Representation of Peano Arithmetic in Separation Logic

Sohei Ito
Nagasaki University, Nagasaki, Japan

Makoto Tatsuta[1]
National Institute of Informatics / Sokendai, Tokyo, Japan

Abstract

Separation logic is successful for software verification of heap-manipulating programs. Numbers are necessary to be added to separation logic for verification of practical software where numbers are important. However, properties of the validity such as decidability and complexity for separation logic with numbers have not been fully studied yet. This paper presents the translation of Pi-0-1 formulas in Peano arithmetic to formulas in a small fragment of separation logic with numbers, which consists only of the intuitionistic points-to predicate, 0 and the successor function. Then this paper proves that a formula in Peano arithmetic is valid in the standard model if and only if its translation in this fragment is valid in the standard interpretation. As a corollary, this paper also gives a perspective proof for the undecidability of the validity in this fragment. Since Pi-0-1 formulas can describe consistency of logical systems and non-termination of computations, this result also shows that these properties discussed in Peano arithmetic can also be discussed in such a small fragment of separation logic with numbers.

2012 ACM Subject Classification Theory of computation → Separation logic

Keywords and phrases First order logic, Separation logic, Peano arithmetic, Presburger arithmetic

Digital Object Identifier 10.4230/LIPIcs.FSCD.2024.18

Funding *Sohei Ito*: MEXT/JSPS KAKENHI Grant Number JP21K11756, ROIS NII Open Collaborative Research 2023(22FP03)
Makoto Tatsuta: ROIS NII Open Collaborative Research 2023(22FP03)

1 Introduction

Separation logic is successful both theoretically and practically [5, 12] and has been actively studied. It enables us to verify heap-manipulating programs by concise description of memory. Separation logic itself is also interesting theoretically.

When one uses separation logic for verifying software where numbers are important, one often has to extend separation logic by arithmetic. In order for handling numbers, we have to add Peano arithmetic or Presburger arithmetic to separation logic. Then the decidability of the validity in the system of separation logic with arithmetic under the standard interpretation of numbers becomes a question, since the logical system with decidable validity is more appropriate for software verification systems. In particular, since Presburger arithmetic is decidable for the validity, we might expect that some fragment of Presburger arithmetic with some decidable fragment of separation logic is decidable for the validity. In this paper, we will show that it is not the case by proving that the separation logic with only $\hookrightarrow$, 0 and the successor s without any other separation logic constructs or any arithmetical operations can simulate Π^0_1 formulas of Peano arithmetic, and consequently it is undecidable for the validity in the standard interpretation. It was surprising for us that such a weak fragment

[1] Corresponding author

of separation logic becomes such a strong logic even if we only add 0 and s, because the fragment of separation logic with only $\hookrightarrow$ is known to be decidable for the validity in the standard interpretation [4].

Our main theorem in this paper is a representation theorem which states that there is a translation of Π^0_1 formulas in Peano arithmetic to formulas in the separation logic with only the intuitionistic points-to predicate and numbers such that a formula in Peano arithmetic is valid in the standard model if and only if its translation in the separation logic with numbers is valid in the standard interpretation. Namely, the translation preserves the validity and the non-validity. The undecidability result is obtained as a corollary of this theorem.

The main technique of the proof of our representation theorem is to have an operation table for addition, multiplication, and inequality in a heap, and to remove $x + y = z$, $x \times y = z$ and $x \leq y$ by referring to the value z or the truth of $x \leq y$. For this, we will code the operation table by the consecutive heap cells that contain $0, x + 3, y + 3, x + y + 3$ or $1, x + 3, y + 3, x \times y + 3$ or $2, x + 3, y + 3$ where 3 is an offset and $0, 1$ and 2 are tags for addition, multiplication, and inequality, respectively. In order to define the translation, we will introduce normal form of a bounded formula of Peano arithmetic.

Our translation can be extended to arbitrary formulas of Peano arithmetic, but the representation theorem does not hold for formulas beyond Π^0_1. We will give some Σ^0_1 formula as a counterexample.

Our result shows that discussion about properties described by Π^0_1 formulas such as consistency of logical systems and strong normalization properties for reduction systems in Peano arithmetic can be simulated in the separation logic with numbers. The undecidability of the validity in the separation logic with numbers itself can be proved in a simpler way, by using a similar idea to [7]. We will also give a proof in that way.

There are some undecidability results about the validity of separation logic and some of them use some translations. The separation logic with the 1-field points-to predicate and the separating implication is known to be undecidable for its validity [4]. The separation logic with the 2-field points-to predicate is also known to be undecidable for its validity [7]. It is proved by translating formulas in a first-order logic with one binary relation into the separation logic with the 2-field points-to predicate. On the other hand, there are some decidability results about the validity of separation logic. The separation logic with the 1-field points-to predicate and without the separating implication is known to be decidable for its validity [4]. The quantifier-free separation logic is known to be decidable [6]. It is proved by translating the separation logic into a first-order logic with empty signature. We do not know any work on a translation from some fragment of arithmetic into such a weak separation logic with only $\{\hookrightarrow, 0, s\}$.

If we restrict ourselves to symbolic heaps in separation logic with arithmetic or inductive definitions, there are some decidability results. Symbolic heap entailment with Presburger arithmetic [15], bounded-treewidth symbolic heap entailment [10], symbolic heap entailment with cone inductive definitions [16, 11], and symbolic heap entailment with lists [2, 3, 8, 1] are known to be decidable for their validity. Even for symbolic heaps, if we weaken conditions, they easily become undecidable. Symbolic heap entailment with unrestricted inductive definitions [10], and symbolic heap entailment with bounded-treewidth inductive definitions and implicit existentials [14] are known to be undecidable for their validity.

This paper is organized as follows: Section 2 defines Peano arithmetic. Section 3 defines the separation logic with numbers SLN. Section 4 defines the translation of normal formulas in Peano arithmetic to formulas in SLN and shows the preservation property of the translation. In Section 5 we define an auxiliary translation to normal form and prove the main theorem for the preservation of the translation for Π^0_1 formulas from PA to SLN. Section 6 gives another proof of undecidability for the validity of SLN. Section 7 concludes.

2 Peano arithmetic

In this section, we define Peano arithmetic PA and its standard model.

Let Vars $= \{x, y, \ldots\}$ be the set of variables. The *terms of* PA are defined by:

$$t ::= x \mid 0 \mid s(t) \mid t + t \mid t \times t.$$

The *formulas of* PA are defined by:

$$A ::= t = t \mid t \leq t \mid \neg A \mid A \wedge A \mid A \vee A \mid \exists x A \mid \forall x A.$$

We will write $A \to B$ for $\neg A \vee B$.

We write $s^n(t)$ for $\overbrace{s(\ldots(s}^{n}(t))\ldots)$. We use the abbreviation $\overline{n} = s^n(0)$. We write $A[x := t]$ for the formula obtained by capture-free substitution of t for x in A.

Let $\mathcal{N}$ be the *standard model* of PA, namely, its universe $|\mathcal{N}|$ is $\mathbb{N} = \{0, 1, 2, \ldots\}$, $0^{\mathcal{N}}=0$, $s^{\mathcal{N}}(x) = x + 1$, $+^{\mathcal{N}}(x, y) = x + y$, $\times^{\mathcal{N}}(x, y) = x \times y$, $(\leq)^{\mathcal{N}}(x, y)$ iff $x \leq y$. Let $\sigma : \text{Vars} \to \mathbb{N}$ be a *variable assignment*. We extend σ to terms in a usual way. We write $\sigma[x := n]$ for the variable assignment that assigns n to x and $\sigma(y)$ to y other than x.

We write $\sigma \models A$ when A is true in $\mathcal{N}$ under the variable assignment σ. This relation is defined in a usual way. If $\sigma \models A$ for every variable assignment σ, A is defined to be *valid*. If A does not contain free variables, A is called *closed*.

A formula $\forall x \leq t.A$ is an abbreviation of $\forall x(x \leq t \to A)$, where t does not contain x. A formula $\exists x \leq t.A$ is an abbreviation of $\exists x(x \leq t \wedge A)$, where t does not contain x. We call $\forall x \leq t$ and $\exists x \leq t$ *bounded quantifiers*. A formula A is defined to be *bounded* if every quantifier in A is bounded. If $A \equiv \forall x B$ and B is bounded, A is called a Π^0_1 *formula*.

3 Separation logic with numbers

In this section, we define a small fragment SLN of separation logic with numbers. We will also define the standard interpretation of SLN.

Let Vars $= \{x, y, \ldots\}$ be the set of variables. The *terms of* SLN are defined by:

$$t ::= x \mid 0 \mid s(t).$$

The *formulas of* SLN are defined by:

$$A ::= t = t \mid t \hookrightarrow t \mid \neg A \mid A \wedge A \mid A \vee A \mid \exists x A \mid \forall x A.$$

We will write $A \to B$ for $\neg A \vee B$.

The predicate $t_1 \hookrightarrow t_2$ is the intuitionistic points-to predicate and means that there is some cell of address t_1 which contains t_2 in the heap.

We use the same abbreviation $\overline{n}$ and substitution $A[x := t]$ as in PA. For simplicity, we write $(t \hookrightarrow t_1, \ldots, t_n)$ for $t \hookrightarrow t_1 \wedge \ldots \wedge s^{n-1}(t) \hookrightarrow t_n$. We sometimes write only one quantifier for consecutive quantifiers in a usual way like $\forall xy \exists zw$ for $\forall x \forall y \exists z \exists w$.

