Logical Methods in Computer Science
Volume 22, Issue 2, 2026, pp. 22:1–22:30
https://lmcs.episciences.org/

Submitted Jul. 02, 2025
Published May 25, 2026

ENCODING PEANO ARITHMETIC IN A MINIMAL FRAGMENT OF SEPARATION LOGIC

SOHEI ITO [a] AND MAKOTO TATSUTA [b]

[a] Nagasaki University, Japan
e-mail address: s-ito@nagasaki-u.ac.jp

[b] Toho University, Japan
e-mail address: tatsuta008@gmail.com

Abstract. This paper investigates the expressive power of a minimal fragment of separation logic extended with natural numbers. Specifically, it demonstrates that the fragment consisting solely of the intuitionistic points-to predicate, the constant 0, and the successor function is sufficient to encode all Π^0_1 formulas of Peano Arithmetic (PA). The authors construct a translation from PA into this fragment, showing that a Π^0_1 formula is valid in the standard model of arithmetic if and only if its translation is valid in the standard interpretation of the separation logic fragment. This result implies the undecidability of validity in the fragment, despite its syntactic simplicity. The translation leverages a heap-based encoding of arithmetic operations—addition, multiplication, and inequality—using structured memory cells. The paper also explores the boundaries of this encoding, showing that the translation does not preserve validity for Σ^0_1 formulas. Additionally, an alternative undecidability proof is presented via a reduction from finite model theory. Finally, the paper establishes that the validity problem for this fragment is Π^0_1-complete, highlighting its theoretical significance in the landscape of logic and program verification.

1. Introduction

Separation logic has proved to be both theoretically robust and practically effective for verifying heap-manipulating programs [CDOY11, O'H19], owing to its concise representation of memory states. However, when verifying software systems that involve numerical computations, it becomes necessary to extend separation logic with arithmetic. This extension raises fundamental questions about the decidability of validity in such systems under standard interpretations of numbers, as logical frameworks with decidable validity are generally more suitable for software verification.

Presburger arithmetic, known for its decidable validity, offers a promising candidate for integration with separation logic. One might expect that combining a decidable fragment of separation logic with Presburger arithmetic would yield a decidable system. Contrary to this expectation, we show that even a minimal extension of separation logic with arithmetic can lead to undecidability.

In this paper, we study a minimal fragment of separation logic—denoted SLN—that includes only the intuitionistic points-to predicate ($\hookrightarrow$), the constant 0, and the successor

Key words and phrases: First order logic, Separation logic, Peano arithmetic, Logical Complexity.

LOGICAL METHODS IN COMPUTER SCIENCE
DOI:10.46298/LMCS-22(2:22)2026

function s. We prove that this fragment is expressive enough to simulate all Π^0_1 formulas of Peano Arithmetic (PA). Our main result is a representation theorem establishing a translation from Π^0_1 formulas in PA to SLN formulas that preserves both validity and non-validity under their respective standard interpretations. As a corollary, we derive the undecidability of validity in SLN. It is surprising that such a weak fragment of separation logic becomes so expressive merely by adding 0 and s, especially considering that the fragment containing only $\hookrightarrow$ is known to be decidable with respect to validity under the standard interpretation [BDL12].

The core technique in proving our representation theorem involves constructing an operation table for addition, multiplication, and inequality within a heap. This allows us to eliminate expressions such as $x + y = z$, $x \times y = z$, and $x \leq y$ by referencing either the value z or the truth of the inequality $x \leq y$. To encode these operations, we use consecutive heap cells containing entries of the form $0, x+3, y+3, x+y+3$ for addition, $1, x+3, y+3, x \times y+3$ for multiplication, and $2, x + 3, y + 3$ for inequality. Here, the constant 3 serves as an offset, and the tags 0, 1, and 2 distinguish between the three operations. To define the translation formally, we introduce the notion of a normal form for bounded formulas in Peano Arithmetic.

While our translation can be extended to arbitrary PA formulas, the representation theorem does not hold beyond the Σ^0_1 class. We provide a counterexample involving a Σ^0_1 formula to illustrate this limitation.

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 validity in the separation logic with numbers itself can be proved in a simpler way, by using a similar idea to [CYO01]. We will also give a proof in that way.

Our representation theorem implies that the validity problem in SLN is Π^0_1-hard, establishing the lower bound of its complexity. For the upper bound, we show that the problem belongs to Π^0_1 by proving (1) that the model-checking problem in SLN is decidable, and (2) that validity in SLN can be expressed as a Π^0_1 formula using (1). Together, these results establish that the validity problem in SLN is Π^0_1-complete.

There are several undecidability results concerning validity in separation logic, some of which rely on translation techniques. For instance, separation logic with the 1-field points-to predicate and the separating implication is known to be undecidable with respect to validity [BDL12]. Similarly, separation logic with the 2-field points-to predicate has also been shown to be undecidable [CYO01], with the proof relying on a translation from first-order logic with a single binary relation into separation logic with the 2-field points-to predicate.

On the other hand, there are some decidability results. Separation logic with the 1-field points-to predicate, when the separating implication is excluded, is known to be decidable [BDL12]. Additionally, the quantifier-free separation logic has been shown to be decidable [CGH05], with the proof involving a translation into first-order logic with an empty signature.

To the best of our knowledge, there is no existing work that translates any fragment of arithmetic into such a weak form of separation logic containing only $\hookrightarrow, 0, s$.

When restricted to symbolic heaps in separation logic with arithmetic or inductive definitions, several decidability results have been established. These include symbolic heap entailment with Presburger arithmetic [TLC16], bounded-treewidth symbolic heap entailment [IRS13], symbolic heap entailment with cone inductive definitions [TNK19, NTKY20], symbolic heap entailment with lists [BCO04, BCO05, CHO+11, AGH+14], and symbolic heap entailment with Presburger arithmetic, arrays, and lists [KT21]. The satisfiability problem for symbolic heaps with general inductive predicates is also known to be decidable [BFPG14].

However, even within symbolic heaps, relaxing certain conditions can lead to undecidability. For instance, symbolic heap entailment with unrestricted inductive definitions [IRS13], and symbolic heap entailment with bounded-treewidth inductive definitions and implicit existentials [TK15], are both known to be undecidable. A comprehensive study on the decidability of symbolic heaps is provided in [KZ20].

Another fragment of separation logic with arithmetic has been shown to have a decidable satisfiability problem. Specifically, when the fragment includes inductive predicates that capture both shape and arithmetic properties, satisfiability remains decidable provided the arithmetic constraints can be expressed as semilinear sets—which are themselves decidable in Presburger arithmetic [LTSC17].

