The Effects of Adding Reachability Predicates in Quantifier-Free Separation Logic

STÉPHANE DEMRI, Université Paris-Saclay, ENS Paris-Saclay, CNRS, LSV, 91190, Gif-sur-Yvette, France
ETIENNE LOZES, I3S, Université Côte d'Azur, France
ALESSIO MANSUTTI, Université Paris-Saclay, ENS Paris-Saclay, CNRS, LSV, 91190, Gif-sur-Yvette, France

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 quantifier-free 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 quantifier-free separation logic.

CCS Concepts: • **Theory of computation → Separation logic**;

Additional Key Words and Phrases: separation logic, reachability, decision problems, complexity, quantifier elimination.

ACM Reference Format:
Stéphane Demri, Etienne Lozes, and Alessio Mansutti. 2021. The Effects of Adding Reachability Predicates in Quantifier-Free Separation Logic. *ACM Trans. Comput. Logic* 1, 1, Article 1 (January 2021), 55 pages. https://doi.org/10.1145/3448269

1 INTRODUCTION

Separation logic [25, 33, 36] 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, 42], many tools supporting separation logic as an assertion language have been developed [3, 9, 10, 20, 22, 42]. Even though the first tools could handle relatively limited fragments of separation logic, like symbolic heaps [13], 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 tree-width fragment [1, 24], whereas satisfiability is ExpTime-complete [7]. Other related results can be found in [26, 27].

Authors' addresses: Stéphane Demri, Université Paris-Saclay, ENS Paris-Saclay, CNRS, LSV, 91190, Gif-sur-Yvette, France, demri@lsv.fr; Etienne Lozes, I3S, Université Côte d'Azur, France, elozes@i3s.unice.fr; Alessio Mansutti, Université Paris-Saclay, ENS Paris-Saclay, CNRS, LSV, 91190, Gif-sur-Yvette, France, alessio.mansutti@lsv.fr.

- 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 [12].
- Quantifier-free separation logic with separating implication, a.k.a. magic wand ($-\!\!*$), is a convenient fragment (called herein SL($*, -\!\!*$)) with decidable frame inference and abduction, two problems 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 [41].

A natural question is how to combine these extensions, and which separation logic fragment admitting Boolean connectives, magic wand and generalised recursive predicates can be decided with some adequate restrictions. As already advocated in [8, 23, 32, 38, 40], 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. [23, 32, 38, 40].

Besides, allowing quantifications is another direction to extend the symbolic heap fragment: in [4], an extension of the symbolic heap fragment with quantification over locations and over arithmetic variables for list lengths is introduced and several fragments are shown decidable (the whole extension is undecidable). Such an extension combines shape and arithmetic specifications (see also [14] for a theory of singly-linked lists with length combining such features) and the decidability results are obtained by using so-called symbolic shape graphs that are finite representations of sets of heaps. In the current paper, we consider only shape analysis (since herein, the heaps are restricted to a single record field) but the separating implication is admitted.

Our contribution. In this paper, we address the question of combining the magic wand and inductive predicates in the extremely limited case where the only inductive predicate is the gentle list segment predicate $\mathtt{ls}$. The starting point of this work is this puzzling question: what is the complexity/decidability status of quantifier-free 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 quantifier-free 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). At this point, it is worth noting that in the presence of the separating conjunction $*$, $\mathtt{ls}$ and $\mathtt{reach}$ are interdefinable, and $\mathtt{reach}^+$ can easily define $\mathtt{ls}$ and $\mathtt{reach}$. Consequently, the complexity upper bounds will be stated with $\mathtt{reach}^+$ and the complexity lower bounds or undecidability results are sharper with $\mathtt{ls}$ or $\mathtt{reach}$.

First, we establish that the satisfiability problem for the quantifier-free separation logic SL($*, -\!\!*, \mathtt{ls}$) is undecidable. Our proof is by reduction from the undecidability of first-order separation logic SL($\forall, -\!\!*$) [6, 16], using an encoding of the variables as heap cells (see Theorem 3.12). 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 [17].

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. [5, 17, 21, 28–30], and the PSPACE-hardness of SL($*$) is inherited from [12]. 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). As a by-product of our proof technique, we obtain that the satisfiability problem for

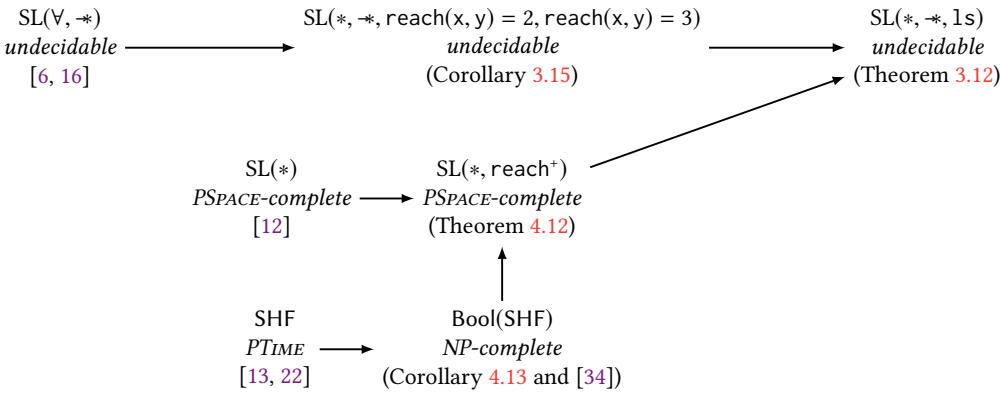


Fig. 1. Main contributions.

Boolean combinations of pure formulae and spatial formulae from the symbolic heap fragment, Bool(SHF), is NP-complete via a proof different from the one in [34]. Figure 1 presents a summary of the main results of the paper. An unlabelled arrow between two logics means that there is a many-one reduction between the satisfiability problem of the first logic and the problem for the second one (sometimes, the reduction is the identity in the case of syntactic fragments).

This paper is an extended and completed version of [18].

2 PRELIMINARIES

2.1 Separation logic with the list segment predicate

Let PVAR = $\{\mathsf{x}, \mathsf{y}, \ldots\}$ 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.

Let h_1 and h_2 be two heaps. h_1 and h_2 are said to be *disjoint*, written $h_1 \bot h_2$, whenever their domains are disjoint, i.e. $\text{dom}(h_1) \cap \text{dom}(h_2) = \emptyset$. When $h_1 \bot h_2$, 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 $\text{dom}(h_1 + h_2) = \text{dom}(h_1) \uplus \text{dom}(h_2)$. If the domains of h_1 and h_2 are not disjoint, then the union $h_1 + h_2$ is not defined. We say that h_1 is a *subheap* of h_2, written $h_1 \sqsubseteq h_2$, if $\text{dom}(h_1) \subseteq \text{dom}(h_2)$ and for all $\ell \in \text{dom}(h_1)$, we have $h_1(\ell) = h_2(\ell)$. For instance, if $h_1 \bot h_2$ then $h_1 \sqsubseteq h_1 + h_2$. Given a heap h, we write h^i for the (partial) function obtained from i functional composition(s) of h. By definition, h^0 is the identity function on LOC, $h^1 \stackrel{\text{def}}{=} h$ and for all $\beta \geq 2$ and $\ell \in \text{LOC}$, we have $h^\beta(\ell) \stackrel{\text{def}}{=} h(h^{\beta-1}(\ell))$, assuming that $h^{\beta-1}(\ell)$ is defined and belongs to $\text{dom}(h)$, otherwise $h^\beta(\ell)$ is undefined.

The formulae φ of the separation logic SL(∗, -∗, ls) and its atomic formulae π are built from the grammars below (where $\mathsf{x}, \mathsf{y} \in \text{PVAR}$ and the connectives $\Rightarrow$, $\Leftrightarrow$ and $\vee$ are defined as usually).

$$\pi ::= \mathsf{emp} \mid \mathsf{x} = \mathsf{y} \mid \mathsf{x} \hookrightarrow \mathsf{y} \mid \mathsf{ls}(\mathsf{x}, \mathsf{y})$$

$$\varphi ::= \pi \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi * \varphi \mid \varphi \mathrel{-\!\!*} \varphi$$

Models of SL(∗, -∗, ls) are memory states and the satisfaction relation $\models$ is defined as follows:

$$
\begin{array}{lcl}
(s,h) \models \mathtt{emp} & \iff & \mathrm{dom}(h) = \emptyset. \\
(s,h) \models \mathtt{x} = \mathtt{y} & \iff & s(\mathtt{x}) = s(\mathtt{y}). \\
(s,h) \models \mathtt{x} \hookrightarrow \mathtt{y} & \iff & s(\mathtt{x}) \in \mathrm{dom}(h) \text{ and } h(s(\mathtt{x})) = s(\mathtt{y}). \\
(s,h) \models \mathtt{ls}(\mathtt{x},\mathtt{y}) & \iff & \text{either } (\mathrm{dom}(h) = \emptyset \text{ and } s(\mathtt{x}) = s(\mathtt{y})) \text{ or} \\
 & & h = \{\ell_0 \mapsto \ell_1, \ell_1 \mapsto \ell_2, \ldots, \ell_{n-1} \mapsto \ell_n\} \text{ for some } n \geq 1, \\
 & & \ell_0 = s(\mathtt{x}),\ \ell_n = s(\mathtt{y}) \text{ and for all } i \neq j \in [0,n],\ \ell_i \neq \ell_j. \\
(s,h) \models \neg\varphi & \iff & (s,h) \not\models \varphi. \\
(s,h) \models \varphi_1 \wedge \varphi_2 & \iff & (s,h) \models \varphi_1 \text{ and } (s,h) \models \varphi_2. \\
(s,h) \models \varphi_1 * \varphi_2 & \iff & \text{there are } h_1 \text{ and } h_2 \text{ such that } h_1 \perp h_2,\ h_1 + h_2 = h, \\
 & & (s,h_1) \models \varphi_1 \text{ and } (s,h_2) \models \varphi_2. \\
(s,h) \models \varphi_1 \mathrel{-\!\!*} \varphi_2 & \iff & \text{for all } h_1 \text{ such that } h_1 \perp h \text{ and } (s,h_1) \models \varphi_1, \text{ we have } (s, h + h_1) \models \varphi_2.
\end{array}
$$

The semantics for $*$ (separating conjunction), $\mathrel{-\!\!*}$ (separating implication), $\hookrightarrow$ (points to), $\mathtt{ls}$ and for all other ingredients is the usual one in separation logic, where $\mathtt{ls}(\mathtt{x},\mathtt{y})$ is the *precise list segment predicate* stating that $h^i(s(\mathtt{x})) = s(\mathtt{y})$ for some $i \in \mathbb{N}$, but this property does not hold in any strict subheap of h (which excludes the presence of cycles).

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,
- $\mathtt{size} = \beta \stackrel{\text{def}}{=} (\mathtt{size} \leq \beta) \wedge (\mathtt{size} \geq \beta)$.

It is easy to see that $(s,h) \models \mathtt{size} \geq \beta$ iff $\mathrm{card}(\mathrm{dom}(h)) \geq \beta$, where $\mathrm{card}(X)$ denotes the cardinality of a (finite) set X. We introduce the *septraction connective* $\circledast$, defined as $\varphi_1 \circledast \varphi_2 \stackrel{\text{def}}{=} \neg(\varphi_1 \mathrel{-\!\!*} \neg\varphi_2)$. The connective $\circledast$ can be viewed as a form of dual operator for the separating implication. So, $(s,h) \models \varphi_1 \circledast \varphi_2$ iff there is some heap h_1 disjoint from h such that $(s,h_1) \models \varphi_1$ and $(s, h + h_1) \models \varphi_2$. We introduce also the following standard abbreviations:

$$\mathtt{alloc}(\mathtt{x}) \stackrel{\text{def}}{=} (\mathtt{x} \hookrightarrow \mathtt{x}) \mathrel{-\!\!*} \bot \qquad\qquad \mathtt{x} \mapsto \mathtt{y} \stackrel{\text{def}}{=} (\mathtt{x} \hookrightarrow \mathtt{y}) \wedge \mathtt{size} = 1.$$

It holds that $(s,h) \models \mathtt{alloc}(\mathtt{x})$ iff $s(\mathtt{x}) \in \mathrm{dom}(h)$, whereas $(s,h) \models \mathtt{x} \mapsto \mathtt{y}$ iff $\mathrm{dom}(h) = \{s(\mathtt{x})\}$ and $h(s(\mathtt{x})) = s(\mathtt{y})$. Without loss of generality, we assume that $\mathrm{LOC} = \mathbb{N}$ (see also Section 3.1). We write $\mathrm{SL}(*, \mathrel{-\!\!*})$ to denote the restriction of $\mathrm{SL}(*, \mathrel{-\!\!*}, \mathtt{ls})$ without $\mathtt{ls}$. Similarly, $\mathrm{SL}(*)$ denotes the restriction of $\mathrm{SL}(*, \mathrel{-\!\!*})$ without $\mathrel{-\!\!*}$ and $\mathrm{SL}(\mathrel{-\!\!*})$ denotes its restriction without $*$. Given two formulae φ, φ' (possibly from different logical languages), we write $\varphi \equiv \varphi'$ whenever for all memory states (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*.

2.2 Variants with other reachability predicates

We use two additional reachability predicates $\mathtt{reach}(\mathtt{x},\mathtt{y})$ and $\mathtt{reach}^+(\mathtt{x},\mathtt{y})$. We write $\mathrm{SL}(*, \mathrel{-\!\!*}, \mathtt{reach})$ (resp. $\mathrm{SL}(*, \mathrel{-\!\!*}, \mathtt{reach}^+)$) to denote the variant of $\mathrm{SL}(*, \mathrel{-\!\!*}, \mathtt{ls})$ in which $\mathtt{ls}$ is replaced by $\mathtt{reach}$ (resp. by $\mathtt{reach}^+$). The satisfaction relation $\models$ is extended as follows:

- $(s,h) \models \mathtt{reach}(\mathtt{x},\mathtt{y})$ holds when there is $i \geq 0$ such that $h^i(s(\mathtt{x})) = s(\mathtt{y})$,
- $(s,h) \models \mathtt{reach}^+(\mathtt{x},\mathtt{y})$ holds when there is $i \geq 1$ such that $h^i(s(\mathtt{x})) = s(\mathtt{y})$.

When the heap h is understood as a directed graph with a finite relation, $\mathtt{reach}(\mathtt{x},\mathtt{y})$ corresponds to the standard reachability predicate (differently from $\mathtt{ls}$, which also imposes constraints on strict subheaps). $\mathtt{reach}^+(\mathtt{x},\mathtt{y})$ corresponds instead to the reachability predicate in at least one step. For instance, $\mathtt{reach}(\mathtt{x},\mathtt{x})$ always holds, whereas $\mathtt{ls}(\mathtt{x},\mathtt{x})$ holds only on the empty heap and $\mathtt{reach}^+(\mathtt{x},\mathtt{x})$ holds on heaps such that there is $i \geq 1$ with $h^i(s(\mathtt{x})) = s(\mathtt{x})$, i.e. there is a non-empty path (cycle) from $s(\mathtt{x})$ to itself.

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 SL($*, -\!\!*, \mathtt{reach}$) and SL($*, -\!\!*, \mathtt{ls}$) have identical decidability status. Besides, these two logics can be seen as fragments of SL($*, -\!\!*, \mathtt{reach}^+$), thanks to the equivalence below:

$$\mathtt{reach}(\mathtt{x},\mathtt{y}) \equiv \mathtt{x} = \mathtt{y} \vee \mathtt{reach}^+(\mathtt{x},\mathtt{y}).$$

Notice that this analysis can be carried out as soon as $*$, $\neg$, $\wedge$ and $\mathtt{emp}$ are parts of the logic (none of the equivalences above uses the separating implication $-\!\!*$). More specifically, SL($*, \mathtt{reach}$) and SL($*, \mathtt{ls}$) have the same decidability status, and can be viewed as fragments of 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.

2.3 Decision problems

Let $\mathfrak{L}$ be a logic defined above. As usual, the *satisfiability problem for* $\mathfrak{L}$ takes as an input a formula φ from $\mathfrak{L}$ and asks whether there is a memory state (s,h) satisfying it, i.e. $(s,h) \models \varphi$. The *validity problem for* $\mathfrak{L}$ asks instead whether φ is satisfied by every memory state. If $\mathfrak{L}$ is not closed under negation, then it is also worth considering the *entailment problem* that takes as inputs two formulae φ and φ', and asks whether for all the memory states (s,h), we have $(s,h) \models \varphi$ implies $(s,h) \models \varphi'$ (written $\varphi \models \varphi'$).

The *model-checking problem for* $\mathfrak{L}$ takes as an input a formula φ from $\mathfrak{L}$ and a finite representation of a memory state (s,h), and asks whether $(s,h) \models \varphi$. Notice that, given a formula φ, it is easy to find a finite representation of (s,h): it is sufficient to restrict s to the variables occurring in φ and to encode h as a finite directed graph such that each vertex has at most one outgoing vertex. Unless otherwise specified, the *size* of a formula φ is understood as the size of its syntax tree. Moreover, the size of the finite representation of a memory state (s,h) is defined naturally considering a reasonably succinct encoding of s (only interpreting the program variables in φ) and the graph representation of h.

Below, we recall a few complexity results about well-known strict fragments of SL($*, -\!\!*, \mathtt{ls}$).

Proposition 2.1.

(I) *The satisfiability problem is PSpace-complete for both* SL($*$) *and* SL($*, -\!\!*$) *[12].*
(II) *The satisfiability and entailment problems for the symbolic heap fragment are in PTime [13].*
(III) *The satisfiability problem for the fragment of* SL($*, \mathtt{ls}$) *restricted to formulae obtained by Boolean combinations of formulae from the symbolic heap fragment is NP-complete [13, 34].*

We refer the reader to [13] for a complete description of the symbolic heap fragment, or to Section 4.3 for its definition. The main purpose of this paper is to study the decidability/complexity status of SL($*, -\!\!*, \mathtt{ls}$), as well as for its fragments and variants.

3 UNDECIDABILITY OF THE SATISFIABILITY PROBLEM FOR SL($*, -\!\!*, \mathtt{ls}$)

In this section, we show that SL($*, -\!\!*, \mathtt{ls}$) has an undecidable satisfiability problem even though it does not admit *explicitly* first-order quantification. We stress the word "explicitly" as we show that SL($*, -\!\!*, \mathtt{ls}$) can simulate the first-order quantification from SL($\forall, -\!\!*$): the first-order extension of SL($-\!\!*$), studied in [6, 16].

In the versions of SL($\forall, -\!\!*$) defined in [6, 16], one distinguishes program variables from quantified variables (used with the first-order quantification $\forall$). This distinction made by the two sets of variables is not necessary herein and for the sake of simplicity, we adopt a version of SL($\forall, -\!\!*$) with a unique type of variables. The formulae φ of SL($\forall, -\!\!*$) are built from the grammars below:

$$\pi ::= \mathtt{x} = \mathtt{y} \mid \mathtt{x} \hookrightarrow \mathtt{y}$$

$$\varphi ::= \pi \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \mathrel{-\!\!*} \varphi \mid \forall \mathsf{x}\, \varphi,$$

where $\mathsf{x}, \mathsf{y} \in \mathrm{PVAR}$. Models of the logic $\mathrm{SL}(\forall, \mathrel{-\!\!*})$ are memory states and the satisfaction relation $\models$ is defined as for $\mathrm{SL}(\mathrel{-\!\!*})$ with the additional clause:

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

where $s[\mathsf{x} \leftarrow \ell]$ is defined from the store s by only changing that x takes the value ℓ. Note that $\mathtt{emp}$ can be easily defined by $\neg\exists\, \mathsf{x}\ (\mathsf{x} \hookrightarrow \mathsf{x} \mathrel{-\!\!*} \bot)$. Without any loss of generality, we can assume that the satisfiability (resp. validity) problem for $\mathrm{SL}(\forall, \mathrel{-\!\!*})$ is defined by taking as inputs closed formulae (i.e. without free occurrences of the variables).

Proposition 3.1. *[6, 16] The satisfiability problem for* $\mathrm{SL}(\forall, \mathrel{-\!\!*})$ *is undecidable and the set of valid formulae for* $\mathrm{SL}(\forall, \mathrel{-\!\!*})$ *is not recursively enumerable.*

We recall that the undecidability proof of $\mathrm{SL}(\forall, \mathrel{-\!\!*})$ makes extensive use of the operator $\mathrel{-\!\!*}$, whereas a similar result can be achieved without $\mathrel{-\!\!*}$ if we interpret the logic on heaps having at least two record fields (i.e. h is of the form $\mathrm{LOC} \rightharpoonup_{\mathrm{fin}} \mathrm{LOC}^k$ with $k \geq 2$) [12]. In a nutshell, we establish the undecidability of $\mathrm{SL}(*, \mathrel{-\!\!*}, \mathtt{ls})$ by a reduction from the satisfiability problem for $\mathrm{SL}(\forall, \mathrel{-\!\!*})$. The reduction is nicely decomposed in two intermediate steps: (1) the undecidability of $\mathrm{SL}(*, \mathrel{-\!\!*})$ 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 $\mathrm{SL}(*, \mathrel{-\!\!*}, \mathtt{ls})$. In particular, Section 3.2 explains how the additional predicates can be used to encode stores as subheaps, and how this helps to mimic first-order quantification. Section 3.3 provides the formal presentation of the translation as well as its correctness. Section 3.4 establishes how the additional predicates can be indeed expressed in $\mathrm{SL}(*, \mathrel{-\!\!*}, \mathtt{ls})$. Finally, Section 3.5 provides the concluding results about undecidability and refines the results of the previous sections.

3.1 Generalised memory states

For technical convenience, in order to reduce (the satisfiability problem of) $\mathrm{SL}(\forall, \mathrel{-\!\!*})$ to $\mathrm{SL}(*, \mathrel{-\!\!*}, \mathtt{ls})$ we consider a slight alternative for the semantics of these two logics, 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 $h : \mathrm{LOC} \rightharpoonup_{\mathrm{fin}} \mathrm{LOC}$ for a *fixed* countably infinite set of locations LOC, e.g. $\mathbb{N}$. 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 countably infinite set, $s_1 : \mathrm{PVAR} \to \mathrm{LOC}_1$ and $h_1 : \mathrm{LOC}_1 \rightharpoonup_{\mathrm{fin}} \mathrm{LOC}_1$. 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 $h_1 : \mathrm{LOC}_1 \rightharpoonup_{\mathrm{fin}} \mathrm{LOC}_1$ that can be represented by $\{\ell_1 \mapsto h_1(\ell_1), \ldots, \ell_n \mapsto h_1(\ell_n)\}$, we write $\mathfrak{f}(h_1)$ to denote the heap $h_2 : \mathrm{LOC}_2 \rightharpoonup_{\mathrm{fin}} \mathrm{LOC}_2$ 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 3.2. Let $(\mathrm{LOC}_1, s_1, h_1)$ and $(\mathrm{LOC}_2, s_2, h_2)$ be generalised memory states and $X \subseteq \mathrm{PVAR}$. An *$X$-isomorphism* 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 $\mathsf{x} \in X$, $\mathfrak{f}(s_1(\mathsf{x})) = s_2(\mathsf{x})$.

Note that if $\mathfrak{f}$ is an X-isomorphism from $(\mathrm{LOC}_1, s_1, h_1)$ to $(\mathrm{LOC}_2, s_2, h_2)$, then $\mathfrak{f}^{-1}$ is also an X-isomorphism from $(\mathrm{LOC}_2, s_2, h_2)$ to $(\mathrm{LOC}_1, s_1, h_1)$. Two generalised memory states $(\mathrm{LOC}_1, s_1, h_1)$ and $(\mathrm{LOC}_2, s_2, h_2)$ are said to be *isomorphic with respect to X*, written $(\mathrm{LOC}_1, s_1, h_1) \approx_X (\mathrm{LOC}_2, s_2, h_2)$, if and only if there exists an X-isomorphism between them.

It is easy to check that $\approx_X$ is an equivalence relation. A folklore result states that isomorphic memory states satisfy the same formulae, which implies that considering generalised memory states over (standard) memory states does not change the notion of satisfiability and validity. Below,

we state the precise result we need in the sequel, the proof being by a standard induction on the formula structure.

Lemma 3.3. *Let* $(\mathrm{LOC}_1, s_1, h_1)$, $(\mathrm{LOC}_2, s_2, h_2)$ *be generalised memory states and* $X \subseteq \mathrm{PVAR}$ *be a finite set of variables such that* $(\mathrm{LOC}_1, s_1, h_1) \approx_X (\mathrm{LOC}_2, s_2, h_2)$. *Given* φ *in* $\mathrm{SL}(*, -\!\!*, \mathtt{ls})$ *or* $\mathrm{SL}(\forall, -\!\!*)$, *with free variables from* X, $(\mathrm{LOC}_1, s_1, h_1) \models \varphi$ *iff* $(\mathrm{LOC}_2, s_2, h_2) \models \varphi$.

Proof. The proof is by induction on the tree structure of φ. Let X be a set of variables that includes the free variables from φ. To be more concise, this is done on formulae from $\mathrm{SL}(\forall, *, -\!\!*, \mathtt{ls})$. Let $\mathfrak{f} : \mathrm{LOC}_1 \to \mathrm{LOC}_2$ be a bijection defined as in Definition 3.2. Since $\approx_X$ is an equivalence relation, showing one direction suffices to prove the result. We start by considering the base case with $\mathtt{ls}(\mathtt{x}, \mathtt{y})$, the cases for $\mathtt{x} = \mathtt{y}$ and $\mathtt{x} \hookrightarrow \mathtt{y}$ being omitted as this poses no difficulty.

base case: $\mathtt{ls}(\mathtt{x}, \mathtt{y})$. If $(\mathrm{LOC}_1, s_1, h_1) \models \mathtt{ls}(\mathtt{x}, \mathtt{y})$, then either ($\mathrm{dom}(h_1) = \emptyset$ and $s_1(\mathtt{x}) = s_1(\mathtt{y})$) or $h_1 = \{\ell_0 \mapsto \ell_1, \ell_1 \mapsto \ell_2, \ldots, \ell_{n-1} \mapsto \ell_n\}$ with $n \geq 1$, $\ell_0 = s_1(\mathtt{x})$, $\ell_n = s_1(\mathtt{y})$ and for all $i \neq j \in [0, n]$, $\ell_i \neq \ell_j$. Notice that since for all $\mathtt{x} \in X$, $\mathfrak{f}(s_1(\mathtt{x})) = s_2(\mathtt{x})$ (Definition 3.2), we have $s_1(\mathtt{x}) = s_1(\mathtt{y})$ if and only if $s_2(\mathtt{x}) = s_2(\mathtt{y})$. Moreover, again from the definition of X-isomorphism, $\mathrm{dom}(h_1)$ and $\mathrm{dom}(h_2)$ have the same cardinality. Thus, if $\mathrm{dom}(h_1) = \emptyset$ and $s_1(\mathtt{x}) = s_1(\mathtt{y})$, we conclude that $\mathrm{dom}(h_2) = \emptyset$ and $s_2(\mathtt{x}) = s_2(\mathtt{y})$. Otherwise, let us consider the case where $h_1 = \{\ell_0 \mapsto \ell_1, \ell_1 \mapsto \ell_2, \ldots, \ell_{n-1} \mapsto \ell_n\}$ with $n \geq 1$, $\ell_0 = s_1(\mathtt{x})$, $\ell_n = s_1(\mathtt{y})$ and for all $i \neq j \in [0, n]$ $\ell_i \neq \ell_j$. Notice that $\mathrm{card}(\mathrm{dom}(h_1)) = n$. By definition of X-isomorphism, $s_2(\mathtt{x}) = \mathfrak{f}(s_1(\mathtt{x})) = \mathfrak{f}(\ell_0)$, $s_2(\mathtt{y}) = \mathfrak{f}(s_1(\mathtt{y})) = \mathfrak{f}(\ell_n)$ and given $i \in [0, n-1]$, $h_2(\mathfrak{f}(\ell_i)) = \mathfrak{f}(h_1(\ell_i)) = \mathfrak{f}(\ell_{i+1})$. So, $\{s_2(\mathtt{x}) \mapsto \mathfrak{f}(\ell_1), \mathfrak{f}(\ell_1) \mapsto \mathfrak{f}(\ell_2), \ldots, \mathfrak{f}(\ell_{n-1}) \mapsto s_2(\mathtt{y})\} \subseteq h_2$. Moreover, from the fact that for all $i \neq j \in [0, n]$, $\ell_i \neq \ell_j$, we conclude that for all $i \neq j \in [0, n]$, $\mathfrak{f}(\ell_i) \neq \mathfrak{f}(\ell_j)$. From $\mathrm{card}(\mathrm{dom}(h_1)) = n$ we derive $\mathrm{card}(\mathrm{dom}(h_2)) = n$, and therefore $h_2 = \{s_2(\mathtt{x}) \mapsto \mathfrak{f}(\ell_1), \mathfrak{f}(\ell_1) \mapsto \mathfrak{f}(\ell_2), \ldots, \mathfrak{f}(\ell_{n-1}) \mapsto s_2(\mathtt{y})\}$, where $s_2(\mathtt{x}), \mathfrak{f}(\ell_1), \ldots, \mathfrak{f}(\ell_{n-1}), s_2(\mathtt{y})$ are $n+1$ distinct locations. We conclude that $(\mathrm{LOC}_2, s_2, h_2)$ satisfies $\mathtt{ls}(\mathtt{x}, \mathtt{y})$.

Concerning the cases for the induction step, we omit the obvious cases when the outermost connective is the conjunction or the negation. We conclude the proof by considering formulae of the form $\varphi_1 * \varphi_2$, $\varphi_1 -\!\!* \varphi_2$ and $\forall \mathtt{x}\, \varphi_1$.

induction step: case with $*$. If $(\mathrm{LOC}_1, s_1, h_1) \models \varphi_1 * \varphi_2$, then there are two heaps h_1' and h_1'' such that $h_1 = h_1' + h_1''$, $(\mathrm{LOC}_1, s_1, h_1') \models \varphi_1$ and $(\mathrm{LOC}_1, s_1, h_1'') \models \varphi_2$. Let $h_2' = \mathfrak{f}(h_1')$ and $h_2'' = \mathfrak{f}(h_1'')$ the images of h_1' and h_1'' via $\mathfrak{f}$. Since $\mathfrak{f}$ is an X-isomorphism between h_1 and h_2 it holds that:

$$h_2 = \mathfrak{f}(h_1) = \mathfrak{f}(h_1' + h_1'') = \mathfrak{f}(h_1') + \mathfrak{f}(h_1'') = h_2' + h_2''$$

and moreover $h_2' \perp h_2''$. Lastly, $(\mathrm{LOC}_1, s_1, h_1') \approx_X (\mathrm{LOC}_2, s_2, h_2')$, since $\mathfrak{f}$ is an X-isomorphism also between these structures. The same holds for h_1'' and h_2''. Therefore, by the induction hypothesis, we get $(\mathrm{LOC}_2, s_2, h_2') \models \varphi_1$ and $(\mathrm{LOC}_2, s_2, h_2'') \models \varphi_2$. We conclude that $(\mathrm{LOC}_2, s_2, h_2) \models \varphi_1 * \varphi_2$.

induction step: case with $-\!\!*$. If $(\mathrm{LOC}_1, s_1, h_1) \models \varphi_1 -\!\!* \varphi_2$, then for every heap h_1', if $h_1' \perp h_1$ and $(\mathrm{LOC}_1, s_1, h_1') \models \varphi_1$ then $(\mathrm{LOC}_1, s_1, h_1 + h_1') \models \varphi_2$. We show that $(\mathrm{LOC}_2, s_2, h_2) \models \varphi_1 -\!\!* \varphi_2$, which is true whenever for all h_2', if $h_2' \perp h_2$ and $(\mathrm{LOC}_2, s_2, h_2') \models \varphi_1$ then $(\mathrm{LOC}_2, s_2, h_2 + h_2') \models \varphi_2$. Consider a heap h_2' such that $h_2' \perp h_2$ and $(\mathrm{LOC}_2, s_2, h_2') \models \varphi_1$. By recalling that $\mathfrak{f}$ is a bijection from LOC_1 to LOC_2, we construct the heap $h_1' = \mathfrak{f}^{-1}(h_2')$. Directly from the fact that $\mathfrak{f}$ is a X-isomorphism between $(\mathrm{LOC}_1, s_1, h_1)$ and $(\mathrm{LOC}_2, s_2, h_2)$, it is easy to see that the following two properties hold:

(1) $(\mathrm{LOC}_1, s_1, h_1') \approx_X (\mathrm{LOC}_2, s_2, h_2')$ and $\mathfrak{f}$ is an X-isomorphism between the two structures.
(2) $h_1' \perp h_1$, from (1) together with $(\mathrm{LOC}_1, s_1, h_1') \approx_X (\mathrm{LOC}_2, s_2, h_2')$, $h_2' \perp h_2$ and $\mathfrak{f}^{-1}(h_2') = h_1'$.

By the induction hypothesis, from (1) we conclude that $(\mathrm{LOC}_1, s_1, h_1') \models \varphi_1$. Then, from (2) and the initial hypothesis $(\mathrm{LOC}_1, s_1, h_1) \models \varphi_1 -\!\!* \varphi_2$, we obtain that $(\mathrm{LOC}_1, s_1, h_1 + h_1')$

satisfies φ_2. By definition $\mathfrak{f}(h_1 + h_1') = \mathfrak{f}(h_1) + \mathfrak{f}(\mathfrak{f}^{-1}(h_2')) = h_2 + h_2'$ and therefore by the induction hypothesis, we get $(\mathrm{LOC}_2, s_2, h_2 + h_2') \models \varphi_2$. Thus, $(\mathrm{LOC}_2, s_2, h_2) \models \varphi_1 \mathrel{-\!\!*} \varphi_2$.

induction step: case with $\forall$. If $(\mathrm{LOC}_1, s_1, h_1) \models \forall \mathsf{x}\ \varphi_1$, then $(\mathrm{LOC}_1, s_1[\mathsf{x} \leftarrow \ell], h_1) \models \varphi_1$ holds for all $\ell \in \mathrm{LOC}_1$. We prove that $(\mathrm{LOC}_2, s_2, h_2) \models \forall \mathsf{x}\ \varphi_1$, which is true whenever for every $\ell' \in \mathrm{LOC}_2$, $(\mathrm{LOC}_2, s_2[\mathsf{x} \leftarrow \ell'], h_2) \models \varphi_1$. Notice that $\mathsf{x} \notin X$ since X contains only the free variables in $\forall \mathsf{x}\ \varphi_1$. Let ℓ' be a location in LOC_2. We have $\mathfrak{f}^{-1}(\ell') \in \mathrm{LOC}_1$ and $(\mathrm{LOC}_1, s_1[\mathsf{x} \leftarrow \mathfrak{f}^{-1}(\ell')], h_1) \models \varphi_1$. From $(\mathrm{LOC}_1, s_1, h_1) \approx_X (\mathrm{LOC}_2, s_2, h_2)$ and $\mathsf{x} \notin X$, we derive $\mathfrak{f}(h_1) = h_2$ and for every $\mathsf{y} \in X$, $\mathfrak{f}(s_1[\mathsf{x} \leftarrow \mathfrak{f}^{-1}(\ell')](\mathsf{y})) = \mathfrak{f}(s_1(\mathsf{y})) = s_2(\mathsf{y}) = s_2[\mathsf{x} \leftarrow \ell'](\mathsf{y})$. Thus, $(\mathrm{LOC}_1, s_1[\mathsf{x} \leftarrow \mathfrak{f}^{-1}(\ell')], h_1) \approx_{X \cup \{\mathsf{x}\}} (\mathrm{LOC}_2, s_2[\mathsf{x} \leftarrow \ell'], h_2)$. Using the induction hypothesis, we derive $(\mathrm{LOC}_2, s_2[\mathsf{x} \leftarrow \ell'], h_2) \models \varphi_1$. Consequently, $(\mathrm{LOC}_2, s_2, h_2) \models \forall \mathsf{x}\ \varphi_1$. □

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 $\mathrm{SL}(\forall, \mathrel{-\!\!*})$. Indeed, suppose that φ is satisfiable in the memory state (s, h), then φ is satisfiable with the generalised semantics by considering the model $(\mathbb{N}, s, h)$. Similarly, suppose that φ is satisfiable in the generalised memory state $(\mathrm{LOC}_1, s_1, h_1)$. As LOC_1 is countably infinite, there is a bijection $\mathfrak{f} : \mathrm{LOC}_1 \to \mathbb{N}$. Let (s, h) such that $\mathfrak{f}(h_1) = h$ and for every $\mathsf{x} \in \mathrm{PVAR}$, $s(\mathsf{x}) \stackrel{\text{def}}{=} \mathfrak{f}(s_1(\mathsf{x}))$. Let X be the set of free variables occurring in φ. By construction of (s, h), we have $(\mathrm{LOC}_1, s_1, h_1) \approx_X (\mathbb{N}, s, h)$. By Lemma 3.3, we conclude $(\mathbb{N}, s, h) \models \varphi$, which is equivalent to $(s, h) \models \varphi$.

3.2 Encoding quantified variables as cells in the heap

In this section, we introduce three additional atomic predicates that allow us to encode the first-order quantification of $\mathrm{SL}(\forall, *, \mathrel{-\!\!*})$ as memory updates of $\mathrm{SL}(*, \mathrel{-\!\!*}, \mathtt{ls})$:

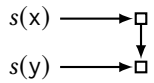
$$\mathtt{alloc}^{-1}(\mathsf{x}), \qquad n(\mathsf{x}) = n(\mathsf{y}), \qquad n(\mathsf{x}) \hookrightarrow n(\mathsf{y}).$$

The characterisation of these predicates in terms of formulae in $\mathrm{SL}(*, \mathrel{-\!\!*}, \mathtt{ls})$ is given in Section 3.4. For the time being, we simply assume that they can be defined in the logic, and work following their semantics, given below:

$$\begin{array}{lcl}
(s,h) \models \mathtt{alloc}^{-1}(\mathsf{x}) & \iff & s(\mathsf{x}) \in \mathrm{ran}(h). \\
(s,h) \models n(\mathsf{x}) = n(\mathsf{y}) & \iff & \{s(\mathsf{x}), s(\mathsf{y})\} \subseteq \mathrm{dom}(h) \text{ and } h(s(\mathsf{x})) = h(s(\mathsf{y})). \\
(s,h) \models n(\mathsf{x}) \hookrightarrow n(\mathsf{y}) & \iff & \{s(\mathsf{x}), s(\mathsf{y})\} \subseteq \mathrm{dom}(h) \text{ and } h^2(s(\mathsf{x})) = h(s(\mathsf{y})).
\end{array}$$

In other words, $\mathtt{alloc}^{-1}(\mathsf{x})$ holds in (s, h) whenever $s(\mathsf{x})$ has a predecessor in h. The satisfaction of $n(\mathsf{x}) = n(\mathsf{y})$ corresponds to the existence of the following pattern in the memory state (s, h):

whereas the satisfaction of $n(\mathsf{x}) \hookrightarrow n(\mathsf{y})$ corresponds to the existence of the following pattern:



As a rule of thumb, '$n(\mathsf{x})$' in $n(\mathsf{x}) = n(\mathsf{y})$ or in $n(\mathsf{x}) \hookrightarrow n(\mathsf{y})$ refers to the "next" value of x understood as $h(s(\mathsf{x}))$ – if it exists. Let us first intuitively explain how the two last predicates will help encoding $\mathrm{SL}(\forall, \mathrel{-\!\!*})$. By definition, the satisfaction of the quantified formula $\forall \mathsf{x}\ \psi$ from $\mathrm{SL}(\forall, \mathrel{-\!\!*})$ requires the satisfaction of the formula ψ for all the values ℓ in LOC assigned to x. The principle of the encoding is to use a value $s(\mathsf{x})$ not in the domain or range of the heap to mimic the store by modifying how $s(\mathsf{x})$ is allocated: typically 'x takes the value ℓ" is encoded by the heap $\{s(\mathsf{x}) \mapsto \ell\}$ where $s(\mathsf{x})$ is not in the range and domain of the heap. Figure 3 intuitively depicts this encoding, highlighting how the store of the memory state in Figure 2 can be simulated directly by the heap. As showed in this picture, the principle of the encoding is to use a set L of locations initially not in the domain or

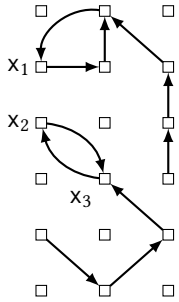


Fig. 2. A memory state.

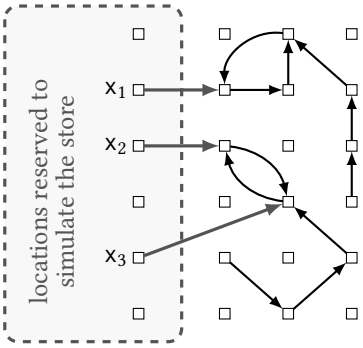


Fig. 3. Simulating the store with the heap.

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 whether $\mathsf{x} \hookrightarrow \mathsf{y}$ (or $\mathsf{x} = \mathsf{y}$) holds, we will check if $n(\mathsf{x}) \hookrightarrow n(\mathsf{y})$ (or $n(\mathsf{x}) = n(\mathsf{y})$) holds, where x and y correspond, after the translation, to the locations in L that mimic the store for those variables.

Let X be the finite set of variables needed for the translation. 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. Afterwards, the universal quantification for the quantifier $\forall$ is simulated by means of the separating implication $-\!\!*$: the formula $\forall \mathsf{x}\ \psi$ will be encoded by a formula of the form

$$(\mathtt{alloc}(\mathsf{x}) \wedge \mathtt{size} = 1) -\!\!* (\mathrm{Safe}(X) \Longrightarrow \mathrm{T}(\psi)),$$

where $\mathrm{Safe}(X)$ (formally defined below) checks whether the locations in L still satisfy the auxiliary conditions just described, i.e. locations in L are not reached by other locations, whereas $\mathrm{T}(\psi)$ is the translation of ψ. So, as hinted earlier, the above formula universally quantifies over the addition of all subheaps of the form $\{s(\mathsf{x}) \mapsto \ell\}$. The locations in L play a special role and this shall be specified with the formula $\mathrm{Safe}(X)$.

The formula $\psi_1 -\!\!* \psi_2$ cannot simply be translated into $\mathrm{T}(\psi_1) -\!\!* (\mathrm{Safe}(X) \Longrightarrow \mathrm{T}(\psi_2))$ 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 x in the formula, X will contain an auxiliary variable $\overline{\mathsf{x}}$, 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 $2q$. Roughly speaking, in the translation of a formula whose outermost connective is the magic wand, the locations corresponding to variables of the form $\overline{\mathsf{x}}$ 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.