Now we define the *standard interpretation* $[\![\cdot]\!]$ of SLN. Let $\mathbb{N}$ be $\{0, 1, \ldots\}$. We use $\mathbb{N}$ for both the sets of addresses and values. Let $[\![0]\!] = 0, [\![s]\!](x) = x + 1$. Let $\sigma : \text{Vars} \to \mathbb{N}$ be a *variable assignment*. The extension of σ to terms and the variable assignment $\sigma[x := n]$ are defined similarly to those in PA. A heap is a finite function $h : \mathbb{N} \to_{\text{fin}} \mathbb{N}$. A heap represents a state of the memory.

For a formula A of SLN, we define $\sigma, h \models A$ by:

$$\begin{array}{lll}
\sigma, h \models t_1 = t_2 & \text{iff} & \sigma(t_1) = \sigma(t_2), \\
\sigma, h \models t_1 \hookrightarrow t_2 & \text{iff} & h(\sigma(t_1)) = \sigma(t_2), \\
\sigma, h \models \neg A & \text{iff} & \sigma, h \not\models A, \\
\sigma, h \models A_1 \wedge A_2 & \text{iff} & \sigma, h \models A_1 \text{ and } \sigma, h \models A_2, \\
\sigma, h \models A_1 \vee A_2 & \text{iff} & \sigma, h \models A_1 \text{ or } \sigma, h \models A_2, \\
\sigma, h \models \exists x A & \text{iff} & \text{for some } n \in \mathbb{N}, \sigma[x := n], h \models A, \\
\sigma, h \models \forall x A & \text{iff} & \text{for all } n \in \mathbb{N}, \sigma[x := n], h \models A.
\end{array}$$

$\sigma, h \models A$ means that A is true under the variable assignment σ and the heap h. A formula A is defined to be *valid* if $\sigma, h \models A$ for all σ and h. If a formula does not contain atoms $t \hookrightarrow u$, the truth of the formula does not depend on heaps.

The validity defined in this section is the validity in the standard interpretation of SLN, and it is different from the ordinary validity for separation logic, since the interpretations depend on the set of addresses in the definition of the ordinary validity.

4 Translation of Normal Formulas in PA into SLN

In this section, we define the translation $(\cdot)^\circ$ of bounded formulas in PA to formulas in SLN, and prove that the translation preserves the validity and the non-validity.

The key of the translation is to keep an operation table for addition, multiplication and inequality in a heap, and a resulting formula in SLN refers to the table instead of using the addition, multiplication and inequality symbols. To state that a heap keeps the operation table, we will use a table heap condition. For proving the preservation of the translation, we will use a simple table heap, which is a heap that contains all the operation entries of some size. Since the table in a heap is finite, to estimate the necessary size of the operation table for translating a given formula, we will use the upper bound of arguments in the formula.

We will first define normal form of a bounded formula in PA, which we will translate into a formula in SLN. Next we will define a table heap condition, which guarantees that a heap has an operation table for addition, multiplication and inequality. Then we will define the translation of a normal formula in PA into a formula in SLN. Then we will define a simple table heap and the upper bound of arguments in a formula. Finally we will prove the preservation of the translation.

We write $\exists(x = t)A$ for an abbreviation of $\exists x(x = t \wedge A)$, where t does not contain x.

Our translation is defined for only normal formulas. This does not lose the generality since any bounded formula can be transformed into a normal formula in Section 5. In a normal formula, $+$ and $\times$ appear only in t of $\exists(x = t)$. Moreover, this t is of the form $a + b$ or $a \times b$ where a, b do not contain $+$ or $\times$.

▶ **Definition 4.1** (Normal form). *Normal forms* of PA are given by A in the following grammar:

$$A ::= B \mid \forall x \leq t.A \mid \exists x \leq t.A \mid \exists(x = t)A$$

satisfying the following conditions: (1) B is a disjunctive normal form of a formula in PA without quantifiers, $+$, $\times$, and formulas of the form $\neg(t \leq u)$, (2) each t in $\forall x \leq t$ and $\exists x \leq t$ does not contain $+$, $\times$, and (3) each t in $\exists(x = t)$ is of the form $a + b$ or $a \times b$ for some terms a and b that do not contain $+$ or $\times$.

The table heap condition is defined as the formula H in the next definition. It guarantees that a heap that satisfies H contains a correct operation table for $+$, $\times$ and $\leq$. The formulas Add, Mult and Ineq in the following definition refer to the operation table when a heap satisfies

the table heap condition. The normal formula enables us to represent each occurrence of $+$, $\times$ and $\leq$ by Add, Mult and Ineq, respectively. We will write $[t]$ for $s^3(t)$ for readability, since the offset is 3.

▶ **Definition 4.2** (Table Heap Condition). H, $\mathsf{Add}(x,y,z)$, $\mathsf{Mult}(x,y,z)$ and $\mathsf{Ineq}(x,y)$ are the formulas defined by:

$$
\begin{aligned}
H_{\mathsf{Add1}} &\equiv \forall ay((a \hookrightarrow 0,[0],[y]) \to s^3(a) \hookrightarrow [y]),\\
H_{\mathsf{Add2}} &\equiv \forall axy((a \hookrightarrow 0,[s(x)],[y])\\
&\qquad \to \exists bz((b \hookrightarrow 0,[x],[y],[z]) \wedge s^3(a) \hookrightarrow [s(z)])),\\
H_{\mathsf{Mult1}} &\equiv \forall ay((a \hookrightarrow \overline{1},[0],[y]) \to s^3(a) \hookrightarrow [0]),\\
H_{\mathsf{Mult2}} &\equiv \forall axy((a \hookrightarrow \overline{1},[s(x)],[y]) \to \exists bz((b \hookrightarrow \overline{1},[x],[y],[z]) \wedge\\
&\qquad \exists cw((c \hookrightarrow 0,[z],[y],[w]) \wedge s^3(a) \hookrightarrow [w]))),\\
H_{\mathsf{Ineq1}} &\equiv \forall axy((a \hookrightarrow \overline{2},[s(x)],[y]) \to \exists zb(y = s(z) \wedge (b \hookrightarrow \overline{2},[x],[z]))),\\
H_{\mathsf{Ineq2}} &\equiv \forall axy((a \hookrightarrow \overline{2},[s(x)],[y]) \to \exists b(b \hookrightarrow \overline{2},[x],[y])),\\
H &\equiv H_{\mathsf{Add1}} \wedge H_{\mathsf{Add2}} \wedge H_{\mathsf{Mult1}} \wedge H_{\mathsf{Mult2}} \wedge H_{\mathsf{Ineq1}} \wedge H_{\mathsf{Ineq2}},\\
\mathsf{Add}(x,y,z) &\equiv \forall a((a \hookrightarrow 0,[x],[y]) \to s^3(a) \hookrightarrow [z]),\\
\mathsf{Mult}(x,y,z) &\equiv \forall a((a \hookrightarrow \overline{1},[x],[y]) \to s^3(a) \hookrightarrow [z]),\\
\mathsf{Ineq}(x,y) &\equiv \exists a(a \hookrightarrow \overline{2},[x],[y]).
\end{aligned}
$$

The formula H forces a heap to have a table that contains results of addition, multiplication and inequality for some natural numbers. Each entry for addition and multiplication consists of four cells, and each entry for inequality consists of three cells. If the first cell contains 0, then the entry is for addition. If the first cell contains 1, then the entry is for multiplication. If the first cell contains 2, then the entry is for inequality. The second and third cells of an entry represent arguments of addition, multiplication or inequality. The entries for $+, \times$ have the forth cells, which contain the results of addition or multiplication. For inequality, if there is an entry for two arguments x and y, then $x \leq y$ holds. Since 0, 1 and 2 have a special meaning, arguments and results are stored by adding three. The definition of H uses the following inductive definitions of addition and multiplication: $s(x) + y = s(x + y)$ and $(x + 1) \times y = x \times y + x$. The formulas H_{Add1} and H_{Mult1} force the base cases of addition and multiplication, respectively, and H_{Add2} and H_{Mult2} force the induction steps of addition and multiplication, respectively. H_{Ineq1} means that if there is an entry for $x + 1 \leq y$, then the entry for $x \leq y - 1$ exists in the heap. H_{Ineq2} means that if there is an entry for $x + 1 \leq y$, then the entry for $x \leq y$ exists in the heap.

We will show that the formula H actually forces the heap to have a correct table for addition, multiplication and inequality (the claims (1), (2) and (3) below). The claim (4) below says that H ensures that if a heap contains an entry for $u \leq u$, then it contains all the entries for $t \leq u$.

▶ **Lemma 4.3.** *Let σ be a variable assignment and h be a heap.*

(1) *If $\sigma, h \models H$, $h(m) = 0$, $h(m+1) = n+3$ and $h(m+2) = k+3$, then $h(m+3) = n+k+3$.*

(2) *If $\sigma, h \models H$, $h(m) = 1$, $h(m+1) = n+3$ and $h(m+2) = k+3$, then $h(m+3) = n \times k+3$.*

(3) *If $\sigma, h \models H$, $h(m) = 2$, $h(m+1) = n+3$, $h(m+2) = k+3$, then $n \leq k$.*

(4) *If $\sigma, h \models H$, $\sigma(t) \leq \sigma(u)$, $\sigma, h \models \mathsf{Ineq}(u,u)$, then $\sigma, h \models \mathsf{Ineq}(t,u)$.*

Proof.