A recent study has established the undecidability of the entailment problem in separation logic with inductively defined spatial predicates when certain forms of theory reasoning are permitted [EP23]. This includes reasoning over theories involving the successor function and numerical values. In contrast, our result focuses on a significantly more restricted setting: we allow only the 1-field intuitionistic points-to predicate as the spatial component, which is far less expressive. Nevertheless, we prove that even within this highly limited fragment, the logic becomes undecidable.

This paper is organized as follows: Section 2 defines Peano arithmetic. Section 3 introduces the SLN fragment and its semantics. Section 4 presents the translation from normal PA formulas to SLN and proves the preservation properties. Section 5 defines an auxiliary translation to normal form and proves the main theorem on the preservation of the translation from Π^0_1 formulas to SLN. Section 6 provides an alternative proof of undecidability of validity in SLN. Section 7 establishes Π^0_1-completeness of SLN. Section 8 concludes.

This paper extends our previous conference publication [IT24] by enriching the related work, clarifying technical details, and adding the new result on Π^0_1-completeness.

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 σ : 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 $\forall x \le t.A$ is an abbreviation of $\forall x(x \le t \to A)$, where t does not contain x. A formula $\exists x \le t.A$ is an abbreviation of $\exists x(x \le t \wedge A)$, where t does not contain x. We call $\forall x \le t$ and $\exists x \le t$ *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 SLN

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. 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 notion of validity defined in this section refers to validity under the standard interpretation of SLN, which differs from the conventional notion of validity in separation logic. This difference arises because, in the conventional definition, the interpretation depends on the set of addresses.

Our fragment SLN contains only equality, the intuitionistic points-to predicate $(t_1 \hookrightarrow t_2)$, the constant 0, successor function s, Boolean connectives, and first-order quantifiers. It omits separating conjunction ($*$) and magic wand ($-\!*$). Thus SLN is essentially first-order logic over a finite partial function $h : \mathbb{N} \to_{\text{fin}} \mathbb{N}$ enriched with a single spatial atom.

We retain the term "separation logic fragment" because the semantics of $\hookrightarrow$ is the standard SL heap-cell predicate, and our results delineate how adding only arithmetic symbols $0, s$ to this minimal spatial core already suffices to represent all Π^0_1 PA sentences. We contrast our setting with classical undecidability and decidability borders for SL fragments in Section 1 related work (e.g., with/without $*$, or with multi-field points to predicate).

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 only for normal formulas. This does not lose the generality since any bounded formula can be transformed into a normal formula, as will be shown 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:

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

satisfying the following conditions: (1) B is a disjunctive normal form of a formula in PA without quantifiers, $+$, $\times$, and formulas of the form $\neg(t \le u)$, (2) each t in $\forall x \le t$ and $\exists x \le t$ 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 $\le$. 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 $\le$ 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:

$$H_{\mathsf{Add1}} \equiv \forall ay((a \hookrightarrow 0, [0], [y]) \to s^3(a) \hookrightarrow [y]),$$

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

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 serve as reserved tags identifying the operation (addition, multiplication, or inequality), the arguments and results are encoded by adding an offset of 3. Any fixed offset greater than 2 would work to avoid collisions with the tags. We use 3 for concreteness. 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.

The conjuncts of H enforce coherence of existing operation-table entries rather than totality. If a heap contains an entry tagged as $\mathsf{Add}/\mathsf{Mult}/\mathsf{Ineq}$ for arguments (x, y), then the successor cells must contain the correct outputs (Lemma 4.3). If no matching entry occurs, the corresponding implication in H is trivially true. Hence H admits any finite consistent partial operation graph that is closed under the predecessor/step conditions spelled out in $H_{\mathsf{Add1}}, H_{\mathsf{Add2}}, H_{\mathsf{Mult1}}, H_{\mathsf{Mult2}}, H_{\mathsf{Ineq1}}, H_{\mathsf{Ineq2}}$. This permissiveness is crucial for our Π^0_1 preservation (Lemmas 4.13-4.15). Validity is required over all heaps, so if some heaps are "too small", the translated formulas become trivially true there, while suitable "large" heaps (e.g., the simple tables h_n defined in Definition 4.9) realize the intended arithmetic and force correctness. By monotonicity of arithmetic facts, truth then aligns across all heaps. In contrast, this permissiveness breaks preservation for Σ^0_1. The witness may demand the presence of a specific table entry that small heaps can avoid (Proposition 5.6).

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 $u \le u$, then it contains all the entries for $t \le u$.

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 $n \le k$.

(4) If $\sigma, h \models H$, $\sigma(t) \le \sigma(u)$, $\sigma, h \models \mathsf{Ineq}(u, u)$, 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

$$\begin{aligned} &h(m) = 0, \\ &h(m+1) = 3, \\ &h(m+2) = k+3, \\ &\sigma, h \models H_{\mathsf{Add1}}, \end{aligned}$$

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, $n-1 \ge 0$. Let $\sigma' = \sigma[a := m, x := n-1, y := k]$. Since

$$\begin{aligned} &h(m) = 0, \\ &h(m+1) = n+3, \\ &h(m+2) = k+3, \\ &\sigma, h \models H_{\mathsf{Add2}}, \end{aligned}$$

we have

$$\sigma', h \models \exists bz((b \hookrightarrow 0, [x], [y], [z]) \wedge s^3(a) \hookrightarrow [s(z)]).$$

Thus, 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,

$$\begin{aligned} &h(q) = 0, \\ &h(q+1) = (n-1)+3, \\ &h(q+2) = k+3, \\ &h(q+3) = \ell+3, \\ &h(m+3) = \ell+4. \end{aligned}$$

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

$$h(m) = 1,$$
$$h(m+1) = 3,$$
$$h(m+2) = k+3,$$
$$\sigma, h \models H_{\mathsf{Mult1}},$$

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

$$h(m) = 1,$$
$$h(m+1) = n+3,$$
$$h(m+2) = k+3,$$
$$\sigma, h \models H_{\mathsf{Mult2}},$$

we have

$$\sigma', h \models \exists bz((b \hookrightarrow \overline{1}, [x], [y], [z]) \wedge \exists cw((c \hookrightarrow 0, [z], [y], [w]) \wedge s^3(a) \hookrightarrow [w])).$$

