The Effects of Adding Reachability Predicates in Propositional Separation Logic

Stéphane Demri, Etienne Lozes, Alessio Mansutti

The Effects of Adding Reachability Predicates in Propositional Separation Logic

S. Demri[1], E. Lozes[2], and A. Mansutti[1]

[1]LSV, CNRS, ENS Paris-Saclay, Université Paris-Saclay, France
[2]I3S, Université Côte d'Azur, France

Abstract. The list segment predicate `ls` used in separation logic for verifying programs with pointers is well-suited to express properties on singly-linked lists. We study the effects of adding `ls` to the full propositional separation logic with the separating conjunction and implication, which is motivated by the recent design of new fragments in which all these ingredients are used indifferently and verification tools start to handle the magic wand connective. This is a very natural extension that has not been studied so far. We show that the restriction without the separating implication can be solved in polynomial space by using an appropriate abstraction for memory states whereas the full extension is shown undecidable by reduction from first-order separation logic. Many variants of the logic and fragments are also investigated from the computational point of view when `ls` is added, providing numerous results about adding reachability predicates to propositional separation logic.

1 Introduction

Separation logic [20,25,28] is a well-known assertion logic for reasoning about programs with dynamic data structures. Since the implementation of Smallfoot and the evidence that the method is scalable [3,33], many tools supporting separation logic as an assertion language have been developed [3,16,33,8,9,17]. Even though the first tools could handle relatively limited fragments of separation logic, like symbolic heaps, there is a growing interest and demand to consider extensions with richer expressive power. We can point out three particular extensions of symbolic heaps (without list predicates) that have been proved decidable.

- Symbolic heaps with generalised inductive predicates, adding a fixpoint combinator to the language, is a convenient logic for specifying data structures that are more advanced than lists or trees. The entailment problem is known to be decidable by means of tree automata techniques for the bounded treewidth fragment [19,1], whereas satisfiability is ExpTime-complete [6]. Other related results can be found in [21].
- List-free symbolic heaps with all classical Boolean connectives $\wedge$ and $\neg$ (and with the separating conjunction $*$), called herein SL($*$), is a convenient extension when combinations of results of various analysis need to be expressed, or when the analysis requires a complementation. This extension already is PSpace-complete [11].

– Propositional separation logic with separating implication, a.k.a. magic wand ($-\!\!*$), is a convenient fragment (called herein SL($*, -\!\!*$)) in which can be solved two problems of frame inference and abduction, that play an important role in static analysers and provers built on top of separation logic. SL($*, -\!\!*$) can be decided in PSPACE thanks to a small model property [32].

A natural question is how to combine these extensions, and which separation logic fragment that allows Boolean connectives, magic wand and generalised recursive predicates can be decided with some adequate restrictions. As already advocated in [7,31,29,18,24], dealing with the separating implication $-\!\!*$ is a desirable feature for program verification and several semi-automated or automated verification tools support it in some way, see e.g. [31,29,24,18].

Our contribution. In this paper, we address the question of combining magic wand and inductive predicates in the extremely limited case where the only inductive predicate is the gentle list segment predicate $\mathtt{ls}$. So the starting point of this work is this puzzling question: what is the complexity/decidability status of propositional separation logic SL($*, -\!\!*$) enriched with the list segment predicate $\mathtt{ls}$ (herein called SL($*, -\!\!*, \mathtt{ls}$))? More precisely, we study the decidability/complexity status of extensions of propositional separation logic SL($*, -\!\!*$) by adding one of the reachability predicates among $\mathtt{ls}$ (precise predicate as usual in separation logic), $\mathtt{reach}$ (existence of a path, possibly empty) and $\mathtt{reach}^+$ (existence of a non-empty path).

First, we establish that the satisfiability problem for the propositional separation logic SL($*, -\!\!*, \mathtt{ls}$) is undecidable. Our proof is by reduction from the undecidability of first-order separation logic [5,14], using an encoding of the variables as heap cells (see Theorem 1). As a consequence, we also establish that SL($*, -\!\!*, \mathtt{ls}$) is not finitely axiomatisable. Moreover, our reduction requires a rather limited expressive power of the list segment predicate, and we can strengthen our undecidability results to some fragments of SL($*, -\!\!*, \mathtt{ls}$). For instance, surprisingly, the extension of SL($*, -\!\!*$) with the atomic formulae of the form $\mathtt{reach}(\mathtt{x}, \mathtt{y}) = 2$ and $\mathtt{reach}(\mathtt{x}, \mathtt{y}) = 3$ (existence of a path between $\mathtt{x}$ and $\mathtt{y}$ of respective length 2 or 3) is already undecidable, whereas the satisfiability problem for SL($*, -\!\!*, \mathtt{reach}(\mathtt{x}, \mathtt{y}) = 2$) is known to be in PSPACE [15].

Second, we show that the satisfiability problem for SL($*, \mathtt{reach}^+$) is PSPACE-complete, extending the well-known result on SL($*$). The PSPACE upper bound relies on a small heap property based on the techniques of test formulae, see e.g. [23,22,4,15], and the PSPACE-hardness of SL($*$) is inherited from [11]. The PSPACE upper bound can be extended to the fragment of SL($*, -\!\!*, \mathtt{reach}^+$) made of Boolean combinations of formulae from SL($*, \mathtt{reach}^+$) $\cup$ SL($*, -\!\!*$) (see the developments in Section 4). Even better, we show that the fragment of SL($*, -\!\!*, \mathtt{reach}^+$) in which $\mathtt{reach}^+$ is not in the scope of $-\!\!*$ is decidable. As far as we know, this is the largest fragment including full Boolean expressivity, $-\!\!*$ and $\mathtt{ls}$ for which decidability is established.

2 Preliminaries

Let PVAR $= \{\mathtt{x}, \mathtt{y}, \ldots\}$ be a countably infinite set of *program variables* and LOC $= \{\ell_0, \ell_1, \ell_2, \ldots\}$ be a countable infinite set of *locations*. A *memory state* is a pair (s, h) such that $s : \text{PVAR} \to \text{LOC}$ is a variable valuation (known as the *store*) and $h : \text{LOC} \to_{\text{fin}} \text{LOC}$ is a partial function with finite domain, known as the *heap*. We write $\text{dom}(h)$ to denote its domain and $\text{ran}(h)$ to denote its range. Given a heap h with $\text{dom}(h) = \{\ell_1, \ldots, \ell_n\}$, we also write $\{\ell_1 \mapsto h(\ell_1), \ldots, \ell_n \mapsto h(\ell_n)\}$ to denote h. Each $\ell_i \mapsto h(\ell_i)$ is understood as a *memory cell* of h.

As usual, the heaps h_1 and h_2 are said to be *disjoint*, written $h_1 \perp h_2$, if $\text{dom}(h_1) \cap \text{dom}(h_2) = \emptyset$; when this holds, we write $h_1 + h_2$ to denote the heap corresponding to the disjoint union of the graphs of h_1 and h_2, hence $\text{dom}(h_1 + h_2) = \text{dom}(h_1) \uplus \text{dom}(h_2)$. When the domains of h_1 and h_2 are not disjoint, the composition $h_1 + h_2$ is not defined. Moreover, we write $h' \sqsubseteq h$ to denote that $\text{dom}(h') \subseteq \text{dom}(h)$ and for all locations $\ell \in \text{dom}(h')$, we have $h'(\ell) = h(\ell)$. The formulae φ of the separation logic $\text{SL}(*, \mathrel{-\!\!*}, \mathtt{ls})$ and its atomic formulae π are built from $\pi ::= \mathtt{x} = \mathtt{y} \mid \mathtt{x} \hookrightarrow \mathtt{y} \mid \mathtt{ls}(\mathtt{x}, \mathtt{y}) \mid \mathtt{emp} \mid \top$ and $\varphi ::= \pi \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi * \varphi \mid \varphi \mathrel{-\!\!*} \varphi$, where $\mathtt{x}, \mathtt{y} \in \text{PVAR}$ ($\Rightarrow$, $\Leftrightarrow$ and $\vee$ are defined as usually). Models of the logic $\text{SL}(*, \mathrel{-\!\!*}, \mathtt{ls})$ are memory states and the satisfaction relation $\models$ is defined as follows (omitting standard clauses for $\neg$,$\wedge$):

$$
\begin{array}{lcl}
(s,h) \models \mathtt{x} = \mathtt{y} & \iff & s(\mathtt{x}) = s(\mathtt{y}) \\
(s,h) \models \mathtt{emp} & \iff & \text{dom}(h) = \emptyset \\
(s,h) \models \mathtt{x} \hookrightarrow \mathtt{y} & \iff & s(\mathtt{x}) \in \text{dom}(h) \text{ and } h(s(\mathtt{x})) = s(\mathtt{y}) \\
(s,h) \models \mathtt{ls}(\mathtt{x},\mathtt{y}) & \iff & \text{either } (\text{dom}(h) = \emptyset \text{ and } s(\mathtt{x}) = s(\mathtt{y})) \text{ or} \\
 & & h = \{\ell_0 \mapsto \ell_1, \ell_1 \mapsto \ell_2, \ldots, \ell_{n-1} \mapsto \ell_n\} \text{ with } n \geq 1, \\
 & & \ell_0 = s(\mathtt{x}),\ \ell_n = s(\mathtt{y}) \text{ and for all } i \neq j \in [0,n],\ \ell_i \neq \ell_j \\
(s,h) \models \varphi_1 * \varphi_2 & \iff & \text{there are } h_1 \text{ and } h_2 \text{ such that } (h_1 \perp h_2,\ (h_1 + h_2) = h, \\
 & & (s,h_1) \models \varphi_1 \text{ and } (s,h_2) \models \varphi_2) \\
(s,h) \models \varphi_1 \mathrel{-\!\!*} \varphi_2 & \iff & \forall h_1 \text{ if } (h_1 \perp h \text{ and } (s,h_1) \models \varphi_1) \text{ then } (s, h + h_1) \models \varphi_2.
\end{array}
$$

