Extending propositional separation logic for robustness properties

Alessio Mansutti
LSV, CNRS, ENS Paris-Saclay, Université Paris-Saclay, Cachan, France

Abstract

We study an extension of *propositional separation logic* that can specify *robustness properties*, such as acyclicity and garbage freedom, for automatic verification of stateful programs with singly-linked lists. We show that its satisfiability problem is PSpace-complete, whereas modest extensions of the logic are shown to be Tower-hard. As separating implication, reachability predicates (under some syntactical restrictions) and a unique quantified variable are allowed, this logic subsumes several PSpace-complete separation logics considered in previous works.

2012 ACM Subject Classification Theory of computation → Logic → Separation logic

Keywords and phrases Separation logic, decision problems, reachability, logics on trees, interval temporal logic, adjuncts and quantifiers elimination.

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2018.42

1 Introduction

Separation Logic [26] is a well-known assertion logic providing a scalable solution for Hoare-style verification of imperative, heap-manipulating programs [7, 16, 29]. To achieve scalability, separation logic relies in two spatial connectives to represent memory regions: the *separating conjunction* ($*$) and the *separating implication* ($-\!\!*$). These operators allow to express complex properties of stateful programs, making this logic the core assertion language of many tools [2, 3, 6, 14, 15, 18, 20, 23]. To achieve automation, the underlying Hoare-style proof system requires these tools to check for the classical decision problems of satisfiability, validity and entailment. The complexity of these problems have been quite studied:

- PTime algorithms for satisfiability and entailment have been defined for the *symbolic heap fragment* (the core logic of many tools) [9]. This complexity is achieved by removing the separating implication from separation logic and heavily restricting the use of Boolean connectives. Decidability results (the general lower-bound is ExpTime) for these problems are also known when the fragment is enriched with inductive predicates [1, 17, 19, 21].
- PSpace-completeness has been shown for propositional separation logic [8]. Its extension with one quantified variable, denoted with $\mathsf{1SL}(*, -\!\!*)$, was also found to be in PSpace [12]. However, adding a second quantified variable causes the logic to become undecidable [11].
- In absence of the separating implication, PSpace-completeness has also been proved for $\mathsf{SL}(*, \mathtt{ls})$, i.e. the propositional fragment enriched with the list-segment predicate $\mathtt{ls}$. Here, adding the separating implication again leads to undecidability [13].

Besides these decision problems, in program analysis it is crucial to be able to check for *robustness properties* such as *garbage freedom* and *acyclicity* (see Section 2 for precise definitions). A recent work [19] tackles these problems for the symbolic heap fragment with user-defined inductive predicates by introducing the framework of heap automata. Within this framework, both garbage freedom and acyclicity are shown to be ExpTime-complete.

A natural question is how to check the satisfaction of robustness properties for propositional separation logic as they are not expressible in $\mathsf{1SL}(*, -\!\!*)$ nor in $\mathsf{SL}(*, \mathtt{ls})$. Indeed,

38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2018).
Editors: Sumit Ganguly and Paritosh Pandya; Article No. 42; pp. 42:1–42:22

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

it would be nice to capture these problems directly in the logic, without introducing any external framework, to then solve them using procedures for the classical decision problems.

Our contribution. In this paper we address this question by studying an extension of propositional separation logic that captures both $\mathsf{1SL}(*, -\!\!*)$ and $\mathsf{SL}(*, \mathtt{ls})$ and whose expressive power allows to directly reduce the robustness properties to entailment. This logic, herein called $\mathsf{1SL}^{\mathtt{R1}}_{\mathtt{R2}}(*, -\!\!*, \mathtt{reach}^+)$, is defined from $\mathsf{1SL}(*, -\!\!*)$ by adding reachability predicates under some syntactical restrictions (the formal definition is given in Section 2). In Section 4 we show that the satisfiability problem (and hence entailment, validity and robustness properties) of this logic can be decided in PSPACE. As far as we know, this makes $\mathsf{1SL}^{\mathtt{R1}}_{\mathtt{R2}}(*, -\!\!*, \mathtt{reach}^+)$ the largest decidable fragment of separation logic including full Boolean connectives, spatial connectives and reachability predicates and the first one where these predicates can be used in the scope of $-\!\!*$, albeit in a controlled way to retain decidability [13]. To show the PSPACE upper-bound we extend the widely used proof technique of test formulæ introduced in [22].

This complexity result is rather surprising as, besides subsuming the results in [12] and [13], slightly extending the logic entails TOWER-hardness (the complexity class TOWER has been introduced in [28] and sits between the class of elementary problems and the class of primitive-recursive problems). Indeed, in Section 3 we show how weakening the syntactic restrictions on reachability predicates allows the logic to capture a TOWER-complete fragment of Moszkowski's propositional interval temporal logic (PITL) [25]. To better formalise this result we first introduce an alternative semantics for PITL and reduce this logic to an intermediate logic interpreted on trees (ALT). We then consider a modest extension of $\mathsf{1SL}^{\mathtt{R1}}_{\mathtt{R2}}(*, -\!\!*, \mathtt{reach}^+)$ and show that it captures ALT, proving its non-elementary complexity.

