Deciding Separation Logic with Pointer Arithmetic and Inductive Definitions

Wanyun Su[2], Zhilin Wu[2], and Mihaela Sighireanu[1]

[1] State Key Laboratory of Computer Science,
Institute of Software, Chinese Academy of Sciences, China
[2] LMF, ENS Paris-Saclay, University Paris-Saclay and CNRS, France

Abstract. Pointer arithmetic is widely used in low-level programs, e.g. memory allocators. The specification of such programs usually requires using pointer arithmetic inside inductive definitions to define the common data structures, e.g. heap lists in memory allocators. In this work, we investigate decision problems for SLAH, a separation logic fragment that allows pointer arithmetic inside inductive definitions, thus enabling specification of properties for programs manipulating heap lists. Pointer arithmetic inside inductive definitions is challenging for automated reasoning. We tackle this challenge and achieve decision procedures for both satisfiability and entailment of SLAH formulas. The crux of our decision procedure for satisfiability is to compute summaries of inductive definitions. We show that although the summary is naturally expressed as an existentially quantified non-linear arithmetic formula, it can actually be transformed into an equivalent linear arithmetic formula. The decision procedure for entailment, on the other hand, has to match and split the spatial atoms according to the arithmetic relation between address variables. We report on the implementation of these decision procedures and their good performance in solving problems issued from the verification of building block programs used in memory allocators.

1 Introduction

Context. Separation Logic (SL, [23,22]), an extension of Hoare logic, is a well-established formalism for the verification of heap manipulating programs. SL features a *separating conjunction operator* and *inductive predicates*, which allow to express how data structures are laid out in memory in an abstract way. Since its introduction, various verification tools based on separation logic have been developed. A notable one among them is the INFER tool [7], which was acquired by Facebook in 2013 and has been actively used in its development process [8].

Decision procedures for separation logic formulas are vital for the automation of the verification process. These decision procedures mostly focused on separation logic fragments called *symbolic heaps* (SH) [4], since they provide a good compromise between expressivity and tractability. The SH fragments comprise existentially quantified formulas that are conjunctions of atoms encoding aliasing constraints $x = y$ and $x \neq y$ between pointers x and y, points-to constraints

$x \mapsto v$ expressing that at the address x is stored the value v, and predicate atoms $P(\vec{x})$ defining unbounded memory regions of a particular structure. The points-to and predicate atoms, also called spatial atoms, are composed using the separating conjunction to specify the disjointness of memory blocks they specify.

Let us briefly summarize the results on the SH fragments in the sequel. For the SH fragment with the singly linked list-segment predicate, arguably the simplest SH fragment, its satisfiability and entailment problems have been shown to be in PTIME [11] and efficient solvers have been developed for it [2]. The situation changes for more complex inductive predicates: The satisfiability problem for the SH fragment with a class of general inductive predicates was shown to be EXPTIME-complete [5]. On the other hand, the entailment problem for the SH fragment with slightly less general inductive predicates was shown to be 2-EXPTIME-complete [16,12].

Motivation. Vast majority of the work on the verification of heap manipulating programs based on SL assumes that the addresses are *nominal*, that is, they can be compared with only equality or disequality, but not ordered or obtained by arithmetic operations. However, pointer arithmetic is widely used in low-level programs to access data inside memory blocks. Memory allocators are such low-level programs. They assume that the memory is organized into a linked list of memory chunks, called heap lists in this paper; pointer arithmetic is used to jump from a memory chunk to the next one [19,24]. There have been some work to use separation logic for the static analysis and deductive verification of these low-level programs [9,21,10]. Moreover, researchers have also started to investigate the decision procedures for separation logic fragments containing pointer arithmetic. For instance, *Array separation logic* (ASL) was proposed in [6], which includes pointer arithmetic, the constraints $\texttt{blk}(x, y)$ denoting a block of memory units from the address x to y, as well as the points-to constraints $x \mapsto v$. It was shown in [6] that for ASL, the satisfiability is NP-complete and the entailment is in coNEXP resp. coNP for quantified resp. quantifier-free formulas. Furthermore, the decidability can be preserved even if ASL is extended with the list-segment predicate [18]. Very recently, Le identified in [20] two fragments of ASL extended with a class of general inductive predicates for which the satisfiability (but not entailment) problem is decidable.

Nevertheless, none of the aforementioned work is capable of reasoning about heap lists, or generally speaking, pointer arithmetic inside inductive definitions, in a *sound and complete* way. The state-of-the-art static analysis and verification tools, e.g. [9,10,21], resort to sound (but incomplete) heuristics or interactive theorem provers, for reasoning about heap lists. On the other hand, the decision procedures for ASL or its extensions, e.g. [6,18,20], are unable to tackle heap lists. This motivates us to raise the following research question: *Can decision procedures be achieved for separation logic fragments allowing pointer arithmetic inside inductive definitions*?

Contribution. In this work, we propose decision procedures for a fragment of separation logic called SLAH, which allows pointer arithmetic inside inductive definitions, so that inductive predicates specifying heap lists can be defined.

We consider both satisfiability and entailment problems and show that they are NP-complete and coNP-complete respectively. The decision procedure for satisfiability is obtained by computing an equi-satisfiable abstraction in Presburger arithmetic, whose crux is to show that the summaries of the heap list predicates, which are naturally formalized as existentially quantified non-linear arithmetic formulas, can actually be transformed into Presburger arithmetic formulas. The decision procedure for entailment, on the other hand, reduces the problem to multiple instances of an ordered entailment problem, where all the address terms of spatial atoms are ordered. The ordered entailment problem is then decided by matching each spatial atom in the consequent to some spatial formula obtained from the antecedent by partitioning and splitting the spatial atoms according to the arithmetic relations between address variables. Splitting a spatial atom into multiple ones in the antecedent is attributed to pointer arithmetic and unnecessary for SH fragments with nominal addresses.

We implemented the decision procedures on top of CompSPEN solver [14]. We evaluate the performance of the new solver, called CompSPEN$^+$ [1], on a set of formulas originating from path conditions and verification conditions of programs working on heap lists in memory allocators. We also randomly generate some formulas, in order to test the scalability of CompSPEN$^+$ further. The experimental results show that CompSPEN$^+$ is able to solve the satisfiability and entailment problems for SLAH efficiently (in average, less than 1 second for satisfiability and less than 15 seconds for entailment).

To the best of our knowledge, this work presents the first decision procedure and automated solver for decision problems in a separation logic fragment allowing both pointer arithmetic and memory blocks inside inductive definitions.

Organization. A motivating example and an overview of the decision procedures are provided in Section 2. The logic SLAH is defined in Section 3. Then the decision procedures for the satisfiability and entailment problems are presented in Sections 4 resp. 5. The implementation details and the experimental evaluation are reported in Section 6.

2 Motivating example and overview

This section illustrates the use of the logic SLAH for the specification of programs manipulating heap lists and gives an overview of the ingredients used by the decision procedures we propose.

Figure 1 presents the motivating example. The function `search` scans a heap list between the addresses `hbeg` (included) and `hend` (excluded) to find a chunk of size greater than the one given as parameter. A heap list is a block of memory divided into *chunks*, such that each chunk stores its size at its start address. For readability, we consider that the size counts the number of machine integers (and not bytes). Scanning the list requires pointer arithmetics to compute the start address of the next chunk. After the size, the chunk may include additional information about the chunk in so-called *chunk's header*. For readability, we consider that the header contains only the size information. The data carried

```
1  int* hbeg; // heap start
2  int* hend; // heap end
3  int* search(int rsz) {
4    int* t = hbeg;
5    while (t < hend) {
6      int sz = *t;
7      if (rsz ≤ sz)
8        return t;
9      t = t+sz;
10   }
11   return NULL;
12 }
```

$$\texttt{pre} \triangleq 0 < b < e : \texttt{hls}(b, e) \quad (1)$$

$$\texttt{path}_{4\text{-}9} \triangleq 0 < b < e \wedge b = t \wedge sz_0 < rsz \quad (2)$$
$$\wedge\ sz_0 = t_1 - t \wedge 2 \leq t_1 - t$$
$$:\ t \mapsto sz_0 * \texttt{blk}(t+1, t_1) * \texttt{hls}(t_1, e)$$

$$I \triangleq 0 < b < e \wedge b \leq t \quad (3)$$
$$:\ \texttt{hls}(b, t; rsz - 1) * \texttt{hls}(t, e)$$

$$\texttt{post}_{5\text{-}10}(I) \triangleq 0 < b < e \wedge b \leq t_0 < e \quad (4)$$
$$\wedge\ 2 \leq sz_0 < rsz \wedge t - t_0 = sz_0$$
$$:\ \texttt{hls}(b, t_0; rsz - 1) * t_0 \mapsto sz_0$$
$$*\ \texttt{blk}(t_0 + 1, t) * \texttt{hls}(t, e)$$

Fig. 1. Searching a chunk in a heap list and some of its specifications in SLAH

by the chunk, called *chunk's body*, starts after the chunk header and ends before the header of the next chunk.

The right part of Figure 1 provides several formulas in the logic SLAH defined in Section 3. The formula `pre` specifies the precondition of `search`, i.e., there is a heap list from `hbeg` to `hend` represented by the logic variables b resp. e. The pure part of `pre` (left side of ':') is a linear constraint on the addresses. The spatial part of `pre` (right side of ':') employs the predicate `hls` defined inductively by the last two rules below and for which we define the following shorthand:

$$\texttt{hls}(x, y) \equiv \texttt{hls}(x, y; \infty) \text{ i.e., no upper bound on chunks' size, where} \quad (5)$$

$$\texttt{hls}(x, y; v) \Leftarrow x = y : \texttt{emp} \quad (6)$$

$$\texttt{hls}(x, y; v) \Leftarrow \exists z \cdot 2 \leq z - x \leq v : x \mapsto z - x * \texttt{blk}(x+1, z) * \texttt{hls}(z, y; v) \quad (7)$$

The inductive definition states that a heap list from x to y is either an empty heap if x and y are aliased (Eq. (6)), or a *chunk* starting at address x and ending at address z followed by a heap list from z to y. The chunk stores its size $z - x$ in the header (atom $x \mapsto z - x$). The chunk's body is specified by a memory block atom, $\texttt{blk}(x + 1, z)$, starting after the header and ending before the next chunk. The parameter v is an upper bound on the size of all chunks in the list.