Note that the semantics for $*$, $\mathrel{-\!\!*}$, $\hookrightarrow$, $\mathtt{ls}$ and for all other ingredients is the usual one in separation logic and $\mathtt{ls}$ is the *precise* list segment predicate. In the sequel, we use the following abbreviations: $\mathtt{size} \geq 0 \stackrel{\text{def}}{=} \top$ and for all $\beta \geq 0$, $\mathtt{size} \geq \beta{+}1 \stackrel{\text{def}}{=} (\mathtt{size} \geq \beta) * \neg\mathtt{emp}$, $\mathtt{size} \leq \beta \stackrel{\text{def}}{=} \neg(\mathtt{size} \geq \beta{+}1)$ and $\mathtt{size} = \beta \stackrel{\text{def}}{=} (\mathtt{size} \leq \beta) \wedge (\mathtt{size} \geq \beta)$. Moreover, $\varphi_1 \mathrel{-\!\!\circledast} \varphi_2 \stackrel{\text{def}}{=} \neg(\varphi_1 \mathrel{-\!\!*} \neg\varphi_2)$ (*septraction connective*), $\mathtt{alloc}(\mathtt{x}) \stackrel{\text{def}}{=} (\mathtt{x} \hookrightarrow \mathtt{x}) \mathrel{-\!\!*} \bot$ and $\mathtt{x} \mapsto \mathtt{y} \stackrel{\text{def}}{=} (\mathtt{x} \hookrightarrow \mathtt{y}) \wedge \mathtt{size} = 1$. W.l.o.g., we can assume that LOC $= \mathbb{N}$ since none of the developments depend on the elements of LOC as the only predicate involving locations is the equality. We write $\text{SL}(*, \mathrel{-\!\!*})$ to denote the restriction of $\text{SL}(*, \mathrel{-\!\!*}, \mathtt{ls})$ without $\mathtt{ls}$. Similarly, we write $\text{SL}(*)$ to denote the restriction of $\text{SL}(*, \mathrel{-\!\!*})$ without $\mathrel{-\!\!*}$. Given two formulae φ, φ' (possibly from different logical languages), we write $\varphi \equiv \varphi'$ whenever for all (s,h), we have $(s,h) \models \varphi$ iff $(s,h) \models \varphi'$. When $\varphi \equiv \varphi'$, the formulae φ and φ' are said to be *equivalent*.

Variants with other reachability predicates. We use two additional reachability predicates $\mathtt{reach}(\mathtt{x}, \mathtt{y})$ and $\mathtt{reach}^{+}(\mathtt{x}, \mathtt{y})$ and we write $\text{SL}(*, \mathrel{-\!\!*}, \mathtt{reach})$ (resp.

$\mathrm{SL}(*, -\!\!*, \mathtt{reach}^+)$) to denote the variant of $\mathrm{SL}(*, -\!\!*, \mathtt{ls})$ in which $\mathtt{ls}$ is replaced by $\mathtt{reach}$ (resp. by $\mathtt{reach}^+$). The relation $\models$ is extended as follows: $(s,h) \models \mathtt{reach}(\mathtt{x},\mathtt{y})$ holds when there is $i \geq 0$ such that $h^i(s(\mathtt{x})) = s(\mathtt{y})$ (i functional composition(s) of h is denoted by h^i) and $(s,h) \models \mathtt{reach}^+(\mathtt{x},\mathtt{y})$ holds when there is $i \geq 1$ such that $h^i(s(\mathtt{x})) = s(\mathtt{y})$. As $\mathtt{ls}(\mathtt{x},\mathtt{y}) \equiv \mathtt{reach}(\mathtt{x},\mathtt{y}) \wedge \neg(\neg\mathtt{emp} * \mathtt{reach}(\mathtt{x},\mathtt{y}))$ and $\mathtt{reach}(\mathtt{x},\mathtt{y}) \equiv \top * \mathtt{ls}(\mathtt{x},\mathtt{y})$, the logics $\mathrm{SL}(*, -\!\!*, \mathtt{reach})$ and $\mathrm{SL}(*, -\!\!*, \mathtt{ls})$ have identical decidability status. As far as computational complexity is concerned, a similar analysis can be done as soon as $*$, $\neg$, $\wedge$ and $\mathtt{emp}$ are parts of the fragments (the details are omitted here). Similarly, we have the equivalences: $\mathtt{reach}(\mathtt{x},\mathtt{y}) \equiv \mathtt{x} = \mathtt{y} \vee \mathtt{reach}^+(\mathtt{x},\mathtt{y})$ and $\mathtt{ls}(\mathtt{x},\mathtt{y}) \equiv (\mathtt{x} = \mathtt{y} \wedge \mathtt{emp}) \vee (\mathtt{reach}^+(\mathtt{x},\mathtt{y}) \wedge \neg(\neg\mathtt{emp} * \mathtt{reach}^+(\mathtt{x},\mathtt{y})))$. So clearly, $\mathrm{SL}(*, \mathtt{reach})$ and $\mathrm{SL}(*, \mathtt{ls})$ can be viewed as fragments of $\mathrm{SL}(*, \mathtt{reach}^+)$ and, $\mathrm{SL}(*, -\!\!*, \mathtt{ls})$ as a fragment of $\mathrm{SL}(*, -\!\!*, \mathtt{reach}^+)$. It is therefore stronger to establish decidability or complexity upper bounds with $\mathtt{reach}^+$ and to show undecidability or complexity lower bounds with $\mathtt{ls}$ or $\mathtt{reach}$. Herein, we provide the optimal results.

Decision problems. Let $\mathfrak{L}$ be a logic defined above. As usual, the *satisfiability problem for* $\mathfrak{L}$ takes as input a formula φ from $\mathfrak{L}$ and asks whether there is (s,h) such that $(s,h) \models \varphi$. The *validity problem* is also defined as usual. The *model-checking problem for* $\mathfrak{L}$ takes as input a formula φ from $\mathfrak{L}$, (s,h) and asks whether $(s,h) \models \varphi$ (s is restricted to the variables occurring in φ and h is encoded as a finite and functional graph). Unless otherwise specified, the *size* of a formula φ is understood as its tree size, i.e. approximately its number of symbols.

The main purpose of this paper is to study the decidability/complexity status of $\mathrm{SL}(*, -\!\!*, \mathtt{ls})$ and its fragments.

3 Undecidability of $\mathrm{SL}(*, -\!\!*, \mathtt{ls})$

In this section, we show that $\mathrm{SL}(*, -\!\!*, \mathtt{ls})$ has an undecidable satisfiability problem even though it does not admit first-order quantification.

Let $\mathrm{SL}(\forall, -\!\!*)$ be the first-order extension of $\mathrm{SL}(-\!\!*)$ obtained by adding the universal quantifier $\forall$. The formulae φ of $\mathrm{SL}(\forall, -\!\!*)$ are built from $\pi ::= \mathtt{x} = \mathtt{y} \mid \mathtt{x} \hookrightarrow \mathtt{y}$ and $\varphi ::= \pi \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi -\!\!* \varphi \mid \forall \mathtt{x}\ \varphi$, where $\mathtt{x}, \mathtt{y} \in \mathrm{PVAR}$. Note that $\mathtt{emp}$ can be easily defined by $\forall\, \mathtt{x}, \mathtt{x}'\ \neg(\mathtt{x} \hookrightarrow \mathtt{x}')$. Models of the logic $\mathrm{SL}(\forall, -\!\!*)$ are memory states and the satisfaction relation $\models$ is defined as for $\mathrm{SL}(-\!\!*)$ with the additional clause:

$$(s,h) \models \forall \mathtt{x}\ \varphi \iff \text{for all } \ell \in \mathrm{LOC},\ \text{we have } (s[\mathtt{x} \leftarrow \ell], h) \models \varphi.$$

Without any loss of generality, we can assume that the satisfiability [resp. validity] problem for $\mathrm{SL}(\forall, -\!\!*)$ is defined by taking as inputs closed formulae (i.e. without free occurrences of the variables).

Proposition 1. *[5,14] The satisfiability problem for* $\mathrm{SL}(\forall, -\!\!*)$ *is undecidable and the set of valid formulae for* $\mathrm{SL}(\forall, -\!\!*)$ *is not recursively enumerable.*

In a nutshell, we establish the undecidability of $\mathrm{SL}(*, -\!\!*, \mathtt{ls})$ by reduction from the satisfiability problem for $\mathrm{SL}(\forall, -\!\!*)$. The reduction is nicely decomposed in two intermediate steps: (1) the undecidability of $\mathrm{SL}(*, -\!\!*)$ extended with a few atomic predicates, to be defined soon, and (2) a *tour de force* resulting in the encoding of these atomic predicates in $\mathrm{SL}(*, -\!\!*, \mathtt{ls})$.

3.1 Encoding quantified variables as cells in the heap

In this section, we assume for a moment that we can express three atomic predicates $\mathtt{alloc}^{-1}(\mathtt{x})$, $n(\mathtt{x}) = n(\mathtt{y})$ and $n(\mathtt{x}) \hookrightarrow n(\mathtt{y})$, that will be used in the translation and have the following semantics:

- $(s,h) \models \mathtt{alloc}^{-1}(\mathtt{x})$ holds whenever $s(\mathtt{x}) \in \mathrm{ran}(h)$,
- $(s,h) \models n(\mathtt{x}) = n(\mathtt{y})$ holds iff $\{s(\mathtt{x}), s(\mathtt{y})\} \subseteq \mathrm{dom}(h)$ and $h(s(\mathtt{x})) = h(s(\mathtt{y}))$,
- $(s,h) \models n(\mathtt{x}) \hookrightarrow n(\mathtt{y})$ holds iff $\{s(\mathtt{x}), s(\mathtt{y})\} \subseteq \mathrm{dom}(h)$ and $h^2(s(\mathtt{x})) = h(s(\mathtt{y}))$.

Let us first intuitively explain how the two last predicates will help encoding $\mathrm{SL}(\forall, -\!\!*)$. By definition, the satisfaction of the quantified formula $\forall \mathtt{x}\ \psi$ from $\mathrm{SL}(\forall, -\!\!*)$ requires the satisfaction of the formula ψ for all the values in LOC assigned to $\mathtt{x}$. The principle of the encoding is to use a set L of locations initially not in the domain or range of the heap to mimic the store by modifying how they are allocated. In this way, a variable will be interpreted by a location in the heap and, instead of checking whenever $\mathtt{x} \hookrightarrow \mathtt{y}$ (or $\mathtt{x} = \mathtt{y}$) holds, we will check if $n(\mathtt{x}) \hookrightarrow n(\mathtt{y})$ (or $n(\mathtt{x}) = n(\mathtt{y})$) holds, where $\mathtt{x}$ and $\mathtt{y}$ correspond, after the translation, to the locations in L that mimic the store for those variables. Let X be the set of variables needed for the translation. In order to properly encode the store, each location in L only mimics exactly one variable, i.e. there is a bijection between X and L, and cannot be reached by any location. As such, the formula $\forall \mathtt{x}\ \psi$ will be encoded by the formula $(\mathtt{alloc}(\mathtt{x}) \wedge \mathtt{size} = 1) -\!\!* (\mathrm{OK}(X) \Rightarrow \mathrm{T}(\psi))$, where $\mathrm{OK}(X)$ (formally defined below) checks whenever the locations in L still satisfy the auxiliary conditions just described, whereas $\mathrm{T}(\psi)$ is the translation of ψ.