Let us formalise the intuition of using part of the memory to mimic the store depicted in Figure 3, by defining a suitable encoding between generalised memory states.

Definition 3.4. Let $X \subseteq Y$ be finite sets of program variables. Let $(\mathrm{LOC}_1, s_1, h_1)$ 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 $(\mathrm{LOC}_2, s_2, h_2)$ with respect to X and Y, written $(\mathrm{LOC}_1, s_1, h_1) \rhd_Y^X (\mathrm{LOC}_2, s_2, h_2)$, if the following conditions hold:

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

Notice that the heap h_2 is equal to the heap h_1 augmented with the heap $\{s_2(\mathsf{x}) \mapsto s_1(\mathsf{x}) \mid \mathsf{x} \in X\}$. From the encoding, we can retrieve the initial heap by removing the memory cells corresponding to variables in X. By way of example, notice how the memory state in Figure 2 satisfies the formula $\mathsf{x}_3 \hookrightarrow \mathsf{x}_2$, whereas its encoding in Figure 3 satisfies the formula $n(\mathsf{x}_3) \hookrightarrow n(\mathsf{x}_2)$. Furthermore, notice that the memory state in Figure 3 satisfies the formula

$$\mathrm{Safe}(\{\mathsf{x}_1, \mathsf{x}_2, \mathsf{x}_3\}) = \mathsf{x}_1 \neq \mathsf{x}_2 \wedge \mathsf{x}_2 \neq \mathsf{x}_3 \wedge \mathsf{x}_1 \neq \mathsf{x}_3 \wedge \neg\mathtt{alloc}^{-1}(\mathsf{x}_1) \wedge \neg\mathtt{alloc}^{-1}(\mathsf{x}_2) \wedge \neg\mathtt{alloc}^{-1}(\mathsf{x}_3).$$

This formula guarantees that the memory states are encodings of other memory states (in particular, the one in Figure 2). In general, we write $\mathrm{Safe}(Y)$ for the following formula:

$$\mathrm{Safe}(Y) \stackrel{\text{def}}{=} \Big(\bigwedge_{\text{distinct } \mathsf{x},\mathsf{y} \in Y} \mathsf{x} \neq \mathsf{y}\Big) \wedge \Big(\bigwedge_{\mathsf{x} \in Y} \neg\mathtt{alloc}^{-1}(\mathsf{x})\Big).$$

3.3 The translation

We are now ready to define the translation of a first-order formula in quantifier-free separation logic extended with the three predicates introduced at the beginning of the section. Let φ be a closed formula of $\mathrm{SL}(\forall, -\!\!*)$ with quantified variables $X = \{\mathsf{x}_1, \ldots, \mathsf{x}_q\}$. Without loss of generality, we can assume that φ is *well-quantified*, i.e. distinct quantifications involve distinct variables. We consider a set $Y = \{\mathsf{x}_1, \ldots, \mathsf{x}_{2q}\}$ and $\overline{(.)}$ to be the (unique) involution on Y such that for all $i \in [1, q]$, $\overline{\mathsf{x}_i} \stackrel{\text{def}}{=} \mathsf{x}_{i+q}$. Notice that $\overline{\mathsf{x}_{i+q}} = \mathsf{x}_i$. We extend the involution to arbitrary subsets of Y, so that in particular $\overline{X} = \{\mathsf{x}_{q+1}, \ldots, \mathsf{x}_{2q}\}$ and $\overline{\overline{X}} = X$.

The translation function $\mathrm{T}(\psi, Y)$ has two arguments: the formula ψ in $\mathrm{SL}(\forall, -\!\!*)$ to be recursively translated, and the total set of variables potentially appearing in the target formula, i.e. Y. The translation function is given below, assuming that the variables in ψ are either included in X or included $\overline{X}$.

$$\begin{aligned}
\mathrm{T}(\mathsf{x}_i = \mathsf{x}_j, Y) &\stackrel{\text{def}}{=} n(\mathsf{x}_i) = n(\mathsf{x}_j)\\
\mathrm{T}(\mathsf{x}_i \hookrightarrow \mathsf{x}_j, Y) &\stackrel{\text{def}}{=} n(\mathsf{x}_i) \hookrightarrow n(\mathsf{x}_j)\\
\mathrm{T}(\neg\psi, Y) &\stackrel{\text{def}}{=} \neg\mathrm{T}(\psi, Y)\\
\mathrm{T}(\psi_1 \wedge \psi_2, Y) &\stackrel{\text{def}}{=} \mathrm{T}(\psi_1, Y) \wedge \mathrm{T}(\psi_2, Y)\\
\mathrm{T}(\forall \mathsf{x}_i\ \psi, Y) &\stackrel{\text{def}}{=} (\mathtt{alloc}(\mathsf{x}_i) \wedge \mathtt{size} = 1) -\!\!* (\mathrm{Safe}(Y) \Rightarrow \mathrm{T}(\psi, Y))
\end{aligned}$$

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

$$\Big(\big(\bigwedge_{\mathsf{z} \in Z} \mathtt{alloc}(\overline{\mathsf{z}})\big) \wedge \big(\bigwedge_{\mathsf{z} \in Y \setminus Z} \neg\mathtt{alloc}(\overline{\mathsf{z}})\big) \wedge \mathrm{Safe}(Y) \wedge \mathrm{T}(\psi_1[\mathsf{x} \leftarrow \overline{\mathsf{x}} \mid \mathsf{x} \in Y], Y)\Big) -\!\!*$$

$$\Big(\big(\big(\bigwedge_{\mathsf{z} \in Z} n(\mathsf{z}) = n(\overline{\mathsf{z}})\big) \wedge \mathrm{Safe}(Y)\big) \Rightarrow \big(\big(\bigwedge_{\mathsf{z} \in Z} \mathtt{alloc}(\overline{\mathsf{z}}) \wedge \mathtt{size} = \mathrm{card}(Z)\big) * \mathrm{T}(\psi_2, Y)\big)\Big)\Big),$$

where Z is the set of free variables in ψ_1, and $\psi_1[\mathsf{x} \leftarrow \overline{\mathsf{x}} \mid \mathsf{x} \in Y]$ denotes the formula obtained from ψ_1 by replacing simultaneously all the variables $\mathsf{x} \in Y$ by $\overline{\mathsf{x}}$.

Example 3.5. Assume that $q = 2$, and therefore $Y = \{\mathsf{x}_1, \mathsf{x}_2, \mathsf{x}_3, \mathsf{x}_4\}$. We have $\overline{\mathsf{x}_1} = \mathsf{x}_3$ and $\overline{\mathsf{x}_2} = \mathsf{x}_4$. Thus, the translation $\mathrm{T}(\mathsf{x}_1 \hookrightarrow \mathsf{x}_2 -\!\!* \mathsf{x}_1 \hookrightarrow \mathsf{x}_2, Y)$ is defined as the formula below:

$$\Big((\mathtt{alloc}(\mathsf{x}_3) \wedge \mathtt{alloc}(\mathsf{x}_4) \wedge \neg\mathtt{alloc}(\mathsf{x}_1) \wedge \neg\mathtt{alloc}(\mathsf{x}_2) \wedge \mathrm{Safe}(Y) \wedge n(\mathsf{x}_3) \hookrightarrow n(\mathsf{x}_4)) -\!\!*$$

$$\big((n(\mathsf{x}_1) = n(\mathsf{x}_3) \wedge n(\mathsf{x}_2) = n(\mathsf{x}_4) \wedge \mathrm{Safe}(Y)) \Rightarrow ((\mathtt{alloc}(\mathsf{x}_3) \wedge \mathtt{alloc}(\mathsf{x}_4) \wedge \mathtt{size} = 2) * n(\mathsf{x}_1) \hookrightarrow n(\mathsf{x}_2)))\Big).$$

Let us informally analyse the translation $\mathrm{T}(\psi_1 \mathrel{-\!\!*} \psi_2, Y)$. It is a formula of the form $\psi_1' \mathrel{-\!\!*} \psi_2'$ where ψ_1' expresses the four constraints below (which follow the four conjuncts of ψ_1', from left to right):

(1) All the variables $\overline{\mathsf{z}}$ with $\mathsf{z} \in Z$ are allocated.
(2) None of the variables $\overline{\mathsf{x}}$ with $\mathsf{x} \in Y \setminus Z$ is allocated.
(3) The environment around the variables $\mathsf{x} \in Y$ is safe. In particular, none of the variables in Y corresponds to a location that is pointed by another location.
(4) Finally, one needs to guarantee that the translated formula $\mathrm{T}(\psi_1, Y)$ holds true when each variable $\mathsf{x} \in Y$ is replaced by its copy $\overline{\mathsf{x}}$, i.e. that $\mathrm{T}(\psi_1[\mathsf{x} \leftarrow \overline{\mathsf{x}} \mid \mathsf{x} \in Y], Y)$ holds.

Essentially, the formula ψ_1' considers heaps satisfying $\mathrm{T}(\psi_1, Y)$, but with respect to the copies $\overline{Z}$ of the free variables Z in ψ_1. On these heaps, ψ_2' checks whether the following two conditions hold (antecedent of ψ_2'):

(A) The values for z and $\overline{\mathsf{z}}$ are equal for all $\mathsf{z} \in Z$.
(B) As in (3), the environment around the variables $\mathsf{x} \in Y$ is safe.

When these two conditions hold, ψ_2' removes the assignments corresponding to the variables in $\overline{Z}$, see subformula $\bigwedge_{\mathsf{z} \in Z} \mathtt{alloc}(\overline{\mathsf{z}}) \wedge \mathtt{size} = \mathrm{card}(Z)$, and then check whether the formula $\mathrm{T}(\psi_2, Y)$ holds. Notice that freeing the variables in $\overline{Z}$ allows us to reuse them whenever a magic wand appears in $\mathrm{T}(\psi_2, Y)$.

Here is the main result of this section, that is the correctness of the translation $\mathrm{T}(\psi, Y)$.

Lemma 3.6. *Let $Y = \{\mathsf{x}_1, \ldots, \mathsf{x}_{2q}\}$ and E be either $\{\mathsf{x}_1, \ldots, \mathsf{x}_q\}$ or $\overline{\{\mathsf{x}_1, \ldots, \mathsf{x}_q\}} = \{\mathsf{x}_{q+1}, \ldots, \mathsf{x}_{2q}\}$. Let $X \subseteq E$. Let ψ be a well-quantified formula in $\mathrm{SL}(\forall, \mathrel{-\!\!*})$ with free variables among X and bound variables among $E \setminus X$. If $(\mathrm{LOC}_1, s_1, h_1) \rhd_Y^X (\mathrm{LOC}_2, s_2, h_2)$ then $(s_1, h_1) \models \psi$ iff $(s_2, h_2) \models \mathrm{T}(\psi, Y)$.*

The statement of Lemma 3.6, as well as its proof, refers to the involution $\overline{(.)}$ such that $\overline{\mathsf{x}_i} = \mathsf{x}_{i+q}$.

Proof. The proof is by induction on the structure of ψ. We start by proving the two base cases with the atomic formulae $\mathsf{x}_i = \mathsf{x}_j$ and $\mathsf{x}_i \hookrightarrow \mathsf{x}_j$.

(1) Let ψ be $\mathsf{x}_i = \mathsf{x}_j$. We have $\{\mathsf{x}_i, \mathsf{x}_j\} \subseteq X \subseteq Y$ and $\mathrm{T}(\psi, Y) = n(\mathsf{x}_i) = n(\mathsf{x}_j)$ by definition. The following equivalences hold.

$$\begin{array}{lll} (s_1, h_1) \models \mathsf{x}_i = \mathsf{x}_j & \iff & s_1(\mathsf{x}_i) = s_1(\mathsf{x}_j), \\ & \iff & h_2(s_2(\mathsf{x}_i)) = h_2(s_2(\mathsf{x}_j)), \qquad \text{(Definition 3.4)} \\ & \iff & (s_2, h_2) \models n(\mathsf{x}_i) = n(\mathsf{x}_j). \end{array}$$

(2) Let ψ be $\mathsf{x}_i \hookrightarrow \mathsf{x}_j$. We have $\{\mathsf{x}_i, \mathsf{x}_j\} \subseteq X \subseteq Y$ and $\mathrm{T}(\psi, Y) = n(\mathsf{x}_i) \hookrightarrow n(\mathsf{x}_j)$ by definition. The following equivalences hold.

$$\begin{array}{lll} (s_1, h_1) \models \mathsf{x}_i \hookrightarrow \mathsf{x}_j & \iff & h_1(s_1(\mathsf{x}_i)) = s_1(\mathsf{x}_j), \\ & \iff & h_2(h_2(s_2(\mathsf{x}_i))) = h_2(s_2(\mathsf{x}_j)), \qquad \text{(Definition 3.4)} \\ & \iff & (s_2, h_2) \models n(\mathsf{x}_i) \hookrightarrow n(\mathsf{x}_j). \end{array}$$

We now consider the induction step and we distinguish the different cases depending on the outermost connective. To be precise, we assume the induction hypothesis to be the following:

Let $N \in \mathbb{N}$. Let E be either $\{\mathsf{x}_1, \ldots, \mathsf{x}_q\}$ or $\overline{\{\mathsf{x}_1, \ldots, \mathsf{x}_q\}} = \{\mathsf{x}_{q+1}, \ldots, \mathsf{x}_{2q}\}$. Let $X \subseteq E$. Let φ be a well-quantified formula in $\mathrm{SL}(\forall, \mathrel{-\!\!*})$ with free variables among X, bound variables among $E \setminus X$, and size at most N. If $(\mathrm{LOC}_1, s_1, h_1) \rhd_Y^X (\mathrm{LOC}_2, s_2, h_2)$ then $(s_1, h_1) \models \varphi$ iff $(s_2, h_2) \models \mathrm{T}(\varphi, Y)$.

In the induction step, we take a formula ψ of size $N + 1$. We omit the obvious cases for the Boolean connectives. We now prove the result for $\forall \mathsf{x}_i\ \varphi$ and $\varphi_1 \mathrel{-\!\!*} \varphi_2$.

(3) Let $\psi = \forall\ \mathsf{x}_i\ \varphi$ with $\mathsf{x}_i \in E \setminus X$ and therefore by definition

$$\mathrm{T}(\forall\ \mathsf{x}_i\ \varphi, Y) = (\mathtt{alloc}(\mathsf{x}_i) \wedge \mathtt{size} = 1) \mathrel{-\!\!*} (\mathrm{Safe}(Y) \Rightarrow \mathrm{T}(\varphi, Y)).$$

By definition, $(s_1, h_1) \models \forall \mathsf{x}_i\ \varphi$ if and only if for all locations $\ell \in \mathrm{LOC}_1$, $(s_1[\mathsf{x}_i \leftarrow \ell], h_1) \models \varphi$. Let us now consider the generalised memory state $(\mathrm{LOC}_2, s_2, h_2 + \{s_2(\mathsf{x}_i) \mapsto \ell\})$. Since $\mathsf{x}_i \notin X$, the heap $h_2 + \{s_2(\mathsf{x}_i) \mapsto \ell\}$ is well-defined. From $(\mathrm{LOC}_1, s_1, h_1) \rhd_Y^X (\mathrm{LOC}_2, s_2, h_2)$ and by Definition 3.4, we obtain

$$(\mathrm{LOC}_1, s_1[\mathsf{x}_i \leftarrow \ell], h_1) \rhd_Y^{X \cup \{\mathsf{x}_i\}} (\mathrm{LOC}_2, s_2, h_2 + \{s_2(\mathsf{x}_i) \mapsto \ell\}).$$

We apply the induction hypothesis, and conclude that:

$$\begin{array}{rl} & \text{for all } \ell \in \mathrm{LOC}_1,\ (s_1[\mathsf{x}_i \leftarrow \ell], h_1) \models \varphi, \\ \text{iff} & \text{for all } \ell \in \mathrm{LOC}_1,\ (s_2, h_2 + \{s_2(\mathsf{x}_i) \mapsto \ell\}) \models \mathrm{T}(\varphi, Y). \end{array}$$

Moreover, due to the fulfilment of the conditions in $\rhd_Y^{X \cup \{\mathsf{x}_i\}}$, any memory state $(s_2, h_2 + \{s_2(\mathsf{x}_i) \mapsto \ell'\})$ satisfies $\mathrm{Safe}(Y)$ if and only if $\ell' \in \mathrm{LOC}_1$. Therefore,

$$\begin{array}{rl} & \text{for all } \ell \in \mathrm{LOC}_1,\ (s_2, h_2 + \{s_2(\mathsf{x}_i) \mapsto \ell\}) \models \mathrm{T}(\varphi, Y), \\ \text{iff} & \text{for all } \ell \in \mathrm{LOC}_2,\ (s_2, h_2 + \{s_2(\mathsf{x}_i) \mapsto \ell\}) \models \mathrm{Safe}(Y) \Rightarrow \mathrm{T}(\varphi, Y). \end{array}$$

Indeed, if $\ell \in \mathrm{LOC}_1$ then $\mathrm{Safe}(Y)$ and $\mathrm{T}(\varphi, Y)$ are both satisfied (for $\mathrm{Safe}(Y)$, remember that all locations in $\{s_2(\mathsf{y}) \mid \mathsf{y} \in X\} \cap \mathrm{dom}(h_2)$ already point to elements of LOC_1, due to the fulfilment of the conditions in $\rhd_Y^X$), otherwise whenever $\ell \in \mathrm{LOC}_2 \setminus \mathrm{LOC}_1$ the premise of the implication does not hold and therefore the formula is trivially satisfied. Observe that a memory state (s_2, h') satisfies $\mathtt{alloc}(\mathsf{x}_i) \wedge \mathtt{size} = 1$ if and only if h' is of the form $\{s_2(\mathsf{x}_i) \mapsto \ell''\}$, for some $\ell'' \in \mathrm{LOC}_2$. Then, we conclude that

$$\begin{array}{rl} & \text{for all } \ell \in \mathrm{LOC}_2,\ (s_2, h_2 + \{s_2(\mathsf{x}_i) \mapsto \ell\}) \models \mathrm{Safe}(Y) \Rightarrow \mathrm{T}(\varphi, Y), \\ \text{iff} & \text{for all } h',\ \text{if } (h' \perp h_2 \text{ and } (s_2, h') \models \mathtt{alloc}(\mathsf{x}_i) \wedge \mathtt{size} = 1) \\ & \quad \text{then } (s_2, h_2 + h') \models \mathrm{Safe}(X) \Rightarrow \mathrm{T}(\varphi, Y) \\ \text{iff} & (s_2, h_2) \models (\mathtt{alloc}(\mathsf{x}_i) \wedge \mathtt{size} = 1) \mathrel{-\!\!*} (\mathrm{Safe}(Y) \Rightarrow \mathrm{T}(\varphi, Y)). \end{array}$$

(4) Let $\psi = \varphi_1 \mathrel{-\!\!*} \varphi_2$. Let Z be the set of free variables in φ_1. Below, we assume that for every $\mathsf{x} \in X$, $s_1(\mathsf{x}) = s_1(\overline{\mathsf{x}})$. Notice that this assumption is without loss of generality, as ψ is written with free variables among X, where $X \cap \overline{X} = \emptyset$, and the memory state $(s_1[\overline{\mathsf{x}} \leftarrow s_1(\mathsf{x}) \mid \mathsf{x} \in X], h_1)$ is X-isomorphic to (s_1, h_1), where $s_1[\overline{\mathsf{x}} \leftarrow s_1(\mathsf{x}) \mid \mathsf{x} \in X]$ is the store obtained from s_1 by assigning $s_1(\mathsf{x})$ to every variable $\overline{\mathsf{x}} \in \overline{X}$. By Lemma 3.3, $(s_1, h_1) \models \psi$ if and only if $(s_1[\overline{\mathsf{x}} \leftarrow s_1(\mathsf{x}) \mid \mathsf{x} \in X], h_1) \models \psi$.
($\Rightarrow$): If $(s_1, h_1) \models \varphi_1 \mathrel{-\!\!*} \varphi_2$, then by definition, for all heaps h_1', if $h_1' \perp h_1$ and $(s_1, h_1') \models \varphi_1$, then we have $(s_1, h_1 + h_1') \models \varphi_2$. We show that $(s_2, h_2) \models \mathrm{T}(\varphi_1 \mathrel{-\!\!*} \varphi_2, Y)$. By definition, $\mathrm{T}(\varphi_1 \mathrel{-\!\!*} \varphi_2, Y)$, holds whenever for all heaps h_2', if
(A1) $h_2' \perp h_2$,
(A2) $(s_2, h_2') \models (\bigwedge_{\mathsf{z} \in Z} \mathtt{alloc}(\overline{\mathsf{z}})) \wedge (\bigwedge_{\mathsf{z} \in Y \setminus Z} \neg\mathtt{alloc}(\overline{\mathsf{z}})) \wedge \mathrm{Safe}(Y) \wedge \mathrm{T}(\varphi_1[\mathsf{x} \leftarrow \overline{\mathsf{x}} \mid \mathsf{x} \in Y], Y)$,
(A3) $(s_2, h_2 + h_2') \models (\bigwedge_{\mathsf{z} \in Z} n(\mathsf{z}) = n(\overline{\mathsf{z}})) \wedge \mathrm{Safe}(Y)$,
then
(C1) $(s_2, h_2 + h_2') \models (\bigwedge_{\mathsf{z} \in Z} \mathtt{alloc}(\overline{\mathsf{z}}) \wedge \mathtt{size} = \mathrm{card}(Z)) * \mathrm{T}(\varphi_2, Y)$.
Let h_2' be some heap that satisfies the premises (A1)–(A3) of the implication. From (A3) together with $(\mathrm{LOC}_1, s_1, h_1) \rhd_Y^X (\mathrm{LOC}_2, s_2, h_2)$, which implies that for all $\mathsf{x} \in X \supseteq Z$, we have $h_2(s_2(\mathsf{x})) = s_1(\mathsf{x})$, we conclude that for all $\mathsf{z} \in Z$, we have $h_2'(s_2(\overline{\mathsf{z}})) = s_1(\mathsf{z})$. Moreover, (A2) entails that h_2' can be written as $h_Z + h_1'$ where $h_Z \stackrel{\text{def}}{=} \{s_2(\overline{\mathsf{z}}) \mapsto s_1(\mathsf{z}) \mid \mathsf{z} \in Z\}$ (h_1' is then defined as the unique subheap satisfying $h_2' = h_Z + h_1'$, once h_Z is defined). Observe that the domain of h_Z is the set of locations in LOC_2 that corresponds to variables in $\overline{Z}$, and its

codomain is included in LOC_1. For h_1' instead, both its domain and codomain are included in LOC_1. By Definition 3.4 and the assumption that for all $\mathsf{x} \in X$, $s_1(\mathsf{x}) = s_1(\overline{\mathsf{x}})$, it holds that

$$(\mathrm{LOC}_1, s_1, h_1') \rhd_Y^{\overline{Z}} (\mathrm{LOC}_2, s_2, h_2').$$

Since Z is the set of free variables in φ_1, $\overline{Z}$ is the set of free variables in $\varphi_1[\mathsf{x} \leftarrow \overline{\mathsf{x}} \mid \mathsf{x} \in Y]$. From $(s_2, h_2') \models \mathrm{T}(\varphi_1[\mathsf{x} \leftarrow \overline{\mathsf{x}} \mid \mathsf{x} \in Y], Y)$, we apply the induction hypothesis and conclude that $(s_1, h_1') \models \varphi_1[\mathsf{x} \leftarrow \overline{\mathsf{x}} \mid \mathsf{x} \in Y]$. Again thanks to the assumption that for all $\mathsf{x} \in X$, $s_1(\mathsf{x}) = s_1(\overline{\mathsf{x}})$, this allows us to conclude that $(s_1, h_1') \models \varphi_1$. Indeed, this holds directly from the well-known axiom of separation logic $\mathsf{x} = \mathsf{y} \wedge \varphi \implies \varphi[\mathsf{x} \leftarrow \mathsf{y}]$ (see e.g. [19]). Since $h_1' \sqsubseteq h_2'$, $h_1 \sqsubseteq h_2$ and $h_2' \perp h_2$ (A1), the property $h_1' \perp h_1$ holds and therefore, by the assumption $(s_1, h_1) \models \varphi_1 \mathrel{-\!\!*} \varphi_2$, we get $(s_1, h_1 + h_1') \models \varphi_2$. As such, since $(\mathrm{LOC}_1, s_1, h_1 + h_1') \rhd_Y^X (\mathrm{LOC}_2, s_2, h_2 + h_1')$ holds because h_1' has domain and codomain included in LOC_1 and is a subset of h_2' which is disjoint from h_2, we can use the induction hypothesis and obtain that $(s_2, h_2 + h_1') \models \mathrm{T}(\varphi_2, Y)$. Together with $h_2' = h_1' + h_Z$ and $(s_2, h_Z) \models \bigwedge_{\mathsf{z} \in Z} \mathtt{alloc}(\overline{\mathsf{z}}) \wedge \mathtt{size} = \mathrm{card}(Z)$, this allows us to derive (C1). ($\Leftarrow$): For the other direction, suppose that $(s_2, h_2) \models \mathrm{T}(\varphi_1 \mathrel{-\!\!*} \varphi_2, Y)$. This means that for all heaps h_2' if (A1)-(A3) holds, then (C1) holds. Let us prove that $(s_1, h_1) \models \varphi_1 \mathrel{-\!\!*} \varphi_2$, which by definition holds whenever for all heaps h_1', if $h_1' \perp h_1$ and $(s_1, h_1') \models \varphi_1$ then $(s_1, h_1 + h_1') \models \varphi_2$. Let h_1' be a heap disjoint from h_1 satisfying $(s_1, h_1') \models \varphi_1$. Let $h_2' \stackrel{\text{def}}{=} h_1' + \{s_2(\overline{\mathsf{z}}) \mapsto s_1(\mathsf{z}) \mid \mathsf{z} \in Z\}$. By definition of $\rhd_Y^X$ together with the assumption that for all $\mathsf{x} \in X$, $s_1(\mathsf{x}) = s_1(\overline{\mathsf{x}})$, we have

$$(\mathrm{LOC}_1, s_1, h_1') \rhd_Y^{\overline{Z}} (\mathrm{LOC}_2, s_2, h_2').$$

By the induction hypothesis, we get $(s_2, h_2') \models \mathrm{T}(\varphi_1[\mathsf{x} \leftarrow \overline{\mathsf{x}} \mid \mathsf{x} \in Y], Y)$, whereas from the definition of $\rhd_Y^{\overline{Z}}$, it holds that (s_2, h_2') satisfies

$$(\bigwedge_{\mathsf{z} \in Z} \mathtt{alloc}(\overline{\mathsf{z}})) \wedge (\bigwedge_{\mathsf{z} \in X \setminus Z} \neg \mathtt{alloc}(\overline{\mathsf{z}})) \wedge \mathrm{Safe}(Y).$$

Thus, (A2) holds. Besides this, h_2 can be written as $h_1 + \{s_2(\mathsf{y}) \mapsto s_1(\mathsf{y}) \mid \mathsf{y} \in X\}$ (condition (3) in Definition 3.4). Consequently, (A1) holds and the heap $h_2 + h_2'$ can be written as

$$h_1 + h_1' + \{s_2(\mathsf{x}) \mapsto s_1(\mathsf{x}) \mid \mathsf{x} \in X\} + \{s_2(\overline{\mathsf{z}}) \mapsto s_1(\mathsf{z}) \mid \mathsf{z} \in Z\}.$$

Therefore, it holds that for all $\mathsf{z} \in Z$, $(h_2 + h_2')(s_2(\mathsf{z})) = (h_2 + h_2')(s_2(\overline{\mathsf{z}}))$ or, equivalently, $(s_2, h_2 + h_2') \models \bigwedge_{\mathsf{z} \in Z} n(\mathsf{z}) = n(\overline{\mathsf{z}})$. Indeed, $Z \subseteq X$, $\mathsf{z} \in Z$, and, $s_2(\mathsf{z})$ and $s_2(\overline{\mathsf{z}})$ point to the same location. Furthermore, $(s_2, h_2 + h_2')$ also satisfies $\mathrm{Safe}(Y)$. (A3) holds and thus so does (C1), i.e.

$$(s_2, h_2 + h_2') \models (\bigwedge_{\mathsf{z} \in Z} \mathtt{alloc}(\overline{\mathsf{z}}) \wedge \mathtt{size} = \mathrm{card}(Z)) * \mathrm{T}(\varphi_2, Y).$$

Again, $h_2 + h_2'$ can be written as $h_1 + h_1' + \{s_2(\mathsf{x}) \mapsto s_1(\mathsf{x}) \mid \mathsf{x} \in X\} + \{s_2(\overline{\mathsf{z}}) \mapsto s_1(\mathsf{z}) \mid \mathsf{z} \in Z\}$, and the formula $\bigwedge_{\mathsf{z} \in Z} \mathtt{alloc}(\overline{\mathsf{z}}) \wedge \mathtt{size} = \mathrm{card}(Z)$ can be only satisfied by the memory state with the heap $\{s_2(\overline{\mathsf{z}}) \mapsto s_1(\mathsf{z}) \mid \mathsf{z} \in Z\}$. Consequently, $h_2 + h_1' = h_1 + h_1' + \{s_2(\mathsf{x}) \mapsto s_1(\mathsf{x}) \mid \mathsf{y} \in X\}$ is such that $(s_2, h_2 + h_1') \models \mathrm{T}(\varphi_2, Y)$. Lastly, since $(\mathrm{LOC}_1, s_1, h_1 + h_1') \rhd_Y^X (\mathrm{LOC}_2, s_2, h_2 + h_1')$, we can use the induction hypothesis to conclude that $(s, h_1 + h_1') \models \varphi_2$. □

Let φ be a formula of $\mathrm{SL}(\forall, \mathrel{-\!\!*})$ written with variables among $\{\mathsf{x}_1, \ldots, \mathsf{x}_q\}$. We define the translation $\mathcal{T}_{\mathrm{SAT}}(\varphi)$ into $\mathrm{SL}(*, \mathrel{-\!\!*}, \mathtt{ls})$ augmented with the new predicates where $\mathrm{T}(\varphi, Y)$ is defined recursively as before ($Y = \{\mathsf{x}_1, \ldots, \mathsf{x}_{2q}\}$).

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

The first two conjuncts specify initial conditions, namely each variable $\mathtt{x}$ in Y 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 $\mathtt{x}$ is isolated. Similarly, let $\mathcal{T}_{\mathrm{VAL}}(\varphi)$ be the formula in SL$(*, -\!\!*, \mathtt{ls})$ defined by

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

As a consequence of Lemma 3.6, φ and $\mathcal{T}_{\mathrm{SAT}}(\varphi)$ are shown equisatisfiable, whereas φ and $\mathcal{T}_{\mathrm{VAL}}(\varphi)$ are shown equivalid.

Corollary 3.7. *Let φ be a closed formula in* SL$(\forall, -\!\!*)$ *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.*

Proof. (I) First, suppose that φ is satisfiable, i.e. there is a memory state (s, h) such that $(s, h) \models \varphi$. It is then easy to define a generalised memory state $(\mathrm{LOC} \setminus \{1, \ldots, 2q\}, s', h')$ isomorphic with respect to the empty set to (s, h) (i.e., using the equivalence relation $\approx_\emptyset$) and satisfying φ by Lemma 3.3. Typically, to define (s', h') from (s, h), it is sufficient to shift all the values with an offset equal to $2q+1$. Now, let (LOC, s'', h'') be the generalised memory state such that $h'' = h'$, and for all $i \in [1, 2q]$, we have $s''(\mathtt{x}_i) = i$. One can check that $(\mathrm{LOC}, s'', h'') \models (\bigwedge_{i\in[1,2q]} \neg\mathtt{alloc}(\mathtt{x}_i)) \wedge \mathrm{Safe}(Y)$ with $Y = \{\mathtt{x}_1, \ldots, \mathtt{x}_{2q}\}$. Moreover, $(\mathrm{LOC} \setminus \{1, \ldots, 2q\}, s', h') \rhd_Y^\emptyset (\mathrm{LOC}, s'', h'')$. By Lemma 3.6, we have $(s'', h'') \models \mathrm{T}(\varphi, Y)$. So, $(s'', h'') \models \mathcal{T}_{\mathrm{SAT}}(\varphi)$.

Conversely, suppose that $(s, h) \models \mathcal{T}_{\mathrm{SAT}}(\varphi)$. Let (LOC', s', h') be the generalised memory state defined as follows:

(1) $\mathrm{LOC}' = \mathrm{LOC} \setminus \{s(\mathtt{x}) \mid \mathtt{x} \in Y\}$,
(2) the heap h' is equal to the restriction of h to locations in LOC'.

Since $(s, h) \models \mathrm{Safe}(Y)$ and by construction of (LOC', s', h'), we have

$$(\mathrm{LOC}', s', h') \rhd_Y^\emptyset (\mathrm{LOC}, s, h).$$

As $(s, h) \models \mathrm{T}(\varphi, Y)$, by Lemma 3.6, we get $(\mathrm{LOC}', s', h') \models \varphi$. Now, note that $\mathrm{LOC}' \subseteq \mathrm{LOC}$ and therefore we also have $(\mathrm{LOC}, s', h') \models \varphi$, where (LOC, s', h') is understood as a standard memory state (s', h').
(II) Similar to (I). □

3.4 Expressing the auxiliary atomic predicates

To complete the reduction, we 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 SL$(*, -\!\!*, \mathtt{ls})$. Let us introduce a few macros that shall be helpful.

- Given φ in SL$(*, -\!\!*, \mathtt{reach}^+)$ 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), we have $(s, h) \models [\varphi]_\gamma$ iff there is $h' \sqsubseteq h$ such that $\mathrm{card}(\mathrm{dom}(h')) = \gamma$ and $(s, h') \models \varphi$. Such formulae $[\varphi]_\gamma$ can be used to ensure that the minimum path between two locations interpreted by program variables is of length γ.
- 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 $h^\gamma(s(\mathtt{x})) = s(\mathtt{y})$ and γ is minimal (no cycles allowed). 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 SL$(*, -\!\!*, \mathtt{ls})$, we need to take advantage of an auxiliary variable $\mathtt{y}$ whose value is different from the one for $\mathtt{x}$. Let $\mathtt{alloc}_{\mathtt{y}}^{-1}(\mathtt{x})$ be the formula

$$\mathtt{x} \hookrightarrow \mathtt{x} \vee \mathtt{y} \hookrightarrow \mathtt{x} \vee (\top * ((\mathtt{alloc}(\mathtt{y}) \wedge \neg(\mathtt{y} \hookrightarrow \mathtt{x}) \wedge \mathtt{size} = 1) \mathbin{-\!\!\circledast} \mathtt{reach}(\mathtt{y}, \mathtt{x}) = 2))$$

Lemma 3.8. *Let* $\mathsf{x}, \mathsf{y} \in \mathrm{PVAR}$.

(I) *For all memory states* (s,h)*, if* $s(\mathsf{x}) \neq s(\mathsf{y})$*, then* $(s,h) \models \mathtt{alloc}_{\mathsf{y}}^{-1}(\mathsf{x})$ *iff* $s(\mathsf{x}) \in \mathrm{ran}(h)$.

(II) *In the translation* $\mathrm{T}(\varphi, Y)$*,* $\mathtt{alloc}^{-1}(\mathsf{x})$ *(with* $\mathsf{x} \in Y$*) can be replaced with* $\mathtt{alloc}_{\overline{\mathsf{x}}}^{-1}(\mathsf{x})$.

As stated in Lemma 3.8(II), we can exploit the fact that in the translation of a formula with variables in $\{\mathsf{x}_1, \ldots, \mathsf{x}_q\}$, we use $2q$ variables that correspond to $2q$ distinguished locations in the heap in order to retain the soundness of the translation while using $\mathtt{alloc}_{\overline{\mathsf{x}}}^{-1}(\mathsf{x})$ as $\mathtt{alloc}^{-1}(\mathsf{x})$.

Proof. For (I), ($\Rightarrow$): Suppose that $(s,h) \models \mathtt{alloc}_{\mathsf{y}}^{-1}(\mathsf{x})$. If (s,h) satisfies either $\mathsf{x} \hookrightarrow \mathsf{x}$ or $\mathsf{y} \hookrightarrow \mathsf{x}$, then obviously $s(\mathsf{x}) \in \mathrm{ran}(h)$. Otherwise, (s,h) must satisfy the third conjunct of $\mathtt{alloc}_{\mathsf{y}}^{-1}(\mathsf{x})$:

$$(s,h) \models \top * ((\mathtt{alloc}(\mathsf{y}) \wedge \mathtt{size} = 1) \mathrel{-\!\!\circledast} \mathtt{reach}(\mathsf{y},\mathsf{x}) = 2).$$

In this case, $(s,h') \models (\mathtt{alloc}(\mathsf{y}) \wedge \mathtt{size} = 1) \mathrel{-\!\!\circledast} \mathtt{reach}(\mathsf{y},\mathsf{x}) = 2$ holds for some heap $h' \sqsubseteq h$. From the semantics of the septraction operator, there is a heap h'' disjoint from h' and such that

$$(1)\ s(\mathsf{y}) \in \mathrm{dom}(h''), \qquad (2)\ \mathrm{card}(h'') = 1, \qquad (3)\ (s, h' + h'') \models \mathtt{reach}(\mathsf{y},\mathsf{x}) = 2.$$

From (1) and $h'' \perp h'$, we conclude that $s(\mathsf{y}) \notin \mathrm{dom}(h')$. Now, (3) implies that there is a location ℓ such that $\{s(\mathsf{y}) \mapsto \ell \mapsto s(\mathsf{x})\} \sqsubseteq h' + h''$, and moreover ℓ must be distinct from both $s(\mathsf{x})$ and $s(\mathsf{y})$ (which are also assumed to be distinct). From (1) and (2), it must be that $h'' = \{s(\mathsf{y}) \mapsto \ell\}$ and therefore $\{\ell \mapsto s(\mathsf{x})\} \sqsubseteq h'$. From $h' \sqsubseteq h$ we then conclude that $s(\mathsf{x}) \in \mathrm{ran}(h)$.

($\Leftarrow$): Suppose there is a location $\ell \in \mathrm{dom}(h)$ such that $h(\ell) = s(\mathsf{x})$ (i.e. $s(\mathsf{x}) \in \mathrm{ran}(h)$). First, suppose that $\ell = s(\mathsf{x})$ or $\ell = s(\mathsf{y})$. In this case, we directly derive that $(s,h) \models \mathsf{x} \hookrightarrow \mathsf{x} \vee \mathsf{y} \hookrightarrow \mathsf{x}$, which in turn shows that $(s,h) \models \mathtt{alloc}_{\mathsf{y}}^{-1}(\mathsf{x})$. Otherwise, consider the case where $\ell \neq s(\mathsf{x})$ and $\ell \neq s(\mathsf{y})$. Let $h' \sqsubseteq h$ be the heap $\{\ell \mapsto s(\mathsf{x})\}$. As $\ell \neq s(\mathsf{y})$, the location $s(\mathsf{y})$ is not a memory cell of h'. Let us consider the heap $h'' = \{s(\mathsf{y}) \mapsto \ell\}$, so that $(s,h'') \models \mathtt{alloc}(\mathsf{y}) \wedge \mathtt{size} = 1$. The heaps h' and h'' are disjoint, and from their definition we have $h' + h'' = \{s(\mathsf{y}) \mapsto \ell \mapsto s(\mathsf{x})\}$. As $s(\mathsf{x})$, ℓ and $s(\mathsf{y})$ are all distinct locations, $(s, h' + h'') \models \mathtt{reach}(\mathsf{y},\mathsf{x}) = 2$ holds, which in turn allows us to conclude that $(s,h') \models (\mathtt{alloc}(\mathsf{y}) \wedge \mathtt{size} = 1) \mathrel{-\!\!\circledast} \mathtt{reach}(\mathsf{y},\mathsf{x}) = 2$. As $h' \sqsubseteq h$, this implies that $(s,h) \models \top * ((\mathtt{alloc}(\mathsf{y}) \wedge \mathtt{size} = 1) \mathrel{-\!\!\circledast} \mathtt{reach}(\mathsf{y},\mathsf{x}) = 2)$, which implies $(s,h) \models \mathtt{alloc}_{\mathsf{y}}^{-1}(\mathsf{x})$.

For (II), since by definition, x and $\overline{\mathsf{x}}$ are interpreted by different locations, they satisfy the additional hypothesis $s(\mathsf{x}) \neq s(\mathsf{y})$ (where $\mathsf{y} = \overline{\mathsf{x}}$). Therefore, we can use one of these two variables to check if the other is in $\mathrm{ran}(h)$. As such, instead of $\mathtt{alloc}^{-1}(\mathsf{x})$ we can use $\mathtt{alloc}_{\overline{\mathsf{x}}}^{-1}(\mathsf{x})$ in the translation. □

Moreover, $\mathtt{alloc}_{\mathsf{y}}^{-1}(\mathsf{x})$ allows us to express in $\mathrm{SL}(*, \mathrel{-\!\!*}, \mathtt{ls})$ whether a location corresponding to a program variable reaches itself in exactly two steps (we use this property in the definition of $n(\mathsf{x}) \hookrightarrow n(\mathsf{y})$). We write $\mathsf{x} \hookrightarrow_{\mathsf{y}}^{2} \mathsf{x}$ to denote the formula

$$\neg(\mathsf{x} \hookrightarrow \mathsf{x}) \wedge (\mathsf{x} \hookrightarrow \mathsf{y} \Rightarrow \mathsf{y} \hookrightarrow \mathsf{x}) \wedge [\mathtt{alloc}(\mathsf{x}) \wedge \mathtt{alloc}_{\mathsf{y}}^{-1}(\mathsf{x}) \wedge (\top \mathrel{-\!\!*} \neg \mathtt{reach}(\mathsf{x},\mathsf{y}) = 2)]_2.$$

Lemma 3.9. *For any memory state* (s,h) *such that* $s(\mathsf{x}) \neq s(\mathsf{y})$*, we have* $(s,h) \models \mathsf{x} \hookrightarrow_{\mathsf{y}}^{2} \mathsf{x}$ *if and only if* $h^2(s(\mathsf{x})) = s(\mathsf{x})$ *and* $h(s(\mathsf{x})) \neq s(\mathsf{x})$.

Proof. So, let (s,h) be a memory state such that $s(\mathsf{x})$ is distinct from $s(\mathsf{y})$.

First, we assume that $h^2(s(\mathsf{x})) = s(\mathsf{x})$ and $h(s(\mathsf{x})) \neq s(\mathsf{x})$. So, there is a location ℓ distinct from $s(\mathsf{x})$ such that $h(s(\mathsf{x})) = \ell$ and $h(\ell) = s(\mathsf{x})$. Obviously, $(s,h) \models \neg(\mathsf{x} \hookrightarrow \mathsf{x})$ as ℓ is distinct from $s(\mathsf{x})$. Below, we distinguish two cases.