(1) We will show the claim by induction on n.
(Base case) Let $n = 0$. Since $h(m) = 0$, $h(m+1) = 3$, $h(m+2) = k+3$ and $\sigma, h \models H_{\mathsf{Add1}}$, we have $\sigma[a := m, x := n, y := k], h \models s^3(a) \hookrightarrow [y]$. Hence, we have $h(m+3) = \sigma[a := m, x := n, y := k]([y]) = k+3 = n+k+3$.
(Induction step) Let $n > 0$. Then, $n-1 \geq 0$. Let $\sigma' = \sigma[a := m, x := n-1, y := k]$. Since $h(m) = 0$, $h(m+1) = n+3$, $h(m+2) = k+3$ and $\sigma, h \models H_{\mathsf{Add2}}$, we have $\sigma', h \models \exists bz((b \hookrightarrow 0, [x], [y], [z]) \wedge s^3(a) \hookrightarrow [s(z)])$. There exist q and ℓ such that $\sigma'[b := q, z := \ell], h \models (b \hookrightarrow 0, [x], [y], [z]) \wedge s^3(a) \hookrightarrow [s(z)]$. That is, $h(q) = 0$, $h(q+1) = (n-1)+3$, $h(q+2) = k+3$, $h(q+3) = \ell+3$ and $h(m+3) = \ell+4$. By induction hypothesis, we have $h(q+3) = (n-1)+k+3 = n+k+2$. That is, $\ell = n+k-1$. Thus, we have $h(m+3) = \ell+4 = n+k-1+4 = n+k+3$.

(2) We will show the claim by induction on n.
(Base case) Let $n = 0$. Since $h(m) = 1$, $h(m+1) = 3$, $h(m+2) = k+3$ and $\sigma, h \models H_{\mathsf{Mult1}}$, we have $\sigma[a := m, x := n, y := k], h \models s^3(a) \hookrightarrow [0]$. Hence, we have $h(m+3) = \sigma[a := m, x := n, y := k]([0]) = 3 = n \times k + 3$.
(Induction step) Let $n > 0$. Then, $n-1 \geq 0$. Let $\sigma' = \sigma[a := m, x := n-1, y := k]$. Since $h(m) = 1$, $h(m+1) = n+3$, $h(m+2) = k+3$ and $\sigma, h \models H_{\mathsf{Mult2}}$, we have $\sigma', h \models \exists bz((b \hookrightarrow \overline{1}, [x], [y], [z]) \wedge \exists cw((c \hookrightarrow 0, [z], [y], [w]) \wedge s^3(a) \hookrightarrow [w]))$. There exist q and ℓ such that $\sigma'[b := q, z := \ell], h \models (b \hookrightarrow \overline{1}, [x], [y], [z]) \wedge \exists cw((c \hookrightarrow 0, [z], [y], [w]) \wedge s^3(a) \hookrightarrow [w])$. That is, $h(q) = 1, h(q+1) = (n-1)+3, h(q+2) = k+3, h(q+3) = \ell+3$. So by induction hypothesis, $h(q+3) = (n-1) \times k + 3$. Thus $\ell = (n-1) \times k$. Furthermore, since $\sigma'[b := q, z := \ell], h \models \exists cw((c \hookrightarrow 0, [z], [y], [w]) \wedge s^3(a) \hookrightarrow [w])$, we have $\sigma'[b := q, z := \ell, c := r, w := p], h \models (c \hookrightarrow 0, [z], [y], [w]) \wedge s^3(a) \hookrightarrow [w]$ for some r and p. That is, $h(r) = 0, h(r+1) = \ell+3, h(r+2) = k+3, h(r+3) = p+3$ and $h(m+3) = p+3$. By (1) of this Lemma, we have $p = \ell + k$. With this and $\ell = (n-1) \times k$, we have $p = (n-1) \times k + k = n \times k$. Hence, $h(m+3) = p+3 = n \times k + 3$.

(3) We will show the claim by induction on n.
(Base case) Let $n = 0$. We immediately have $n \leq k$.
(Induction step) Let $n > 0$. Then, $n-1 \geq 0$. Let $\sigma' = \sigma[a := m, x := n-1, y := k]$. Since $h(m) = 2$, $h(m+1) = n+3$, $h(m+2) = k+3$ and $\sigma, h \models H_{\mathsf{Ineq1}}$, we have $\sigma', h \models \exists zb(y = s(z) \wedge (b \hookrightarrow \overline{2}, [x], [z]))$. Thus, there exist ℓ and q such that $\sigma'[z := \ell, b := p], h \models y = s(z) \wedge (b \hookrightarrow \overline{2}, [x], [z])$, that is, $h(p) = 2$, $h(p+1) = (n-1)+3$, $h(p+2) = \ell+3 = (k-1)+3$. By induction hypothesis, $n-1 \leq k-1$, that is, $n \leq k$.

(4) We will show the claim by induction on $\sigma(u) - \sigma(t)$.
(Base case) Let $\sigma(u) - \sigma(t) = 0$, i.e. $\sigma(t) = \sigma(u)$. By assumption, we have $\sigma, h \models \mathsf{Ineq}(u, u)$. Since $\sigma(t) = \sigma(u)$, we have the claim.
(Induction step) Let $\sigma(u) - \sigma(t) > 0$, i.e. $\sigma(t) < \sigma(u)$. Since $\sigma(u) - (\sigma(t)+1) < \sigma(u) - \sigma(t)$, we have $\sigma, h \models \mathsf{Ineq}(s(t), u)$ by induction hypothesis. That is, $\sigma, h \models \exists a(a \hookrightarrow \overline{2}, [s(t)], [u])$. Since $\sigma, h \models H_{\mathsf{Ineq2}}$, we have $\sigma, h \models \exists b(b \hookrightarrow \overline{2}, [t], [u])$, that is, $\sigma, h \models \mathsf{Ineq}(t, u)$. ◀

Now we define the translation of normal formulas in PA into formulas in SLN. In the translation, $+$, $\times$ and $\leq$ are replaced by Add, Mult and Ineq with the table heap condition.

▶ **Definition 4.4** (Translation $(\cdot)^\circ$)**.** Let A be a normal formula in PA. We define SLN formula $(\forall x A)^\circ$ as:

$$\begin{aligned}
B^\circ &\equiv B^{\leq} \text{ if } B \text{ is quantifier-free,}\\
(\exists x \leq t.B)^\circ &\equiv H \to \neg\mathsf{Ineq}(t,t) \vee \exists x(\mathsf{Ineq}(x,t) \wedge B^\circ),\\
(\forall x \leq t.B)^\circ &\equiv H \to \forall x(\neg\mathsf{Ineq}(x,t) \vee B^\circ),\\
(\exists (x = t+u)B)^\circ &\equiv H \to \exists x(\mathsf{Add}(t,u,x) \wedge B^\circ),\\
(\exists (x = t\times u)B)^\circ &\equiv H \to \exists x(\mathsf{Mult}(t,u,x) \wedge B^\circ),\\
(\forall x A)^\circ &\equiv \forall x A^\circ,
\end{aligned}$$

where $B^{\leq}$ is obtained from B by replacing each positive occurrence of $t \leq u$ by $H \to \neg\mathsf{Ineq}(u,t) \vee t = u$.

Example. For a normal formula

$$\begin{aligned}
A \equiv &\exists(x_1 = x + s(x))\exists(x_2 = x + x_1)\forall y \leq x_2.\\
&\exists(x_3 = x + y)\exists(x_4 = y \times x_3)\exists(x_5 = x + x_4)(0 \leq x_5),
\end{aligned}$$

its translation A° is

$$\begin{aligned}
A^\circ \equiv &H \to \exists x_1(\mathsf{Add}(x, s(x), x_1) \wedge (H \to \exists x_2(\mathsf{Add}(x, x_1, x_2) \wedge (H \to \forall y(\neg\mathsf{Ineq}(y, x_2) \vee\\
&(H \to \exists x_3(\mathsf{Add}(x, y, x_3) \wedge (H \to \exists x_4(\mathsf{Mult}(y, x_3, x_4) \wedge\\
&(H \to \exists x_5(\mathsf{Add}(x, x_4, x_5) \wedge (H \to \neg\mathsf{Ineq}(x_5, 0) \vee 0 = x_5)))))))))))).
\end{aligned}$$

Our goal is to show that for any Π^0_1 formula A of PA, A is valid in PA if and only if A° is valid in SLN. Therefore, A° should hold for *every* heap h. By the definition of $(\cdot)^\circ$, $x = t+u$ and $x = t\times u$ are translated into $H \to \mathsf{Add}(t,u,x)$ and $H \to \mathsf{Mult}(t,u,x)$, respectively. Thus, if a heap h does not have a sufficiently large table for $x = t+u$ and $x = t\times u$, the translated formulas are trivially true. Since we demand that A° hold for all heaps, there is h that contains a sufficiently large table. Furthermore, if the addition and multiplication in the formula are correct in such a sufficiently large heap, they must be correct in every heap, because addition and multiplication are numeric properties and do not depend on heaps. The same is true for inequality. This is the key idea to prove our goal. That is, $\sigma \models A$ if and only if $\sigma, h \models A^\circ$ for *sufficiently large* h if and only if $\sigma, h \models A^\circ$ for *all* h. We will prove them in Lemmas 4.10 and 4.11 later.