2 The separation logic $\mathsf{1SL}(*, -\!\!*, \mathtt{reach}^+)$

Let VAR be a countably infinite set of program variables and let LOC be a countably infinite set of locations. A *memory state* is a pair (s, h) consisting of a variable valuation function (the *store*) $s : \mathsf{VAR} \to \mathsf{LOC}$ and a partial function with finite domain (the *heap*) $h : \mathsf{LOC} \to_{\mathtt{fin}} \mathsf{LOC}$. We denote with $\mathrm{dom}(h)$ the domain of definition of a heap h and with $\mathrm{ran}(h)$ its range. Each element in $\mathrm{dom}(h)$ is understood as a *memory cell* of h. With h^δ we denote $\delta \geq 0$ functional composition(s) of h. Two heaps h_1 and h_2 are said to be disjoint, written $h_1 \perp h_2$, whenever $\mathrm{dom}(h_1) \cap \mathrm{dom}(h_2) = \emptyset$. We define the union $h_1 + h_2$ of h_1 and h_2 as the standard sum of two functions $(h_1 + h_2)(\ell) \stackrel{\text{def}}{=} \textbf{if } \ell \in \mathrm{dom}(h_1) \textbf{ : } h_1(\ell) \textbf{ else } h_2(\ell)$, defined only whenever $h_1 \perp h_2$.

We extend propositional separation logic with reachability predicates and one quantified variable denoted by $\mathtt{u} \notin \mathsf{VAR}$. We call this logic $\mathsf{1SL}(*, -\!\!*, \mathtt{reach}^+)$. Its formulæ φ are from

$$\varphi \coloneqq \mathtt{emp} \mid e_1 = e_2 \mid e_1 \hookrightarrow e_2 \mid \mathtt{reach}^+(e_1, e_2) \mid \varphi \wedge \varphi \mid \neg\varphi \mid \exists \mathtt{u}\ \varphi \mid \varphi * \varphi \mid \varphi -\!\!* \varphi$$

where $e_1, e_2 \in \mathsf{VAR} \cup \{\mathtt{u}\}$. We denote with $\mathsf{fv}(\varphi)$ the set of free variables in φ. $\mathsf{1SL}(*, -\!\!*, \mathtt{reach}^+)$ is interpreted on triples (s, h, l), where (s, h) is a memory state and $l \in \mathsf{LOC}$ is the current assignment of the only quantified variable $\mathtt{u}$. The satisfaction relation $\models$ is defined as follows (standard clauses for $\neg$ and $\wedge$ are omitted throughout the paper)

- $(s, h, l) \models \mathtt{emp}$ if and only if $\mathrm{dom}(h) = \emptyset$.
- $(s, h, l) \models e_1 = e_2$ if and only if $[\![e_1]\!] = [\![e_2]\!]$, with $[\![\mathtt{u}]\!] \stackrel{\text{def}}{=} l$ and $[\![\mathtt{x}]\!] \stackrel{\text{def}}{=} s(\mathtt{x})$ for every $\mathtt{x} \in \mathsf{VAR}$.
- $(s, h, l) \models e_1 \hookrightarrow e_2$ if and only if $s([\![e_1]\!]) = [\![e_2]\!]$.
- $(s, h, l) \models \mathtt{reach}^+(e_1, e_2)$ if and only if there is $\delta \geq 1$ such that $h^\delta([\![e_1]\!]) = [\![e_2]\!]$.
- $(s, h, l) \models \exists \mathtt{u}\ \varphi$ if and only if there is $l' \in \mathsf{LOC}$ such that $(s, h, l') \models \varphi$.
- $(s, h, l) \models \varphi_1 * \varphi_2$ iff $h_1 + h_2 = h$, $(s, h_1, l) \models \varphi_1$ and $(s, h_2, l) \models \varphi_2$, for some h_1 and h_2.
- $(s, h, l) \models \varphi_1 -\!\!* \varphi_2$ iff for all h', if $h' \perp h$ and $(s, h', l) \models \varphi_1$ then $(s, h + h', l) \models \varphi_2$.

Standard connectives $\Rightarrow$, $\Leftrightarrow$, $\vee$ and the universal quantification $\forall$ are derived as usual. We denote with $\top$ the tautology $\mathtt{emp} \vee \neg\mathtt{emp}$ and with $\bot$ its negation. $\mathtt{alloc}(e)$ stands for $e \hookrightarrow e \mathrel{-\!\!*} \bot$, the formula satisfied if and only if $[\![e]\!] \in \mathrm{dom}(h)$. We recursively define the formula $\mathtt{size} \geq \beta$ as $\mathtt{size} \geq 0 \stackrel{\text{def}}{=} \top$ and $\mathtt{size} \geq \beta{+}1 \stackrel{\text{def}}{=} \neg\mathtt{emp} * \mathtt{size} \geq \beta$. $\mathtt{size} \geq \beta$ is satisfied if and only if $\mathrm{card}(\mathrm{dom}(h)) \geq \beta$ (we write $\mathrm{card}(.)$ to denote the cardinality of a set). We write $\mathtt{size} = \beta$ for the formula $\mathtt{size} \geq \beta \wedge \neg\mathtt{size} \geq \beta{+}1$. For a complete description of separation logic we refer the reader to the classical paper by Reynolds [26]. Note that the *heap-precise* predicates defined in [26] can be retrieved in our logic. Indeed, the *points-to* relation $e_1 \mapsto e_2$ can be expressed as $e_1 \hookrightarrow e_2 \wedge \mathtt{size} = 1$ whereas the *list-segment* relation $\mathtt{ls}(e_1, e_2)$ can be defined as $(e_1 = e_2 \wedge \mathtt{emp}) \vee (e_1 \neq e_2 \wedge \mathtt{reach}^+(e_1, e_2) \wedge \neg(\mathtt{size} = 1 * \mathtt{reach}^+(e_1, e_2)))$.