Unfortunately, the formula $\psi_1 -\!\!* \psi_2$ cannot simply be translated into $\mathrm{T}(\psi_1) -\!\!* (\mathrm{OK}(X) \Rightarrow \mathrm{T}(\psi_2))$ because the evaluation of $\mathrm{T}(\psi_1)$ in a disjoint heap may need the values of free variables occurring in ψ_1 but our encoding of the variable valuations via the heap does not allow to preserve these values through disjoint heaps. In order to solve this problem, for each variable $\mathtt{x}$ in the formula, X will contain an auxiliary variable $\overline{\mathtt{x}}$, or alternatively we define on X an involution $\overline{(.)}$. If the translated formula has q variables then the set X of variables needed for the translation will have cardinality $2q$. In the translation of a formula whose outermost connective is the magic wand, the locations corresponding to variables of the form $\overline{\mathtt{x}}$ will be allocated on the left side of the magic wand, and checked to be equal to their non-bar versions on the right side of the magic wand. As such, the left side of the magic wand will be translated into

$$((\bigwedge_{\mathtt{z} \in Z} \mathtt{alloc}(\overline{\mathtt{z}})) \wedge (\bigwedge_{\mathtt{z} \in X \setminus Z} \neg \mathtt{alloc}(\mathtt{z})) \wedge \mathrm{OK}(Z) \wedge \mathrm{T}(\psi_1)[\mathtt{z} \leftarrow \overline{\mathtt{z}} \mid \mathtt{z} \in X]),$$

where Z is the set of free variables in ψ_1, whereas the right side will be

$$(((\bigwedge_{\mathtt{z}\in Z} n(\mathtt{z}) = n(\overline{\mathtt{z}})) \wedge \mathrm{OK}(X)) \Rightarrow ((\bigwedge_{\mathtt{z}\in Z} \mathtt{alloc}(\overline{\mathtt{z}}) \wedge \mathtt{size} = \mathrm{card}(Z)) * \mathrm{T}(\psi_2))).$$

The use of the separating conjunction before the formula $\mathrm{T}(\psi_2)$ separates the memory cells corresponding to $\overline{\mathtt{x}}$ from the rest of the heap. By doing this, we can reuse $\overline{\mathtt{x}}$ whenever a magic wand appears in $\mathrm{T}(\psi_2)$.

For technical convenience, we consider a slight alternative for the semantics of the logics $\mathrm{SL}(\forall, \mathrel{-\!\!*})$ and $\mathrm{SL}(*, \mathrel{-\!\!*}, \mathtt{ls})$, which does not modify the notion of satisfiability/validity and such that the set of formulae and the definition of the satisfaction relation $\models$ remain unchanged. So far, the memory states are pairs of the form (s, h) with $s : \mathrm{PVAR} \to \mathrm{LOC}$ and $h : \mathrm{LOC} \to_{\mathrm{fin}} \mathrm{LOC}$ for a *fixed* countably infinite set of locations LOC, say $\mathrm{LOC} = \mathbb{N}$. Alternatively, the models for $\mathrm{SL}(\forall, \mathrel{-\!\!*})$ and $\mathrm{SL}(*, \mathrel{-\!\!*}, \mathtt{ls})$ can be defined as triples $(\mathrm{LOC}_1, s_1, h_1)$ such that LOC_1 is a countable infinite set, $s_1 : \mathrm{PVAR} \to \mathrm{LOC}_1$ and $h_1 : \mathrm{LOC}_1 \to_{\mathrm{fin}} \mathrm{LOC}_1$. As shown below, this does not change the notion of satisfiability and validity, but this generalisation will be handy in a few places. Most of the time, a generalised memory state $(\mathrm{LOC}_1, s_1, h_1)$ shall be written (s_1, h_1) when no confusion is possible.

Given a bijection $\mathfrak{f} : \mathrm{LOC}_1 \to \mathrm{LOC}_2$ and a heap $h_1 : \mathrm{LOC}_1 \to_{\mathrm{fin}} \mathrm{LOC}_1$ equal to $\{\ell_1 \mapsto h_1(\ell_1), \ldots, \ell_n \mapsto h_1(\ell_n)\}$, we write $\mathfrak{f}(h_1)$ to denote the heap $h_2 : \mathrm{LOC}_2 \to_{\mathrm{fin}} \mathrm{LOC}_2$ with $h_2 = \{\mathfrak{f}(\ell_1) \mapsto \mathfrak{f}(h_1(\ell_1)), \ldots, \mathfrak{f}(\ell_n) \mapsto \mathfrak{f}(h_1(\ell_n))\}$.

Definition 1. *Let* $(\mathrm{LOC}_1, s_1, h_1)$ *and* $(\mathrm{LOC}_2, s_2, h_2)$ *be generalised memory states and* $X \subseteq \mathrm{PVAR}$*. A* partial isomorphism *with respect to* X *from* $(\mathrm{LOC}_1, s_1, h_1)$ *to* $(\mathrm{LOC}_2, s_2, h_2)$ *is a bijection* $\mathfrak{f} : \mathrm{LOC}_1 \to \mathrm{LOC}_2$ *such that* $h_2 = \mathfrak{f}(h_1)$ *and for all* $\mathtt{x} \in X$*,* $\mathfrak{f}(s_1(\mathtt{x})) = s_2(\mathtt{x})$ *(we write* $(\mathrm{LOC}_1, s_1, h_1) \approx_X (\mathrm{LOC}_2, s_2, h_2)$*).*

A folklore result states that isomorphic memory states satisfy the same formulae since the logics $\mathrm{SL}(\forall, \mathrel{-\!\!*})$, $\mathrm{SL}(*, \mathrel{-\!\!*}, \mathtt{ls})$ can only perform equality tests.

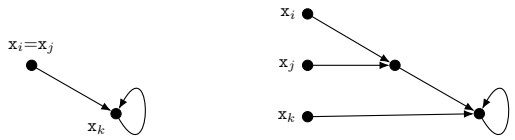
Lemma 1. *Let* $(\mathrm{LOC}_1, s_1, h_1)$ *and* $(\mathrm{LOC}_2, s_2, h_2)$ *be two generalised memory states such that* $(\mathrm{LOC}_1, s_1, h_1) \approx_X (\mathrm{LOC}_2, s_2, h_2)$*, for some* $X \subseteq \mathrm{PVAR}$*. (I) For all formulae* φ *in* $\mathrm{SL}(\forall, \mathrel{-\!\!*})$ *whose free variables are among* X*, we have* $(\mathrm{LOC}_1, s_1, h_1) \models \varphi$ *iff* $(\mathrm{LOC}_2, s_2, h_2) \models \varphi$*. (II) For all formulae* φ *in* $\mathrm{SL}(*, \mathrel{-\!\!*}, \mathtt{ls})$ *built on variables among* X*, we have* $(\mathrm{LOC}_1, s_1, h_1) \models \varphi$ *iff* $(\mathrm{LOC}_2, s_2, h_2) \models \varphi$*.*

As a direct consequence, satisfiability in $\mathrm{SL}(*, \mathrel{-\!\!*}, \mathtt{ls})$ as defined in Section 2, is equivalent to satisfiability with generalised memory states, the same holds for $\mathrm{SL}(\forall, \mathrel{-\!\!*})$. Next, we define the encoding of a generalised memory state. This can be seen as the semantical counterpart of the syntactical translation process and, as such, formalise the intuition of using part of a heap to mimic the store.

Definition 2. *Let* $X = \{\mathtt{x}_1, \ldots, \mathtt{x}_{2q}\}$*,* $Y \subseteq \{\mathtt{x}_1, \ldots, \mathtt{x}_q\}$ *and,* $(\mathrm{LOC}_1, s_1, h_1)$ *and* $(\mathrm{LOC}_2, s_2, h_2)$ *be two (generalised) memory states. We say that* $(\mathrm{LOC}_1, s_1, h_1)$ *is encoded by* $(\mathrm{LOC}_2, s_2, h_2)$ *w.r.t.* X, Y*, written* $(\mathrm{LOC}_1, s_1, h_1) \rhd_q^Y (\mathrm{LOC}_2, s_2, h_2)$*, if the following conditions hold:*

– $\mathrm{LOC}_1 = \mathrm{LOC}_2 \setminus \{s_2(\mathtt{x}) \mid \mathtt{x} \in X\}$,
– *for all* $\mathtt{x} \neq \mathtt{y} \in X$, $s_2(\mathtt{x}) \neq s_2(\mathtt{y})$,
– $h_2 = h_1 + \{s_2(\mathtt{x}) \mapsto s_1(\mathtt{x}) \mid \mathtt{x} \in Y\}$.

Notice that h_2 is equal to h_1 plus the heap $\{s_2(\mathtt{x}) \mapsto s_1(\mathtt{x}) \mid \mathtt{x} \in Y\}$ that encodes the store s_1. The picture below presents a memory state (left) and its encoding (right), where $Y = \{\mathtt{x}_i, \mathtt{x}_j, \mathtt{x}_k\}$. From the encoding, we can retrieve the initial heap by removing the memory cells corresponding to $\mathtt{x}_i$, $\mathtt{x}_j$ and $\mathtt{x}_k$. By way of example, the memory state on the left satisfies the formulae $\mathtt{x}_i = \mathtt{x}_j$, $\mathtt{x}_i \hookrightarrow \mathtt{x}_k$ and $\mathtt{x}_k \hookrightarrow \mathtt{x}_k$ whereas its encoding satisfies the formulae $n(\mathtt{x}_i) = n(\mathtt{x}_j)$, $n(\mathtt{x}_i) \hookrightarrow n(\mathtt{x}_k)$ and $n(\mathtt{x}_k) \hookrightarrow n(\mathtt{x}_k)$.



3.2 The translation

We are now ready to define the translation of a first-order formula in propositional separation logic extended with the three predicates introduced at the beginning of the section. Let φ be a closed formula of $\mathrm{SL}(\forall, \mathrel{-\!\!*})$ with quantified variables $\{\mathtt{x}_1, \ldots, \mathtt{x}_q\}$. W.l.o.g., we can assume that distinct quantifications involve distinct variables. Moreover, let $X = \{\mathtt{x}_1, \ldots, \mathtt{x}_{2q}\}$ and $\overline{(.)}$ be the involution on X such that for all $i \in [1, q]$ $\overline{\mathtt{x}}_i \stackrel{\text{def}}{=} \mathtt{x}_{i+q}$.