Since we demand that A° hold for all heaps, we define the translation of $t \leq u$ to be $H \to \neg\mathsf{Ineq}(u,t) \vee t = u$ and we do not straightforwardly define it to be $H \to \mathsf{Ineq}(t,u)$, because $\mathsf{Ineq}(t,u)$ demands the heap to contain the entry for $t \leq u$, which is not possible if the heap is not sufficiently large. Furthermore, the translation of $\exists x \leq t.B$ is not simply $H \to \exists x(\mathsf{Ineq}(x,t) \wedge B)$ but rather seemingly tricky $H \to \neg\mathsf{Ineq}(t,t) \vee \exists x(\mathsf{Ineq}(x,t) \wedge B^\circ)$. If we adopt the simple translation, we may not be able to find x such that the entry for $x \leq t$ is in the heap when it is not sufficiently large. Our idea is to let such a case be true. Therefore, we allow the case $\neg\mathsf{Ineq}(t,t)$, which is true if the heap may not contain some entries for $\cdot \leq t$.

First, we estimate the necessary size of the operation table for a given formula and a given variable assignment. This size is defined in the next definition.

▶ **Definition 4.5.** Let A be prenex and disjunctive normal form of a bounded formula in PA and σ be a variable assignment. We define the number $\max(\sigma, A)$ by:

$$\begin{aligned}
&\max(\sigma, t \le u) = \max\{\sigma(t), \sigma(u)\},\\
&\max(\sigma, t = u) = \{0\},\\
&\max(\sigma, \neg B) = \max(\sigma, B)\\
&\max(\sigma, B \wedge C) = \max\{\max(\sigma, B), \max(\sigma, C)\}\\
&\max(\sigma, B \vee C) = \max\{\max(\sigma, B), \max(\sigma, C)\}\\
&\max(\sigma, \forall x \le t.B) = \max\{\sigma(t), \max(\sigma, B[x := t])\},\\
&\max(\sigma, \exists x \le t.B) = \max\{\sigma(t), \max(\sigma, B[x := t])\},\\
&\max(\sigma, \exists (x = t)B) = \max\{\sigma(t), \max(\sigma, B[x := t])\}.
\end{aligned}$$

Example. By using the above definition one by one, we have

$$\begin{aligned}
&\max(\sigma, \exists(x_1 = x + s(x))\exists(x_2 = x + x_1)\forall y \le x_2.\\
&\quad \exists(x_3 = x + y)\exists(x_4 = y \times x_3)\exists(x_5 = x + x_4)(0 \le x_5))\\
&= \max\{\sigma(x + s(x)), \max(\sigma, \exists(x_2 = x + (x + s(x)))\forall y \le x_2.\\
&\quad \exists(x_3 = x + y)\exists(x_4 = y \times x_3)\exists(x_5 = x + x_4)(0 \le x_5))\}\\
&= \max\{\sigma(x + s(x)), \max\{\sigma(x + (x + s(x))), \max(\sigma, \forall y \le x + (x + s(x)).\\
&\quad \exists(x_3 = x + y)\exists(x_4 = y \times x_3)\exists(x_5 = x + x_4)(0 \le x_5))\}\}\\
&= \max\{\sigma(x + s(x)), \max\{\sigma(x + (x + s(x))), \max\{\sigma(x + (x + s(x))),\\
&\quad \max(\sigma, \exists(x_3 = x + (x + (x + s(x))))\exists(x_4 = (x + (x + s(x))) \times x_3)\\
&\quad \exists(x_5 = x + x_4)(0 \le x_5))\}\}\}\\
&= \max\{\sigma(x + s(x)), \max\{\sigma(x + (x + s(x))), \max\{\sigma(x + (x + s(x))),\\
&\quad \max\{\sigma(x + (x + (x + s(x)))),\\
&\quad \max(\sigma, \exists(x_4 = (x + (x + s(x))) \times (x + (x + (x + s(x)))))\exists(x_5 = x + x_4)(0 \le x_5))\}\}\}\}\\
&= \max\{\sigma(x + s(x)), \max\{\sigma(x + (x + s(x))), \max\{\sigma(x + (x + s(x))),\\
&\quad \max\{\sigma(x + (x + (x + s(x)))),\\
&\quad \max\{\sigma((x + (x + s(x))) \times (x + (x + (x + s(x))))),\\
&\quad \max(\sigma, \exists(x_5 = x + (x + (x + s(x))) \times (x + (x + (x + s(x)))))(0 \le x_5))\}\}\}\}\}\\
&= \max\{\sigma(x + s(x)), \max\{\sigma(x + (x + s(x))), \max\{\sigma(x + (x + s(x))),\\
&\quad \max\{\sigma(x + (x + (x + s(x)))),\\
&\quad \max\{\sigma((x + (x + s(x)) \times (x + (x + (x + s(x))))),\\
&\quad \max\{\sigma(x + (x + (x + s(x))) \times (x + (x + (x + s(x))))),\\
&\quad \max(\sigma, 0 \le x + (x + (x + s(x))) \times (x + (x + (x + s(x)))))\}\}\}\}\}\\
&= \max\{\sigma(x + s(x)), \max\{\sigma(x + (x + s(x))), \max\{\sigma(x + (x + s(x))),\\
&\quad \max\{\sigma(x + (x + (x + s(x)))),\\
&\quad \max\{\sigma((x + s(x)) \times (x + (x + (x + s(x))))),\\
&\quad \max\{\sigma(x + (x + s(x)) \times (x + (x + (x + s(x))))),\\
&\quad \max\{0, \sigma(x + (x + (x + s(x))) \times (x + (x + (x + s(x)))))\}\}\}\}\}\}\\
&= \sigma(x) + (3\sigma(x) + 1)(4\sigma(x) + 1).
\end{aligned}$$

Next, for a given size n we will define a heap that covers addition of arguments $\le n^2$ and multiplication and inequality of arguments $\le n$. We call it a *simple table heap*.

▶ **Definition 4.6.** For a number n, we define a heap h_n as the heap defined by:

$$h_n(x) = \begin{cases} 0 & (x = 4i, i < (n^2+1)^2) \\ i \bmod (n^2+1) + 3 & (x = 4i+1, i < (n^2+1)^2) \\ \lfloor i/(n^2+1) \rfloor + 3 & (x = 4i+2, i < (n^2+1)^2) \\ h_n(x-2) + h_n(x-1) - 3 & (x = 4i+3, i < (n^2+1)^2) \\ 1 & (x = c_1 + 4i, i < (n+1)^2) \\ i \bmod (n+1) + 3 & (x = c_1 + 4i + 1, i < (n+1)^2) \\ \lfloor i/(n+1) \rfloor + 3 & (x = c_1 + 4i + 2, i < (n+1)^2) \\ (h_n(x-2) - 3) \times (h_n(x-1) - 3) + 3 & (x = c_1 + 4i + 3, i < (n+1)^2) \\ 2 & (x = c_2 + 3i, i < (n+1)^2) \\ i \bmod (n+1) + 3 & (x = c_2 + 3i + 1, i < (n+1)^2) \\ n + 3 & (x = c_2 + 3i + 2, i < (n+1)^2, \\ & \quad \lfloor i/(n+1) \rfloor < i \bmod (n+1)) \\ \lfloor i/(n+1) \rfloor + 3 & (x = c_2 + 3i + 2, i < (n+1)^2, \\ & \quad \lfloor i/(n+1) \rfloor \geq i \bmod (n+1)) \\ \text{undefined} & \text{otherwise} \end{cases}$$

where $c_1 = 4(n^2+1)^2$ and $c_2 = c_1 + 4(n+1)^2$.

h_n has the operation table that has entries of $+$ for arguments $\leq n^2$ and the entries of $\times$ and $\leq$ for arguments $\leq n$. The i-th entry for $+$ contains the result of addition of $x = i \bmod (n^2+1)$ and $y = \lfloor i/(n^2+1) \rfloor$, that is, $h(4i) = 0$, $h(4i+1) = x+3$, $h(4i+2) = y+3$ and $h(4i+3) = x+y+3$. The i-th entry for $\times$ contains the result of multiplication of $x = i \bmod (n+1)$ and $y = \lfloor i/(n+1) \rfloor$, that is, $h(c_1 + 4i) = 0$, $h(c_1 + 4i + 1) = x + 3$, $h(c_1 + 4i + 2) = y + 3$ and $h(c_1 + 4i + 3) = x \times y + 3$. The i-th entry for $\leq$ signifies inequality of $x = i \bmod (n+1)$ and $y = \lfloor i/(n+1) \rfloor$ or $y = n$, where $h(c_2 + 4i) = 2$, $h(c_2 + 4i + 1) = x + 3$, and $h(c_2 + 4i + 2) = y + 3$ if $x \leq y$ and $h(c_2 + 4i + 2) = n + 3$ if $x > y$.

The next lemma shows that the simple table heap h_n satisfies the table heap condition H.

▶ **Lemma 4.7.** *For a variable assignment σ, we have $\sigma, h_n \models H$.*

Proof. h_n clearly satisfies H. ◀

The next lemma shows that the truth of $t + u = v$, $t \times u = v$ and $t \leq u$ in PA for the standard model is equivalent to the truth of their translations in SLN for the standard interpretation for the simple table heap.

▶ **Lemma 4.8.** *For $n \geq \max\{\sigma(t), \sigma(u)\}$, the following hold.*

(1) *$\sigma \models t + u = v$ if and only if $\sigma, h_n \models \mathsf{Add}(t, u, v)$.*

(2) *$\sigma \models t \times u = v$ if and only if $\sigma, h_n \models \mathsf{Mult}(t, u, v)$.*

(3) *$\sigma \models t \leq u$ if and only if $\sigma, h_n \models \mathsf{Ineq}(t, u)$.*

Proof.