Decision problems and robustness properties. The *satisfiability problem* takes as input a formula φ and asks whether there is a model (s, h, l) such that $(s, h, l) \models \varphi$. The *validity problem* asks whether φ is satisfied by every memory state. Given a second formula ψ, the *entailment problem* $\varphi \models \psi$ asks whether each memory state satisfying φ also satisfies ψ.

As advocated in [19], besides these decision problems, in program analysis we are also interested in the *robustness properties* of *acyclicity* and *garbage freedom*. The acyclicity property asks every model satisfying φ to be acyclic. Instead, garbage freedom holds whenever in every model satisfying φ, each memory cell is reachable from a program variable of $\mathsf{fv}(\varphi)$. In $\mathsf{1SL}(*, \mathrel{-\!\!*}, \mathtt{reach}^+)$ both problems can be reduced to entailment:

- acyclicity requires us to be able to solve $\varphi \models \forall \mathtt{u}\ \neg\mathtt{reach}^+(\mathtt{u}, \mathtt{u})$;
- garbage freedom can be expressed as $\varphi \models \forall \mathtt{u}\ (\mathtt{alloc}(\mathtt{u}) \Rightarrow \bigvee_{\mathtt{x} \in \mathsf{fv}(\varphi)} (\mathtt{reach}^+(\mathtt{x}, \mathtt{u}) \vee \mathtt{x} = \mathtt{u}))$.

As our logic is closed under Boolean connectives, validity and entailment reduce to satisfiability. The main purpose of this paper is then to study the complexity status of the latter problem for fragments of $\mathsf{1SL}(*, \mathrel{-\!\!*}, \mathtt{reach}^+)$ that can still express both robustness properties.

Decidability status and restrictions. As shown in [13], extending propositional separation logic so that it can express bounded reachability up to distance three leads to undecidability. Unfortunately this makes the satisfiability problem of $\mathsf{1SL}(*, \mathrel{-\!\!*}, \mathtt{reach}^+)$ undecidable. Indeed, two-steps reachability between two program variables $\mathtt{x}$ and $\mathtt{y}$ can be expressed as $\exists \mathtt{u}\ (\mathtt{x} \hookrightarrow \mathtt{u} \wedge \mathtt{u} \hookrightarrow \mathtt{y} \wedge \mathtt{u} \neq \mathtt{x} \wedge \mathtt{u} \neq \mathtt{y})$ whereas three-steps reachability is captured with

$$(\mathtt{reach}^+(\mathtt{x}, \mathtt{y}) \wedge \mathtt{size} = 3 \wedge \underbrace{\neg(\mathtt{size} = 1 * \mathtt{reach}^+(\mathtt{x}, \mathtt{y}))}_{\text{isolating any memory cell makes impossible for } s(\mathtt{x}) \text{ to reach } s(\mathtt{y})}) * \top$$

To retain decidability we propose to restrict the logic to those formulæ where each occurrence of $\mathtt{reach}^+(e_1, e_2)$ is constrained so that **(R1)** it is not on the right side of its first $\mathrel{-\!\!*}$ ancestor (seeing the formula as a tree), and **(R2)** if $e_1 = \mathtt{u}$ then $e_2 = \mathtt{u}$. For instance, given two formulæ φ and ψ satisfying these conditions, the formula $\mathtt{reach}^+(\mathtt{u}, \mathtt{x}) * (\varphi \mathrel{-\!\!*} \psi)$ only satisfies R1, both conditions are satisfied in $\varphi \mathrel{-\!\!*} (\mathtt{reach}^+(\mathtt{x}, \mathtt{y}) \mathrel{-\!\!*} \psi)$ whereas the formula $\varphi \mathrel{-\!\!*} (\psi * \mathtt{reach}^+(\mathtt{u}, \mathtt{u}))$ only satisfies R2. A grammar of this logic is given in Section 4, where we show that under these conditions the satisfiability problem of $\mathsf{1SL}(*, \mathrel{-\!\!*}, \mathtt{reach}^+)$ can be decided in PSPACE.

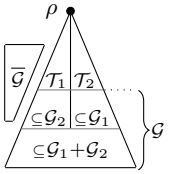
From the results in [13], weakening even slightly R1 seems to be a challenge. Indeed, after enforcing R1 the logic can still freely express two-steps reachability between program variables and it is unable to do the same for paths of length three only in positions of the formula where $\mathtt{reach}^+$ cannot occur. On these positions, even the modest addition of a second quantified variable causes then undecidability (coherently with the results in [11]). We could hope for the satisfiability problem to still be in PSPACE without the R2 condition. In the next section we show that this is not the case: without R2 the problem is TOWER-hard. Nevertheless, under these conditions the logic is still able to express both robustness properties.