We write $\mathrm{OK}(X)$ to denote the formula $(\bigwedge_{i \neq j} \mathtt{x}_i \neq \mathtt{x}_j) \wedge (\bigwedge_i \neg \mathtt{alloc}^{-1}(\mathtt{x}_i))$. The translation function T has two arguments: the formula in $\mathrm{SL}(\forall, \mathrel{-\!\!*})$ to be recursively translated and the total set of variables potentially appearing in the target formula (useful to check that $\mathrm{OK}(X)$ holds on every heap involved in the satisfaction of the translated formula). Let us come back to the definition of $\mathrm{T}(\psi, X)$ (homomorphic for Boolean connectives) with the assumption that the variables in ψ are among $\mathtt{x}_1, \ldots, \mathtt{x}_q$.

$$\mathrm{T}(\mathtt{x}_i = \mathtt{x}_j, X) \stackrel{\text{def}}{=} n(\mathtt{x}_i) = n(\mathtt{x}_j)$$
$$\mathrm{T}(\mathtt{x}_i \hookrightarrow \mathtt{x}_j, X) \stackrel{\text{def}}{=} n(\mathtt{x}_i) \hookrightarrow n(\mathtt{x}_j)$$
$$\mathrm{T}(\forall \mathtt{x}_i\ \psi, X) \stackrel{\text{def}}{=} (\mathtt{alloc}(\mathtt{x}_i) \wedge \mathtt{size} = 1) \mathrel{-\!\!*} (\mathrm{OK}(X) \Rightarrow \mathrm{T}(\psi, X))$$

Lastly, the translation $\mathrm{T}(\psi_1 \mathrel{-\!\!*} \psi_2, X)$ is defined as

$$((\bigwedge_{\mathtt{z} \in Z} \mathtt{alloc}(\overline{\mathtt{z}})) \wedge (\bigwedge_{\mathtt{z} \in X \setminus Z} \neg \mathtt{alloc}(\overline{\mathtt{z}})) \wedge \mathrm{OK}(X) \wedge \mathrm{T}(\psi_1, X)[\mathtt{x} \leftarrow \overline{\mathtt{x}}]) \mathrel{-\!\!*}$$

$$(((\bigwedge_{\mathtt{z} \in Z} n(\mathtt{z}) = n(\overline{\mathtt{z}})) \wedge \mathrm{OK}(X)) \Rightarrow ((\bigwedge_{\mathtt{z} \in Z} \mathtt{alloc}(\overline{\mathtt{z}}) \wedge \mathtt{size} = \mathrm{card}(Z)) * \mathrm{T}(\psi_2, X))),$$

where $Z \subseteq \{\mathtt{x}_1, \ldots, \mathtt{x}_q\}$ is the set of free variables in ψ_1.

Here is the main result of this section, which is essential for the correctness of $\mathcal{T}_{\mathrm{SAT}}(\varphi)$, defined below.

Lemma 2. *Let $X = \{\mathtt{x}_1, \ldots, \mathtt{x}_{2q}\}$, $Y \subseteq \{\mathtt{x}_1, \ldots, \mathtt{x}_q\}$, ψ be a formula in $\mathrm{SL}(\forall, \mathrel{-\!\!*})$ with free variables among Y that does not contain any bound variable of ψ and $(\mathrm{LOC}_1, s_1, h_1) \rhd_q^Y (\mathrm{LOC}_2, s_2, h_2)$. We have $(s_1, h_1) \models \psi$ iff $(s_2, h_2) \models \mathrm{T}(\psi, X)$.*

We define the translation $\mathcal{T}_{\mathrm{SAT}}(\varphi)$ in $\mathrm{SL}(*, \mathrel{-\!\!*}, \mathtt{ls})$ where $\mathrm{T}(\varphi, X)$ is defined recursively.

$$\mathcal{T}_{\mathrm{SAT}}(\varphi) \stackrel{\text{def}}{=} (\bigwedge_{i \in [1,2q]} \neg \mathtt{alloc}(\mathtt{x}_i)) \wedge \mathrm{OK}(X) \wedge \mathrm{T}(\varphi, X).$$

The first two conjuncts specify initial conditions, namely each variable $\mathtt{y}$ in X is interpreted by a location that is unallocated, it is not in the heap range and it is distinct from the interpretation of all other variables; in other words, the value for $\mathtt{y}$ is isolated. Similarly, let $\mathcal{T}_{\mathrm{VAL}}(\varphi)$ be the formula in $\mathrm{SL}(*, \mathrel{-\!\!*}, \mathtt{ls})$ defined by $((\bigwedge_{i \in [1,2q]} \neg \mathtt{alloc}(\mathtt{x}_i)) \wedge \mathrm{OK}(X)) \Rightarrow \mathrm{T}(\varphi, X)$. As a consequence of Lemma 2, φ and $\mathcal{T}_{\mathrm{SAT}}(\varphi)$ are shown equisatisfiable, whereas φ and $\mathcal{T}_{\mathrm{VAL}}(\varphi)$ are shown equivalid.

Corollary 1. *Let φ be a closed formula in $\mathrm{SL}(\forall, \mathrel{-\!\!*})$ using quantified variables among $\{\mathtt{x}_1, \ldots, \mathtt{x}_q\}$. (I) φ and $\mathcal{T}_{\mathrm{SAT}}(\varphi)$ are equisatisfiable. (II) φ and $\mathcal{T}_{\mathrm{VAL}}(\varphi)$ are equivalid.*

3.3 Expressing the auxiliary atomic predicates

To complete the reduction, we briefly explain how to express the formulae $\mathtt{alloc}^{-1}(\mathtt{x})$, $n(\mathtt{x}) = n(\mathtt{y})$ and $n(\mathtt{x}) \hookrightarrow n(\mathtt{y})$ within $\mathrm{SL}(*, \mathrel{-\!\!*}, \mathtt{ls})$. Let us introduce a few macros that shall be helpful.

- Given φ in $\mathrm{SL}(*, \mathrel{-\!\!*}, \mathtt{reach}^+)$ and $\gamma \geq 0$, we write $[\varphi]_\gamma$ to denote the formula $(\mathtt{size} = \gamma \wedge \varphi) * \top$. It is easy to show that for any memory state (s, h), $(s, h) \models [\varphi]_\gamma$ iff there is $h' \sqsubseteq h$ such that $\mathrm{card}(\mathrm{dom}(h')) = \gamma$ and $(s, h') \models \varphi$.
- We write $\mathtt{reach}(\mathtt{x}, \mathtt{y}) = \gamma$ to denote the formula $[\mathtt{ls}(\mathtt{x}, \mathtt{y})]_\gamma$, which is satisfied in any memory state (s, h) where $h^\gamma(s(\mathtt{x})) = s(\mathtt{y})$. Lastly, we write $\mathtt{reach}(\mathtt{x}, \mathtt{y}) \leq \gamma$ to denote the formula $\bigvee_{0 \leq \gamma' \leq \gamma} \mathtt{reach}(\mathtt{x}, \mathtt{y}) = \gamma'$.

In order to define the existence of a predecessor (i.e. $\mathtt{alloc}^{-1}(\mathtt{x})$) in $\mathrm{SL}(*, \mathrel{-\!\!*}, \mathtt{ls})$, we need to take advantage of an auxiliary variable $\mathtt{y}$ whose value is different from the one for $\mathtt{x}$. Let $\mathtt{alloc}^{-1}_{\mathtt{y}}(\mathtt{x})$ be the formula

$$\mathtt{x} \hookrightarrow \mathtt{x} \vee \mathtt{y} \hookrightarrow \mathtt{x} \vee [(\mathtt{alloc}(\mathtt{y}) \wedge \neg(\mathtt{y} \hookrightarrow \mathtt{x}) \wedge \mathtt{size} = 1) \circledast \mathtt{reach}(\mathtt{y}, \mathtt{x}) = 2]_1$$

Lemma 3. *Let $\mathtt{x}, \mathtt{y} \in \mathrm{PVAR}$. (I) For all memory states (s, h) such that $s(\mathtt{x}) \neq s(\mathtt{y})$, we have $(s, h) \models \mathtt{alloc}^{-1}_{\mathtt{y}}(\mathtt{x})$ iff $s(\mathtt{x}) \in \mathrm{ran}(h)$. (II) In the translation, $\mathtt{alloc}^{-1}(\mathtt{x})$ can be replaced with $\mathtt{alloc}^{-1}_{\mathtt{x}}(\mathtt{x})$.*

As stated in Lemma 3(II), we can exploit the fact that in the translation of a formula with variables in $\{\mathtt{x}_1, \ldots, \mathtt{x}_q\}$, we use $2q$ variables that correspond to $2q$ distinguished locations in the heap in order to retain the soundness of the translation while using $\mathtt{alloc}^{-1}_{\overline{\mathtt{x}}}(\mathtt{x})$ as $\mathtt{alloc}^{-1}(\mathtt{x})$. Moreover, $\mathtt{alloc}^{-1}_{\mathtt{y}}(\mathtt{x})$ allows to express in $\mathrm{SL}(*, -\!\!*, \mathtt{ls})$ whenever a location corresponding to a program variable reaches itself in exactly two steps (we use this property in the definition of $n(\mathtt{x}) \hookrightarrow n(\mathtt{y})$). We write $\mathtt{x} \hookrightarrow^2_{\mathtt{y}} \mathtt{x}$ to denote the formula $\neg(\mathtt{x} \hookrightarrow \mathtt{x}) \wedge (\mathtt{x} \hookrightarrow \mathtt{y} \Leftrightarrow \mathtt{y} \hookrightarrow \mathtt{x}) \wedge [\mathtt{alloc}(\mathtt{x}) \wedge \mathtt{alloc}^{-1}_{\mathtt{y}}(\mathtt{x}) \wedge (\top -\!\!* \neg\mathtt{reach}(\mathtt{x}, \mathtt{y}) = 2)]_2$. For any memory state (s, h) such that $s(\mathtt{x}) \neq s(\mathtt{y})$, we have $(s, h) \models \mathtt{x} \hookrightarrow^2_{\mathtt{y}} \mathtt{x}$ if and only if $h^2(s(\mathtt{x})) = s(\mathtt{x})$ and $h(s(\mathtt{x})) \neq s(\mathtt{x})$.