(1) Only-if-direction: Since $n \geq \max\{\sigma(t), \sigma(u)\}$, by the definition of h_n, there exists p such that $\sigma[a := p], h_n \models (a \hookrightarrow 0, [t], [u])$. That is, $h_n(p) = 0, h_n(p+1) = \sigma(t) + 3, h_n(p+2) = \sigma(u) + 3$. By the definition of h_n, we have $h_n(p+3) = \sigma(t) + \sigma(u) + 3$. By assumption, $\sigma(t) + \sigma(u) = \sigma(v)$. Therefore, $h_n(p+3) = \sigma(v) + 3$. Thus, $\sigma[a := p], h_n \models s^3(a) \hookrightarrow [v]$. Hence, $\sigma, h_n \models \mathsf{Add}(t, u, v)$.

If-direction: Since $n \geq \max\{\sigma(t), \sigma(u)\}$, by the definition of h_n, there exists p such that $\sigma[a := p], h_n \models (a \hookrightarrow 0, [t], [u], [v])$. Thus, $h_n(p) = 0, h_n(p+1) = \sigma(t) + 3, h_n(p+2) = \sigma(u) + 3, h_n(p+3) = \sigma(v) + 3$. By the definition of h_n, we have $h_n(p+3) = (h_n(p+1) - 3) + (h_n(p+2) - 3) + 3$. Since $h_n(p+1) = \sigma(t) + 3$ and $h_n(p+2) = \sigma(u) + 3$, we have $h_n(p+3) = \sigma(t) + \sigma(u) + 3$. Thus, we have $\sigma(t) + \sigma(u) = \sigma(v)$. Hence, $\sigma \models t + u = v$.

(2) The claim can be shown similarly to (1).

(3) Only-if-direction: Suppose $\sigma(t) \leq \sigma(u)$. Let $i = \sigma(t) \cdot (n+1) + \sigma(u)$. Since $n \geq \max\{\sigma(t), \sigma(u)\}$, we have $i < (n+1)^2$. Furthermore, $\sigma(t) = \lfloor i/(n+1) \rfloor$ and $\sigma(u) = i \bmod (n+1)$. For $p = 4(n^2+1)^2 + 4(n+1)^2 + 3i$, we have $h_n(p) = 2$, $h_n(p+1) = \sigma(t) + 3$ by the definition of h_n. Since $\sigma(t) \leq \sigma(u)$, we have $h_n(p+2) = \sigma(u) + 3$ by the definition of h_n. From this, we have $\sigma, h_n \models \exists a(a \hookrightarrow \overline{2}, [t], [u])$, that is, $\sigma, h_n \models \mathsf{Ineq}(t, u)$.
If-direction: Suppose $\sigma, h_n \models \mathsf{Ineq}(t, u)$. Since $n \geq \max\{\sigma(t), \sigma(u)\}$, there exists p such that $h_n(p) = 2$, $h_n(p+1) = \sigma(t) + 3$, $h_n(p+2) = \sigma(u) + 3$. By Lemma 4.3 (3), we have $h_n(p+1) \leq h_n(p+2)$, that is, $\sigma(t) \leq \sigma(u)$. ◀

The next lemma shows that if Add, Mult and $\neg\mathsf{Ineq}$ are true for a sufficiently large simple table heap, they are also true for all heaps.

▶ **Lemma 4.9.** *For $n \geq \max\{\sigma(t), \sigma(u)\}$, the following hold.*

(1) *$\sigma, h_n \models \mathsf{Add}(t, u, v)$ if and only if $\sigma, h \models H \to \mathsf{Add}(t, u, v)$ for all h.*

(2) *$\sigma, h_n \models \mathsf{Mult}(t, u, v)$ if and only if $\sigma, h \models H \to \mathsf{Mult}(t, u, v)$ for all h.*

(3) *$\sigma, h_n \models \neg\mathsf{Ineq}(t, u)$ if and only if $\sigma, h \models H \to \neg\mathsf{Ineq}(t, u)$ for all h.*

Proof. The if-direction is obvious. We will show the only-if-direction.

(1) Since $\sigma, h_n \models \mathsf{Add}(t, u, v)$ by assumption, we have $\sigma(t) + \sigma(u) = \sigma(v)$ by Lemma 4.8 (1). We fix h in order to show $\sigma, h \models H \to \mathsf{Add}(t, u, v)$.

Case 1. If $\sigma, h \not\models H$, the claim follows trivially.

Case 2. Assume $\sigma, h \models H$.

Case 2.1 If $\sigma, h \models \forall a \neg(a \hookrightarrow 0, [t], [u])$, the claim follows trivially, because $\sigma, h \models \forall a((a \hookrightarrow 0, [t], [u]) \to s^3(a) \hookrightarrow [u])$.

Case 2.2 Assume $\sigma, h \models \exists a(a \hookrightarrow 0, [t], [u])$. We assume $h(p) = 0, h(p+1) = \sigma(t) + 3, h(p+2) = \sigma(u) + 3$ for arbitrary p. Since $\sigma, h \models H$, we have $h(p+3) = (h(p+1) - 3) + (h(p+2) - 3) + 3$ by Lemma 4.3 (1). Therefore, $h(p+3) = \sigma(t) + \sigma(u) + 3$. That is, $h(p+3) = \sigma(v) + 3$. Thus $\sigma, h \models s^3(a) \hookrightarrow [v]$. Then, we have $\sigma[a := p], h \models (a \hookrightarrow 0, [t], [u]) \to s^3(a) \hookrightarrow [v]$ for all p.

Hence in both cases $\sigma, h \models H \to \mathsf{Add}(t, u, v)$.

(2) The claim can be shown similarly to (1) (except it uses Lemma 4.3 (2)).

(3) By Lemma 4.8 (3), we have $\sigma \models \neg(t \leq u)$. We fix h in order to show $\sigma, h \models H \to \neg\mathsf{Ineq}(t, u)$.

Case 1. If $\sigma, h \not\models H$, the claim follows trivially.

Case 2. Assume $\sigma, h \models H$. Assume $\sigma, h \models \mathsf{Ineq}(t, u)$ for contradiction. Then, there is q such that $h(q) = 2$, $h(q+1) = \sigma(t) + 3$, $h(q+2) = \sigma(u) + 3$. By Lemma 4.3 (3), we have $\sigma(t) \leq \sigma(u)$, a contradiction. ◀

The next lemma says that the truth in PA is equivalent to the truth of the translation in SLN for a large simple table heap.

▶ **Lemma 4.10.** *For a normal formula A in PA and $n \geq \max(\sigma, A)$, $\sigma \models A$ if and only if $\sigma, h_n \models A^\circ$.*

Proof. We will show the claim by induction on A.

Case 1. A is quantifier-free. We will only show the cases for $A \equiv (t \leq u)$ since the cases $t = u$ and $t \neq u$ are obvious and the cases $A \wedge B$ and $A \vee B$ follow from the induction hypothesis. $\sigma \models t \leq u$ is equivalent to $\sigma \models \neg(u \leq t) \vee t = u$. Since $n \geq \max\{\sigma(t), \sigma(u)\}$, by Lemma 4.8 (3), $\sigma \models \neg(u \leq t)$ is equivalent to $\sigma, h_n \models \neg\mathsf{Ineq}(u, t)$. Hence, $\sigma \models t \leq u$ is equivalent to $\sigma, h_n \models \neg\mathsf{Ineq}(u, t) \vee t = u$. Since $\sigma, h_n \models H$ by Lemma 4.7, $\sigma, h_n \models \neg\mathsf{Ineq}(u, t) \vee t = u$ is equivalent to $\sigma, h_n \models H \to \neg\mathsf{Ineq}(u, t) \vee t = u$.

Case 2. $A \equiv \exists x \leq t.B$.

Only-if-direction: By assumption, there is k such that $\sigma[x := k] \models x \leq t \wedge B$. That is, $\sigma[x := k] \models x \leq t$ and $\sigma[x := k] \models B$. Thus, we have $k \leq \sigma(t)$. Since $n \geq \max(\sigma[x := k], B)$, by induction hypothesis, we have $\sigma[x := k], h_n \models B^\circ$. Furthermore, by Lemma 4.8 (3), $\sigma[x := k], h_n \models \mathsf{Ineq}(x, t)$. Thus, we have $\sigma[x := k], h_n \models \mathsf{Ineq}(x, t) \wedge B^\circ$. Hence, $\sigma[x := k], h_n \models \neg\mathsf{Ineq}(t, t) \vee (\mathsf{Ineq}(x, t) \wedge B^\circ)$.

If-direction: Suppose $\sigma[x := k], h_n \models H \to \neg\mathsf{Ineq}(t, t) \vee (\mathsf{Ineq}(x, t) \wedge B^\circ)$ for some k. Since $\sigma[x := k], h_n \models H$, we have $\sigma[x := k], h_n \models \neg\mathsf{Ineq}(t, t) \vee (\mathsf{Ineq}(x, t) \wedge B^\circ)$. Since $n \geq \max(\sigma, A) \geq \sigma(t)$, by the definition of h_n, $\sigma[x := k], h_n \models \mathsf{Ineq}(t, t)$. Thus, we have $\sigma[x := k], h_n \models \mathsf{Ineq}(x, t) \wedge B^\circ$. Since $\sigma[x := k], h_n \models \mathsf{Ineq}(x, t)$, by Lemma 4.8 (3), we have $k \leq \sigma(t)$. Then, since $k \leq \sigma(t) \leq n$, by the induction hypothesis for B, $\sigma[x := k] \models x \leq t \wedge B$.