3 Tower-hardness of $\mathsf{1SL}(*, -\!\!*, \mathtt{reach}^+)$ under $\mathtt{R1}$

In this section we show that $\mathsf{1SL}_{\mathtt{R1}}(*, -\!\!*, \mathtt{reach}^+)$, i.e. the fragment of $\mathsf{1SL}(*, -\!\!*, \mathtt{reach}^+)$ formulæ satisfying the condition $\mathtt{R1}$, is Tower-hard. To do so, we first introduce an auxiliary logic (ALT) interpreted on trees. At its core, ALT is a simple logic whose formulæ can only split a tree and check whether the only (quantified) variable points to a node in the tree or not. In a way, ALT represents a small subset of the properties expressible in $\mathsf{1SL}_{\mathtt{R1}}(*, -\!\!*, \mathtt{reach}^+)$ that are sufficient to reach Tower-hardness. Indeed, despite its simplicity, we show that ALT is Tower-complete. In particular, the hardness proof is achieved by reduction of Moszkowski's propositional interval temporal logic [25] with locality principle (PITL). This reduction is done by first defining an alternative semantics for PITL based on marked words.

3.1 An auxiliary logic on trees (ALT)

To easily relate ALT with separation logic, finite trees are here defined using heaps encoding the parent relation. We assume a location $\rho \in \mathsf{LOC}$ as the root of all trees. A heap $\mathcal{T}$ is a *tree* whenever $\rho \notin \mathrm{dom}(\mathcal{T})$ and for each $\ell \in \mathrm{dom}(\mathcal{T})$ there is $\delta \geq 1$ such that $\mathcal{T}^\delta(\ell) = \rho$. Then, ℓ is a *descendant* (resp. *child*) of $\ell' \in \mathrm{ran}(\mathcal{T})$ whenever there is $\delta \geq 1$ (resp. $\delta = 1$) such that $\mathcal{T}^\delta(\ell) = \ell'$. It is straightforward to see that these definitions are equivalent to the classical ones. As formally introduced below, ALT formulæ are able to chop a tree, preventing some memory cells to reach ρ. These locations form a heap $\mathcal{G}$, called *garbage*, such that $\rho \notin \mathrm{dom}(\mathcal{G}) \cup \mathrm{ran}(\mathcal{G})$. We denote with $\mathbb{T}$ the domain of pairs $(\mathcal{T}, \mathcal{G})$ where $\mathcal{T}$ and $\mathcal{G}$ are respectively a tree and a garbage such that $\mathrm{dom}(\mathcal{T}) \cap (\mathrm{dom}(\mathcal{G}) \cup \mathrm{ran}(\mathcal{G})) = \emptyset$. The notions of disjointness ($\perp$) and union of disjoint heaps ($+$) are naturally extended to elements of $\mathbb{T}$. $(\mathcal{T}_1, \mathcal{G}_1)$ and $(\mathcal{T}_2, \mathcal{G}_2)$ are disjoint whenever $(\mathcal{T}_1 + \mathcal{G}_1) \perp (\mathcal{T}_2 + \mathcal{G}_2)$. If they are disjoint then their union $(\mathcal{T}_1, \mathcal{G}_1) + (\mathcal{T}_2, \mathcal{G}_2)$ (see picture on the left) is the pair $(\mathcal{T}_1 + \mathcal{T}_2 + \mathcal{G}, \overline{\mathcal{G}}) \in \mathbb{T}$ such that $\mathcal{G} + \overline{\mathcal{G}} = \mathcal{G}_1 + \mathcal{G}_2$ and $\mathrm{dom}(\mathcal{G}) = \{\ell \in \mathrm{dom}(\mathcal{G}_1 + \mathcal{G}_2) \mid (\mathcal{G}_1 + \mathcal{G}_2)^\delta(\ell) \in \mathrm{dom}(\mathcal{T}_1 + \mathcal{T}_2) \text{ for some } \delta \geq 1\}$. ALT-formulæ φ are built from



$$\varphi \coloneqq \mathsf{T}(\mathtt{u}) \mid \mathsf{G}(\mathtt{u}) \mid \varphi \wedge \varphi \mid \neg\varphi \mid \exists \mathtt{u}\ \varphi \mid \varphi * \varphi$$

and interpreted on *states* $(\mathcal{T}, \mathcal{G}, l)$ where $(\mathcal{T}, \mathcal{G}) \in \mathbb{T}$ and $l \in \mathsf{LOC} \setminus \{\rho\}$ is the current assignment of $\mathtt{u}$. The satisfaction relation $\models$ is defined as follows:

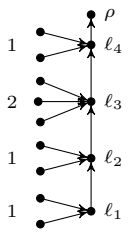
- $(\mathcal{T}, \mathcal{G}, l) \models \mathsf{T}(\mathtt{u})$ if only if $l \in \mathrm{dom}(\mathcal{T})$.
- $(\mathcal{T}, \mathcal{G}, l) \models \mathsf{G}(\mathtt{u})$ if and only if $l \in \mathrm{dom}(\mathcal{G})$.
- $(\mathcal{T}, \mathcal{G}, l) \models \exists \mathtt{u}\ \varphi$ if and only if there is $l' \in \mathsf{LOC} \setminus \{\rho\}$ such that $(\mathcal{T}, \mathcal{G}, l') \models \varphi$.
- $(\mathcal{T}, \mathcal{G}, l) \models \varphi_1 * \varphi_2$ iff $(\mathcal{T}_1, \mathcal{G}_1, l) \models \varphi_1$ and $(\mathcal{T}_2, \mathcal{G}_2, l) \models \varphi_2$ for some $(\mathcal{T}_1, \mathcal{G}_1) + (\mathcal{T}_2, \mathcal{G}_2) = (\mathcal{T}, \mathcal{G})$.

