z$, and from Congruence Condition 2.2.38(5),

$$\mathcal{I}'(\tau_0(t)\sigma_2) = \mathcal{I}'(b_{i,j}) = b_{i,j}^{\mathcal{A}'} = b_j^{\mathcal{A}}(\alpha_i)$$

Step 1.3: $t = t_1 + t_2$

Follows from the induction hypothesis for t_1 and t_2 (since $+$ is interpreted in the same way in $\mathcal{A}$ and $\mathcal{A}'$). □

Lemma 2.2.40. Let $\mathcal{A}, \mathcal{A}'$ be two congruent structures for Sum vocabulary Σ, bound $\tilde{\kappa}$ and formula φ. For any term t with at most r free variables $x_1, \ldots, x_r$, for any indices $i_1, \ldots, i_r \in [1,z]$, for any assignment Δ we define

$$\Delta' = \Delta[\alpha_{i_1}/x_1, \ldots, \alpha_{i_r}/x_r]$$

and the following holds:

$$\mathcal{I}_{\Delta'}(t) = \mathcal{I}'\big(\tau_0(t)\big[a^*_{i_1}/x^*_1, \ldots, a^*_{i_r}/x^*_r\big]\sigma_2\big)$$

Proof. We prove the lemma using structural induction over all possible terms with free variables:
Step 1.4: $t = b_j(x)$

For any $i \in [1, z]$,

$$\mathcal{I}_{\Delta'}(t) = \mathcal{I}(b_j)\big(\Delta'(x)\big) = b_j^{\mathcal{A}}(\alpha_i)$$

By definition, $\tau_0(t) = b^*_j(x^*)$, and therefore

$$\begin{aligned}\mathcal{I}'(\tau_0(t)[a^*_i/x^*]\sigma_2) &= \mathcal{I}'\big(b^*_j(a^*_i)\sigma_2\big)\\ &= \mathcal{I}'(b_{i,j})\\ &= b_{i,j}^{\mathcal{A}'}\\ &= b_j^{\mathcal{A}}(\alpha_i)\end{aligned}$$

since $i \in [1, z]$.
Step 1.5: $t = t_1 + t_2$

Either t_1 or t_2 has free variables, and let us assume w.l.o.g. that t_1 does. Therefore, t_2 is either a ground term, or also has free variables. If t_2 has no free variables, the substitution of free variables wouldn't affect it.

In both cases, from the induction hypothesis and from Lemma 2.2.39, for any $i_1, \ldots, i_r \in [1, z]$,

$$\mathcal{I}_{\Delta'}(t_v) = \mathcal{I}'(\tau_0(t_v)\big[a^*_{i_1}/x^*_1, \ldots, a^*_{i_r}/x^*_r\big])$$

where $v \in \{1, 2\}$, and we get desired equality for t as well. □

Lemma 2.2.41. Let $\mathcal{A}, \mathcal{A}'$ be two congruent structures for Sum vocabulary Σ, bound $\tilde{\kappa}$ and formula φ. Let ξ be a sub-formula of φ, therefore:

1. If ξ is a closed formula, then $\mathcal{A}' \vDash \tau(\xi) \iff \mathcal{A} \vDash \xi$.
2. If ξ is a formula with free variables $x_1, \ldots, x_r$ then for any $i_1, \ldots, i_r \in [1, z]$:

$$\begin{aligned}&\mathcal{A}' \vDash \tau_1(\xi)\big[a^*_{i_1}/x^*_1, \ldots, a^*_{i_r}/x^*_r\big]\sigma_1\sigma_2\\ &\quad\iff \mathcal{A} \vDash \xi[\alpha_{i_1}/x_1, \ldots, \alpha_{i_r}/x_r]\end{aligned}$$

Proof. Let us separate into the following steps:
Step 1.6: $\xi = t_1 \approx t_2$ without free variables, where t_1, t_2 are of Address sort

Since t_1, t_2 are Addresses, there are two indices $i_1, i_2 \in [1, l]$ such that $t_1 = a_{i_1}, t_2 = a_{i_2}$. From Congruence Condition 2.2.38(7), $\mathcal{A}$ is distinct and therefore

$$\mathcal{A} \vDash \xi \iff i_1 = i_2.$$

As for $\tau(\xi)$,

$$\begin{aligned}\tau(\xi) &= (\tau_0(t_1) \approx \tau_0(t_2))\sigma_1\sigma_2\\ &= \big(a^*_{i_1} \approx a^*_{i_2}\big)\sigma_1\\ &= \begin{cases}\top & \text{if } i_1 = i_2\\ \bot & \text{otherwise}\end{cases}\end{aligned}$$

Which means that $\mathcal{A}' \vDash \tau(\xi) \iff i_1 = i_2 \iff \mathcal{A} \vDash \xi$.

Step 1.7: $\xi = t_1 \approx t_2$ without free variables, where t_1, t_2 are of sort *Nat*

In this case, σ_1 would not change the formula and we can apply σ_2 to each term:

$$\tau(\xi) = \tau_0(t_1)\sigma_2 \approx \tau_0(t_2)\sigma_2$$

Since t_1, t_2 are of sort *Nat*, from Lemma 2.2.39 we get that

$$\begin{aligned} \mathcal{A}' \vDash \tau(\xi) &\iff \mathcal{I}'(\tau_0(t_1)\sigma_2) = \mathcal{I}'(\tau_0(t_2)\sigma_2) \\ &\iff \mathcal{I}(t_1) = \mathcal{I}(t_2) && \text{(Lemma 2.2.39)} \\ &\iff \mathcal{A} \vDash t_1 \approx t_2 = \xi \end{aligned}$$

Step 1.8: $\xi = t_1 \approx t_2$ with free variables $x_1, \ldots, x_r$, where t_1, t_2 are of Address sort

Let us first define $\sigma = [\alpha_{i_1}/x_1, \ldots, \alpha_{i_r}/x_r]$, $\sigma' = \left[a^*_{i_1}/x^*_1, \ldots, a^*_{i_r}/x^*_r\right]$.

If t_1, t_2 both have free variables then we can write them as $t_1 = x_1, t_2 = x_2$ and after substituting σ we get that

$$\xi\sigma = \alpha_{i_1} \approx \alpha_{i_2}.$$

Therefore, $\mathcal{A} \vDash \xi\sigma \iff i_1 = i_2$.

As for $\mathcal{A}'$, we get

$$\begin{aligned} \tau_1(\xi)\sigma\sigma_1\sigma_2 &= (x^*_1 \approx x^*_2)\sigma'\sigma_1\sigma_2 \\ &= \left(a^*_{i_1} \approx a^*_{i_2}\right)\sigma_1\sigma_2 \\ &= \left(a^*_{i_1} \approx a^*_{i_2}\right)\sigma_1 \\ &= \begin{cases} \top & \text{if } i_1 = i_2 \\ \bot & \text{otherwise} \end{cases} \end{aligned}$$

And we get that $\mathcal{A}' \vDash \tau_1(\xi)\sigma'\sigma_1\sigma_2 \iff i_1 = i_2 \iff \mathcal{A} \vDash \xi\sigma$.

Step 1.9: $\xi = t_1 \approx t_2$ with free variables $x_1, \ldots, x_r$, where t_1, t_2 are of sort *Nat*

Similarly to the case above, we define $\sigma = [\alpha_{i_1}/x_1, \ldots, \alpha_{i_r}/x_r]$, $\sigma' = \left[a^*_{i_1}/x^*_1, \ldots, a^*_{i_r}/x^*_r\right]$. For any assignment Δ, we define

$$\Delta' = \Delta\sigma$$

Since t_1, t_2 are of sort *Nat*, σ_1 has no effect, and from Lemma 2.2.40,

$$\begin{aligned} &\mathcal{A}' \vDash \tau_1(\xi)\sigma'\sigma_1\sigma_2 \\ &\quad\iff \mathcal{A}' \vDash [\tau_0(t_1) \approx \tau_0(t_2)]\sigma'\sigma_2 \\ &\quad\iff \mathcal{I}'(\tau_0(t_1)\sigma'\sigma_2) = \mathcal{I}'(\tau_0(t_2)\sigma'\sigma_2) \\ &\quad\iff \mathcal{I}_{\Delta'}(t_1\sigma) = \mathcal{I}_{\Delta'}(t_2\sigma) && \text{(Lemma 2.2.40)} \\ &\quad\iff \mathcal{A} \vDash \xi\sigma \end{aligned}$$

Step 1.10: $\xi = \neg\zeta$ without free variables

Follows from the induction hypothesis for ζ:

$$\begin{aligned}\mathcal{A}' \vDash \tau(\xi) &\iff \mathcal{A}' \vDash \neg\tau(\zeta)\\ &\iff \mathcal{A}' \nvDash \tau(\zeta)\\ &\iff \mathcal{A} \nvDash \zeta\\ &\iff \mathcal{A} \vDash \neg\zeta\\ &\iff \mathcal{A} \vDash \xi\end{aligned}$$