The formula $\texttt{path}_{4\text{-}9}$ is generated by the symbolic execution of `search` starting from `pre` and executing the statements from line 4 to 9. It's satisfiability means that the line 9 is reachable from a state satisfying `pre`.

The decision procedure for the satisfiability of SLAH in Section 4 is based on the translation of a SLAH formula φ into an equi-satisfiable Presburger arithmetic (PA) formula φ^P. The delicate point with respect to the previous work, e.g., [18], is to compute a summary in PA for the `hls` atoms. The summary computed for the atom $\texttt{hls}(x, y; v)$ when the heap it denotes is not empty is $(v = 2 \wedge \exists k \cdot k > 0 \wedge 2k = y - x) \vee (2 < v \wedge 2 < y - x)$, i.e., either all chunks have size 2 and the heap-list has an even size or v and the size of the heap-list are strictly greater than 2. For the empty case, the summary is trivially $x = y$. The

other spatial atoms a (e.g., $x \mapsto v$ and $\texttt{blk}(x, y)$) are summarized by constraints on their start address denoted by $\texttt{start}(a)$ (e.g., x) and their end address denoted by $\texttt{end}(a)$ (e.g., $x + 1$ resp. y). For the points-to atom, this constraint is true, but for the $\texttt{blk}(x, y)$ atom, the constraint is $x = \texttt{start}(a) < \texttt{end}(a) = y$. Therefore, the spatial part of $\texttt{path}_{\texttt{4-9}}$ is translated into the PA formula $\texttt{pb}^{\Sigma}_{\texttt{4-9}}$:

$$\underbrace{t + 1 < t_1}_{\texttt{blk}(t+1,t_1)} \wedge \underbrace{(t_1 = e \vee 2 < e - t_1)}_{\texttt{hls}(t_1,e)}.$$

Alone, $\texttt{pb}^{\Sigma}_{\texttt{4-9}}$ does not capture the semantics of the separating conjunction in the spatial part. For that, we add a set of constraints expressing the disjointness of memory blocks occupied by the spatial atoms. For our example, these constraints are $\texttt{pb}^{*}_{\texttt{4-9}} \triangleq t_1 < e \leq t \ \vee \ t + 1 < t_1 \leq e$. By conjoining the pure part of $\texttt{path}_{\texttt{4-9}}$ with formulas $\texttt{pb}^{\Sigma}_{\texttt{4-9}}$ and $\texttt{pb}^{*}_{\texttt{4-9}}$, we obtain an equi-satisfiable existentially quantified PA formula whose satisfiability is a NP-complete problem.

The PA abstraction is also used to decide the validity of entailments in SLAH in combination with a matching procedure between spatial parts. To illustrate this decision procedure presented in Section 5, we consider the verification conditions generated by the proof of the invariant I from Equation (3) for the `search`'s loop. It states that `t` splits the heap list in two parts. To illustrate a non-trivial case of the matching procedure used in the decision procedure for entailment, we consider the verification condition (VC) for the inductiveness of I. The antecedent of the VC is the formula $\texttt{post}_{\texttt{5-10}}(I)$ in Figure 1, obtained by symbolically executing the path including the statements at lines 5–7 and 9 starting from I. The PA abstraction of $\texttt{post}_{\texttt{5-10}}(I)$ is satisfiable and entails the following ordering constraint on the terms used by the spatial atoms: $0 < b \leq t_0 < t_0 + 1 < t \leq e$. The spatial atoms used in the antecedent and consequent are ordered using the order given by this contraint as follows:

$$\begin{array}{lllll} \text{antecedent: } \texttt{hls}(b, t_0; rsz - 1) * & t_0 \mapsto sz_0 * & \texttt{blk}(t_0 + 1, t) * & \texttt{hls}(t, e) \\ \text{consequent: } & \texttt{hls}(b, t; rsz - 1) & & * \texttt{hls}(t, e) \end{array}$$

The matching procedure starts by searching a prefix of the sequence of atoms in the antecedent that matches the first atom in the consequent, $\texttt{hls}(b, t; rsz - 1)$, such that the start and end addresses of the sequence are respectively b and t. The sequence found is $\texttt{hls}(b, t_0; rsz - 1) * t_0 \mapsto sz_0 * \texttt{blk}(t_0 + 1, t)$ which also satisfies the condition (encoded in PA) that it defines a contiguous memory block between b and t. The algorithm continues by trying to prove the matching found using a composition lemma $\texttt{hls}(b, t_0; rsz - 1) * \texttt{hls}(t_0, t; rsz - 1) \models \texttt{hls}(b, t; rsz - 1)$ and the unfolding of the atom $\texttt{hls}(t_0, t; rsz - 1)$. The PA abstraction of the antecedent is used to ensure that sz_0 is the size of the heap list starting at t_0, i.e., $sz_0 = t - t_0$ and the constraint $2 \leq sz_0 \leq rsz - 1$ is satisfied. For this ordering of terms (i.e., $0 < b \leq t_0 < t_0 + 1 < t \leq e$), the algorithm is able to prove the matching. Since this ordering is the only one compatible with the PA abstraction of the antecedent, we conclude that the entailment is valid.

3 SLAH, a separation logic fragment for heap-list

This section defines the syntax and the semantics of the logic SLAH, which extends array separation logic (ASL) [6] with the `hls` predicate. Notice that ASL borrows the structure of formulas from the symbolic heap fragment of SL [4] and introduces a new spatial atom for memory blocks.

Definition 1 (SLAH Syntax). *Let $\mathcal{V}$ denote an infinite set of address variables ranging over $\mathbb{N}$, the set of natural numbers. The syntax of terms t, pure formulas Π, spatial formulas Σ and symbolic heaps φ is given by the following grammar:*

$$
\begin{array}{lll}
t & ::= x \mid n \mid t + t & \textit{terms} \\
\Pi & ::= \top \mid \bot \mid t = t \mid t \neq t \mid t \leq t \mid t < t \mid \Pi \wedge \Pi & \textit{pure formulas} \\
\Sigma & ::= \mathtt{emp} \mid t \mapsto t \mid \mathtt{blk}(t,t) \mid \mathtt{hls}(t,t;t^{\infty}) \mid \Sigma * \Sigma & \textit{spatial formulas} \\
\varphi & ::= \exists \vec{z} \cdot \Pi : \Sigma & \textit{formulas}
\end{array}
$$

where x and $\vec{z}$ are variables resp. set of variables from $\mathcal{V}$, $n \in \mathbb{N}$, t^{∞} is either a term or ∞, `hls` is the predicate defined inductively by the rules in Equations (6) and (7), where v is a variable interpreted over $\mathbb{N} \cup \{\infty\}$. An atom $\mathtt{hls}(x, y; \infty)$ is also written $\mathtt{hls}(x, y)$; Whenever one of Π or Σ is empty, we omit the colon. We write $\mathtt{fv}(\varphi)$ for the set of free variables occurring in φ. If $\varphi = \exists \vec{z} \cdot \Pi : \Sigma$, we write $\mathtt{qf}(\varphi)$ for $\Pi : \Sigma$, the quantifier-free part of φ. We define $\mathtt{start}(a)$ and $\mathtt{end}(a)$ where a is a spatial atom as follows:

- *if $a \equiv t_1 \mapsto t_2$ then $\mathtt{start}(a) \triangleq t_1$, $\mathtt{end}(a) \triangleq t_1 + 1$,*
- *if $a \equiv \mathtt{blk}(t_1, t_2)$ then $\mathtt{start}(a) \triangleq t_1$ and $\mathtt{end}(a) \triangleq t_2$,*
- *if $a \equiv \mathtt{hls}(t_1, t_2; t_3)$ then $\mathtt{start}(a) \triangleq t_1$ and $\mathtt{end}(a) \triangleq t_2$.*

For $n, n' \in \mathbb{N}$ such that $n \leq n'$, we use $[n, n']$ to denote the set $\{n, \cdots, n'\}$. Moreover, we use $[n]$ as an abbreviation of $[1, n]$. We interpret the formulas in the classic model of separation logic built from a stack s and a heap h. The stack is a function $s : \mathcal{V} \to \mathbb{N}$. It is extended to terms, $s(t)$, to denote the interpretation of terms in the given stack; $s(t)$ is defined by structural induction on terms: $s(n) = n$, and $s(t + t') = s(t) + s(t')$. We denote $s[x \leftarrow n]$ for a stack defined as s except for the interpretation of x which is n. Notice that ∞ is used only to give up the upper bound on the value of the chunk size in the definition of `hls` (see Equations (6)–(5)). The heap h is a partial function $\mathbb{N} \rightharpoonup \mathbb{N}$. We denote by $\mathtt{dom}(h)$ the domain of a heap h. We use $h_1 \uplus h_2$ to denote the *disjoint* union of h_1 and h_2, that is, $\mathtt{dom}(h_1) \cap \mathtt{dom}(h_2) = \emptyset$, and for $i \in \{1, 2\}$, we have $(h_1 \uplus h_2)(n) = h_i(n)$ if $n \in \mathtt{dom}(h_i)$.

Definition 2 (SLAH Semantics). *The satisfaction relation $s, h \models \varphi$, where s is a stack, h a heap, and φ a SLAH formula, is defined by:*

- *$s, h \models \top$ always and never $s, h \models \bot$,*
- *$s, h \models t_1 \sim t_2$ iff $s(t_1) \sim s(t_2)$, where $\sim \in \{=, \neq, \leq, <\}$,*
- *$s, h \models \Pi_1 \wedge \Pi_2$ iff $s, h \models \Pi_1$ and $s, h \models \Pi_2$,*
- *$s, h \models \mathtt{emp}$ iff $\mathtt{dom}(h) = \emptyset$,*

- $s, h \models t_1 \mapsto t_2$ *iff* $\exists n \in \mathbb{N}$ *s.t.* $s(t_1) = n$, $\mathtt{dom}(h) = \{n\}$, *and* $h(n) = s(t_2)$,
- $s, h \models \mathtt{blk}(t_1, t_2)$ *iff* $\exists n, n' \in \mathbb{N}$ *s.t.* $s(t_1) = n$, $s(t_2) = n'$, $n < n'$, *and* $\mathtt{dom}(h) = [n, n' - 1]$,
- $s, h \models \mathtt{hls}(t_1, t_2; t_3)$ *iff* $\exists k \in \mathbb{N}$ *s.t.* $s, h \models \mathtt{hls}^k(t_1, t_2; t_3)$,
- $s, h \models \mathtt{hls}^0(t_1, t_2; t^\infty)$ *iff* $s, h \models t_1 = t_2 : \mathtt{emp}$,
- $s, h \models \mathtt{hls}^{\ell+1}(t_1, t_2; t^\infty)$ *iff* $s, h \models \exists z \cdot 2 \leq z - t_1 \wedge \Pi' : t_1 \mapsto z - t_1 \ * \ \mathtt{blk}(t_1 + 1, z) * \mathtt{hls}^\ell(z, t_2; t^\infty)$, *where if* $t^\infty \equiv \infty$, *then* $\Pi' \equiv \top$, *otherwise,* $\Pi' \equiv z - t_1 \leq t^\infty$,
- $s, h \models \Sigma_1 * \Sigma_2$ *iff* $\exists h_1, h_2$ *s.t.* $h = h_1 \uplus h_2, s, h_1 \models \Sigma_1$ *and* $s, h_2 \models \Sigma_2$,
- $s, h \models \exists \vec{z} \cdot \Pi : \Sigma$ *iff* $\exists \vec{n} \in \mathbb{N}^{|\vec{z}|}$ *s.t.* $s[\vec{z} \leftarrow \vec{n}], h \models \Pi$ *and* $s[\vec{z} \leftarrow \vec{n}], h \models \Sigma$.