Case 1: $\ell \neq s(\mathsf{y})$**.** So, $(s,h) \models \neg \mathsf{x} \hookrightarrow \mathsf{y}$ and therefore $(s,h) \models \mathsf{x} \hookrightarrow \mathsf{y} \Rightarrow \mathsf{y} \hookrightarrow \mathsf{x}$. Let h' be the subheap of h with $\mathrm{dom}(h') = \{s(\mathsf{x}), \ell\}$. We have $(s,h') \models \mathtt{alloc}(\mathsf{x})$ and $(s,h') \models \mathtt{alloc}_{\mathsf{y}}^{-1}(\mathsf{x})$

by Lemma 3.8(I). Moreover, for all heaps h'' such that $h' \sqsubseteq h''$, we have $(s, h'') \not\models \mathtt{reach}(\mathtt{x}, \mathtt{y}) = 2$, as $\ell \in \mathrm{dom}(h')$ and $h' \sqsubseteq h''$.

Case 2: $\ell = s(\mathtt{y})$**.** So, $(s, h) \models \mathtt{x} \hookrightarrow \mathtt{y} \wedge \mathtt{y} \hookrightarrow \mathtt{x}$ and therefore $(s, h) \models \mathtt{x} \hookrightarrow \mathtt{y} \Rightarrow \mathtt{y} \hookrightarrow \mathtt{x}$. Let h' be the subheap of h with $\mathrm{dom}(h') = \{s(\mathtt{x}), \ell\}$. We have $(s, h) \models \mathtt{alloc}(\mathtt{x})$ and $(s, h) \models \mathtt{alloc}_{\mathtt{y}}^{-1}(\mathtt{x})$ by Lemma 3.8(I). Moreover, for all heaps h'' such that $h' \sqsubseteq h''$, we have $(s, h'') \not\models \mathtt{reach}(\mathtt{x}, \mathtt{y}) = 2$ as $(s, h'') \models \mathtt{x} \hookrightarrow \mathtt{y}$.

So, in both cases, $(s, h) \models [\mathtt{alloc}(\mathtt{x}) \wedge \mathtt{alloc}_{\mathtt{y}}^{-1}(\mathtt{x}) \wedge (\top \mathrel{-\!\!*} \neg \mathtt{reach}(\mathtt{x}, \mathtt{y}) = 2)]_2$ and therefore $(s, h) \models \mathtt{x} \hookrightarrow^2_{\mathtt{y}} \mathtt{x}$.

Conversely, assume that $(s, h) \models \mathtt{x} \hookrightarrow^2_{\mathtt{y}} \mathtt{x}$ and let us show that $s(\mathtt{x})$ can reach itself in two steps but not in one step.

Case 1: $(s, h) \models \mathtt{x} \hookrightarrow \mathtt{y} \wedge \mathtt{y} \hookrightarrow \mathtt{x}$**.** Let $\ell = s(\mathtt{y})$. So, $h(s(\mathtt{x})) = \ell$, $h(\ell) = s(\mathtt{x})$, and $\ell \neq s(\mathtt{x})$. Hence, we are done.

Case 2: $(s, h) \models \neg \mathtt{x} \hookrightarrow \mathtt{y}$**.** First, note that the case $\mathtt{x} \hookrightarrow \mathtt{y} \wedge \neg(\mathtt{y} \hookrightarrow \mathtt{x})$ is rule out because the memory state satisfies $\mathtt{x} \hookrightarrow \mathtt{y} \Rightarrow \mathtt{y} \hookrightarrow \mathtt{x}$. As $(s, h) \models [\mathtt{alloc}(\mathtt{x}) \wedge \mathtt{alloc}_{\mathtt{y}}^{-1}(\mathtt{x}) \wedge (\top \mathrel{-\!\!*} \neg \mathtt{reach}(\mathtt{x}, \mathtt{y}) = 2)]_2$, we conclude that $(s, h) \models \mathtt{alloc}(\mathtt{x})$ and therefore $s(\mathtt{x}) \in \mathrm{dom}(h)$. Similarly, we can conclude that $(s, h) \models \mathtt{alloc}_{\mathtt{y}}^{-1}(\mathtt{x})$. Let $\ell = h(s(\mathtt{x}))$ and by assumption ℓ is distinct from $s(\mathtt{y})$ and ℓ is different from $s(\mathtt{x})$, whence $h(s(\mathtt{x})) \neq s(\mathtt{x})$. *Ad absurdum*, suppose that either $\ell \notin \mathrm{dom}(h)$ or $h(\ell) \neq s(\mathtt{x})$. So, there is a location ℓ' distinct from ℓ such that $h(\ell') = s(\mathtt{x})$ (indeed, we have $(s, h) \models \mathtt{alloc}_{\mathtt{y}}^{-1}(\mathtt{x})$ and then use Lemma 3.8(I)). As $(s, h) \models [\mathtt{alloc}(\mathtt{x}) \wedge \mathtt{alloc}_{\mathtt{y}}^{-1}(\mathtt{x}) \wedge (\top \mathrel{-\!\!*} \neg \mathtt{reach}(\mathtt{x}, \mathtt{y}) = 2)]_2$, the only heap h' with two memory cells satisfying $(s, h') \models \mathtt{alloc}(\mathtt{x}) \wedge \mathtt{alloc}_{\mathtt{y}}^{-1}(\mathtt{x})$ is the one with $\mathrm{dom}(h') = \{s(\mathtt{x}), \ell'\}$ and therefore $\ell \notin \mathrm{dom}(h')$, which is important at this point. Let h'' be any heap disjoint from h' such that $h''(\ell) = s(\mathtt{y})$. We know such a heap exists as $\ell \notin \mathrm{dom}(h')$. As $s(\mathtt{x})$, ℓ and $s(\mathtt{y})$ are pairwise distinct, we get that $(s, h'') \models \mathtt{reach}(\mathtt{x}, \mathtt{y}) = 2$ and therefore, $(s, h') \not\models (\top \mathrel{-\!\!*} \neg \mathtt{reach}(\mathtt{x}, \mathtt{y}) = 2)$, which leads to a contradiction. Consequently, $\ell \in \mathrm{dom}(h)$ and $h(\ell) = s(\mathtt{x})$, so $h^2(s(\mathtt{x})) = s(\mathtt{x})$. □

The predicate $n(\mathtt{x}) = n(\mathtt{y})$ can be defined in $\mathrm{SL}(*, \mathrel{-\!\!*}, \mathtt{ls})$ as the formula $\mathrm{ext}(n(\mathtt{x}) = n(\mathtt{y}))$ ('ext' stands for 'extension')

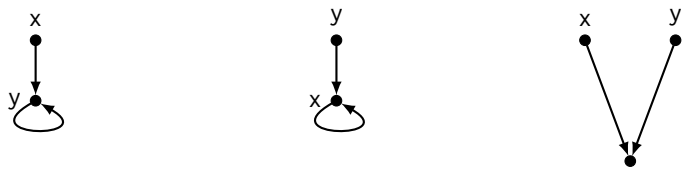
$$(\mathtt{x} \neq \mathtt{y} \Rightarrow [\mathtt{alloc}(\mathtt{x}) \wedge \mathtt{alloc}(\mathtt{y}) \wedge ((\mathtt{x} \hookrightarrow \mathtt{y} \wedge \mathtt{y} \hookrightarrow \mathtt{y}) \vee (\mathtt{y} \hookrightarrow \mathtt{x} \wedge \mathtt{x} \hookrightarrow \mathtt{x}) \vee$$
$$((\bigwedge_{\mathtt{z},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

Lemma 3.10. *Let* $\mathtt{x}, \mathtt{y} \in \mathrm{PVAR}$. *For all memory states* (s, h)*, we have* $(s, h) \models \mathrm{ext}(n(\mathtt{x}) = n(\mathtt{y}))$ *iff* $h(s(\mathtt{x})) = h(s(\mathtt{y}))$.

Proof. First, suppose $h(s(\mathtt{x})) = h(s(\mathtt{y}))$. Then obviously $(s, h) \models \mathtt{alloc}(\mathtt{x}) \wedge \mathtt{alloc}(\mathtt{y})$. Suppose $s(\mathtt{x}) \neq s(\mathtt{y})$. We need to show that (s, h) satisfies the formula

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

Let $h' \sqsubseteq h$ be the two-memory-cells heap such that $h'(s(\mathtt{x})) = h'(s(\mathtt{y}))$. In particular, $\mathrm{dom}(h') = \{s(\mathtt{x}), s(\mathtt{y})\}$ and therefore $(s, h') \models \mathtt{alloc}(\mathtt{x}) \wedge \mathtt{alloc}(\mathtt{y})$. Moreover, h' is represented by one of the following memory states.



Each memory state above satisfies one of the three disjuncts from the third conjunct of the above formula. The first memory state satisfies $\mathsf{x} \hookrightarrow \mathsf{y} \wedge \mathsf{y} \hookrightarrow \mathsf{y}$, the second one satisfies $\mathsf{y} \hookrightarrow \mathsf{x} \wedge \mathsf{x} \hookrightarrow \mathsf{x}$ and the third one satisfies

$$(\bigwedge_{\mathsf{z},\mathsf{z}' \in \{\mathsf{x},\mathsf{y}\}} \neg \mathsf{z} \hookrightarrow \mathsf{z}') \wedge (\top \mathrel{-\!\!*} \neg(\mathtt{reach}(\mathsf{x},\mathsf{y}) = 2 \wedge \mathtt{reach}(\mathsf{y},\mathsf{x}) = 2)).$$

The first two cases are trivial. The last one represents a memory state with a location ℓ such that $\ell = h'(s(\mathsf{x})) = h'(s(\mathsf{y}))$ and $s(\mathsf{x}) \neq \ell \neq s(\mathsf{y})$. As such, this memory state trivially satisfies $\bigwedge_{\mathsf{z},\mathsf{z}' \in \{\mathsf{x},\mathsf{y}\}} \neg \mathsf{z} \hookrightarrow \mathsf{z}'$. Now, consider h'' disjoint from h' and $(s, h' + h'') \models \mathtt{reach}(\mathsf{x},\mathsf{y}) = 2$. In particular, it must hold that $h''(\ell) = s(\mathsf{y})$. As such, $(h' + h'')^2(s(\mathsf{y})) = s(\mathsf{y})$ and so $(s, h' + h'') \not\models \mathtt{reach}(\mathsf{y},\mathsf{x}) = 2$. It follows that the last memory state of the picture satisfies $(\bigwedge_{\mathsf{z},\mathsf{z}' \in \{\mathsf{x},\mathsf{y}\}} \neg \mathsf{z} \hookrightarrow \mathsf{z}') \wedge (\top \mathrel{-\!\!*} \neg(\mathtt{reach}(\mathsf{x},\mathsf{y}) = 2 \wedge \mathtt{reach}(\mathsf{y},\mathsf{x}) = 2))$. Thus, $h(s(\mathsf{x})) = h(s(\mathsf{y}))$ implies $(s, h) \models \mathrm{ext}(n(\mathsf{x}) = n(\mathsf{y}))$.

Conversely, suppose $(s, h) \models \mathrm{ext}(n(\mathsf{x}) = n(\mathsf{y}))$. Then $(s, h) \models \mathtt{alloc}(\mathsf{x})$. If $s(\mathsf{x}) = s(\mathsf{y})$, $h(s(\mathsf{x})) = h(s(\mathsf{y}))$ follows trivially. Instead, if $s(\mathsf{x}) \neq s(\mathsf{y})$, there must exist a two-memory-cells heap $h' \sqsubseteq h$ such that (s, h') satisfies $\mathtt{alloc}(\mathsf{x}) \wedge \mathtt{alloc}(\mathsf{y}) \wedge ((\mathsf{x} \hookrightarrow \mathsf{y} \wedge \mathsf{y} \hookrightarrow \mathsf{y}) \vee (\mathsf{y} \hookrightarrow \mathsf{x} \wedge \mathsf{x} \hookrightarrow \mathsf{x}) \vee ((\bigwedge_{\mathsf{z},\mathsf{z}' \in \{\mathsf{x},\mathsf{y}\}} \neg \mathsf{z} \hookrightarrow \mathsf{z}') \wedge (\top \mathrel{-\!\!*} \neg(\mathtt{reach}(\mathsf{x},\mathsf{y}) = 2 \wedge \mathtt{reach}(\mathsf{y},\mathsf{x}) = 2))))$. As such, $s(\mathsf{x})$ and $s(\mathsf{y})$ are both in $\mathrm{dom}(h')$. Trivially, if $(s, h') \models (\mathsf{x} \hookrightarrow \mathsf{y} \wedge \mathsf{y} \hookrightarrow \mathsf{y}) \vee (\mathsf{y} \hookrightarrow \mathsf{x} \wedge \mathsf{x} \hookrightarrow \mathsf{x})$ then $h'(s(\mathsf{x})) = h'(s(\mathsf{y}))$. The same holds true if $(s, h') \models (\bigwedge_{\mathsf{z},\mathsf{z}' \in \{\mathsf{x},\mathsf{y}\}} \neg \mathsf{z} \hookrightarrow \mathsf{z}') \wedge (\top \mathrel{-\!\!*} \neg(\mathtt{reach}(\mathsf{x},\mathsf{y}) = 2 \wedge \mathtt{reach}(\mathsf{y},\mathsf{x}) = 2))$. Indeed, $(s, h') \models \bigwedge_{\mathsf{z},\mathsf{z}' \in \{\mathsf{x},\mathsf{y}\}} \neg \mathsf{z} \hookrightarrow \mathsf{z}'$ leaves open only two possible memory states, that are represented below.

The last part of the formula, $\top \mathrel{-\!\!*} \neg(\mathtt{reach}(\mathsf{x},\mathsf{y}) = 2 \wedge \mathtt{reach}(\mathsf{y},\mathsf{x}) = 2)$ allows to differentiate these two cases by excluding the left heap. Indeed, that very formula holds on (s, h') if and only if there is no heap h'' such that $h'' \perp h'$ and $(s, h' + h'') \models \mathtt{reach}(\mathsf{x},\mathsf{y}) = 2 \wedge \mathtt{reach}(\mathsf{y},\mathsf{x}) = 2$. This property holds on the right heap, as already discussed in the first part of the proof, but not on the left one, as can be shown by defining h'' as $\{h'(s(\mathsf{x})) \mapsto s(\mathsf{y}), h'(s(\mathsf{y})) \mapsto s(\mathsf{x})\}$. □

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

$$\begin{aligned}\varphi^{=}_{\hookrightarrow}(\mathsf{x},\mathsf{y},\mathsf{z}) \stackrel{\text{def}}{=}\ & (\mathsf{x} \hookrightarrow \mathsf{x} \wedge \mathsf{y} \hookrightarrow \mathsf{x}) \vee (\mathsf{y} \hookrightarrow \mathsf{y} \wedge \mathsf{x} \hookrightarrow \mathsf{y}) \vee (\mathsf{x} \hookrightarrow \mathsf{z} \wedge \mathsf{z} \hookrightarrow \mathsf{z}) \\ & \vee [\mathtt{alloc}(\mathsf{x}) \wedge \neg \mathtt{alloc}^{-1}_{\mathsf{z}}(\mathsf{x}) \wedge (\top \mathrel{-\!\!*} \neg\mathtt{reach}(\mathsf{x},\mathsf{z}) \leq 3)]_2\end{aligned}$$

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

$$\varphi^{\neq}_{\hookrightarrow}(x,y) \stackrel{\text{def}}{=} (x \hookrightarrow y \wedge \mathtt{alloc}(y)) \vee (y \hookrightarrow y \wedge \mathtt{reach}(x,y) = 2) \vee (y \hookrightarrow x \wedge x \hookrightarrow^2_y x) \vee$$
$$[\mathtt{alloc}(x) \wedge \mathtt{alloc}(y) \wedge (\textstyle\bigwedge_{z,z' \in \{x,y\}} \neg z \hookrightarrow z') \wedge \neg \mathtt{reach}(x,y) \leq 3$$
$$\wedge ((\mathtt{size} = 1 \wedge \mathtt{alloc}^{-1}_x(y)) \circledast (\mathtt{reach}(x,y) = 3 \wedge y \hookrightarrow^2_x y))]_3$$

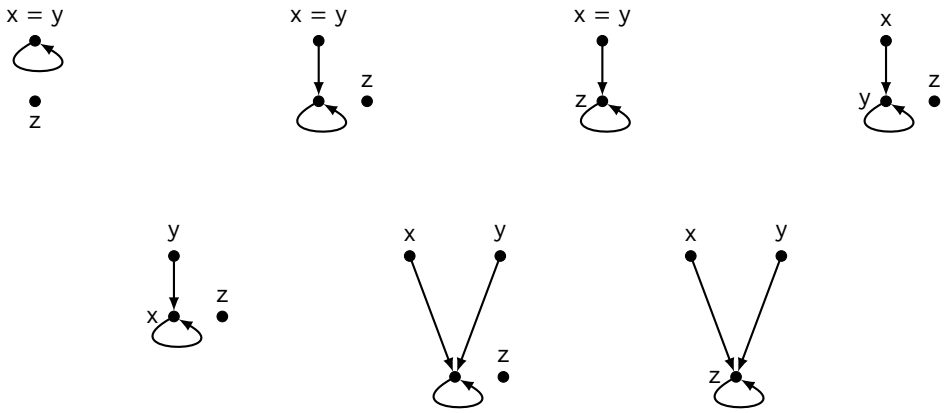
Lemma 3.11. *Let* $x, y, z \in \mathrm{PVAR}$.

(I) *For all memory states* (s, h) *such that* $s(x) \neq s(z)$ *and* $s(y) \neq s(z)$, *we have* $(s, h) \models \varphi_{\hookrightarrow}(x, y, z)$ *iff* $\{s(x), s(y)\} \subseteq \mathrm{dom}(h)$ *and* $h(h(s(x))) = h(s(y))$.

(II) *In the translation* $\mathrm{T}(\varphi, X)$, $n(x) \hookrightarrow n(y)$ *can be replaced by* $\varphi_{\hookrightarrow}(x, y, \overline{x})$.

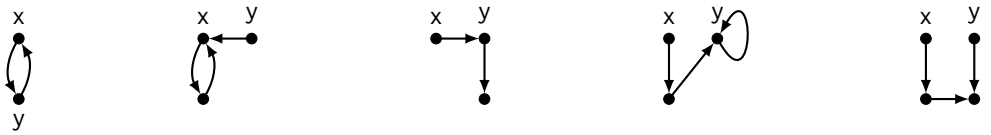
Proof. (I) First, suppose $\{s(x), s(y)\} \subseteq \mathrm{dom}(h)$, $h(h(s(x))) = h(s(y))$, $s(x) \neq s(z)$ and $s(y) \neq s(z)$. Let us distinguish two cases.

Case 1: $h(s(x)) = h(s(y))$. Equivalently, $\mathrm{ext}(n(x) = n(y))$ holds from Lemma 3.10 and we must prove that (s, h) satisfies $\varphi^{=}_{\hookrightarrow}(x, y, z)$. Consider the subheap $h' \sqsubseteq h$ such that $\mathrm{dom}(h') = \{s(x), s(y), h(s(x))\}$. There are only a bounded number of subheaps of this kind (up to isomorphism with respect to $\{x, y, z\}$). Remember that $h(s(x)) = h(s(y))$, $h(h(s(x))) = h(s(y))$, $s(x) \neq s(z)$ and $s(y) \neq s(z)$.



We need to check that for each case, the memory state satisfies one of the disjuncts of the formula $\varphi^{=}_{\hookrightarrow}(x, y, z)$. Indeed, it is easy to see that the first and fifth memory states (enumerated from the top left to the bottom right) satisfy $x \hookrightarrow x \wedge y \hookrightarrow x$, the third and seventh memory states satisfy $x \hookrightarrow z \wedge z \hookrightarrow z$ and the fourth memory state satisfies $y \hookrightarrow y \wedge x \hookrightarrow y$. Lastly, the second and sixth memory states satisfy $[\mathtt{alloc}(x) \wedge \neg \mathtt{alloc}^{-1}_z(x) \wedge (\top \mathrel{-\!\!*} \neg \mathtt{reach}(x, z) \leq 3)]_2$. For these last two cases, consider the two-memory-cells heap $h'' = \{s(x) \mapsto h'(s(x)), h'(s(x)) \mapsto h'(s(x))\} \sqsubseteq h'$. Trivially, $(s, h'') \models \mathtt{alloc}(x) \wedge \neg \mathtt{alloc}^{-1}_z(x)$. Moreover, since $h''(h'(s(x))) = h'(s(x))$, for any heap h''' such that $h''' \perp h''$, $h''(h'(s(x)))$ cannot be in $\mathrm{dom}(h''')$ and therefore $(s, h'' + h''')$ does not satisfy $\mathtt{reach}(x, z) \leq 3$. It follows that (s, h'') satisfies $(\top \mathrel{-\!\!*} \neg \mathtt{reach}(x, z) \leq 3)$.

Case 2: $h(s(x)) \neq h(s(y))$. Equivalently, from Lemma 3.10, $\mathrm{ext}(n(x) = n(y))$ does not hold and we must prove that $(s, h) \models \varphi^{\neq}_{\hookrightarrow}(x, y)$.
Moreover, $s(x) \neq s(y)$. Consider the subheap $h' \sqsubseteq h$ such that $\mathrm{dom}(h') = \{s(x), s(y), h(s(x))\}$. There are only a bounded number of subheaps of this kind (up to isomorphism with respect to $\{x, y\}$). Remember that $h(h(s(x))) = h(s(y))$ and $h(s(x)) \neq h(s(y))$.



Let us check that for each of the five cases, one of the disjuncts of $\varphi^{\neq}_{\hookrightarrow}(\mathsf{x},\mathsf{y})$ holds. Trivially, the first and third memory states satisfy the first disjunct $\mathsf{x} \hookrightarrow \mathsf{y} \wedge \mathtt{alloc}(\mathsf{y})$, the second one satisfies $\mathsf{y} \hookrightarrow \mathsf{x} \wedge \mathsf{x} \hookrightarrow^2_{\mathsf{y}} \mathsf{x}$, whereas the fourth one satisfies $\mathsf{y} \hookrightarrow \mathsf{y} \wedge \mathtt{reach}(\mathsf{x},\mathsf{y}) = 2$. Lastly, the fifth one satisfies the disjunct

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

For the fifth memory state, trivially $(s,h') \models \mathtt{alloc}(\mathsf{x}) \wedge \mathtt{alloc}(\mathsf{y}) \wedge (\bigwedge_{\mathsf{z},\mathsf{z}' \in \{\mathsf{x},\mathsf{y}\}} \neg \mathsf{z} \hookrightarrow \mathsf{z}') \wedge \neg\mathtt{reach}(\mathsf{x},\mathsf{y}) \leq 3$. Let $h'' = \{h'(h'(s(\mathsf{x}))) \mapsto s(\mathsf{y})\}$. As $h'(h'(s(\mathsf{x}))) \notin \mathrm{dom}(h')$, it holds that $h'' \bot h'$. Moreover, (s,h'') satisfies $\mathtt{size} = 1 \wedge \mathtt{alloc}^{-1}_{\mathsf{x}}(\mathsf{y})$. Consider now $(s, h' + h'')$. Since $h''(h'(h'(s(\mathsf{x})))) = s(\mathsf{y})$, this memory state satisfies $\mathtt{reach}(\mathsf{x},\mathsf{y}) = 3$, whereas $\mathsf{y} \hookrightarrow^2_{\mathsf{x}} \mathsf{y}$ is satisfied from the hypothesis $s(\mathsf{x}) \neq s(\mathsf{y})$ and $h(h(s(\mathsf{x}))) = h(s(\mathsf{y}))$. Indeed, $h''(h'(s(\mathsf{y}))) = h''(h'(h'(s(\mathsf{x})))) = s(\mathsf{y})$. We derive $(s,h') \models ((\mathtt{size} = 1 \wedge \mathtt{alloc}^{-1}_{\mathsf{x}}(\mathsf{y})) \circledast (\mathtt{reach}(\mathsf{x},\mathsf{y}) = 3 \wedge \mathsf{y} \hookrightarrow^2_{\mathsf{x}} \mathsf{y}))$.

Conversely, let us suppose that $(s,h) \models \varphi_{\hookrightarrow}(\mathsf{x},\mathsf{y},\mathsf{z})$, $s(\mathsf{x}) \neq s(\mathsf{z})$ and $s(\mathsf{y}) \neq s(\mathsf{z})$. Two cases are considered.

Case 1: $(s,h) \models \mathrm{ext}(n(\mathsf{x}) = n(\mathsf{y})) \wedge \varphi^{=}_{\hookrightarrow}(\mathsf{x},\mathsf{y},\mathsf{z})$. If one of the three first disjuncts of $\varphi^{=}_{\hookrightarrow}(\mathsf{x},\mathsf{y},\mathsf{z})$ holds, we have immediately $\{s(\mathsf{x}), s(\mathsf{y})\} \subseteq \mathrm{dom}(h)$ and $h(h(s(\mathsf{x}))) = h(s(\mathsf{y}))$. Indeed, let us consider the three possible situations.

(1) In the case $(s,h) \models \mathrm{ext}(n(\mathsf{x}) = n(\mathsf{y})) \wedge \mathsf{x} \hookrightarrow \mathsf{x} \wedge \mathsf{y} \hookrightarrow \mathsf{x}$, we get $h(h(s(\mathsf{x}))) = h(s(\mathsf{y})) = s(\mathsf{x})$ and therefore $(s,h) \models n(\mathsf{x}) \hookrightarrow n(\mathsf{y})$.

(2) In the case $(s,h) \models \mathrm{ext}(n(\mathsf{x}) = n(\mathsf{y})) \wedge \mathsf{y} \hookrightarrow \mathsf{y} \wedge \mathsf{x} \hookrightarrow \mathsf{y}$, we get $h(h(s(\mathsf{x}))) = h(s(\mathsf{y})) = s(\mathsf{y})$ and therefore $(s,h) \models n(\mathsf{x}) \hookrightarrow n(\mathsf{y})$.

(3) In the case $(s,h) \models \mathrm{ext}(n(\mathsf{x}) = n(\mathsf{y})) \wedge \mathsf{x} \hookrightarrow \mathsf{z} \wedge \mathsf{z} \hookrightarrow \mathsf{z}$, we get $h(h(s(\mathsf{x}))) = h(s(\mathsf{y})) = s(\mathsf{z})$ and therefore $(s,h) \models n(\mathsf{x}) \hookrightarrow n(\mathsf{y})$.

The remaining case is when none of the three first disjuncts holds and therefore $(s,h) \models [\mathtt{alloc}(\mathsf{x}) \wedge \neg\mathtt{alloc}^{-1}_{\mathsf{z}}(\mathsf{x}) \wedge (\top \mathrel{-\!\!*} \neg\mathtt{reach}(\mathsf{x},\mathsf{z}) \leq 3)]_2$. Let $h' \sqsubseteq h$ be some heap with $\mathrm{card}(\mathrm{dom}(h')) = 2$ and $(s,h') \models \mathtt{alloc}(\mathsf{x}) \wedge \neg\mathtt{alloc}^{-1}_{\mathsf{z}}(\mathsf{x}) \wedge (\top \mathrel{-\!\!*} \neg\mathtt{reach}(\mathsf{x},\mathsf{z}) \leq 3)$. Obviously, $s(\mathsf{x}) \in \mathrm{dom}(h')$, say $h'(s(\mathsf{x})) = \ell$ and $\ell \in \mathrm{dom}(h')$ (otherwise $(s, h' + \{\ell \mapsto s(\mathsf{z})\}) \models \mathtt{reach}(\mathsf{x},\mathsf{z}) = 2$, which leads to a contradiction). We can assume that ℓ is distinct from $s(\mathsf{x})$ as the first disjunct of $\varphi^{=}_{\hookrightarrow}(\mathsf{x},\mathsf{y},\mathsf{z})$ does not hold. Similarly, $(s,h') \models \neg(\mathsf{x} \hookrightarrow \mathsf{z}) \wedge \neg(\mathtt{reach}(\mathsf{x},\mathsf{z}) = 2)$, otherwise we reach a contradiction with $(s,h') \models \top \mathrel{-\!\!*} \neg\mathtt{reach}(\mathsf{x},\mathsf{z}) \leq 3$. To sump up, the heap h' satisfies the following properties:

- $\mathrm{card}(\mathrm{dom}(h')) = 2$, $s(\mathsf{x}) \in \mathrm{dom}(h')$ and $h'(s(\mathsf{x})) = \ell$ with $\ell \neq s(\mathsf{x})$ and $\ell \in \mathrm{dom}(h')$.
- $(s,h') \models \neg\mathtt{alloc}^{-1}_{\mathsf{z}}(\mathsf{x}) \wedge (\top \mathrel{-\!\!*} \neg\mathtt{reach}(\mathsf{x},\mathsf{z}) \leq 3) \wedge \neg(\mathsf{x} \hookrightarrow \mathsf{z}) \wedge \neg(\mathtt{reach}(\mathsf{x},\mathsf{z}) = 2)$.

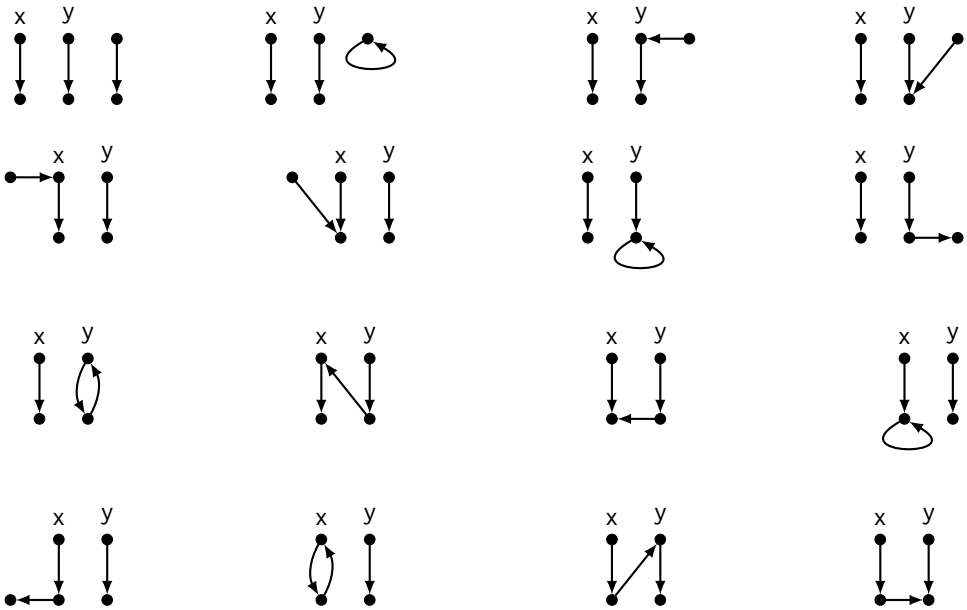
If $h(h(s(\mathsf{x}))) \neq h(s(\mathsf{x}))$ or $h(h(s(\mathsf{x}))) = s(\mathsf{x})$, then we would have $(s,h') \models \mathtt{alloc}^{-1}_{\mathsf{z}}(\mathsf{x}) \vee (\top \circledast (\mathtt{reach}(\mathsf{x},\mathsf{z}) \leq 3))$, which is precisely excluded from above. Consequently, we conclude that necessarily $h(h(s(\mathsf{x}))) = h(s(\mathsf{x})) \neq s(\mathsf{x})$. Moreover, by Lemma 3.10, $(s,h) \models \mathrm{ext}(n(\mathsf{x}) = n(\mathsf{y}))$ implies that $h(s(\mathsf{x})) = h(s(\mathsf{y}))$. $h(h(s(\mathsf{x}))) = h(s(\mathsf{y}))$ and $\{s(\mathsf{x}), s(\mathsf{y})\} \subseteq \mathrm{dom}(h)$ follows.

Case 2: $(s,h) \models \neg\mathrm{ext}(n(\mathsf{x}) = n(\mathsf{y})) \wedge \varphi^{\neq}_{\hookrightarrow}(\mathsf{x},\mathsf{y})$. This implies $s(\mathsf{x}) \neq s(\mathsf{y})$. If one of the three first disjuncts of $\varphi^{\neq}_{\hookrightarrow}(\mathsf{x},\mathsf{y})$ holds, we have immediately $\{s(\mathsf{x}), s(\mathsf{y})\} \subseteq \mathrm{dom}(h)$ and $h(h(s(\mathsf{x}))) = h(s(\mathsf{y}))$. The remaining case is when none of the three first disjuncts holds and (s,h) satisfies

the fourth one. So, there is a heap $h' \sqsubseteq h$ with $\mathrm{card}(\mathrm{dom}(h')) = 3$ such that (s, h') satisfies

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

Since $(s, h) \models \neg \mathrm{ext}(n(\mathtt{x}) = n(\mathtt{y}))$, the only memory states with three memory cells satisfying $\mathtt{alloc}(\mathtt{x}) \wedge \mathtt{alloc}(\mathtt{y}) \wedge (\bigwedge_{\mathtt{z},\mathtt{z}' \in \{\mathtt{x},\mathtt{y}\}} \neg \mathtt{z} \hookrightarrow \mathtt{z}')$ are the following (up to isomorphism with respect to $\{\mathtt{x}, \mathtt{y}\}$):



The formula $(\mathtt{size} = 1 \wedge \mathtt{alloc}_{\mathtt{x}}^{-1}(\mathtt{y})) \mathrel{-\!\!\circledast} (\mathtt{reach}(\mathtt{x},\mathtt{y}) = 3 \wedge \mathtt{y} \hookrightarrow_{\mathtt{x}}^{2} \mathtt{y})$ can only be satisfied on heaps h' where there exists a way of adding a one-memory-cell heap h'' with $s(\mathtt{y}) \in \mathrm{ran}(h'')$ and $(h' + h'')^3(s(\mathtt{x})) = s(\mathtt{y})$. This rules out all the memory states of the figure but the first and last one of the last row:

Typically, h'' can only take the value $\{h'(h'(s(\mathtt{x}))) \mapsto s(\mathtt{y})\}$. However, only the second memory state is able to verify the condition $(s, h' + h'') \models \mathtt{y} \hookrightarrow_{\mathtt{x}}^{2} \mathtt{y}$. Then $h'(s(\mathtt{y})) = h'(h'(s(\mathtt{x})))$ and we conclude that $\{s(\mathtt{x}), s(\mathtt{y})\} \subseteq \mathrm{dom}(h)$ and $h(h(s(\mathtt{x}))) = h(s(\mathtt{y}))$.

(II) It is easy to show that $n(\mathtt{x}) \hookrightarrow n(\overline{\mathtt{x}})$ will never occur in the translation. Indeed, by always translating formulae with variables in $\{\mathtt{x}_1, \ldots, \mathtt{x}_q\}$ we can only have $n(\mathtt{x}_i) \hookrightarrow n(\mathtt{x}_j)$ or $n(\overline{\mathtt{x}_i}) \hookrightarrow n(\overline{\mathtt{x}_j})$ (the latter case due to the renaming in the translation of the left side of the magic wand). As such, $\varphi_{\hookrightarrow}(\mathtt{x}, \mathtt{y}, \overline{\mathtt{x}})$ can be used to check whenever $n(\mathtt{x}) \hookrightarrow n(\mathtt{y})$. □

As for $\mathtt{alloc}_{\mathtt{y}}^{-1}(\mathtt{x})$, 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 3.11(II)). By looking at the formulae herein defined, the predicate 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 SL($*, -\!\!*$) enriched with $\mathtt{reach}(\mathsf{x}, \mathsf{y}) = 2$ and $\mathtt{reach}(\mathsf{x}, \mathsf{y}) = 3$. Observe that $\mathtt{reach}(\mathsf{x}, \mathsf{y}) = 0$ and $\mathtt{reach}(\mathsf{x}, \mathsf{y}) = 1$ are already definable in the logic SL($*, -\!\!*$).

3.5 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 3.7 and the undecidability of SL($\forall, -\!\!*$), here is one of the main results of the paper.

Theorem 3.12. *The satisfiability problem for* SL($*, -\!\!*, \mathtt{ls}$) *is undecidable.*

As a by-product, we get the following (negative) result.

Theorem 3.13. *The set of valid formulae for* SL($*, -\!\!*, \mathtt{ls}$) *is not recursively enumerable.*

Indeed, suppose that the set of valid formulae for SL($*, -\!\!*, \mathtt{ls}$) were recursively enumerable, then one can enumerate the valid formulae of the form $\mathcal{T}_{\mathrm{VAL}}(\varphi)$ as it is decidable in PTime whether ψ in SL($*, -\!\!*, \mathtt{ls}$) is syntactically equal to $\mathcal{T}_{\mathrm{VAL}}(\varphi)$ for some SL($\forall, -\!\!*$) formula φ. This leads to a contradiction since this would allow the enumeration of valid formulae in SL($\forall, -\!\!*$). Note also that the set of satisfiable formulae for SL($*, -\!\!*, \mathtt{ls}$) is not recursively enumerable. Indeed, suppose φ is a formula built over $V = \{\mathsf{x}_1, \ldots, \mathsf{x}_n\}$. Given a partition X of V, we write ψ_X to denote conjunctions of equalities and inequalities that state that two variables (resp. not) in the same element of the partition are equal (resp. are different). One can show that φ is valid iff $\mathtt{emp} \wedge \psi_X \wedge (\top -\!\!* \varphi)$ is satisfiable, for all partitions X. As the set of valid formulae is not r.e., the same applies to the set of satisfiable formulae.

Below, the essential ingredients to establish the undecidability of SL($*, -\!\!*, \mathtt{ls}$) allow us to propose several refinements. For instance, the key property rests on the fact that the following properties $n(\mathsf{x}) = n(\mathsf{y})$, $n(\mathsf{x}) \hookrightarrow n(\mathsf{y})$ and $\mathtt{alloc}^{-1}(\mathsf{x})$ are expressible in the logic.

Corollary 3.14. SL($*, -\!\!*$) *augmented with formulae of the form* $n(\mathsf{x}) = n(\mathsf{y})$, $n(\mathsf{x}) \hookrightarrow n(\mathsf{y})$ *and* $\mathtt{alloc}^{-1}(\mathsf{x})$ *admits an undecidable satisfiability problem.*

Corollary 3.14 can be refined a bit further by noting in which context the reachability predicates ls and reach are used in the formulae. This allows us to get the result below.

Corollary 3.15. SL($*, -\!\!*$) *augmented with built-in formulae of the form* $\mathtt{reach}(\mathsf{x}, \mathsf{y}) = 2$ *and* $\mathtt{reach}(\mathsf{x}, \mathsf{y}) = 3$ *admits an undecidable satisfiability problem.*

It is the addition of $\mathtt{reach}(\mathsf{x}, \mathsf{y}) = 3$ that is crucial for undecidability since the satisfiability problem for SL($*, -\!\!*, \mathtt{reach}(\mathsf{x}, \mathsf{y}) = 2$) is in PSpace [17].

Following a similar analysis, let SL1($\forall, *, -\!\!*$) (resp. SL2($\forall, *, -\!\!*$)) be the restriction of SL($\forall, *, -\!\!*$) (i.e. SL($\forall, -\!\!*$) plus $*$) to formulae of the form $\exists \mathsf{x}_1 \cdots \exists \mathsf{x}_q \, \varphi$, where $q \geq 1$, the variables in φ are among $\{\mathsf{x}_1, \ldots, \mathsf{x}_{q+1}\}$ (resp. are among $\{\mathsf{x}_1, \ldots, \mathsf{x}_{q+2}\}$) and the only quantified variable in φ is x_{q+1} (resp. and the only quantified variables in φ are x_{q+1} and x_{q+2}). Note that SL2($\forall, *, -\!\!*$) should not be confused with SL($\forall, -\!\!*$) restricted to two quantified variables. The satisfiability problem for SL1($\forall, *, -\!\!*$) is PSpace-complete [17]. Observe that SL1($\forall, *, -\!\!*$) can easily express $n(\mathsf{x}) = n(\mathsf{y})$ and $\mathtt{alloc}^{-1}(\mathsf{x})$. The distance between the decidability for SL1($\forall, *, -\!\!*$) and the undecidability for SL($*, -\!\!*, \mathtt{ls}$), is best witnessed by Corollary 3.16(I) below, which solves an open problem [17, Section 6]. Below, we state several undecidability results for the record, but we do not claim that all these variants happen to be interesting in practice.

Corollary 3.16. *The satisfiability problem of the following logics is undecidable:*

(I) *SL1($\forall, *, -\!\!*$) augmented with* $n(\mathsf{x}) \hookrightarrow n(\mathsf{y})$,

(II) *SL1*$(\forall, *, -\!\!*)$ *augmented with* $\mathtt{ls}$,
(III) *SL2*$(\forall, *, -\!\!*)$,
(IV) $\mathrm{SL}(*, -\!\!*, \mathtt{ls})$ *restricted to four program variables,*
(V) $\mathrm{SL}(-\!\!*, n(\mathsf{x}) = n(\mathsf{y}), n(\mathsf{x}) \hookrightarrow n(\mathsf{y}), \mathtt{alloc}^{-1}(\mathsf{x}))$.

Proof. (I) Consequence of Corollary 3.14 by observing that $\mathtt{emp}$, $n(\mathsf{x}) = n(\mathsf{y})$ and $\mathtt{alloc}^{-1}(\mathsf{x})$ can be expressed with one quantified variable.

(II) Consequence of (I), as $n(\mathsf{x}) \hookrightarrow n(\mathsf{y})$ can be expressed with $\mathtt{ls}$.

(III) Consequence of (I), as $n(\mathsf{x}) \hookrightarrow n(\mathsf{y})$ can be expressed with two quantified variables.

(IV) It is shown in [16] that $\mathrm{SL}(\forall, -\!\!*)$ restricted to two quantified variables is undecidable. The translation provided in Section 3.3 assumes that distinct quantifications involve distinct variables. In order to translate $\mathrm{SL}(\forall, -\!\!*)$ restricted to two quantified variables, it is necessary to give up that assumption and to update the definition of T. Actually, only the clause for formulae of the form $\forall \mathsf{x}_i\, \psi$ requires a change ($i \in \{1, 2\}$ and the formulae to be translated contains at most two variables). Here is the new value for $\mathrm{T}(\forall \mathsf{x}_i\, \psi, X)$:

$$((\mathtt{alloc}(\mathsf{x}_i) \wedge \mathtt{size} = 1) \vee \mathtt{emp}) * (\neg\mathtt{alloc}(\mathsf{x}_i) \wedge (\mathtt{alloc}(\mathsf{x}_i) \wedge \mathtt{size} = 1) -\!\!* (\mathrm{Safe}(X) \Rightarrow \mathrm{T}(\psi, X))).$$

The proof of Lemma 3.6 can be updated accordingly.