Step 1.11: $\xi = \neg\zeta$ with free variables $x_1, \ldots, x_r$

Follows from the induction hypothesis for ζ, for any $i_1, \ldots, i_r \in [1, z]$:

$$\begin{aligned}&\mathcal{A}' \vDash \tau_1(\neg\zeta)\big[a^*_{i_1}/x^*_1, \ldots, a^*_{i_r}/x^*_r\big]\sigma_1\sigma_2\\ &\quad\iff \mathcal{A}' \vDash \neg\tau_1(\zeta)\big[a^*_{i_1}/x^*_1, \ldots, a^*_{i_r}/x^*_r\big]\sigma_1\sigma_2\\ &\quad\iff \mathcal{A}' \nvDash \tau_1(\zeta)\big[a^*_{i_1}/x^*_1, \ldots, a^*_{i_r}/x^*_r\big]\sigma_1\sigma_2\\ &\quad\iff \mathcal{A} \nvDash \zeta[\alpha_{i_1}/x_1, \ldots, \alpha_{i_r}/x_r] \qquad \text{(Induction hypothesis)}\\ &\quad\iff \mathcal{A} \vDash \neg\zeta[\alpha_{i_1}/x_1, \ldots, \alpha_{i_r}/x_r]\\ &\quad\iff \mathcal{A} \vDash \xi[\alpha_{i_1}/x_1, \ldots, \alpha_{i_r}/x_r]\end{aligned}$$

Step 1.12: $\xi = \zeta_1 \vee \zeta_2$ without free variables

Follows from the induction hypothesis for ζ_1 and ζ_2:

$$\begin{aligned}\mathcal{A}' \vDash \tau(\xi) &\iff \mathcal{A}' \vDash \tau(\zeta_1) \vee \tau(\zeta_2)\\ &\iff \mathcal{A}' \vDash \tau(\zeta_1) \text{ or } \mathcal{A}' \vDash \tau(\zeta_2)\\ &\iff \mathcal{A} \vDash \zeta_1 \text{ or } \mathcal{A} \vDash \zeta_2\\ &\iff \mathcal{A} \vDash \zeta_1 \vee \zeta_2\\ &\iff \mathcal{A} \vDash \xi\end{aligned}$$

Step 1.13: $\xi = \zeta_1 \vee \zeta_2$ with free variables $x_1, \ldots, x_r$

Follows from the induction hypothesis for ζ_1 and ζ_2, similar to the no free variables case above, since $\tau_1(\zeta_1 \vee \zeta_2) = \tau_1(\zeta_1) \vee \tau_1(\zeta_2)$.

Step 1.14: $\xi = \forall x.\zeta$ without free variables

Since $a_i^{\mathcal{A}'} = 0 \iff i > z$ we get the following:

$$\begin{aligned}\mathcal{A}' \vDash \tau(\xi) &\iff \mathcal{A}' \vDash \bigwedge_{i=1}^{\tilde{\kappa}} (a_i \approx 0 \vee \tau_1(\zeta)[a^*_i/x^*])\sigma_1\sigma_2\\ &\iff \mathcal{A}' \vDash \bigwedge_{i=1}^{z} \tau_1(\zeta)[a^*_i/x^*]\sigma_1\sigma_2\\ &\iff \mathcal{A}' \vDash \tau_1(\zeta)[a^*_i/x^*]\sigma_1\sigma_2 \text{ for all } i \in [1, z]\\ &\iff \mathcal{A} \vDash \zeta[\alpha_i/x] \text{ for all } i \in [1, z] \qquad \text{(Induction hypothesis for } \zeta)\\ &\iff \mathcal{A} \vDash \forall x.\zeta = \xi \qquad \text{(The set } A \text{ is covered by } \alpha_1, \ldots, \alpha_z)\end{aligned}$$

Step 1.15: $\xi = \forall x.\zeta$ with free variables $x_1, \ldots, x_r$

Similar to the case above, using the induction hypothesis for ζ as a formula with free variables $x, x_1, \ldots, x_r$.

□

Part 2: Proof of Theorem 2.2.28 ($\Rightarrow$): If φ has a *distinct*, *small* Sum model, then φ' has a model

Let there be a *distinct*, *small* Sum model for φ:

$$\mathcal{A} = \left(A, a_1^{\mathcal{A}}, \ldots, a_l^{\mathcal{A}}, b_1^{\mathcal{A}}, \ldots, b_m^{\mathcal{A}}, c_1^{\mathcal{A}}, \ldots, c_n^{\mathcal{A}}, s_1^{\mathcal{A}}, \ldots, s_m^{\mathcal{A}}\right).$$

We can represent its Addresses set as $A = \{\alpha_1, \ldots, \alpha_z\}$ where $z = |A|$, and for every $i \in [1, l]$, $a_i^{\mathcal{A}} = \alpha_i$ since $\mathcal{A}$ is distinct. Combined with the fact that $\mathcal{A}$ is small we know that $z \leq \tilde{\kappa}$.

We define $\mathcal{A}'$ the model for φ' as follows:

$$\mathcal{A}' = \left(a_1^{\mathcal{A}'}, \ldots, a_{\tilde{\kappa}}^{\mathcal{A}'}, b_{1,1}^{\mathcal{A}'}, \ldots, b_{\tilde{\kappa},m}^{\mathcal{A}'}, c_1^{\mathcal{A}'}, \ldots, c_n^{\mathcal{A}'}\right)$$

where the indicators are

$$a_i^{\mathcal{A}'} = \begin{cases} 1 & \text{if } i \leq z \\ 0 & \text{otherwise} \end{cases};$$

the balances are

$$b_{i,j}^{\mathcal{A}'} = \begin{cases} b_j^{\mathcal{A}}(\alpha_i) & \text{if } i \leq z \\ 0 & \text{otherwise} \end{cases};$$

and the natural constants are $c_k^{\mathcal{A}'} = c_k^{\mathcal{A}}$.

Claim 2.2.42. The structure $\mathcal{A}'$ satisfies $\eta(\varphi)$.

Proof. We show that $\mathcal{A}'$ satisfies $\eta_1(\varphi)$, $\eta_2(\varphi)$ and $\eta_3(\varphi)$:
Step 2.1: $\mathcal{A}'$ satisfies $\eta_1(\varphi)$

We need to show that for each $i \in [1, \tilde{\kappa}]$,

$$\mathcal{A}' \vDash \left[(a_i \approx 0) \to \left(\bigwedge_{j=1}^{n} b_{i,j} \approx 0\right)\right].$$

I.e. for each $i \in [1, \tilde{\kappa}]$ and $j \in [1, m]$, if $a_i^{\mathcal{A}'} = 0$, then $b_{i,j}^{\mathcal{A}'} = 0$.

By definition, $a_i^{\mathcal{A}'} = 0 \iff i > z$, in which case, for any $j \in [1, m]$, $b_{i,j}^{\mathcal{A}'} = 0$, as required.
Step 2.2: $\mathcal{A}'$ satisfies $\eta_2(\varphi)$

We need to show that for each $i \in [1, l]$,

$$\mathcal{A}' \vDash a_i \not\approx 0,$$

i.e. $a_i^{\mathcal{A}'} \neq 0$.

Since $\mathcal{A}$ is a distinct model, it has at least l addresses: $l \leq z$. By definition, for any $i \in [1, z]$, $a_i^{\mathcal{A}'} = 1 > 0$, in particular for any $i \in [1, l] \subseteq [1, z]$.
Step 2.3: $\mathcal{A}'$ satisfies $\eta_3(\varphi)$

We need to show that for each $i \in [1, \tilde{\kappa}]$, if $a_i^{\mathcal{A}'} = 0$, then for any $i' > i$, $a_{i'}^{\mathcal{A}'} = 0$.

Let there be some index i such that $a_i^{\mathcal{A}'} = 0$, therefore, by definition, $i > z$. For any $i' > i$ it also holds that $i' > z$ and therefore $a_{i'}^{\mathcal{A}'} = 0$. □

Claim 2.2.43. The structure $\mathcal{A}'$ satisfies $\tau(\varphi)$.

Proof. We show that $\mathcal{A}$ and $\mathcal{A}'$ are congruent, and since $\mathcal{A} \vDash \varphi$, from Lemma 2.2.41, $\mathcal{A}' \vDash \tau(\varphi)$:

1. $\mathcal{A}$ is a Sum model of φ, therefore it holds the Sum property.
2. $\mathcal{A}'$ satisfies $\eta(\varphi)$ from Claim 2.2.42.
3. $z = |A| \leq \tilde{\kappa}$ as explained above.
4. For any $i \in [1, l]$, $a_i^{\mathcal{A}} = \alpha_i$ by definition.
5. By construction of $\mathcal{A}'$, for any $j \in [1, m], i \in [1, z]$, $b_{i,j}^{\mathcal{A}'} = b_j^{\mathcal{A}}(\alpha_i)$ and for any $i > z$, $b_{i,j}^{\mathcal{A}'} = 0$.
6. By construction of $\mathcal{A}'$, for any $i \in [1, z]$, $a_i^{\mathcal{A}'} = 1 > 0$, and for any $i > z$, $a_i^{\mathcal{A}'} = 0$.
7. $\mathcal{A}$ is given to be distinct.

□

Corollary 2.2.43.1. The structure $\mathcal{A}'$ is a model for $\varphi' = \tau(\varphi) \wedge \eta(\varphi)$.