The predicate $n(\mathtt{x}) = n(\mathtt{y})$ can be defined in $\mathrm{SL}(*, -\!\!*, \mathtt{ls})$ as

$$(\mathtt{x} \neq \mathtt{y} \Rightarrow [\mathtt{alloc}(\mathtt{x}) \wedge \mathtt{alloc}(\mathtt{y}) \wedge ((\mathtt{x} \hookrightarrow \mathtt{y} \wedge \mathtt{y} \hookrightarrow \mathtt{y}) \vee (\mathtt{y} \hookrightarrow \mathtt{x} \wedge \mathtt{x} \hookrightarrow \mathtt{x}) \vee$$
$$((\bigwedge_{\mathtt{z},\mathtt{z}' \in \{\mathtt{x},\mathtt{y}\}} \neg(\mathtt{z} \hookrightarrow \mathtt{z}')) \wedge (\top -\!\!* \neg(\mathtt{reach}(\mathtt{x}, \mathtt{y}) = 2 \wedge \mathtt{reach}(\mathtt{y}, \mathtt{x}) = 2))))]_2) \wedge \mathtt{alloc}(\mathtt{x})$$

Lemma 4. *Let* $\mathtt{x}, \mathtt{y} \in \mathrm{PVAR}$. *For all memory states* (s, h), *we have* $(s, h) \models n(\mathtt{x}) = n(\mathtt{y})$ *iff* $h(s(\mathtt{x})) = h(s(\mathtt{y}))$.

Similarly to $\mathtt{alloc}^{-1}(\mathtt{x})$, we can show that $n(\mathtt{x}) \hookrightarrow n(\mathtt{y})$ is definable in $\mathrm{SL}(*, -\!\!*, \mathtt{ls})$ by using one additional variable $\mathtt{z}$ whose value is different from both $\mathtt{x}$ and $\mathtt{y}$. Let $\varphi_{\hookrightarrow}(\mathtt{x}, \mathtt{y}, \mathtt{z})$ be $(n(\mathtt{x}) = n(\mathtt{y}) \wedge \varphi^{=}_{\hookrightarrow}(\mathtt{x}, \mathtt{y}, \mathtt{z})) \vee (n(\mathtt{x}) \neq n(\mathtt{y}) \wedge \varphi^{\neq}_{\hookrightarrow}(\mathtt{x}, \mathtt{y}))$ where $\varphi^{=}_{\hookrightarrow}(\mathtt{x}, \mathtt{y}, \mathtt{z})$ is defined as

$$(\mathtt{x} \hookrightarrow \mathtt{x} \wedge \mathtt{y} \hookrightarrow \mathtt{x}) \vee (\mathtt{y} \hookrightarrow \mathtt{y} \wedge \mathtt{x} \hookrightarrow \mathtt{y}) \vee (\mathtt{x} \hookrightarrow \mathtt{z} \wedge \mathtt{z} \hookrightarrow \mathtt{z})$$
$$\vee [\mathtt{alloc}(\mathtt{x}) \wedge \neg\mathtt{alloc}^{-1}_{\mathtt{z}}(\mathtt{x}) \wedge (\top -\!\!* \neg\mathtt{reach}(\mathtt{x}, \mathtt{z}) \leq 3)]_2$$

whereas $\varphi^{\neq}_{\hookrightarrow}(\mathtt{x}, \mathtt{y})$ is defined as

$$(\mathtt{x} \hookrightarrow \mathtt{y} \wedge \mathtt{alloc}(\mathtt{y})) \vee (\mathtt{y} \hookrightarrow \mathtt{y} \wedge \mathtt{reach}(\mathtt{x}, \mathtt{y}) = 2) \vee (\mathtt{y} \hookrightarrow \mathtt{x} \wedge \mathtt{x} \hookrightarrow^2_{\mathtt{y}} \mathtt{x}) \vee$$
$$[\mathtt{alloc}(\mathtt{x}) \wedge \mathtt{alloc}(\mathtt{y}) \wedge (\textstyle\bigwedge_{\mathtt{z},\mathtt{z}' \in \{\mathtt{x},\mathtt{y}\}} \neg\mathtt{z} \hookrightarrow \mathtt{z}') \wedge \neg\mathtt{reach}(\mathtt{x}, \mathtt{y}) \leq 3$$
$$\wedge((\mathtt{size} = 1 \wedge \mathtt{alloc}^{-1}_{\mathtt{x}}(\mathtt{y})) \mathrel{-\!\!\circledast} (\mathtt{reach}(\mathtt{x}, \mathtt{y}) = 3 \wedge \mathtt{y} \hookrightarrow^2_{\mathtt{x}} \mathtt{y}))]_3$$

Lemma 5. *Let* $\mathtt{x}, \mathtt{y}, \mathtt{z} \in \mathrm{PVAR}$. *(I) For all memory states* (s, h) *such that* $s(\mathtt{x}) \neq s(\mathtt{z})$ *and* $s(\mathtt{y}) \neq s(\mathtt{z})$, *we have* $(s, h) \models \varphi_{\hookrightarrow}(\mathtt{x}, \mathtt{y}, \mathtt{z})$ *iff* $\{s(\mathtt{x}), s(\mathtt{y})\} \subseteq \mathrm{dom}(h)$ *and* $h(h(s(\mathtt{x}))) = h(s(\mathtt{y}))$; *(II) In the translation,* $n(\mathtt{x}) \hookrightarrow n(\mathtt{y})$ *can be replaced by* $\varphi_{\hookrightarrow}(\mathtt{x}, \mathtt{y}, \overline{\mathtt{x}})$.

As for $\mathtt{alloc}^{-1}_{\mathtt{y}}(\mathtt{x})$, the properties of the translation imply the equivalence between $n(\mathtt{x}) \hookrightarrow n(\mathtt{y})$ and $\varphi_{\hookrightarrow}(\mathtt{x}, \mathtt{y}, \overline{\mathtt{x}})$ (as stated in Lemma 5(II)). By looking at the formulae herein defined, the predicate $\mathtt{reach}$ only appears bounded, i.e. in the form of $\mathtt{reach}(\mathtt{x}, \mathtt{y}) = 2$ and $\mathtt{reach}(\mathtt{x}, \mathtt{y}) = 3$. The three new predicates can therefore be defined in $\mathrm{SL}(*, -\!\!*)$ enriched with $\mathtt{reach}(\mathtt{x}, \mathtt{y}) = 2$ and $\mathtt{reach}(\mathtt{x}, \mathtt{y}) = 3$.

3.4 Undecidability results and non-finite axiomatization

It is time to collect the fruits of all our efforts and to conclude this part about undecidability. As a direct consequence of Corollary 1 and the undecidability of $\mathrm{SL}(\forall, -\!\!*)$, here is one of the main results of the paper.

Theorem 1. *The satisfiability problem for* $\mathrm{SL}(*, -\!\!*, \mathtt{ls})$ *is undecidable.*

As a by-product, the set of valid formulae for $\mathrm{SL}(*, -\!\!*, \mathtt{ls})$ is not recursively enumerable. Indeed, suppose that the set of valid formulae for $\mathrm{SL}(*, -\!\!*, \mathtt{ls})$ were r.e., then one can enumerate the valid formulae of the form $\mathcal{T}_{\mathrm{VAL}}(\varphi)$ as it is decidable in PTIME whether ψ in $\mathrm{SL}(*, -\!\!*, \mathtt{ls})$ is syntactically equal to $\mathcal{T}_{\mathrm{VAL}}(\varphi)$ for some $\mathrm{SL}(\forall, -\!\!*)$ formula φ. This leads to a contradiction since this would allow the enumeration of valid formulae in $\mathrm{SL}(\forall, -\!\!*)$.

The essential ingredients to establish the undecidability of $\mathrm{SL}(*, -\!\!*, \mathtt{ls})$ are the fact that the following properties $n(\mathtt{x}) = n(\mathtt{y})$, $n(\mathtt{x}) \hookrightarrow n(\mathtt{y})$ and $\mathtt{alloc}^{-1}(\mathtt{x})$ are expressible in the logic.

Corollary 2. $\mathrm{SL}(*, -\!\!*)$ *augmented with built-in formulae of the form* $n(\mathtt{x}) = n(\mathtt{y})$, $n(\mathtt{x}) \hookrightarrow n(\mathtt{y})$ *and* $\mathtt{alloc}^{-1}(\mathtt{x})$ *(resp. of the form* $\mathtt{reach}(\mathtt{x}, \mathtt{y}) = 2$ *and* $\mathtt{reach}(\mathtt{x}, \mathtt{y}) = 3$*) admits an undecidable satisfiability problem.*

This is the addition of $\mathtt{reach}(\mathtt{x}, \mathtt{y}) = 3$ that is crucial for undecidability since the satisfiability problem for $\mathrm{SL}(*, -\!\!*, \mathtt{reach}(\mathtt{x}, \mathtt{y}) = 2)$ is in PSPACE [15]. Following a similar analysis, let $\mathrm{SL1}(\forall, *, -\!\!*)$ be the restriction of $\mathrm{SL}(\forall, *, -\!\!*)$ (i.e. $\mathrm{SL}(\forall, -\!\!*)$ plus $*$) to formulae of the form $\exists \mathtt{x}_1 \cdots \exists \mathtt{x}_q \, \varphi$, where $q \geq 1$, the variables in φ are among $\{\mathtt{x}_1, \ldots, \mathtt{x}_{q+1}\}$ and the only quantified variable in φ is $\mathtt{x}_{q+1}$. The satisfiability problem for $\mathrm{SL1}(\forall, *, -\!\!*)$ is PSPACE-complete [15]. Note that $\mathrm{SL1}(\forall, *, -\!\!*)$ can easily express $n(\mathtt{x}) = n(\mathtt{y})$ and $\mathtt{alloc}^{-1}(\mathtt{x})$. The distance between the decidability for $\mathrm{SL1}(\forall, *, -\!\!*)$ and the undecidability for $\mathrm{SL}(*, -\!\!*, \mathtt{ls})$, is best witnessed by the corollary below, which solves an open problem [15, Section 6].