(V) In the translation T to $\mathrm{SL}(*, -\!\!*, \mathtt{ls})$ formulae, the separating connective $*$ appears only in the definition of $\mathrm{T}(\psi_1 -\!\!* \psi_2, X)$, assuming that $n(\mathsf{x}) = n(\mathsf{y})$, $n(\mathsf{x}) \hookrightarrow n(\mathsf{y})$ and $\mathtt{alloc}^{-1}(\mathsf{x})$ are built-in predicates. It is easy to check that if we remove

$$\text{“}(\bigwedge_{\mathsf{z} \in Z} \mathtt{alloc}(\mathsf{z}) \wedge \mathtt{size} = \mathrm{card}(Z)) * \text{”}$$

from $\mathrm{T}(\psi_1 -\!\!* \psi_2, X)$ (i.e., we disallow the recycling of variables), the proof of Lemma 3.6 can be updated accordingly. □

4 DECISION PROCEDURES IN PSPACE FOR $\mathrm{SL}(*, \mathtt{reach}^+)$ AND VARIANTS

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 that it is possible to push further the PSpace upper bound to the formulae expressible as a Boolean combination of formulae from the two fragments $\mathrm{SL}(*, \mathtt{reach}^+)$ and $\mathrm{SL}(*, -\!\!*)$. The latter logic is shown to be PSpace-complete in [28, 41]. Moreover, as shown in Section 2, $\mathrm{SL}(*, \mathtt{ls})$ can be understood as a fragment of $\mathrm{SL}(*, \mathtt{reach}^+)$ and therefore the PSpace upper bound for $\mathrm{SL}(*, \mathtt{reach}^+)$ established here also holds for $\mathrm{SL}(*, \mathtt{ls})$.

Our 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 [11, 41] and using test formulae as in [5, 17, 21, 28, 29].

4.1 Introduction to test formulae

In order to show the PSpace upper bound for $\mathrm{SL}(*, \mathtt{reach}^+)$, we would like to adapt the technique introduced in [28] for $\mathrm{SL}(*, -\!\!*)$, which relies on defining a set of *test formulae* capturing the essential properties expressible in $\mathrm{SL}(*, -\!\!*)$, as depicted by the following result.

Proposition 4.1. *[15, 28] Any formula* φ *in* $\mathrm{SL}(*, -\!\!*)$ *built over variables in* $\mathsf{x}_1, \ldots, \mathsf{x}_q$ *is logically equivalent to a Boolean combination of test formulae, i.e. formulae among* $\mathtt{size} \geq \beta$, $\mathtt{alloc}(\mathsf{x}_i)$, $\mathsf{x}_i \hookrightarrow \mathsf{x}_j$ *and* $\mathsf{x}_i = \mathsf{x}_j$, *with* $\beta \geq 0$ *and* $i, j \in [1, q]$.

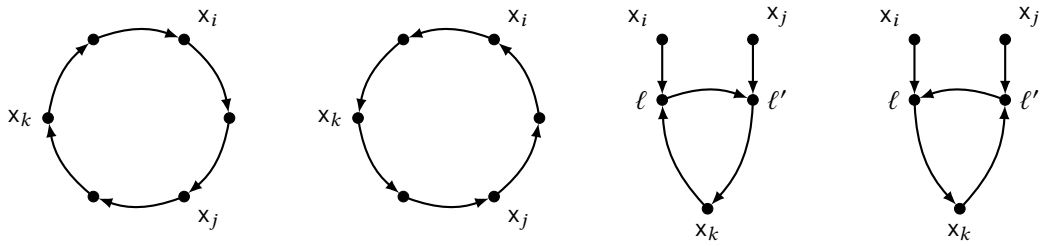


Fig. 4. Memory states $(s_1, h_1), \ldots, (s_4, h_4)$ (from left to right)

The formulae of the form $\mathtt{size} \geq \beta$ and $\mathtt{alloc}(\mathsf{x}_i)$ are introduced in Section 2 and $\mathtt{alloc}(\mathsf{x}_i)$ holds when $s(\mathsf{x}_i)$ belongs to the heap domain and $\mathtt{size} \geq \beta$ holds when the heap has at least β memory cells. By way of example, $(\neg\mathtt{emp} * ((\mathsf{x}_1 \hookrightarrow \mathsf{x}_1) \mathrel{-\!\!*} \bot))$ is equivalent to $\mathtt{size} \geq 2 \wedge \mathtt{alloc}(\mathsf{x}_1)$. As a corollary of its proof, Proposition 4.1 can be refined so that in the formulae $\mathtt{size} \geq \beta$, we can enforce that $\beta \leq 2\times|\varphi|$ (rough upper bound) where $|\varphi|$ is the size of φ (seen as a tree). Consequently, for any satisfiable formula φ in SL($*, \mathrel{-\!\!*}$), there is a memory state with less than $2 \times |\varphi|$ memory cells that satisfies it. 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 formulae in SL($*, \mathtt{reach}^+$). For example, consider the memory states from Figure 4. The memory states (s_1, h_1) and (s_2, h_2) can be distinguished by the formula

$$\top * (\mathtt{reach}^+(\mathsf{x}_i, \mathsf{x}_j) \wedge \mathtt{reach}^+(\mathsf{x}_j, \mathsf{x}_k) \wedge \neg\mathtt{reach}^+(\mathsf{x}_k, \mathsf{x}_i)).$$

Indeed, (s_1, h_1) satisfies this formula (a witness is obtained by removing the only edge that leads to $s_1(\mathsf{x}_i)$), whereas (s_2, h_2) does not, as every subheap of h_2 that retains the path from $s(\mathsf{x}_i)$ to $s(\mathsf{x}_j)$ and the one from $s(\mathsf{x}_j)$ to $s(\mathsf{x}_k)$ necessarily has a path from $s(\mathsf{x}_k)$ to $s(\mathsf{x}_i)$. This suggests that SL($*, \mathtt{reach}^+$) can express whether, for example, any path from $s(\mathsf{x}_i)$ to $s(\mathsf{x}_j)$ also contains $s(\mathsf{x}_k)$. We will introduce the test formula $\mathtt{sees}_q(\mathsf{x}_i, \mathsf{x}_j) \geq \beta$ to capture this property.

Similarly, the memory states (s_3, h_3) and (s_4, h_4) can be distinguished by the formula

$$(\mathtt{size} = 1) * \big(\mathtt{reach}^+(\mathsf{x}_j, \mathsf{x}_k) \wedge \neg\mathtt{reach}^+(\mathsf{x}_i, \mathsf{x}_k) \wedge \neg\mathtt{reach}^+(\mathsf{x}_k, \mathsf{x}_k)\big).$$

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 $s(\mathsf{x}_k)$ to itself by removing just one location from the heap while retaining the path from $s(\mathsf{x}_j)$ to $s(\mathsf{x}_k)$ and loosing the path from $s(\mathsf{x}_i)$ to $s(\mathsf{x}_k)$. 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*.

Informally, given a memory state (s, h), a meet-point between $s(\mathsf{x})$ and $s(\mathsf{y})$ leading to $s(\mathsf{z})$ is a location ℓ such that ℓ reaches $s(\mathsf{z})$, both locations $s(\mathsf{x})$ and $s(\mathsf{y})$ reach ℓ, and there is no location ℓ' satisfying these properties and reachable from $s(\mathsf{x})$ in strictly less steps. Let us formalise this notion. Let Terms_q be the set $\{\mathsf{x}_1, \ldots, \mathsf{x}_q\} \cup \{m_q(\mathsf{x}_i, \mathsf{x}_j) \mid i, j \in [1, q]\}$ understood as the set of *terms* that are either variables or expressions denoting a meet-point. We write $[\![\mathsf{x}_i]\!]^q_{s,h}$ to denote $s(\mathsf{x}_i)$ and $[\![m_q(\mathsf{x}_i, \mathsf{x}_j)]\!]^q_{s,h}$ to denote (if it exists) the first location reachable from $s(\mathsf{x}_i)$ that is also reachable from $s(\mathsf{x}_j)$. Moreover we require that this location can reach another location corresponding to a program variable. Formally, $[\![m_q(\mathsf{x}_i, \mathsf{x}_j)]\!]^q_{s,h}$ is defined as the unique location ℓ (if it exists) such that

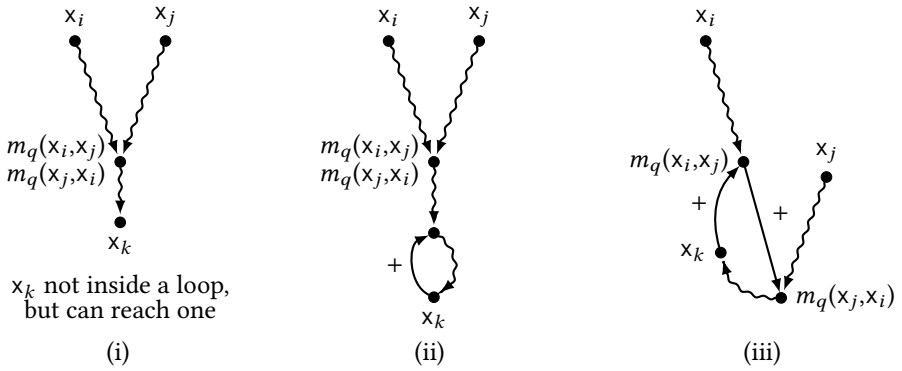


Fig. 5. A taxonomy of meet-points

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

One can easily show that the notion $[\![m_q(\mathsf{x}_i, \mathsf{x}_j)]\!]^q_{s,h}$ is well-defined.

Lemma 4.2. *Let $i, j \in [1, q]$. At most one location satisfies the conditions of $[\![m_q(\mathsf{x}_i, \mathsf{x}_j)]\!]^q_{s,h}$.*

Proof. Ad absurdum, assume that there are two different locations ℓ_1 and ℓ_2 that satisfy the conditions of $[\![m_q(\mathsf{x}_i, \mathsf{x}_j)]\!]^q_{s,h}$. In particular, there are $L_1, L_2, L_3, L_4 \geq 0$ such that

(A) $h^{L_1}(s(\mathsf{x}_i)) = h^{L_2}(s(\mathsf{x}_j)) = \ell_1$ and $h^{L_3}(s(\mathsf{x}_i)) = h^{L_4}(s(\mathsf{x}_j)) = \ell_2$,
(B) for all $L_1' < L_1$ and for all $L_2' \geq 0$, $h^{L_1'}\big(s(\mathsf{x}_i)\big) \neq h^{L_2'}\big(s(\mathsf{x}_j)\big)$.
(C) for all $L_3' < L_3$ and for all $L_4' \geq 0$, $h^{L_3'}\big(s(\mathsf{x}_i)\big) \neq h^{L_4'}\big(s(\mathsf{x}_j)\big)$.

Notice that (A) corresponds to the first condition of $[\![m_q(\mathsf{x}_i, \mathsf{x}_j)]\!]^q_{s,h}$, with respect to ℓ_1 and ℓ_2, whereas (B) and (C) correspond to the second condition for ℓ_1 and ℓ_2, respectively. Let $L_{\min} = \min\{n \mid h^n(s(\mathsf{x}_i)) \in \{\ell_1, \ell_2\}\}$. Let us assume that $h^{L_{\min}}(s(\mathsf{x}_i)) = \ell_1$ and so, as $\ell_1 \neq \ell_2$, for all $L \geq 0$, $h^L(s(\mathsf{x}_i)) = \ell_2$ implies $L_{\min} < L$. From (A) this implies that $L_{\min} < L_1'$ and $h^{L_{\min}}(s(\mathsf{x}_i)) = \mathrm{heap}^{L_2}s(h_j)$, which however contradicts (C). The case where $h^{L_{\min}}(s(\mathsf{x}_i)) = \ell_2$ is analogous, the contradiction being reached with (B). □

The notion of meet-point is quite natural for studying fragments of separation logic with 1s. For instance, a similar notion of a *cut point*, although satisfying different conditions, is considered for the fragments studied in [4]. Figure 5 provides a taxonomy of meet-points, where arrows labelled by '+' represent paths of non-zero length and zig-zag arrows any path (possibly of length zero). Symmetrical cases, obtained by swapping x_i and x_j, are omitted. Following Figure 5, the taxonomy with three distinct shapes depends on the following conditions on $[\![m_q(\mathsf{x}_i, \mathsf{x}_j)]\!]^q_{s,h}$ and $s(\mathsf{x}_k)$:

- $s(\mathsf{x}_k)$ is not inside a loop iff h witnesses the shape (i) of the taxonomy.
- $s(\mathsf{x}_k)$ is inside a loop and $[\![m_q(\mathsf{x}_i, \mathsf{x}_j)]\!]^q_{s,h} = [\![m_q(\mathsf{x}_j, \mathsf{x}_i)]\!]^q_{s,h}$ iff h witnesses the shape (ii).
- $s(\mathsf{x}_k)$ is inside a loop and $[\![m_q(\mathsf{x}_i, \mathsf{x}_j)]\!]^q_{s,h} \neq [\![m_q(\mathsf{x}_j, \mathsf{x}_i)]\!]^q_{s,h}$ iff h witnesses the shape (iii).

Notice that, in the case (i), $s(\mathsf{x}_k)$ can a priori be in $\mathrm{dom}(h)$, which also means that it can reach a loop, even though it is not inside one. The asymmetrical definition of meet-points is captured by the case (iii). Consider the memory states from Figure 4, (s_3, h_3) and (s_4, h_4) can be seen as an instance of this case of the taxonomy and, as such, it holds that $[\![m_q(\mathsf{x}_i, \mathsf{x}_j)]\!]^q_{s_3,h_3} = \ell$ and $[\![m_q(\mathsf{x}_j, \mathsf{x}_i)]\!]^q_{s_3,h_3} = \ell'$.

We identify as special locations the $s(\mathsf{x}_i)$'s and the meet-points of the form $[\![m_q(\mathsf{x}_i, \mathsf{x}_j)]\!]^q_{s,h}$ when it exists ($i, j \in [1, q]$). We call such locations, *labelled* locations and denote by $\mathsf{Lab}_q(s, h)$ the set of labelled locations. The following lemma states an important property of labelled locations: taking subheaps can only reduce their set.

Lemma 4.3. *Let (s, h) be a memory state, $h' \sqsubseteq h$ and $q \geq 1$. We have $\mathsf{Lab}_q(s, h') \subseteq \mathsf{Lab}_q(s, h)$.*

Proof. Let $\ell \in \mathsf{Lab}_q(s, h')$. We want to prove that $\ell \in \mathsf{Lab}_q(s, h)$. The only interesting case is when $\ell \in \{[\![m_q(\mathsf{x}_i, \mathsf{x}_j)]\!]^q_{s,h'} \mid [\![m_q(\mathsf{x}_i, \mathsf{x}_j)]\!]^q_{s,h'}$ is defined, $i, j \in [1, q]\} \setminus \{s(\mathsf{x}_i) \mid i \in [1, q]\}$ as (s, h) and (s, h') share the same store and therefore the result is trivial for locations that correspond to the interpretation of program variables. So, suppose that $\ell = [\![m_q(\mathsf{x}_i, \mathsf{x}_j)]\!]^q_{s,h'}$ for some i, j. We just need to prove that there exist $i^\star, j^\star \in [1, q]$ such that $[\![m_q(\mathsf{x}_{i^\star}, \mathsf{x}_{j^\star})]\!]^q_{s,h} = \ell$.

By definition, $[\![m_q(\mathsf{x}_i, \mathsf{x}_j)]\!]^q_{s,h'} = \ell$ if and only if there exist $k \in [1, q]$ and L such that $h'^L(\ell) = s(\mathsf{x}_k)$ and there are L_1, L_2 such that $h'^{L_1}(s(\mathsf{x}_i)) = h'^{L_2}(s(\mathsf{x}_j)) = \ell$ and for all $L'_1 < L_1$ and L'_2 it holds $h'^{L'_1}(s(\mathsf{x}_i)) \neq h'^{L'_2}(s(\mathsf{x}_j))$. As $h' \sqsubseteq h$, each path of h' is also a path in h. In particular, $h^L(\ell) = s(\mathsf{x}_k)$. As such, we only need to prove that that there exist $i^\star, j^\star \in [1, q]$ such that there are L_1, L_2 such that $h^{L_1}(s(\mathsf{x}_{i^\star})) = h^{L_2}(s(\mathsf{x}_{j^\star})) = \ell$ and for all $L'_1 < L_1$ and L'_2 it holds $h^{L'_1}(s(\mathsf{x}_{i^\star})) \neq h^{L'_2}(s(\mathsf{x}_{j^\star}))$. We consider the taxonomy of $m_q(\mathsf{x}_i, \mathsf{x}_j)$ shown in Figure 5. Based on the classification, $[\![m_q(\mathsf{x}_i, \mathsf{x}_j)]\!]^q_{s,h'} = \ell$ entails that the heap h' witnesses one of the situations among (i)–(iii).

If the heap h' belongs to (ii) or (iii), the locations $s(\mathsf{x}_i)$ and $s(\mathsf{x}_j)$ eventually reach a location $\tilde{\ell}$ belonging to a loop, i.e. $h'^L(\tilde{\ell}) = \tilde{\ell}$ for some $L \geq 1$, and since $h' \sqsubseteq h$, the same holds true in h. Hence, in h, the location ℓ is still the first location reachable from x_i that is also reachable from x_j and therefore $i^\star = i$ and $j^\star = j$.

Alternatively, if h' belongs to (i), the heap h may belong to any of the three situations (i)–(iii) as far as ℓ is concerned. It is easy to see that in h, either $\ell = [\![m_q(\mathsf{x}_i, \mathsf{x}_j)]\!]^q_{s,h}$ or $\ell = [\![m_q(\mathsf{x}_j, \mathsf{x}_i)]\!]^q_{s,h}$, which guarantees that ℓ is a labelled location in h. Let us justify this claim. As (i) holds, either (a) $s(\mathsf{x}_k)$ can reach a location that is not in the domain of the heap h' or (b) $s(\mathsf{x}_k)$ does not reach such a location, that is, there is a cycle after x_k. If (b) holds, then a reasoning analogous to the above cases for (ii) and (iii) works. Otherwise ((a) holds), in h' there is a location ℓ' reachable from $s(\mathsf{x}_k)$ that is not in $\mathrm{dom}(h')$. We now consider how h extends h'.

- If h belongs to (i), then $\ell = [\![m_q(\mathsf{x}_i, \mathsf{x}_j)]\!]^q_{s,h} = [\![m_q(\mathsf{x}_i, \mathsf{x}_j)]\!]^q_{s,h'}$. The same holds true if h extends h' by introducing a loop so that h belongs to (ii).
- If h extends h' by adding a path from ℓ' to a location in the path between $s(\mathsf{x}_j)$ to ℓ (ℓ excluded), then $\ell = [\![m_q(\mathsf{x}_i, \mathsf{x}_j)]\!]^q_{s,h} = [\![m_q(\mathsf{x}_i, \mathsf{x}_j)]\!]^q_{s,h'}$ and h belongs to (iii).
- Lastly, if h extends h' by adding a path from ℓ' to a location in the path between $s(\mathsf{x}_i)$ to ℓ (ℓ excluded), then h belongs also to (iii). Unlike the previous case however, we have $\ell = [\![m_q(\mathsf{x}_j, \mathsf{x}_i)]\!]^q_{s,h} = [\![m_q(\mathsf{x}_i, \mathsf{x}_j)]\!]^q_{s,h'}$, so ℓ is again in $\mathsf{Lab}_q(s, h)$. □

As shown in Proposition 4.1, the test formulae explicit what are the essential features that are expressible in SL($*, -\!\!*$). For SL($*, \mathsf{reach}^+$), these test formulae primarily speak about relationships between labelled locations and specific shapes of the heap. In order to highlight these relationships, below we introduce a class of abstract structures, called *support graphs*. The semantics for test formulae shall be provided directly on such support graphs.

Definition 4.4. Let $q \geq 1$ and (s, h) be a memory state. Its *support graph* $\mathrm{SG}_q(s, h)$ is defined as the tuple (V, E, Alloc, TEq, Inter, Rem) such that:

(V) $\mathrm{V} \stackrel{\text{def}}{=} \mathsf{Lab}_q(s, h)$ is the set of labelled locations.

(E) E is the (functional) binary relation on V such that $(\ell, \ell') \in \mathrm{E} \overset{\text{def}}{\Leftrightarrow}$ there is $L \geq 1$ such that $h^L(\ell) = \ell'$ and for all $0 < L' < L$, it holds that $h^{L'}(\ell) \notin \mathrm{V}$. Informally, $\mathrm{E}(\ell, \ell')$ if and only if there is a non-empty path from ℓ to ℓ' whose intermediate locations are not labelled.

(A) $\mathrm{Alloc} \overset{\text{def}}{=} \mathrm{V} \cap \mathrm{dom}(h)$.

(T) $\mathrm{TEq} : \mathrm{V} \to \mathcal{P}(\mathrm{Terms}_q)$ is a function such that for any labelled location $\ell \in \mathrm{V}$, is the (non-empty) set of terms corresponding to ℓ. Formally, $\mathrm{TEq}(\ell) \overset{\text{def}}{=} \{v \in \mathrm{Terms}_q \mid [\![v]\!]^q_{s,h} = \ell\}$.

(I) $\mathrm{Inter} : \mathrm{E} \to \mathcal{P}(\mathrm{LOC})$ is a function such that for any $(\ell, \ell') \in \mathrm{E}$, is the set of intermediate locations of the shortest path beginning in ℓ and ending with ℓ'. Formally,

$$\mathrm{Inter}(\ell, \ell') \overset{\text{def}}{=} \left\{ \ell'' \in \mathrm{LOC} \;\middle|\; \begin{array}{r} \text{there are } L, L' \geq 1, \text{ such that } h^L(\ell) = \ell'' \text{ and } h^{L'}(\ell'') = \ell' \\ \text{and for all } L'' \in [1, L],\ h^{L''}(\ell) \notin V \end{array} \right\}$$

Despite the definition of E, the condition "for all $L'' \in [1, L]$, $h^{L''}(\ell) \notin V$" in this definition is needed in order to handle the case when ℓ and ℓ' belongs to the same cycle. Moreover, notice that every location $\ell'' \in \mathrm{Inter}(\ell, \ell')$ is such that $\ell'' \notin V$ (hence, ℓ'' is not a labelled location).

(R) $\mathrm{Rem} \overset{\text{def}}{=} \mathrm{dom}(h) \setminus (\mathrm{Alloc} \cup \bigcup_{(\ell,\ell') \in \mathrm{E}} \mathrm{Inter}(\ell, \ell'))$, i.e. the set of locations in the domain that are not labelled locations and that do not belong to paths between labelled locations.

One can think of the support graph $\mathrm{SG}_q(s, h)$ as a selection of properties from (s, h). Indeed, from its definition, it is straightforward to see that $\{\mathrm{Alloc}, \mathrm{Rem}\} \cup \{\mathrm{Inter}(\ell, \ell') \mid (\ell, \ell') \in \mathrm{E}\}$ is a partition of $\mathrm{dom}(h)$. Moreover, for all $i, j \in [1, q]$ and $\ell \in \mathrm{LOC}$, $\ell = s(\mathtt{x}_i) = s(\mathtt{x}_j)$ iff $\{\mathtt{x}_i, \mathtt{x}_j\} \subseteq \mathrm{TEq}(\ell)$.

We are now ready to introduce the test formulae. Given $q, \alpha \geq 1$, we write $\mathrm{Test}(q, \alpha)$ to denote the following set of atomic *test formulae*:

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

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 (see Theorem 4.15). Let (s, h) be a memory state. Below, we present the formal semantics for the test formulae, provided with the elements from the support graph $\mathrm{SG}_q(s, h) = (\mathrm{V}, \mathrm{E}, \mathrm{Alloc}, \mathrm{TEq}, \mathrm{Inter}, \mathrm{Rem})$ of (s, h):

- $(s, h) \models v = v' \overset{\text{def}}{\Leftrightarrow}$ there is $\ell \in \mathrm{V}$ such that $\{v, v'\} \subseteq \mathrm{TEq}(\ell)$.
- $(s, h) \models \mathtt{alloc}(v) \overset{\text{def}}{\Leftrightarrow}$ there is $\ell \in \mathrm{Alloc}$ such that $v \in \mathrm{TEq}(\ell)$.
- $(s, h) \models v \hookrightarrow v' \overset{\text{def}}{\Leftrightarrow} v \in \mathrm{TEq}(\ell)$, $v' \in \mathrm{TEq}(\ell')$, $\mathrm{card}(\mathrm{Inter}(\ell, \ell')) = 0$ for some $(\ell, \ell') \in \mathrm{E}$.
- $(s, h) \models \mathtt{sees}_q(v, v') \geq \beta + 1 \overset{\text{def}}{\Leftrightarrow}$ there is $(\ell, \ell') \in \mathrm{E}$ such that $v \in \mathrm{TEq}(\ell)$, $v' \in \mathrm{TEq}(\ell')$ and $\mathrm{card}(\mathrm{Inter}(\ell, \ell')) \geq \beta$.
- $(s, h) \models \mathtt{sizeR}_q \geq \beta \overset{\text{def}}{\Leftrightarrow} \mathrm{card}(\mathrm{Rem}) \geq \beta$.

The semantics is deliberately defined with the help of support graphs but of course, the clauses could be formulated with s and h only. For instance, the formula $\mathtt{sees}_q(m_q(\mathtt{x}_i, \mathtt{x}_j), \mathtt{x}_k) \geq 4$ is satisfied whenever $[\![m_q(\mathtt{x}_i, \mathtt{x}_j)]\!]^q_{s,h}$ is defined and there is a path of length at least 4 beginning in this location, ending in $s(\mathtt{x}_k)$ and whose intermediate locations are not labelled. Similarly, $\mathtt{sizeR}_q \geq \beta$ holds true whenever the number of allocated locations that are neither between two locations interpreted by terms nor the interpretation of a term, is at least β. Notice that there is no need for test formulae of the form $\mathtt{sees}_q(v, v') \geq 1$ since it is equivalent to $v \hookrightarrow v' \vee \mathtt{sees}_q(v, v') \geq 2$. In the sequel, occurrences of $\mathtt{sees}_q(v, v') \geq 1$ are understood as abbreviations. One can check whether $[\![m_q(\mathtt{x}_i, \mathtt{x}_j)]\!]^q_{s,h}$ is defined thanks to the satisfaction of the formula $m_q(\mathtt{x}_i, \mathtt{x}_j) = m_q(\mathtt{x}_i, \mathtt{x}_j)$. 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 atomic formulae of the form $T \xrightarrow{h \setminus T''} T'$ in the logic GRASS [34]. Indeed,

by way of example, $\mathtt{x}_1 \xrightarrow{h\setminus \mathtt{x}_3} \mathtt{x}_2$ states that there is a path in the graph of the heap h between $s(\mathtt{x}_1)$ and $s(\mathtt{x}_2)$ without passing via $s(\mathtt{x}_3)$. The satisfaction of $\mathtt{sees}_q(v, v') \geq \beta + 1$ requires to exclude labelled locations strictly between the interpretation of the terms v and v'. Our test formulae are quite expressive since they capture the atomic formulae from SL($*$, $\mathtt{reach}^+$) and the test formulae for SL($*$, $-\!\!*$) (introduced in Proposition 4.1).

Lemma 4.5. *Let $\alpha, q \geq 1$, $i, j \in [1, q]$. Any atomic formula among $\mathtt{reach}^+(\mathtt{x}_i, \mathtt{x}_j)$, $\mathtt{emp}$ and $\mathtt{size} \geq \beta$ (with $\beta \leq \alpha$), is equivalent to a Boolean combination of test formulae from* Test(q, α).

Proof. Let $q, \alpha \geq 1$ and consider the family of test formulae Test(q, α). Let (s, h) be a memory state and $\mathrm{SG}_q(s, h) = (\mathrm{V}, \mathrm{E}, \mathrm{Alloc}, \mathrm{TEq}, \mathrm{Inter}, \mathrm{Rem})$ be its support graph.

- We show that $\mathtt{emp}$ is equivalent to the Boolean combination of test formulae $\neg\mathtt{sizeR}_q \geq 1 \wedge \bigwedge_{i\in[1,q]} \neg\mathtt{alloc}(\mathtt{x}_i)$. Suppose $(s, h) \models \mathtt{emp}$. Then trivially, for all $i \in [1, q]$ $(s, h) \models \neg\mathtt{alloc}(\mathtt{x}_i)$ and $(s, h) \models \neg\mathtt{sizeR}_q \geq 1$.
  Conversely, suppose $(s, h) \models \neg\mathtt{sizeR}_q \geq 1 \wedge \bigwedge_{i\in[1,q]} \neg\mathtt{alloc}(\mathtt{x}_i)$. As for all $i \in [1, q]$ $\mathtt{x}_i$ does not correspond to a location in $\mathrm{dom}(h)$, it holds that $(s, h) \models \bigwedge_{v,v'\in\mathrm{Terms}_q} \neg\mathtt{sees}_q(v, v') \geq 1$. Therefore, directly from its definition, $\mathrm{Rem} = \mathrm{dom}(h)$. The emptiness of $\mathrm{dom}(h)$ then follows from $(s, h) \models \neg\mathtt{sizeR}_q \geq 1$.
- The formula $\mathtt{reach}^+(\mathtt{x}_i, \mathtt{x}_j)$ can be shown equivalent to

$$\varphi \stackrel{\text{def}}{=} \bigvee_{\substack{v_1,\ldots,v_n\in\mathrm{Terms}_q,\\ \text{pairwise distinct } v_1,\ldots,v_{n-1},\\ \mathtt{x}_i=v_1,\mathtt{x}_j=v_n}} \bigwedge_{\delta\in[1,n-1]} \mathtt{sees}_q(v_\delta, v_{\delta+1}) \geq 1.$$

  In the expression above, n is arbitrary in the disjunction as soon as the other constraints are satisfied (whence, $n \leq \mathrm{card}(\mathrm{Terms}_q)$). First, suppose $(s, h) \models \mathtt{reach}^+(\mathtt{x}_i, \mathtt{x}_j)$. Then, there exists $L \geq 1$ such that $h^L(s(\mathtt{x}_i)) = s(\mathtt{x}_j)$. Let $\ell_1 = s(\mathtt{x}_i)$, $\ell_n = s(\mathtt{x}_j)$ and let $\ell_2, \ldots, \ell_{n-1} \in \mathtt{Lab}_q(s, h)$ be all labelled locations in the (minimal) path witnessing $(s, h) \models \mathtt{reach}^+(\mathtt{x}_i, \mathtt{x}_j)$, such that $h^{L_\delta}(\ell_\delta) = \ell_{\delta+1}$, where $L_\delta \geq 1$, and there are no labelled locations between ℓ_δ and $\ell_{\delta+1}$, $1 \leq \delta \leq n - 1 \leq \mathrm{card}(\mathtt{Lab}_q(s, h))$. The path from $s(\mathtt{x}_i)$ to $s(\mathtt{x}_j)$ is therefore uniquely split into $n - 1$ subpaths starting and ending with a labelled location and without labelled locations in between. Let $v_1 = \mathtt{x}_i$, $v_n = \mathtt{x}_j$ and $v_2, \ldots, v_{n-1} \in \mathrm{Terms}_q$ be such that $[\![v_\delta]\!]^q_{s,h} = \ell_\delta$ for $\delta \in [2, n - 1]$. From the definition of $\ell_2, \ldots, \ell_{n-1}$, we conclude that (s, h) satisfies $\bigwedge_{1\leq\delta\leq n-1} \mathtt{sees}_q(v_\delta, v_{\delta+1}) \geq 1$. The minimality of the path is needed to guarantee that all the v_i's are distinct.
  Conversely, suppose $(s, h) \models \varphi$. Then, there are $v_1, \ldots, v_n \in \mathrm{Terms}_q$ such that $v_1 = \mathtt{x}_i$, $v_n = \mathtt{x}_j$, $v_1, \ldots, v_{n-1}$ are pairwise distinct and for all $\delta \in [1, n - 1]$ $(s, h) \models \mathtt{sees}_q(v_\delta, v_{\delta+1}) \geq 1$. From the semantics of $\mathtt{sees}$, this implies that there are $L_1, \ldots, L_{n-1} \geq 1$ such that $h^{L_\delta}([\![v_\delta]\!]^q_{s,h}) = [\![v_{\delta+1}]\!]^q_{s,h}$ for all $\delta \in [1, n - 1]$. Therefore $h(s(\mathtt{x}_i))^{\sum_\delta L_\delta} = s(\mathtt{x}_j)$, where $\sum_\delta L_\delta \geq 1$ and $\mathtt{reach}^+(\mathtt{x}_i, \mathtt{x}_j)$ is satisfied.
- We define $\mathtt{size} \geq \beta$, where $\beta \leq \alpha$. First of all, we introduce the formula $\mathtt{sizeV}_q(v) \geq \beta$, with $v \in \mathrm{Terms}_q$, below:

$$\mathtt{sizeV}_q(v) \geq 0 \stackrel{\text{def}}{=} \top \qquad\qquad \mathtt{sizeV}_q(v) \geq 1 \stackrel{\text{def}}{=} \mathtt{alloc}(v)$$

$$\mathtt{sizeV}_q(v) \geq \beta + 1 \stackrel{\text{def}}{=} \bigvee_{v'\in\mathrm{Terms}_q} \mathtt{sees}_q(v, v') \geq \beta + 1 \qquad \text{for } \beta \in [1, \alpha]$$

  $\mathtt{sizeV}_q(v) \geq 0$ is always true, $\mathtt{sizeV}_q(v) \geq 1$ holds in a memory state (s, h) if and only if $(s, h) \models \mathtt{alloc}(v)$, whereas $\mathtt{sizeV}_q(v) \geq \beta + 1$ holds if and only if there exists $v' \in \mathrm{Terms}_q$

such that $(s,h) \models \mathtt{sees}_q(v,v') \geq \beta+1$. As such, $\mathtt{sizeV}_q(v) \geq \beta+1$ holds whenever for all $L \in [0,\beta]$, $h^L([\![v]\!]^q_{s,h}) \in \mathrm{dom}(h)$, $L \geq 1$ implies $h^L([\![v]\!]^q_{s,h}) \notin \mathtt{Lab}_q(s,h)$, and there is $L' > \beta$ such that $h^{L'}([\![v]\!]^q_{s,h}) = [\![v']\!]^q_{s,h}$, for some $v' \in \mathrm{Terms}_q$. From the definition of sees and meet-points, it follows that the locations considered for the satisfaction of $\mathtt{sizeV}_q(v) \geq \beta$ do not play any role in the satisfaction of $\mathtt{sizeV}_q(v') \geq \beta'$, where $[\![v]\!]^q_{s,h} \neq [\![v']\!]^q_{s,h}$. Therefore, it is easy to prove that if $(s,h) \models \mathtt{sizeV}_q(v) \geq \beta \wedge \mathtt{sizeV}_q(v') \geq \beta' \wedge v \neq v'$ then $\mathrm{card}(\mathrm{dom}(h)) \geq \beta+\beta'$. Similarly, the locations considered for the satisfaction of this formula are not in Rem and therefore $(s,h) \models \mathtt{sizeV}_q(v) \geq \beta \wedge \mathtt{sizeR}_q \geq \beta'$ implies $\mathrm{card}(\mathrm{dom}(h)) \geq \beta+\beta'$. We can then use this formula to define $\mathtt{size} \geq \beta$ as follows, where we write $\mathtt{sizeR}_q \geq 0$ instead of $\mathtt{x}_1 = \mathtt{x}_1$ (any tautological Boolean combination of test formulae would be fine).

$$\bigvee_{\substack{V \subseteq \mathrm{Terms}_q \\ \beta \leq \beta_R + \sum_{v\in V}\beta_v \\ \beta_R \in [0,\alpha]\ \forall v \in V \beta_v \in [0,\alpha+1]}} \Big(\mathtt{sizeR}_q \geq \beta_R \wedge \bigwedge_{v\in V} (\mathtt{sizeV}_q(v) \geq \beta_v \wedge \bigwedge_{v'\in V\setminus\{v\}} v \neq v') \Big)$$

Suppose that this formula is satisfied by (s,h). Then there exists a subset of terms V such that for all $v,v' \in V$, if $v \neq v'$ then $[\![v]\!]^q_{s,h} \neq [\![v']\!]^q_{s,h}$ also follows (from the last conjunct of the formula). From the property just stated about $\mathtt{sizeV}_q(v) \geq \beta$, it must therefore hold that $\mathrm{card}(\mathrm{dom}(h)) \geq \beta_R + \sum_{v\in V}\beta_v \geq \beta$.
Conversely, suppose $\mathrm{card}(\mathrm{dom}(h)) \geq \beta$. We can define a partition $h_R + \sum_{\ell \in \mathtt{Lab}_q(s,h)} h_\ell = h$ such that each subheap h_ℓ of the partition contains exactly the locations of $\mathrm{dom}(h)$ considered for the satisfaction of the test formulae $\mathtt{sizeV}_q(v) \geq \beta'$, $\beta' \in [0,\alpha+1]$, for a specific labelled location $\ell = [\![v]\!]^q_{s,h}$, whereas h_R contains all the locations considered for the satisfaction of $\mathtt{sizeR}_q \geq \beta'$, $\beta' \in [0,\alpha]$. Consider a representative $v \in \mathrm{Terms}_q$ for each subheap h_ℓ, where $[\![v]\!]^q_{s,h} = \ell$. Let V be the set of these representatives and let $\beta_R = \min(\alpha, \mathrm{card}(\mathrm{dom}(h_R)))$, $\beta_v = \min(\alpha+1, \mathrm{card}(\mathrm{dom}(h_{[\![v]\!]^q_{s,h}})))$. Since $\beta \leq \alpha$ and $\mathrm{card}(\mathrm{dom}(h)) = \mathrm{card}(\mathrm{dom}(h_R)) + \sum_{v\in V} \geq \beta$, it follows that $\beta_R + \sum_{v\in V}\beta_v \geq \beta$. Moreover, for each $v \in V$, $\beta_v \in [0,\alpha+1]$, whereas $\beta_R \in [0,\alpha]$. From the definition of V, it immediately holds that $(s,h) \models \bigwedge_{v\neq v'\in V} v \neq v'$. Lastly, from their definition, it holds that $(s,h) \models \mathtt{sizeR}_q \geq \beta_R$ and for all $v \in V$ $(s,h) \models \mathtt{sizeV}_q(v) \geq \beta_v$. We conclude that the formula defining $\mathtt{size} \geq \beta$ is satisfied. □

4.2 Expressive power and the small heap property

Now, we show that 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 4.10) and deduce the small heap property of this logic (Theorem 4.11). We introduce an indistinguishability relation $\approx^q_\alpha$ between memory states based on test formulae, see analogous relations in [14, 17, 28].

Definition 4.6. Let $q, \alpha \geq 1$ and, (s,h) and (s',h') be memory states. $(s,h) \approx^q_\alpha (s',h') \overset{\text{def}}{\Leftrightarrow} (s,h) \models \psi$ iff $(s',h') \models \psi$, for all $\psi \in \mathrm{Test}(q,\alpha)$.

Forthcoming Theorem 4.9 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). The following technical lemma lifts the relationship $\approx^q_\alpha$ to an equivalence between support graphs, consolidating this idea of indistinguishable memory states.

Lemma 4.7. *Let $q, \alpha \geq 1$, and (s,h), (s',h') be two memory states with support graphs respectively* $\mathrm{SG}_q(s,h) = (\mathrm{V}, \mathrm{E}, \mathrm{Alloc}, \mathrm{TEq}, \mathrm{Inter}, \mathrm{Rem})$ *and* $\mathrm{SG}_q(s',h') = (\mathrm{V}', \mathrm{E}', \mathrm{Alloc}', \mathrm{TEq}', \mathrm{Inter}', \mathrm{Rem}')$. *We have $(s,h) \approx^q_\alpha (s',h')$ iff there is a map $\dagger : \mathrm{V} \to \mathrm{V}'$ such that*

(A1) $\mathfrak{f}$ *is a graph isomorphism between* (V, E) *and* $(\mathrm{V}', \mathrm{E}')$*;*
(A2) *for all* $\ell \in \mathrm{V}$*, we have* $\ell \in \mathrm{Alloc}$ *iff* $\mathfrak{f}(\ell) \in \mathrm{Alloc}'$*;*
(A3) *for all* $\ell \in \mathrm{V}$*, we have* $\mathrm{TEq}(\ell) = \mathrm{TEq}'(\mathfrak{f}(\ell))$*;*
(A4) *for all* $(\ell, \ell') \in \mathrm{E}$*, we have* $\min(\alpha, \mathrm{card}(\mathrm{Inter}(\ell, \ell'))) = \min(\alpha, \mathrm{card}(\mathrm{Inter}'(\mathfrak{f}(\ell), \mathfrak{f}(\ell'))))$*;*
(A5) $\min(\alpha, \mathrm{card}(\mathrm{Rem})) = \min(\alpha, \mathrm{card}(\mathrm{Rem}'))$.

Proof. Suppose $(s, h) \approx^q_\alpha (s', h')$. Let $\mathfrak{f} : \mathrm{V} \to \mathrm{V}'$ be the map such that for all locations $\ell \in \mathrm{V}$, we have $\mathfrak{f}(\ell) \stackrel{\text{def}}{=} \ell'$ if and only if there is $v \in \mathrm{Terms}_q$ such that $[\![v]\!]^q_{s,h} = \ell$ and $[\![v]\!]^q_{s',h'} = \ell'$. Let us show that $\mathfrak{f}$ is well-defined. To do so, assume that there are v, v' such that $[\![v]\!]^q_{s,h} = [\![v']\!]^q_{s,h} = \ell$, $[\![v]\!]^q_{s',h'} = \ell'$ and $[\![v']\!]^q_{s',h'} = \ell''$. Since $(s, h) \approx^q_\alpha (s', h')$, we have that $(s, h) \models v = v'$ iff $(s', h') \models v = v'$. Therefore, ($[\![v]\!]^q_{s,h}$, $[\![v']\!]^q_{s,h}$ are defined and $[\![v]\!]^q_{s,h} = [\![v']\!]^q_{s,h}$) iff ($[\![v]\!]^q_{s',h'}$, $[\![v']\!]^q_{s',h'}$ are defined and $[\![v]\!]^q_{s',h'} = [\![v']\!]^q_{s',h'}$). Consequently, $\ell' = \ell''$ and $\mathfrak{f}$ is well-defined. Actually, the equivalence above induced by $(s, h) \approx^q_\alpha (s', h')$ allows us to show in a similar way that $\mathfrak{f}$ is a bijection from V to V′ and that the condition **(A3)** holds true. Indeed, the statements below are equivalent:

- $\{v, v'\} \subseteq \mathrm{TEq}(\ell)$,
- $(s, h) \models v = v'$ and $[\![v]\!]^q_{s,h} = [\![v']\!]^q_{s,h} = \ell$ (by definition of $\models$ and $\mathrm{SG}_q(s, h)$),
- $(s', h') \models v = v'$ and $[\![v]\!]^q_{s',h'} = [\![v']\!]^q_{s',h'} = \ell'$ for some ℓ' (by $(s, h) \approx^q_\alpha (s', h')$),
- $(s', h') \models v = v'$ and $[\![v]\!]^q_{s',h'} = [\![v']\!]^q_{s',h'} = \mathfrak{f}(\ell)$ (by definition of $\mathfrak{f}$),
- $\{v, v'\} \subseteq \mathrm{TEq}'(\mathfrak{f}(\ell))$ (by definition of $\mathrm{SG}_q(s', h')$).