Thus, 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,

$$h(r) = 0,$$
$$h(r+1) = \ell+3,$$
$$h(r+2) = k+3,$$
$$h(r+3) = p+3,$$
$$h(m+3) = p+3.$$

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

$$h(m) = 2,$$
$$h(m+1) = n+3,$$
$$h(m+2) = k+3,$$
$$\sigma, h \models H_{\mathsf{Ineq1}},$$

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:

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

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

For a normal formula, the translation computes $t+u$ by referring to the operation table in the current heap. H guarantees that the operation table in the heap is correct. However, the operation table may not be sufficiently large for computing $t+u$. When the operation table is not sufficiently large for computing $t+u$, then $\mathsf{Add}(t,u,x)$ returns true. The use of Ineq and Mult is similar to Add. Note that $\mathsf{Ineq}(t,t)$ means that t is in the operation table. Hence $\neg\mathsf{Ineq}(t,t)$ means that t is not in the operation table and we define $(\exists x \le t.B)^\circ$ as true in this case. For a non-normal formula, we just keep $\forall$ in the translation.

Example 4.5. For a normal formula

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

its translation A° is

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

Example 4.6. For a normal formula

$$B \equiv \exists(x_1 = x + s(x))\exists y \le x_1.(0 \le y),$$

its translation B° is

$$\begin{aligned} B^\circ \equiv\ & H \to \exists x_1(\mathsf{Add}(x, s(x), x_1) \wedge (H \to \neg\mathsf{Ineq}(x_1, x_1) \vee \\ & \exists y(\mathsf{Ineq}(y, x_1) \wedge (H \to \neg\mathsf{Ineq}(y, 0) \vee y = 0)))) \end{aligned}$$

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. Furthermore, the formulas $\mathsf{Add}(t,u,x)$ and $\mathsf{Mult}(t,u,x)$ state that for any address a, if the cells at $a+1$ and $a+2$ contain the operands t and u, respectively, then the cell at $a+3$ contains the result x. Consequently, if a heap h does not include a sufficiently large table to store the operands for $x = t+u$ or $x = t \times u$, 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 4.13 and 4.14 later.

Since we demand that A° hold for all heaps, we define the translation of $t \le u$ 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 $t \le u$, which is not possible if the heap is not sufficiently large. Furthermore, the translation of $\exists x \le t.B$ 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 $x \le t$ 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 $\cdot \le t$.

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 4.7. 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:

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

$\max(\sigma, A)$ computes the maximum in values of expressions by σ in A' where A' is obtained by expanding bounded quantifiers of A.

Example 4.8. By using the above definition one by one, we have

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

$$
\begin{aligned}
&= \max\{\sigma(x + s(x)), \max\{\sigma(x + (x + s(x))), \max\{\sigma(x + (x + s(x))),\\
&\quad \max\{\sigma(x + (x + (x + s(x)))),\\
&\quad \max\{\sigma((x + s(x)) \times (x + (x + (x + s(x))))),\\
&\quad \max\{\sigma(x + (x + s(x)) \times (x + (x + (x + s(x))))),\\
&\quad \max\{0, \sigma(x + (x + (x + s(x))) \times (x + (x + (x + s(x)))))\}\}\}\}\}\}\\
&= \sigma(x) + (3\sigma(x) + 1)(4\sigma(x) + 1).
\end{aligned}
$$

Next, for a given size n, we define a heap that supports addition for arguments up to n^2, and multiplication and inequality for arguments up to n. We refer to this as a *simple table heap*.

Definition 4.9. 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 $c_2 = c_1 + 4(n+1)^2$.

The heap h_n has the operation table that has entries of $+$ for arguments up to n^2 and the entries of $\times$ and $\leq$ for arguments up to 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, $h(c_1+4i) = 1$, $h(c_1+4i+1) = x+3$, $h(c_1+4i+2) = y+3$ 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 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 4.10. *For a variable assignment σ, we have $\sigma, h_n \models H$.*

Proof. We check each conjunct of H.

H_{Add1}: For any a, y, if $h_n(a) = 0, h_n(a+1) = 3$, and $h_n(a+2) = y+3$, then by Definition 4.9 the block starting at a is the $(0, 0, y)$-entry and $h_n(a+3) = y+3$.

H_{Add2}: Suppose $h_n(a) = 0, h_n(a+1) = x+4, h_n(a+2) = y+3$. By Definition 4.9, $h_n(a+3) = (x+1)+y+3$. Since h_n has a block for $(0, x, y)$ at some b with $h_n(b) = 0, h_n(b+1) = x+3, h_n(b+2) = y+3, h_n(b+3) = x+y+3$, the claim holds.

H_{Mult1}, H_{Mult2}: Analogous, using the multiplication lines in Definition 4.9 and the identity $(x+1) \times y = x \times y + y$.

$H_{\mathsf{Ineq1}}, H_{\mathsf{Ineq2}}$: For $h_n(a) = 2$ and arguments $(x+1, y)$, Definition 4.9 places entries for the predecessor pair $(x, y-1)$ (sine $y > 0$ by $x+1 \leq y$) and ensures the monotone closure (x, y) as required. ☐

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.

For $t+u$, since n is greater than or equal to the arguments t, u, the heap h_n is sufficiently large to compute the addition $t+u$ by referring to the operation table in h_n. Hence $\mathsf{Add}(t, u, v)$ exactly computes $t + u = v$. The ideas are similar for $t \times u$ and $t \leq u$.

Lemma 4.11. *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,

$$\sigma(t) = \lfloor i/(n+1) \rfloor,$$
$$\sigma(u) = i \bmod (n+1).$$

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.

In (1), since n is greater than or equal to the arguments t, u, the heap h_n is sufficiently large to compute the addition $t + u$ by referring to the operation table in h_n. Hence $\mathsf{Add}(t, u, v)$ in the left-hand side exactly computes $t + u = v$. The right-hand side takes all heaps h. When the heap h is not sufficiently large, the right-hand side becomes true. When the heap h is sufficiently large, the right-hand side exactly computes $t + u = v$. For this reason the equivalence holds. The ideas are similar in (2) and (3).

Lemma 4.12. *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 4.11 (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

$$\sigma[a := p], h \models (a \hookrightarrow 0, [t], [u]) \to s^3(a) \hookrightarrow [v] \quad \text{for all } p.$$

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 4.11 (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 4.11 already proved this statement to atomic formulas. The next lemma is proved by extending it to a normal formula.