Part 3: Proof of Theorem 2.2.28 ($\Leftarrow$): If φ' has a model, then φ has a *distinct*, *small* Sum model

Let $\mathcal{A}' = \left(a_1^{\mathcal{A}'}, \ldots, a_{\tilde{\kappa}}^{\mathcal{A}'}, b_{1,1}^{\mathcal{A}'}, \ldots, b_{\tilde{\kappa},m}^{\mathcal{A}'}, c_1^{\mathcal{A}'}, \ldots, c_n^{\mathcal{A}'}\right)$ be a model for φ'. Since $\mathcal{A}' \vDash \eta_3(\varphi)$, we know that there exists some maximal index $z \leq \tilde{\kappa}$ such that $a_z^{\mathcal{A}'} \neq 0$ and for any $i > z$, $a_i^{\mathcal{A}'} = 0$. Since $\mathcal{A}' \vDash \eta_1(\varphi)$ we know that $z \geq l$.

We construct a model $\mathcal{A}$ for φ as follows:

$$\mathcal{A} = \left(A, a_1^{\mathcal{A}}, \ldots, a_l^{\mathcal{A}}, b_1^{\mathcal{A}}, \ldots, b_m^{\mathcal{A}}, c_1^{\mathcal{A}}, \ldots, c_n^{\mathcal{A}}, s_1^{\mathcal{A}}, \ldots, s_m^{\mathcal{A}}\right)$$

where the Addresses set is

$$A = [1, z];$$

the Address constants are

$$a_i^{\mathcal{A}} = i$$

for any $i \in [1, l]$; the balances are

$$b_j^{\mathcal{A}}(i) = b_{i,j}^{\mathcal{A}'}$$

for any $i \in A, j \in [1, m]$; the natural constants are $c_k^{\mathcal{A}} = c_k^{\mathcal{A}'}$ and the sums are defined as

$$s_j^{\mathcal{A}} = \sum_{\alpha \in A} b_j^{\mathcal{A}}(\alpha).$$

We show that $\mathcal{A}, \mathcal{A}'$ are congruent:

1. By construction, $\mathcal{A}$ holds the Sum property.
2. It is given that $\mathcal{A}'$ satisfies $\eta(\varphi)$.
3. We define A to be the set $[1, z]$, and therefore $|A| \leq \tilde{\kappa}$.
4. $a_i^{\mathcal{A}} = \alpha_i$ as defined above.
5. By construction, for any $j \in [1.m], i \in [1, z]$, $b_j^{\mathcal{A}}(\alpha_i) = b_{i,j}^{\mathcal{A}'}$ and z was chosen such that for any $i > z$, $b_{i,j}^{\mathcal{A}'} = 0$.
6. z was chosen such that for any $i \in [1, z]$, $a_i^{\mathcal{A}'} > 0$ and for any $i > z$, $a_i^{\mathcal{A}'} = 0$.

7. $\mathcal{A}$ is distinct by construction.

Given that $\mathcal{A}' \vDash \varphi'$, we know in particular that $\mathcal{A}' \vDash \tau(\varphi)$, and from Lemma 2.2.41, $\mathcal{A} \vDash \varphi$ as a many-sorted, first-order formula. Since $\mathcal{A}$ holds the Sum property, $\mathcal{A} \vDash_{\text{SL}} \varphi$. In addition, by construction $|A| = z \leq \tilde{\kappa}$. Therefore, $\mathcal{A}$ is a *distinct*, *small* Sum model for φ.

Q.E.D. Theorem 2.2.28.

2.3 Undecidable Sum Logics

We show how to encode the halting problem of a 2-counter machine using SL with 3 balance functions, thereby proving that it is an undecidable logic.

Let there be some 2-counter machine, whose transitions are encoded with the formula $\pi(c_1, c_2, p, c_1', c_2', p')$ with 6 free variables: 2 for each register, including the program counter (PC).

We assume w.l.o.g. that all three counters are strictly within the naturals, excluding 0. This allows us to use a zero balance as a special separating marker.

In addition, we assume that the program counter is 1 at the start of the execution, and that there exists a single halting statement at line H (where H is some known natural constant). I.e. the 2-counter machine halts if and only if the PC is equal to H.

2.3.1 Outline

We have 4 *Address* elements for each time-step, 3 of them hold one register each, and one is used to separate between each group of *Address* (see Figure 2.1, Table 2.2). We have 3 uninterpreted functions from *Address* to *Nat* ("balances"):

1. c: Cardinality function, used to force size constraints. We set its value for all addresses to be 1, and therefore the amount of addresses is s_c.

2. l: Labeling function, to order the time-steps. We choose one element to have a maximal value of $s_c - 1$, and ensure that l is injective. This means that the values of l are $[0, s_c - 1]$.

3. g: General purpose function, which holds either one of the registers, or zero to mark the *Address* element as a separating one.

Each group representing a time-step is 4 *Address* elements, ordered as follows:

1. First, a separating *Address* element x (where $g(x)$ is 0).

2. Then, the two general-purpose counters.

3. Lastly, the program counter.

In addition we have 2 *Address* constants, a_0 and a_1 which represent the program counter value at the start and end of the execution. The element a_1 also holds the maximal value of l — that is, $l(a_1) + 1 \approx s_c$ — and a_0 holds the fourth-minimal value, since each group has four elements, and the PC is last.

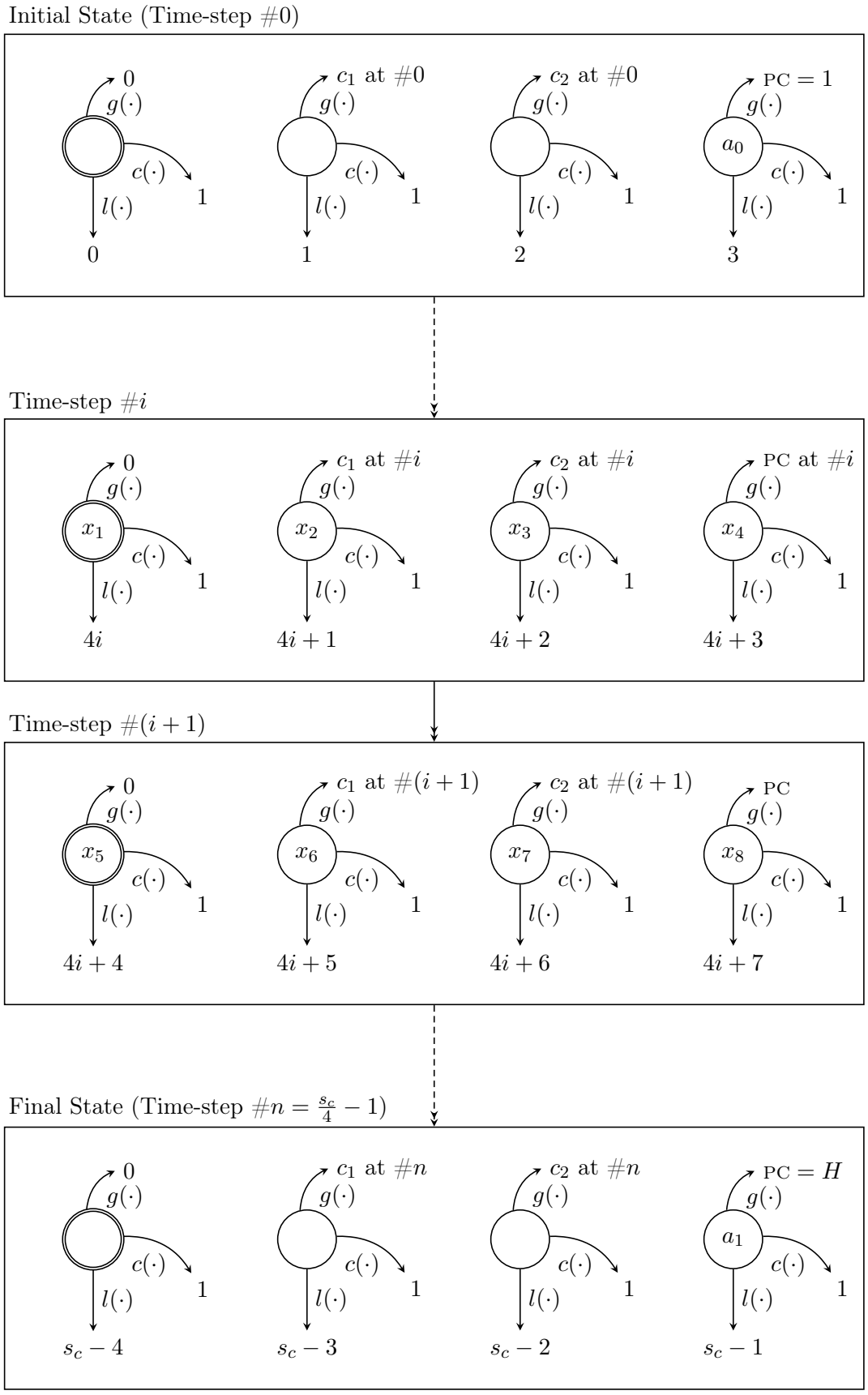

FIGURE 2.1: Transition System of a 2-Counter Machine in SL
Each ◯ represents an *Address*, doubled vertices are separators.