Case 3. $A \equiv \forall x \leq t.B$. We will show the claim: for all k, $\sigma[x := k] \models \neg(x \leq t) \vee B$ if and only if $\sigma[x := k], h_n \models \neg\mathsf{Ineq}(x, t) \vee B^\circ$. If $k \leq n$, then by Lemma 4.8 (3) and the induction hypothesis for B, the claim holds. If $k > n$, then since $k > n \geq \sigma(t)$, we have $\sigma[x := k] \models \neg(x \leq t)$. On the other hand, by the definition of h_n, we have $\sigma[x := k], h_n \models \neg\mathsf{Ineq}(x, t)$. So the claim holds.

Case 4. $A \equiv \exists(x = t + u)B$. $\sigma \models \exists(x = t + u)B$ is equivalent to $\sigma[x := k] \models x = t + u$ and $\sigma[x := k] \models B$ for some k. Since $n \geq \max\{\sigma(t), \sigma(u)\}$, by Lemma 4.8 (1), $\sigma[x := k] \models x = t + u$ is equivalent to $\sigma[x := k], h_n \models \mathsf{Add}(t, u, x)$. Furthermore, since $n \geq \max(\sigma[x := k], B)$, by induction hypothesis for B, $\sigma[x := k] \models B$ is equivalent to $\sigma[x := k], h_n \models B^\circ$. Therefore, $\sigma \models A$ is equivalent to $\sigma[x := k], h_n \models \mathsf{Add}(t, u, x) \wedge B^\circ$ for some k, which is equivalent to $\sigma, h_n \models \exists x(\mathsf{Add}(t, u, x) \wedge B^\circ)$.

Case 5. $A \equiv \exists(x = y \times z)B$. This case can be shown similarly to Case 4 (except it uses Lemma 4.8 (2)). ◀

The next lemma says that for the translation of a normal formula in PA, the truth for a large simple table heap is the same as the truth for all heaps in the standard interpretation of SLN.

▶ **Lemma 4.11.** *Let A be a normal formula in* PA *and $n \geq \max(\sigma, A)$. Then, $\sigma, h_n \models A^\circ$ if and only if $\sigma, h \models A^\circ$ for all h.*

Proof. The if-direction is trivial. We will show the only-if-direction by induction on A.

Case 1. A is quantifier-free. We will only show the cases for $A \equiv (t \leq u)$ since the cases $t = u$ and $t \neq u$ are obvious and the cases $A \wedge B$ and $A \vee B$ follow from the induction hypothesis. Since $\sigma, h_n \models H$ by Lemma 4.7, $\sigma, h_n \models H \to \neg\mathsf{Ineq}(u, t) \vee t = u$ is equivalent to $\sigma, h_n \models \neg\mathsf{Ineq}(u, t) \vee t = u$. Since $n \geq \max\{\sigma(t), \sigma(u)\}$, by Lemma 4.9 (3), $\sigma, h_n \models \neg\mathsf{Ineq}(u, t)$ is equivalent to $\sigma, h \models H \to \neg\mathsf{Ineq}(u, t)$ for all h. Clearly, $\sigma, h_n \models t = u$ is equivalent to $\sigma, h \models t = u$ for all h. Therefore, we have $\sigma, h \models (H \to \neg\mathsf{Ineq}(u, t)) \vee t = u$ for all h, which is equivalent to $\sigma, h \models H \to \neg\mathsf{Ineq}(u, t) \vee t = u$ for all h.

Case 2. $A \equiv \exists x \le t.B$. Suppose $\sigma, h_n \models H \to \neg\mathsf{lneq}(t,t) \lor \exists x(\mathsf{lneq}(x,t) \land B^\circ)$. Since $\sigma, h_n \models H$ and $\sigma, h_n \models \mathsf{lneq}(t,t)$, we have $\sigma, h_n \models \exists x(\mathsf{lneq}(x,t) \land B^\circ)$, that is, for some k, $\sigma[x := k], h_n \models \mathsf{lneq}(x,t) \land B^\circ$. Let this fact be (a). We fix h in order to show $\sigma, h \models H \to \neg\mathsf{lneq}(t,t) \lor \exists x(\mathsf{lneq}(x,t) \land B^\circ)$. Assume $\sigma, h \models H$. If $\sigma, h \models \neg\mathsf{lneq}(t,t)$, the claim trivially holds. Consider the case $\sigma, h \models \mathsf{lneq}(t,t)$. By (a), we have $\sigma[x := k], h_n \models \mathsf{lneq}(x,t)$. Thus, by Lemma 4.8 (3), $k \le \sigma(t)$. By the case condition, $\sigma, h \models \mathsf{lneq}(t,t)$. Then, by Lemma 4.3 (4), we have $\sigma[x := k], h \models \mathsf{lneq}(x,t)$. Moreover, since $n \ge \max(\sigma[x := k], B)$, by induction hypothesis, we have $\sigma[x := k], h' \models B^\circ$ for all h'. Therefore, we have $\sigma[x := k], h \models B^\circ$. Thus, we have $\sigma[x := k], h \models \mathsf{lneq}(x,t) \land B^\circ$, that is, $\sigma, h \models \exists x(\mathsf{lneq}(x,t) \land B^\circ)$.

Case 3. $A \equiv \forall x \le t.B$. Suppose $\sigma, h_n \models H \to \forall x(\neg\mathsf{lneq}(x,t) \lor B^\circ)$. Since $\sigma, h_n \models H$, we have $\sigma, h_n \models \forall x(\neg\mathsf{lneq}(x,t) \lor B^\circ)$. We fix h in order to show $\sigma, h \models H \to \forall x(\neg\mathsf{lneq}(x,t) \lor B^\circ)$. Assume $\sigma, h \models H$. We fix k in order to show $\sigma[x := k], h \models \neg\mathsf{lneq}(x,t) \lor B^\circ$. We consider the cases for $\sigma[x := k], h \models \mathsf{lneq}(x,t)$ and $\sigma[x := k], h \models \neg\mathsf{lneq}(x,t)$ separately.

Case 3.1. The case $\sigma[x := k], h \models \mathsf{lneq}(x,t)$. Then, there is p such that $h(p) = 2$, $h(p+1) = \sigma[x := k](x) + 3 = k + 3$, $h(p+2) = \sigma[x := k](t) + 3 = \sigma(t) + 3$. By Lemma 4.3 (3), we have $k \le \sigma(t)$. Hence, by Lemma 4.8 (3), we have $\sigma[x := k], h_n \models \mathsf{lneq}(x,t)$. Then, $\sigma[x := k], h_n \models B^\circ$ must be the case. Since $k \le \sigma(t) \le n$, we apply the induction hypothesis to B and obtain $\sigma[x := k], h' \models B^\circ$ for all h'. Hence, we have $\sigma[x := k], h \models B^\circ$. Then, we have the desired result $\sigma[x := k], h \models \neg\mathsf{lneq}(x,t) \lor B^\circ$.

Case 3.2. If $\sigma[x := k], h \models \neg\mathsf{lneq}(x,t)$, then $\sigma[x := k], h \models \neg\mathsf{lneq}(x,t) \lor B^\circ$ trivially holds.

Hence in both cases, we have $\sigma[x := k], h \models \neg\mathsf{lneq}(x,t) \lor B^\circ$.

Case 4. $A \equiv \exists(x = t + u)B$. Then, $A^\circ \equiv H \to \exists x(\mathsf{Add}(t,u,x) \land B^\circ)$. We fix h and assume $\sigma, h \models H$ in order to show $\sigma, h \models \exists x(\mathsf{Add}(t,u,x) \land B^\circ)$. Since $\sigma, h_n \models H$, we have $\sigma, h_n \models \exists x(\mathsf{Add}(t,u,x) \land B^\circ)$. That is, there exists k such that $\sigma[x := k], h_n \models \mathsf{Add}(t,u,x) \land B^\circ$, which is equivalent to $\sigma[x := k], h_n \models \mathsf{Add}(t,u,x)$ and $\sigma[x := k], h_n \models B^\circ$. By Lemma 4.9 (1), $\sigma[x := k], h_n \models \mathsf{Add}(t,u,x)$ is equivalent to $\sigma[x := k], h' \models H \to \mathsf{Add}(t,u,x)$ for all h'. Since we assumed $\sigma, h \models H$, we have $\sigma[x := k], h \models \mathsf{Add}(t,u,x)$. Moreover, since $n \ge \max(\sigma[x := k], B)$, by induction hypothesis for B, we have $\sigma[x := k], h' \models B^\circ$ for all h'. Thus, we have $\sigma[x := k], h \models B^\circ$. Therefore, we have $\sigma[x := k], h \models \mathsf{Add}(t,u,x) \land B^\circ$, that is, $\sigma, h \models \exists x(\mathsf{Add}(t,u,x) \land B^\circ)$.

Case 5. $A \equiv \exists(x = y \times z)B$. This case can be shown similarly to Case 4 (except it uses Lemma 4.9 (2)). ◀

Now we have the main lemma, which says that the truth of a normal formula with $\forall$ in PA for the standard model is the same as the truth of its translation in SLN for the standard interpretation for all heaps.