Corollary 3. *SL1*$(\forall, *, -\!\!*)$ *augmented with* $n(\mathtt{x}) \hookrightarrow n(\mathtt{y})$ *(resp. SL1*$(\forall, *, -\!\!*)$ *augmented with* $\mathtt{ls}$*) admits an undecidable satisfiability problem.*

4 $\mathrm{SL}(*, \mathtt{reach}^{+})$ and other PSPACE variants

As already seen in Section 2, $\mathrm{SL}(*, \mathtt{ls})$ can be understood as a fragment of $\mathrm{SL}(*, \mathtt{reach}^{+})$. Below, we show that the satisfiability problem for $\mathrm{SL}(*, \mathtt{reach}^{+})$ can be solved in polynomial space. Refining the arguments used in our proof, we also show the decidability of the fragment of $\mathrm{SL}(*, -\!\!*, \mathtt{reach}^{+})$ where $\mathtt{reach}^{+}$ is constrained not to occur in the scope of $-\!\!*$, i.e. φ belongs to that fragment iff for any subformula ψ of φ of the form $\psi_1 -\!\!* \psi_2$, $\mathtt{reach}^{+}$ does not occur in ψ_1 and in ψ_2.

The proof relies on a small heap property: a formula φ is satisfiable if and only if it admits a model with a polynomial amount of memory cells. The PSPACE upper bound then follows by establishing that the model-checking problem for $\mathrm{SL}(*, \mathtt{reach}^{+})$ is in PSPACE too. To establish the small heap property, an equivalence relation on memory states with finite index is designed, following the standard approach in [32,10] and using test formulae as in [23,22,4,15].

4.1 Introduction to test formulae

Before presenting the test formulae for SL($*, \mathtt{reach}^+$), let us recall the standard result for SL($*, -\!\!*$) (that will be also used at some point later on).

Proposition 2. *[32,22] Any formula φ in* SL($*, -\!\!*$) *built over variables in* $\mathtt{x}_1$, *...,*$\mathtt{x}_q$ *is logically equivalent to a Boolean combination of formulae among* $\mathtt{x}_i = \mathtt{x}_j$, $\mathtt{alloc}(\mathtt{x}_i)$, $\mathtt{x}_i \hookrightarrow \mathtt{x}_j$ *and* $\mathtt{size} \geq \beta$ *($i, j \in \{1, \ldots, q\}$, $\beta \in \mathbb{N}$).*

By way of example, $\big(\neg\mathtt{emp} * \big((\mathtt{x}_1 \hookrightarrow \mathtt{x}_2) -\!\!* \bot\big)\big)$ is equivalent to $\mathtt{size} \geq 2 \wedge \mathtt{alloc}(\mathtt{x}_1)$. As a corollary of the proof of Proposition 2, in $\mathtt{size} \geq \beta$ we can enforce that $\beta \leq 2 \times |\varphi|$ (rough upper bound) where $|\varphi|$ is the size of φ. Similar results will be shown for SL($*, \mathtt{reach}^+$) and for some of its extensions.

In order to define a set of test formulae that captures the expressive power of SL($*, \mathtt{reach}^+$), we need to study which basic properties on memory states can be expressed by SL($*, \mathtt{reach}^+$) formulae. For example, consider the memory states from Figure 1.

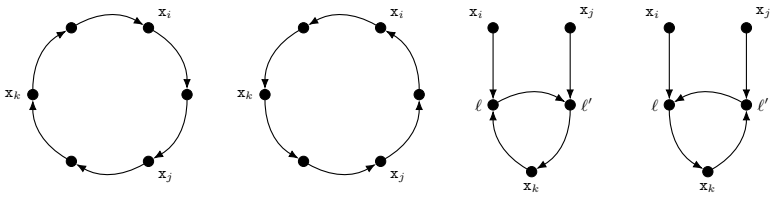


Fig. 1. Memory states (s_1, h_1), ..., (s_4, h_4) (from left to right)

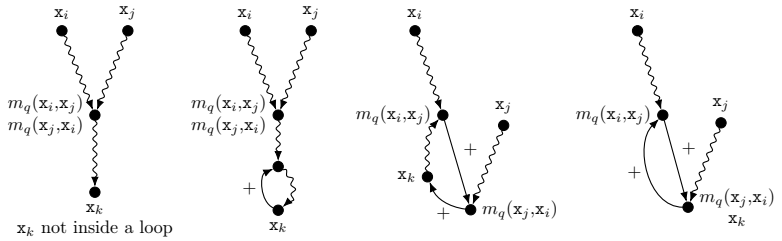
The fragment memory states (s_1, h_1) and (s_2, h_2) can be distinguished by the formula $\top * (\mathtt{reach}(\mathtt{x}_i, \mathtt{x}_j) \wedge \mathtt{reach}(\mathtt{x}_j, \mathtt{x}_k) \wedge \neg\mathtt{reach}(\mathtt{x}_k, \mathtt{x}_i))$. Indeed, (s_1, h_1) satisfies this formula by considering a subheap that does not contain a path from $s(\mathtt{x}_k)$ to $s(\mathtt{x}_i)$, whereas it is impossible to find a subheap for (s_2, h_2) that retains the path from $s(\mathtt{x}_i)$ to $s(\mathtt{x}_j)$, the one from $s(\mathtt{x}_j)$ to $s(\mathtt{x}_k)$ but where the path from $s(\mathtt{x}_k)$ to $s(\mathtt{x}_i)$ is lost. This suggests that SL($*, \mathtt{reach}^+$) can express whether, for example, any path from $s(\mathtt{x}_i)$ to $s(\mathtt{x}_j)$ also contains $s(\mathtt{x}_k)$. We will introduce the test formula $\mathtt{sees}_q(\mathtt{x}_i, \mathtt{x}_j) \geq \beta$ to capture this property.

Similarly, the memory states (s_3, h_3) and (s_4, h_4) can be distinguished by the formula $(\mathtt{size} = 1) * \big(\mathtt{reach}(\mathtt{x}_j, \mathtt{x}_k) \wedge \neg\mathtt{reach}(\mathtt{x}_i, \mathtt{x}_k) \wedge \neg\mathtt{reach}^+(\mathtt{x}_k, \mathtt{x}_k)\big)$. The memory state (s_3, h_3) satisfies this formula by separating $\{\ell \mapsto \ell'\}$ from the rest of the heap, whereas the formula is not satisfied by (s_4, h_4). Indeed, there is no way to break the loop from $s(\mathtt{x}_k)$ to itself by removing just one location from the heap while retaining the path from $s(\mathtt{x}_j)$ to $s(\mathtt{x}_k)$ and loosing the path from $s(\mathtt{x}_i)$ to $s(\mathtt{x}_k)$. This suggests that the two locations ℓ and ℓ' are particularly interesting since they are reachable from several locations corresponding to program variables. Therefore by separating them from the rest of the heap, several paths are lost. In order to capture this, we introduce the notion of *meet-points*.

Let Terms_q be the set $\{\mathtt{x}_1, \ldots, \mathtt{x}_q\} \cup \{m_q(\mathtt{x}_i, \mathtt{x}_j) \mid i, j \in [1, q]\}$ understood as the set of *terms* that are either variables or expressions denoting a meet-point. We write $[\![\mathtt{x}_i]\!]^q_{s,h}$ to denote $s(\mathtt{x}_i)$ and $[\![m_q(\mathtt{x}_i, \mathtt{x}_j)]\!]^q_{s,h}$ to denote (if it exists) the first location reachable from $s(\mathtt{x}_i)$ that is also reachable from $s(\mathtt{x}_j)$. Moreover we require that this location can reach another location corresponding to a program variable. Formally, $[\![m_q(\mathtt{x}_i, \mathtt{x}_j)]\!]^q_{s,h}$ is defined as the unique location ℓ such that

- there are $L_1, L_2 \geq 0$ such that $h^{L_1}(s(\mathtt{x}_i)) = h^{L_2}(s(\mathtt{x}_j)) = \ell$, and
- for all $L'_1 < L_1$ and for all $L'_2 \geq 0$, $h^{L'_1}\big(s(\mathtt{x}_i)\big) \neq h^{L'_2}\big(s(\mathtt{x}_j)\big)$, and
- there exist $k \in [1, q]$ and $L \geq 0$ such that $h^L(\ell) = s(\mathtt{x}_k)$.

These conditions hold for at most one location ℓ. One can easily show that the notion $[\![m_q(\mathtt{x}_i, \mathtt{x}_j)]\!]^q_{s,h}$ is well-defined. The picture below provides a taxonomy of meet-points, where arrows labelled by '+' represent paths of non-zero length and zig-zag arrows any path (possibly of zero length). Symmetrical cases, obtained by swapping $\mathtt{x}_i$ and $\mathtt{x}_j$, are omitted.



Notice how the asymmetrical definition of meet-points is captured in the two rightmost heaps. Consider the memory states from Figure 1, (s_3, h_3) and (s_4, h_4) can be seen as an instance of the third case of the taxonomy and, as such, it holds that $[\![m_q(\mathtt{x}_i, \mathtt{x}_j)]\!]^q_{s_3,h_3} = \ell$ and $[\![m_q(\mathtt{x}_j, \mathtt{x}_i)]\!]^q_{s_3,h_3} = \ell'$.

Given $q, \alpha \geq 1$, we write $\mathrm{Test}(q, \alpha)$ to denote the following set of atomic formulae (also called *test formulae*):

$$v = v' \quad v \hookrightarrow v' \quad \mathtt{alloc}(v) \quad \mathtt{sees}_q(v, v') \geq \beta + 1 \quad \mathtt{sizeR}_q \geq \beta,$$

where $v, v' \in \mathrm{Terms}_q$ and $\beta \in [1, \alpha]$. It is worth noting that the $\mathtt{alloc}(v)$'s are not needed for the logic $\mathrm{SL}(*, \mathtt{reach}^+)$ but it is required for extensions.