	Address	$l($***Address***$)$	$c($***Address***$)$	$g($***Address***$)$
Time-step $\#0$		0	1	0
		1	1	c_1 at $\#0$
		2	1	c_2 at $\#0$
	a_0	3	1	PC at $\#0 = 1$
	$\vdots$	$\vdots$	$\vdots$	$\vdots$
Time-step $\#i$	x_1	$4i$	1	0
	x_2	$4i+1$	1	c_1 at $\#i$
	x_3	$4i+2$	1	c_2 at $\#i$
	x_4	$4i+3$	1	PC at $\#i$
Time-step $\#(i+1)$	x_5	$4i+4$	1	0
	x_6	$4i+5$	1	c_1 at $\#(i+1)$
	x_7	$4i+6$	1	c_2 at $\#(i+1)$
	x_8	$4i+7$	1	PC at $\#(i+1)$
	$\vdots$	$\vdots$	$\vdots$	$\vdots$
Time-step $\#n = \frac{s_c}{4} - 1$		$s_c - 4$	1	0
		$s_c - 3$	1	c_1 at $\#n$
		$s_c - 2$	1	c_2 at $\#n$
	a_1	$s_c - 1$	1	PC at $\#n = H$

TABLE 2.2: Transition System of a 2-Counter Machine, Array View

2.3.2 Formalization

- Labeling is injective:

$$\varphi_1 = \forall x, y.(l(x) \approx l(y)) \rightarrow (x \approx y)$$

- The *Address* constant that represents the program counter value of the last time-step has the maximal labeling:

$$\varphi_2 = \forall x.l(x) \leq l(a_1)$$

- The *Address* constant that represents the program counter value of the first time-step has the fourth labeling:

$$\varphi_3 = l(a_0) \approx 3$$

- The first and last values of the program counter are 1 and H respectively:

$$\varphi_4 = g(a_0) \approx 1 \wedge g(a_1) \approx H$$

- Cardinality constraints; there are as many *Address* elements as the labeling of the last *Address* constant $(a_1) + 1$:

$$\varphi_5 = (s_c \approx l(a_1) + 1) \wedge \forall x.(c(x) \approx 1)$$

- Encoding the transitions of the 2-counter machine - for every 8 *Address* elements, if they represent two sequential time-steps, then the formula for the transitions of the 2-counter machine must be true for the registers they hold:

$$\begin{aligned}\varphi_6 = \forall x_1, \ldots, x_8.(&x_1, \ldots, x_8 \text{ represent two sequential time-steps}) \\ &\rightarrow \pi(g(x_2), g(x_3), g(x_4), g(x_6), g(x_7), g(x_8))\end{aligned}$$

  Representing sequential time-steps means having sequential labeling, and starting with one zero-valued *Address* element, continuing with 3 non-zero elements:

  - Sequential:
$$l(x_2) \approx l(x_1) + 1 \wedge \cdots \wedge l(x_8) \approx l(x_7) + 1$$
  - Time-steps:
$$g(x_1) \approx 0 \wedge g(x_2) > 0 \wedge g(x_3) > 0 \wedge g(x_4) > 0$$
    and
$$g(x_5) \approx 0 \wedge g(x_6) > 0 \wedge g(x_7) > 0 \wedge g(x_8) > 0$$

Combining all of the formulas above, we get that $\varphi = \varphi_1 \wedge \cdots \wedge \varphi_6$ is satisfiable $\iff$ the two-counter machine halts within a finite amount of time-steps (and the exact amount would be given by $\frac{s_c}{4}$).

Since the halting problem for 2-counter machines is undecidable, SL (with 3 uninterpreted functions and their associated sums) is also undecidable.

Remark. It is interesting to note that the only use of associated sums in the above formalization is for expressing the size of the set of *Address* elements.

The uninterpreted function $c(\cdot)$ is always 1 — $\forall x.c(x) \approx 1$ — and its sum s_c is thus simply the amount of addresses.

This gives the following corollary: we can encode the halting problem for 2-counter machines in an almost identical way, using an extension of Presburger arithmetic with two uninterpreted functions (for $l(\cdot)$ and $g(\cdot)$), and a *size operation* (which would replace $c(\cdot)$).

Chapter 3

Theory of Explicit Coins for Implicit Summations (ECIS)

3.1 Syntax & Semantics

In essence, ECIS is a theory of two-sorted, first-order logic, with uninterpreted relations. We represent the verification problem as m markets where money is represented by discrete *Coin* elements, and similarly to SL, we have *Address* elements that hold those *Coin* elements.

3.1.1 Syntax

Definition 3.1.1 (ECIS vocabulary)**.** A vocabulary

$$\Sigma^{l,m} = \left(a_1, \ldots, a_l, I_1^1, \ldots, I_m^1, H_1^2, \ldots, H_m^2\right)$$

where

1. We have in mind two sorts: *Address* and *Coin*.
2. The constants $a_1, \ldots, a_l$ are of *Address* sort.
3. $I_1^1, \ldots, I_m^1$ ("is-active") are predicates of *Coin* sort, marking each element as active in a specific market.
4. $H_1^2, \ldots, H_m^2$ ("has-coin") are binary relations between *Address* and *Coin*, which mark whether some *Address* element is the owner of some *Coin* element.

For each $j \in [1, m]$, we call the pair $\left(I_j^1, H_j^2\right)$ the j^{th} *market*. We almost always consider the case where $m = 2$, where we have a "before" market and "after" market (in relation to some transition).

Notation. When the cardinalities of the vocabulary are clear from context, we simply denote it as Σ.

Remark. The reason we use a "has-coin" relation, instead of a function from *Coin* to *Address* is that we want to make sure to stay inside the EPR fragment of ECIS, whenever possible, so we avoid uninterpreted functions. In addition, inactive coins should not be able to denote an *Address* element as their "owner".

3.1.2 Semantics

Definition 3.1.2. Let Σ be an ECIS vocabulary. A two-sorted structure $\mathcal{A} = (\mathcal{D}, \mathcal{I}) \in$ STRUCT[Σ] is a tuple

$$\mathcal{A} = \left(A, C, a_1^{\mathcal{A}}, \ldots, a_l^{\mathcal{A}}, I_1^{\mathcal{A}}, \ldots, I_m^{\mathcal{A}}, H_1^{\mathcal{A}}, \ldots, H_m^{\mathcal{A}}\right)$$

where $A = \mathcal{D}(Address)$ and $C = \mathcal{D}(Coin)$ are some sets; $a_i^{\mathcal{A}} = \mathcal{I}(a_i) \in A$; $I_j^{\mathcal{A}} = \mathcal{I}(I_j) \subseteq C$; and $H_j^{\mathcal{A}} = \mathcal{I}(H_j) \subseteq A \times C$.

Remark. In quantifiers we usually omit the sort markers, with the convention that *Address* quantifiers use variables x, x', x_1, etc. and *Coin* quantifiers use variables c, c', c_1, etc.

Definition 3.1.3. Let φ be a sentence over an ECIS vocabulary Σ. An ECIS model for φ is an ECIS structure $\mathcal{A} \in \text{STRUCT}[\Sigma]$ such that $\mathcal{A} \vDash \varphi$.

3.1.3 Axioms

When verifying programs using ECIS we want to maintain several axioms that ensure the encoding in ECIS indeed represents *Address* elements with balances. For this, we have the following three formulas, for each market $j \in [1, m]$:

1. Only active *Coin* elements can be owned:

$$\psi_{j,1} \triangleq \forall c.[(\exists x.H_j(x,c)) \to I_j(c)] \qquad \text{(3.1 "Active Coins")}$$

2. Every active *Coin* element is owned by some *Address* element:

$$\psi_{j,2} \triangleq \forall c.[I_j(c) \to \exists x.H_j(x,c)] \qquad \text{(3.2 "At Least")}$$

3. Every active *Coin* element is owned by at most one *Address* element:

$$\psi_{j,3} \triangleq \forall c.\forall x_1, x_2. \qquad \text{(3.3 "At Most")}$$
$$[(H_j(x_1,c) \wedge H_j(x_2,c)) \to x_1 \approx x_2]$$

Remark. Though a more straight-forward formulation of these axioms would have used only one or two formulas — specifically merging Axioms 3.2 "At Least" and 3.3 "At Most" into one — we prefer this formulation. The simpler, multiple formulas are easier for solvers to handle, and thus make it more likely that verification finishes within a reasonable time-frame.

The entire axiomatic requirement for market j is then $\psi_j \triangleq \psi_{j,1} \wedge \psi j, 2 \wedge \psi j, 3$ and we get the following theorem:

Theorem 3.1.4. Let $\Sigma = \Sigma^{l,m}$ be an ECIS vocabulary, and let $\mathcal{A} \in \text{STRUCT}[\Sigma]$ be some structure of Σ. For any market $j \in [1, m]$:

$$\mathcal{A} \vDash \psi_j \Rightarrow \left|I_j^{\mathcal{A}}\right| = \sum_{\alpha \in A} \left|\left\{c \in C \mid (a,c) \in H_j^{\mathcal{A}}\right\}\right|$$

I.e., ψ_j exactly encodes the condition that the amount of the active *Coin* elements equals to the sum of the amount of *Coin* elements that belong to each *Address* element.