We write $A \models B$ for A and B (sub-)formula in SLAH for A entails B, i.e., that for any model (s, h) such that $s, h \models A$ then $s, h \models B$.

Notice that the semantics of $\mathtt{blk}(x, y)$ differs from the one given in [6] for $\mathtt{array}(x, y)$ because we consider that the location y is the first after the last location in the memory block, as proposed in [9]. Intuitively, an atom $\mathtt{hls}(x, y; v)$ with v a variable defines a heap lists where all chunks have sizes between 2 and the value of v. Notice that if $v < 2$ then the atom $\mathtt{hls}(x, y; v)$ has a model iff $x = y$. With this semantics, the `blk` and `hls` predicates are compositional predicates [13] and therefore they satisfy the following composition lemmas:

$$\mathtt{blk}(x, y) * \mathtt{blk}(y, z) \models \mathtt{blk}(x, z) \tag{8}$$

$$\mathtt{hls}(x, y; v) * \mathtt{hls}(y, z; v) \models \mathtt{hls}(x, z; v) \tag{9}$$

4 Satisfiability problem of SLAH

The satisfiability problem for an SLAH formula φ is to decide whether there is a stack s and a heap h such that $s, h \models \varphi$. In this section, we propose a decision procedure for the satisfiability problem, thus showing that the satisfiability problem is NP-complete.

Theorem 1. *The satisfiability problem of SLAH is NP-complete.*

The NP lower bound follows from that of ASL in [6]. The NP upper bound is achieved by encoding the satisfiability problem of SLAH as that of an existentially quantified Presburger arithmetic formula. The rest of this section is devoted to the proof of the NP upper bound.

Presburger arithmetic (PA) is the first-order theory with equality of the signature $\langle 0, 1, +, <, (\equiv_n)_{n \in \mathbb{N} \setminus \{0\}} \rangle$ interpreted over the domain of naturals $\mathbb{N}$ with '$+$' interpreted as the addition, '$<$' interpreted as the order relation, and $\equiv_n$ interpreted as the congruence relation modulo n.[3] PA is a useful tool for showing complexity classes because its satisfiability problem belongs to various complexity classes depending on the number of quantifier alternations [15]. In this

[3] Although '$<$' may be encoded using existential quantification in PA over naturals, we prefer to keep it in the signature of PA to obtain quantifier free formulas.

paper, we consider quantifier-free PA formulas (abbreviated as QFPA) and the Σ_1-fragment of PA (abbreviated as EPA), which contains existentially quantified Presburger arithmetic formulas. We recall that the satisfiability problem of QFPA and EPA is NP-complete.

We basically follow the same idea as ASL to build a QFPA abstraction of a SLAH formula φ, denoted by $\texttt{Abs}(\varphi)$, that encodes its satisfiability:

- At first, points-to atoms $t_1 \mapsto t_2$ are transformed into $\texttt{blk}(t_1, t_1 + 1)$.
- Then, the block atoms $\texttt{blk}(t_1, t_2)$ are encoded by the constraint $t_1 < t_2$.
- The predicate atoms $\texttt{hls}(t_1, t_2; t_3)$, absent in ASL, are encoded by a formula in QFPA, $t_1 = t_2 \vee (t_1 < t_2 \wedge \texttt{Abs}^+(\texttt{hls}(t_1, t_2; t_3)))$.
- Lastly, the separating conjunction is encoded by an QFPA formula constraining the address terms of spatial atoms.

The Appendix A provides more details. The crux of this encoding and its originality with respect to the ones proposed for ASL in [6] is the computation of $\texttt{Abs}^+(\texttt{hls}(t_1, t_2; t_3))$, which are the least-fixed-point summaries in QFPA for $\texttt{hls}(t_1, t_2; t_3)$. In the sequel, we show how to compute them.

Intuitively, the abstraction of the predicate atoms $\texttt{hls}(t_1, t_2; t_3)$ shall summarize the relation between t_1, t_2 and t_3 for all $k \geq 1$ unfoldings of the predicate atom. From the fact that the pure constraint in the inductive rule of `hls` is $2 \leq x' - x \leq v$, it is easy to observe that for each $k \geq 1$, $\texttt{hls}^k(t_1, t_2; t_3)$ can be encoded by $2k \leq t_2 - t_1 \leq kt_3$. It follows that $\texttt{hls}(t_1, t_2; t_3)$ can be encoded by $\exists k.\ k \geq 1 \wedge 2k \leq t_2 - t_1 \leq kt_3$. If $t_3 \equiv \infty$, then $\exists k.\ k \geq 1 \wedge 2k \leq t_2 - t_1 \leq kt_3$ is equivalent to $\exists k.\ k \geq 1 \wedge 2k \leq t_2 - t_1 \equiv 2 \leq t_2 - t_1$, thus a QFPA formula. Otherwise, $2k \leq t_2 - t_1 \leq kt_3$ is a non-linear formula since kt_3 is a non-linear term if t_3 contains variables. The following lemma states that $\exists k.\ k \geq 1 \wedge 2k \leq t_2 - t_1 \leq kt_3$ can actually be turned into an equivalent QFPA formula.

Lemma 1 (Summary of `hls` atoms). *Let $\texttt{hls}(x, y; z)$ be an atom in SLAH representing a non-empty heap, where x, y, z are three distinct variables in $\mathcal{V}$. We can construct in polynomial time an QFPA formula $\texttt{Abs}^+(\texttt{hls}(x, y; z))$ which summarizes $\texttt{hls}(x, y; z)$, namely we have for every stack s, $s \models \texttt{Abs}^+(\texttt{hls}(x, y; z))$ iff there exists a heap h such that $s, h \models \texttt{hls}(x, y, z)$.*

Since the satisfiability problem of QFPA is NP-complete, the satisfiability problem of SLAH is in NP. The correctness of $\texttt{Abs}(\varphi)$ is guaranteed by the following result.

Proposition 1. *A SLAH formula φ is satisfiable iff $\texttt{Abs}(\varphi)$ is satisfiable.*

From now on, we shall assume that $\texttt{Abs}(\varphi)$ **is a QFPA formula**. This enables using the off-the-shelf SMT solvers, e.g. Z3, to solve the satisfiability problem of SLAH formulas.

5 Entailment problem of SLAH

We consider the following entailment problem: Given the symbolic heaps φ and ψ in SLAH such that ψ is quantifier free and $\texttt{fv}(\psi) \subseteq \texttt{fv}(\varphi)$, decide if $\varphi \models \psi$. Notice that the existential quantifiers in φ, if there is any, can be dropped.

Our goal in this section is to show that the entailment problem is decidable, as stated in the following theorem.

Theorem 2. *The entailment problem of SLAH is coNP-complete.*

The coNP lower bound follows from the fact that the entailment problem of quantifier-free ASL formulas is also coNP-complete [6]. The remainder of this section is devoted to the proof of the coNP upper bound.

In a nutshell, we show in Section 5.1 that the entailment problem $\varphi \models \psi$ can be decomposed into a finite number of *ordered entailment problems* $\varphi' \models_{\preceq} \psi'$ where all the terms used as start and end addresses of spatial atoms in φ' and ψ' are ordered by a preorder $\preceq$. Then, we propose a decision procedure to solve ordered entailment problems. In Section 5.2, we consider the special case where the consequent ψ' has a unique spatial atom; this part reveals a delicate point which appears when the consequent and the antecedent are `hls` atoms because of the constraint on the chunk sizes. The general case is dealt with in Section 5.3; the procedure calls the special case for the first atom of the consequent with all the compatible prefixes of the antecedent, and it does a recursive call for the remainders of the consequent and the antecedent. Note that to find all the compatible prefixes of the antecedent, some spatial atoms in the antecedent might be split into several ones. We derive the coNP upper bound from the aforementioned decision procedure as follows:

1. The entailment problem is reduced to at most exponentially many *ordered entailment problems* since there are exponentially many total preorders.
2. Each ordered entailment problem can be reduced further to exponentially many *special ordered entailment problems* where there is one spatial atom in the consequent.
3. The original entailment problem is invalid iff there is an invalid special ordered entailment problem instance.
4. The special ordered entailment problem is in coNP.

In the sequel, we assume that $\texttt{Abs}(\varphi)$ is satisfiable and $\texttt{Abs}(\varphi) \models \texttt{Abs}(\psi)$. Otherwise, the entailment is trivially unsatisfiable.

5.1 Decomposition into ordered entailments

Given the entailment problem $\varphi \models \psi$, we denote by $\mathcal{A}(\varphi)$ (and $\mathcal{A}(\psi)$) the set of terms used as start and end addresses of spatial atoms in φ (resp. ψ). Recall that a *preorder* $\preceq$ over a set A is a reflexive and transitive relation on A. The preorder $\preceq$ on A is *total* if for every $a, b \in A$, either $a \preceq b$ or $b \preceq a$. For $a, b \in A$, we denote by $a \simeq b$ the fact that $a \preceq b$ and $b \preceq a$, and we use $a \prec b$ for $a \preceq b$ but not $b \preceq a$.