We identify as special locations the $s(\mathtt{x}_i)$'s and the meet-points of the form $[\![m_q(\mathtt{x}_i, \mathtt{x}_j)]\!]^q_{s,h}$ when it exists ($i, j \in [1, q]$). We call such locations, *labelled* locations, and the set of labelled locations is written $\mathrm{Labels}^q_{s,h}$. The formal semantics of the test formulae is provided below:

$$\begin{array}{lcl}
(s,h) \models v = v' & \iff & [\![v]\!]^q_{s,h}, [\![v']\!]^q_{s,h} \text{ are defined}, [\![v]\!]^q_{s,h} = [\![v']\!]^q_{s,h} \\
(s,h) \models \mathtt{alloc}(v) & \iff & [\![v]\!]^q_{s,h} \text{ is defined and belongs to } \mathrm{dom}(h) \\
(s,h) \models v \hookrightarrow v' & \iff & h([\![v]\!]^q_{s,h}) = [\![v']\!]^q_{s,h} \\
(s,h) \models \mathtt{sees}_q(v, v') \geq \beta + 1 & \iff & \exists L \geq \beta + 1,\ h^L([\![v]\!]^q_{s,h}) = [\![v']\!]^q_{s,h} \text{ and} \\
 & & \forall\ 0 < L' < L,\ h^{L'}([\![v]\!]^q_{s,h}) \notin \mathrm{Labels}^q_{s,h} \\
(s,h) \models \mathtt{sizeR}_q \geq \beta & \iff & \mathrm{card}(\mathrm{Rem}^q_{s,h}) \geq \beta
\end{array}$$

where $\mathrm{Rem}^q_{s,h}$ is the set of locations that neither belong to a path between two locations interpreted by program variables nor are equal to program variable interpretations, i.e. $\mathrm{Rem}^q_{s,h} \stackrel{\text{def}}{=} \{\ell \in \mathrm{dom}(h) \mid \forall i \in [1,q],\ s(\mathtt{x}_i) \neq \ell \text{ and } \forall j \in [1,q]\ \nexists L, L' \geq 1, h^L(s(\mathtt{x}_i)) = \ell \text{ and } h^{L'}(\ell) = s(\mathtt{x}_j)\}$. There is no need for test formulae of the form $\mathtt{sees}_q(v,v') \geq 1$ since they are equivalent to $v \hookrightarrow v' \vee \mathtt{sees}_q(v,v') \geq 2$. One can check whether $[\![m_q(\mathtt{x}_i,\mathtt{x}_j)]\!]^q_{s,h}$ is defined thanks to the formula $m_q(\mathtt{x}_i,\mathtt{x}_j) = m_q(\mathtt{x}_i,\mathtt{x}_j)$. By contrast, $\mathtt{sizeR}_q \geq \beta$ states that the cardinality of the set $\mathrm{Rem}^q_{s,h}$ is at least β. Furthermore, $\mathtt{sees}_q(v,v') \geq \beta+1$ states that there is a minimal path between v and v' of length at least $\beta+1$ and strictly between v and v', there are no labelled locations. The satisfaction of $\mathtt{sees}_q(v,v') \geq \beta+1$ entails the exclusion of labelled locations in the witness path, which is reminiscent to $T \xrightarrow{h \setminus T''} T'$ in the logic GRASS [26]. So, the test formulae are quite expressive since they capture the atomic formulae from $\mathrm{SL}(*, \mathtt{reach}^+)$ and the test formulae for $\mathrm{SL}(*, -\!\!*)$.

Lemma 6. *Given $\alpha, q \geq 1$, $i,j \in [1,q]$, for any atomic formula among $\mathtt{ls}(\mathtt{x}_i,\mathtt{x}_j)$, $\mathtt{reach}(\mathtt{x}_i,\mathtt{x}_j)$, $\mathtt{reach}^+(\mathtt{x}_i,\mathtt{x}_j)$, $\mathtt{emp}$ and $\mathtt{size} \geq \beta$ with $\beta \leq \alpha$, there is a Boolean combination of test formulae from $\mathrm{Test}(q,\alpha)$ logically equivalent to it.*

4.2 Expressive power and small model property

The sets of test formulae $\mathrm{Test}(q,\alpha)$ are sufficient to capture the expressive power of $\mathrm{SL}(*, \mathtt{reach}^+)$ (as shown below, Theorem 2) and deduce the small heap property of this logic (Theorem 3). We introduce an indistinguishability relation between memory states based on test formulae, see analogous relations in [22,13,15].

Definition 3. *Given $q, \alpha \geq 1$, we write $(s,h) \approx^q_\alpha (s',h') \stackrel{\text{def}}{\Leftrightarrow}$ for all $\psi \in \mathrm{Test}(q,\alpha)$, we have $(s,h) \models \psi$ iff $(s',h') \models \psi$.*

Theorem 2(I) states that if $(s,h) \approx^q_\alpha (s',h')$, then the two memory states cannot be distinguished by formulae whose syntactic resources are bounded in some way by q and α (details will follow, see the definition for $\mathtt{msize}(\varphi)$).

Below, we state the key intermediate result of the section that can be viewed as a distributivity lemma. The expressive power of the test formulae allows us to mimic the separation between two equivalent memory states with respect to the relation $\approx^q_\alpha$, which is essential in the proof of Theorem 2(I).

Lemma 7. *Let $q, \alpha, \alpha_1, \alpha_2 \geq 1$ with $\alpha = \alpha_1 + \alpha_2$ and (s,h), (s',h') be such that $(s,h) \approx^q_\alpha (s',h')$. For all heaps h_1, h_2 such that $h = h_1 + h_2$ there are heaps h'_1, h'_2 such that $h = h'_1 + h'_2$, $(s,h_1) \approx^q_{\alpha_1} (s',h'_1)$ and $(s,h_2) \approx^q_{\alpha_2} (s',h'_2)$.*

For each formula φ in $\mathrm{SL}(*, \mathtt{reach}^+)$, we define its *memory size* $\mathtt{msize}(\varphi)$ following the clauses below (see also [32]).

$$\begin{aligned}
\mathtt{msize}(\pi) &\stackrel{\text{def}}{=} 1 \quad \text{for any atomic formula } \pi \\
\mathtt{msize}(\psi * \psi') &\stackrel{\text{def}}{=} \mathtt{msize}(\psi) + \mathtt{msize}(\psi') \\
\mathtt{msize}(\psi \wedge \psi') &\stackrel{\text{def}}{=} \max(\mathtt{msize}(\psi), \mathtt{msize}(\psi')) \\
\mathtt{msize}(\neg\psi) &\stackrel{\text{def}}{=} \mathtt{msize}(\psi).
\end{aligned}$$

We have $1 \leq \mathtt{msize}(\varphi) \leq |\varphi|$. Theorem 2 below establishes the properties that formulae in $\mathrm{SL}(*, \mathtt{reach}^+)$ can express.

Theorem 2. *Let φ be in $\mathrm{SL}(*, \mathtt{reach}^+)$ built over the variables in $\mathtt{x}_1, \ldots, \mathtt{x}_q$. (I) For all $\alpha \geq 1$ such that $\mathtt{msize}(\varphi) \leq \alpha$ and for all memory states (s,h), (s',h') such that $(s,h) \approx^q_\alpha (s',h')$, we have $(s,h) \models \varphi$ iff $(s',h') \models \varphi$. (II) φ is logically equivalent to a Boolean combination of test formulae from $\mathrm{Test}(q, \mathtt{msize}(\varphi))$.*

The proof of Theorem 2(I) is by structural induction on φ. The basic cases for atomic formulae follow from Lemma 6 whereas the inductive cases for Boolean connectives are immediate. For the separating conjunction, suppose $(s,h) \models \varphi_1 * \varphi_2$ and $\mathtt{msize}(\varphi_1 * \varphi_2) \leq \alpha$. There are heaps h_1 and h_2 such that $h = h_1 + h_2$, $(s,h_1) \models \psi_1$ and $(s,h_2) \models \psi_2$. As $\alpha \geq \mathtt{msize}(\psi_1 * \psi_2) = \mathtt{msize}(\psi_1) + \mathtt{msize}(\psi_2)$, there exist α_1 and α_2 such that $\alpha = \alpha_1 + \alpha_2$, $\alpha_1 \geq \mathtt{msize}(\psi_1)$ and $\alpha_2 \geq \mathtt{msize}(\psi_2)$. By Lemma 7, there exist heaps h'_1 and h'_2 such that $h' = h'_1 + h'_2$, $(s,h_1) \approx^q_{\alpha_1} (s',h'_1)$ and $(s,h_2) \approx^q_{\alpha_2} (s',h'_2)$. By the induction hypothesis, we get $(s',h'_1) \models \psi_1$ and $(s',h'_2) \models \psi_2$. Consequently, we obtain $(s',h') \models \psi_1 * \psi_2$.

As an example, we can apply this result to the memory states from Figure 1. We have already shown how we can distinguish (s_1,h_1) from (s_2,h_2) using a formula with only one separating conjunction. Theorem 2 ensures that these two memory states do not satisfy the same set of test formulae for $\alpha \geq 2$. Indeed, only (s_1,h_1) satisfies $\mathtt{sees}_q(\mathtt{x}_i, \mathtt{x}_j) \geq 2$. The same argument can be used with (s_3,h_3) and (s_4,h_4): only (s_3,h_3) satisfies the test formula $m_q(\mathtt{x}_i, \mathtt{x}_j) \hookrightarrow m_q(\mathtt{x}_j, \mathtt{x}_i)$. Clearly, Theorem 2(II) relates separation logic with classical logic as advocated also in the works [23,10]. Now, it is possible to establish a small heap property.

Theorem 3. *Let φ be a satisfiable $\mathrm{SL}(*, \mathtt{reach}^+)$ formula built over $\mathtt{x}_1, \ldots, \mathtt{x}_q$. There is (s,h) such that $(s,h) \models \varphi$ and $\mathrm{card}(\mathrm{dom}(h)) \leq (q^2+q) \cdot (|\varphi|+1) + |\varphi|$.*

The small heap property for $\mathrm{SL}(*, \mathtt{reach}^+)$ is inherited from the small heap property for the Boolean combinations of test formulae, which is analogous to the small model property for other theories of singly linked lists, see e.g. [27,13].

4.3 Complexity upper bounds

Let us draw some consequences of Theorem 3. First, for the logic $\mathrm{SL}(*, \mathtt{reach}^+)$, we get a PSPACE upper, which matches the lower bound for $\mathrm{SL}(*)$ [11].

Theorem 4. *The satisfiability problem for $\mathrm{SL}(*, \mathtt{reach}^+)$ is PSPACE-complete.*

Besides, we may consider restricting the usage of Boolean connectives. We note $\mathsf{Bool}(\mathsf{SHF})$ for the Boolean combinations of formulae from the symbolic heap fragment [2]. A PTIME upper bound for the entailment/satisfiability problem for the symbolic heap fragment is successfully solved in [12,17], whereas the satisfiability problem for a slight variant of $\mathsf{Bool}(\mathsf{SHF})$ is shown in NP in [26, Theorem 4]. Theorem 3 allows us to conclude this NP upper bound result as a by-product (we conjecture that our quadratic upper bound on the number of cells could be improved to a linear one in that case).