***Proof of* Theorem 3.1.4**

Let $\mathcal{A}$ be some model that holds Axioms 3.1 "Active Coins", 3.2 "At Least" and 3.3 "At Most".

First, we define 3 sets of errors that can occur within a model:

Definition 3.1.5. Inactivity errors, where an inactive Coin is owned by some Address:

$$E_{\text{inactive}}^{\mathcal{A}} \triangleq \left\{\gamma \in C \mid \gamma \notin I_j^{\mathcal{A}} \wedge \exists \alpha \in A.(\alpha, \gamma) \in H_j^{\mathcal{A}}\right\}$$

Definition 3.1.6. Dangling errors, where an active Coin is not owned by any Address:

$$E^{\mathcal{A}}_{\text{dangling}} \triangleq \left\{\gamma \in I^{\mathcal{A}}_j \mid \forall \alpha \in A.(\alpha,\gamma) \notin H^{\mathcal{A}}_j\right\}$$

Definition 3.1.7. Double-owned errors, where a Coin is owned by more than one Address:

$$E^{\mathcal{A}}_{\text{doubled}} \triangleq \left\{\gamma \in C \mid \exists \alpha \neq \beta.(\alpha,\gamma),(\beta,\gamma) \in H^{\mathcal{A}}_j\right\}$$

Now we can define the total amount of errors:

Definition 3.1.8. Total amount of errors:

$$e^{\mathcal{A}} \triangleq \left|E^{\mathcal{A}}_{\text{inactive}}\right| + \left|E^{\mathcal{A}}_{\text{dangling}}\right| + \left|E^{\mathcal{A}}_{\text{doubled}}\right|$$

Finally, we define the sets of owned Coins:

Definition 3.1.9. Let $\alpha \in A$ be some Address:

$$S^{\mathcal{A}}_{j,\alpha} \triangleq \left\{\gamma \in C \mid (\alpha,\gamma) \in H^{\mathcal{A}}_j\right\}$$

We need to prove that

$$\left|I^{\mathcal{A}}_j\right| = \sum_{\alpha \in A} \left|S^{\mathcal{A}}_{j,\alpha}\right|,$$

for which we need the following lemmas:

Lemma 3.1.10. If $e^{\mathcal{A}} = 0$, then

$$I^{\mathcal{A}}_j = \bigcup_{\alpha \in A} S^{\mathcal{A}}_{j,\alpha}$$

Proof. We will show bidirectional inclusion:

Let there be some $\gamma \in I^{\mathcal{A}}_j$. Since $e^{\mathcal{A}} = 0$ we know that $E^{\mathcal{A}}_{\text{dangling}} = \emptyset$. Therefore, there exists some $\alpha \in A$ such that $(\alpha,\gamma) \in H^{\mathcal{A}}_j$ and therefore $\gamma \in S^{\mathcal{A}}_{j,\alpha}$.

Now, let there be some $\gamma \in \bigcup_{\alpha \in A} S^{\mathcal{A}}_{j,\alpha}$. This means that there exists some $\alpha \in A$ such that $\gamma \in S^{\mathcal{A}}_{j,\alpha}$, or put differently, $(\alpha,\gamma) \in H^{\mathcal{A}}_j$. Since $E^{\mathcal{A}}_{\text{inactive}} = \emptyset$ it must hold that $\gamma \in I^{\mathcal{A}}_j$. □

Lemma 3.1.11. If $e^{\mathcal{A}} = 0$, then the sets $\left\{S^{\mathcal{A}}_{j,\alpha} \mid \alpha \in A\right\}$ are pairwise-disjoint.

Proof. Let there be two Addresses $\alpha, \beta \in A$. The set $S^{\mathcal{A}}_{j,\alpha} \cap S^{\mathcal{A}}_{j,\beta}$ is exactly the set of coins γ such that $(\alpha,\gamma),(\beta,\gamma) \in H^{\mathcal{A}}_j$, and in particular

$$S^{\mathcal{A}}_{j,\alpha} \cap S^{\mathcal{A}}_{j,\beta} \subseteq E^{\mathcal{A}}_{\text{doubled}}$$

However, since $e^{\mathcal{A}} = 0$ we know that $E^{\mathcal{A}}_{\text{doubled}} = \emptyset$ and therefore sets $S^{\mathcal{A}}_{j,\alpha}, S^{\mathcal{A}}_{j,\beta}$ are disjoint. □

Corollary 3.1.11.1. If $e^{\mathcal{A}} = 0$, then

$$\left|I^{\mathcal{A}}_j\right| = \left|\bigcup_{\alpha \in A} S^{\mathcal{A}}_{j,\alpha}\right| = \sum_{\alpha \in A} \left|S^{\mathcal{A}}_{j,\alpha}\right|$$

Proof. From the previous lemmas we know that since $e^{\mathcal{A}} = 0$, $I^{\mathcal{A}}_j = \bigcup_{\alpha \in A} S^{\mathcal{A}}_{j,\alpha}$ and that the sets are disjoint. This corollary is therefore trivial. □

Lemma 3.1.12. If $\mathcal{A}$ holds Axiom 3.1 "Active Coins", then $E^{\mathcal{A}}_{\text{inactive}} = \emptyset$.

Proof. Axiom 3.1 "Active Coins" states that

$$\forall c.(\exists a.H_j(a,c)) \to I_j(c).$$

This formula is equivalent to

$$\forall c.\forall a.\neg H_j(a,c) \vee I_j(c).$$

Let us assume towards contradiction that there exists some $\gamma \in E^{\mathcal{A}}_{\text{inactive}}$, therefore there exists some Address α such that $(\alpha,c) \in H^{\mathcal{A}}_j$ but $\gamma \notin I^{\mathcal{A}}_j$.

This means that $\mathcal{A} \nvDash \neg H_j(a,c) \vee I_j(c)[\alpha/a,\gamma/c]$, and therefore $\mathcal{A} \nvDash \forall c.\forall a.\neg H_j(a,c) \vee I_j(c)$, which means that $\mathcal{A} \nvDash$ Axiom 3.1 "Active Coins" — contradiction. □

Lemma 3.1.13. If $\mathcal{A}$ holds Axiom 3.2 "At Least" then $E^{\mathcal{A}}_{\text{dangling}} = \emptyset$.

Proof. Axiom 3.2 "At Least" states that

$$\forall c.I_j(c) \to \exists a.H_j(a,c).$$

Let us assume towards contradiction that $E^{\mathcal{A}}_{\text{dangling}} \neq \emptyset$, and let there be some $\gamma \in E^{\mathcal{A}}_{\text{dangling}}$. Therefore $\gamma \in I^{\mathcal{A}}_j$ but there exists no Address $\alpha \in A$ such that $(\alpha,\gamma) \in H^{\mathcal{A}}_j$. Therefore, $\mathcal{A} \nvDash$ Axiom 3.2 "At Least" — contradiction. □

Lemma 3.1.14. If $\mathcal{A}$ holds Axiom 3.3 "At Most" then $E^{\mathcal{A}}_{\text{doubled}} = \emptyset$.

Proof. Axiom 3.3 "At Most" states that

$$\forall c.\forall a_1,a_2.(I_j(c) \wedge H_j(a_1,c) \wedge H_j(a_2,c)) \to a_1 \approx a_2,$$

or equivalently

$$\forall c.\forall a_1,a_2.\neg H_j(a_1,c) \vee \neg H_j(a_2,c) \vee a_1 \approx a_2$$

Let us assume towards contradiction that $E^{\mathcal{A}}_{\text{doubled}} \neq \emptyset$ and let $\gamma \in E^{\mathcal{A}}_{\text{doubled}}$. Therefore, there exist $\alpha \neq \beta \in A$ such that $(\alpha,\gamma),(\beta,\gamma) \in H^{\mathcal{A}}_j$. So we get:

1. $\mathcal{A} \vDash H_j(a_1,c)[\alpha/a_1,\gamma/c]$.
2. $\mathcal{A} \vDash H_j(a_2,c)[\beta/a_2,\gamma/c]$.
3. $\mathcal{A} \nvDash a_1 \approx a_2[\alpha/a_1,\beta/a_2]$.

Combined we get that there exist $\gamma \in C, \alpha,\beta \in A$ such that

$$\mathcal{A} \nvDash \neg H_j(a_1,c) \vee \neg H_j(a_2,c) \vee a_1 \approx a_2[\alpha/a_1,\beta/a_2,\gamma/c]$$

and therefore $\mathcal{A} \nvDash$ Axiom 3.3 "At Most" — contradiction. □

Combining all of the above, since $\mathcal{A}$ holds Axioms 3.1 "Active Coins", 3.2 "At Least" and 3.3 "At Most" then $e^{\mathcal{A}} = 0+0+0 = 0$ and therefore $\left|I^{\mathcal{A}}_j\right| = \sum_{\alpha\in A}\left|S^{\mathcal{A}}_{j,\alpha}\right|$.

Q.E.D. Theorem 3.1.4.

Remark. It is important to note that although SL and ECIS encode similar higher-order settings, in ECIS we can express much less about that setting. Whereas in SL the balances were explicit and reified in the uninterpreted functions, in ECIS the balances are an emergent phenomenon, described implicitly by the "has-coin" relation. In fact, we see in Section 3.3 that when the balances become tangible in ECIS, we have the same undecidability issues as in SL.