The tautology $\top$ is defined as $\mathsf{T}(\mathtt{u}) \vee \neg\mathsf{T}(\mathtt{u})$. $\mathtt{alloc}(\mathtt{u}) \stackrel{\text{def}}{=} \mathsf{T}(\mathtt{u}) \vee \mathsf{G}(\mathtt{u})$ is the formula satisfied if and only if $l \in \mathrm{dom}(\mathcal{T} + \mathcal{G})$. The size $|\varphi|$ of a formula φ is defined as: 1 for the atomic predicates $\mathsf{T}(\mathtt{u})$ and $\mathsf{G}(\mathtt{u})$, $|\varphi_1 \wedge \varphi_2| \stackrel{\text{def}}{=} \max(|\varphi_1|, |\varphi_2|)$, $|\neg\varphi| \stackrel{\text{def}}{=} |\exists \mathtt{u}\ \varphi| \stackrel{\text{def}}{=} |\varphi|$ and $|\varphi_1 * \varphi_2| \stackrel{\text{def}}{=} |\varphi_1| + |\varphi_2|$.

Notice how the $*$ operator splits the model similarly to the separating conjunction in separation logic. In Section 3.3 we explore the similarities between these two logics by providing the formal translation from ALT to $\mathsf{1SL}_{\mathtt{R1}}(*, -\!\!*, \mathtt{reach}^+)$. ALT is also reminiscent of static ambient logic [5, 22], where the *composition* operator $\varphi \mid \psi$ cuts the tree into two parts. However, differently from the $*$ operator of ALT, this operation preserves the parent relation. Then, $\varphi \mid \psi$ holds on trees that can be divided into a tree satisfying φ, one satisfying ψ and no garbage locations are generated by the split. Given a model $(\mathcal{T}, \mathcal{G}) \in \mathbb{T}$ where $\mathcal{G} = \emptyset$, this semantics can be retrieved in ALT with the formula $(\varphi \wedge \neg\exists \mathtt{u}\ \mathsf{G}(\mathtt{u})) * (\psi \wedge \neg\exists \mathtt{u}\ \mathsf{G}(\mathtt{u}))$.

Expressive power: encoding words in ALT. Despite using one single variable, the ability of splitting the model with a operator having the semantics of the separating conjunction greatly increases the expressive power of ALT. In particular, we show that ALT is able to characterise finite words. We first establish a correspondence between words and trees of a particular shape. Let $\Sigma = [1, n]$ be the alphabet of natural numbers between 1 and n. Let $\mathfrak{w} = \mathtt{a}_1 \cdots \mathtt{a}_k$ be a k-letters word in Σ^* and $\{\ell_1, \cdots, \ell_k\}$ be a set of k locations. For every $i \in [1, k]$, let $\mathsf{L}(i)$ be a set of $\mathtt{a}_i + 1$ locations different from $\ell_1, \cdots, \ell_k$ and so that for each distinct $i, j \in [1, k]$, $\mathsf{L}(i) \cap \mathsf{L}(j) = \emptyset$. An encoding of $\mathfrak{w}$ is a tree $\mathcal{T}$ defined on these sets as

$$\mathcal{T}(\ell_k) \stackrel{\text{def}}{=} \rho; \quad \mathcal{T}(\ell_i) \stackrel{\text{def}}{=} \ell_{i+1} \text{ for each } i \in [1, k-1]; \quad \mathcal{T}(\ell) \stackrel{\text{def}}{=} \ell_i \text{ for each } i \in [1, k] \text{ and } \ell \in \mathsf{L}(i)$$



The locations $\ell_1, \cdots, \ell_k$ are the *main path* of $\mathcal{T}$, where $\mathcal{T}(\ell_i) = \ell_{i+1}$ for $i \in [1, k-1]$, and $\mathcal{T}(\ell_k) = \rho$. These are the only locations in $\mathrm{dom}(\mathcal{T})$ with at least one child, with ℓ_1 being the only location with the same number of descendants and children. We say that a location $\ell \in \mathrm{dom}(\mathcal{T})$ encodes the symbol $\mathtt{a} \in \Sigma$ if it has exactly $\mathtt{a} + 1$ children that are not in the main path. Then the locations of the main path of $\mathcal{T}$ are the only ones encoding symbols, where ℓ_i encodes $\mathtt{a}_i$ for any $i \in [1, k]$. The tree on the left encodes the word 1121. We now show how to capture these trees with a logical formula. As symbols in Σ are represented using the number of children of elements in $\mathcal{T}$, we need a formula that expresses this number for the location assigned to $\mathtt{u}$. We first define $\mathtt{size}_{\mathsf{T}+\mathsf{G}} \geq \beta$ and $\mathtt{size}_{\mathsf{G}} \geq \beta$, two formulæ respectively stating that $\mathrm{dom}(\mathcal{T}+\mathcal{G})$ and $\mathrm{dom}(\mathcal{G})$ have size at least $\beta \in \mathbb{N}$. They are $\top$ for $\beta = 0$ and otherwise