Lemma 4.13. *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 4.11 (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 4.10, $\sigma, h_n \models \neg\mathsf{Ineq}(u, t) \vee t = u$ 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,

$$\sigma[x := k] \models x \leq t \text{ and } \sigma[x := k] \models B.$$

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 4.11 (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,

$$\sigma[x := k], h_n \models \neg\mathsf{Ineq}(t, t) \vee (\mathsf{Ineq}(x, t) \wedge B^\circ).$$

Thus, we have

$$\sigma, h_n \models \neg\mathsf{Ineq}(t, t) \vee \exists x(\mathsf{Ineq}(x, t) \wedge B^\circ).$$

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 4.11 (3), we have $k \leq \sigma(t)$. Then, since $k \leq \sigma(t) \leq n$, by the induction hypothesis for B,

$$\sigma[x := k] \models x \leq t \wedge B.$$

That is,

$$\sigma \models \exists x \leq t.B.$$

Case 3. $A \equiv \forall x \leq t.B$. We will show the claim: For all k,

$$\sigma[x := k] \models \neg(x \leq t) \vee B \text{ if and only if } \sigma[x := k], h_n \models \neg\mathsf{Ineq}(x, t) \vee B^\circ.$$

If $k \leq n$, then by Lemma 4.11 (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

$$\sigma[x := k], h_n \models \neg\mathsf{Ineq}(x, t).$$

Thus, the claim holds, and the original statement follows directly from it.

Case 4. $A \equiv \exists(x = t + u)B$. $\sigma \models \exists(x = t + u)B$ is equivalent to

$$\sigma[x := k] \models x = t + u \text{ and } \sigma[x := k] \models B$$

for some k. Since $n \geq \max\{\sigma(t), \sigma(u)\}$, by Lemma 4.11 (1), $\sigma[x := k] \models x = t + u$ is equivalent to

$$\sigma[x := k], h_n \models \mathsf{Add}(t, u, x).$$

Furthermore, since

$$n \geq \max(\sigma, \exists(x = t + u)B) = \max\{\sigma(t + u), \max(\sigma, B[x := t + u])\}$$

$$\geq \max(\sigma, B[x := t+u]) = \max(\sigma, B[x := \overline{k}]) = \max(\sigma[x := k], B),$$

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 4.11 (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 4.14. *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 case 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 4.10,

$$\sigma, h_n \models H \to \neg\mathsf{Ineq}(u, t) \vee t = u$$

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 4.12 (3), $\sigma, h_n \models \neg\mathsf{Ineq}(u, t)$ is equivalent to

$$\sigma, h \models H \to \neg\mathsf{Ineq}(u, t) \quad \text{for all } h.$$

Clearly, $\sigma, h_n \models t = u$ is equivalent to

$$\sigma, h \models t = u \quad \text{for all } h.$$

Therefore, we have

$$\sigma, h \models (H \to \neg\mathsf{Ineq}(u, t)) \vee t = u \quad \text{for all } h,$$

which is equivalent to

$$\sigma, h \models H \to \neg\mathsf{Ineq}(u, t) \vee t = u \quad \text{for all } h.$$

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

$$\sigma, h_n \models H \to \neg\mathsf{Ineq}(t, t) \vee \exists x(\mathsf{Ineq}(x, t) \wedge B^\circ).$$

Since $\sigma, h_n \models H$ and $\sigma, h_n \models \mathsf{Ineq}(t, t)$, we have

$$\sigma, h_n \models \exists x(\mathsf{Ineq}(x, t) \wedge B^\circ),$$

that is, for some k

$$\sigma[x := k], h_n \models \mathsf{Ineq}(x, t) \wedge B^\circ. \tag{a}$$

We fix h in order to show

$$\sigma, h \models H \to \neg\mathsf{Ineq}(t, t) \vee \exists x(\mathsf{Ineq}(x, t) \wedge B^\circ).$$

Assume $\sigma, h \models H$. If $\sigma, h \models \neg\mathsf{Ineq}(t,t)$, the claim trivially holds. Consider the case $\sigma, h \models \mathsf{Ineq}(t,t)$. By (a), we have $\sigma[x := k], h_n \models \mathsf{Ineq}(x,t)$ for some k. Thus, by Lemma 4.11 (3), $k \leq \sigma(t)$. By the case condition, $\sigma, h \models \mathsf{Ineq}(t,t)$. Then, by Lemma 4.3 (4), we have

$$\sigma[x := k], h \models \mathsf{Ineq}(x,t).$$

Moreover, since $n \geq \max(\sigma[x := k], B)$, by induction hypothesis, we have

$$\sigma[x := k], h' \models B^\circ \quad \text{for all } h'.$$

Therefore, we have

$$\sigma[x := k], h \models B^\circ.$$

Thus, we have

$$\sigma[x := k], h \models \mathsf{Ineq}(x,t) \wedge B^\circ,$$

that is,

$$\sigma, h \models \exists x(\mathsf{Ineq}(x,t) \wedge B^\circ).$$

Case 3. $A \equiv \forall x \leq t.B$. Suppose

$$\sigma, h_n \models H \to \forall x(\neg\mathsf{Ineq}(x,t) \vee B^\circ).$$

Since $\sigma, h_n \models H$, we have

$$\sigma, h_n \models \forall x(\neg\mathsf{Ineq}(x,t) \vee B^\circ).$$

We fix h in order to show

$$\sigma, h \models H \to \forall x(\neg\mathsf{Ineq}(x,t) \vee B^\circ).$$

Assume $\sigma, h \models H$. We fix k in order to show

$$\sigma[x := k], h \models \neg\mathsf{Ineq}(x,t) \vee B^\circ.$$

We consider the cases for $\sigma[x := k], h \models \mathsf{Ineq}(x,t)$ and $\sigma[x := k], h \models \neg\mathsf{Ineq}(x,t)$ separately.

Case 3.1. The case $\sigma[x := k], h \models \mathsf{Ineq}(x,t)$. Then, there is p such that

$$\begin{aligned} h(p) &= 2, \\ h(p+1) &= \sigma[x := k](x) + 3 = k + 3, \\ h(p+2) &= \sigma[x := k](t) + 3 = \sigma(t) + 3. \end{aligned}$$

By Lemma 4.3 (3), we have $k \leq \sigma(t)$. Hence, by Lemma 4.11 (3), we have

$$\sigma[x := k], h_n \models \mathsf{Ineq}(x,t).$$

Then, $\sigma[x := k], h_n \models B^\circ$ must be the case. Since $k \leq \sigma(t) \leq n$, we apply the induction hypothesis to B and obtain

$$\sigma[x := k], h' \models B^\circ \quad \text{for all } h'.$$

Hence, we have

$$\sigma[x := k], h \models B^\circ.$$

Then, we have the desired result

$$\sigma[x := k], h \models \neg\mathsf{Ineq}(x,t) \vee B^\circ.$$

Case 3.2. If $\sigma[x := k], h \models \neg\mathsf{Ineq}(x,t)$, then

$$\sigma[x := k], h \models \neg\mathsf{Ineq}(x,t) \vee B^\circ$$

trivially holds.