▶ **Lemma 4.12.** *If A is a normal formula in* PA*, $\sigma \models \forall xA$ if and only if $\sigma, h \models (\forall xA)^\circ$ for all h.*

Proof. $\sigma \models \forall xA$ is equivalent to $\sigma[x := k] \models A$ for all $k \in \mathbb{N}$. We fix k. Let $n \ge \max(\sigma[x := k], A)$. By Lemma 4.10, $\sigma[x := k] \models A$ is equivalent to $\sigma[x := k], h_n \models A^\circ$. Then, by Lemma 4.11, this is equivalent to $\sigma[x := k], h \models A^\circ$ for all h. Therefore, $\sigma[x := k] \models A$ is equivalent to $\sigma[x := k], h \models A^\circ$ for all h. Hence, $\sigma[x := k] \models A$ for all k is equivalent to $\sigma[x := k], h \models A^\circ$ for all h for all k. Thus, $\sigma \models \forall xA$ is equivalent to $\sigma, h \models \forall xA^\circ$ for all h, that is, $\sigma, h \models (\forall xA)^\circ$ for all h. ◀

5 Translation from PA into SLN

In this section, we will present the translation of a Π^0_1 formula in PA to a formula in SLN and prove that the translation preserves the validity and the non-validity. In order to define the translation, first we will define a translation of a Π^0_1 formula in PA into an equivalent normal formula with one universal quantifier in PA. Finally we will define the translation by combining the two translations and will present the main theorem, which says a Π^0_1 formula in PA can be simulated in the weak fragment SLN of separation logic. We also discuss a counterexample for the translation when we extend it to Σ^0_1 formulas.

First we will transform a Π^0_1 formula in PA into a normal formula with one universal quantifier in PA. For simplicity, we use vector notation $\overrightarrow{e}$ for a sequence $e_1, ..., e_n$ of objects.

▶ **Proposition 5.1.** *If A is a bounded formula in* PA*, there is a normal formula B such that $A \leftrightarrow B$ is valid.*

Proof. First, transform A into a prenex normal form and replace $\neg(t \leq u)$ by $u \leq t \wedge u \neq t$ to obtain $A' \equiv \overrightarrow{Qx \leq t}.C$, where C is a quantifier-free disjunctive normal form without formulas of the form $\neg(t \leq u)$. Choose the leftmost occurrence among the innermost occurrences of $u + v$ or $u \times v$ in A' and explicitly denote it by $A'[u + v]$ or $A'[u \times v]$.

Let $A'[z]$ be the formula obtained from $A'[u + v]$ or $A'[u \times v]$ by replacing the occurrence of $u + v$ or $u \times v$ in A' by a fresh z. Define $\overrightarrow{Qx' \leq t'}.D$ by $A'[z] \equiv \overrightarrow{Qx' \leq t'}.D$ where $\overrightarrow{Qx' \leq t'}$ is the longest prefix such that z is not in t', namely, it has the longest $\overrightarrow{Qx' \leq t'}$ among such $\overrightarrow{Qx' \leq t'}$'s. We transform D into $\exists(z = u + v)D$ or $\exists(z = u \times v)D$.

We repeat this process until we have the form $\overrightarrow{\{Qx \leq y, \exists(x = t)\}}A''$, where t is of the form $a + b$ or $a \times b$ for some terms a, b that do not contain $+$ or $\times$, and A'' and u do not contain $+, \times$. Define B as this result. ◀

We define the translation $A^\square$ by using the proof of the previous proposition.

▶ **Definition 5.2** (Translation $(\cdot)^\square$)**.** Let $A \equiv \forall x B$ be a Π^0_1 formula in PA, where B contains only bounded quantifiers. Let B' be a normal form of B obtained by the procedure described in the proof of Proposition 5.1. We define $A^\square \equiv \forall x B'$.

Example. For a formula

$$A \equiv \forall y \leq x + (x + s(x)).(0 \leq x + (y \times (x + y))),$$

its translation $A^\square$ is

$$\begin{aligned} A^\square \equiv & \exists(x_1 = x + s(x))\exists(x_2 = x + x_1)\forall y \leq x_2. \\ & \exists(x_3 = x + y)\exists(x_4 = y \times x_3)\exists(x_5 = x + x_4)(0 \leq x_5). \end{aligned}$$

Now, we have the main theorem which says that Π^0_1 formulas can be translated into SLN formulas preserving the validity and the non-validity.

▶ **Theorem 5.3.** *For a Π^0_1 formula A in* PA*, A is valid in the standard model of* PA *if and only if $A^{\square\circ}$ is valid in the standard interpretation of* SLN*.*

Proof. By Proposition 5.1 and Lemma 4.12. ◀

As a by-product of the above theorem, we have the undecidability of SLN.

▶ **Corollary 5.4.** *The validity of* SLN *formulas is undecidable.*

Proof. Given a Turing machine, its halting problem statement P is Σ^0_1, since it can be expressed as $\exists z.T(e,e,z)$, where e is the index of the given Turing machine and T is Kleene's T-predicate which is primitive recursive (for rigorous definition, see e.g. [13]). Thus, $\neg P$ is Π^0_1. By Theorem 5.3, $\neg P$ is valid in PA if and only if $(\neg P)^{\Box\circ}$ is valid in SLN. If the validity in SLN is decidable, we can decide whether P is true in the standard model and this contradicts the undecidability of the halting problem. Hence the validity in SLN is undecidable. ◀

We have just proved that Π^0_1 formulas can be translated preserving the validity and the non-validity. We might extend the translation $(\cdot)^\circ$ by $(\exists x A)^\circ \equiv \exists x A^\circ$. However, the extended translation does not preserve the validity and the non-validity, which is shown in the next proposition.

▶ **Proposition 5.5.** *There is some* Σ^0_1 *closed formula* A *such that* A *is not valid in* PA *but* $A^{\Box\circ}$ *is valid in* SLN.

Proof. Consider the formula $A \equiv \exists x(x+0 \neq x)$. This sentence is clearly not valid in PA. However, we can prove that $\sigma, h \models A^{\Box\circ}$ for all σ, h as follows. By the procedure in the proof of Proposition 5.1, $A^{\Box} \equiv \exists x \exists (z = x+0)(z \neq x)$. Thus, $A^{\Box\circ} \equiv \exists x(H \to \exists z(\mathsf{Add}(x,0,z) \land z \neq x))$. We fix σ, h in order to prove $\sigma, h \models A^{\Box\circ}$. Let $n = \max\{k \mid h(p) = 0, h(p+1) = k+3, h(p+2) = 3\} + 1$, and $m = n+1$. Let $\sigma' = \sigma[x := n, z := m]$. We will show $\sigma', h \models \mathsf{Add}(x,0,z) \land z \neq x$ assuming $\sigma', h \models H$. By choice of n, we have $\sigma', h \models \forall a \neg(a \hookrightarrow 0, s^3(\overline{n}), \overline{3})$. Thus, $\sigma', h \models \mathsf{Add}(\overline{n}, 0, \overline{m})$ holds, because the premise of $\mathsf{Add}(\overline{n}, 0, \overline{m})$ is false. Therefore, $\sigma', h \models \mathsf{Add}(x,0,z)$. Furthermore, clearly $\sigma', h \models z \neq x$. Hence, $\sigma, h \models A^{\Box\circ}$ for all h. ◀

6 Another undecidability proof

In this section, we will give another proof of the undecidability of the validity of SLN given in Corollary 5.4, where it can be proved in a way similar to that in [7]. This proof is simpler than the proof of Theorem 5.3, but this proof cannot show the representation of Peano arithmetic in the separation logic SLN with numbers.

A first-order language L is defined as that with a binary predicate symbol P and without any constants or function symbols. Namely,

Terms $t ::= x$.

Formulas $A ::= t = t \mid P(t,t) \mid \neg A \mid A \land A \mid \exists x.A$.

A finite structure is defined as (U, R) where $U \subseteq \mathbb{N}$ and U is finite and $R \subseteq U^2$. σ is a variable assignment of (U, R) if $\sigma : \mathrm{Vars} \to U$. We define σ_0 as $\sigma_0(x) = 0$ for all variables x.

We write $M, \sigma \models A$ to denote that a formula A is true by a variable assignment σ of a structure M.

The idea of this proof is to encode a finite structure (U, R) for the language L by a heap h such that

- $n \in U$ iff h has some entry of $0, n+2$, and
- $(n,m) \in R$ iff h has some entry of $1, n+2, m+2$.

▶ **Definition 6.1.** For a given finite structure $M = (U, R)$ of L, we define the heap h_M by

$$\begin{aligned}
&\mathsf{Dom}(h) = \{0, 1, \ldots, 2k + 3l - 1\},\\
&h_M(x) = 0 \quad (x = 2i, i < k),\\
&h_M(x) = p_i + 2 \quad (x = 2i + 1, i < k),\\
&h_M(x) = 1 \quad (x = 2k + 3i, i < l),\\
&h_M(x) = n_i + 2 \quad (x = 2k + 3i + 1, i < l),\\
&h_M(x) = m_i + 2 \quad (x = 2k + 3i + 2, i < l),
\end{aligned}$$

where $U = \{p_i \mid i < k\}$ and $R = \{(n_i, m_i) \mid i < l\}$.

The heap h_M has information of a given structure M.

▶ **Definition 6.2.** For a given heap h, if $\sigma_0, h \models \exists a x(a \hookrightarrow 0, s^2(x))$, we define a structure $M_h = (U_h, R_h)$ by

$$\begin{aligned}
&U_h = \{n \mid \sigma_0[x := n], h \models \exists a(a \hookrightarrow 0, s^2(x))\},\\
&R_h = \{(n, m) \mid \sigma_0[x := n, y := m], h \models \exists a(a \hookrightarrow \overline{1}, s^2(x), s^2(y))\}.
\end{aligned}$$

The structure M_h is a structure represented by a given heap h.