3.2 Encoding in a Limited Fragment of ECIS

The EPR fragment of ECIS is naturally decidable, since it is a special case of EPR in many-sorted logic, without function symbols. The interesting question is therefore how expressible this fragment is (since decidability is trivial). We see that some interesting properties and transactions can still be described in this limited logic. This is inspired by a similar process used in [10].

3.2.1 Axioms in ECIS ∩ EPR

All quantifier alternations in Axioms 3.1 "Active Coins", 3.2 "At Least" and 3.3 "At Most" are between different sorts, and always "$\forall$ *Coin* $\exists$ *Address*". Therefore the formulas are in EPR, since no cycles in the sort dependency graph are possible.

3.2.2 Transitions in ECIS ∩ EPR

For each transition we assume to have two markets, a "before" unmarked market $I(\cdot), H(\cdot,\cdot)$ and an "after" primed market $I'(\cdot), H'(\cdot,\cdot)$. In the ECIS formulation, each transition is concerned with a single coin, instead of a `uint` parameter that denotes the amount of tokens that change hands.

The transitions' formulas are written as $\texttt{transition}(x_1, \ldots, x_r)$ where r is the arity of the transition and $x_1, \ldots, x_r$ are free variables of the *Address* sort. *Coin* elements affected by a transition are usually existentially quantified over since they are not thought of as arguments to the transition. However, they can be expressed as arguments just as well. In which case, we have $\texttt{transition}(x_1, \ldots, x_r, c_1, \ldots, c_{\tilde{r}})$.

`transferFrom`(x_1, x_2) Transition

This transition encodes a transfer of a single *Coin* element from x_1 to x_2, if x_1 holds at least one element (otherwise, the final state should be identical to the initial state).

This is encoded by the conjunction of the following 4 formulas:

1. Arguments must be distinct:

$$\varphi_1 = x_1 \not\approx x_2$$

2. No *Coin* element changes its activity:

$$\varphi_2 = \forall c.\big(I(c) \leftrightarrow I'(c)\big)$$

3. If x_1 holds no *Coin* elements, nothing happens:

$$\varphi_3 = (\neg\exists c.H(x_1, c)) \rightarrow \forall c.\forall x.\big(H(x, c) \leftrightarrow H'(x, c)\big)$$

4. Finally, if x_1 holds some *Coin* element, then some element that previously belonged to x_1 will now belong to x_2 and no other coin will change hands:

$$\begin{aligned}\varphi_4 = &\Big(\exists c.H(x_1,c)\Big)\\ &\to \exists c.\Bigg(H(x_1,c)\wedge\neg H'(x_1,c)\wedge H'(x_2,c)\\ &\qquad\qquad\wedge\\ &\qquad\forall x'.\forall c'.\Bigg(\Big(c'\not\approx c\vee(x'\not\approx x_1\wedge x'\not\approx x_2)\Big)\\ &\qquad\qquad\to\Big(H(x',c')\leftrightarrow H'(x',c')\Big)\Bigg)\Bigg)\end{aligned}$$

`throw`(x_1) and `catch`(x_2) Transitions

This is an alternative way to express a transfer of a *Coin* element, in two steps. We require 3 markets in order to encode the complete transition: An initial state (I,H), an intermediary state (I',H') and a final state (I'',H''). The intermediary state does not hold the axioms of the theory, but as long as the initial state is valid, the final state will be as well.

The `throw` transition is encoded as a conjunction of the following formulas:

1. No *Coin* element changes its activity:

$$\varphi_{1,1} = \forall c.\big(I(c)\leftrightarrow I'(c)\big)$$

2. If x_1 has a *Coin* element, it frees up some element it holds, and no other element is affected:

$$\begin{aligned}\varphi_{1,2} = &\Big(\exists c.H(x_1,c)\Big)\\ &\to \exists c.\Bigg(H(x_1,c)\wedge\neg H'(x_1,c)\\ &\qquad\qquad\wedge\\ &\qquad\forall x'.\forall c'.\Bigg(\Big(c'\not\approx c\vee x'\not\approx x_1\Big)\\ &\qquad\qquad\to\Big(H(x',c')\leftrightarrow H'(x',c')\Big)\Bigg)\Bigg)\end{aligned}$$

The `catch` transition is encoded as a conjunction of the following formulas:

1. No *Coin* element changes its activity:

$$\varphi_{2,1} = \forall c.\big(I'(c)\leftrightarrow I''(c)\big)$$

2. If there is a free *Coin* element "up in the air", then x_2 will take it (and no other element is affected):

$$\begin{aligned}\varphi_{2,2} = &\Big(\exists c.(I'(c) \wedge \forall x.\neg H'(x,c))\Big) \\ &\rightarrow \exists c.\Bigg(I'(c) \wedge (\forall x.\neg H'(x,c)) \wedge H''(x_2,c) \\ &\qquad\qquad \wedge \\ &\qquad \forall x'.\forall c'.\Bigg(\Big(c' \not\approx c \vee x' \not\approx x_2\Big) \\ &\qquad\qquad \rightarrow \Big(H'(x',c') \leftrightarrow H''(x',c')\Big)\Bigg)\Bigg)\end{aligned}$$

Note. In this formulation, there is no need for x_1 and x_2 to be distinct. I.e. a single *Address* element can throw and catch a coin. In which case I, H and I'', H'' will be identical.

`transfer`(x, c) Transition

A single *Coin* c is minted (becomes active) and put into *Address* a. This transition is encoded by the conjunction of the following addresses:

1. The *Coin* element was previously inactive, and will activated:

$$\varphi_1 = \neg I(c) \wedge I'(c)$$

2. The coin is now owned by x:

$$\varphi_2 = H(x,c)$$

3. Nothing else is changed:

$$\begin{aligned}\varphi_3 = \forall x'.\forall c'.\Big(&(x' \not\approx a \vee c' \not\approx c) \\ &\rightarrow \big(I(c') \leftrightarrow I'(c') \\ &\qquad\qquad \wedge \\ &\qquad H(x',c') \leftrightarrow H'(x',c')\big)\Big)\end{aligned}$$

`burn`(c) Transition

A single *Coin* element c is removed from circulation (and taken out of the *Address* element that holds it). This is the reverse of `transfer`(x, c), and though it is not part of the ERC-20 Token Standard interface, it is presented here for completeness. It is encoded by the conjunction of the following formulas:

1. The *Coin* was previously active, and is now deactivated:

$$\varphi_1 = I(c) \wedge \neg I'(c)$$

2. The Coin no longer belongs to any Address:

$$\varphi_2 = \forall x.\neg H'(x,c)$$

3. Nothing else changes:

$$\varphi_3 = \forall x'.\forall c'.\Big((c' \not\approx c) \rightarrow \big(I(c') \leftrightarrow I'(c') \wedge H(x',c') \leftrightarrow H'(x',c')\big)\Big)$$

3.3 Undecidabiliy Result in ECIS

In this section we show that by extending ECIS with *Nat* sort, uninterpreted functions (of sort $Coin \rightarrow Nat$, and $(Address, Coin) \rightarrow Nat$), addition, and ordering over *Nat*, we can encode for each market $j \in [1,m]$ its size s_j and for each *Address* a, its balance $b_j(a)$.

We denote this extension as ECIS $\cup$ Nat, and we note that we have no quantifiers over *Nat*.

By Theorem 3.1.4, any model $\mathcal{A}$ that holds Axioms 3.1 "Active Coins", 3.2 "At Least" and 3.3 "At Most" for market j, also holds the Sum property for it, i.e.

$$s_j^{\mathcal{A}} = \sum_{\alpha \in A} b_j^{\mathcal{A}}(\alpha)$$

without requiring it semantically of the model.

Once we have the s_j constants and the b_j^1 functions, the same encoding of the halting problem of 2-counter machines as in Section 2.3 can be expressed in ECIS; hence, ECIS $\cup$ Nat is undecidable (for 3 markets or more).

3.3.1 General Idea

The theme of this encoding is that we order coins in a successive sequence, starting with 0, and therefore the amount of coins can be expressed as one over the "place" of the last coin. We encode this type of ordering for each market (i.e. we order all of the active coins within it), and for each address in each market (we order its owned coins).

3.3.2 Encoding Ordering

Global Ordering

First, we need m ordering functions from *Coin* to *Nat*. These allow us to count the active *Coin* elements in each market. We add an uninterpreted function $g_j : Coin \rightarrow Nat$ for each $j \in [1,m]$.