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

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

$$\sigma[x := k], h_n \models \mathsf{Add}(t, u, x) \text{ and } \sigma[x := k], h_n \models B^\circ.$$

By Lemma 4.12 (1), $\sigma[x := k], h_n \models \mathsf{Add}(t, u, x)$ is equivalent to

$$\sigma[x := k], h' \models H \to \mathsf{Add}(t, u, x) \quad \text{for all } h'.$$

Since we assumed $\sigma, h \models H$, we have

$$\sigma[x := k], h \models \mathsf{Add}(t, u, x).$$

Moreover, since $n \geq \max(\sigma[x := k], B)$, by induction hypothesis for B, we have

$$\sigma[x := k], h' \models B^\circ \quad \text{for all } h'.$$

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 4.12 (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 4.15. *If A is a normal formula in* PA*, $\sigma \models \forall x A$ if and only if $\sigma, h \models (\forall x A)^\circ$ for all h.*

Proof. $\sigma \models \forall x A$ is equivalent to

$$\sigma[x := k] \models A \quad \text{for all } k \in \mathbb{N}.$$

We fix k. Let $n \geq \max(\sigma[x := k], A)$. By Lemma 4.13, $\sigma[x := k] \models A$ is equivalent to

$$\sigma[x := k], h_n \models A^\circ.$$

Then, by Lemma 4.14, this is equivalent to $\sigma[x := k], h \models A^\circ$ for all h. Therefore, $\sigma[x := k] \models A$ is equivalent to

$$\sigma[x := k], h \models A^\circ \quad \text{for all } h.$$

Hence, $\sigma[x := k] \models A$ for all k is equivalent to

$$\sigma[x := k], h \models A^\circ \quad \text{for all } h \text{ for all } k.$$

Thus, $\sigma \models \forall x A$ is equivalent to

$$\sigma, h \models \forall x A^\circ \quad \text{for all } h,$$

that is,

$$\sigma, h \models (\forall x A)^\circ \quad \text{for all } h. \qquad \square$$

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.

The next proposition says that for a given bounded formula in PA we get some equivalent normal formula. To prove the next proposition, we will translate a given formula by replacing some $u + v$ or $u \times v$ by a fresh variable z and adding $\exists(z = u + v)$ or $\exists(z = u \times v)$ so that $+$ and $\times$ appear only in the form of $\exists(z = u + v)$ or $\exists(z = u \times v)$. To get this, we take an innermost occurrence of $u + v$ or $u \times v$. Moreover, to avoid overlapping, we take a leftmost occurrence of them.

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 each occurrence of

$$\neg(t \le u)$$

by

$$u \le t \wedge u \ne t$$

to obtain

$$A' \equiv \overrightarrow{Qx \le t}.C,$$

where C is a quantifier-free disjunctive normal form without formulas of the form $\neg(t \le u)$. 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 variable z. Define

$$\overrightarrow{Qx' \le t'}.D$$

by

$$A'[z] \equiv \overrightarrow{Qx' \le t'}.D$$

where $\overrightarrow{Qx' \le t'}$ is the longest prefix such that z is not in t', namely, it has the longest $\overrightarrow{Qx' \le t'}$ among such $\overrightarrow{Qx' \le t'}$'s. We transform D into

$$\exists(z = u + v)D \text{ or } \exists(z = u \times v)D.$$

We repeat this process until we have the form

$$\overrightarrow{\{Qx \le y, \exists(x = t)\}}A'',$$

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'' does not contain $+, \times$ and formulas of the form $\neg(t \le u)$. Define B as this result. $\square$

The prenexing and disjunctive normal form steps used to obtain B from a bounded PA formula A can incur an exponential blow-up in $|A|$. This does not affect our expressivity and undecidability results. For the Π^0_1 completeness statement proved in Section 7, the upper bound argument is independent of this blow-up. It proceeds by model-checking and arithmetical coding rather than relying on a size-efficient translation.

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 5.3. For a formula

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

its translation $A^\square$ is

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

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 5.4. *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 4.15. $\square$

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

Corollary 5.5. *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. [Sho67]). Thus, $\neg P$ is Π^0_1. By Theorem 5.4, $\neg P$ is valid in PA if and only if $(\neg P)^{\square\circ}$ is valid in SLN. If validity in SLN were decidable, we could determine whether P is true in the standard model, contradicting the undecidability of the halting problem. Therefore, validity in SLN is undecidable. $\square$

We have just shown that Π^0_1 formulas can be translated in a way that preserves both the validity and the non-validity. One might consider extending the translation $(\cdot)^\circ$ by defining $(\exists x A)^\circ \equiv \exists x A^\circ$. However, this extended translation does not preserve the validity and the non-validity, as demonstrated in the following proposition.

Proposition 5.6. *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

$$\sigma, h \models A^{\Box\circ} \quad \text{for all } \sigma, h$$

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,

$$A^{\Box\circ} \equiv \exists x(H \to \exists z(\mathsf{Add}(x, 0, z) \wedge z \neq x)).$$

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

$$\sigma', h \models \mathsf{Add}(x, 0, z) \wedge z \neq x$$

assuming $\sigma', h \models H$. By choice of n, we have

$$\sigma', h \models \forall a \neg(a \hookrightarrow 0, [\overline{n}], [0]).$$

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. □

For the counterexample $\exists x(x + 0 \neq x)$, when a heap is given, we can take some large argument x for which the heap is not sufficiently large. Then $x + 0$ is not computed by referring to the operation table in the heap. Hence the translation of $z = x + 0$ becomes true. Then by taking some z different from x, the translation of the counterexample becomes true.

6. Another undecidability proof

In this section, we present alternative proof of the undecidability of validity in SLN given in Corollary 5.5, This proof follows an approach similar to that used in [CYO01]. Although simpler than the proof of Theorem 5.4, it does not establish the representation of Peano arithmetic within 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, the set of terms is defined by:

$$t ::= x,$$

and the set of formulas is defined by:

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

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

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

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 $\sigma_0, h \models \exists ax(a \hookrightarrow 0, s^2(x))$, we define a structure $M_h = (U_h, R_h)$ by

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

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

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

The next is a well-known theorem for finite structures [EF95].