Definition 3 (Total preorder compatible with $\texttt{Abs}(\varphi)$). *Let $\preceq$ be a total preorder over $\mathcal{A}(\varphi) \cup \mathcal{A}(\psi)$. Then $\preceq$ is said to be* compatible with φ *if $C_{\preceq} \wedge \texttt{Abs}(\varphi)$ is satisfiable, where*

$$C_{\preceq} \triangleq \bigwedge_{t_1, t_2 \in \mathcal{A}(\varphi) \cup \mathcal{A}(\psi), t_1 \simeq t_2} t_1 = t_2 \wedge \bigwedge_{t_1, t_2 \in \mathcal{A}(\varphi) \cup \mathcal{A}(\psi), t_1 \prec t_2} t_1 < t_2. \quad (10)$$

Example 1. Let $\varphi \equiv \mathtt{blk}(x_1, x_2) * \mathtt{hls}(x_2, x_3; y)$ and $\psi \equiv \mathtt{blk}(x_1, x_3)$. Then $\mathcal{A}(\varphi) \cup \mathcal{A}(\psi) = \{x_1, x_2, x_3\}$. From $\mathtt{Abs}(\varphi) \models x_1 < x_2 \wedge x_2 \leq x_3$, there are two total preorders compatible with φ, namely, $x_1 \prec_1 x_2 \prec_1 x_3$ and $x_1 \prec_2 x_2 \simeq_2 x_3$.

Definition 4 ($\varphi \models_\preceq \psi$). *Let $\preceq$ be a total preorder over $\mathcal{A}(\varphi) \cup \mathcal{A}(\psi)$ that is compatible with φ. Then we say $\varphi \models_\preceq \psi$ if $C_\preceq \wedge \Pi : \Sigma \models \Pi' : \Sigma'$.*

Lemma 2. *$\varphi \models \psi$ iff for every total preorder $\preceq$ over $\mathcal{A}(\varphi) \cup \mathcal{A}(\psi)$ that is compatible with φ, we have $\varphi \models_\preceq \psi$.*

The proof of the above lemma is immediate. There may be exponentially many total preorders over $\mathcal{A}(\varphi) \cup \mathcal{A}(\psi)$ that are compatible with φ in the worst case.

The procedure to decide $\varphi \models_\preceq \psi$ is presented in the rest of this section. We assume that $\varphi \equiv \Pi : a_1 * \cdots * a_m$ and $\psi \equiv \Pi' : b_1 * \cdots * b_n$ such that

$$C_\preceq \wedge \mathtt{Abs}(\varphi) \text{ is satisfiable and } C_\preceq \wedge \mathtt{Abs}(\varphi) \models \mathtt{Abs}(\psi). \tag{11}$$

We consider that the atoms $\mathtt{hls}(t_1, t_2; t_3)$ in φ or ψ such that $C_\preceq \models t_1 = t_2$ are removed because they correspond to an empty heap. Moreover, after a renaming, we assume that the spatial atoms are sorted in each formula, namely, the following two PA entailments hold:

$$C_\preceq \models \bigwedge_{i \in [1,m]} \mathtt{start}(a_i) < \mathtt{end}(a_i) \wedge \bigwedge_{1 \leq i < m} \mathtt{end}(a_i) \leq \mathtt{start}(a_{i+1}), \tag{12}$$

$$C_\preceq \models \bigwedge_{i \in [1,n]} \mathtt{start}(b_i) < \mathtt{end}(b_i) \wedge \bigwedge_{1 \leq i < n} \mathtt{end}(b_i) \leq \mathtt{start}(b_{i+1}). \tag{13}$$

Section 5.2 considers the special case of a consequent ψ having only one spatial atom. Section 5.3 considers the general case.

5.2 Consequent with one spatial atom

Consider the ordered entailment $\varphi \models_\preceq \psi$, where $\varphi \equiv \Pi : a_1 * \cdots * a_m$, $\psi \equiv \Pi' : b_1$ and the constraints (11)–(13) are satisfied. From (11) and the definition of $\mathtt{Abs}$, we have that $C_\preceq \wedge \mathtt{Abs}(\varphi)$ implies Π', so we simplify this entailment to deciding:

$$C_\preceq \wedge \Pi : a_1 * \cdots * a_m \models_\preceq b_1,$$

where, the atoms a_i ($i \in [m]$) and b_1 represent *non-empty* heaps, and the start and end addresses of atoms a_i as well as those of b_1 are totally ordered by $C_\preceq$.

Because b_1 defines a continuous memory region, the procedure checks the following necessary condition in PA:

$$C_\preceq \models \mathtt{start}(a_1) = \mathtt{start}(b_1) \wedge \mathtt{end}(a_m) = \mathtt{end}(b_1) \wedge \bigwedge_{1 \leq i < m} \mathtt{end}(a_i) = \mathtt{start}(a_{i+1}).$$

Then, the procedure does a case analysis on the form of b_1. If $b_1 \equiv t_1 \mapsto t_2$ then $\varphi \models_\preceq \psi$ holds iff $m = 1$ and $a_1 = t_1' \mapsto t_2'$. If $b_1 \equiv \mathtt{blk}(t_1, t_2)$ then $\varphi \models_\preceq \psi$ holds. For the last case, $b_1 \equiv \mathtt{hls}(t_1, t_2; t_3)$, we distinguish between $m = 1$ or not.

One atom in the antecedent: A case analysis on the form of a_1 follows.
$\boxed{a_1 \equiv t'_1 \mapsto t'_2}$ Then $\varphi \models_{\preceq} \psi$ does not hold, since a nonempty heap modeling b_1 has to contain at least two memory cells.
$\boxed{a_1 \equiv \mathtt{blk}(t'_1, t'_2)}$ Then the entailment $\varphi \models_{\preceq} \psi$ does not hold because a memory block of size $t'_2 - t'_1$ where the first memory cell stores the value 1 satisfies $\mathtt{blk}(t'_1, t'_2)$ but does not satisfy $b_1 \equiv \mathtt{hls}(t_1, t_2; t_3)$ where, by the inductive rule of $\mathtt{hls}$, $t_1 \mapsto z - t_1$ and $2 \leq z - t_1 \leq t_3$.
$\boxed{a_1 \equiv \mathtt{hls}(t'_1, t'_2; t'_3)}$ Then the entailment problem seems easy to solve. One may conjecture that $C_{\preceq} \wedge \Pi : \mathtt{hls}(t'_1, t'_2; t'_3) \models \mathtt{hls}(t_1, t_2; t_3)$ iff $C_{\preceq} \wedge \mathtt{Abs}(\varphi) \models t'_3 \leq t_3$, which is *not* the case as a matter of fact, as illustrated by the following example. (Recall that, from (10) we have that $C_{\preceq} \wedge \mathtt{Abs}(\varphi) \models t'_1 = t_1 \wedge t'_2 = t_2$.)

Example 2. Consider $x < y \wedge y - x = 4 : \mathtt{hls}(x, y; 3) \models \mathtt{hls}(x, y; 2)$. The entailment is valid, while we have $3 > 2$. The reason behind this seemly counterintuitive fact is that when we unfold $\mathtt{hls}(x, y; 3)$ to meet the constraint $y - x = 4$, it is impossible to have a memory chunk of size 3. (Actually every memory chunk is of size 2 during the unfolding.)

We are going to show how to tackle this issue in the sequel.

Definition 5 (Unfolding scheme of a predicate atom and effective upper bound). *Let $\varphi \equiv \Pi : \mathtt{hls}(t'_1, t'_2; t'_3)$ be an SLAH formula and $s : \mathcal{V} \to \mathbb{N}$ be a stack such that $s \models \mathtt{Abs}(\varphi)$ and $s(t'_2) - s(t'_1) \geq 2$. An* unfolding scheme *of φ w.r.t. s is a sequence of numbers $(sz_1, \cdots, sz_\ell)$ such that $2 \leq sz_i \leq s(t'_3)$ for every $i \in [\ell]$ and $s(t'_2) = s(t'_1) + \sum_{i \in [\ell]} sz_i$. Moreover, $\max(sz_1, \cdots, sz_\ell)$ is called the* chunk size upper bound *associated with the unfolding scheme. The* effective upper bound *of φ w.r.t. s, denoted by $\mathtt{EUB}_\varphi(s)$, is defined as the maximum chunk size upper bound associated with the unfolding schemes of φ w.r.t. s.*

Example 3. Let $\varphi \equiv x < y : \mathtt{hls}(x, y; 3)$ and s be a store such that $s(x) = 1$ and $s(y) = 7$. Then there are two unfolding schemes of φ w.r.t. s, namely, $(2, 2, 2)$ and $(3, 3)$, whose chunk size upper bounds are 2 and 3 respectively. Therefore, $\mathtt{EUB}_\varphi(s)$, the effective upper bound of φ w.r.t. s, is 3.

The following lemma (proved in the appendix) states that the effective upper bounds of chunks in heap lists atoms of φ with respect to stacks can be captured by a QFPA formula.

Lemma 3. *For an SLAH formula $\varphi \equiv \Pi : \mathtt{hls}(t'_1, t'_2; t'_3)$, a QFPA formula $\xi_{eub,\varphi}(z)$ can be constructed in linear time such that for every store s satisfying $s \models \mathtt{Abs}(\varphi)$, we have $s[z \leftarrow \mathtt{EUB}_\varphi(s)] \models \xi_{eub,\varphi}(z)$ and $s[z \leftarrow n] \not\models \xi_{eub,\varphi}(z)$ for all $n \neq \mathtt{EUB}_\varphi(s)$.*

The following lemma (proof in the appendix) provides the correct test used for the case $a_1 \equiv \mathtt{hls}(t'_1, t'_2; t'_3)$.

Lemma 4. *Let $\varphi \equiv \Pi : \mathtt{hls}(t'_1, t'_2; t'_3)$, $\psi \equiv \mathtt{hls}(t_1, t_2; t_3)$, and $\preceq$ be a total preorder over $\mathcal{A}(\varphi) \cup \mathcal{A}(\psi)$ such that $C_{\preceq} \models t'_1 < t'_2 \wedge t'_1 = t_1 \wedge t'_2 = t_2$. Then $\varphi \models_{\preceq} \psi$ iff $C_{\preceq} \wedge \mathtt{Abs}(\varphi) \models \forall z.\ \xi_{eub,\varphi}(z) \to z \leq t_3$.*

From Lemma 4, it follows that $\varphi \equiv \Pi : \mathtt{hls}(t'_1, t'_2; t'_3) \models_{\preceq} \mathtt{hls}(t_1, t_2; t_3)$ is invalid iff $C_{\preceq} \wedge \mathtt{Abs}(\varphi) \wedge \exists z.\ \xi_{eub,\varphi}(z) \wedge \neg z \leq t_3$ is satisfiable, which is an EPA formula. Therefore, this special case of the ordered entailment problem is in coNP.