$$\mathtt{size}_{\mathsf{T}+\mathsf{G}} \geq \beta \stackrel{\text{def}}{=} \exists \mathtt{u}\ \mathtt{alloc}(\mathtt{u}) * \mathtt{size}_{\mathsf{T}+\mathsf{G}} \geq \beta - 1$$
$$\mathtt{size}_{\mathsf{G}} \geq \beta \stackrel{\text{def}}{=} \exists \mathtt{u}\ (\mathsf{G}(\mathtt{u}) \wedge \underbrace{((\mathtt{alloc}(\mathtt{u}) \wedge \mathtt{size}_{\mathsf{T}+\mathsf{G}} = 1) * \mathtt{size}_{\mathsf{G}} \geq \beta - 1)}_{\text{by excluding a location in } \mathcal{G}\text{, at least other } \beta - 1 \text{ such locations can be found}})$$

where $\mathtt{size}_{\mathsf{T}+\mathsf{G}} = \beta$ is $\mathtt{size}_{\mathsf{T}+\mathsf{G}} \geq \beta \wedge \neg \mathtt{size}_{\mathsf{T}+\mathsf{G}} \geq \beta + 1$, the formula that checks if $\mathcal{T} + \mathcal{G}$ has exactly β elements. In $\mathtt{size}_{\mathsf{G}} \geq \beta$, notice how the $*$ operator is used to isolate the memory cell of $\mathcal{G}$ corresponding to $\mathtt{u}$ from the remaining part of the model and then search for other $\beta - 1$ elements of $\mathcal{G}$. This "trick" is often used in our formulæ, including the one that checks the number of descendants of a location $l \in \mathrm{dom}(\mathcal{T})$ corresponding to $\mathtt{u}$:

$$\#\mathtt{desc}(\mathtt{u}) \geq \beta \stackrel{\text{def}}{=} \top * (\underbrace{(\forall \mathtt{u}\ \neg \mathsf{G}(\mathtt{u}))}_{\mathcal{G} \text{ is empty}} \wedge \underbrace{((\mathtt{alloc}(\mathtt{u}) \wedge \mathtt{size}_{\mathsf{T}+\mathsf{G}} = 1) * \mathtt{size}_{\mathsf{G}} \geq \beta)}_{\text{isolating } l \text{ creates a garbage of at least } \beta \text{ locations}})$$

▶ **Proposition.** *$l \in \mathrm{dom}(\mathcal{T})$ and has at least β descendants $\iff$ $(\mathcal{T}, \mathcal{G}, l) \models \#\mathtt{desc}(\mathtt{u}) \geq \beta$.*

$\#\mathtt{desc}(\mathtt{u}) \geq \beta$ can then be used to define a formula that checks the number of children of the location l corresponding to $\mathtt{u}$: $\#\mathtt{child}(\mathtt{u}) \geq 0 \stackrel{\text{def}}{=} \mathsf{T}(\mathtt{u})$, whereas $\#\mathtt{child}(\mathtt{u}) \geq \beta + 1$ is

$$\top * (\underbrace{(\forall \mathtt{u}\ \neg \mathsf{G}(\mathtt{u}))}_{\mathcal{G} \text{ is empty}} \wedge \overbrace{\#\mathtt{desc}(\mathtt{u}) \geq \beta + 1}^{l \text{ has at least } \beta + 1 \text{ descendants}} \wedge \underbrace{\neg(\mathtt{size}_{\mathsf{T}+\mathsf{G}} = \beta * (\mathsf{T}(\mathtt{u}) \wedge \neg \#\mathtt{desc}(\mathtt{u}) \geq 1))}_{\text{isolating } \beta \text{ memory cells makes impossible for } l \text{ to reach } \rho \text{ and have no descendants}})$$

▶ **Proposition.** *$l \in \mathrm{dom}(\mathcal{T})$ and has at least β children $\iff$ $(\mathcal{T}, \mathcal{G}, l) \models \#\mathtt{child}(\mathtt{u}) \geq \beta$.*

Notice that the size of every formula introduced above is linear with respect to β. We denote with $\mathtt{symbol}(\mathtt{u})$ the formula $\#\mathtt{desc}(\mathtt{u}) \geq 1$, which in our encoding is satisfied if and only if $\mathtt{u}$ is interpreted by a location in the main path. We define $\mathtt{1st}_{\mathsf{S}}(\mathtt{u})$ as the formula that check if $\mathtt{u}$ corresponds to the location that encodes the first letter of a word and that symbol is in $\mathsf{S} \subseteq \Sigma$. As stated above, this is the only location of the main path with the same number of descendants and children. Then, $\mathtt{1st}_{\mathsf{S}}(\mathtt{u})$ can be easily defined as follows:

$$\mathtt{1st}_{\mathsf{S}}(\mathtt{u}) \stackrel{\text{def}}{=} \bigvee_{\beta \in \mathsf{S}} (\#\mathtt{desc}(\mathtt{u}) = \beta + 1 \wedge \#\mathtt{child}(\mathtt{u}) = \beta + 1)$$