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 $\sigma, h \models \exists ax(a \hookrightarrow 0, s^2(x)) \to \bigwedge_{x \in \mathsf{FV}(A)} \exists a(a \hookrightarrow 0, s^2(x)) \to A^\triangle$ 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

$$\sigma_0, h \models \exists ax(a \hookrightarrow 0, s^2(x)),$$

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

$$\sigma_0, h \models \exists ax(a \hookrightarrow 0, s^2(x)),$$

we have p, q such that

$$h(p) = 0, h(p+1) = q + 2,$$

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 5.5. 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, validity in SLN for the standard interpretation is undecidable. □

7. Π^0_1-completeness of validity in SLN

In this section, we will show that the validity problem for SLN is Π^0_1-complete. Our proof strategy involves two key steps: (1) showing that the model-checking problem for SLN formulas is decidable, and (2) encoding the validity of SLN formulas as a Π^0_1 formula using (1).

From Theorem 5.4, the lower bound for the validity problem in SLN follows immediately.

Proposition 7.1. *The validity problem for* SLN *formulas is* Π^0_1*-hard.*

The model-checking problem for SLN formulas is defined as:

Definition 7.2 (Model-checking problem for SLN)**.** The *model-checking problem* for SLN formula A is to decide whether $\sigma, h \models A$ holds for given variable assignment σ and heap h.

We call the arithmetic with only 0 and s *successor arithmetic*.

Our main idea for proving the decidability of the model-checking problem in SLN is to bound the search space for variable values that appear in the intuitionistic points-to operator $\hookrightarrow$. For instance, the formula $x \hookrightarrow t$ is automatically false if x takes a value greater than $\max \mathsf{Dom}(h)$. Similarly, $t \hookrightarrow x$ is automatically false if x exceeds $\max\{h(a) \mid a \in \mathsf{Dom}(h)\}$. This observation allows us to restrict the search space for variables involved in the intuitionistic points-to operator. Consequently, variables requiring unbounded search appear only in equality formulas. In other words, quantified variables occur solely within formulas of successor arithmetic, which is known to be decidable because it is a fragment of Presburger arithmetic (the decidable first-theory of addition) [Pre29].

We now proceed to formalize this idea.

Definition 7.3 (Address-freeness and value-freeness)**.** Let A be a formula of SLN, and x be a variable. We say that x is *address-free* in A if A does not contain any atom of the form $s^n(x) \hookrightarrow t$ within the scope of the quantifier Qx. If every bound variable in A is address-free, then A is called *address-free*.

Similarly, we say that x is *value-free* in A if A does not contain any atom of the form $t \hookrightarrow s^n(x)$ within the scope of Qx. If every bound variable in A is value-free, then A is called *value-free*.

For example, the formula $\forall x(x \hookrightarrow s(y) \vee x = s(z))$ is not address-free, since x appears in an atom of the form $x \hookrightarrow s(y)$ within the scope of its quantifier.

The following lemma states that if a variable is not address-free in a formula, then there exists an equivalent formula in which the variable is address-free.

Lemma 7.4. *Let V be a finite set of variables, σ be a variable assignment, h be a heap, QxA be an SLN formula, and the variables in V are address-free in A. Then, there is a formula A' such that the variables in $V \cup \{x\}$ are address-free in A' and $\sigma, h \models QxA \Leftrightarrow \sigma, h \models A'$.*

Proof. We consider the case $Q = \exists$. Let $M = \max \mathsf{Dom}(h)$. Then,

$$\begin{aligned}
&\sigma, h \models \exists x A \\
\Leftrightarrow &\sigma[x := 0], h \models A \text{ or } \sigma[x := 1], h \models A \text{ or } \ldots \text{ or } \sigma[x := M], h \models A \text{ or} \\
&\quad \sigma[x := M+1], h \models A \text{ or } \sigma[x := M+2], h \models A \text{ or } \ldots.
\end{aligned}$$

Let B be a formula obtained from A by replacing each occurrence of $s^n(x) \hookrightarrow t$ by false. Since $\sigma[x := d], h \not\models s^n(x) \hookrightarrow t$ for $d \geq M+1$, we can replace each occurrence of $s^n(x) \hookrightarrow t$ in A be false for $\sigma[x := d]$. Since the variables in V are address-free in A, the variables in $V \cup \{x\}$ are address-free in B. Then,

$$\begin{aligned}
&\sigma, h \models \exists x A \\
\Leftrightarrow &\sigma[x := 0], h \models A \text{ or } \sigma[x := 1], h \models A \text{ or } \ldots \text{ or } \sigma[x := M], h \models A \text{ or} \\
&\quad \sigma[x := M+1], h \models B \text{ or } \sigma[x := M+2], h \models B \text{ or } \ldots \\
\Leftrightarrow &\sigma[x := 0], h \models A \text{ or } \sigma[x := 1], h \models A \text{ or } \ldots \text{ or } \sigma[x := M], h \models A \text{ or} \\
&\quad \sigma, h \models \forall x \geq M+1.B \\
\Leftrightarrow &\sigma, h \models A[x := 0] \text{ or } \sigma, h \models A[x := \overline{1}] \text{ or } \ldots \text{ or } \sigma, h \models A[x := \overline{M}] \text{ or} \\
&\quad \sigma, h \models \forall x \geq M+1.B \\
\Leftrightarrow &\sigma, h \models A[x := 0] \vee A[x := \overline{1}] \vee \cdots \vee A[x := \overline{M}] \vee \forall x \geq M+1.B
\end{aligned}$$

Clearly, the variables in $V \cup \{x\}$ are address-free in $A[x := 0], \ldots, A[x := \overline{M}]$. Then, we take A' to be $A[x := 0] \vee A[x := \overline{1}] \cdots \vee A[x := \overline{M}] \vee \forall x \geq M+1.B$.