Consequently, the condition **(A3)** holds true. A similar reasoning allows us to establish **(A2)** and it is omitted below. In order to conclude the first part of the proof, first we show **(A5)** and then we focus on **(A1)** and **(A4)**. Let us first establish **(A5)**. As $(s, h) \approx^q_\alpha (s', h')$, we have (†) for all $\beta \in [1, \alpha]$, $(s, h) \models \mathtt{sizeR}_q \geq \beta$ iff $(s', h') \models \mathtt{sizeR}_q \geq \beta$ and (†) is equivalent to the statements below:

- for all $\beta \in [1, \alpha]$, $\mathrm{card}(\mathrm{Rem}) \geq \beta$ iff $\mathrm{card}(\mathrm{Rem}') \geq \beta$ (by definition of $\models$),
- $\min(\alpha, \mathrm{card}(\mathrm{Rem})) = \min(\alpha, \mathrm{card}(\mathrm{Rem}'))$ (by a simple arithmetical reasoning).

Consequently, the condition **(A5)** holds true. Now, let us show **(A1)** and **(A4)**. First, the statements below are equivalent:

- $(\ell, \ell') \in \mathrm{E}$ and $\mathrm{Inter}(\ell, \ell') = \emptyset$,
- $(s, h) \models v \hookrightarrow v'$, $[\![v]\!]^q_{s,h} = \ell$ and $[\![v']\!]^q_{s,h} = \ell'$ for some $v, v' \in \mathrm{Terms}_q$ (by definition of $\models$),
- $(s', h') \models v \hookrightarrow v'$, $[\![v]\!]^q_{s',h'} = \ell''$ and $[\![v']\!]^q_{s',h'} = \ell'''$ for some $v, v' \in \mathrm{Terms}_q$, for some locations ℓ'', ℓ''', (by $(s, h) \approx^q_\alpha (s', h')$),
- $(s', h') \models v \hookrightarrow v'$, $[\![v]\!]^q_{s',h'} = \mathfrak{f}(\ell)$ and $[\![v']\!]^q_{s',h'} = \mathfrak{f}(\ell')$ for some $v, v' \in \mathrm{Terms}_q$ (by definition of $\mathfrak{f}$),
- $(\mathfrak{f}(\ell), \mathfrak{f}(\ell')) \in \mathrm{E}'$ and $\mathrm{Inter}'(\mathfrak{f}(\ell), \mathfrak{f}(\ell')) = \emptyset$ (by definition of $\models$ and $\mathrm{SG}_q(s', h')$).

Similarly, we have the following equivalences, where $\beta \in [1, \alpha]$:

- $(\ell, \ell') \in \mathrm{E}$ and $\mathrm{card}(\mathrm{Inter}(\ell, \ell')) \geq \beta$,
- $(s, h) \models \mathtt{sees}_q(v, v') \geq \beta + 1$, $[\![v]\!]^q_{s,h} = \ell$, $[\![v']\!]^q_{s,h} = \ell'$ for some $v, v' \in \mathrm{Terms}_q$ (by definition of $\models$),
- $(s', h') \models \mathtt{sees}_q(v, v') \geq \beta + 1$, $[\![v]\!]^q_{s',h'} = \ell''$ and $[\![v']\!]^q_{s',h'} = \ell'''$ for some $v, v' \in \mathrm{Terms}_q$, for some locations ℓ'', ℓ''', (by $(s, h) \approx^q_\alpha (s', h')$),
- $(s', h') \models \mathtt{sees}_q(v, v') \geq \beta + 1$, $[\![v]\!]^q_{s',h'} = \mathfrak{f}(\ell)$, $[\![v']\!]^q_{s',h'} = \mathfrak{f}(\ell')$ for some $v, v' \in \mathrm{Terms}_q$, (by $\mathfrak{f}$),
- $(\mathfrak{f}(\ell), \mathfrak{f}(\ell')) \in \mathrm{E}'$ and $\mathrm{card}(\mathrm{Inter}'(\mathfrak{f}(\ell), \mathfrak{f}(\ell'))) \geq \beta$ (by definition of $\models$ and $\mathrm{SG}_q(s', h')$).

Consequently, we get **(A1)** and **(A4)**.

We omit below the proof of the other direction as it is similar to the first direction. □

Now, 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 $\approx^q_\alpha$. Separating conjunction can therefore be eliminated from the logic in favour of test formulae, which is essential in the proof of Theorem 4.9.

Lemma 4.8. *Let $q, \alpha, \alpha_1, \alpha_2 \geq 1$ with $\alpha = \alpha_1 + \alpha_2$ and (s,h), (s',h') be memory states 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 $(s,h_2) \approx^q_{\alpha_2} (s',h'_2)$.*

The proof of Lemma 4.8 is rather long as it first constructs the subheaps h'_1 and h'_2, and then check that $(s,h_i) \approx^q_{\alpha_i} (s',h'_i)$ holds (for both $i \in \{1,2\}$) by verifying the conditions **(A1)**–**(A5)**. Moreover, the verification of some of the conditions requires first to establish additional preliminary properties. Though the principle of the proof structure is quite simple, checking carefully each property is quite lengthy. We believe that having all the material in a single proof is helpful to emphasize the proof structure, or alternatively to skip the proof in a first reading of the document (see also Appendix A).

Proof. Let q, α, α_1, α_2, (s,h), (s',h'), h_1 and h_2 be defined as in the statement. Let

- $\mathrm{SG}_q(s,h) = (\mathrm{V}, \mathrm{E}, \mathrm{Alloc}, \mathrm{TEq}, \mathrm{Inter}, \mathrm{Rem})$ and
- $\mathrm{SG}_q(s',h') = (\mathrm{V}', \mathrm{E}', \mathrm{Alloc}', \mathrm{TEq}', \mathrm{Inter}', \mathrm{Rem}')$

be the support graphs of (s,h) and (s',h') respectively, with respect to q. As $(s,h) \approx^q_\alpha (s',h')$, let $\mathfrak{f} : \mathrm{V} \to \mathrm{V}'$ be a map satisfying **(A1)**–**(A5)** from Lemma 4.7. Below, for the sake of conciseness, let $k \in \{1,2\}$ and let $\mathrm{SG}_q(s,h_k) = (\mathrm{V}_k, \mathrm{E}_k, \mathrm{Alloc}_k, \mathrm{TEq}_k, \mathrm{Inter}_k, \mathrm{Rem}_k)$ be the support graph of (s,h_k). The proof is rather long and can be summed up with the following steps.

(1) First, we define a strategy to split h' into h'_1 and h'_2 by closely following the way that h is split into h_1 and h_2. To do this, we look at the support graphs. For instance, suppose that Rem is split into two sets $R_1 \subseteq \mathrm{dom}(h_1)$ and $R_2 \subseteq \mathrm{dom}(h_2)$. By definition, it is quite easy to see that the sets R_1 and R_2 must be subsets of Rem_1 and Rem_2, respectively. Then, following Lemma 4.7, to obtain $(s,h_k) \approx^q_{\alpha_k} (s',h'_k)$, we are required to split Rem′ into $R'_1 \subseteq \mathrm{dom}(h'_1)$ and $R'_2 \subseteq \mathrm{dom}(h'_2)$ so that $\min(\alpha_k, \mathrm{card}(R_k)) = \min(\alpha_k, \mathrm{card}(R'_k))$. Indeed, otherwise the equisatisfaction of the test formulae of the form $\mathtt{sizeR}_q \geq \beta$ is not ensured (details on this are formalised later).

(2) After defining h'_1 and h'_2, we show that $(s,h_k) \approx^q_{\alpha_k} (s',h'_k)$. To do so, again, we follow Lemma 4.7 and we show that we can find suitable bijections from labelled locations of h_k to the ones of h'_k satisfying **(A1)**–**(A5)**.

According to the summary above, let us first define explicitly h'_1 and h'_2 via an iterative process that consists in adding locations to $\mathrm{dom}(h'_1)$ or to $\mathrm{dom}(h'_2)$. Whenever we enforce that $\ell \in \mathrm{dom}(h'_k)$, implicitly we have $h'_k(\ell) \stackrel{\text{def}}{=} h'(\ell)$ as h'_k is intended to be a subheap of h'.

(CA) For all $\ell \in \mathrm{Alloc}'$, $\ell \in \mathrm{dom}(h'_k) \stackrel{\text{def}}{\Leftrightarrow} \mathfrak{f}^{-1}(\ell) \in \mathrm{dom}(h_k)$. This step of the construction, as well as its usefulness, should be self-explanatory. For example, if $(s,h_k) \models \mathtt{alloc}(\mathtt{x}_i)$ then, by relying on **(A3)**, this step allows us to conclude that $(s',h'_k) \models \mathtt{alloc}(\mathtt{x}_i)$ (independently on how the definition of h'_k will be completed in the next steps of the construction).

(CR) The heaps h'_1 and h'_2 are further populated depending on Rem. Let $R_k = \mathrm{Rem} \cap \mathrm{dom}(h_k)$. By definition, we have $R_1 \uplus R_2 = \mathrm{Rem}$. Below, we partition Rem′ into two sets R'_1 and R'_2 so that by definition $R'_k \subseteq \mathrm{dom}(h'_k)$. The strategy for defining the partition is split into three cases:

(CR.C1) If $\mathrm{card}(R_1) < \alpha_1$ then R'_1 is a set of $\mathrm{card}(R_1)$ locations from Rem′ and $R'_2 \stackrel{\text{def}}{=} \mathrm{Rem}' \setminus R'_1$.

(CR.C2) Otherwise, if $\mathrm{card}(R_2) < \alpha_2$ then R'_2 is a set of $\mathrm{card}(R_2)$ locations of Rem' and $R'_1 \stackrel{\text{def}}{=} \mathrm{Rem}' \setminus R'_2$.

(CR.C3) Otherwise we have $\mathrm{card}(R_1) \geq \alpha_1$ and $\mathrm{card}(R_2) \geq \alpha_2$. Then, R'_1 is a set of α_1 locations from Rem' and $R'_2 \stackrel{\text{def}}{=} \mathrm{Rem}' \setminus R'_1$.

It is easy to show that the construction satisfies the following property (where $k \in \{1, 2\}$):

$$\min(\alpha_k, \mathrm{card}(R_k)) = \min(\alpha_k, \mathrm{card}(R'_k)) \qquad \textbf{(CR.P1)}$$

The property **(CR.P1)** directly follows from the property **(A5)** satisfied by $\mathfrak{f}$. The proof (that can be applied also for the next step of the construction, see **(CI.P2)**), works as follows.
First, suppose that the sets of remaining locations in the heap domain are small, i.e.

$$\min(\alpha, \mathrm{card}(\mathrm{Rem})) = \min(\alpha, \mathrm{card}(\mathrm{Rem}')) < \alpha_1 + \alpha_2.$$

So, $\mathrm{card}(\mathrm{Rem}) = \mathrm{card}(\mathrm{Rem}')$ and therefore $\mathrm{card}(R_1) + \mathrm{card}(R_2) = \mathrm{card}(R'_1) + \mathrm{card}(R'_2)$. By definition, $\mathrm{card}(R_1) = \mathrm{card}(R'_1)$ and $\mathrm{card}(R_2) = \mathrm{card}(R'_2)$ trivially hold for the cases **(CR.C1)** and **(CR.C2)**, whereas the case **(CR.C3)** ($\mathrm{card}(R_1) \geq \alpha_1$ and $\mathrm{card}(R_2) \geq \alpha_2$) can never be applied since $\mathrm{card}(R_1) + \mathrm{card}(R_2) < \alpha_1 + \alpha_2$. We conclude that $\min(\alpha_k, \mathrm{card}(R_k)) = \min(\alpha_k, \mathrm{card}(R'_k))$.
Second, suppose instead

$$\min(\alpha, \mathrm{card}(\mathrm{Rem})) = \min(\alpha, \mathrm{card}(\mathrm{Rem}')) = \alpha_1 + \alpha_2.$$

If the first case **(CR.C1)** applies, i.e. $\mathrm{card}(R_1) < \alpha_1$, then $\mathrm{card}(R_2) \geq \alpha_2$ and by definition $\mathrm{card}(R'_1) = \mathrm{card}(R_1)$. Then, $\mathrm{card}(R'_2) \geq \alpha_2$ trivially follows from $\mathrm{card}(R'_1) + \mathrm{card}(R'_2) \geq \alpha_1 + \alpha_2$. Symmetrically, $\min(\alpha_k, \mathrm{card}(R_k)) = \min(\alpha_k, \mathrm{card}(R'_k))$ holds when the second case **(CR.C2)** applies ($\mathrm{card}(R_2) < \alpha_2$). Lastly, suppose $\mathrm{card}(R_1) \geq \alpha_1$ and $\mathrm{card}(R_2) \geq \alpha_2$. Then, the third case **(CR.C3)** applies and by definition $\mathrm{card}(R'_1) = \alpha_1$. Again, we conclude that $\min(\alpha_k, \mathrm{card}(R_k)) = \min(\alpha_k, \mathrm{card}(R'_k))$ since $\mathrm{card}(R'_2) \geq \alpha_2$ trivially follows from $\mathrm{card}(R'_1) + \mathrm{card}(R'_2) \geq \alpha_1 + \alpha_2$.

(CI) Lastly, the heaps h'_1 and h'_2 are further populated with respect to the memory cells in $\mathrm{Inter}(\ell, \ell')$. For all $(\ell, \ell') \in \mathrm{E}$, let $L_k \stackrel{\text{def}}{=} \mathrm{Inter}(\ell, \ell') \cap \mathrm{dom}(h_k)$. We have $L_1 \uplus L_2 = \mathrm{Inter}(\ell, \ell')$. Below, we partition $\mathrm{Inter}'(\mathfrak{f}(\ell), \mathfrak{f}(\ell'))$ into L'_1 and L'_2 so that by definition $L'_k \subseteq \mathrm{dom}(h'_k)$:

(CI.C1) If $\mathrm{card}(L_1) < \alpha_1$ then L'_1 is a set of $\mathrm{card}(L_1)$ locations from $\mathrm{Inter}'(\mathfrak{f}(\ell), \mathfrak{f}(\ell'))$, whereas $L'_2 \stackrel{\text{def}}{=} \mathrm{Inter}'(\mathfrak{f}(\ell), \mathfrak{f}(\ell')) \setminus L'_1$.

(CI.C2) Else, if $\mathrm{card}(L_2) < \alpha_2$ then L'_2 is a set of $\mathrm{card}(L_2)$ locations from $\mathrm{Inter}'(\mathfrak{f}(\ell), \mathfrak{f}(\ell'))$, whereas $L'_1 \stackrel{\text{def}}{=} \mathrm{Inter}'(\mathfrak{f}(\ell), \mathfrak{f}(\ell')) \setminus L'_2$.

(CI.C3) Otherwise, we have $\mathrm{card}(L_1) \geq \alpha_1$ and $\mathrm{card}(L_2) \geq \alpha_2$. Then L'_1 is a set of α_1 locations from $\mathrm{Inter}'(\mathfrak{f}(\ell), \mathfrak{f}(\ell'))$ and $L'_2 \stackrel{\text{def}}{=} \mathrm{Inter}'(\mathfrak{f}(\ell), \mathfrak{f}(\ell')) \setminus L'_1$.

It is easy to show that the construction satisfies the following properties (where $k \in \{1, 2\}$):

$$\text{if } L'_k = \emptyset \text{ then } \mathrm{Inter}'(\mathfrak{f}(\ell), \mathfrak{f}(\ell')) \subseteq \mathrm{dom}(h'_{3-k}) \qquad \textbf{(CI.P1)}$$

$$\min(\alpha_k, \mathrm{card}(L_k)) = \min(\alpha_k, \mathrm{card}(L'_k)) \qquad \textbf{(CI.P2)}$$

(Proof of **(CI.P1)**) The first property trivially holds from the cases **(CI.C1)** and **(CI.C2)** of the construction. Notice that given $k \in \{1, 2\}$, $3 - k$ corresponds to the index in $\{1, 2\}$ that is different from k.
(Proof of **(CI.P2)**) The second property directly follows from **(A4)** (which is satisfied by $\mathfrak{f}$) and is proved as done for **(CR.P1)**.

This ends the construction of h'_1 and h'_2 as any location in $\mathrm{dom}(h')$ has been assigned to one of the two heaps. Indeed, $\{\mathrm{Alloc}', \mathrm{Rem}'\} \cup \{\mathrm{Inter}'(\ell, \ell') \mid (\ell, \ell') \in \mathrm{E}'\}$ is a partition of $\mathrm{dom}(h')$. As $(s, h) \approx^q_\alpha (s', h')$, the support graphs $\mathrm{SG}_q(s, h)$ and $\mathrm{SG}_q(s', h')$ witness the existence of a map $\mathfrak{f} : \mathrm{V} \to \mathrm{V}'$ satisfying **(A1)**–**(A5)** and therefore there is an underlying isomorphism between these structures satisfying quantitative properties up to the value α. The construction above can be understood as a way to split h' into h'_1 and h'_2 mimicking the splitting of h into h_1 and h_2. It remains to show below that this is done in a way that guarantees that $(s, h_k) \approx^q_{\alpha_k} (s', h'_k)$ $(k \in \{1, 2\})$.

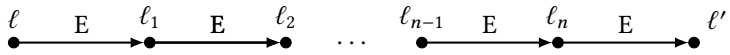
In the following, we denote the support graphs of (s, h_k) and (s', h'_k) respectively as

- $\mathrm{SG}_q(s, h_k) = (\mathrm{V}_k, \mathrm{E}_k, \mathrm{Alloc}_k, \mathrm{TEq}_k, \mathrm{Inter}_k, \mathrm{Rem}_k)$ and,
- $\mathrm{SG}_q(s', h'_k) = (\mathrm{V}'_k, \mathrm{E}'_k, \mathrm{Alloc}'_k, \mathrm{TEq}'_k, \mathrm{Inter}'_k, \mathrm{Rem}'_k)$.

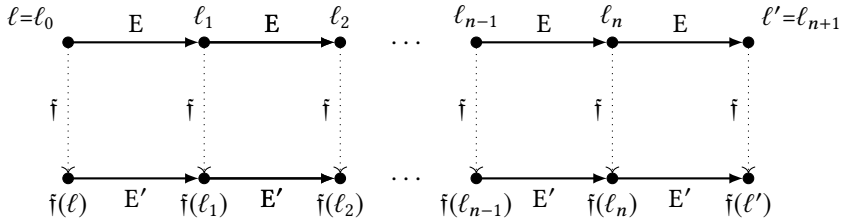
First, let us formalise an essential property of the construction of h'_1 and h'_2.

(Paths) Let $k \in \{1, 2\}$ and let $\ell, \ell' \in \mathrm{V}$ be two labelled locations w.r.t. (s, h). h_k witnesses a non-empty path from ℓ to ℓ' if and only if h'_k witnesses a non-empty path from $\mathfrak{f}(\ell)$ to $\mathfrak{f}(\ell')$.

(Proof of **(Paths)**) The proof mainly relies on the properties **(CI.P1)** and **(CI.P2)** of the construction. Recall that $\mathfrak{f} : \mathrm{V} \to \mathrm{V}'$ is a bijection satisfying **(A1)**–**(A5)** w.r.t. (s, h) and (s', h').
($\Rightarrow$) Let $\ell, \ell' \in \mathrm{V}$ be such that h_k witnesses a non-empty path from ℓ to ℓ'. Since $h_k \sqsubseteq h$, then h also witnesses a non-empty path from ℓ to ℓ'. In particular, by Definition 4.4, this path corresponds to a path in the support graph $\mathrm{SG}_q(s, h)$:



Let us define $\ell_0 \stackrel{\text{def}}{=} \ell$ and $\ell_{n+1} \stackrel{\text{def}}{=} \ell'$. In particular, following the picture above, the support graph $\mathrm{SG}_q(s, h)$ witnesses a path $\{(\ell_0, \ell_1), \ldots, (\ell_n, \ell_{n+1})\} \subseteq \mathrm{E}$ from ℓ_0 to ℓ_{n+1}. Since $\mathfrak{f}$ is a graph isomorphism from (V, E) to $(\mathrm{V}', \mathrm{E}')$ (by **(A1)**), $\mathrm{SG}_q(s', h')$ witnesses a similar structure, as depicted below:



Let us consider $i \in [0, n]$. Since the path belongs to h_k, it must hold that $\ell_i \in \mathrm{dom}(h_k)$ and $\mathrm{Inter}(\ell_i, \ell_{i+1}) \subseteq \mathrm{dom}(h_k)$. We show that then $\mathfrak{f}(\ell_i) \in \mathrm{dom}(h'_k)$ and $\mathrm{Inter}'(\mathfrak{f}(\ell_i), \mathfrak{f}(\ell_{i+1})) \subseteq \mathrm{dom}(h'_k)$, which entails that h'_k witnesses a path from $\mathfrak{f}(\ell)$ to $\mathfrak{f}(\ell')$, concluding the proof.

- From $(\ell_i, \ell_{i+1}) \in \mathrm{E}$, by Definition 4.4 we conclude that $\ell_i \in \mathrm{V}$. Since $\ell_i \in \mathrm{dom}(h)$, again by Definition 4.4, we have $\ell_i \in \mathrm{Alloc}$ and therefore by **(A2)**, $\mathfrak{f}(\ell_i) \in \mathrm{Alloc}'$. By **(CA)** together with the fact that $\ell_i \in \mathrm{dom}(h_k)$, we then conclude that $\mathfrak{f}(\ell_i) \in \mathrm{dom}(h'_k)$.
- If $\mathrm{Inter}(\ell_i, \ell_{i+1})$ is empty then by **(A4)** $\mathrm{Inter}'(\mathfrak{f}(\ell_i), \mathfrak{f}(\ell_{i+1}))$ is also empty and the inclusion w.r.t. $\mathrm{dom}(h'_k)$ trivially holds. Suppose now $\mathrm{Inter}(\ell_i, \ell_{i+1})$ is non-empty. From $\mathrm{Inter}(\ell_i, \ell_{i+1}) \subseteq \mathrm{dom}(h_k)$ and the fact that h_k and h_{3-k} are disjoint we conclude that $\mathrm{Inter}(\ell_i, \ell_{i+1}) \cap \mathrm{dom}(h_{3-k}) = \emptyset$. Hence, by **(CI.P2)** we conclude that $\mathrm{Inter}'(\mathfrak{f}(\ell_i), \mathfrak{f}(\ell_{i+1})) \cap \mathrm{dom}(h'_{3-k}) = \emptyset$ (notice that in **(CI.P2)**, L_{3-k} corresponds to $\mathrm{Inter}(\ell_i, \ell_{i+1}) \cap \mathrm{dom}(h_{3-k})$ whereas L'_{3-k} corresponds to $\mathrm{Inter}'(\mathfrak{f}(\ell_i), \mathfrak{f}(\ell_{i+1})) \cap \mathrm{dom}(h'_{3-k})$). Since $\mathrm{Inter}'(\mathfrak{f}(\ell_i), \mathfrak{f}(\ell_{i+1})) \cap \mathrm{dom}(h'_{3-k}) = \emptyset$, by **(CI.P1)** we then conclude that $\mathrm{Inter}'(\mathfrak{f}(\ell_i), \mathfrak{f}(\ell_{i+1})) \subseteq \mathrm{dom}(h'_k)$.

($\Leftarrow$) The right-to-left direction is analogous (thanks to the fact that $\mathfrak{f}^{-1}$ is a graph isomorphism from (V', E') to (V, E)).

Here is the last step of the proof. Given $k \in \{1, 2\}$, let $\mathfrak{f}_k$ be the restriction of $\mathfrak{f}$ to V_k and V'_k. We prove that $\mathfrak{f}_k$ satisfies **(A1)**–**(A5)** w.r.t. the memory states (s, h_k) and (s', h'_k) and α_k. Thanks to Lemma 4.7, this implies $(s, h_1) \approx^q_{\alpha_1} (s', h'_1)$ and $(s, h_2) \approx^q_{\alpha_2} (s, h'_2)$, ending the proof. Because of lack of space, below we prove **(A3)** and the proofs for **(A2)**, **(A1)**, **(A4)** and **(A5)** can be found in Appendix A.

$\mathfrak{f}_k$ **satisfies (A3)**: We prove that for every $\ell \in V$, the set of terms corresponding to ℓ in (s, h_k) is equal to the set of terms corresponding to $\mathfrak{f}(\ell)$ in (s', h'_k). Formally:

for every $\ell \in V$,
(a) $\ell \in V_k$ iff $\mathfrak{f}(\ell) \in V'_k$;
(b) if $\ell \in V_k$, $\mathrm{TEq}_k(\ell) = \mathrm{TEq}'_k(\mathfrak{f}(\ell))$.

Notice that (a) implies that $\mathfrak{f}_k$ (which we recall being the restriction of $\mathfrak{f}$ to V_k and V'_k) is well-defined and it is a bijection from the labelled locations of (s, h_k) (i.e. V_k) to the labelled locations of (s', h'_k) (i.e. V'_k). This is due to the fact that $\mathfrak{f}$ is a bijection from V to V' and by Lemma 4.3, we have $V_k \subseteq V$ and $V'_k \subseteq V'$. Again by $V_k \subseteq V$ (Lemma 4.3), (b) is then equivalent to **(A3)**.
We prove (a) and (b) together, by showing that for every $\ell \in V$ the set of terms corresponding to ℓ in (s, h_k) is equivalent to the set of terms corresponding to $\mathfrak{f}(\ell)$ in (s', h'_k). We first show the result for program variables, and then for meet-points.
(Program variables) Let $\ell \in V$ and $i \in [1, q]$. It holds that $\mathsf{x}_i \in \mathrm{TEq}_k(\ell)$ if and only if $s(\mathsf{x}_i) = \ell$, or equivalently $\mathsf{x}_i \in \mathrm{TEq}(\ell)$ which, by **(A3)** in Lemma 4.7, holds whenever $\mathsf{x}_i \in \mathrm{TEq}'(\mathfrak{f}(\ell))$. The latter is equivalent to $s'(\mathsf{x}_i) = \mathfrak{f}(\ell)$, or equivalently $\mathsf{x}_i \in \mathrm{TEq}'_k(\mathfrak{f}(\ell))$.
(Meet-points) In order to conclude the proof, we show that for all $i, j \in [1, q]$ and $\ell \in V$,

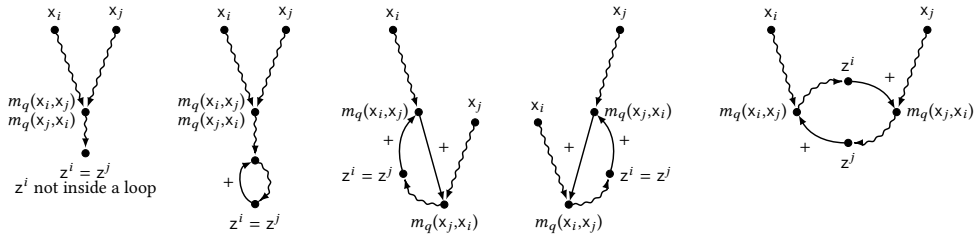
$$m_q(\mathsf{x}_i, \mathsf{x}_j) \in \mathrm{TEq}_k(\ell) \text{ if and only if } m_q(\mathsf{x}_i, \mathsf{x}_j) \in \mathrm{TEq}'_k(\mathfrak{f}(\ell)).$$

If $[\![m_q(\mathsf{x}_i, \mathsf{x}_j)]\!]^q_{s,h}$ is undefined, then so is $[\![m_q(\mathsf{x}_j, \mathsf{x}_i)]\!]^q_{s,h}$ (by def.) and by $(s, h) \approx^q_\alpha (s', h')$, so are $[\![m_q(\mathsf{x}_i, \mathsf{x}_j)]\!]^q_{s',h'}$ and $[\![m_q(\mathsf{x}_j, \mathsf{x}_i)]\!]^q_{s',h'}$. By Lemma 4.3, if $[\![m_q(\mathsf{x}_i, \mathsf{x}_j)]\!]^q_{s,h}$ is undefined, then so are $[\![m_q(\mathsf{x}_i, \mathsf{x}_j)]\!]^q_{s,h_k}$, $[\![m_q(\mathsf{x}_j, \mathsf{x}_i)]\!]^q_{s,h_k}$, $[\![m_q(\mathsf{x}_i, \mathsf{x}_j)]\!]^q_{s',h'_k}$ and $[\![m_q(\mathsf{x}_j, \mathsf{x}_i)]\!]^q_{s',h'_k}$. Otherwise, if $[\![m_q(\mathsf{x}_i, \mathsf{x}_j)]\!]^q_{s,h} = \ell$ then by definition $[\![m_q(\mathsf{x}_j, \mathsf{x}_i)]\!]^q_{s,h} = \ell'$ for some $\ell' \in V$ and moreover $[\![m_q(\mathsf{x}_i, \mathsf{x}_j)]\!]^q_{s',h'} = \mathfrak{f}(\ell)$ and $[\![m_q(\mathsf{x}_j, \mathsf{x}_i)]\!]^q_{s',h'} = \mathfrak{f}(\ell')$ by $(s, h) \approx^q_\alpha (s', h')$. Let z^i (resp. z^j) be the program variable in $\{\mathsf{x}_1, \ldots, \mathsf{x}_q\}$ such that $s(\mathsf{z}^i)$ is the first location corresponding to a program variable that is reachable from $[\![m_q(\mathsf{x}_i, \mathsf{x}_j)]\!]^q_{s,h}$ (resp. $[\![m_q(\mathsf{x}_j, \mathsf{x}_i)]\!]^q_{s,h}$), itself included. The characterisation of such a program variable z^i can be captured by the formula $\mathtt{firstvar}(m_q(\mathsf{x}_i, \mathsf{x}_j), \mathsf{z}^i)$ defined as the following Boolean combination of test formulae:

$$m_q(\mathsf{x}_i, \mathsf{x}_j) = \mathsf{z}^i \vee \bigvee_{\substack{v_1, \ldots, v_n \in \mathrm{Terms}_q, n>1 \\ \text{pairwise distinct } v_1, \ldots, v_{n-1}, \\ v_1 = m_q(\mathsf{x}_i, \mathsf{x}_j), v_n = \mathsf{z}^i}} \bigwedge_{\delta \in [1, n-1]} \mathtt{sees}_q(v_\delta, v_{\delta+1}) \geq 1 \wedge \bigwedge_{\substack{m<n \\ k \in [1,q]}} \mathsf{x}_k \neq v_m$$

Indeed, this formula is satisfied only by memory states where there is a (possibly empty) path from the location corresponding to $m_q(\mathsf{x}_i, \mathsf{x}_j)$ to the location $\overline{\ell}$ corresponding to z^i so that each labelled location in the path, apart from $\overline{\ell}$, does not correspond to the interpretation of such program variables. Now, we recall the taxonomy of meet-points, where the rightmost

case from Figure 5 is split into three cases, highlighting additional cases depending on z^i and z^j.



(i) (ii) (iii) (iv) (v)

Distinct structures satisfy different test formulae, though they all satisfy the formula

$$\mathtt{firstvar}(m_q(\mathsf{x}_i, \mathsf{x}_j), \mathsf{z}^i) \wedge \mathtt{firstvar}(m_q(\mathsf{x}_j, \mathsf{x}_i), \mathsf{z}^j).$$

For instance, (i) is the only form not satisfying $\mathtt{reach}^+(\mathsf{z}^i, \mathsf{z}^i)$ (recall that this form can be expressed as a Boolean combination of test formulae, as shown in Lemma 4.5), whereas (ii) can be distinguished as the only form satisfying both $\mathtt{reach}^+(\mathsf{z}^i, \mathsf{z}^i)$ and $m_q(\mathsf{x}_i, \mathsf{x}_j) = m_q(\mathsf{x}_j, \mathsf{x}_i)$. Moreover, the last structure is the only one satisfying $\mathsf{z}^i \neq \mathsf{z}^j$ whereas (iii) and (iv) can be distinguished with a formula, similar to $\mathtt{firstvar}(m_q(\mathsf{x}_i, \mathsf{x}_j), \mathsf{z}^i)$, stating that from the location corresponding to z^i, it is possible to reach the location corresponding to $m_q(\mathsf{x}_i, \mathsf{x}_j)$ without reaching the location corresponding to $m_q(\mathsf{x}_j, \mathsf{x}_i)$:

$$\bigvee_{\substack{v_1,\ldots,v_n \in \mathrm{Terms}_q,\ n>1 \\ \text{pairwise distinct } v_1,\ldots,v_{n-1}, \\ v_1=\mathsf{z}^i, v_n=m_q(\mathsf{x}_i,\mathsf{x}_j)}} \bigwedge_{\delta\in[1,n-1]} \mathtt{sees}_q(v_\delta, v_{\delta+1}) \geq 1 \wedge \bigwedge_{1<m<n} m_q(\mathsf{x}_j, \mathsf{x}_i) \neq v_m.$$

Differently from (iii), the structure (iv) does not satisfy this formula. Since $(s,h) \approx^q_\alpha (s',h')$, the heaps h and h' agree on the structure of every meet-point. The proof that for all $i,j \in [1,q]$ and $\ell \in \mathrm{V}$, $m_q(\mathsf{x}_i,\mathsf{x}_j) \in \mathrm{TEq}_k(\ell)$ iff $m_q(\mathsf{x}_i,\mathsf{x}_j) \in \mathrm{TEq}'_k(\tilde{\mathsf{f}}(\ell))$, essentially relies on **(Paths)**. To show the result we need to proceed by cases, according to the taxonomy.

Case: h witnesses (i), (ii) or (iv). Since the heaps h and h' agree on the structure of every meet-point, h' also witnesses the same form among (i), (ii) and (iv) as h. Regarding h_k, one of the following holds:

- The path from $s(\mathsf{x}_i)$ to $s(\mathsf{z}^i)$ and the path from $s(\mathsf{x}_j)$ to $s(\mathsf{z}^j)$ are both preserved in h_k. Then h_k witnesses (i) (ii) or (iv). By **(Paths)** the same holds for h'_k. In every case ((i), (ii) and (iv)), we conclude that $[\![m_q(\mathsf{x}_i,\mathsf{x}_j)]\!]^q_{s,h_k} = [\![m_q(\mathsf{x}_i,\mathsf{x}_j)]\!]^q_{s,h}$ and $[\![m_q(\mathsf{x}_i,\mathsf{x}_j)]\!]^q_{s',h'_k} = [\![m_q(\mathsf{x}_i,\mathsf{x}_j)]\!]^q_{s',h'} = \tilde{\mathsf{f}}([\![m_q(\mathsf{x}_i,\mathsf{x}_j)]\!]^q_{s,h})$. Then,

$$m_q(\mathsf{x}_i,\mathsf{x}_j) \in \mathrm{TEq}_k([\![m_q(\mathsf{x}_i,\mathsf{x}_j)]\!]^q_{s,h})$$
$$m_q(\mathsf{x}_i,\mathsf{x}_j) \in \mathrm{TEq}'_k(\tilde{\mathsf{f}}([\![m_q(\mathsf{x}_i,\mathsf{x}_j)]\!]^q_{s,h})).$$

- The path from $s(\mathsf{x}_i)$ to $s(\mathsf{z}^i)$ or the path from $s(\mathsf{x}_j)$ to $s(\mathsf{z}^j)$ are not preserved in h_k. Then, again by **(Paths)**, the same holds for h'_k with respect to (s',h'). By definition of meet-points, both $[\![m_q(\mathsf{x}_i,\mathsf{x}_j)]\!]^q_{s,h_k}$ and $[\![m_q(\mathsf{x}_i,\mathsf{x}_j)]\!]^q_{s',h'_k}$ are therefore not defined.

Case: h witnesses (iii). If instead h and h' witness (iii) then one of the following holds.

- h_k also witnesses (iii), meaning that the path from $s(\mathsf{x}_i)$ to $[\![m_q(\mathsf{x}_j,\mathsf{x}_i)]\!]^q_{s,h}$ and the path from $s(\mathsf{x}_j)$ to $[\![m_q(\mathsf{x}_i,\mathsf{x}_j)]\!]^q_{s,h}$ are both preserved in h_k. Then by **(Paths)** the heap h'_k

also witnesses (iii). We have $[\![m_q(\mathsf{x}_i,\mathsf{x}_j)]\!]^q_{s,h_k} = [\![m_q(\mathsf{x}_i,\mathsf{x}_j)]\!]^q_{s,h}$ and $[\![m_q(\mathsf{x}_i,\mathsf{x}_j)]\!]^q_{s',h'_k} = [\![m_q(\mathsf{x}_i,\mathsf{x}_j)]\!]^q_{s',h'} = \mathfrak{f}([\![m_q(\mathsf{x}_i,\mathsf{x}_j)]\!]^q_{s,h})$. We conclude that

$$m_q(\mathsf{x}_i,\mathsf{x}_j) \in \mathrm{TEq}_k([\![m_q(\mathsf{x}_i,\mathsf{x}_j)]\!]^q_{s,h_k})$$
$$m_q(\mathsf{x}_i,\mathsf{x}_j) \in \mathrm{TEq}'_k(\mathfrak{f}([\![m_q(\mathsf{x}_i,\mathsf{x}_j)]\!]^q_{s,h_k})).$$

- The path from $s(\mathsf{x}_i)$ to $s(\mathsf{z}^i)$ and the path from $s(\mathsf{x}_j)$ to $s(\mathsf{z}^i)$ are preserved in h_k, whereas the path from $s(\mathsf{z}^i)$ to $[\![m_q(\mathsf{x}_i,\mathsf{x}_j)]\!]^q_{s,h}$ is not preserved in h_k (i.e. at least one of its locations is assigned to the other heap h_{3-k}). Then h_k witnesses (i) and by definition of meet-points, it holds that

$$[\![m_q(\mathsf{x}_i,\mathsf{x}_j)]\!]^q_{s,h_k} = [\![m_q(\mathsf{x}_j,\mathsf{x}_i)]\!]^q_{s,h_k} = [\![m_q(\mathsf{x}_j,\mathsf{x}_i)]\!]^q_{s,h}$$

By **(Paths)**, h'_k also witnesses (i). Then,

$$[\![m_q(\mathsf{x}_i,\mathsf{x}_j)]\!]^q_{s',h'_k} = [\![m_q(\mathsf{x}_j,\mathsf{x}_i)]\!]^q_{s',h'_k} = [\![m_q(\mathsf{x}_j,\mathsf{x}_i)]\!]^q_{s',h'} = \mathfrak{f}([\![m_q(\mathsf{x}_j,\mathsf{x}_i)]\!]^q_{s,h}).$$

Then, we conclude that

$$m_q(\mathsf{x}_i,\mathsf{x}_j) \in \mathrm{TEq}_k([\![m_q(\mathsf{x}_j,\mathsf{x}_i)]\!]^q_{s,h})$$
$$m_q(\mathsf{x}_i,\mathsf{x}_j) \in \mathrm{TEq}'_k(\mathfrak{f}([\![m_q(\mathsf{x}_j,\mathsf{x}_i)]\!]^q_{s,h})).$$

- The path from $s(\mathsf{x}_i)$ to $s(\mathsf{z}^i)$ or the path from $s(\mathsf{x}_j)$ to $s(\mathsf{z}^i)$ are not preserved in h_k. Then, by **(Paths)**, the same holds for h'_k with respect to (s',h'). By definition of meet-points, neither $[\![m_q(\mathsf{x}_i,\mathsf{x}_j)]\!]^q_{s,h_k}$ nor $[\![m_q(\mathsf{x}_i,\mathsf{x}_j)]\!]^q_{s',h'_k}$ is defined.

Case: h witnesses (v). Lastly, suppose that h and h' witness (v). One of the following holds.

- The path from $s(\mathsf{x}_i)$ to $s(\mathsf{z}^i)$ and the path from $s(\mathsf{x}_j)$ to $s(\mathsf{z}^i)$ are both preserved in h_k. Then, depending on whether or not the path from $s(\mathsf{z}^i)$ to $[\![m_q(\mathsf{x}_j,\mathsf{x}^i)]\!]^q_{s,h}$ is also preserved, h_k witnesses (i) or (v). From **(Paths)**, the same holds for h'_k (where h'_k witnesses (i) iff h_k witnesses (i)). In both cases ((i) and (v)), it holds that $[\![m_q(\mathsf{x}_i,\mathsf{x}_j)]\!]^q_{s,h_k} = [\![m_q(\mathsf{x}_i,\mathsf{x}_j)]\!]^q_{s,h}$ and $[\![m_q(\mathsf{x}_i,\mathsf{x}_j)]\!]^q_{s',h'_k} = [\![m_q(\mathsf{x}_i,\mathsf{x}_j)]\!]^q_{s',h'} = \mathfrak{f}([\![m_q(\mathsf{x}_i,\mathsf{x}_j)]\!]^q_{s,h})$. Then,

$$m_q(\mathsf{x}_i,\mathsf{x}_j) \in \mathrm{TEq}_k([\![m_q(\mathsf{x}_i,\mathsf{x}_j)]\!]^q_{s,h})$$
$$m_q(\mathsf{x}_i,\mathsf{x}_j) \in \mathrm{TEq}'_k(\mathfrak{f}([\![m_q(\mathsf{x}_i,\mathsf{x}_j)]\!]^q_{s,h})).$$

- The path from $s(\mathsf{x}_i)$ to $s(\mathsf{z}^j)$ and the path from $s(\mathsf{x}_j)$ to $s(\mathsf{z}^j)$ are preserved in h_k, whereas the path from $s(\mathsf{z}^j)$ to $[\![m_q(\mathsf{x}_i,\mathsf{x}_j)]\!]^q_{s,h}$ is not preserved in h_k. Then h_k witnesses (i) and by definition of meet-points, it holds that

$$[\![m_q(\mathsf{x}_i,\mathsf{x}_j)]\!]^q_{s,h_k} = [\![m_q(\mathsf{x}_j,\mathsf{x}_i)]\!]^q_{s,h_k} = [\![m_q(\mathsf{x}_j,\mathsf{x}_i)]\!]^q_{s,h}.$$

By **(Paths)**, the heap h'_k also witnesses (i). Then,

$$[\![m_q(\mathsf{x}_i,\mathsf{x}_j)]\!]^q_{s',h'_k} = [\![m_q(\mathsf{x}_j,\mathsf{x}_i)]\!]^q_{s',h'_k} = [\![m_q(\mathsf{x}_j,\mathsf{x}_i)]\!]^q_{s',h'} = \mathfrak{f}([\![m_q(\mathsf{x}_j,\mathsf{x}_i)]\!]^q_{s,h})$$

Then, we conclude that

$$m_q(\mathsf{x}_i,\mathsf{x}_j) \in \mathrm{TEq}_k([\![m_q(\mathsf{x}_j,\mathsf{x}_i)]\!]^q_{s,h})$$
$$m_q(\mathsf{x}_i,\mathsf{x}_j) \in \mathrm{TEq}'_k(\mathfrak{f}([\![m_q(\mathsf{x}_j,\mathsf{x}_i)]\!]^q_{s,h})).$$

- The path from $s(\mathsf{x}_i)$ to $s(\mathsf{z}^j)$ or the path from $s(\mathsf{x}_j)$ to $s(\mathsf{z}^j)$ is not preserved in h_k. Then, by **(Paths)**, the same holds for h'_k with respect to (s', h'). By definition of meet-points, both $[\![m_q(\mathsf{x}_i, \mathsf{x}_j)]\!]^q_{s,h_k}$ and $[\![m_q(\mathsf{x}_i, \mathsf{x}_j)]\!]^q_{s',h'_k}$ are undefined.