Corollary 4. *The satisfiability problem for* Bool(SHF) *is* NP*-complete.*

It is possible to push further the PSPACE upper bound by allowing occurrences of $-\!\!*$ in a controlled way. Let $\mathrm{SL}(*, \mathtt{reach}^+, \bigcup_{q,\alpha} \mathrm{Test}(q,\alpha))$ be the extension of $\mathrm{SL}(*, \mathtt{reach}^+)$ augmented with the test formulae. The memory size function is also extended: $\mathtt{msize}(v \hookrightarrow v') \stackrel{\text{def}}{=} 1$, $\mathtt{msize}(\mathtt{sees}_q(v,v') \geq \beta + 1) \stackrel{\text{def}}{=} \beta + 1$, $\mathtt{msize}(\mathtt{sizeR} \geq \beta) \stackrel{\text{def}}{=} \beta$ and $\mathtt{msize}(\mathtt{alloc}(v)) \stackrel{\text{def}}{=} 1$. When formulae are encoded as trees, we have $1 \leq \mathtt{msize}(\varphi) \leq |\varphi|\alpha_\varphi$ where α_φ is the maximal constant in φ. Theorem 2(I) admits a counterpart for $\mathrm{SL}(*, \mathtt{reach}^+, \bigcup_{q,\alpha} \mathrm{Test}(q,\alpha))$ and consequently, any formula built over $\mathtt{x}_1, \ldots, \mathtt{x}_q$ can be shown equivalent to a Boolean combination of test formulae from $\mathrm{Test}(q, |\varphi|\alpha_\varphi)$. By Theorem 3, any satisfiable formula has therefore a model with $\mathrm{card}(\mathrm{dom}(h)) \leq (q^2+q)\cdot(|\varphi|\alpha_\varphi+1)+|\varphi|\alpha_\varphi$. Hence, the satisfiability problem for $\mathrm{SL}(*, \mathtt{reach}^+, \bigcup_{q,\alpha} \mathrm{Test}(q,\alpha))$ is in PSPACE when the constants are encoded in unary. Now, we can state the new PSPACE upper bound for Boolean combinations of formulae from $\mathrm{SL}(*, -\!\!*) \cup \mathrm{SL}(*, \mathtt{reach}^+)$.

Theorem 5. *The satisfiability problem for Boolean combinations of formulae from* $\mathrm{SL}(*, -\!\!*) \cup \mathrm{SL}(*, \mathtt{reach}^+)$ *is* PSPACE*-complete.*

To conclude, let us introduce the largest fragment including $-\!\!*$ and $\mathtt{ls}$ for which decidability can be established so far.

Theorem 6. *The satisfiability problem for the fragment of* $\mathrm{SL}(*, -\!\!*, \mathtt{reach}^+)$ *in which* $\mathtt{reach}^+$ *is not in the scope of* $-\!\!*$ *is decidable.*

5 Conclusion

We studied the effects of adding $\mathtt{ls}$ to $\mathrm{SL}(*, -\!\!*)$ and variants. $\mathrm{SL}(*, -\!\!*, \mathtt{ls})$ is shown undecidable (Theorem 1) and non-finitely axiomatisable, which remains quite unexpected since there are no first-order quantifications. This result is strengthened to even weaker extensions of $\mathrm{SL}(*, -\!\!*)$ such as the one augmented with $n(\mathtt{x}) = n(\mathtt{y})$, $n(\mathtt{x}) \hookrightarrow n(\mathtt{y})$ and $\mathtt{alloc}^{-1}(\mathtt{x})$, or the one augmented with $\mathtt{reach}(\mathtt{x},\mathtt{y}) = 2$ and $\mathtt{reach}(\mathtt{x},\mathtt{y}) = 3$. If the magic wand is discarded, we have established that the satisfiability problem for $\mathrm{SL}(*, \mathtt{ls})$ is PSPACE-complete by introducing a class of test formulae that captures the expressive power of $\mathrm{SL}(*, \mathtt{ls})$ and that leads to a small heap property. Such a logic contains the Boolean combinations of symbolic heaps and our proof technique allows us to get an NP upper bound for such formulae. Moreover, we show that the satisfiability problem for $\mathrm{SL}(*, -\!\!*, \mathtt{reach}^+)$ restricted to formulae in which $\mathtt{reach}^+$ is not in the scope of $-\!\!*$ is decidable, leading to the largest known decidable fragment for which $-\!\!*$ and $\mathtt{reach}^+$ (or $\mathtt{ls}$) cohabit. So, we have provided proof techniques to establish undecidability when $*$, $-\!\!*$ and $\mathtt{ls}$ are present and to establish decidability based on test formulae. This paves the way to investigate the decidability status of $\mathrm{SL}(-\!\!*, \mathtt{ls})$ as well as of the positive fragment of $\mathrm{SL}(*, -\!\!\circledast, \mathtt{ls})$ from [30,31].

References

1. T. Antonopoulos, N. Gorogiannis, C. Haase, M. Kanovich, and J. Ouaknine. Foundations for decision problems in separation logic with general inductive predicates. In *FOSSACS'14*, volume 8412 of *LNCS*, pages 411–425. Springer, 2014.
2. J. Berdine, C. Calcagno, and P. O'Hearn. A decidable fragment of separation logic. In *FSTTCS'04*, volume 3328 of *LNCS*, pages 97–109. Springer, 2004.
3. J. Berdine, C. Calcagno, and P. O'Hearn. Smallfoot: Modular automatic assertion checking with separation logic. In *FMCO'05*, volume 4111 of *LNCS*, pages 115–137. Springer, 2005.
4. R. Brochenin, S. Demri, and E. Lozes. Reasoning about sequences of memory states. *APAL*, 161(3):305–323, 2009.
5. R. Brochenin, S. Demri, and E. Lozes. On the almighty wand. *IC*, 211:106–137, 2012.
6. J. Brotherston, C. Fuhs, N. Gorogiannis, and J. Navarro Perez. A decision procedure for satisfiability in separation logic with inductive predicates. In *CSL-LICS'14*, 2014.
7. J. Brotherston and J. Villard. Parametric completeness for separation theories. In *POPL'14*, pages 453–464. ACM, 2014.
8. C. Calcagno and D. Distefano. Infer: An automatic program verifier for memory safety of C programs. In *NASA Formal Methods*, volume 6617 of *LNCS*, pages 459–465. Springer, 2011.
9. C. Calcagno, D. Distefano, P. O'Hearn, and H. Yang. Compositional shape analysis by means of bi-abduction. *JACM*, 58(6):26:1–26:66, 2011.
10. C. Calcagno, P. Gardner, and M. Hague. From separation logic to first-order logic. In *FOSSACS'05*, volume 3441 of *LNCS*, pages 395–409. Springer, 2005.
11. C. Calcagno, P. O'Hearn, and H. Yang. Computability and complexity results for a spatial assertion language for data structures. In *FSTTCS'01*, volume 2245 of *LNCS*, pages 108–119. Springer, 2001.
12. B. Cook, C. Haase, J. Ouaknine, M. Parkinson, and J. Worrell. Tractable reasoning in a fragment of separation logic. In *CONCUR'11*, volume 6901 of *LNCS*, pages 235–249. Springer, 2011.
13. C. David, D. Kroening, and M. Lewis. Propositional reasoning about safety and termination of heap-manipulating programs. In *ESOP'15*, volume 9032 of *LNCS*, pages 661–684. Springer, 2015.
14. S. Demri and M. Deters. Expressive completeness of separation logic with two variables and no separating conjunction. *ACM ToCL*, 17(2):12, 2016.
15. S. Demri, D. Galmiche, D. Larchey-Wendling, and D. Mery. Separation logic with one quantified variable. *Theory of Computing Systems*, 61:371–461, 2017.
16. D. Distefano, P. O'Hearn, and H. Yang. A local shape analysis based on separation logic. In *TACAS'06*, volume 3920 of *LNCS*, pages 287–302. Springer, 2006.
17. C. Haase, S. Ishtiaq, J. Ouaknine, and M. Parkinson. SeLoger: A tool for graph-based reasoning in separation logic. In *CAV'13*, volume 8044 of *LNCS*, pages 790–795. Springer, 2013.
18. Z. Hou, R. Goré, and A. Tiu. Automated theorem proving for assertions in separation logic with all connectives. In *CADE'15*, volume 9195 of *LNCS*, pages 501–516. Springer, 2015.
19. R. Iosif, A. Rogalewicz, and J. Simacek. The tree width of separation logic with recursive definitions. In *CADE'13*, volume 7898 of *LNCS*, pages 21–38. Springer, 2013.

20. S. Ishtiaq and P. O'Hearn. BI as an assertion language for mutable data structures. In *POPL'01*, pages 14–26. ACM, 2001.
21. Q. Le, M. Tatsuta, J. Sun, and W. Chin. A decidable fragment in separation logic with inductive predicates and arithmetic. In *CAV'17*, volume 10427 of *LNCS*, pages 495–517. Springer, 2017.
22. E. Lozes. *Expressivité des Logiques Spatiales*. Phd thesis, ENS Lyon, 2004.
23. E. Lozes. Separation logic preserves the expressive power of classical logic. In *SPACE'04*, 2004.
24. P. Müller, M. Schwerhoff, and A. Summers. Viper: A verification infrastructure for permission-based reasoning. In *VMCAI'16*, volume 9583 of *LNCS*, pages 41–62. Springer, 2016.
25. P. O'Hearn, J. Reynolds, and H. Yang. Local reasoning about programs that alter data structures. In *CSL'01*, volume 2142 of *LNCS*, pages 1–19. Springer, 2001.
26. R. Piskać, T. Wies, and D. Zufferey. Automating separation logic using SMT. In *CAV'13*, volume 8044 of *LNCS*, pages 773–789. Springer, 2013.
27. S. Ranise and C. Zarba. A theory of singly-linked lists and its extensible decision procedure. In *SEFM'06*, pages 206–215. IEEE, 2006.
28. J. Reynolds. Separation logic: a logic for shared mutable data structures. In *LICS'02*, pages 55–74. IEEE, 2002.
29. M. Schwerhoff and A. Summers. Lightweight support for magic wands in an automatic verifier. In *ECOOP'15*, pages 999–1023. Leibniz-Zentrum für Informatik, LIPICS, 2015.
30. A. Thakur. *Symbolic Abstraction: Algorithms and Applications*. PhD thesis, University of Wisconsin-Madison, 2014.
31. A. Thakur, J. Breck, and T. Reps. Satisfiability modulo abstraction for separation logic with linked lists. In *SPIN'14*, pages 58–67. ACM, 2014.
32. H. Yang. *Local Reasoning for Stateful Programs*. PhD thesis, University of Illinois, Urbana-Champaign, 2001.
33. H. Yang, O. Lee, J. Berdine, C. Calcagno, B. Cook, D. Distefano, and P. O'Hearn. Scalable shape analysis for systems code. In *CAV'08*, volume 5123 of *LNCS*, pages 385–398. Springer, 2008.