At least two atoms in the antecedent: Recall that $\varphi \equiv C_{\preceq} \wedge \Pi : a_1 * \cdots * a_m$; a case analysis on the form of the first atom of the antecedent, a_1, follows.

$\boxed{a_1 \equiv \mathtt{blk}(t'_1, t'_2)}$ Then $\varphi \models_{\preceq} \mathtt{hls}(t_1, t_2; t_3)$ does not hold (see case $m = 1$).

$\boxed{a_1 \equiv \mathtt{hls}(t'_1, t'_2; t'_3)}$ Then $\varphi \models_{\preceq} \mathtt{hls}(t_1, t_2; t_3)$ iff $\mathtt{Abs}(\varphi) : \mathtt{hls}(t'_1, t'_2; t'_3) \models_{\preceq} \mathtt{hls}(t_1, t'_2; t_3)$ and $\mathtt{Abs}(\varphi) : a_2 * \cdots * a_m \models_{\preceq} \mathtt{hls}(t'_2, t_2; t_3)$.

$\boxed{a_1 \equiv t'_1 \mapsto t'_2}$ Then the analysis is more involved because we have to check that t'_2 is indeed the size of the first chunk in $\mathtt{hls}(t_1, t_2; t_3)$ (i.e., satisfies $2 \leq t'_2 \leq t_3$) and the address $t'_1 + t'_2$, the end of the chunk starting at $t'_1 = t_1$, is the start of a heap list in the antecedent. The last condition leads to the following cases:

- $t'_1 + t'_2$ is the end of some a_j where $j \in [m]$ such that $t'_1 + t'_2 = \mathtt{end}(a_j) \wedge C_{\preceq} \wedge \mathtt{Abs}(\varphi)$ is satisfiable. Then the following entailment shall hold:

$$2 \leq t'_2 \leq t_3 \wedge t'_1 + t'_2 = \mathtt{end}(a_j) \wedge \mathtt{Abs}(\varphi) : a_{j+1} * \cdots * a_m \models_{\preceq} \mathtt{hls}(t'_1 + t'_2, t_2; t_3).$$

- $t'_1 + t'_2$ is inside a block atom a_j: where $j \in [m]$ such that $a_j \equiv \mathtt{blk}(t''_1, t''_2)$ and $t''_1 < t'_1 + t'_2 < t''_2 \wedge \mathtt{Abs}(\varphi)$ is satisfiable. Then $\varphi \not\models_{\preceq} \psi$ because a block atom cannot match the head of a heap list in the consequent.
- $t'_1 + t'_2$ is inside a heap-list atom a_j where $j \in [m]$ such that $a_j \equiv \mathtt{hls}(t''_1, t''_2; t''_3)$, $t''_1 < t'_1 + t'_2 < t''_2 \wedge \mathtt{Abs}(\varphi)$ is satisfiable. Then the following ordered entailment stating that the suffix of the antecedent starting at $t'_1 + t'_2$ matches the tail of the consequent, shall hold:

$$\begin{array}{l} 2 \leq t'_2 \leq t_3 \wedge t''_1 < t'_1 + t'_2 < t''_2 \wedge \mathtt{Abs}(\varphi) : \\ \qquad \mathtt{hls}(t'_1 + t'_2, t''_2; t''_3) * \ a_{j+1} * \cdots * a_m \models_{\preceq} \mathtt{hls}(t'_1 + t'_2, t_2; t_3) \end{array}$$

  and the following formula, expressing that $t'_1 + t'_2$ is inside a block of a chunk in a_j, shall be unsatisfiable (since otherwise the remaining suffix of the antecedent will start by a block atom and cannot match a heap list):

$$\begin{array}{l} \mathtt{Abs}(\, t''_1 \leq x' < t'_1 + t'_2 < x'' \leq t''_2 \wedge 2 \leq x'' - x' \leq t''_3 \wedge C_{\preceq} \wedge \Pi : \\ \qquad a_1 * \cdots * a_{j-1} * \mathsf{Ufld}_{x',x''}(\mathtt{hls}(t''_1, t''_2; t''_3)) \ * a_{j+1} * \cdots * a_m) \end{array}$$

  where x', x'' are two fresh variables and the formula Ufld specifies a splitting of a_j into a heap list from t''_1 to x', a chunk starting at x' and ending at x'', and a heap list starting at x'':

$$\begin{array}{rl} \mathsf{Ufld}_{x',x''}(\mathtt{hls}(t''_1, t''_2; t''_3)) \triangleq & \mathtt{hls}(t''_1, x'; t''_3) * x' \mapsto x'' - x' * \\ & \mathtt{blk}(x' + 1, x'') * \mathtt{hls}(x'', t''_2; t''_3), \end{array}$$

Notice that all $j \in [m]$ shall be considered above; if one j satisfying the premises does not lead to a valid conclusion then the entailment is not valid.

5.3 Consequent with more spatial atoms

Using the arguments similar to the one given for the case $n = 1$, we simplify $\varphi \models_{\preceq} \psi$ with $\varphi \equiv \Pi : a_1 * \cdots * a_m$ and $\psi \equiv \Pi' : b_1 * \cdots * b_n$ to:

$$C_{\preceq} \wedge \Pi : a_1 * \cdots * a_m \models_{\preceq} b_1 * \cdots * b_n.$$

For $n > 1$, the decision procedure tries all the possible partitions of the sequence $a_1 * \cdots * a_m$ into a prefix $a_1 * \cdots * a'_k$ to be matched by b_1 and a suffix $a''_k * \cdots * a_m$ to be matched by $b_2 * \cdots * b_n$, where a'_k and a''_k are obtained by splitting the atom a_k. The partition process depends on the relative ordering of $\mathtt{end}(b_1)$, $\mathtt{start}(a_k)$ and $\mathtt{end}(a_k)$. Formally, for every $k \in [m]$, the procedure considers all the following cases for which it generates recursive calls to check the entailments:

$$C_{\preceq} \wedge \Pi_k : a_1 * \cdots * a'_k \models_{\preceq} b_1 \text{ and } C_{\preceq} \wedge \Pi_k : a''_k * \cdots * a_m \quad \models_{\preceq} b_2 * \cdots * b_n,$$

where Π_k, a'_k, a''_k are defined as follows.

- If $\mathtt{end}(b_1) = \mathtt{end}(a_k) \wedge C_{\preceq} \wedge \mathtt{Abs}(\varphi)$ is satisfiable, then $a'_k \triangleq a_k$, $a''_k \triangleq \mathtt{emp}$, $\Pi_k \triangleq \mathtt{end}(b_1) = \mathtt{end}(a_k) \wedge \mathtt{Abs}(\varphi)$.
- If $\mathtt{start}(a_k) < \mathtt{end}(b_1) < \mathtt{end}(a_k) \wedge C_{\preceq} \wedge \mathtt{Abs}(\varphi)$ is satisfiable, then a case analysis on the form of a_k is done to apply the suitable composition lemma:
  - If $a_k = \mathtt{blk}(t''_1, t''_2)$, then $a'_k \triangleq \mathtt{blk}(t''_1, \mathtt{end}(b_1))$, $a''_k \triangleq \mathtt{blk}(\mathtt{end}(b_1), t''_2)$, and $\Pi_k \triangleq t''_1 < \mathtt{end}(b_1) < t''_2 \wedge \mathtt{Abs}(\varphi)$.
  - If $a_k = \mathtt{hls}(t''_1, t''_2; t''_3)$, then we distinguish the following cases:
    - $\mathtt{end}(b_1)$ starts a chunk in a_k, that is, $a'_k \triangleq \mathtt{hls}(t''_1, \mathtt{end}(b_1); t''_3)$, $a''_k \triangleq \mathtt{hls}(\mathtt{end}(b_1), t''_2; t''_3)$, and $\Pi_k \triangleq t''_1 < \mathtt{end}(b_1) < t''_2 \wedge \mathtt{Abs}(\varphi)$;
    - $\mathtt{end}(b_1)$ splits the body of a chunk in a_k starting at some (fresh) address x; depending on the position of the chunk in the list (the first, the last, in the middle, or the only chunk in the heap list), we obtain four cases. This case splitting is due to the fact that the ordered entailment problems always assume non-empty `hls` atoms in the formulas. For example, if the chunk starting at x is in the middle of the heap list, then $a'_k \triangleq \mathtt{hls}(t''_1, x; t''_3) * x \mapsto x' - x * \mathtt{blk}(x+1, \mathtt{end}(b_1))$, $a''_k \triangleq \mathtt{blk}(\mathtt{end}(b_1), x') * \mathtt{hls}(x', t''_2; t''_3)$, and $\Pi_k \triangleq t''_1 < x < \mathtt{end}(b_1) < x' < t''_2 \wedge \mathtt{Abs}(\varphi)$, where both x and x' are fresh variables.

6 Implementation and experiments

Implementation. The decision procedures presented are implemented as an extension of the CompSPEN solver [14], called CompSPEN$^+$, available at [1]. Let us briefly recall some information about CompSPEN. CompSPEN is written in C++ and includes several decision procedures for symbolic heap fragments including (i) inductive predicates that are compositional [13] (the predicate $\mathtt{ls}(x, y)$ for list segments is a simple example) and (ii) integer data constraints. It uses SMT solvers (e.g., Z3) for solving linear integer arithmetic constraints.

CompSPEN ranked third among the eleven solvers in the general podium of the last edition of SL-COMP, the competition of separation-logic solvers [2].

CompSPEN$^+$ supports the new theory SLAH. Internally, CompSPEN$^+$ parses the input file which shall include the definition of `hls` and the satisfiability or entailment queries in the SL-COMP format [2].