We conclude that for every $\ell \in \mathrm{V}$, $m_q(\mathsf{x}_i, \mathsf{x}_j) \in \mathrm{TEq}_k(\ell)$ if and only if $m_q(\mathsf{x}_i, \mathsf{x}_j) \in \mathrm{TEq}'_k(\mathfrak{f}(\ell))$.

This concludes the proof: for the support graphs of h_k and h'_k, $\mathfrak{f}$ restricted to the domain V_k and the codomain V'_k satisfies all conditions of Lemma 4.7 with respect to α_k. Therefore, it holds that $(s, h_1) \approx^q_{\alpha_1} (s', h'_1)$ and $(s, h_2) \approx^q_{\alpha_2} (s', h'_2)$. □

Thanks to Lemma 4.8, we can now characterise every formula of SL$(*, \mathtt{reach}^+)$ by a Boolean combination of test formulae in Test(q, α), where α is related to the *memory size* $\mathtt{msize}(\varphi)$ of a formula φ in SL$(*, \mathtt{reach}^+)$ defined as follows (see also [41]):

- $\mathtt{msize}(\pi) \stackrel{\text{def}}{=} 1$ for any atomic formula π,
- $\mathtt{msize}(\psi \wedge \psi') \stackrel{\text{def}}{=} \max(\mathtt{msize}(\psi), \mathtt{msize}(\psi'))$,
- $\mathtt{msize}(\neg\psi) \stackrel{\text{def}}{=} \mathtt{msize}(\psi)$,
- $\mathtt{msize}(\psi * \psi') \stackrel{\text{def}}{=} \mathtt{msize}(\psi) + \mathtt{msize}(\psi')$.

We have $1 \leq \mathtt{msize}(\varphi) \leq |\varphi|$, where $|\varphi|$ is the size of the syntax tree for φ. Below, we establish the characterisation of SL$(*, \mathtt{reach}^+)$ formulae in terms of test formulae.

Theorem 4.9. *Let φ be in* SL$(*, \mathtt{reach}^+)$ *built over the variables* $\mathsf{x}_1, \ldots, \mathsf{x}_q$. *For all* $\alpha \geq \mathtt{msize}(\varphi)$ *and all memory states* (s, h), (s', h') *such that* $(s, h) \approx^q_\alpha (s', h')$, *we have* $(s, h) \models \varphi$ *iff* $(s', h') \models \varphi$.

Proof. Assume that φ is a formula with $\mathtt{msize}(\varphi) \leq \alpha$ and $(s, h) \approx^q_\alpha (s', h')$. By structural induction we show that $(s, h) \models \varphi$ iff $(s', h') \models \varphi$. It is sufficient to establish one direction of the equivalence thanks to its symmetry. First, note that $(s, h) \approx^q_\alpha (s', h')$ implies that (s, h) and (s', h') agree on the satisfaction of Boolean combinations built over atomic formulae in Test(q, α). Indeed, at the atomic level, this is exactly what $(s, h) \approx^q_\alpha (s', h')$ means, whereas for Boolean connectives $\neg$ and $\wedge$, this is also straighforward by induction.

The basic cases for the atomic formulae $\mathsf{x}_i \hookrightarrow \mathsf{x}_j$ and $\mathsf{x}_i = \mathsf{x}_j$ are immediate since these are test formulae. For $\mathtt{emp}$ and $\mathtt{reach}^+(\mathsf{x}_i, \mathsf{x}_j)$ we use directly Lemma 4.5. Suppose $\psi = \mathtt{emp}$. Then $\mathtt{msize}(\psi) = 1 \leq \alpha$ and we can express $\mathtt{emp}$ with the Boolean combination of test formulae

$$\neg\mathtt{sizeR}_q \geq 1 \wedge \bigwedge_{i \in [1,q]} \neg\mathtt{alloc}(\mathsf{x}_i),$$

as shown in the proof of Lemma 4.5. Notice that all the test formulae appearing in the formula above are from Test$(q, 1)$. Thus, from $(s, h) \approx^q_\alpha (s', h')$ we conclude that $(s, h) \models \psi$ iff $(s', h') \models \psi$.

Similarly, suppose that $\psi = \mathtt{reach}^+(\mathsf{x}_i, \mathsf{x}_j)$. Then $\mathtt{msize}(\psi) = 1 \leq \alpha$ and we can express ψ with the Boolean combination of test formulae:

$$\bigvee_{\substack{v_1, \ldots, v_n \in \mathrm{Terms}_q, \\ \text{pairwise distinct } v_1, \ldots, v_{n-1}, \\ \mathsf{x}_i = v_1, \mathsf{x}_j = v_n}} \bigwedge\nolimits_{\delta \in [1, n-1]} \mathtt{sees}_q(v_\delta, v_{\delta+1}) \geq 1,$$

as shown in the proof of Lemma 4.5. As in the case for the formula $\mathtt{emp}$, notice that all the test formulae appearing in the formula above are from Test$(q, 1)$. Again, from $(s, h) \approx^q_\alpha (s', h')$ we conclude that $(s, h) \models \psi$ iff $(s', h') \models \psi$.

We omit the obvious cases with the Boolean connectives. Let us consider the last case $\psi = \psi_1 * \psi_2$. Suppose that $(s, h) \models \psi_1 * \psi_2$ and $\mathtt{msize}(\psi_1 * \psi_2) \leq \alpha$. 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 4.8, 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 4, page 23. 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 4.9 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 $m_q(\mathtt{x}_i, \mathtt{x}_j) \hookrightarrow m_q(\mathtt{x}_j, \mathtt{x}_i)$. As a result, we can relate separation logic with classical logic, as advocated in the works [11, 17, 21, 29] and shown with the following theorem.

Theorem 4.10. *Let φ be a formula in* $\mathrm{SL}(*, \mathtt{reach}^+)$ *built over the variables in* $\mathtt{x}_1, \ldots, \mathtt{x}_q$. *The formula φ is logically equivalent to a Boolean combination of test formulae from* $\mathrm{Test}(q, \mathtt{msize}(\varphi))$.

Proof. The proof is rather standard. Let $\alpha = \mathtt{msize}(\varphi)$. Given a memory state (s, h), we write $\mathrm{LIT}(s, h)$ to denote the following set of literals:

$$\{\psi \in \mathrm{Test}(q, \alpha) \mid (s, h) \models \psi\} \cup \{\neg\psi \mid (s, h) \not\models \psi \text{ with } \psi \in \mathrm{Test}(q, \alpha)\}.$$

Since $\mathrm{Test}(q, \alpha)$ is a finite set, $\mathrm{LIT}(s, h)$ is finite too and let us consider the well-defined formula $\bigwedge_{\psi \in \mathrm{LIT}(s,h)} \psi$. We have the following equivalence:

$$(s', h') \models \bigwedge_{\psi \in \mathrm{LIT}(s,h)} \psi \quad \text{iff} \quad (s, h) \approx^q_\alpha (s', h').$$

The expression $\psi' \stackrel{\text{def}}{=} \bigvee_{(s,h) \models \varphi} (\bigwedge_{\psi \in \mathrm{LIT}(s,h)} \psi)$ is equivalent to a Boolean combination φ' of formulae from $\mathrm{Test}(q, \alpha)$ because $\mathrm{LIT}(s, h)$ ranges over the finite set of elements from $\mathrm{Test}(q, \alpha)$ (just select a finite amount of disjuncts). Though the number of memory states is infinite, the number of formulae of the form $(\bigwedge_{\psi \in \mathrm{LIT}(s,h)} \psi)$ is finite and therefore, we understand ψ' as a finite disjunction. This is not a constructive way to define ψ' (which can be also done by some other means) but this is sufficient for the existence. By Theorem 4.9, the formula φ is logically equivalent to ψ', which concludes the proof. Indeed, suppose that $(s, h) \models \varphi$. Obviously, we get $(s, h) \models \bigwedge_{\psi \in \mathrm{LIT}(s,h)} \psi$ and therefore $(s, h) \models \psi'$. Conversely, suppose that $(s, h) \models \psi'$. This means that there is a memory state (s', h') such that $(s', h') \models \varphi$ and $(s, h) \models \bigwedge_{\psi \in \mathrm{LIT}(s',h')} \psi$. Thus $(s, h) \approx^q_\alpha (s', h')$, $\mathtt{msize}(\varphi) \leq \alpha$ and since $(s', h') \models \varphi$, by Theorem 4.9 we get $(s, h) \models \varphi$. □

It is now possible to establish a small heap property of $\mathrm{SL}(*, \mathtt{reach}^+)$ by inheriting it from the small heap property for Boolean combinations of test formulae, which is analogous to the small model property for other theories of singly linked lists, see e.g. [14, 35]. Indeed, following Lemma 4.7, now it is straightforward to derive an upper bound on the size of a small model satisfying a formula in $\mathrm{SL}(*, \mathtt{reach}^+)$.

Let $\mathbf{P}(q, n)$ be the polynomial $(q^2 + q) \cdot (n + 1) + n$ used in the sequel.

Theorem 4.11. *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* $\mathrm{card}(\mathrm{dom}(h)) \leq \mathbf{P}(q, |\varphi|)$.

Proof. Let φ be a formula built over $\mathtt{x}_1, \ldots, \mathtt{x}_q$ with $\alpha = \mathtt{msize}(\varphi) \leq |\varphi|$ and let (s, h) be a memory state satisfying φ. Let $(\mathrm{V}, \mathrm{E}, \mathrm{Alloc}, \mathrm{TEq}, \mathrm{Inter}, \mathrm{Rem})$ be equal to $\mathrm{SG}_q(s, h)$ with respect to q and ℓ^* be an arbitrary location not in the domain of h. We construct a heap h' such that $\mathrm{card}(\mathrm{dom}(h')) \leq \mathbf{P}(q, |\varphi|)$ and $(s, h') \models \varphi$.

(1) Let $R \subseteq \mathrm{Rem}$ be a set of $\min(\alpha, \mathrm{card}(\mathrm{Rem}))$ locations. For all $\ell \in \mathrm{Rem}$, $\ell \in \mathrm{dom}(h') \stackrel{\text{def}}{\Leftrightarrow} \ell \in R$ and for all $\ell \in R$, we have $h'(\ell) \stackrel{\text{def}}{=} \ell^*$. As it will be soon clear, these locations in R will happen to be the only ones in Rem' (from h'). In that way, (s, h) and (s, h') shall agree on all test formulae of the form $\mathtt{sizeR}_q \geq \beta$ with $\beta \in [1, \alpha]$.

(2) For all $(\ell, \ell') \in \mathrm{E}$, let $L = \{\ell_1, \ldots, \ell_A\}$ be a set of $A = \min(\alpha, \mathrm{card}(\mathrm{Inter}(\ell, \ell')))$ locations in $\mathrm{Inter}(\ell, \ell')$. Notice that if $\mathrm{card}(\mathrm{Inter}(\ell, \ell')) \geq \alpha$, then $A = \alpha$, else $A = \mathrm{card}(\mathrm{Inter}(\ell, \ell'))$. Then, for all $\bar{\ell} \in \mathrm{Inter}(\ell, \ell')$, $\bar{\ell} \in \mathrm{dom}(h') \stackrel{\text{def}}{\Leftrightarrow} \bar{\ell} \in L$. Moreover $h'(\ell) \stackrel{\text{def}}{=} \ell_1$ (and therefore $\ell \in \mathrm{dom}(h')$), $h'(\ell_\alpha) \stackrel{\text{def}}{=} \ell'$ and for all $i \in [1, \alpha - 1]$, $h'(\ell_i) \stackrel{\text{def}}{=} \ell_{i+1}$. Since all the locations in $\mathrm{Inter}(\ell, \ell')$ are not labelled locations and we preserve the existence of paths between labelled locations, this step guarantees that for all $v \in \mathrm{Terms}_q$, we have $[\![v]\!]^q_{s,h} = [\![v]\!]^q_{s,h'}$. This implies that (s, h) and (s, h') satisfy the same set of test formulae of the form $v = v'$, where $v, v' \in \mathrm{Terms}_q$. Furthermore, the path from ℓ to ℓ' in h' has length $\min(\alpha, \mathrm{card}(\mathrm{Inter}(\ell, \ell')))+1$. Consequently, for all $\beta \in [1, \alpha]$ and for all $v, v' \in \mathrm{Terms}_q$, $(s, h') \models \mathtt{sees}_q(v, v') \geq \beta + 1$ if and only if $(s, h) \models \mathtt{sees}_q(v, v') \geq \beta + 1$, and these two memory states agree on the test formulae of the form $v \hookrightarrow v'$. As $\mathrm{card}(\mathrm{Terms}_q) = q^2 + q$, with this construction, each path between two labelled locations has at most length $\alpha + 1$. Additionally, the heap graphs are functional, and therefore $(s, h) \models \mathtt{sees}_q(v, v_1) \geq \beta_1 + 1 \wedge \mathtt{sees}_q(v, v_2) \geq \beta_2 + 1$ implies $v_1 = v_2$, which entails that this step adds less than $(q^2 + q)(|\varphi| + 1)$ locations to $\mathrm{dom}(h')$.

(3) Lastly, for all $\ell \in \mathrm{Alloc}$, $h'(\ell) \stackrel{\text{def}}{=} h(\ell)$ and therefore $\ell \in \mathrm{dom}(h')$. So, for all $v \in \mathrm{Terms}_q$ $(s, h') \models \mathtt{alloc}(v)$ if and only if $(s, h) \models \mathtt{alloc}(v)$. After this step, this implies that the two memory states satisfy the same set of test formulae. Note that in the computation of the upper bound, there is no need to take into account the location introduced in this step, since the upper bound mentioned in the previous step already includes this case.

It follows that h' is a heap such that $\mathrm{card}(\mathrm{dom}(h')) \leq (q^2 + q) \cdot (|\varphi| + 1) + |\varphi|$ and $(s, h') \models \varphi$. □

4.3 Complexity upper bounds

Let us draw some consequences of Theorem 4.11. First, for the logic $\mathrm{SL}(*, \mathtt{reach}^+)$, we get a PSpace upper bound, which matches the lower bound for $\mathrm{SL}(*)$ [12]. Indeed, $\mathrm{SL}(*)$ is a syntactic fragment of $\mathrm{SL}(*, \mathtt{reach}^+)$ and the satisfiability problem for $\mathrm{SL}(*)$ is shown PSpace-hard in [12].

Theorem 4.12. *The satisfiability problem for* $\mathrm{SL}(*, \mathtt{reach}^+)$ *is PSpace-complete.*

Proof. Let φ be a formula in $\mathrm{SL}(*, \mathtt{reach}^+)$ built over $\mathtt{x}_1, \ldots, \mathtt{x}_q$. By Theorem 4.11, φ is satisfiable if and only if there is a memory state satisfying φ with $\mathrm{card}(\mathrm{dom}(h)) \leq \mathbf{P}(q, |\varphi|)$. The nondeterministic polynomial-space algorithm (leading to the PSpace upper bound by Savitch's Theorem [37]) works as follows. First, guess a heap h with $\mathrm{card}(\mathrm{dom}(h)) \leq \mathbf{P}(q, |\varphi|)$ and $\mathrm{dom}(h) \cup \mathrm{ran}(h) \subseteq [0, 2 \times \mathbf{P}(q, |\varphi|)]$ and a store restricted to $\mathtt{x}_1, \ldots, \mathtt{x}_q$ such that $\mathrm{ran}(s) \subseteq [0, 2 \times \mathbf{P}(q, |\varphi|) + q]$ (in the worst case, all the variables have different values, and the memory cells have different values too).

Then, checking whether $(s, h) \models \varphi$ can be done in polynomial-space as for the standard $\mathrm{SL}(*)$ by using a recursive algorithm that internalises the semantics (see e.g. [12]): the recursive depth is linear and at each call, the algorithm uses at most linear space in the size of (s, h) and φ, which is polynomial in $|\varphi|$. We only need to guarantee that $(s, h) \models \mathtt{reach}^+(\mathtt{x}, \mathtt{y})$ can be checked in polynomial space, actually this can be done in polynomial time in the size of (s, h), that is therefore in polynomial space in $|\varphi|$. □

Besides, we may consider restricting the usage of Boolean connectives. Let us briefly define the symbolic heap fragment formulae φ as conjunctions $\Pi \wedge \Sigma$ where Π is a *pure formula* and Σ is a *spatial formula* Σ:

$$\Pi ::= \bot \mid \top \mid \mathtt{x}_i = \mathtt{x}_j \mid \neg(\mathtt{x}_i = \mathtt{x}_j) \mid \Pi \wedge \Pi$$
$$\Sigma ::= \mathtt{emp} \mid \top \mid \mathtt{x}_i \mapsto \mathtt{x}_j \mid \mathtt{ls}(\mathtt{x}_i, \mathtt{x}_j) \mid \Sigma * \Sigma$$

As usual, $\mathtt{x}_i \mapsto \mathtt{x}_j$ is interpreted as the exact points-to relation. We write Bool(SHF) to denote the set of 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 [13, 22], whereas the satisfiability problem for a slight variant of Bool(SHF) is shown in NP in [34, Theorem 4]. This NP upper bound can be obtained as a by-product of Theorem 4.11.

Corollary 4.13. *The satisfiability problem for* Bool(SHF) *is NP-complete.*

Proof. The NP-hardness is obtained thanks to the presence of equalities and Boolean connectives. Indeed, let us show how to simply reduce SAT to the satisfiability problem for Bool(SHF). Let φ be a formula from the propositional calculus built over the propositional variables $p_1, \ldots, p_n$. Let us define the translation T such that $\mathrm{T}(p_i) \stackrel{\text{def}}{=} \mathsf{x}_i = \mathsf{y}_i$ (x_i and y_i are program variables dedicated to p_i) and T is homomorphic for Boolean connectives. It is easy to prove that φ is satisfiable iff $\mathrm{T}(\varphi)$ is satisfiable.

As far as the complexity upper bound is concerned, let φ be a Boolean combination of pure or spatial formulae built over $\mathsf{x}_1, \ldots, \mathsf{x}_q$. By Theorem 4.11, φ is satisfiable iff there is (s, h) satisfying φ with $\mathrm{card}(\mathrm{dom}(h)) \leq \mathbf{P}(q, |\psi|)$, where ψ is the translation of φ where

- every occurrences of $\mathsf{x}_i \mapsto \mathsf{x}_j$ is rewritten as the equivalent formula $\mathsf{x}_i \hookrightarrow \mathsf{x}_j \wedge \mathtt{size} = 1$, and
- every $\mathtt{ls}(\mathsf{x}_i, \mathsf{x}_j)$ is rewritten as the equivalent formula

$$(\mathsf{x}_i = \mathsf{x}_j \wedge \mathtt{emp}) \vee (\mathsf{x}_i \neq \mathsf{x}_j \wedge \mathtt{reach}^+(\mathsf{x}_i, \mathsf{x}_j) \wedge \neg(\neg\mathtt{emp} * \mathtt{reach}^+(\mathsf{x}_i, \mathsf{x}_j))).$$

This technicality is introduced as Theorem 4.9 requires φ to be in $\mathrm{SL}(*, \mathtt{reach}^+)$ and, as such, the formula ψ is not then used to check for satisfiability.

The non-deterministic polynomial-time algorithm works as follows. Similarly to the proof of Theorem 4.12, guess a heap h with $\mathrm{card}(\mathrm{dom}(h)) \leq \mathbf{P}(q, |\psi|)$, $\mathrm{dom}(h) \cup \mathrm{ran}(h) \subseteq [0, 2 \times \mathbf{P}(q, |\psi|)]$ and a store restricted to $\mathsf{x}_1, \ldots, \mathsf{x}_q$ and such that $\mathrm{ran}(s) \subseteq [0, 2 \times \mathbf{P}(q, |\psi|) + q]$. Then, confirming that the valuation is correct can be done in PTime: checking whether a pure formula is satisfied by (s, h) can be done in linear time. Similarly, checking whether a spatial formula Σ is satisfied by (s, h) can be done in PTime [2, Lemma 1]. So, the satisfiability problem for Boolean combinations of symbolic heap fragment formulae can be solved in NP. □

We have seen that we can take advantage of the small heap property to derive complexity results for fragments of $\mathrm{SL}(*, \mathtt{reach}^+)$. However, it is also possible to push further the PSpace upper bound by allowing occurrences of $-\!\!*$ in a controlled way (as unrestricted use of the magic wand leads to undecidability, see Theorem 3.12). The following result can be shown thanks to Proposition 4.1 and Lemma 4.5.

Lemma 4.14. *Let φ be in* $\mathrm{SL}(*, -\!\!*)$ *built over* $\mathsf{x}_1, \ldots, \mathsf{x}_q$. *The formula φ is logically equivalent to a Boolean combination of test formulae from* $\mathrm{Test}(q, 2|\varphi|)$.

Proof. First, translate φ into a Boolean combination of formulae from

$$\mathsf{x}_i = \mathsf{x}_j \qquad \mathtt{alloc}(\mathsf{x}_i) \qquad \mathsf{x}_i \hookrightarrow \mathsf{x}_j \qquad \mathtt{size} \geq \beta,$$

as stated in Proposition 4.1 and $\beta \leq 2|\varphi|$. Then, rewrite every occurrence of $\mathtt{size} \geq \beta$ into the equivalent Boolean combination of test formulae shown in Lemma 4.5. The resulting formula is in $\mathrm{Test}(q, 2|\varphi|)$. □

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 extended as follows.

- $\mathtt{msize}(\mathtt{alloc}(v)) \stackrel{\text{def}}{=} 1$,
- $\mathtt{msize}(\mathtt{sees}_q(v, v') \geq \beta + 1) \stackrel{\text{def}}{=} \beta + 1$,
- $\mathtt{msize}(v \hookrightarrow v') \stackrel{\text{def}}{=} 1$,
- $\mathtt{msize}(\mathtt{sizeR} \geq \beta) \stackrel{\text{def}}{=} \beta$.

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 4.10 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 4.11, 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. Then, we conclude by stating the new PSpace upper bound for Boolean combinations of formulae from $\mathrm{SL}(*, \mathrel{-\!\!*}) \cup \mathrm{SL}(*, \mathtt{reach}^+)$.

Theorem 4.15. *The satisfiability problem for Boolean combinations of formulae from* $\mathrm{SL}(*, \mathrel{-\!\!*}) \cup \mathrm{SL}(*, \mathtt{reach}^+)$ *is PSpace-complete.*

Proof. Let φ be a Boolean combination of formulae from $\mathrm{SL}(*, \mathrel{-\!\!*}) \cup \mathrm{SL}(*, \mathtt{reach}^+)$. First, replace every maximal subformula ψ in $\mathrm{SL}(*, \mathrel{-\!\!*})$ by an equivalent Boolean combination of test formulae from $\mathrm{Test}(q, 2|\psi|)$, as shown by Lemma 4.14. This replacement may require exponential time in the worst case but this is still fine to establish the PSpace upper bound as we aim at showing a small heap property. We obtain a formula φ' in $\mathrm{SL}(*, \mathtt{reach}^+, \bigcup_{q,\alpha} \mathrm{Test}(q,\alpha))$ with $\alpha_{\varphi'} \leq 2|\varphi|$. So, φ is satisfiable iff φ has a model (s,h) with $\mathrm{card}(\mathrm{dom}(h)) \leq \mathbf{P}(q, 2|\varphi|\alpha_{\varphi'})$, which is still polynomial in $|\varphi|$. Again, the non-deterministic polynomial-space algorithm works as follows. First, guess a small heap h with $\mathrm{card}(\mathrm{dom}(h)) \leq \mathbf{P}(q, 2|\varphi|\alpha_{\varphi'})$, $\mathrm{dom}(h) \cup \mathrm{ran}(h) \subseteq [0, 2 \times \mathbf{P}(q, 2|\varphi|\alpha_{\varphi'})]$ and a store restricted to $\mathtt{x}_1, \ldots, \mathtt{x}_q$ and such that $\mathrm{ran}(s) \subseteq [0, 2 \times \mathbf{P}(q, 2|\varphi|\alpha_{\varphi'}) + q]$. Since the respective model-checking problem for $\mathrm{SL}(*, \mathrel{-\!\!*})$ and $\mathrm{SL}(*, \mathtt{reach}^+)$ are both in PSpace (see [12] and Theorem 4.12) and (s,h) is of polynomial size in $|\varphi|$, checking whether $(s,h) \models \varphi$ can be done in PSpace by performing several instances of the model-checking problem with maximal subformulae ψ from either $\mathrm{SL}(*, \mathrel{-\!\!*})$ or $\mathrm{SL}(*, \mathtt{reach}^+)$. □

The fragment from Theorem 4.15 forbids formulae with $\mathtt{ls}$ in the scope of the separating implication $\mathrel{-\!\!*}$. A fragment with $\mathtt{ls}$ in the scope of $\mathrel{-\!\!*}$ is considered in [40] but decidability is still open. By contrast, a recent work [30] has established a PSpace upper bound for fragments with $\mathtt{ls}$ in the scope of $\mathrel{-\!\!*}$ but restrictions apply (full proofs soon available in [31]).

5 CONCLUSION

We studied the effects of adding $\mathtt{ls}$ to $\mathrm{SL}(*, \mathrel{-\!\!*})$, giving us the opportunity to consider several variants. $\mathrm{SL}(*, \mathrel{-\!\!*}, \mathtt{ls})$ is shown undecidable (Theorem 3.12) and non-finitely axiomatisable, which remains quite unexpected since there are no first-order quantifications. This result is strengthened to even weaker extensions of $\mathrm{SL}(*, \mathrel{-\!\!*})$ such as the one augmented with $n(\mathsf{x}) = n(\mathsf{y})$, $n(\mathsf{x}) \hookrightarrow n(\mathsf{y})$ and $\mathtt{alloc}^{-1}(\mathsf{x})$, or the one augmented with $\mathtt{reach}(\mathsf{x},\mathsf{y}) = 2$ and $\mathtt{reach}(\mathsf{x},\mathsf{y}) = 3$. 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 have shown that the satisfiability problem for $\mathrm{SL}(*, \mathrel{-\!\!*}, \mathtt{reach}^+)$ restricted to Boolean combination of formulae from $\mathrm{SL}(*, \mathrel{-\!\!*})$ and $\mathrm{SL}(*, \mathtt{reach}^+)$ is also PSpace-complete. So, we have provided proof techniques to establish undecidability when $*$, $\mathrel{-\!\!*}$ and $\mathtt{ls}$ are present and to establish decidability based on test formulae. This paves the way to investigate the decidability status of $\mathrm{SL}(\mathrel{-\!\!*}, \mathtt{ls})$ as well as of the positive fragment of $\mathrm{SL}(*, \circledast, \mathtt{ls})$ from [39, 40].

Acknowledgments. We would like to thank the anonymous reviewers for their numerous suggestions and remarks that help us to improve the quality of the document.

REFERENCES

[1] T. Antonopoulos, N. Gorogiannis, C. Haase, M. Kanovich, and J. Ouaknine. 2014. Foundations for Decision Problems in Separation Logic with General Inductive Predicates. In *FoSSaCS'14 (Lecture Notes in Computer Science)*, Vol. 8412. Springer, 411–425.

[2] J. Berdine, C. Calcagno, and P. O'Hearn. 2004. A decidable fragment of separation logic. In *FSTTCS'04 (Lecture Notes in Computer Science)*, Vol. 3328. Springer, 97–109.

[3] J. Berdine, C. Calcagno, and P. O'Hearn. 2005. Smallfoot: Modular Automatic Assertion Checking with Separation Logic. In *FMCO'05 (Lecture Notes in Computer Science)*, Vol. 4111. Springer, 115–137.

[4] M. Bozga, R. Iosif, and S. Perarnau. 2010. Quantitative Separation Logic and Programs with Lists. *Journal of Automated Reasoning* 45, 2 (2010), 131–156.

[5] R. Brochenin, S. Demri, and E. Lozes. 2009. Reasoning about sequences of memory states. *Annals of Pure and Applied Logic* 161, 3 (2009), 305–323.

[6] R. Brochenin, S. Demri, and E. Lozes. 2012. On the Almighty Wand. *Information and Computation* 211 (2012), 106–137.

[7] J. Brotherston, C. Fuhs, N. Gorogiannis, and J. Navarro Perez. 2014. A Decision Procedure for Satisfiability in Separation Logic with Inductive Predicates. In *CSL-LiCS'14*.

[8] J. Brotherston and J. Villard. 2014. Parametric completeness for separation theories. In *POPL'14*. ACM, 453–464.

[9] C. Calcagno and D. Distefano. 2011. Infer: An Automatic Program Verifier for Memory Safety of C Programs. In *NASA Formal Methods (Lecture Notes in Computer Science)*, Vol. 6617. Springer, 459–465.

[10] C. Calcagno, D. Distefano, P.W. O'Hearn, and H. Yang. 2011. Compositional Shape Analysis by Means of Bi-Abduction. *J. ACM* 58, 6 (2011), 26:1–26:66.

[11] C. Calcagno, Ph. Gardner, and M. Hague. 2005. From separation logic to first-order logic. In *FoSSaCS'05 (Lecture Notes in Computer Science)*, Vol. 3441. Springer, 395–409.

[12] C. Calcagno, P. O'Hearn, and H. Yang. 2001. Computability and Complexity Results for a Spatial Assertion Language for Data Structures. In *FSTTCS'01 (Lecture Notes in Computer Science)*, Vol. 2245. Springer, 108–119.

[13] B. Cook, C. Haase, J. Ouaknine, M. Parkinson, and J. Worrell. 2011. Tractable Reasoning in a Fragment of Separation Logic. In *CONCUR'11 (Lecture Notes in Computer Science)*, Vol. 6901. Springer, 235–249.

[14] C. David, D. Kroening, and M. Lewis. 2015. Propositional Reasoning about Safety and Termination of Heap-Manipulating Programs. In *ESOP'15 (Lecture Notes in Computer Science)*, Vol. 9032. Springer, 661–684.

[15] S. Demri and M. Deters. 2015. Logical Investigations on Separation Logics. (Aug. 2015). http://www.lsv.ens-cachan.fr/Publis/PAPERS/PDF/DD-esslli15.pdf Lecture Notes, European Summer School on Logic, Language and Information (ESSLLI'15).

[16] S. Demri and M. Deters. 2016. Expressive Completeness of Separation Logic With Two Variables and No Separating Conjunction. *ACM Transations on Computational Logics* 17, 2 (2016), 12.

[17] S. Demri, D. Galmiche, D. Larchey-Wendling, and D. Mery. 2017. Separation Logic with One Quantified Variable. *Theory of Computing Systems* 61 (2017), 371–461.

[18] S. Demri, E. Lozes, and A. Mansutti. 2018. The Effects of Adding Reachability Predicates in Propositional Separation Logic. In *FoSSaCS'18 (Lecture Notes in Computer Science)*, Vol. 10803. Springer, 476–493.

[19] S. Demri, É. Lozes, and A. Mansutti. 2020. Internal Calculi for Separation Logics. In *CSL (LIPIcs)*, Vol. 152. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 19:1–19:18.

[20] D. Distefano, P. O'Hearn, and H. Yang. 2006. A Local Shape Analysis Based on Separation Logic. In *TACAS'06 (Lecture Notes in Computer Science)*, Vol. 3920. Springer, 287–302.

[21] M. Echenim, R. Iosif, and N. Peltier. 2020. The Bernays-Schönfinkel-Ramsey Class of Separation Logic with Uninterpreted Predicates. *ACM Transations on Computational Logics* 21, 3 (2020), 19:1–19:46.

[22] C. Haase, S. Ishtiaq, J. Ouaknine, and M. Parkinson. 2013. SeLoger: A Tool for Graph-Based Reasoning in Separation Logic. In *CAV'13 (Lecture Notes in Computer Science)*, Vol. 8044. Springer, 790–795.

[23] Z. Hou, R. Goré, and A. Tiu. 2015. Automated Theorem Proving for Assertions in Separation Logic with All Connectives. In *CADE'15 (Lecture Notes in Computer Science)*, Vol. 9195. Springer, 501–516.

[24] R. Iosif, A. Rogalewicz, and J. Simacek. 2013. The Tree Width of Separation Logic with Recursive Definitions. In *CADE'13 (Lecture Notes in Computer Science)*, Vol. 7898. Springer, 21–38.

[25] S. Ishtiaq and P. O'Hearn. 2001. BI as an Assertion Language for Mutable Data Structures. In *POPL'01*. ACM, 14–26.

[26] J. Katelaan, Ch. Matheja, and F. Zuleger. 2019. Effective Entailment Checking for Separation Logic with Inductive Definitions. In *TACAS'19 (Lecture Notes in Computer Science)*, Vol. 11428. Springer, 319–336.

[27] Q.L. Le, M. Tatsuta, J. Sun, and W.-N. Chin. 2017. A Decidable Fragment in Separation Logic with Inductive Predicates and Arithmetic. In *CAV'17 (Lecture Notes in Computer Science)*, Vol. 10427. Springer, 495–517.

[28] E. Lozes. 2004. *Expressivité des Logiques Spatiales*. PhD thesis. ENS Lyon.

[29] E. Lozes. 2004. Separation Logic preserves the expressive power of classical logic. In *SPACE'04*.

[30] A. Mansutti. 2018. Extending propositional separation logic for robustness properties. In *FSTTCS'18 (Leibniz International Proceedings in Informatics)*. Leibniz-Zentrum für Informatik, 42:1–42:23.
[31] A. Mansutti. 2020. *Reasoning with Separation Logics: Complexity, Expressive Power, Proof Systems*. Ph.D. Dissertation. Université Paris-Saclay.
[32] P. Müller, M. Schwerhoff, and A.J. Summers. 2016. Viper: A Verification Infrastructure for Permission-Based Reasoning. In *VMCAI'16 (Lecture Notes in Computer Science)*, Vol. 9583. Springer, 41–62.
[33] P.W. O'Hearn, J.C. Reynolds, and H. Yang. 2001. Local Reasoning about Programs that Alter Data Structures. In *CSL'01 (Lecture Notes in Computer Science)*, Vol. 2142. Springer, 1–19.
[34] R. Piskać, Th. Wies, and D. Zufferey. 2013. Automating Separation Logic using SMT. In *CAV'13 (Lecture Notes in Computer Science)*, Vol. 8044. Springer, 773–789.
[35] S. Ranise and C. Zarba. 2006. A theory of singly-linked lists and its extensible decision procedure. In *SEFM'06*. IEEE, 206–215.
[36] J.C. Reynolds. 2002. Separation logic: a logic for shared mutable data structures. In *LiCS'02*. IEEE, 55–74.
[37] W.J. Savitch. 1970. Relationships between nondeterministic and deterministic tape complexities. *J. Comput. System Sci.* 4, 2 (1970), 177–192.
[38] M. Schwerhoff and A. Summers. 2015. Lightweight support for magic wands in an automatic verifier. In *ECOOP'15*. Leibniz-Zentrum für Informatik, LIPICS, 999–1023.
[39] A. Thakur. 2014. *Symbolic Abstraction: Algorithms and Applications*. Ph.D. Dissertation. University of Wisconsin-Madison.
[40] A. Thakur, J. Breck, and T. Reps. 2014. Satisfiability modulo abstraction for separation logic with linked lists. In *SPIN'14*. ACM, 58–67.
[41] H. Yang. 2001. *Local Reasoning for Stateful Programs*. Ph.D. Dissertation. University of Illinois, Urbana-Champaign.
[42] H. Yang, O. Lee, J. Berdine, C. Calcagno, B. Cook, D. Distefano, and P. O'Hearn. 2008. Scalable Shape Analysis for Systems Code. In *CAV'08 (Lecture Notes in Computer Science)*, Vol. 5123. Springer, 385–398.

A ELECTRONIC APPENDIX

In this appendix, we present the full proof of Lemma 4.8. The material about the proof included in the body of the paper is a subset of the material below.

Proof. Let q, α, α_1, α_2, (s,h), (s',h'), h_1 and h_2 be defined as in the statement. Let

- $\mathrm{SG}_q(s,h) = (\mathrm{V},\mathrm{E},\mathrm{Alloc},\mathrm{TEq},\mathrm{Inter},\mathrm{Rem})$ and
- $\mathrm{SG}_q(s',h') = (\mathrm{V}',\mathrm{E}',\mathrm{Alloc}',\mathrm{TEq}',\mathrm{Inter}',\mathrm{Rem}')$

be the support graphs of (s,h) and (s',h') respectively, with respect to q. As $(s,h) \approx^q_\alpha (s',h')$, let $\mathfrak{f} : \mathrm{V} \to \mathrm{V}'$ be a map satisfying **(A1)**–**(A5)** from Lemma 4.7. Below, for the sake of conciseness, let $k \in \{1,2\}$ and $\mathrm{SG}_q(s,h_k) = (\mathrm{V}_k,\mathrm{E}_k,\mathrm{Alloc}_k,\mathrm{TEq}_k,\mathrm{Inter}_k,\mathrm{Rem}_k)$ be the support graph of (s,h_k).

The proof is rather long and can be summed up with the following steps.

(1) First, we define a strategy to split h' into h'_1 and h'_2 by closely following the way that h is split into h_1 and h_2. To do this, we look at the support graphs. For instance, suppose that Rem is split into two sets $R_1 \subseteq \mathrm{dom}(h_1)$ and $R_2 \subseteq \mathrm{dom}(h_2)$. By definition, it is quite easy to see that the sets R_1 and R_2 must be subsets of Rem_1 and Rem_2, respectively. Then, following Lemma 4.7, to obtain $(s,h_k) \approx^q_{\alpha_k} (s',h'_k)$, we are required to split Rem′ into $R'_1 \subseteq \mathrm{dom}(h'_1)$ and $R'_2 \subseteq \mathrm{dom}(h'_2)$ so that $\min(\alpha_k, \mathrm{card}(R_k)) = \min(\alpha_k, \mathrm{card}(R'_k))$. Indeed, otherwise the equisatisfaction of the test formulae of the form $\mathtt{sizeR}_q \geq \beta$ is not ensured (details on this are formalised later).

(2) After defining h'_1 and h'_2, we show that $(s,h_k) \approx^q_{\alpha_k} (s',h'_k)$. To do so, again, we follow Lemma 4.7 and we show that we can find suitable bijections from labelled locations of h_k to the ones of h'_k satisfying **(A1)**–**(A5)**.

According to the summary above, let us first define explicitly h'_1 and h'_2 via an iterative process that consists in adding locations to $\mathrm{dom}(h'_1)$ or to $\mathrm{dom}(h'_2)$. Whenever we enforce that $\ell \in \mathrm{dom}(h'_k)$, implicitly we have $h'_k(\ell) \stackrel{\text{def}}{=} h'(\ell)$ as h'_k is intended to be a subheap of h'.

(CA) For all $\ell \in \mathrm{Alloc}'$, $\ell \in \mathrm{dom}(h'_k) \stackrel{\text{def}}{\Leftrightarrow} \mathfrak{f}^{-1}(\ell) \in \mathrm{dom}(h_k)$. This step of the construction, as well as its usefulness, should be self-explanatory. For example, if $(s,h_k) \models \mathtt{alloc}(\mathtt{x}_i)$ then, by relying on **(A3)**, this step allows us to conclude that $(s',h'_k) \models \mathtt{alloc}(\mathtt{x}_i)$ (independently on how the definition of h'_k will be completed in the next steps of the construction).

(CR) The heaps h'_1 and h'_2 are further populated depending on Rem. Let $R_k = \mathrm{Rem} \cap \mathrm{dom}(h_k)$. By definition, we have $R_1 \uplus R_2 = \mathrm{Rem}$. Below, we partition Rem′ into two sets R'_1 and R'_2 so that by definition $R'_k \subseteq \mathrm{dom}(h'_k)$. The strategy for defining the partition is split into three cases:

(CR.C1) If $\mathrm{card}(R_1) < \alpha_1$ then R'_1 is a set of $\mathrm{card}(R_1)$ locations from Rem′ and $R'_2 \stackrel{\text{def}}{=} \mathrm{Rem}' \setminus R'_1$.

(CR.C2) Otherwise, if $\mathrm{card}(R_2) < \alpha_2$ then R'_2 is a set of $\mathrm{card}(R_2)$ locations of Rem′ and $R'_1 \stackrel{\text{def}}{=} \mathrm{Rem}' \setminus R'_2$.

(CR.C3) Otherwise we have $\mathrm{card}(R_1) \geq \alpha_1$ and $\mathrm{card}(R_2) \geq \alpha_2$. Then, R'_1 is a set of α_1 locations from Rem′ and $R'_2 \stackrel{\text{def}}{=} \mathrm{Rem}' \setminus R'_1$.