Lastly, we define a formula, linear in the size of $\Sigma = [1, n]$, that characterises the family of trees encoding a word by capturing the properties of the encoding. $\mathtt{word}_n \stackrel{\text{def}}{=} \psi \wedge \chi_n$ where

$$\psi \stackrel{\text{def}}{=} \overbrace{\neg(\exists \mathtt{u}\ \mathsf{T}(\mathtt{u}) * \exists \mathtt{u}\ \mathsf{T}(\mathtt{u}))}^{\rho \text{ has at most 1 child}} \wedge \overbrace{(\exists \mathtt{u}\ \mathtt{symbol}(\mathtt{u}) \vee \forall \mathtt{u}\ \neg \mathsf{T}(\mathtt{u}))}^{\mathcal{T} \text{ is empty or it encodes symbols}}$$

$$\chi_n \stackrel{\text{def}}{=} \forall \mathtt{u}\ (\mathtt{symbol}(\mathtt{u}) \implies \mathtt{1st}_{[1,n]}(\mathtt{u}) \vee \underbrace{((\mathtt{size}_{\mathsf{T}+\mathsf{G}} = 1 * \mathtt{1st}_{[1,n]}(\mathtt{u})) \wedge \neg \mathtt{1st}_{[1,n+1]}(\mathtt{u}))}_{\text{the location corresponding to u encodes a symbol in } [1,n] \text{ and exactly one of its children encodes a letter}})$$

▶ **Proposition.** *Let $(\mathcal{T}, \mathcal{G}, l)$ be a state. $\mathcal{T}$ encodes a word in $[1, n]^*$ $\iff$ $(\mathcal{T}, \mathcal{G}, l) \models \mathtt{word}_n$.*

3.2 An alternative semantics for PITL

Propositional interval temporal logic (PITL) [25] was introduced for the verification of hardware components. Similarly to separation logic, it contains an operator called *chop* $\mid$ that splits the model into two parts. We refer the reader to [24] for a complete description of PITL and consider here its interpretation under *locality principle*, known to be Tower-complete (the full logic is undecidable). Every formula of PITL is built from

$$\varphi \coloneqq \mathtt{a} \mid \mathtt{pt} \mid \varphi \wedge \varphi \mid \neg\varphi \mid \varphi \mid \varphi$$

and is interpreted on a non-empty finite word $\mathfrak{w} \in \Sigma^+$, where the satisfaction relation $\models$ is

- $\mathfrak{w} \models \mathtt{a}$ if and only if $\mathfrak{w}$ is headed by the symbol $\mathtt{a}$, i.e. there is $\mathfrak{w}' \in \Sigma^*$ such that $\mathfrak{w} = \mathtt{a}\mathfrak{w}'$.
- $\mathfrak{w} \models \mathtt{pt}$ if and only if the length of $\mathfrak{w}$ is 1, i.e. $\mathfrak{w} \in \Sigma$.
- $\mathfrak{w} \models \varphi_1 \mid \varphi_2$ iff there are $\mathtt{a} \in \Sigma$ and $\mathfrak{w}', \mathfrak{w}'' \in \Sigma^*$ s.t. $\mathfrak{w} = \mathfrak{w}'\mathtt{a}\mathfrak{w}''$, $\mathfrak{w}'\mathtt{a} \models \varphi_1$ and $\mathtt{a}\mathfrak{w}'' \models \varphi_2$.

PITL seems a good candidate for a reduction, since finite words and the predicates $\mathtt{a}$, $\mathtt{pt}$ can be easily encoded in it. Capturing $\varphi_1 \mid \varphi_2$ seems however to be challenging. Let $\mathfrak{w} = \mathtt{a}_1 \cdots \mathtt{a}_k$ be a non-empty word and let $\mathcal{T}$ be a tree encoding $\mathfrak{w}$. Let $\ell_1, \cdots, \ell_k$ be the main path of $\mathcal{T}$. From the semantics of the chop operator, $\mathfrak{w} \models \varphi_1 \mid \varphi_2$ if and only if there is a position $i \in [1, k]$ so that $\mathtt{a}_1 \cdots \mathtt{a}_i \models \varphi_1$ and $\mathtt{a}_i \cdots \mathtt{a}_k \models \varphi_2$. Alternatively, we would like to split the main path of $\mathcal{T}$ so that we are able to check that its prefix with main path $\ell_1, \cdots, \ell_i$ encodes a word satisfying φ_1 whereas its suffix with main path $\ell_i, \cdots, \ell_k$ encodes a word satisfying φ_2. However, using $*$ to naïvely cutting $\mathcal{T}$ causes the locations $\ell_1, \cdots, \ell_i$ not to encode any word (as they do not reach ρ anymore), making it impossible to check the satisfaction of φ_1. To solve this issue we define an equivalent interpretation of PITL based on marked symbols.