- For satisfiability queries, it constructs the EPA abstraction of the SLAH formulas as shown in Section 4, and queries an SMT solver on its satisfiability.
- For entailment queries $\varphi \models \psi$, CompSPEN$^+$ has to enumerate all the total preorders over the set of the start and end addresses of spatial atoms in φ and ψ, as described in Section 5. This is time-consuming and a bottleneck for the performance. We introduced some heuristics based on the preprocessing of the formula to extract preorders between addresses or to decompose the entailment on simpler ones (i.e., with less atoms). For instance, we check for every block atom b in ψ, whether there is a collection of spatial atoms, say $a_i * a_{i+1} * \cdots * a_j$ with $i < j$, such that they are contiguous (i.e., the ending address of a_k is the starting address of a_{k+1}), $\texttt{start}(b) = \texttt{start}(a_i)$ and $\texttt{end}(b) = \texttt{end}(a_j)$. If this is the case, then we generate the entailment query $\texttt{Abs}(\varphi) : a_i * a_{i+1} * \cdots * a_j \models b$ and remove all these spatial atoms from φ and ψ, thus reducing the original entailment query to a smaller one, for which the number of addresses for the total preorder enumeration is decreased.

Benchmarks. We generated 190 benchmarks, available at [3], classified into four suites, whose sizes are given in Table 1, as follows:

- MEM-SAT and MEM-ENT are satisfiability resp. entailment problems generated by the verification of programs that are building blocks of heap-list based memory allocators, including: create a heap-list with one element, split a memory chunk into two consecutive memory chunks, join two memory chunks, search a memory chunk of size bigger than a given parameter (our running example), or search an address inside a heap-list.
- RANDOM-SAT and RANDOM-ENT are satisfiability resp. entailment problems which are randomly or manually generated. Starting from the path and verification conditions for programs manipulating heap lists, we replace some `hls` atoms with their unfoldings in order to generate formulas with more spatial atoms. RANDOM-SAT includes also formulas where the atoms and their start and end address terms are generated randomly. In addition, we generate some benchmarks manually. This suite is motivated by testing the scalability of CompSPEN$^+$.

Experiments. We run CompSPEN$^+$over the four benchmark suites, using a Ubuntu-16.04 64-bit lap-top with an Intel Core i5-8250U CPU and 2GB RAM. The experimental results are summarized in Table 1. We set the timeout to 60 seconds. The statistics of average time and maximum time do not include the time of timeout instances. To the best of our knowledge, there have not been solvers that are capable of solving the SLAH formulas that include, points-to, block, and `hls` atoms. The solver SLar [17] was designed to solve entailment problems involving points-to, block, and `ls` atoms. Nevertheless, we are unable

to find a way to access SLar, thus failing to compare with it on ASL formulas. Moreover, the examples used by SLar, available online, are in a format that seems nontrivial to translate into the SL-COMP format.

Table 1. Experimental results, time measured in seconds

Benchmark suite	#instances	Timeout	Avg. time	Min. time	Max. time
MEM-SAT	38	0	0.05	0.03	0.12
RANDOM-SAT	50	0	0.09	0.02	0.53
TOTAL	88	0	0.07	0.02	0.53
MEM-ENT	43	0	3.05	0.34	9.98
RANDOM-ENT	59	2	13.39	0.04	48.85
TOTAL	102	2	8.94	0.04	48.85

As expected, solving entailment instances is more expensive than solving satisfiability instances. We recall from Section 5 that the procedure for entailment queries satisfiability of several formulas. CompSPEN$^+$ efficiently solves the benchmark instances originated from program's verification, namely MEM-SAT and MEM-ENT, with the average time in 0.05 and 3.05 seconds respectively.

Table 1 shows that some entailment instances are challenging for CompSPEN$^+$. For instance, the maximum time in MEM-ENT suite is 9.98, and there are 2 timeout instances in RANDOM-ENT suite (more than 2 min). By inspecting these challenging instances, we found that (i) they require splitting some spatial atoms (`blk` or `hls`) in the antecedent, which is potentially time-consuming, and (ii) they correspond to valid entailment problems where CompSPEN$^+$ has to explore all the total preorders, which is time-consuming. We noticed that when the entailment problem is invalid, the heuristics implemented are able to quickly find some total preorder under which the entailment does not hold.

7 Conclusion

In this work, we investigated SLAH, a separation logic fragment that allows pointer arithmetic inside inductive definitions so that the commonly used data structures e.g. heap lists can be defined. We show that the satisfiability problem of SLAH is NP-complete and the entailment problem is coNP-complete. We implemented the decision procedures in a solver, CompSPEN$^+$, and use it to efficiently solve more than hundred problems issued from verification of program manipulating heap lists or randomly generated problems. For future work, it is interesting to see whether the logic SLAH and its decision procedures can be extended to specify free lists, another common data structure in memory allocators [19] in addition to heap lists. Moreover, since bit operations are also widely used in memory allocators, it would also be interesting to automate the reasoning about bit operations inside inductive definitions.

References

1. The CompSpen solver, https://github.com/suwy123/compspen2
2. SL-COMP website, https://sl-comp.github.io/
3. benchmarks, C.: CompSpen+ benchmarks (2021), https://github.com/suwy123/compspen2/tree/master/samples/PAsamples
4. Berdine, J., Calcagno, C., O'Hearn, P.W.: A decidable fragment of separation logic. In: FSTTCS. LNCS, vol. 3328. Springer (2005)
5. Brotherston, J., Fuhs, C., Pérez, J.A., Gorogiannis, N.: A decision procedure for satisfiability in separation logic with inductive predicates. In: CSL-LICS. ACM (2014)
6. Brotherston, J., Gorogiannis, N., Kanovich, M.: Biabduction (and related problems) in array separation logic. In: CADE 26. LNCS, vol. 10395. Springer (2017)
7. Calcagno, C., Distefano, D.: Infer: An automatic program verifier for memory safety of C programs. In: NFM. LNCS, vol. 6617. Springer (2011)
8. Calcagno, C., Distefano, D., Dubreil, J., Gabi, D., Hooimeijer, P., Luca, M., O'Hearn, P.W., Papakonstantinou, I., Purbrick, J., Rodriguez, D.: Moving fast with software verification. In: NFM. LNCS, vol. 9058. Springer (2015)
9. Calcagno, C., Distefano, D., O'Hearn, P.W., Yang, H.: Beyond Reachability: Shape Abstraction in the Presence of Pointer Arithmetic. In: SAS. LNCS, vol. 4134. Springer (2006)
10. Chlipala, A.: Mostly-automated verification of low-level programs in computational separation logic. In: PLDI. ACM (2011)
11. Cook, B., Haase, C., Ouaknine, J., Parkinson, M.J., Worrell, J.: Tractable reasoning in a fragment of separation logic. In: CONCUR. LNCS, vol. 6901. Springer (2011)
12. Echenim, M., Iosif, R., Peltier, N.: Entailment checking in separation logic with inductive definitions is 2-exptime hard. In: LPAR. EPiCSC, EasyChair (2020)
13. Enea, C., Sighireanu, M., Wu, Z.: On automated lemma generation for separation logic with inductive definitions. In: ATVA. LNCS, vol. 9364. Springer (2015)
14. Gu, X., Chen, T., Wu, Z.: A complete decision procedure for linearly compositional separation logic with data constraints. In: IJCAR. LNAI, vol. 9706. Springer (2016)
15. Haase, C.: A survival guide to Presburger arithmetic. ACM SIGLOG News **5**, 67–82 (2018), https://dl.acm.org/citation.cfm?id=3242964
16. Katelaan, J., Matheja, C., Zuleger, F.: Effective entailment checking for separation logic with inductive definitions. In: TACAS. LNCS, vol. 11428. Springer (2019)
17. Kimura, D., Tatsuta, M.: Decision procedure for entailment of symbolic heaps with arrays. In: APLAS. LNCS, vol. 10695. Springer (2017)
18. Kimura, D., Tatsuta, M.: Decidability for entailments of symbolic heaps with arrays. CoRR **arXiv:1802.05935v3** (2018), http://arxiv.org/abs/1802.05935
19. Knuth, D.E.: The Art of Computer Programming, Volume 1 (3rd Ed.): Fundamental Algorithms. Addison Wesley Longman Publishing Co., Inc., USA (1997)
20. Le, Q.L.: Compositional satisfiability solving in separation logic. In: VMCAI. LNCS, vol. 12597. Springer (2021)
21. Marti, N., Affeldt, R., Yonezawa, A.: Formal verification of the heap manager of an operating system using separation logic. In: ICFEM. LNCS, Springer (2006)
22. O'Hearn, P.: Separation logic. Commun. ACM **62**(2) (2019)
23. Reynolds, J.: Separation logic: A logic for shared mutable data structures. In: LICS. IEEE Computer Society (2002)
24. Wilson, P.R., Johnstone, M.S., Neely, M., Boles, D.: Dynamic storage allocation: A survey and critical review. In: IWMM. Springer (1995)

A QFPA abstraction for SLAH formulas

A.1 QFPA summary of hls atoms

Lemma 1. *Let* $\texttt{hls}(x, y; z)$ *be an atom in SLAH representing a non-empty heap, where* x, y, z *are three distinct variables in* $\mathcal{V}$*. Then there is an QFPA formula, denoted by* $\texttt{Abs}^+(\texttt{hls}(x, y; z))$*, which summarizes* $\texttt{hls}(x, y; z)$*, namely for every stack* s $s \models \texttt{Abs}^+(\texttt{hls}(x, y; z))$ *iff there exists a heap* h *such that* $s, h \models \texttt{hls}(x, y, z)$*.*

Proof. The constraint that the atom represents a non-empty heap means that the inductive rule defining hls in Equation (7) should be applied at least once. Notice that the semantics of this rule defines, at each inductive step, a memory block starting at x and ending before x' of size $x' - x$. By induction on $k \geq 1$, we obtain that $\texttt{hls}^k(x, y; z)$ defines a memory block of length $y - x$ such that $2k \leq y - x \leq kz$. Then $\texttt{hls}(x, y; z)$ is summarized by the formula $\exists k.\ k \geq 1 \wedge 2k \leq y - x \leq kz$, which is a non-linear arithmetic formula.