It is easy to show that the construction satisfies the following property (where $k \in \{1,2\}$):

$$\min(\alpha_k, \mathrm{card}(R_k)) = \min(\alpha_k, \mathrm{card}(R'_k)) \qquad \textbf{(CR.P1)}$$

The property **(CR.P1)** directly follows from the property **(A5)** satisfied by $\mathfrak{f}$. The proof (that can be applied also for the next step of the construction, see **(CI.P2)**), works as follows. First, suppose that the sets of remaining locations in the heap domain are small, i.e.

$$\min(\alpha, \mathrm{card}(\mathrm{Rem})) = \min(\alpha, \mathrm{card}(\mathrm{Rem}')) < \alpha_1 + \alpha_2.$$

So, $\mathrm{card}(\mathrm{Rem}) = \mathrm{card}(\mathrm{Rem}')$ and therefore $\mathrm{card}(R_1) + \mathrm{card}(R_2) = \mathrm{card}(R'_1) + \mathrm{card}(R'_2)$. By definition, $\mathrm{card}(R_1) = \mathrm{card}(R'_1)$ and $\mathrm{card}(R_2) = \mathrm{card}(R'_2)$ trivially hold for the cases **(CR.C1)** and **(CR.C2)**, whereas the case **(CR.C3)** ($\mathrm{card}(R_1) \geq \alpha_1$ and $\mathrm{card}(R_2) \geq \alpha_2$) can never be applied since $\mathrm{card}(R_1) + \mathrm{card}(R_2) < \alpha_1 + \alpha_2$. We conclude that $\min(\alpha_k, \mathrm{card}(R_k)) = \min(\alpha_k, \mathrm{card}(R'_k))$.
Second, suppose instead

$$\min(\alpha, \mathrm{card}(\mathrm{Rem})) = \min(\alpha, \mathrm{card}(\mathrm{Rem}')) = \alpha_1 + \alpha_2.$$

If the first case **(CR.C1)** applies, i.e. $\mathrm{card}(R_1) < \alpha_1$, then $\mathrm{card}(R_2) \geq \alpha_2$ and by definition $\mathrm{card}(R'_1) = \mathrm{card}(R_1)$. Then, $\mathrm{card}(R'_2) \geq \alpha_2$ trivially follows from $\mathrm{card}(R'_1) + \mathrm{card}(R'_2) \geq \alpha_1 + \alpha_2$. Symmetrically, $\min(\alpha_k, \mathrm{card}(R_k)) = \min(\alpha_k, \mathrm{card}(R'_k))$ holds when the second case **(CR.C2)** applies ($\mathrm{card}(R_2) < \alpha_2$). Lastly, suppose $\mathrm{card}(R_1) \geq \alpha_1$ and $\mathrm{card}(R_2) \geq \alpha_2$. Then, the third case **(CR.C3)** applies and by definition $\mathrm{card}(R'_1) = \alpha_1$. Again, we conclude that $\min(\alpha_k, \mathrm{card}(R_k)) = \min(\alpha_k, \mathrm{card}(R'_k))$ since $\mathrm{card}(R'_2) \geq \alpha_2$ trivially follows from $\mathrm{card}(R'_1) + \mathrm{card}(R'_2) \geq \alpha_1 + \alpha_2$.

(CI) Lastly, the heaps h'_1 and h'_2 are further populated with respect to the memory cells in $\mathrm{Inter}(\ell, \ell')$. For all $(\ell, \ell') \in \mathrm{E}$, let $L_k \stackrel{\mathrm{def}}{=} \mathrm{Inter}(\ell, \ell') \cap \mathrm{dom}(h_k)$. We have $L_1 \uplus L_2 = \mathrm{Inter}(\ell, \ell')$. Below, we partition $\mathrm{Inter}'(\mathfrak{f}(\ell), \mathfrak{f}(\ell'))$ into L'_1 and L'_2 so that by definition $L'_k \subseteq \mathrm{dom}(h'_k)$:

(CI.C1) If $\mathrm{card}(L_1) < \alpha_1$ then L'_1 is a set of $\mathrm{card}(L_1)$ locations from $\mathrm{Inter}'(\mathfrak{f}(\ell), \mathfrak{f}(\ell'))$, whereas $L'_2 \stackrel{\mathrm{def}}{=} \mathrm{Inter}'(\mathfrak{f}(\ell), \mathfrak{f}(\ell')) \setminus L'_1$.

(CI.C2) Else, if $\mathrm{card}(L_2) < \alpha_2$ then L'_2 is a set of $\mathrm{card}(L_2)$ locations from $\mathrm{Inter}'(\mathfrak{f}(\ell), \mathfrak{f}(\ell'))$, whereas $L'_1 \stackrel{\mathrm{def}}{=} \mathrm{Inter}'(\mathfrak{f}(\ell), \mathfrak{f}(\ell')) \setminus L'_2$.

(CI.C3) Otherwise, we have $\mathrm{card}(L_1) \geq \alpha_1$ and $\mathrm{card}(L_2) \geq \alpha_2$. Then L'_1 is a set of α_1 locations from $\mathrm{Inter}'(\mathfrak{f}(\ell), \mathfrak{f}(\ell'))$ and $L'_2 \stackrel{\mathrm{def}}{=} \mathrm{Inter}'(\mathfrak{f}(\ell), \mathfrak{f}(\ell')) \setminus L'_1$.

It is easy to show that the construction satisfies the following properties (where $k \in \{1, 2\}$):

$$\text{if } L'_k = \emptyset \text{ then } \mathrm{Inter}'(\mathfrak{f}(\ell), \mathfrak{f}(\ell')) \subseteq \mathrm{dom}(h'_{3-k}) \qquad \textbf{(CI.P1)}$$

$$\min(\alpha_k, \mathrm{card}(L_k)) = \min(\alpha_k, \mathrm{card}(L'_k)) \qquad \textbf{(CI.P2)}$$

(Proof of **(CI.P1)**) The first property trivially holds from the cases **(CI.C1)** and **(CI.C2)** of the construction. Notice that given $k \in \{1, 2\}$, $3 - k$ corresponds to the index in $\{1, 2\}$ that is different from k.
(Proof of **(CI.P2)**) The second property directly follows from **(A4)** (which is satisfied by $\mathfrak{f}$) and is proved as done for **(CR.P1)**.

This ends the construction of h'_1 and h'_2 as any location in $\mathrm{dom}(h')$ has been assigned to one of the two heaps. Indeed, $\{\mathrm{Alloc}', \mathrm{Rem}'\} \cup \{\mathrm{Inter}'(\ell, \ell') \mid (\ell, \ell') \in \mathrm{E}'\}$ is a partition of $\mathrm{dom}(h')$. As $(s, h) \approx^q_\alpha (s', h')$, the support graphs $\mathrm{SG}_q(s, h)$ and $\mathrm{SG}_q(s', h')$ witness the existence of a map $\mathfrak{f} : \mathrm{V} \to \mathrm{V}'$ satisfying **(A1)**–**(A5)** and therefore there is an underlying isomorphism between these structures satisfying quantitative properties up to the value α. The construction above can be understood as a way to split h' into h'_1 and h'_2 mimicking the splitting of h into h_1 and h_2. It remains to show below that this is done in a way that guarantees that $(s, h_k) \approx^q_{\alpha_k} (s', h'_k)$ ($k \in \{1, 2\}$).

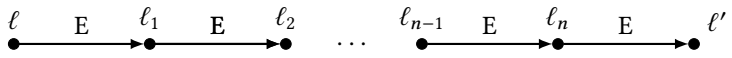
In the following, we denote the support graphs of (s, h_k) and (s', h'_k) respectively as

- $\mathrm{SG}_q(s, h_k) = (\mathrm{V}_k, \mathrm{E}_k, \mathrm{Alloc}_k, \mathrm{TEq}_k, \mathrm{Inter}_k, \mathrm{Rem}_k)$ and,
- $\mathrm{SG}_q(s', h'_k) = (\mathrm{V}'_k, \mathrm{E}'_k, \mathrm{Alloc}'_k, \mathrm{TEq}'_k, \mathrm{Inter}'_k, \mathrm{Rem}'_k)$.

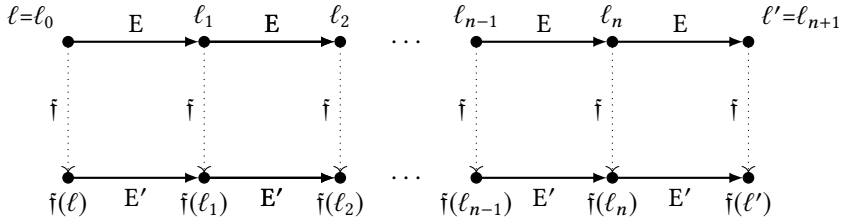
First, let us formalise an essential property of the construction of h'_1 and h'_2.

(Paths) Let $k \in \{1,2\}$ and let $\ell, \ell' \in \mathrm{V}$ be two labelled locations w.r.t. (s,h). h_k witnesses a non-empty path from ℓ to ℓ' if and only if h'_k witnesses a non-empty path from $\mathfrak{f}(\ell)$ to $\mathfrak{f}(\ell')$.

(Proof of **(Paths)**) The proof mainly relies on the properties **(CI.P1)** and **(CI.P2)** of the construction. Recall that $\mathfrak{f} : \mathrm{V} \to \mathrm{V}'$ is a bijection satisfying **(A1)**–**(A5)** w.r.t. (s,h) and (s',h').
($\Rightarrow$) Let $\ell, \ell' \in \mathrm{V}$ be such that h_k witnesses a non-empty path from ℓ to ℓ'. Since $h_k \sqsubseteq h$, then h also witnesses a non-empty path from ℓ to ℓ'. In particular, by Definition 4.4, this path corresponds to a path in the support graph $\mathrm{SG}_q(s,h)$:



Let us define $\ell_0 \stackrel{\text{def}}{=} \ell$ and $\ell_{n+1} \stackrel{\text{def}}{=} \ell'$. In particular, following the picture above, the support graph $\mathrm{SG}_q(s,h)$ witnesses a path $\{(\ell_0,\ell_1),\ldots,(\ell_n,\ell_{n+1})\} \subseteq \mathrm{E}$ from ℓ_0 to ℓ_{n+1}. Since $\mathfrak{f}$ is a graph isomorphism from (V,E) to $(\mathrm{V}',\mathrm{E}')$ (by **(A1)**), $\mathrm{SG}_q(s',h')$ witnesses a similar structure, as depicted below:



Let us consider $i \in [0,n]$. Since the path belongs to h_k, it must hold that $\ell_i \in \mathrm{dom}(h_k)$ and $\mathrm{Inter}(\ell_i,\ell_{i+1}) \subseteq \mathrm{dom}(h_k)$. We show that then $\mathfrak{f}(\ell_i) \in \mathrm{dom}(h'_k)$ and $\mathrm{Inter}'(\mathfrak{f}(\ell_i),\mathfrak{f}(\ell_{i+1})) \subseteq \mathrm{dom}(h'_k)$, which entails that h'_k witnesses a path from $\mathfrak{f}(\ell)$ to $\mathfrak{f}(\ell')$, concluding the proof.

- From $(\ell_i,\ell_{i+1}) \in \mathrm{E}$, by Definition 4.4 we conclude that $\ell_i \in \mathrm{V}$. Since $\ell_i \in \mathrm{dom}(h)$, again by Definition 4.4, we have $\ell_i \in \mathrm{Alloc}$ and therefore by **(A2)**, $\mathfrak{f}(\ell_i) \in \mathrm{Alloc}'$. By **(CA)** together with the fact that $\ell_i \in \mathrm{dom}(h_k)$, we then conclude that $\mathfrak{f}(\ell_i) \in \mathrm{dom}(h'_k)$.
- If $\mathrm{Inter}(\ell_i,\ell_{i+1})$ is empty then by **(A4)** $\mathrm{Inter}'(\mathfrak{f}(\ell_i),\mathfrak{f}(\ell_{i+1}))$ is also empty and the inclusion w.r.t. $\mathrm{dom}(h'_k)$ trivially holds. Suppose now $\mathrm{Inter}(\ell_i,\ell_{i+1})$ is non-empty. From $\mathrm{Inter}(\ell_i,\ell_{i+1}) \subseteq \mathrm{dom}(h_k)$ and the fact that h_k and h_{3-k} are disjoint we conclude that $\mathrm{Inter}(\ell_i,\ell_{i+1}) \cap \mathrm{dom}(h_{3-k}) = \emptyset$. Hence, by **(CI.P2)** we conclude that $\mathrm{Inter}'(\mathfrak{f}(\ell_i),\mathfrak{f}(\ell_{i+1})) \cap \mathrm{dom}(h'_{3-k}) = \emptyset$ (notice that in **(CI.P2)**, L_{3-k} corresponds to $\mathrm{Inter}(\ell_i,\ell_{i+1}) \cap \mathrm{dom}(h_{3-k})$ whereas L'_{3-k} corresponds to $\mathrm{Inter}'(\mathfrak{f}(\ell_i),\mathfrak{f}(\ell_{i+1})) \cap \mathrm{dom}(h'_{3-k})$). Since $\mathrm{Inter}'(\mathfrak{f}(\ell_i),\mathfrak{f}(\ell_{i+1})) \cap \mathrm{dom}(h'_{3-k}) = \emptyset$, by **(CI.P1)** we then conclude that $\mathrm{Inter}'(\mathfrak{f}(\ell_i),\mathfrak{f}(\ell_{i+1})) \subseteq \mathrm{dom}(h'_k)$.

($\Leftarrow$) The right-to-left direction is analogous (thanks to the fact that $\mathfrak{f}^{-1}$ is a graph isomorphism from $(\mathrm{V}',\mathrm{E}')$ to (V,E)).

Here is the last step of the proof. Given $k \in \{1,2\}$, let $\mathfrak{f}_k$ be the restriction of $\mathfrak{f}$ to V_k and V'_k. We prove that $\mathfrak{f}_k$ satisfies **(A1)**–**(A5)** w.r.t. the memory states (s,h_k) and (s',h'_k). Thanks to Lemma 4.7, this implies $(s,h_1) \approx^q_{\alpha_1} (s',h'_1)$ and $(s,h_2) \approx^q_{\alpha_2} (s',h'_2)$, ending the proof. For convenience, we prove the five properties in the following order: **(A3)**, **(A2)**, **(A1)**, **(A4)** and **(A5)** (in the body of the paper only the proof of **(A3)** is provided).

$\mathfrak{f}_k$ satisfies (A3): We prove that for every $\ell \in \mathrm{V}$, the set of terms corresponding to ℓ in (s,h_k) is equivalent to the set of terms corresponding to $\mathfrak{f}(\ell)$ in (s',h'_k). Formally:

for every $\ell \in V$,

(a) $\ell \in V_k$ iff $\mathfrak{f}(\ell) \in V'_k$;
(b) if $\ell \in V_k$, $\mathrm{TEq}_k(\ell) = \mathrm{TEq}'_k(\mathfrak{f}(\ell))$.

Notice that (a) implies that $\mathfrak{f}_k$ (which we recall being the restriction of $\mathfrak{f}$ to V_k and V'_k) is well-defined and it is a bijection from the labelled locations of (s, h_k) (i.e. V_k) to the labelled locations of (s', h'_k) (i.e. V'_k). This is due to the fact that $\mathfrak{f}$ is a bijection from V to V′ and by Lemma 4.3, we have $V_k \subseteq V$ and $V'_k \subseteq V'$. Again by $V_k \subseteq V$ (Lemma 4.3), (b) is then equivalent to **(A3)**.

We prove (a) and (b) together, by showing that for every $\ell \in V$, the set of terms corresponding to ℓ in (s, h_k) is equivalent to the set of terms corresponding to $\mathfrak{f}(\ell)$ in (s', h'_k). We first show the result for program variables, and then for meet-points.

(Program variables) Let $\ell \in V$ and $i \in [1, q]$. It holds that $\mathsf{x}_i \in \mathrm{TEq}_k(\ell)$ if and only if $s(\mathsf{x}_i) = \ell$, or equivalently $\mathsf{x}_i \in \mathrm{TEq}(\ell)$ which, by **(A3)** in Lemma 4.7, holds whenever $\mathsf{x}_i \in \mathrm{TEq}'(\mathfrak{f}(\ell))$. The latter is equivalent to $s'(\mathsf{x}_i) = \mathfrak{f}(\ell)$, or equivalently $\mathsf{x}_i \in \mathrm{TEq}'_k(\mathfrak{f}(\ell))$.

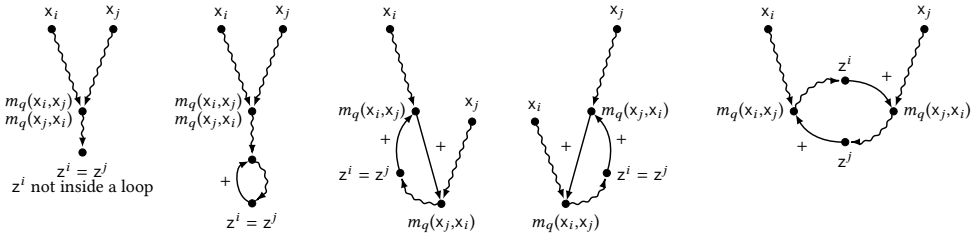
(Meet-points) In order to conclude the proof, we show that for all $i, j \in [1, q]$ and $\ell \in V$,

$$m_q(\mathsf{x}_i, \mathsf{x}_j) \in \mathrm{TEq}_k(\ell) \text{ if and only if } m_q(\mathsf{x}_i, \mathsf{x}_j) \in \mathrm{TEq}'_k(\mathfrak{f}(\ell)).$$

If $[\![m_q(\mathsf{x}_i, \mathsf{x}_j)]\!]^q_{s,h}$ is undefined, then so is $[\![m_q(\mathsf{x}_j, \mathsf{x}_i)]\!]^q_{s,h}$ (by def.) and by $(s, h) \approx^q_\alpha (s', h')$, so are $[\![m_q(\mathsf{x}_i, \mathsf{x}_j)]\!]^q_{s',h'}$ and $[\![m_q(\mathsf{x}_j, \mathsf{x}_i)]\!]^q_{s',h'}$. By Lemma 4.3, if $[\![m_q(\mathsf{x}_i, \mathsf{x}_j)]\!]^q_{s,h}$ is undefined, then so are $[\![m_q(\mathsf{x}_i, \mathsf{x}_j)]\!]^q_{s,h_k}$, $[\![m_q(\mathsf{x}_j, \mathsf{x}_i)]\!]^q_{s,h_k}$, $[\![m_q(\mathsf{x}_i, \mathsf{x}_j)]\!]^q_{s',h'_k}$ and $[\![m_q(\mathsf{x}_j, \mathsf{x}_i)]\!]^q_{s',h'_k}$. Otherwise, if $[\![m_q(\mathsf{x}_i, \mathsf{x}_j)]\!]^q_{s,h} = \ell$ then by definition $[\![m_q(\mathsf{x}_j, \mathsf{x}_i)]\!]^q_{s,h} = \ell'$ for some $\ell' \in V$ and moreover $[\![m_q(\mathsf{x}_i, \mathsf{x}_j)]\!]^q_{s',h'} = \mathfrak{f}(\ell)$ and $[\![m_q(\mathsf{x}_j, \mathsf{x}_i)]\!]^q_{s',h'} = \mathfrak{f}(\ell')$ by $(s, h) \approx^q_\alpha (s', h')$. Let z^i (resp. z^j) be the program variable in $\{\mathsf{x}_1, \ldots, \mathsf{x}_q\}$ such that $s(\mathsf{z}^i)$ is the first location corresponding to a program variable that is reachable from $[\![m_q(\mathsf{x}_i, \mathsf{x}_j)]\!]^q_{s,h}$ (resp. $[\![m_q(\mathsf{x}_j, \mathsf{x}_i)]\!]^q_{s,h}$), itself included. The characterisation of such a program variable z^i can be captured by the formula $\mathtt{firstvar}(m_q(\mathsf{x}_i, \mathsf{x}_j), \mathsf{z}^i)$ defined as the following Boolean combination of test formulae:

$$m_q(\mathsf{x}_i, \mathsf{x}_j) = \mathsf{z}^i \vee \bigvee_{\substack{v_1, \ldots, v_n \in \mathrm{Terms}_q, n>1 \\ \text{pairwise distinct } v_1, \ldots, v_{n-1}, \\ v_1 = m_q(\mathsf{x}_i, \mathsf{x}_j), v_n = \mathsf{z}^i}} \bigwedge_{\delta \in [1, n-1]} \mathtt{sees}_q(v_\delta, v_{\delta+1}) \geq 1 \wedge \bigwedge_{\substack{m<n \\ k \in [1,q]}} \mathsf{x}_k \neq v_m$$

Indeed, this formula is satisfied only by memory states where there is a (possibly empty) path from the location corresponding to $m_q(\mathsf{x}_i, \mathsf{x}_j)$ to the location $\overline{\ell}$ corresponding to z^i so that each labelled location in the path, apart from $\overline{\ell}$, does not correspond to the interpretation of such program variables. Now, we recall the taxonomy of meet-points, where the rightmost case from Figure 5 is split into three cases, highlighting additional cases depending on z^i and z^j.



(i) (ii) (iii) (iv) (v)

Distinct structures satisfy different test formulae, though they all satisfy the formula

$$\texttt{firstvar}(m_q(\mathsf{x}_i, \mathsf{x}_j), \mathsf{z}^i) \wedge \texttt{firstvar}(m_q(\mathsf{x}_j, \mathsf{x}_i), \mathsf{z}^j).$$

For instance, (i) is the only form not satisfying $\texttt{reach}^+(\mathsf{z}^i, \mathsf{z}^i)$ (recall that this form can be expressed as a Boolean combination of test formulae, as shown in Lemma 4.5), whereas (ii) can be distinguished as the only form satisfying both $\texttt{reach}^+(\mathsf{z}^i, \mathsf{z}^i)$ and $m_q(\mathsf{x}_i, \mathsf{x}_j) = m_q(\mathsf{x}_j, \mathsf{x}_i)$. Moreover, the last structure is the only one satisfying $\mathsf{z}^i \neq \mathsf{z}^j$ whereas (iii) and (iv) can be distinguished with a formula, similar to $\texttt{firstvar}(m_q(\mathsf{x}_i, \mathsf{x}_j), \mathsf{z}^i)$, stating that from the location corresponding to z^i, it is possible to reach the location corresponding to $m_q(\mathsf{x}_i, \mathsf{x}_j)$ without reaching the location corresponding to $m_q(\mathsf{x}_j, \mathsf{x}_i)$:

$$\bigvee_{\substack{v_1,\ldots,v_n \in \mathrm{Terms}_q,\ n>1 \\ \text{pairwise distinct } v_1,\ldots,v_{n-1}, \\ v_1=\mathsf{z}^i, v_n=m_q(\mathsf{x}_i,\mathsf{x}_j)}} \bigwedge_{\delta\in[1,n-1]} \texttt{sees}_q(v_\delta, v_{\delta+1}) \geq 1 \wedge \bigwedge_{1<m<n} m_q(\mathsf{x}_j, \mathsf{x}_i) \neq v_m.$$

Differently from (iii), the structure (iv) does not satisfy this formula. Since $(s,h) \approx^q_\alpha (s',h')$, the heaps h and h' agree on the structure of every meet-point. The proof that for all $i,j \in [1,q]$ and $\ell \in \mathrm{V}$, $m_q(\mathsf{x}_i,\mathsf{x}_j) \in \mathrm{TEq}_k(\ell)$ iff $m_q(\mathsf{x}_i,\mathsf{x}_j) \in \mathrm{TEq}'_k(\tilde{\mathsf{f}}(\ell))$, essentially relies on **(Paths)**. To show the result we need to proceed by cases, according to the taxonomy.

Case: h witnesses (i), (ii) or (iv). Since the heaps h and h' agree on the structure of every meet-point, h' also witnesses the same form among (i), (ii) and (iv) as h. Regarding h_k, one of the following holds:

- The path from $s(\mathsf{x}_i)$ to $s(\mathsf{z}^i)$ and the path from $s(\mathsf{x}_j)$ to $s(\mathsf{z}^j)$ are both preserved in h_k. Then h_k witnesses (i) (ii) or (iv). By **(Paths)** the same holds for h'_k. In every case ((i), (ii) and (iv)), we conclude that $[\![m_q(\mathsf{x}_i,\mathsf{x}_j)]\!]^q_{s,h_k} = [\![m_q(\mathsf{x}_i,\mathsf{x}_j)]\!]^q_{s,h}$ and $[\![m_q(\mathsf{x}_i,\mathsf{x}_j)]\!]^q_{s',h'_k} = [\![m_q(\mathsf{x}_i,\mathsf{x}_j)]\!]^q_{s',h'} = \tilde{\mathsf{f}}([\![m_q(\mathsf{x}_i,\mathsf{x}_j)]\!]^q_{s,h})$. Then,

$$m_q(\mathsf{x}_i,\mathsf{x}_j) \in \mathrm{TEq}_k([\![m_q(\mathsf{x}_i,\mathsf{x}_j)]\!]^q_{s,h})$$
$$m_q(\mathsf{x}_i,\mathsf{x}_j) \in \mathrm{TEq}'_k(\tilde{\mathsf{f}}([\![m_q(\mathsf{x}_i,\mathsf{x}_j)]\!]^q_{s,h})).$$

- The path from $s(\mathsf{x}_i)$ to $s(\mathsf{z}^i)$ or the path from $s(\mathsf{x}_j)$ to $s(\mathsf{z}^j)$ are not preserved in h_k. Then, again by **(Paths)**, the same holds for h'_k with respect to (s',h'). By definition of meet-points, both $[\![m_q(\mathsf{x}_i,\mathsf{x}_j)]\!]^q_{s,h_k}$ and $[\![m_q(\mathsf{x}_i,\mathsf{x}_j)]\!]^q_{s',h'_k}$ are therefore not defined.

Case: h witnesses (iii). If instead h and h' witness (iii) then one of the following holds.

- h_k also witnesses (iii), meaning that the path from $s(\mathsf{x}_i)$ to $[\![m_q(\mathsf{x}_j,\mathsf{x}_i)]\!]^q_{s,h}$ and the path from $s(\mathsf{x}_j)$ to $[\![m_q(\mathsf{x}_i,\mathsf{x}_j)]\!]^q_{s,h}$ are both preserved in h_k. Then by **(Paths)** the heap h'_k also witnesses (iii). We have $[\![m_q(\mathsf{x}_i,\mathsf{x}_j)]\!]^q_{s,h_k} = [\![m_q(\mathsf{x}_i,\mathsf{x}_j)]\!]^q_{s,h}$ and $[\![m_q(\mathsf{x}_i,\mathsf{x}_j)]\!]^q_{s',h'_k} = [\![m_q(\mathsf{x}_i,\mathsf{x}_j)]\!]^q_{s',h'} = \tilde{\mathsf{f}}([\![m_q(\mathsf{x}_i,\mathsf{x}_j)]\!]^q_{s,h})$. We conclude that

$$m_q(\mathsf{x}_i,\mathsf{x}_j) \in \mathrm{TEq}_k([\![m_q(\mathsf{x}_i,\mathsf{x}_j)]\!]^q_{s,h_k})$$
$$m_q(\mathsf{x}_i,\mathsf{x}_j) \in \mathrm{TEq}'_k(\tilde{\mathsf{f}}([\![m_q(\mathsf{x}_i,\mathsf{x}_j)]\!]^q_{s,h_k})).$$

- The path from $s(\mathsf{x}_i)$ to $s(\mathsf{z}^i)$ and the path from $s(\mathsf{x}_j)$ to $s(\mathsf{z}^i)$ are preserved in h_k, whereas the path from $s(\mathsf{z}^i)$ to $[\![m_q(\mathsf{x}_i,\mathsf{x}_j)]\!]^q_{s,h}$ is not preserved in h_k (i.e. at least one of its locations is assigned to the other heap h_{3-k}). Then h_k witnesses (i) and by definition of meet-points, it holds that

$$[\![m_q(\mathsf{x}_i,\mathsf{x}_j)]\!]^q_{s,h_k} = [\![m_q(\mathsf{x}_j,\mathsf{x}_i)]\!]^q_{s,h_k} = [\![m_q(\mathsf{x}_j,\mathsf{x}_i)]\!]^q_{s,h}$$

By **(Paths)**, h'_k also witnesses (i). Then,

$$[\![m_q(\mathsf{x}_i,\mathsf{x}_j)]\!]^q_{s',h'_k} = [\![m_q(\mathsf{x}_j,\mathsf{x}_i)]\!]^q_{s',h'_k} = [\![m_q(\mathsf{x}_j,\mathsf{x}_i)]\!]^q_{s',h'} = \mathsf{f}([\![m_q(\mathsf{x}_j,\mathsf{x}_i)]\!]^q_{s,h}).$$

Then, we conclude that

$$m_q(\mathsf{x}_i,\mathsf{x}_j) \in \mathrm{TEq}_k([\![m_q(\mathsf{x}_j,\mathsf{x}_i)]\!]^q_{s,h})$$
$$m_q(\mathsf{x}_i,\mathsf{x}_j) \in \mathrm{TEq}'_k(\mathsf{f}([\![m_q(\mathsf{x}_j,\mathsf{x}_i)]\!]^q_{s,h})).$$

- The path from $s(\mathsf{x}_i)$ to $s(\mathsf{z}^i)$ or the path from $s(\mathsf{x}_j)$ to $s(\mathsf{z}^i)$ are not preserved in h_k. Then, by **(Paths)**, the same holds for h'_k with respect to (s',h'). By definition of meet-points, both $[\![m_q(\mathsf{x}_i,\mathsf{x}_j)]\!]^q_{s,h_k}$ and $[\![m_q(\mathsf{x}_i,\mathsf{x}_j)]\!]^q_{s',h'_k}$ are not defined.

Case: h witnesses (v). Lastly, suppose that h and h' witness (v). One of the following holds.

- The path from $s(\mathsf{x}_i)$ to $s(\mathsf{z}^i)$ and the path from $s(\mathsf{x}_j)$ to $s(\mathsf{z}^i)$ are both preserved in h_k. Then, depending on whether or not the path from $s(\mathsf{z}^i)$ to $[\![m_q(\mathsf{x}_j,\mathsf{x}^i)]\!]^q_{s,h}$ is also preserved, h'_k witnesses (i) or (v). From **(Paths)**, the same holds for h'_k (where h'_k witnesses (i) iff h_k witnesses (i)). In both cases ((i) and (v)), it holds that $[\![m_q(\mathsf{x}_i,\mathsf{x}_j)]\!]^q_{s,h_k} = [\![m_q(\mathsf{x}_i,\mathsf{x}_j)]\!]^q_{s,h}$ and $[\![m_q(\mathsf{x}_i,\mathsf{x}_j)]\!]^q_{s',h'_k} = [\![m_q(\mathsf{x}_i,\mathsf{x}_j)]\!]^q_{s',h'} = \mathsf{f}([\![m_q(\mathsf{x}_i,\mathsf{x}_j)]\!]^q_{s,h})$. Then,

$$m_q(\mathsf{x}_i,\mathsf{x}_j) \in \mathrm{TEq}_k([\![m_q(\mathsf{x}_i,\mathsf{x}_j)]\!]^q_{s,h})$$
$$m_q(\mathsf{x}_i,\mathsf{x}_j) \in \mathrm{TEq}'_k(\mathsf{f}([\![m_q(\mathsf{x}_i,\mathsf{x}_j)]\!]^q_{s,h})).$$

- The path from $s(\mathsf{x}_i)$ to $s(\mathsf{z}^j)$ and the path from $s(\mathsf{x}_j)$ to $s(\mathsf{z}^j)$ are preserved in h_k, whereas the path from $s(\mathsf{z}^j)$ to $[\![m_q(\mathsf{x}_i,\mathsf{x}_j)]\!]^q_{s,h}$ is not preserved in h_k. Then h_k witnesses (i) and by definition of meet-points, it holds that

$$[\![m_q(\mathsf{x}_i,\mathsf{x}_j)]\!]^q_{s,h_k} = [\![m_q(\mathsf{x}_j,\mathsf{x}_i)]\!]^q_{s,h_k} = [\![m_q(\mathsf{x}_j,\mathsf{x}_i)]\!]^q_{s,h}.$$

By **(Paths)**, the heap h'_k also witnesses (i). Then,

$$[\![m_q(\mathsf{x}_i,\mathsf{x}_j)]\!]^q_{s',h'_k} = [\![m_q(\mathsf{x}_j,\mathsf{x}_i)]\!]^q_{s',h'_k} = [\![m_q(\mathsf{x}_j,\mathsf{x}_i)]\!]^q_{s',h'} = \mathsf{f}([\![m_q(\mathsf{x}_j,\mathsf{x}_i)]\!]^q_{s,h})$$

Then, we conclude that

$$m_q(\mathsf{x}_i,\mathsf{x}_j) \in \mathrm{TEq}_k([\![m_q(\mathsf{x}_j,\mathsf{x}_i)]\!]^q_{s,h})$$
$$m_q(\mathsf{x}_i,\mathsf{x}_j) \in \mathrm{TEq}'_k(\mathsf{f}([\![m_q(\mathsf{x}_j,\mathsf{x}_i)]\!]^q_{s,h})).$$

- The path from $s(\mathsf{x}_i)$ to $s(\mathsf{z}^j)$ and the path from $s(\mathsf{x}_j)$ to $s(\mathsf{z}^j)$ are not preserved in h_k. Then, by **(Paths)**, the same holds for h'_k with respect to (s',h'). By definition of meet-points, both $[\![m_q(\mathsf{x}_i,\mathsf{x}_j)]\!]^q_{s,h_k}$ and $[\![m_q(\mathsf{x}_i,\mathsf{x}_j)]\!]^q_{s',h'_k}$ are undefined.

We conclude that for every $\ell \in \mathrm{V}$, $m_q(\mathsf{x}_i,\mathsf{x}_j) \in \mathrm{TEq}_k(\ell)$ if and only if $m_q(\mathsf{x}_i,\mathsf{x}_j) \in \mathrm{TEq}'_k(\mathsf{f}(\ell))$.

f_k satisfies (A2): We prove that f_k is such that

$$\text{for every } \ell \in \mathrm{V}_k,\ \ell \in \mathrm{Alloc}_k \text{ iff } \mathsf{f}_k(\ell) \in \mathrm{Alloc}'_k.$$

From (a) in the proof that f_k satisfies **(A3)**, we know that for every $\ell \in \mathrm{V}_k$, $\mathsf{f}_k(\ell) = \mathsf{f}(\ell) \in \mathrm{V}'_k$.
($\Rightarrow$) For the left-to-right direction, let $\ell \in \mathrm{V}_k$ such that $\ell \in \mathrm{Alloc}_k$. By definition we have $\ell \in \mathrm{dom}(h_k)$ and therefore $\ell \in \mathrm{dom}(h)$ (from $h_k \sqsubseteq h$). Since $\mathrm{V}_k \subseteq \mathrm{V}$ (Lemma 4.3) we have $\ell \in \mathrm{V}$ and hence $\ell \in \mathrm{Alloc}$. From $(s,h) \approx^q_\alpha (s',h')$ we obtain $\mathsf{f}(\ell) \in \mathrm{Alloc}'$. Hence, from the construction of h'_k done in **(CA)** we have $\mathsf{f}(\ell) \in \mathrm{dom}(h'_k)$. Moreover, since $\mathsf{f}_k(\ell) = \mathsf{f}(\ell) \in \mathrm{V}'_k$, we conclude $\mathsf{f}_k(\ell) \in \mathrm{Alloc}'_k$.
($\Leftarrow$) The right-to-left direction follows by using a similar argument.

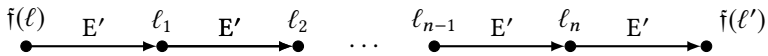
$\mathfrak{f}_k$ **satisfies (A1)**: We prove that $\mathfrak{f}_k$ is a graph isomorphism between $(\mathrm{V}_k, \mathrm{E}_k)$ and $(\mathrm{V}'_k, \mathrm{E}'_k)$. From (a) in the proof that $\mathfrak{f}_k$ satisfies **(A3)** we already know that $\mathfrak{f}_k$ is a bijection from V_k to V'_k. Hence, we only need to show that for all $\ell, \ell' \in \mathrm{V}_k$, $(\ell, \ell') \in \mathrm{E}_k \iff (\mathfrak{f}_k(\ell), \mathfrak{f}_k(\ell')) \in \mathrm{E}'_k$.
($\Rightarrow$) By definition of $\mathrm{SG}_q(s, h_k)$, we have that $(\ell, \ell') \in \mathrm{E}_k$ if and only if $\ell, \ell' \in \mathrm{V}_k$ and h_k witnesses a non-empty (minimal) path from ℓ to ℓ' such that none of the intermediate locations of this path are in V_k. From (a) in the proof that $\mathfrak{f}_k$ satisfies **(A3)** we have

$$\ell \in \mathrm{V}_k \text{ iff } \mathfrak{f}_k(\ell) \in \mathrm{V}'_k; \qquad \ell' \in \mathrm{V}_k \text{ iff } \mathfrak{f}_k(\ell') \in \mathrm{V}'_k.$$

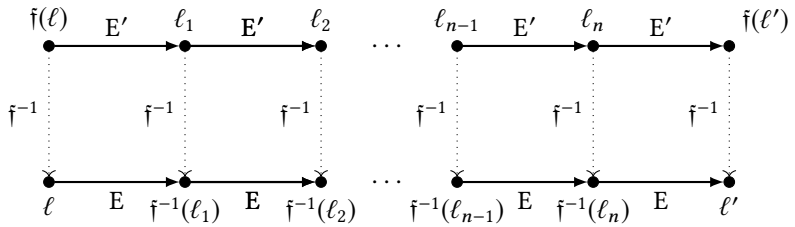
As moreover $\mathrm{V}_k \subseteq \mathrm{V}$ and $\mathrm{V}'_k \subseteq \mathrm{V}'$ (by Lemma 4.3), by **(Paths)** it holds that h_k witnesses a non-empty path from ℓ to ℓ' if and only if h'_k witnesses a non-empty path from $\mathfrak{f}_k(\ell)$ to $\mathfrak{f}_k(\ell')$. Therefore, to conclude the proof, it remains to show that there is a non-empty path from $\mathfrak{f}_k(\ell)$ to $\mathfrak{f}_k(\ell')$ in h'_k such that no intermediate locations of this path are in V'_k. If this property is satisfied, then it is satisfied by the minimal non-empty path from $\mathfrak{f}_k(\ell)$ to $\mathfrak{f}_k(\ell')$, which we suppose being of length $L \geq 1$. Let us denote this path with $\sigma(\mathfrak{f}_k(\ell), \mathfrak{f}_k(\ell'))$. If $L = 1$ then the property is trivially satisfied (as there are no intermediate locations between $\mathfrak{f}_k(\ell)$ and $\mathfrak{f}_k(\ell')$). Otherwise, *ad absurdum*, suppose there exists $\ell'' \in \mathrm{V}'_k$, different from $\mathfrak{f}_k(\ell)$ and $\mathfrak{f}_k(\ell')$, such that for some $L_1, L_2 \geq 1$ such that $L = L_1 + L_2$ we have ${h'_k}^{L_1}(\mathfrak{f}_k(\ell)) = \ell''$ and ${h'_k}^{L_2}(\ell'') = \mathfrak{f}_k(\ell')$. Then, we show that $\mathfrak{f}_k^{-1}(\ell'')$ (which by **(A3)** is in V_k) is an intermediate location on the minimal non-empty path from ℓ to ℓ', in h_k, in contradiction with $(\ell, \ell') \in \mathrm{E}_k$. To do so, we start by considering the set A of locations of V' that belongs to the minimal path from $\mathfrak{f}_k(\ell)$ to $\mathfrak{f}_k(\ell')$ (excluding these two locations). Formally,

$$A \stackrel{\text{def}}{=} \left\{ \tilde{\ell} \in \mathrm{V}' \;\middle|\; \begin{array}{r} \text{there are } L'_1, L'_2 \geq 1, \text{ such that } L = L'_1 + L'_2, \\ {h'_k}^{L'_1}(\mathfrak{f}_k(\ell)) = \tilde{\ell} \text{ and } {h'_k}^{L'_2}(\tilde{\ell}) = \mathfrak{f}_k(\ell') \end{array} \right\}$$

For simplicity, let $A = \{\ell_1, \ldots, \ell_n\}$. Since by Lemma 4.3 we have $\ell'' \in \mathrm{V}'$, we conclude $\ell'' \in A$. Without loss of generality, we assume ℓ_i and ℓ_{i+1} ($i \in [1, n-1]$) to be two consecutive labelled locations in the minimal path from $\mathfrak{f}_k(\ell)$ to $\mathfrak{f}_k(\ell')$, i.e. none of the intermediate locations in the minimal path from ℓ_i to ℓ_{i+1} is labelled. Then, by minimality of $\sigma(\mathfrak{f}_k(\ell), \mathfrak{f}_k(\ell'))$ and from the definition of support graph, $\mathrm{SG}_q(s', h')$ witnesses the following linear structure, where the arrow between two locations ℓ_i and ℓ_{i+1} labelled by E' means that $\mathrm{E}'(\ell_i, \ell_{i+1})$.



Since $\mathfrak{f}$ is a graph isomorphism from (V, E) to $(\mathrm{V}', \mathrm{E}')$ (by **(A1)**), $\mathrm{SG}_q(s, h)$ witnesses a similar structure, as depicted below:



Since $\mathfrak{f}$ is a bijection, for all distinct $i, j \in [1, n]$, we have $\mathfrak{f}^{-1}(\ell_i) \neq \mathfrak{f}^{-1}(\ell_j)$. Thus, every $\mathfrak{f}^{-1}(\ell_i)$ ($i \in [1, n]$) is in the minimal path from ℓ to ℓ' in h, and therefore from $(\ell, \ell') \in \mathrm{E}_k$ we obtain that $\{\mathfrak{f}^{-1}(\ell_1), \ldots, \mathfrak{f}^{-1}(\ell_n)\} \subseteq \mathrm{Inter}_k(\ell, \ell')$. This leads to a contradiction. Indeed, from $\ell'' \in A$ we conclude $\mathfrak{f}^{-1}(\ell'') \in \mathrm{Inter}_k(\ell, \ell')$. Then, as $\mathfrak{f}_k^{-1}(\ell'') \in \mathrm{V}_k$ (by **(A3)**), we conclude that h_k witnesses a labelled intermediate location in the minimal non-empty path from ℓ to ℓ' in

contradiction with $(\ell, \ell') \in \mathrm{E}_k$. Hence, in h'_k there are no labelled locations in the minimal path from $\mathfrak{f}_k(\ell)$ to $\mathfrak{f}_k(\ell')$, allowing us to conclude that $(\mathfrak{f}_k(\ell), \mathfrak{f}_k(\ell')) \in \mathrm{E}'$.
($\Leftarrow$) The right-to-left direction is analogous (thanks to the fact that $\mathfrak{f}$ and $\mathfrak{f}_k$ are bijections).
It remains to check the satisfaction of the conditions **(A4)** and **(A5)** that involve arithmetical constraints.

$\mathfrak{f}_k$ **satisfies (A4)**: We show that for every $(\ell, \ell') \in \mathrm{E}_k$ we have

$$\min(\alpha_k, \mathrm{card}(\mathrm{Inter}_k(\ell, \ell'))) = \min(\alpha_k, \mathrm{card}(\mathrm{Inter}'_k(\mathfrak{f}_k(\ell), \mathfrak{f}_k(\ell')))).$$

First, we show the following intermediate result.