We define a translation $(\cdot)^\triangle$ from L into SLN.

▶ **Definition 6.3.** For a formula A in the language L, we define the formula $A^\triangle$ in SLN by

$$\begin{aligned}
&(x = y)^\triangle \equiv x = y \wedge \exists a(a \hookrightarrow 0, s^2(x)),\\
&(P(x, y))^\triangle \equiv \exists a(a \hookrightarrow \overline{1}, s^2(x), s^2(y)) \wedge \exists b(b \hookrightarrow 0, s^2(x)) \wedge \exists c(c \hookrightarrow 0, s^2(y)),\\
&(\exists x.A)^\triangle \equiv \exists x(\exists a(a \hookrightarrow 0, s^2(x)) \wedge A^\triangle),\\
&(\neg A)^\triangle \equiv \neg A^\triangle,\\
&(A \wedge B)^\triangle \equiv A^\triangle \wedge B^\triangle.
\end{aligned}$$

The next is a well-known theorem for finite structures [9].

▶ **Theorem 6.4** (Trakhtenbrot). *The validity of formulas in the language L for every finite structure is undecidable.*

The next lemma shows the equivalence for any formulas.

▶ **Lemma 6.5.** *$M, \sigma \models A$ for all finite M for all variable assignments σ of M iff $\sigma, h \models \exists a x(a \hookrightarrow 0, s^2(x)) \to \bigwedge_{x \in \mathsf{FV}(A)} \exists a(a \hookrightarrow 0, s^2(x)) \to A^\triangle$ for all h and all variable assignments σ.*

Proof. If-direction: For a given finite structure M, we can construct the heap h_M and by induction on A we can show that $\sigma, h_M \models A^\triangle$ iff $M, \sigma \models A$, for every variable assignment σ of M.

Only-if-direction: For a given heap h such that $\sigma_0, h \models \exists a x(a \hookrightarrow 0, s^2(x))$, we can construct the finite structure M_h and by induction on A we can show that $M_h, \sigma \models A$ iff $\sigma, h \models A^\triangle$, for every variable assignment σ of M_h. To show the only-if-direction in the statement of the lemma by using this claim, from the assumption $\sigma_0, h \models \exists a x(a \hookrightarrow 0, s^2(x))$, we have p, q such that $h(p) = 0$ and $h(p + 1) = q + 2$, and for a given σ we apply this claim with the variable assignment σ' of M_h such that $\sigma'(x) = \sigma(x)$ $(x \in \mathsf{FV}(A))$ and $\sigma'(x) = q$ $(otherwise)$. ◀

Another proof of Corollary 5.4. Taking a closed formula A in Lemma 6.5, we have the equivalence: A is true in all finite structure M iff $\exists ax(a \hookrightarrow 0, s^2(x)) \to A^{\triangle}$ is valid in the standard interpretation of SLN.

By Theorem 6.4, the validity of SLN for the standard interpretation is undecidable. ◀

7 Conclusion

We have presented the translation from Π^0_1 formulas in PA into formulas in the fragment SLN of separation logic with numbers, which has only $\hookrightarrow$, 0 and the successor function, and proved that this translation preserves the validity and the non-validity for the standard model of PA and the standard interpretation of SLN. By this, we have shown that Π^0_1 formulas in Peano arithmetic can be simulated by SLN. As a corollary, we have proved the validity of SLN is undecidable. We have also given a counterexample when we extend this translation to Σ^0_1 formulas.

Future work would be to present a translation from other logical systems into SLN, and prove the preservation of the validity and the non-validity by extending our technique used in this paper. Another future work would be to try to show the validity of SLN is Π^0_1-complete.

References

1 Timos Antonopoulos, Nikos Gorogiannis, Christoph Haase, Max I. Kanovich, and Joël Ouaknine. Foundations for decision problems in separation logic with general inductive predicates. In Anca Muscholl, editor, *Foundations of Software Science and Computation Structures – 17th International Conference, FOSSACS 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014, Proceedings*, volume 8412 of *Lecture Notes in Computer Science*, pages 411–425. Springer, 2014. doi:10.1007/978-3-642-54830-7_27.

2 Josh Berdine, Cristiano Calcagno, and Peter W. O'Hearn. A decidable fragment of separation logic. In Kamal Lodaya and Meena Mahajan, editors, *FSTTCS 2004: Foundations of Software Technology and Theoretical Computer Science, 24th International Conference, Chennai, India, December 16-18, 2004, Proceedings*, volume 3328 of *Lecture Notes in Computer Science*, pages 97–109. Springer, 2004. doi:10.1007/978-3-540-30538-5_9.

3 Josh Berdine, Cristiano Calcagno, and Peter W. O'Hearn. Symbolic execution with separation logic. In Kwangkeun Yi, editor, *Programming Languages and Systems, Third Asian Symposium, APLAS 2005, Tsukuba, Japan, November 2-5, 2005, Proceedings*, volume 3780 of *Lecture Notes in Computer Science*, pages 52–68. Springer, 2005. doi:10.1007/11575467_5.

4 Rémi Brochenin, Stéphane Demri, and Étienne Lozes. On the almighty wand. *Inf. Comput.*, 211:106–137, 2012. doi:10.1016/j.ic.2011.12.003.

5 Cristiano Calcagno, Dino Distefano, Peter W. O'Hearn, and Hongseok Yang. Compositional shape analysis by means of bi-abduction. *J. ACM*, 58(6):26:1–26:66, 2011. doi:10.1145/2049697.2049700.

6 Cristiano Calcagno, Philippa Gardner, and Matthew Hague. From separation logic to first-order logic. In Vladimiro Sassone, editor, *Foundations of Software Science and Computational Structures, 8th International Conference, FOSSACS 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2005, Edinburgh, UK, April 4-8, 2005, Proceedings*, volume 3441 of *Lecture Notes in Computer Science*, pages 395–409. Springer, 2005. doi:10.1007/978-3-540-31982-5_25.

7 Cristiano Calcagno, Hongseok Yang, and Peter W. O'Hearn. Computability and complexity results for a spatial assertion language for data structures. In *The Second Asian Workshop on Programming Languages and Systems, APLAS'01, Korea Advanced Institute of Science and Technology, Daejeon, Korea, December 17-18, 2001, Proceedings*, pages 289–300, 2001.

8 Byron Cook, Christoph Haase, Joël Ouaknine, Matthew J. Parkinson, and James Worrell. Tractable reasoning in a fragment of separation logic. In Joost-Pieter Katoen and Barbara König, editors, *CONCUR 2011 – Concurrency Theory – 22nd International Conference, CONCUR 2011, Aachen, Germany, September 6-9, 2011. Proceedings*, volume 6901 of *Lecture Notes in Computer Science*, pages 235–249. Springer, 2011. doi:10.1007/978-3-642-23217-6_16.

9 Heinz-Dieter Ebbinghaus and Jörg Flum. *Finite Model Theory*. Springer-Verlag, 1995.

10 Radu Iosif, Adam Rogalewicz, and Jirí Simácek. The tree width of separation logic with recursive definitions. In Maria Paola Bonacina, editor, *Automated Deduction – CADE-24 – 24th International Conference on Automated Deduction, Lake Placid, NY, USA, June 9-14, 2013. Proceedings*, volume 7898 of *Lecture Notes in Computer Science*, pages 21–38. Springer, 2013. doi:10.1007/978-3-642-38574-2_2.

11 Koji Nakazawa, Makoto Tatsuta, Daisuke Kimura, and Mitsuru Yamamura. Spatial factorization in cyclic-proof system for separation logic. *Computer Software*, 37(1):1_125–1_144, 2020. doi:10.11309/jssst.37.1_125.

12 Peter W. O'Hearn. Separation logic. *Commun. ACM*, 62(2):86–95, 2019. doi:10.1145/3211968.

13 Joseph R. Shoenfield. *Mathematical Logic*. Addison-Wesley, 1967.

14 Makoto Tatsuta and Daisuke Kimura. Separation logic with monadic inductive definitions and implicit existentials. In Xinyu Feng and Sungwoo Park, editors, *Programming Languages and Systems – 13th Asian Symposium, APLAS 2015, Pohang, South Korea, November 30 – December 2, 2015, Proceedings*, volume 9458 of *Lecture Notes in Computer Science*, pages 69–89. Springer, 2015. doi:10.1007/978-3-319-26529-2_5.

15 Makoto Tatsuta, Quang Loc Le, and Wei-Ngan Chin. Decision procedure for separation logic with inductive definitions and presburger arithmetic. In Atsushi Igarashi, editor, *Programming Languages and Systems – 14th Asian Symposium, APLAS 2016, Hanoi, Vietnam, November 21-23, 2016, Proceedings*, volume 10017 of *Lecture Notes in Computer Science*, pages 423–443, 2016. doi:10.1007/978-3-319-47958-3_22.

16 Makoto Tatsuta, Koji Nakazawa, and Daisuke Kimura. Completeness of cyclic proofs for symbolic heaps with inductive definitions. In Anthony Widjaja Lin, editor, *Programming Languages and Systems – 17th Asian Symposium, APLAS 2019, Nusa Dua, Bali, Indonesia, December 1-4, 2019, Proceedings*, volume 11893 of *Lecture Notes in Computer Science*, pages 367–387. Springer, 2019. doi:10.1007/978-3-030-34175-6_19.