The formula $\exists k.\ k \geq 1 \wedge 2k \leq y - x \leq kz$ is actually equivalent to the disjunction of the two formulas corresponding to the following two cases:

- If $2 = z$, then $\texttt{Abs}^+(\texttt{hls}(x, y; z))$ has the formula $\exists k.\ k \geq 1 \wedge y - x = 2k$, which is equivalent to the QFPA formula $2 \leq y - x \wedge y - x \equiv_2 0$, as a disjunct.
- If $2 < z$, then we consider the following sub-cases:
  - if $k = 1$ then $\texttt{Abs}^+(\texttt{hls}(x, y; z))$ contains the formula $2 \leq y - x \leq z$ as a disjunct;
  - if $k \geq 2$, then we observe that the intervals $[2k, kz]$ and $[2(k+1), (k+1)z]$ overlap. Indeed,

$$kz - 2(k + 1) = k(z - 2) - 2 \geq k - 2 \geq 0.$$

Therefore, $\bigcup_{k \geq 2} [2k, kz] = [4, \infty)$. It follows that the formula $\exists k.\ k \geq 2 \wedge 2k \leq y - x \leq kz$ is equivalent to $4 \leq y - x$. Therefore, $\texttt{Abs}^+(\texttt{hls}(x, y; z))$ contains $4 \leq y - x$ as a disjunct.

To sum up, we obtain

$$\begin{aligned}\texttt{Abs}^+(\texttt{hls}(x, y; z)) \triangleq\ & \big(2 = z\ \wedge\ \exists k.\ k \geq 1 \wedge 2k = y - x\big) \\ & \vee\ \big(2 < z\ \wedge\ \big(2 \leq y - x \leq z \vee 4 \leq y - x\big)\big),\end{aligned}$$

which can be further simplified into

$$\begin{aligned}\texttt{Abs}^+(\texttt{hls}(x, y; z)) \triangleq\ & \big(2 = z \wedge 2 \leq y - x \wedge y - x \equiv_2 0\big) \\ & \vee\ \big(2 < z \wedge 2 \leq y - x\big).\end{aligned}$$

□

A.2 $\texttt{Abs}(\varphi)$: The QFPA abstraction of φ

We utilize $\texttt{Abs}^+(\texttt{hls}(x, y; z))$ to obtain in polynomial time an equi-satisfiable QFPA abstraction for a symbolic heap φ, denoted by $\texttt{Abs}(\varphi)$.

We introduce some notations first. Given a formula $\varphi \equiv \Pi : \Sigma$, $\texttt{Atoms}(\varphi)$ denotes the set of spatial atoms in Σ, and $\texttt{PAtoms}(\varphi)$ denotes the set of predicate atoms in Σ.

Definition 6. *(Presburger abstraction of SLAH formula) Let $\varphi \equiv \Pi : \Sigma$ be a SLAH formula. The abstraction of $\varphi \equiv \Pi : \Sigma$, denoted by $\texttt{Abs}(\varphi)$ is the formula $\Pi \wedge \phi_\Sigma \wedge \phi_*$ where:*

- $\phi_\Sigma \triangleq \bigwedge_{a \in \texttt{Atoms}(\varphi)} \texttt{Abs}(a)$ *such that*

$$\texttt{Abs}(t_1 \mapsto t_2) \triangleq \texttt{true} \tag{14}$$
$$\texttt{Abs}(\texttt{blk}(t_1, t_2)) \triangleq t_1 < t_2 \tag{15}$$
$$\texttt{Abs}(\texttt{hls}(t_1, t_2, t_3)) \triangleq t_1 = t_2 \tag{16}$$
$$\vee\ (t_1 < t_2 \wedge \texttt{Abs}^+(\texttt{hls}(x, y; z))[t_1/x, t_2/y, t_3/z]). \tag{17}$$

- $\phi_* \triangleq \phi_1 \wedge \phi_2 \wedge \phi_3$ *specifies the semantics of separating conjunction, where*

$$\phi_1 \triangleq \bigwedge_{a_i, a_j \in \texttt{PAtoms}(\varphi), i<j} (\texttt{isNonEmp}_{a_i} \wedge \texttt{isNonEmp}_{a_j}) \rightarrow (\texttt{end}(a_j) \leq \texttt{start}(a_i) \vee \texttt{end}(a_i) \leq \texttt{start}(a_j)) \tag{18}$$
$$\phi_2 \triangleq \bigwedge_{a_i \in \texttt{PAtoms}, a_j \notin \texttt{PAtoms}} (\texttt{isNonEmp}_{a_i}) \rightarrow (\texttt{end}(a_j) \leq \texttt{start}(a_i) \vee \texttt{end}(a_i) \leq \texttt{start}(a_j)) \tag{19}$$
$$\phi_3 \triangleq \bigwedge_{a_i, a_j \notin \texttt{PAtoms}(\varphi), i<j} \texttt{end}(a_j) \leq \texttt{start}(a_i) \vee \texttt{end}(a_i) \leq \texttt{start}(a_j) \tag{20}$$

and for each spatial atom a_i, $\texttt{isNonEmp}_{a_i}$ is an abbreviation of the formula $\texttt{start}(a_i) < \texttt{end}(a_i)$.

For formulas $\varphi \equiv \exists \vec{z} \cdot \Pi : \Sigma$, we define $\texttt{Abs}(\varphi) \triangleq \texttt{Abs}(\Pi : \Sigma)$ since $\exists \vec{z} \cdot \Pi : \Sigma$ and $\Pi : \Sigma$ are equi-satisfiable.

Proposition 1. *A SLAH formula φ is satisfiable iff $\texttt{Abs}(\varphi)$ is satisfiable.*

Proof. "Only if" direction: Suppose that φ is satisfiable.

Then there exists a stack s and a heap h such that $s, h \models \varphi$. We show $s \models \texttt{Abs}(\varphi)$. The semantics of SLAH implies that at least one disjunct in $\texttt{Abs}(a)$ is true for every predicate atom a. Therefore, $s \models \phi_\Sigma$. Moreover, $s \models \phi_*$, as a result of the semantics of separating conjunction. Thus $s \models \texttt{Abs}(\varphi)$ and $\texttt{Abs}(\varphi)$ is satisfiable.

"If" direction: Suppose that $\texttt{Abs}(\varphi)$ is satisfiable.

Then there is an interpretation s such that $s \models \texttt{Abs}(\varphi)$. We build a heap h from s and φ as follows:

- For each points-to atom $t_1 \mapsto t_2$ in φ, $h(s(t_1)) = s(t_2)$.

- For each block atom $\mathtt{blk}(t_1, t_2)$ in φ, $h(n) = 1$ for each $n \in [s(t_1), s(t_2) - 1]$.
- For every predicate atom $\mathtt{hls}(t_1, t_2; t_3)$ in φ, we have $s \models \mathtt{Abs}(\mathtt{hls}(t_1, t_2; t_3))$. Then either $s(t_1) = s(t_2)$ or $s(t_1) < s(t_2)$ and $s \models \mathtt{Abs}^+(\mathtt{hls}(x, y; z))[t_1/x, t_2/y, t_3/z]$.
  - For the first case, we let $h(s(t_1))$ undefined.
  - For the second case, $s \models (2 = t_3 \wedge t_1 < t_2 \wedge t_2 - t_1 \equiv 0 \bmod 2) \vee (2 < t_3 \wedge 2 \leq t_2 - t_1)$.
    - If $s(t_3) = 2$, then let $h(s(t_1) + 2(i-1)) = 2$ and $h(s(t_1) + 2i - 1) = 1$ for every $i : 1 \leq i \leq \frac{s(t_2) - s(t_1)}{2}$.
    - If $s(t_3) > 2$, then from $s(t_2) - s(t_1) \geq 2$, we know that there is a sequence of numbers $n_1, \cdots, n_\ell$ such that $2 \leq n_i \leq s(t_3)$ for every $i \in [\ell]$ and $s(t_2) = s(t_1) + \sum_{i \in [\ell]} n_i$. We then let $h(s(t_1) + \sum_{j \in [i-1]} n_j) = n_i$ for every $i \in [\ell]$, and let $h(n') = 1$ for all the other addresses $n' \in [s(t_1), s(t_2) - 1]$.

    From the construction above, we know that the subheap of the domain $[s(t_1), s(t_2) - 1]$ satisfies $\mathtt{hls}(t_1, t_2; t_3)$.

From the definition of h, we know that $s, h \models \varphi$. Therefore, φ is satisfiable. □

Remark 1. From the definition of $\mathtt{Abs}(\varphi)$, it follows that $\mathtt{Abs}(\varphi)$ is in QFPA and the size of $\mathtt{Abs}(\varphi)$ is polynomial in that of φ. From the fact that the satisfiability of QFPA is in NP, we conclude that the satisfiability of SLAH is in NP.

B Proof of Lemma 3

Lemma 3. *For an SLAH formula $\varphi \equiv \Pi : \mathtt{hls}(t_1', t_2'; t_3')$, a QFPA formula $\xi_{eub,\varphi}(z)$ can be constructed in linear time such that for every store s satisfying $s \models \mathtt{Abs}(\varphi)$, we have $s[z \leftarrow \mathtt{EUB}_\varphi(s)] \models \xi_{eub,\varphi}(z)$ and $s[z \leftarrow n] \not\models \xi_{eub,\varphi}(z)$ for all $n \neq \mathtt{EUB}_\varphi(s)$.*

Proof. From the construction of $\mathtt{Abs}^+(\mathtt{hls}(t_1', t_2'; t_3'))$ in Section 4, we know that for every store s, there is a heap h such that $s, h \models \Pi : \mathtt{hls}(t_1', t_2'; t_3')$ iff $s \models \Pi \wedge \big((t_3' = 2 \wedge 2 \leq t_2' - t_1' \wedge t_2' - t_1' \equiv_2 0) \vee (2 < t_3' \wedge 2 \leq t_2' - t_1')\big)$. Then the fact that z appears in some unfolding scheme of φ w.r.t. some store s can be specified by the formula

$$\Pi \wedge \begin{pmatrix} (t_3' = 2 \wedge z = 2 \wedge 2 \leq t_2' - t_1' \wedge t_2' - t_1' \equiv_2 0) \vee \\ (2 < t_3' \wedge 2 \leq z \leq t_3' \ \wedge (2 \leq t_2' - t_1' - z \vee t_2' - t_1' = z)) \end{pmatrix},$$

where $t_2' - t_1' - z$ denotes the remaining size of the heap after removing a memory chunk of size z. Therefore, we define $\xi_{eub,\varphi}(z)$ as

$$\Pi \wedge \left(\begin{array}{l} (t_3' = 2 \wedge z = 2 \wedge 2 \leq t_2' - t_1' \wedge t_2' - t_1' \equiv_2 0) \vee \\ \left(\begin{array}{l} 2 < t_3' \wedge 2 \leq z \leq t_3' \wedge (2 \leq t_2' - t_1' - z \vee t_2' - t_1' = z) \wedge \\ \forall z'.\ z < z' \leq t_3' \rightarrow \neg(2 \leq t_2' - t_1' - z' \vee t_2' - t_1' = z') \end{array} \right) \end{array} \right),$$

where $\forall z'.\ z < z' \leq t'_3 \rightarrow \neg(2 \leq t'_2 - t'_1 - z' \vee t'_2 - t'_1 = z')$ asserts that z is the maximum chunk size upper bound among unfolding schemes of φ. The formula $\forall z'.\ z < z' \leq t'_3 \rightarrow \neg(2 \leq t'_2 - t'_1 - z' \vee t'_2 - t'_1 = z')$ is equivalent to $\forall z'.\ z < z' \leq t'_3 \rightarrow (t'_2 - t'_1 - z' \leq 1 \wedge t'_2 - t'_1 \neq z')$, and can be simplified into

$$\begin{array}{l} z+1 \leq t'_3 \rightarrow (t'_2 - t'_1 \leq z+2 \wedge t'_2 - t'_1 \neq z+1) \wedge \\ z+2 \leq t'_3 \rightarrow t'_2 - t'_1 \neq z+2, \end{array}$$

where $t'_2 - t'_1 \neq z+1$ is an abbreviation of $t'_2 - t'_1 < z+1 \vee z+1 < t'_2 - t'_1$, similarly for $t'_2 - t'_1 \neq z+2$.