($\mathfrak{f}_k$-A4.I) Let $(\ell, \ell') \in \mathrm{E}_k$. There is a (possibly empty) set $\{\ell_1, \ldots, \ell_n\} \subseteq \mathrm{V}$ such that

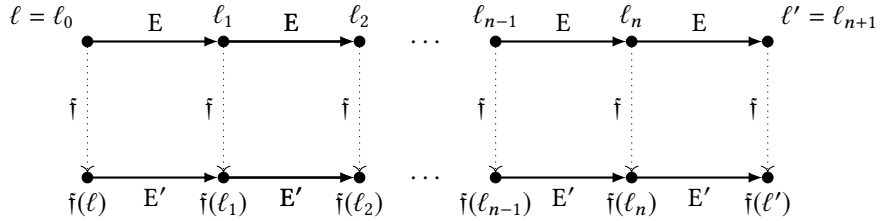
(a) By defining $\ell_0 \stackrel{\text{def}}{=} \ell$ and $\ell_{n+1} \stackrel{\text{def}}{=} \ell'$, we have that for every $i \in [0, n]$, $(\ell_i, \ell_{i+1}) \in \mathrm{E}$;

(b) $\{\ell_1, \ldots, \ell_n\} = \mathrm{Inter}_k(\ell, \ell') \cap \mathrm{V}$ and $\{\mathfrak{f}(\ell_1), \ldots, \mathfrak{f}(\ell_n)\} = \mathrm{Inter}'_k(\mathfrak{f}(\ell), \mathfrak{f}(\ell')) \cap \mathrm{V}'$;

(c) $\mathrm{Inter}_k(\ell, \ell') = \{\ell_1, \ldots, \ell_n\} \cup \bigcup_{i \in [0,n]} \mathrm{Inter}(\ell_i, \ell_{i+1})$;

(d) $\mathrm{Inter}'_k(\mathfrak{f}(\ell), \mathfrak{f}(\ell')) = \{\mathfrak{f}(\ell_1), \ldots, \mathfrak{f}(\ell_n)\} \cup \bigcup_{i \in [0,n]} \mathrm{Inter}'(\mathfrak{f}(\ell_i), \mathfrak{f}(\ell_{i+1}))$.

(Proof of ($\mathfrak{f}_k$-A4.I)) The proof follows rather closely the arguments used for the proof of **(A1)**. Suppose $(\ell, \ell') \in \mathrm{E}_k$. Since $\mathfrak{f}_k$ satisfies **(A1)** we have $(\mathfrak{f}_k(\ell), \mathfrak{f}_k(\ell')) \in \mathrm{E}'_k$. By Lemma 4.3, $\mathrm{V}_k \subseteq \mathrm{V}$ and $\mathrm{V}'_k \subseteq \mathrm{V}'$. Informally, this means that a subheap cannot contain new labelled locations w.r.t. the original heap. It can however be the case that the set $\mathrm{Inter}_k(\ell, \ell')$ contains locations that are labelled w.r.t. (s, h) (i.e. locations in V) and that are not labelled for (s, h_k) (by definition of $\mathrm{Inter}_k(\ell, \ell')$). More precisely, these locations correspond to meet-points of h. Hence, let us consider the set $\{\ell_1, \ldots, \ell_n\} = \mathrm{Inter}_k(\ell, \ell') \cap \mathrm{V}$ (as required by (b)). Let us define $\ell_0 \stackrel{\text{def}}{=} \ell$, $\ell_{n+1} \stackrel{\text{def}}{=} \ell'$. Since $\mathrm{Inter}_k(\ell, \ell')$ describes the set of locations of the minimal path from ℓ to ℓ' in h_k, and moreover $h_k \sqsubseteq h$, there is an ordering on the locations $\ell_1, \ldots, \ell_n$, w.l.o.g. say $\ell_1 < \cdots < \ell_n$ such that for every $i \in [0, n]$ $(\ell_i, \ell_{i+1}) \in \mathrm{E}$ (property (a) of ($\mathfrak{f}_k$-A4.I)). So, $\{\ell_1, \ldots, \ell_n\}$ is a set of locations in $\mathrm{Inter}_k(\ell, \ell')$ that correspond to meet-points of (s, h). Now, as done in the proof of **(A1)**, $\mathfrak{f}$ is a graph isomorphism from (V, E) to $(\mathrm{V}', \mathrm{E}')$ and therefore these two structures witness the following correspondence



(★): where in particular for every $i \in [0, n]$, $(\mathfrak{f}(\ell_i), \mathfrak{f}(\ell_{i+1})) \in \mathrm{E}'$. Notice that, since $\mathfrak{f}$ is a bijection, for all two distinct $i, j \in [1, n]$, we have $\mathfrak{f}(\ell_i) \neq \mathfrak{f}(\ell_j)$. Most importantly, as $\ell, \ell' \notin \mathrm{Inter}_k(\ell, \ell')$ (this holds by definition of this set), we conclude $\ell, \ell' \notin \{\ell_1, \ldots, \ell_n\}$ and therefore $\mathfrak{f}(\ell), \mathfrak{f}(\ell') \notin \{\mathfrak{f}(\ell_1), \ldots, \mathfrak{f}(\ell_n)\}$. Thanks to this property, $\{\mathfrak{f}(\ell_1), \ldots, \mathfrak{f}(\ell_n)\}$ is the set of labelled locations in the minimal path from $\mathfrak{f}(\ell)$ to $\mathfrak{f}(\ell')$ in h'. Equivalently, $\{\mathfrak{f}(\ell_1), \ldots, \mathfrak{f}(\ell_n)\} = \mathrm{Inter}'_k(\mathfrak{f}(\ell), \mathfrak{f}(\ell')) \cap \mathrm{V}'$ (property (b) of ($\mathfrak{f}_k$-A4.I)). Lastly, by (a) and (b), from the fact that heaps are functional and h_k witnesses a path from ℓ to ℓ', it follows that for every $i \in [0, n]$ $\mathrm{Inter}(\ell_i, \ell_{i+1}) \subseteq \mathrm{Inter}_k(\ell, \ell')$. Similarly, for every $i \in [0, n]$ $\mathrm{Inter}'(\mathfrak{f}(\ell_i), \mathfrak{f}(\ell_{i+1})) \subseteq \mathrm{Inter}'_k(\mathfrak{f}(\ell), \mathfrak{f}(\ell'))$. Therefore, we conclude that

- $\mathrm{Inter}_k(\ell, \ell') = \{\ell_1, \ldots, \ell_n\} \cup \bigcup_{i \in [0,n]} \mathrm{Inter}(\ell_i, \ell_{i+1})$,
- $\mathrm{Inter}'_k(\mathfrak{f}(\ell), \mathfrak{f}(\ell')) = \{\mathfrak{f}(\ell_1), \ldots, \mathfrak{f}(\ell_n)\} \cup \bigcup_{i \in [0,n]} \mathrm{Inter}'(\mathfrak{f}(\ell_i), \mathfrak{f}(\ell_{i+1}))$,

which ends the proof of ($\mathfrak{f}_k$-A4.I). Notice that the two properties (c) and (d) (now proved) are well depicted by the figure above, as it shows the structure of the minimal path from ℓ to ℓ' in h (and h_k), and the structure of the minimal path from $\mathfrak{f}(\ell)$ to $\mathfrak{f}(\ell')$ in h' (and h'_k).

We are now ready to show that $\mathfrak{f}_k$ satisfies (A4). Let $(\ell, \ell') \in \mathrm{E}_k$ and let $\{\ell_1, \ldots, \ell_n\} \subseteq \mathrm{V}_k$ satisfying the four properties in ($\mathfrak{f}_k$-A4.I). Let us define $\ell_0 \stackrel{\text{def}}{=} \ell$, $\ell_{n+1} \stackrel{\text{def}}{=} \ell'$. As already stated in the proof of ($\mathfrak{f}_k$-A4.I), $\mathfrak{f}$ is a bijection and therefore for all two distinct $i, j \in [1, n]$, we have $\mathfrak{f}(\ell_i) \neq \mathfrak{f}(\ell_j)$. Hence,

$$\mathrm{card}(\{\ell_1, \ldots, \ell_n\}) = \mathrm{card}(\{\mathfrak{f}(\ell_1), \ldots, \mathfrak{f}(\ell_n)\}) = n.$$

Moreover, by recalling (from Definition 4.4) that

- $\{\mathrm{Alloc}, \mathrm{Rem}\} \cup \{\mathrm{Inter}(\ell, \ell') \mid (\ell, \ell') \in \mathrm{E}\}$ is a partition of $\mathrm{dom}(h)$;
- $\{\mathrm{Alloc}', \mathrm{Rem}'\} \cup \{\mathrm{Inter}'(\ell, \ell') \mid (\ell, \ell') \in \mathrm{E}'\}$ is a partition of $\mathrm{dom}(h')$,

we conclude that for every $i \in [0, n]$,

- $\{\ell_1, \ldots, \ell_n\} \cap \mathrm{Inter}(\ell_i, \ell_{i+1}) = \emptyset$ and for all $j \in [0, n] \setminus \{i\}$, $\mathrm{Inter}(\ell_i, \ell_{i+1}) \cap \mathrm{Inter}(\ell_j, \ell_{j+1}) = \emptyset$;
- $\{\mathfrak{f}(\ell_1), \ldots, \mathfrak{f}(\ell_n)\} \cap \mathrm{Inter}'(\mathfrak{f}(\ell_i), \mathfrak{f}(\ell_{i+1})) = \emptyset$ and for all $j \in [0, n] \setminus \{i\}$, $\mathrm{Inter}'(\mathfrak{f}(\ell_i), \mathfrak{f}(\ell_{i+1})) \cap \mathrm{Inter}'(\mathfrak{f}(\ell_j), \mathfrak{f}(\ell_{j+1})) = \emptyset$.

Thus, the cardinalities of $\mathrm{Inter}_k(\ell, \ell')$ and $\mathrm{Inter}'_k(\mathfrak{f}_k(\ell), \mathfrak{f}_k(\ell'))$ can be written respectively as

$$\mathrm{card}(\mathrm{Inter}_k(\ell, \ell')) = n + \sum_{i \in [0, n]} \mathrm{card}(\mathrm{Inter}(\ell_i, \ell_{i+1})),$$

$$\mathrm{card}(\mathrm{Inter}'_k(\mathfrak{f}_k(\ell), \mathfrak{f}_k(\ell'))) = n + \sum_{i \in [0, n]} \mathrm{card}(\mathrm{Inter}'(\mathfrak{f}(\ell_i), \mathfrak{f}(\ell_{i+1}))).$$

Now, $\mathfrak{f}$ satisfies (A4) w.r.t. the memory states (s, h) and (s', h'), and therefore for every $i \in [0, n]$

$$\min(\alpha, \mathrm{card}(\mathrm{Inter}(\ell_i, \ell_{i+1}))) = \min(\alpha, \mathrm{card}(\mathrm{Inter}'(\mathfrak{f}(\ell_i), \mathfrak{f}(\ell_{i+1})))). \qquad \textbf{($\mathfrak{f}$-A4)}$$

We distinguish two cases.

- If there exists $i \in [0, n]$ such that $\mathrm{card}(\mathrm{Inter}(\ell_i, \ell_{i+1})) \geq \alpha$, then by ($\mathfrak{f}$-A4) we have $\mathrm{card}(\mathrm{Inter}'(\mathfrak{f}(\ell_i), \mathfrak{f}(\ell_{i+1}))) \geq \alpha$. Since $\alpha_k \leq \alpha$, we conclude

$$\mathrm{card}(\mathrm{Inter}_k(\ell, \ell')) \geq \alpha_k; \qquad \mathrm{card}(\mathrm{Inter}'_k(\mathfrak{f}_k(\ell), \mathfrak{f}_k(\ell'))) \geq \alpha_k.$$

- Otherwise (for all $i \in [0, n]$, $\mathrm{card}(\mathrm{Inter}(\ell_i, \ell_{i+1})) < \alpha$) from ($\mathfrak{f}$-A4) we conclude that for all $i \in [0, n]$,

$$\mathrm{card}(\mathrm{Inter}(\ell_i, \ell_{i+1})) = \mathrm{card}(\mathrm{Inter}'(\mathfrak{f}(\ell_i), \mathfrak{f}(\ell_{i+1}))),$$

which allows us to conclude that $\mathrm{card}(\mathrm{Inter}_k(\ell, \ell')) = \mathrm{card}(\mathrm{Inter}'_k(\mathfrak{f}_k(\ell), \mathfrak{f}_k(\ell')))$.

In both cases we have shown that

$$\min(\alpha_k, \mathrm{card}(\mathrm{Inter}_k(\ell, \ell'))) = \min(\alpha_k, \mathrm{card}(\mathrm{Inter}'_k(\mathfrak{f}_k(\ell), \mathfrak{f}_k(\ell')))),$$

concluding the proof of (A4).

$\mathfrak{f}_k$ **satisfies (A5)**: Lastly, we show that

$$\min(\alpha_k, \mathrm{card}(\mathrm{Rem}_k)) = \min(\alpha_k, \mathrm{card}(\mathrm{Rem}'_k)).$$

We take advantage of the following five intermediate results.

(I) For every $\ell \in \mathrm{V} \cap \mathrm{dom}(h_k)$, $\ell \in \mathrm{Rem}_k$ if and only if $\mathfrak{f}(\ell) \in \mathrm{Rem}'_k$.

(Proof of (I)) ($\Rightarrow$) Suppose $\ell \in \mathrm{V}$. By Definition 4.4, it holds that $\ell \in \mathrm{Rem}_k$ if and only if

(i) $\ell \notin \mathrm{V}_k$ and $\ell \in \mathrm{dom}(h_k)$;

(ii) there is no $(\ell', \ell'') \in \mathrm{E}_k$ such that $\ell \in \mathrm{Inter}_k(\ell', \ell'')$.

From (a) in the proof that $\mathfrak{f}_k$ satisfies **(A3)**, we have $\mathfrak{f}(\ell) = \mathfrak{f}_k(\ell) \notin \mathrm{V}'_k$. By $\ell \in \mathrm{dom}(h_k)$ and $\ell \in \mathrm{Rem}_k$, from the construction step **(CA)** we conclude $\mathfrak{f}(\ell) \in \mathrm{dom}(h'_k)$.
Besides, it cannot be that there is $(\ell_1, \ell_2) \in \mathrm{E}'_k$ such that $\mathfrak{f}(\ell) \in \mathrm{Inter}'_k(\ell_1, \ell_2)$. Indeed, if that was the case, we can apply the proof ad absurdum showed in the left-to-right direction of the proof that $\mathfrak{f}_k$ satisfies **(A1)**, and derive that $\ell \in \mathrm{Inter}'_k(\mathfrak{f}_k^{-1}(\ell_1), \mathfrak{f}_k^{-1}(\ell_2))$, in contradiction with (ii). Hence, as we proved that (i) and (ii) hold for $\mathfrak{f}(\ell)$, by Definition 4.4, we conclude that $\mathfrak{f}(\ell) \in \mathrm{Rem}'_k$.
($\Leftarrow$) The right-to-left direction follows by using similar arguments.

(II) $\mathrm{card}(\mathrm{Rem}_k \cap \mathrm{V}) = \mathrm{card}(\mathrm{Rem}'_k \cap \mathrm{V}')$ and $\mathrm{card}(\mathrm{Rem}_k \cap \mathrm{Alloc}) = \mathrm{card}(\mathrm{Rem}'_k \cap \mathrm{Alloc}')$.

(Proof of **(II)**) The first equality follows directly from **(I)** and the fact that $\mathfrak{f}$ is a bijection. Then, since $\mathrm{Alloc} \subseteq \mathrm{V}$, $\mathrm{Alloc}' \subseteq \mathrm{V}'$ (by definition) and $\ell \in \mathrm{Alloc} \iff \mathfrak{f}(\ell) \in \mathrm{Alloc}'$ (from the property **(A2)** of $\mathfrak{f}$), the second equality also holds.

(III) For all $(\ell, \ell') \in \mathrm{E}$,

$$\mathrm{Inter}(\ell, \ell') \cap \mathrm{dom}(h_k) \subseteq \mathrm{Rem}_k \text{ if and only if } \mathrm{Inter}'(\mathfrak{f}(\ell), \mathfrak{f}(\ell')) \cap \mathrm{dom}(h'_k) \subseteq \mathrm{Rem}'_k.$$

(Proof of **(III)**) Recall that $(\ell, \ell') \in \mathrm{E}$ implies $(\mathfrak{f}(\ell), \mathfrak{f}(\ell')) \in \mathrm{E}'$. From **(CI.P2)**, we notice that

$$\mathrm{Inter}(\ell, \ell') \cap \mathrm{dom}(h_k) = \emptyset \text{ if and only if } \mathrm{Inter}'(\mathfrak{f}(\ell), \mathfrak{f}(\ell')) \cap \mathrm{dom}(h'_k) = \emptyset,$$

as by definition $L_k = \mathrm{Inter}(\ell, \ell') \cap \mathrm{dom}(h_k)$ and $L'_k = \mathrm{Inter}'(\mathfrak{f}(\ell), \mathfrak{f}(\ell')) \cap \mathrm{dom}(h'_k)$ in the case **(CI)** of the construction (where we consider $\mathrm{Inter}(\ell, \ell')$). Therefore, **(III)** trivially holds when $\mathrm{Inter}(\ell, \ell') \cap \mathrm{dom}(h_k)$ is empty. In the following, we assume $\mathrm{Inter}(\ell, \ell') \cap \mathrm{dom}(h_k)$ and $\mathrm{Inter}'(\mathfrak{f}(\ell), \mathfrak{f}(\ell')) \cap \mathrm{dom}(h'_k)$ to be non-empty.
($\Rightarrow$) Let $(\ell, \ell') \in \mathrm{E}$ and suppose $\mathrm{Inter}(\ell, \ell') \cap \mathrm{dom}(h_k) \subseteq \mathrm{Rem}_k$. Let us consider a location $\tilde{\ell} \in \mathrm{Inter}'(\mathfrak{f}(\ell), \mathfrak{f}(\ell')) \cap \mathrm{dom}(h'_k)$. We show that $\tilde{\ell} \in \mathrm{Rem}'_k$. As shown in the proof of **(I)**, $\tilde{\ell} \in \mathrm{Rem}'_k$ holds if and only if
(i) $\tilde{\ell} \notin \mathrm{V}'_k$ and $\tilde{\ell} \in \mathrm{dom}(h'_k)$;
(ii) there is no $(\tilde{\ell}_1, \tilde{\ell}_2) \in \mathrm{E}'_k$ such that $\tilde{\ell} \in \mathrm{Inter}'_k(\tilde{\ell}_1, \tilde{\ell}_2)$.
The proof of (i) is straightforward: by $\tilde{\ell} \in \mathrm{Inter}'(\mathfrak{f}(\ell), \mathfrak{f}(\ell')) \cap \mathrm{dom}(h'_k)$, we directly conclude that $\tilde{\ell} \in \mathrm{dom}(h'_k)$ (first condition in (i)) and $\tilde{\ell} \in \mathrm{Inter}'(\mathfrak{f}(\ell), \mathfrak{f}(\ell'))$. From the latter membership and by Definition 4.4, we conclude $\tilde{\ell} \notin \mathrm{V}'$. Lastly, by Lemma 4.3, $\mathrm{V}'_k \subseteq \mathrm{V}'$ and therefore $\tilde{\ell} \notin \mathrm{V}'_k$ (second condition in (i)). Let us now prove (ii). *Ad absurdum*, suppose that there is $(\tilde{\ell}_1, \tilde{\ell}_2) \in \mathrm{E}'_k$ such that $\tilde{\ell} \in \mathrm{Inter}'_k(\tilde{\ell}_1, \tilde{\ell}_2)$. Then, as we have shown that $\mathfrak{f}_k$ satisfies **(A1)**, $(\mathfrak{f}_k^{-1}(\tilde{\ell}_1), \mathfrak{f}_k^{-1}(\tilde{\ell}_n)) \in \mathrm{E}_k$. We apply **($\mathfrak{f}_k$-A4.I)**: there is a set $\{\ell_1, \ldots, \ell_n\} \subseteq \mathrm{V}$ such that

(a) by defining $\ell_0 \stackrel{\text{def}}{=} \mathfrak{f}_k^{-1}(\tilde{\ell}_1)$ and $\ell_{n+1} \stackrel{\text{def}}{=} \mathfrak{f}_k^{-1}(\tilde{\ell}_2)$, for every $i \in [0, n]$, $(\ell_i, \ell_{i+1}) \in \mathrm{E}$;

(b) $\{\ell_1, \ldots, \ell_n\} = \mathrm{Inter}_k(\mathfrak{f}_k^{-1}(\tilde{\ell}_1), \mathfrak{f}_k^{-1}(\tilde{\ell}_2)) \cap \mathrm{V}$ and $\{\mathfrak{f}(\ell_1), \ldots, \mathfrak{f}(\ell_n)\} = \mathrm{Inter}'_k(\tilde{\ell}_1, \tilde{\ell}_2) \cap \mathrm{V}'$;

(c) $\mathrm{Inter}_k(\mathfrak{f}_k^{-1}(\tilde{\ell}_1), \mathfrak{f}_k^{-1}(\tilde{\ell}_2)) = \{\ell_1, \ldots, \ell_n\} \cup \bigcup_{i \in [0,n]} \mathrm{Inter}(\ell_i, \ell_{i+1})$;

(d) $\mathrm{Inter}'_k(\tilde{\ell}_1, \tilde{\ell}_2) = \{\mathfrak{f}(\ell_1), \ldots, \mathfrak{f}(\ell_n)\} \cup \bigcup_{i \in [0,n]} \mathrm{Inter}'(\mathfrak{f}(\ell_i), \mathfrak{f}(\ell_{i+1}))$.

Recalling that $\{\mathrm{Alloc}', \mathrm{Rem}'\} \cup \{\mathrm{Inter}'(\ell, \ell') \mid (\ell, \ell') \in \mathrm{E}'\}$ is a partition of $\mathrm{dom}(h')$ (by Definition 4.4), from (d) together with $\tilde{\ell} \in \mathrm{Inter}'(\mathfrak{f}(\ell), \mathfrak{f}(\ell'))$ and $\tilde{\ell} \in \mathrm{Inter}'_k(\tilde{\ell}_1, \tilde{\ell}_2)$, we conclude that there is $i \in [0, n]$ such that $(\mathfrak{f}(\ell), \mathfrak{f}(\ell')) = (\mathfrak{f}(\ell_i), \mathfrak{f}(\ell_{i+1}))$. From (c), we conclude that $\mathrm{Inter}(\ell, \ell') \subseteq \mathrm{Inter}_k(\mathfrak{f}_k^{-1}(\tilde{\ell}_1), \mathfrak{f}_k^{-1}(\tilde{\ell}_2))$. Notice that, by Definition 4.4, this inclusion also entails $\mathrm{Inter}(\ell, \ell') \cap \mathrm{dom}(h_k) = \mathrm{Inter}(\ell, \ell')$, i.e. every element in $\mathrm{Inter}(\ell, \ell')$ is a memory cell of h_k. Hence, $\mathrm{Inter}(\ell, \ell') \cap \mathrm{dom}(h_k) \subseteq \mathrm{Inter}_k(\mathfrak{f}_k^{-1}(\tilde{\ell}_1), \mathfrak{f}_k^{-1}(\tilde{\ell}_2))$, in contradiction with $\mathrm{Inter}(\ell, \ell') \cap \mathrm{dom}(h_k) \subseteq \mathrm{Rem}_k$. Indeed, we assumed $\mathrm{Inter}(\ell, \ell') \cap \mathrm{dom}(h_k)$ to be non-empty,

and by Definition 4.4, $\mathrm{Rem}_k \cap \mathrm{Inter}_k(\tilde{\mathfrak{f}}_k^{-1}(\tilde{\ell}_1), \tilde{\mathfrak{f}}_k^{-1}(\tilde{\ell}_2)) = \emptyset$. Therefore, it cannot be that there is $(\tilde{\ell}_1, \tilde{\ell}_2) \in \mathrm{E}'_k$ such that $\tilde{\ell} \in \mathrm{Inter}'_k(\tilde{\ell}_1, \tilde{\ell}_2)$. As (i) and (ii) hold, we conclude that $\tilde{\ell} \in \mathrm{Rem}'_k$. ($\Leftarrow$) The right-to-left direction follows by using similar arguments.

(IV) For every $(\ell, \ell') \in \mathrm{E}$, $\mathrm{Rem}_k \cap \mathrm{Inter}(\ell, \ell') = \emptyset$ or $\mathrm{Inter}(\ell, \ell') \cap \mathrm{dom}(h_k) \subseteq \mathrm{Rem}_k$. Similarly, for every $(\ell, \ell') \in \mathrm{E}'$, $\mathrm{Rem}'_k \cap \mathrm{Inter}'(\ell, \ell') = \emptyset$ or $\mathrm{Inter}'(\ell, \ell') \cap \mathrm{dom}(h'_k) \subseteq \mathrm{Rem}'_k$.

(Proof of **(IV)**) Let us prove the first statement (the second one is proven analogously). By Definition 4.4, we have

(a) $\{\mathrm{Alloc}, \mathrm{Rem}\} \cup \{\mathrm{Inter}(\ell, \ell') \mid (\ell, \ell') \in \mathrm{E}\}$ is a partition of $\mathrm{dom}(h)$;

(b) $\{\mathrm{Alloc}_k, \mathrm{Rem}_k\} \cup \{\mathrm{Inter}_k(\ell, \ell') \mid (\ell, \ell') \in \mathrm{E}_k\}$ is a partition of $\mathrm{dom}(h_k)$.

Let $(\ell, \ell') \in \mathrm{E}$. *Ad absurdum*, suppose that $\mathrm{Rem}_k \cap \mathrm{Inter}(\ell, \ell') \neq \emptyset$ and there is $\tilde{\ell} \in (\mathrm{Inter}(\ell, \ell') \cap \mathrm{dom}(h_k)) \setminus \mathrm{Rem}_k$. Since $\tilde{\ell} \in \mathrm{Inter}(\ell, \ell')$, by Definition 4.4, $\tilde{\ell}$ is not a labelled location w.r.t. (s, h), i.e. $\tilde{\ell} \notin \mathrm{V}$. Moreover, from $\tilde{\ell} \in \mathrm{dom}(h_k)$ and $\tilde{\ell} \notin \mathrm{Rem}_k$, by (b) we conclude that $\tilde{\ell} \in \mathrm{Alloc}_k$ or there is $(\ell'_1, \ell'_2) \in \mathrm{E}_k$ such that $\tilde{\ell} \in \mathrm{Inter}_k(\ell'_1, \ell'_2)$.

- If $\tilde{\ell} \in \mathrm{Alloc}_k$, then $\tilde{\ell} \in \mathrm{V}_k$ (by Definition 4.4). Recall that, by Lemma 4.3, $\mathrm{V}_k \subseteq \mathrm{V}$. Therefore, we conclude $\tilde{\ell} \in \mathrm{V}$ (contradiction).
- Otherwise, there is $(\ell'_1, \ell'_2) \in \mathrm{E}_k$ such that $\tilde{\ell} \in \mathrm{Inter}_k(\ell'_1, \ell'_2)$. By (**$\tilde{\mathfrak{f}}_k$-A4.I**), there is a set $\{\ell_1, \ldots, \ell_n\} \subseteq \mathrm{V}$ such that

  (c) by defining $\ell_0 \stackrel{\text{def}}{=} \ell'_1$ and $\ell_{n+1} \stackrel{\text{def}}{=} \ell'_2$, we have that for every $i \in [0, n]$, $(\ell_i, \ell_{i+1}) \in \mathrm{E}$;

  (d) $\{\ell_1, \ldots, \ell_n\} = \mathrm{Inter}_k(\ell'_1, \ell'_2) \cap \mathrm{V}$;

  (e) $\mathrm{Inter}_k(\ell'_1, \ell'_2) = \{\ell_1, \ldots, \ell_n\} \cup \bigcup_{i \in [0,n]} \mathrm{Inter}(\ell_i, \ell_{i+1})$.

  Since $\tilde{\ell} \notin \mathrm{V}$, from (e) we conclude that there is $i \in [0, n]$ such that $\tilde{\ell} \in \mathrm{Inter}(\ell_i, \ell_{i+1})$. Then, by (a) and $\tilde{\ell} \in \mathrm{Inter}(\ell, \ell')$, it must be that $(\ell_i, \ell_{i+1}) = (\ell, \ell')$. Hence, from (e), $\mathrm{Inter}(\ell, \ell') \subseteq \mathrm{Inter}_k(\ell'_1, \ell'_2)$. By (b) $\mathrm{Rem}_k \cap \mathrm{Inter}_k(\ell'_1, \ell'_2) = \emptyset$ and therefore we conclude that $\mathrm{Rem}_k \cap \mathrm{Inter}(\ell, \ell') = \emptyset$, in contradiction with the hypothesis $\mathrm{Rem}_k \cap \mathrm{Inter}(\ell, \ell') \neq \emptyset$.

As in both cases we reached a contradiction, it must be that $\mathrm{Rem}_k \cap \mathrm{Inter}(\ell, \ell') = \emptyset$ or $\mathrm{Inter}(\ell, \ell') \cap \mathrm{dom}(h_k) \subseteq \mathrm{Rem}_k$, concluding the proof.

(V) It holds that $\mathrm{Rem} \cap \mathrm{dom}(h_k) \subseteq \mathrm{Rem}_k$ and $\mathrm{Rem}' \cap \mathrm{dom}(h'_k) \subseteq \mathrm{Rem}'_k$.

(Proof of **(V)**) The proof is rather immediate. Let us prove that $\mathrm{Rem} \cap \mathrm{dom}(h_k) \subseteq \mathrm{Rem}_k$ (the proof of $\mathrm{Rem}' \cap \mathrm{dom}(h'_k) \subseteq \mathrm{Rem}'_k$ is analogous). By Definition 4.4, we have

(a) $\{\mathrm{Alloc}, \mathrm{Rem}\} \cup \{\mathrm{Inter}(\ell, \ell') \mid (\ell, \ell') \in \mathrm{E}\}$ is a partition of $\mathrm{dom}(h)$;

(b) $\{\mathrm{Alloc}_k, \mathrm{Rem}_k\} \cup \{\mathrm{Inter}_k(\ell, \ell') \mid (\ell, \ell') \in \mathrm{E}_k\}$ is a partition of $\mathrm{dom}(h_k)$.

Suppose $\ell \in \mathrm{Rem} \cap \mathrm{dom}(h_k)$. *Ad absurdum*, suppose $\ell \notin \mathrm{Rem}_k$. Then, as $\ell \in \mathrm{dom}(h_k)$, from (b) it holds that $\ell \in \mathrm{Alloc}_k$ or there is $(\ell_1, \ell_2) \in \mathrm{E}_k$ such that $\ell \in \mathrm{Inter}_k(\ell_1, \ell_2)$.

- If $\ell \in \mathrm{Alloc}_k$ then $\ell \in \mathrm{V}_k$ (by Definition 4.4). By Lemma 4.3, $\mathrm{V}_k \subseteq \mathrm{V}$ and therefore $\ell \in \mathrm{V}$. Together with $h_k \sqsubseteq h$ (hence $\mathrm{dom}(h_k) \subseteq \mathrm{dom}(h)$), the fact that $\ell \in \mathrm{V}$ implies $\ell \in \mathrm{Alloc}$ (again by Definition 4.4). From (a) we then obtain $\ell \notin \mathrm{Rem}$, a contradiction.
- Otherwise, let $(\ell_1, \ell_2) \in \mathrm{E}_k$ such that $\ell \in \mathrm{Inter}_k(\ell_1, \ell_2)$. Therefore, by Definition 4.4 there are $L_1, L_2 \geq 1$ such that $h_k^{L_1}(\ell_1) = \ell$ and $h_k^{L_2}(\ell) = \ell_2$. Since $h_k \sqsubseteq h$, we conclude that there are $L_1, L_2 \geq 1$ such that $h^{L_1}(\ell_1) = \ell$ and $h^{L_2}(\ell) = \ell_2$. Now, notice that $(\ell_1, \ell_2) \in \mathrm{E}_k$ implies $\ell_1, \ell_2 \in \mathrm{V}_k$ (by Definition 4.4) and therefore $\ell_1, \ell_2 \in \mathrm{V}$ (since by Lemma 4.3 $\mathrm{V}_k \subseteq \mathrm{V}$). We conclude that ℓ is an intermediate location in the path of h from the labelled location ℓ_1 to the labelled location ℓ_2. Hence, either $\ell \in \mathrm{Alloc}$ or, by Definition 4.4, there is $(\tilde{\ell}_1, \tilde{\ell}_2) \in \mathrm{E}$ such that $\ell \in \mathrm{Inter}(\tilde{\ell}_1, \tilde{\ell}_2)$. In both cases, by (a) we have $\ell \notin \mathrm{Rem}$, a contradiction.

In both cases we conclude that $\ell \notin \mathrm{Rem}_k$ cannot hold, ending the proof of **(V)**.

By taking advantage of **(I)**–**(V)**, we are now ready to show that $\mathfrak{f}_k$ satisfies **(A5)**, i.e.

$$\min(\alpha_k, \mathrm{card}(\mathrm{Rem}_k)) = \min(\alpha_k, \mathrm{card}(\mathrm{Rem}'_k)).$$

First of all, we recall that by Definition 4.4 we have
(a) $\{\mathrm{Alloc}, \mathrm{Rem}\} \cup \{\mathrm{Inter}(\ell, \ell') \mid (\ell, \ell') \in \mathrm{E}\}$ is a partition of $\mathrm{dom}(h)$;
(b) $\{\mathrm{Alloc}', \mathrm{Rem}'\} \cup \{\mathrm{Inter}'(\ell, \ell') \mid (\ell, \ell') \in \mathrm{E}'\}$ is a partition of $\mathrm{dom}(h')$.
Since $\mathrm{Rem}_k \subseteq \mathrm{dom}(h_k) \subseteq \mathrm{dom}(h)$ and $\mathrm{Rem}'_k \subseteq \mathrm{dom}(h'_k) \subseteq \mathrm{dom}(h')$, we can use (a) and (b) to decompose Rem_k and Rem'_k as follows:

- $\mathrm{Rem}_k = (\mathrm{Rem}_k \cap \mathrm{Alloc}) \cup (\mathrm{Rem}_k \cap \mathrm{Rem}) \cup \bigcup_{(\ell,\ell') \in \mathrm{E}} \big(\mathrm{Rem}_k \cap \mathrm{Inter}(\ell, \ell')\big)$,
- $\mathrm{Rem}'_k = (\mathrm{Rem}'_k \cap \mathrm{Alloc}') \cup (\mathrm{Rem}'_k \cap \mathrm{Rem}') \cup \bigcup_{(\ell,\ell') \in \mathrm{E}'} \big(\mathrm{Rem}'_k \cap \mathrm{Inter}'(\ell, \ell')\big)$,

where all unions are performed on disjoint sets. By **(IV)**, $\bigcup_{(\ell,\ell') \in \mathrm{E}} (\mathrm{Rem}_k \cap \mathrm{Inter}(\ell, \ell'))$ and $\bigcup_{(\ell,\ell') \in \mathrm{E}'} (\mathrm{Rem}'_k \cap \mathrm{Inter}'(\ell, \ell'))$ are respectively equal to

$$\bigcup_{\substack{(\ell,\ell') \in \mathrm{E} \\ \mathrm{Inter}(\ell,\ell') \cap \mathrm{dom}(h_k) \subseteq \mathrm{Rem}_k}} \big(\mathrm{Inter}(\ell, \ell') \cap \mathrm{dom}(h_k)\big) \qquad \bigcup_{\substack{(\ell,\ell') \in \mathrm{E}' \\ \mathrm{Inter}'(\ell,\ell') \cap \mathrm{dom}(h'_k) \subseteq \mathrm{Rem}'_k}} \big(\mathrm{Inter}'(\ell, \ell') \cap \mathrm{dom}(h'_k)\big)$$

Since $\mathrm{Rem}_k \subseteq \mathrm{dom}(h_k)$ and $\mathrm{Rem}'_k \subseteq \mathrm{dom}(h'_k)$, by **(V)** we have $\mathrm{Rem}_k \cap \mathrm{Rem} = \mathrm{Rem} \cap \mathrm{dom}(h_k)$ and $\mathrm{Rem}'_k \cap \mathrm{Rem}' = \mathrm{Rem}' \cap \mathrm{dom}(h'_k)$. Hence, the previous equalities can be rewritten as

$$\mathrm{Rem}_k = (\mathrm{Rem}_k \cap \mathrm{Alloc}) \cup (\mathrm{Rem} \cap \mathrm{dom}(h_k)) \cup \bigcup_{\substack{(\ell,\ell') \in \mathrm{E} \\ \mathrm{Inter}(\ell,\ell') \cap \mathrm{dom}(h_k) \subseteq \mathrm{Rem}_k}} \big(\mathrm{Inter}(\ell, \ell') \cap \mathrm{dom}(h_k)\big)$$

$$\mathrm{Rem}'_k = (\mathrm{Rem}'_k \cap \mathrm{Alloc}') \cup (\mathrm{Rem}' \cap \mathrm{dom}(h'_k)) \cup \bigcup_{\substack{(\ell,\ell') \in \mathrm{E}' \\ \mathrm{Inter}'(\ell,\ell') \cap \mathrm{dom}(h'_k) \subseteq \mathrm{Rem}'_k}} \big(\mathrm{Inter}'(\ell, \ell') \cap \mathrm{dom}(h'_k)\big).$$

Since all unions are performed on disjoint sets (by (a) and (b)), we conclude that $\mathrm{card}(\mathrm{Rem}_k)$ and $\mathrm{card}(\mathrm{Rem}'_k)$ satisfy the following equalities:

$$\mathrm{card}(\mathrm{Rem}_k) = \mathrm{card}(\mathrm{Rem}_k \cap \mathrm{Alloc}) + \mathrm{card}(\mathrm{Rem} \cap \mathrm{dom}(h_k)) + \sum_{\substack{(\ell,\ell') \in \mathrm{E} \\ \mathrm{Inter}(\ell,\ell') \cap \mathrm{dom}(h_k) \subseteq \mathrm{Rem}_k}} \mathrm{card}(\mathrm{Inter}(\ell, \ell') \cap \mathrm{dom}(h_k))$$

$$\mathrm{card}(\mathrm{Rem}'_k) = \mathrm{card}(\mathrm{Rem}'_k \cap \mathrm{Alloc}') + \mathrm{card}(\mathrm{Rem}' \cap \mathrm{dom}(h_k)) + \sum_{\substack{(\ell,\ell') \in \mathrm{E}' \\ \mathrm{Inter}'(\ell,\ell') \cap \mathrm{dom}(h'_k) \subseteq \mathrm{Rem}'_k}} \mathrm{card}(\mathrm{Inter}'(\ell, \ell') \cap \mathrm{dom}(h'_k))$$

Now, we are able to compare the values $\min(\alpha_k, \mathrm{card}(\mathrm{Rem}_k))$ and $\min(\alpha_k, \mathrm{card}(\mathrm{Rem}'_k))$. By using the two equalities above and recalling that $\min(A, B + C) = \min(A, B + \min(A, C))$, the following set of equalities holds for $\min(\alpha_k, \mathrm{card}(\mathrm{Rem}_k))$:

$$\begin{aligned}
\min(\alpha_k, \mathrm{card}(\mathrm{Rem}_k)) &= \min(\alpha_k, \mathrm{card}(\mathrm{Rem}_k \cap \mathrm{Alloc}) + R) \\
R &= \min(\alpha_k, \mathrm{card}(\mathrm{Rem} \cap \mathrm{dom}(h_k)) + I) \\
I &= \min(\alpha_k, \sum_{\substack{(\ell,\ell') \in \mathrm{E} \\ \mathrm{Inter}(\ell,\ell') \cap \mathrm{dom}(h_k) \subseteq \mathrm{Rem}_k}} I_{\ell,\ell'}) \\
I_{\ell,\ell'} &= \min(\alpha_k, \mathrm{card}(\mathrm{Inter}(\ell, \ell') \cap \mathrm{dom}(h_k))) \qquad \text{where } (\ell, \ell') \in \mathrm{E}.
\end{aligned}$$

The same can be done for $\min(\alpha_k, \mathrm{card}(\mathrm{Rem}'_k))$:

$$\begin{aligned}
\min(\alpha_k, \mathrm{card}(\mathrm{Rem}'_k)) &= \min(\alpha_k, \mathrm{card}(\mathrm{Rem}'_k \cap \mathrm{Alloc}') + R') \\
R' &= \min(\alpha_k, \mathrm{card}(\mathrm{Rem}' \cap \mathrm{dom}(h'_k)) + I') \\
I' &= \min(\alpha_k, \sum_{\substack{(\ell,\ell')\in \mathrm{E}' \\ \mathrm{Inter}'(\ell,\ell')\cap \mathrm{dom}(h'_k)\subseteq \mathrm{Rem}'_k}} I'_{\ell,\ell'}) \\
I'_{\ell,\ell'} &= \min(\alpha_k, \mathrm{card}(\mathrm{Inter}'(\ell,\ell') \cap \mathrm{dom}(h'_k))) \qquad \text{where } (\ell,\ell') \in \mathrm{E}'.
\end{aligned}$$

By **(CI.P2)**, for every $(\ell, \ell') \in \mathrm{E}$ we have $I_{\ell,\ell'} = I'_{\dagger(\ell),\dagger(\ell')}$. From **(III)** together with the fact that $\dagger$ witnesses a graph isomorphism between (s, h) and (s', h') we conclude that

$$\begin{aligned}
&\{(\ell,\ell') \in \mathrm{E}' \mid \mathrm{Inter}'(\ell,\ell') \cap \mathrm{dom}(h'_k) \subseteq \mathrm{Rem}'_k\} \\
&= \{(\dagger(\ell), \dagger(\ell')) \mid (\ell,\ell') \in \mathrm{E}, \mathrm{Inter}(\ell,\ell') \cap \mathrm{dom}(h_k) \subseteq \mathrm{Rem}_k\},
\end{aligned}$$

which allows us to conclude that $I = I'$. Furthermore, by **(CR.P1)**, it follows also that $R = R'$. Lastly, by **(II)** we conclude:

$$\min(\alpha_k, \mathrm{card}(\mathrm{Rem}_k)) = \min(\alpha_k, \mathrm{card}(\mathrm{Rem}'_k)).$$

This concludes the proof: for the support graphs of h_k and h'_k, $\dagger$ restricted to the domain V_k and the codomain V'_k satisfies all conditions of Lemma 4.7 with respect to α_k. Therefore, it holds that $(s, h_1) \approx^q_{\alpha_1} (s', h'_1)$ and $(s, h_2) \approx^q_{\alpha_2} (s', h'_2)$. □

Received September 2020; accepted January 2021