We require the following:

1. The range of g_j for active *Coin* elements is continuous; i.e. every coin is either the last one, or it has a coin that directly follows it:

$$\forall c.\Big(I_j(c) \rightarrow \Big(\big(\forall c'.(I_j(c') \rightarrow g_j(c') \leq g_j(c))\big) \bigvee \big(\exists c'.I_j(c') \wedge g_j(c') \approx g_j(c)+1\big)\Big)\Big) \qquad \text{(3.4 "Global Continuous")}$$

2. The function g_j is 1:1 for active *Coin* elements:

$$\forall c_1, c_2.[(I_j(c_1) \wedge I_j(c_2) \wedge c_1 \not\approx c_2) \rightarrow (g_j(c_1) \not\approx g_j(c_2))] \quad \text{(3.5 "Global Injective")}$$

3. If there is at least one active *Coin* element, then some active element is mapped to 0 (which is the minimal possible mapping):

$$(\exists c.I_j(c)) \rightarrow \exists c.(I_j(c) \wedge g_j(c) \approx 0) \quad \text{(3.6 "Global Zero")}$$

Per-Address, "Local" Ordering

Next we need another m ordering functions, which take a pair of (*Address*, *Coin*) and give the order of that *Coin*, within that *Address* space (if the *Address* holds that *Coin*). We add an interpreted function $l_j : (Address, Coin) \rightarrow Nat$ for each $j \in [1, m]$.

We require the following:

1. The range of l_j for owned *Coin* elements is continuous:

$$\forall x.\forall c.\Big(H_j(x,c) \rightarrow \Big(\big(\forall c'.(H_j(x,c') \rightarrow l_j(x,c') \leq l_j(x,c))\big) \bigvee \big(\exists c'.I_j(c') \wedge l_j(x,c') \approx l_j(x,c)+1\big)\Big)\Big) \quad \text{(3.7 "Per-Address Continuous")}$$

2. The function l_j is 1:1 for owned *Coin* elements:

$$\forall x.\forall c_1, c_2.\Big(\Big(H_j(x,c_1) \wedge H_j(x,c_2) \wedge c_1 \not\approx c_2\Big) \rightarrow \Big(l_j(x,c_1) \not\approx l_j(x,c_2)\Big)\Big) \quad \text{(3.8 "Per-Address Injective")}$$

3. If the *Address* element owns at least one *Coin* element, then some owned element is mapped to 0 (which is the minimal possible mapping):

$$\forall x.[(\exists c.H_j(x,c)) \rightarrow \exists c.H_j(c) \wedge l_j(x,c) \approx 0] \quad \text{(3.9 "Per-Address Zero")}$$

3.3.3 Encoding Market Size

Once we have the ordering of the *Coin* elements, and we know that for active *Coin* elements the range is exactly $\left[0, \max_{c \in I_j^{\mathcal{A}}} \left\{g_j^{\mathcal{A}}(c)\right\}\right]$, we can encode the size of the market as the successor of the last *Coin* element (or 0, when there are no elements):

$$\Big(\exists c.\Big(I_j(c) \wedge \big(\forall c'.[I_j(c') \rightarrow g_j(c') \leq g_j(c)]\big) \wedge s_j \approx g_j(c)+1\Big)\Big) \bigvee \Big(s_j \approx 0 \wedge \forall c.[\neg I_j(c)]\Big) \quad \text{(3.10 "Market Size")}$$

3.3.4 Encoding Balances

In a similar way, we can encode the balance of each *Address* element in each market as the successor of the last owned *Coin* element (again, or 0 when there are none):

$$\forall x.\Bigg(\Big(\exists c.\Big(H_j(x,c) \\ \wedge\big(\forall c'.[H_j(x,c') \rightarrow l_j(x,c') \leq l_j(x,c)]\big) \\ \wedge b_j(x) \approx g_j(c)+1\Big)\Big) \\ \bigvee\Big(b_j(x) \approx 0 \wedge \forall c.[\neg H_j(x,c)]\Big)\Bigg) \tag{3.11 "Balance"}$$

3.3.5 Expressing SL using ECIS ∪ Nat

Using the axioms defined in the previous sections, we can express any formula in SL. Moreover, there's no need to semantically require that models hold the Sum property — the conjunction of Axioms 3.1 "Active Coins", 3.2 "At Least" and 3.3 "At Most" ensures that for any model $\mathcal{A}$, if $s_j^{\mathcal{A}} = |I_j|$ and $b_j^{\mathcal{A}}(\alpha) = \left|\left\{\gamma \in C \mid (\alpha,\gamma) \in H_j^{\mathcal{A}}\right\}\right|$, then it must hold that

$$s_j^{\mathcal{A}} = \sum_{\alpha \in A} b_j^{\mathcal{A}}(\alpha_)$$

from Theorem 3.1.4. We will prove that indeed if $\mathcal{A}$ holds Axioms 3.4 "Global Continuous", 3.5 "Global Injective", 3.6 "Global Zero" and 3.10 "Market Size", then $s_j^{\mathcal{A}} = |I_j|$. The proof for $b_j^{\mathcal{A}}(\alpha)$ — given Axioms 3.7 "Per-Address Continuous", 3.8 "Per-Address Injective", 3.9 "Per-Address Zero" and 3.11 "Balance" — is identical, and therefore omitted.

Lemma 3.3.1. Let $\mathcal{A}$ be a structure for ECIS vocabulary $\Sigma^{l,m}$, and let j be some market such of size $n = \left|I_j^{\mathcal{A}}\right| > 0$.

If $\mathcal{A}$ holds Axioms 3.4 "Global Continuous", 3.5 "Global Injective" and 3.6 "Global Zero", then

$$\max_{\gamma \in I_j}\left\{g^{\mathcal{A}}(\gamma)\right\} = n-1.$$

Proof. Let us denote the co-domain of $g_j^{\mathcal{A}}$ when projected on $I_j^{\mathcal{A}}$ as S:

$$S \triangleq \left\{g_j^{\mathcal{A}}(\gamma) \mid \gamma \in I_j^{\mathcal{A}}\right\}.$$

We will prove that $S = [0, n-1]$, and in particular $\max S = n-1$.

Since $\left|I_j^{\mathcal{A}}\right| > 0$ we know that there's some coin $\gamma \in I_j$ such that $g_j^{\mathcal{A}}(\gamma) = 0$, according to Axiom 3.6 "Global Zero":

$$(\exists c.I_j(c)) \rightarrow (\exists c.I_j(c) \wedge g_j(c) \approx 0).$$

This means that $0 \in S$.

Since $\mathcal{A} \vDash$ Axiom 3.5 "Global Injective", there are no two elements $\gamma_1 \neq \gamma_2 \in I_j^{\mathcal{A}}$ such that $g_j^{\mathcal{A}}(\gamma_1) = g_j^{\mathcal{A}}(\gamma_2)$. I.e. $|S| = \left|I_j^{\mathcal{A}}\right| = n$.

Lastly, there can be no gap S. Let assume towards contradiction that there exist some elements $x, y \in S$ such that $y > x+1$ and $x+1 \notin S$. Let $\gamma_1, \gamma_2 \in I_j^{\mathcal{A}}$ be the

sources of x and y respectively:

$$g_j^{\mathcal{A}}(\gamma_1) = x, \qquad g^{\mathcal{A}}(\gamma_2) = y.$$

$\mathcal{A}$ holds Axiom 3.4 "Global Continuous", and in particular

$$\mathcal{A} \vDash I_j(c) \to \big(\big(\forall c'.I_j(c') \to g_j(c') \leq g_j(c)\big) \vee \big(\exists c'.I_j(c') \wedge g_j(c') \approx g_j(c) + 1\big)\big)[\gamma_1/c].$$

Since $\gamma_1 \in I_j$, it must be that

$$\mathcal{A} \vDash \big(\big(\forall c'.I_j(c') \to g_j(c') \leq g_j(c)\big) \vee \big(\exists c'.I_j(c') \wedge g_j(c') \approx g_j(c) + 1\big)\big)[\gamma_1/c].$$

We know that for $\gamma_2 \in I_j$, $g_j^{\mathcal{A}}(\gamma_2) = y > x = g_j^{\mathcal{A}}(\gamma_1)$, and therefore

$$\mathcal{A} \nvDash I_j(c') \to g_j(c') \leq g_j(c)\big[\gamma_1/c, \gamma_2/c'\big].$$

So it must be the case that

$$\mathcal{A} \vDash \big(\exists c'.I_j(c') \wedge g_j(c') \approx g_j(c) + 1\big)[\gamma_1/c]$$

and there must be some γ_3 such that

$$\mathcal{A} \vDash I_j(c') \wedge g_j(c') \approx g_j(c) + 1\big[\gamma_1/c, \gamma_3/c'\big]$$

I.e. $\gamma_3 \in I_j^{\mathcal{A}}$ and $g_j^{\mathcal{A}}(\gamma_3) = g_j^{\mathcal{A}}(\gamma_1) + 1$ — contradicting our assumption that $x+1 \notin S$.

Combining all of the above, we get that S has no gaps, contains 0, and is of size n — meaning it must be that $S = [0, n-1]$, as required. □

Theorem 3.3.2. Let $\mathcal{A}$ be a structure for ECIS vocabulary $\Sigma^{l,m}$, and let j be some market such of size $n = \left|I_j^{\mathcal{A}}\right|$.

If $\mathcal{A}$ holds Axioms 3.4 "Global Continuous", 3.5 "Global Injective", 3.6 "Global Zero" and 3.10 "Market Size", then $s_j^{\mathcal{A}} = n$.

***Proof of* Theorem 3.3.2**

Let us consider the following two cases:

Case 1: $n = 0$

We know that $\left|I_j^{\mathcal{A}}\right| = 0$, and therefore, for any $\gamma \in C$, $\gamma \notin I_j^{\mathcal{A}}$, meaning

$$\mathcal{A} \nvDash \exists c.I_j(c).$$

Since $\mathcal{A}$ holds Axiom 3.10 "Market Size", it must be that

$$\mathcal{A} \vDash s_j \approx 0 \wedge \forall c.\neg I_j(c).$$

In particular, $s_j^{\mathcal{A}} = 0 = n$, as required.