It follows that $\xi_{eub,\varphi}(z)$ can be simplified into

$$\Pi \wedge \left(\begin{array}{l} (t'_3 = 2 \wedge z = 2 \wedge 2 \leq t'_2 - t'_1 \wedge t'_2 - t'_1 \equiv_2 0) \vee \\ \left(\begin{array}{l} 2 < t'_3 \wedge 2 \leq z \leq t'_3 \wedge (2 \leq t'_2 - t'_1 - z \vee t'_2 - t'_1 = z) \wedge \\ z+1 \leq t'_3 \rightarrow (t'_2 - t'_1 \leq z+2 \wedge t'_2 - t'_1 \neq z+1) \wedge \\ z+2 \leq t'_3 \rightarrow t'_2 - t'_1 \neq z+2 \end{array} \right) \end{array} \right),$$

which is a QFPA formula. □

C Proof of Lemma 4

Lemma 4 *Let $\varphi \equiv \Pi : \mathtt{hls}(t'_1, t'_2; t'_3)$, $\psi \equiv \mathtt{hls}(t_1, t_2; t_3)$, and $\preceq$ be a total preorder over $\mathcal{A}(\varphi) \cup \mathcal{A}(\psi)$ such that $C_\preceq \models t'_1 < t'_2 \wedge t'_1 = t_1 \wedge t'_2 = t_2$. Then $\varphi \models_\preceq \psi$ iff $C_\preceq \wedge \mathtt{Abs}(\varphi) \models \forall z.\ \xi_{eub,\varphi}(z) \rightarrow z \leq t_3$.*

Proof. "Only if" direction: Suppose $\varphi \models_\preceq \psi$.

At first, we observe that $C_\preceq \wedge \mathtt{Abs}(\varphi)$ and φ contain the same number of variables.

Let s be an interpretation such that $s \models C_\preceq \wedge \mathtt{Abs}(\varphi)$. From $C_\preceq \models t'_1 < t'_2 \wedge t'_1 = t_1 \wedge t'_2 = t_2$, we have $s(t'_1) < s(t'_2)$, $s(t'_1) = s(t_1)$, and $s(t'_2) = s(t_2)$.

At first, from $s \models \mathtt{Abs}(\varphi)$ and Lemma 3, we deduce that $s[z \rightarrow \mathtt{EUB}_\varphi(s)] \models \xi_{eub,\varphi}(z)$. Moreover, from the definition of $\mathtt{EUB}_\varphi(s)$, we know that there is an unfolding scheme, say $sz_1, \cdots, sz_\ell$, such that $\mathtt{EUB}_\varphi(s) = \max(sz_1, \cdots, sz_\ell)$. From the definition of unfolding schemes, we have $2 \leq sz_i \leq s(t'_3)$ for every $i \in [\ell]$ and $s(t'_2) = s(t'_1) + \sum_{i \in [\ell]} sz_i$. Therefore, there is a heap h such that $s, h \models \varphi$ and for every $i \in [\ell]$, $h(s(t'_1) + \sum_{j \in [i-1]} sz_j) = sz_i$. From the assumption $\varphi \models_\preceq \psi$, we deduce that $s, h \models \psi \equiv \mathtt{hls}(t_1, t_2; t_3)$. Then each chunk of h has to be matched exactly to one unfolding of $\mathtt{hls}(t_1, t_2; t_3)$. Thus, $2 \leq sz_i \leq s(t_3)$ for every $i \in [\ell]$. Therefore, $\mathtt{EUB}_\varphi(s) = \max(sz_1, \cdots, sz_\ell) \leq s(t_3)$. We deduce that $s[z \leftarrow \mathtt{EUB}_\varphi(s)] \models \xi_{eub,\varphi}(z) \wedge z \leq t_3$. Moreover, from Lemma 3, we know that for all $n \neq \mathtt{EUB}_\varphi(s)$, $s[z \leftarrow n] \not\models \xi_{eub,\varphi}(z)$. Consequently, $s \models \forall z.\ \xi_{eub,\varphi}(z) \rightarrow z \leq t_3$.

"If" direction: Suppose $C_\preceq \wedge \mathtt{Abs}(\varphi) \models \forall z.\ \xi_{eub,\varphi}(z) \rightarrow z \leq t_3$ and $s, h \models C_\preceq \wedge \Pi : \mathtt{hls}(t'_1, t'_2; t'_3)$. We show $s, h \models \psi$.

From $s, h \models C_\preceq \wedge \Pi : \mathtt{hls}(t'_1, t'_2; t'_3)$, we know that there are $sz_1, \cdots, sz_\ell$ such that they are the sequence of chunk sizes in h. Therefore, $(sz_1, \cdots, sz_\ell)$ is an

unfolding scheme of φ w.r.t. s. From the definition of $\texttt{EUB}_\varphi(s)$, we have $\texttt{EUB}_\varphi(s)$ is the maximum chunk size upper bound of the unfolding schemes of φ w.r.t. s. Therefore, $\max(sz_1, \cdots, sz_\ell) \leq \texttt{EUB}_\varphi(s)$.

From $s, h \models C_\preceq \wedge \Pi : \texttt{hls}(t'_1, t'_2; t'_3)$, we deduce that $s \models C_\preceq \wedge \texttt{Abs}(\varphi)$. Because $C_\preceq \wedge \texttt{Abs}(\varphi) \models \forall z.\ \xi_{eub,\varphi}(z) \rightarrow z \leq t_3$, we have $s \models \forall z.\ \xi_{eub,\varphi}(z) \rightarrow z \leq t_3$. From Lemma 3, we know that $s[z \leftarrow \texttt{EUB}_\varphi(s)] \models \xi_{eub,\varphi}(z)$. Therefore, we deduce that $s[z \leftarrow \texttt{EUB}_\varphi(s)] \models z \leq t_3$, namely, $\texttt{EUB}_\varphi(s) \leq s(t_3)$. Then $\max(sz_1, \cdots, sz_\ell) \leq \texttt{EUB}_\varphi(s) \leq s(t_3)$. It follows that for every $i \in [\ell]$, $2 \leq sz_i \leq s(t_3)$. We conclude that $s, h \models \texttt{hls}(t_1, t_2; t_3) \equiv \psi$. □

D Complexity of the entailment problem

Theorem 2 states that the upper bound of the entailment problem is EXPTIME. The lower bound is given by the complexity of the entailment problem for quantifier free formulas in ASL, which is shown to be coNP in [6].

Claim: The entailment problem in SLAH is in **coNP**.

Proof. Let fix $\varphi \models \psi$ an entailment problem in SLAH.

We build in polynomial time a PA formula χ from φ and ψ such that for any stack s, $s \models \chi$ iff $\exists h.\ s, h \models \varphi$ and $s, h \not\models \psi$.

We suppose that both formula φ and ψ have been transformed in normal form as follows:

- atoms $\texttt{hls}(x, y; v')$ with $\texttt{Abs}(\varphi) \models (v' < 2 \vee x = y)$ have been removed and replaced by $x = y$
- atoms $\texttt{hls}(x, y; v')$ with $\texttt{Abs}(\varphi) \models y - x = 2 \wedge v' \geq 2$ have been replaced by $\exists z.\ \ z \leq v' \wedge y - x = 2 :\ x \mapsto z \star \texttt{blk}(x + 1, y)$
- ...

M: use ξ ?

M: TODO

The normalization can be done in a linear number of calls to the EPA procedure.

The formula χ shall ensure that φ is satisfiable, so it contains as conjunct $\texttt{Abs}(\varphi)$.

The entailment may be invalid iff one of the following holds:

1. ψ is not satisfiable, i.e., $\texttt{Abs}(\psi)$ is not satisfiable.
2. there exists an address inside a spatial atom of φ which is not inside a spatial atom of ψ; we denote this formula by $cov(\varphi, \psi)$;
3. similarly, $cov(\psi, \varphi)$ (the semantics is non intuitionistic),
4. a points-to or a heap-list atom of ψ "covers" a domain of addresses which is inside the one of a block atom of φ; we denote this formula by $nblk(\varphi, \psi)$;
5. a points-to atom of ψ defines an address inside (strictly) a heap-list atom of φ; we denote this formula by $npto(\varphi, \psi)$;
6. a points-to atom $x \mapsto v$ of ψ defines the same address as the start address of a heap-list atom $\texttt{hls}(x, y; v')$ in φ; we denote this formula by $npto'(\varphi, \psi)$; (this is true because we are in the normal form);
7. a points-to atoms $x \mapsto v$ of ψ defines the same address as a points-to atom $x \mapsto v'$ of φ but $v \neq v'$;

8. a heap-list atom $\texttt{hls}(x, y; v')$ of ψ starts at an address at which is defined a points-to atom $x \mapsto v$ of φ such that $v > v'$;
9. a heap-list atom $\texttt{hls}(x, y; v')$ of ψ "covers" the domain of addresses if an atom $\texttt{hls}(z, w; v)$ of φ but $v' < v$;
10. a heap-list atom $\texttt{hls}(x, y; v')$ of ψ "covers" a domain of addresses that can not be filded into a `hls` atom with chunks of maximal size less or equal to v'

M: difficult in P!