The case $Q = \forall$ can be proved in a similar manner. ◻

Note that, strictly speaking, $\forall x \geq M+1.B$ is not part of the syntax of SLN. However, it can be regarded as an abbreviation of the following formula:

$$\forall x(x = 0 \vee x = \overline{1} \vee \ldots \vee x = \overline{M} \vee B).$$

Example 7.5. Let $\max \mathsf{Dom}(h) = M$. Then, for a formula $\forall x(x \hookrightarrow s(y) \vee x = s(z))$, we have the following equivalent address-free formula:

$$
\begin{aligned}
&\forall x(x \hookrightarrow s(y) \vee x = s(z)) \\
\Leftrightarrow &(0 \hookrightarrow s(y) \vee 0 = s(z)) \wedge (\overline{1} \hookrightarrow s(y) \vee \overline{1} = s(z)) \wedge \ldots \wedge (\overline{M} \hookrightarrow s(y) \vee \overline{M} = s(z)) \\
\wedge &(\overline{M+1} \hookrightarrow s(y) \vee \overline{M+1} = s(z)) \wedge (\overline{M+2} \hookrightarrow s(y) \vee \overline{M+2} = s(z)) \wedge \ldots \\
\Leftrightarrow &(0 \hookrightarrow s(y) \vee 0 = s(z)) \wedge (\overline{1} \hookrightarrow s(y) \vee \overline{1} = s(z)) \wedge \ldots \wedge (\overline{M} \hookrightarrow s(y) \vee \overline{M} = s(z)) \\
\wedge &(\text{false} \vee \overline{M+1} = s(z)) \wedge (\text{false} \vee \overline{M+2} = s(z)) \wedge \ldots \\
\Leftrightarrow &(0 \hookrightarrow s(y) \vee 0 = s(z)) \wedge (\overline{1} \hookrightarrow s(y) \vee \overline{1} = s(z)) \wedge \ldots \wedge (\overline{M} \hookrightarrow s(y) \vee \overline{M} = s(z)) \\
\wedge &(\overline{M+1} = s(z)) \wedge (\overline{M+2} = s(z)) \wedge \ldots \\
\Leftrightarrow &(0 \hookrightarrow s(y) \vee 0 = s(z)) \wedge (\overline{1} \hookrightarrow s(y) \vee \overline{1} = s(z)) \wedge \ldots \wedge (\overline{M} \hookrightarrow s(y) \vee \overline{M} = s(z)) \\
\wedge &\forall x \geq M+1(x = s(z)).
\end{aligned}
$$

The next lemma states that every formula A has an equivalent address-free formula.

Lemma 7.6. *Let σ be a variable assignment, h be a heap, and A be a quantifier-free formula of* SLN*. Then, there is a formula A' such that $\sigma, h \models Q_1x_1 \ldots Q_nx_nA \Leftrightarrow \sigma, h \models A'$, where $x_1, \ldots, x_n$ are address-free in A'.*

Proof. By induction on n. Suppose $n > 0$. By induction hypothesis, there is a formula A'' such that

$$\sigma[x := d], h \models Q_2x_2 \ldots Q_nx_nA \Leftrightarrow \sigma[x := d], h \models A'',$$

where $x_2, \ldots, x_n$ are address-free in A''. Therefore, we have

$$\sigma, h \models Q_1x_1 \ldots Q_nx_nA \Leftrightarrow \sigma, h \models Q_1x_1A''.$$

By Lemma 7.4, there is a formula A' such that

$$\sigma, h \models Q_1x_1 \ldots Q_nx_nA \Leftrightarrow \sigma, h \models A',$$

where $x_1, \ldots, x_n$ are address-free in A'. □

In a similar manner, we obtain a corresponding result for value-free formulas. In this case, we use the bound $M = \max\{h(a) \mid a \in \mathsf{Dom}(h)\}$ when proving the next lemma similar to Lemma 7.6.

Lemma 7.7. *Let σ be a variable assignment, h be a heap, and A be a quantifier-free formula of* SLN*. Then, there is a formula A' such that $\sigma, h \models Q_1x_1 \ldots Q_nx_nA \Leftrightarrow \sigma, h \models A'$, where $x_1, \ldots, x_n$ are value-free in A'.*

Now we will show that the model-checking for SLN is decidable.

Proposition 7.8. *For any variable assignment σ, heap h, and formula A of* SLN*, it is decidable whether $\sigma, h \models A$ holds.*

Proof. Without loss of generality, we can assume that A is a prenex normal form. Let $A \equiv Q_1x_1 \ldots Q_nx_nB$, where B is quantifier-free. By Lemma 7.6, there is an address-free A' such that

$$\sigma, h \models A \Leftrightarrow \sigma, h \models A'.$$

Let B be a prenex normal form equivalent to A'. By applying Lemma 7.7 to B, we have a value-free B' such that

$$\sigma, h \models B \Leftrightarrow \sigma, h \models B',$$

where B' is address-free and value-free. Then, we can decide whether $\sigma, h \models t_1 \hookrightarrow t_2$ for all $t_1 \hookrightarrow t_2$ in B, since t_1 and t_2 are closed. We replace each $t_1 \hookrightarrow t_2$ with true or false according to the validity of the closed formula, and obtain a formula C. Since C is a formula of successor arithmetic, we can decide whether $\sigma, h \models C$. □

For example, the formula $\forall x(x \hookrightarrow s(y) \vee x = s(z))$ has an equivalent address-free (and value-free) formula (here $M = \max \mathsf{Dom}(h)$):

$$\begin{aligned}&(0 \hookrightarrow s(y) \vee 0 = s(z)) \wedge (\overline{1} \hookrightarrow s(y) \vee \overline{1} = s(z)) \wedge \ldots (\overline{M} \hookrightarrow s(y) \vee \overline{M} = s(z)) \\ \wedge &\forall x \geq M+1(x = s(z)).\end{aligned}$$

Given σ and h, we can check whether $h(i) = \sigma(y) + 1$, so we can determine whether $\sigma, h \models \bar{i} \hookrightarrow s(y)$ for $1 \leq i \leq M$. Obviously, $\sigma, h \models \bar{i} = s(y)$ can also be determined. Furthermore, $\sigma, h \models \forall x \geq M+1(x = s(z))$ is decidable, because this is a formula of pure successor arithmetic (clearly, it is false). This way, we can decide whether $\sigma, h \models \forall x(x \hookrightarrow s(y) \vee x = s(z))$.

We are now ready to establish our main result: the validity problem in SLN is Π^0_1-complete.

Proposition 7.9. *The validity problem for* SLN *formulas belongs to the class* Π^0_1.