Case 2: $n > 0$

In a similar fashion, we can conclude that

$$\mathcal{A} \vDash \exists c.\big(I_j(c) \wedge \big(\forall c'.\big(I_j(c') \to g_j(c') \leq g_j(c)\big)\big) \wedge s_j \approx g_j(c) + 1\big),$$

since

$$\mathcal{A} \nvDash s_j \approx 0 \wedge \forall c.\neg I_j(c).$$

Let there be some $\gamma \in C$ such that

$$A \vDash I_j(c) \wedge \big(\forall c'.\big(I_j(c') \rightarrow g_j(c') \leq g_j(c)\big)\big) \wedge s_j \approx g_j(c) + 1[\gamma/c]$$

and it must be that $g^{\mathcal{A}}(\gamma)$ is maximal among $I_j^{\mathcal{A}}$.

From Lemma 3.3.1, we know that $g^{\mathcal{A}}(\gamma) = n-1$, and therefore $s_j^{\mathcal{A}} = n-1+1 = n$, as required.

Q.E.D. Theorem 3.3.2.

Chapter 4

Conclusions

In this thesis we have presented two approaches to reason about unbounded sums in first-order logic, with the specific goal of formally verifying the correctness of programs written with the ERC-20 Token Standard interface. Using two complementary variants of first-order logic we have shown different ways to express sums, and to maintain invariants concerning them.

The first variant is Sum Logic (SL), a syntactic and semantic extension to Presburger arithmetic, that adds uninterpreted "balance" functions, and associated "sum" constants. We have proven that for fragments of SL that hold the *small model property* (as defined in Section 2.2), it is possible to reduce formulas of SL back into pure Presburger arithmetic, and by that we have outlined a decision procedure for SL.

We have proven that for a simple fragment of SL this *small model property* holds, and the structure of the proof encourages us to believe that other fragments might hold this property as well.

However, we have shown that for fragments of SL with at least 3 uninterpreted functions — or any extension to Presburger arithmetic with 2 functions and a size operation — we are able to encode the halting problem of a 2-counter machine. Since this problem is undecidable, there can be no decision procedure for systematically checking the satisfiability of formulas in this fragment. This gives us an upper-bound on the decidability of SL fragments.

The second variant we discussed is Explicit Coins for Implicit Summations (ECIS), a theory of two-sorted, first-order logic, which encodes balances as a relation between owners (of sort *Address*) to tokens (of sort *Coin*). We have described axioms that ensure the integrity of the state, and preserve the relation between sums and balances.

We have shown that even when we only consider EPR formulas in ECIS, some non-trivial expressiveness is maintained, and at the very least, we are able to encode the safety properties, as well as a discretized version of the transitions in the ERC-20 Token Standard.

But we reached a similar undecidability result in ECIS, when we extended the logic enough to express the balances as integers, and not just as an opaque relation between two (non-numeric) sorts. Since the axioms indeed maintain the higher-order state, we are in fact no less expressive than SL, which is undecidable in some scenarios, as we have proven.

In summary, we have seen that some programs can be verified using either of these approaches, but for others the resulting formulas would not be amenable to existing automated reasoning tools, and may well be undecidable in any logical encoding.

4.1 Related Work

We do not aim to provide a comprehensible summary of all results relating to first-order reasoning about sums (and aggregates in general) or formal verification of smart

contracts, but we focus on selected works that are closely related to the themes explored in this thesis.

Most closely related is [11], a recent master's thesis that touches on many of the same subjects as this thesis, but focuses more on the relation between the higher-order and the first-order setting, and does not deal with semantic extension such as we have in SL.

A recent work in using formal methods to verify smart contracts is ZEUS [12], which uses symbolic model checking to analyze contracts written in Solidity (with the ERC-20 Token Standard interface). The approach there aims to ensure generic properties, not necessarily related to the semantics of the program. This differs from this work, which specifically aims to verify safety properties relating to the semantics of balances and sums in ERC-20.

A similar but different approach is manually proving safety properties of such programs. This was done in [13], using Isabella [14]. This is in contrast to this work, which uses automatic techniques.

Another prominent work on combining aggregates with Presburger arithmeic was BAPA [15], which adds sets to first-order logic and devises a decision procedure for it. However, it does not handle uninterpreted functions directly, and requires the removal of them, which is impossible in the cases discussed here, since the semantics of the program are expressed by the "balance" functions.

Further research into BAPA was done in [16], which provides decision procedures for two-variable logics with counting and quantifier-free multisets with cardinality constraints (among others). Both of these are done via a reduction to BAPA, but does not cover quite the same use cases as we did in this thesis. Specifically sums and richer logics than two-variable.

4.2 Future Research

There are still various interesting directions for future research. It is clear that there exists a wide gap between our decidability and undecidability results. In particular, the decidability of a fragment of SL with 2 sets of uninterpreted functions and sums is important, since this is usually how we encode transition systems.

Since the most expressive fragment of SL with dual balances may be too expressive for decidability, the challenge for future work in this direction might be finding a fragment that is somewhat restricted, decidable, but surprisingly apt for encoding real-world programs.

Another idea is extending the sum constants to a general-purpose summation operation. The reduction to Presburger suggests that this might be possible in some cases, as long as the *small model property* is preserved. This gives us a much more powerful framework, that can express complex transitions and verification conditions, even when limited in other ways. E.g. a single "balance" function that is summed over with two different expressions that represent the state before and after a transition.

Finally, we would also like to tighten our undecidability result, and see what is the bare minimum required to encode the halting problem of a 2-counter machine — or in general, any undecidable problem — in a variant of first-order logic with sums. We believe it is possible to shave off some expressiveness and still be able to encode problems which are undecidable in nature.

References

[1] C. A. R. Hoare. An axiomatic basis for computer programming. *Commun. ACM*, 12(10):576–580, 1969.

[2] Leonid Libkin. Logics with counting. In *Elements of Finite Model Theory*, pages 141–161. Springer, 2004.

[3] Jouko A. Väänänen. Generalized quantifiers. *Bull. EATCS*, 62, 1997.

[4] Kousha Etessami. Counting quantifiers, successor relations, and logarithmic space. *J. Comput. Syst. Sci.*, 54(3):400–411, 1997.

[5] Fabian Vogelsteller and Vitalik Buterin. EIP-20: ERC-20 token standard. In *Ethereum Improvement Proposals*, no. 20. 2015. Available: https://eips.ethereum.org/EIPS/eip-20.

[6] Frank P. Ramsey. On a problem in formal logic. In *Proceedings of the London Mathematical Society*, volume 30, pages 264–286, 1930.

[7] Peter G Hinman. *Fundamentals of mathematical logic*. CRC Press, 2005.

[8] Mojżesz Presburger. Über die vollständigkeit eines gewissen systems der arithmetik ganzer zahlen, in welchem die addition als einzige operation hervortritt. *Comptes Rendus du I congrès de Mathématiciens des Pays Slaves*, pages 92–101, 1929.

[9] Marvin Minsky. *Computation: Finite and Infinite Machines*, pages 255–258. Prentice Hall, 1967.

[10] Oded Padon, Giuliano Losa, Mooly Sagiv, and Sharon Shoham. Paxos made EPR: Decidable reasoning about distributed protocols. 1(OOPSLA), 2017.

[11] Sophie Rain. First-order reasoning with aggregates. Master's thesis, TU Wien, 2020.

[12] Sukrit Kalra, Seep Goel, Mohan Dhawan, and Subodh Sharma. ZEUS: analyzing safety of smart contracts. In *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018*. The Internet Society, 2018.

[13] Yoichi Hirai. Defining the ethereum virtual machine for interactive theorem provers. In Michael Brenner, Kurt Rohloff, Joseph Bonneau, Andrew Miller, Peter Y.A. Ryan, Vanessa Teague, Andrea Bracciali, Massimiliano Sala, Federico Pintore, and Markus Jakobsson, editors, *Financial Cryptography and Data Security*, pages 520–535, Cham, 2017. Springer International Publishing.

[14] Tobias Nipkow. Interactive proof: Introduction to isabelle/hol. In Tobias Nipkow, Orna Grumberg, and Benedikt Hauptmann, editors, *Software Safety and Security - Tools for Analysis and Verification*, volume 33 of *NATO Science for Peace and*

Security Series - D: Information and Communication Security, pages 254–285. IOS Press, 2012.

[15] Viktor Kuncak, Huu Hai Nguyen, and Martin C. Rinard. An algorithm for deciding BAPA: boolean algebra with presburger arithmetic. In Robert Nieuwenhuis, editor, *Automated Deduction - CADE-20, 20th International Conference on Automated Deduction, Tallinn, Estonia, July 22-27, 2005, Proceedings*, volume 3632 of *Lecture Notes in Computer Science*, pages 260–277. Springer, 2005.

[16] Ruzica Piskac. *Decision Procedures for Program Synthesis and Verification*. PhD thesis, IIF, Lausanne, 2011.