Proof. Given variable assignment σ, heap h, and SLN formula A, the model-checking problem $\sigma, h \models A$ is decidable by Proposition 7.8. Note that the value of σ on variables that does not appear in A is irrelevant. Thus, σ can be regarded as a finite map. Moreover, h is a finite map. Since both the variable assignment σ (restricted to the finitely many free variables of A) and heap h have finite graphs, we encode them as natural numbers by a standard Gödel coding. One convenient choice is:

$$\begin{aligned}\ulcorner \sigma \urcorner &= \mathsf{Seq}(\langle v_1, \sigma(v_1)\rangle, \ldots, \langle v_i, \sigma(v_i)\rangle),\\ \ulcorner h \urcorner &= \mathsf{Seq}(\langle a_1, h(a_1)\rangle, \ldots, \langle a_j, h(a_j)\rangle)\end{aligned}$$

where $\langle \cdot, \cdot \rangle$ is Cantor's pairing function, the pairs $(v_1, \sigma(v_1)), \ldots, (v_i, \sigma(v_i))$ and $(a_1, h(a_1)), \ldots, (a_j, h(a_j))$ enumerate the finite graphs of σ and h, respectively, and $\mathsf{Seq}(b_1, \ldots, b_k)$ denotes the standard sequence encoding (for a precise definition, see, for example, [Sho67, Chapter 6]). Therefore, we can express the relation $\sigma, h \models A$ as a decidable arithmetical predicate. Let the predicate be $R(\sigma, h, A)$. Then, the validity can be expressed by

$$\forall \sigma \forall h R(\sigma, h, A),$$

which is a Π^0_1 formula. □

Theorem 7.10. *The validity problem for* SLN *formulas is* Π^0_1*-complete.*

Proof. By Proposition 7.1 and Proposition 7.9. □

8. Conclusion

In this paper, we have shown that a minimal fragment of separation logic—comprising only the intuitionistic points-to predicate $\hookrightarrow$, the constant 0, and the successor function—is sufficiently expressive to simulate all Π^0_1 formulas of Peano Arithmetic. Through a carefully constructed translation, we proved that validity in Peano Arithmetic corresponds precisely to validity in this fragment under the standard interpretation. This result establishes the undecidability of validity in the fragment, despite its syntactic simplicity.

We further showed that the validity problem in this fragment is Π^0_1-complete by proving the decidability of model-checking and expressing validity as a Π^0_1 formula. Additionally, we provided an alternative undecidability proof via a reduction from finite model theory, reinforcing the robustness of our main result.

Our findings reveal that even a highly restricted form of separation logic can encode significant arithmetic reasoning, including properties such as consistency and non-termination. This contributes to a deeper understanding of the expressive boundaries of separation logic and its interaction with arithmetic.

Future work includes exploring translations from other logical systems into this minimal fragment, investigating whether further restrictions or alternative arithmetic theories could yield decidable fragments, and examining the implications of our results for automated reasoning and program verification.

Acknowledgment

This work was supported by JSPS KAKENHI Grant Number JP25K14999 and MEXT KAKENHI Grant Number JP25H00446.

References

[AGH+14] Timos Antonopoulos, Nikos Gorogiannis, Christoph Haase, Max I. Kanovich, and Joël Ouaknine. Foundations for Decision Problems in Separation Logic with General Inductive Predicates. In 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.

[BCO04] 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.

[BCO05] 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.

[BDL12] 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.

[BFPG14] James Brotherston, Carsten Fuhs, Juan Antonio Navarro Pérez, and Nikos Gorogiannis. A Decision Procedure for Satisfiability in Separation Logic with Inductive Predicates. In Thomas A. Henzinger and Dale Miller, editors, *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on*

Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14 - 18, 2014, pages 25:1–25:10. ACM, 2014. doi:10.1145/2603088.2603091.

[CDOY11] 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.

[CGH05] 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.

[CHO+11] 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.

[CYO01] Cristiano Calcagno, Hongseok Yang, and Peter W. O'Hearn. Computability and Complexity Results for a Spatial Assertion Language for Data Structures. In *The Second Asian Workshop on Programming Languages and Systems, APLAS'01, Korea Advanced Institute of Science and Technology, Daejeon, Korea, December 17-18, 2001, Proceedings*, pages 289–300, 2001. doi:10.1007/3-540-45294-X_10.

[EF95] Heinz-Dieter Ebbinghaus and Jörg Flum. *Finite Model Theory*. Springer-Verlag, 1995.

[EP23] Mnacho Echenim and Nicolas Peltier. An undecidability result for Separation Logic with theory reasoning. *Inf. Process. Lett.*, 182:106359, 2023. doi:10.1016/J.IPL.2023.106359.

[IRS13] 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.

[IT24] Sohei Ito and Makoto Tatsuta. Representation of Peano Arithmetic in Separation Logic. In Jakob Rehof, editor, *9th International Conference on Formal Structures for Computation and Deduction, FSCD 2024, July 10-13, 2024, Tallinn, Estonia*, volume 299 of *LIPIcs*, pages 18:1–18:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.FSCD.2024.18.

[KT21] Daisuke Kimura and Makoto Tatsuta. Decidability for Entailments of Symbolic Heaps with Arrays. *Log. Methods Comput. Sci.*, 17(2), 2021. doi:10.23638/LMCS-17(2:15)2021.

[KZ20] Jens Katelaan and Florian Zuleger. Beyond Symbolic Heaps: Deciding Separation Logic With Inductive Definitions. In Elvira Albert and Laura Kovács, editors, *LPAR 2020: 23rd International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Alicante, Spain, May 22-27, 2020*, volume 73 of *EPiC Series in Computing*, pages 390–408. EasyChair, 2020. doi:10.29007/VKMJ.

[LTSC17] Quang Loc Le, Makoto Tatsuta, Jun Sun, and Wei-Ngan Chin. A Decidable Fragment in Separation Logic with Inductive Predicates and Arithmetic. In Rupak Majumdar and Viktor Kuncak, editors, *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part II*, volume 10427 of *Lecture Notes in Computer Science*, pages 495–517. Springer, 2017. doi:10.1007/978-3-319-63390-9_26.

[NTKY20] 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.

[O'H19] Peter W. O'Hearn. Separation logic. *Commun. ACM*, 62(2):86–95, 2019. doi:10.1145/3211968.

[Pre29] Mojżesz Presburger. Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen. In *welchen die Addition als einzige Operation hervortritt in Comptes-Rendus du ler Congress des Mathematiciens des Pays Slavs*, pages 92–101, 1929. URL: https://cir.nii.ac.jp/crid/1570009750729585920.

[Sho67] Joseph R. Shoenfield. *Mathematical Logic*. Addison-Wesley, 1967.

[TK15] 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.

[TLC16] 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.

[TNK19] 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.