A *marking* of an alphabet Σ is a bijection $\overline{(.)} : \Sigma \to \overline{\Sigma}$, relating each symbol $\mathtt{a} \in \Sigma$ to its *marked representation* $\overline{\mathtt{a}} \in \overline{\Sigma}$. We denote with $\mathbb{\Sigma}$ the extended alphabet $\Sigma \cup \overline{\Sigma}$. A word $\mathbb{\Sigma}^+$ is *marked* if it has some symbols from $\overline{\Sigma}$. The satisfaction relation $\models_\bullet$ on a marked word $\mathfrak{w}$ is

- $\mathfrak{w} \models_\bullet \mathtt{a}$ iff $\mathfrak{w}$ is headed by $\mathtt{a}$ or $\overline{\mathtt{a}}$.
- $\mathfrak{w} \models_\bullet \mathtt{pt}$ iff $\mathfrak{w}$ is headed by a marked symbol.
- $\mathfrak{w} \models_\bullet \varphi_1 \mid \varphi_2$ iff there are $\overline{\mathtt{a}} \in \overline{\Sigma}$, $\mathtt{b} \in \Sigma$, $\mathfrak{w}', \mathfrak{w}'_1, \mathfrak{w}'_2 \in \Sigma^*$ and $\mathfrak{w}'' \in \mathbb{\Sigma}^*$ s.t. $(\mathfrak{w} = \mathfrak{w}'\overline{\mathtt{a}}\mathfrak{w}''$, $\mathfrak{w}'\overline{\mathtt{a}}\mathfrak{w}'' \models_\bullet \varphi_1$ and $\overline{\mathtt{a}}\mathfrak{w}'' \models_\bullet \varphi_2)$ or $(\mathfrak{w}' = \mathfrak{w}'_1\mathtt{b}\mathfrak{w}'_2$, $\mathfrak{w}'_1\overline{\mathtt{b}}\mathfrak{w}'_2\overline{\mathtt{a}}\mathfrak{w}'' \models_\bullet \varphi_1$ and $\mathtt{b}\mathfrak{w}'_2\overline{\mathtt{a}}\mathfrak{w}'' \models_\bullet \varphi_2)$.

Here, the satisfaction of a formula is only checked on the prefix $\mathtt{a}_1 \cdots \mathtt{a}_{i-1}\overline{\mathtt{a}}_i$ of $\mathfrak{w}$ that ends with the first marked letter. To check whether $\mathfrak{w} \models_\bullet \varphi_1 \mid \varphi_2$ we search for a position $j \in [1, i]$ inside this prefix so that φ_1 is satisfied by $\mathfrak{w}$ updated so that its j-th letter is marked, and φ_2 is satisfied by the suffix of $\mathfrak{w}$ starting in j. As shown in the next section, taking the suffix of a word and marking a symbol can be simulated in ALT. As the two semantics of PITL are shown to be equivalent (by induction on the structure of φ), this makes ALT capture PITL.

▶ **Theorem 1** ($\models$ equiv. $\models_\bullet$). *Let $\mathfrak{w} \in \Sigma^+$. Let $\overline{\mathfrak{w}} = \mathfrak{w}'\overline{\mathtt{a}}\mathfrak{w}''$ with $\mathfrak{w}' \in \Sigma^*$, $\overline{\mathtt{a}} \in \overline{\Sigma}$ and $\mathfrak{w}'' \in \mathbb{\Sigma}^*$. If $\mathfrak{w} = \mathfrak{w}'\mathtt{a}$ then for every PITL formula φ we have $\mathfrak{w} \models \varphi \iff \overline{\mathfrak{w}} \models_\bullet \varphi$.*

3.3 Tower-completeness of ALT and other complexity results

We reduce the satisfiability problem of PITL on marked words to the satisfiability problem of ALT. Let $\Sigma = [1, n]$, $\mathbb{\Sigma} = \Sigma \cup \overline{\Sigma}$ and let $\mathfrak{f} : \mathbb{\Sigma} \to [1, 2n]$ be the bijection $\mathfrak{f}(\mathtt{a}) \stackrel{\text{def}}{=} 2\mathtt{a}$ for $\mathtt{a} \in \Sigma$ and $\mathfrak{f}(\overline{\mathtt{a}}) \stackrel{\text{def}}{=} 2\mathtt{a} - 1$ for $\overline{\mathtt{a}} \in \overline{\Sigma}$. $\mathfrak{f}(\mathtt{a}_1 \cdots \mathtt{a}_k)$ denotes the translated word $\mathfrak{f}(\mathtt{a}_1) \cdots \mathfrak{f}(\mathtt{a}_k)$. $\mathfrak{f}$ maps $\mathbb{\Sigma}$ into the alphabet $[1, 2n]$, whose words can be encoded into trees (as in Section 3.1). In these trees each symbol $\mathtt{a} \in \Sigma$ (resp. $\overline{\mathtt{a}} \in \overline{\Sigma}$) corresponds to a location ℓ in the main path having $2\mathtt{a} + 1$ (resp. $2\mathtt{a}$) children not in this path. Then, removing exactly one of these children is equal to marking a symbol. We can check if the assignment of $\mathtt{u}$ encodes marked symbols:

$$\mathtt{marked}_n(\mathtt{u}) \stackrel{\text{def}}{=} \bigvee_{i \in [1,n]} ((\#\mathtt{child}(\mathtt{u}) = 2i \wedge \mathtt{1st}_{[1,2n]}(\mathtt{u})) \vee (\underbrace{\#\mathtt{child}(\mathtt{u}) = 2i + 1}_{\text{the location corresponding to } \mathtt{u} \text{ has } 2i \text{ children not in the main path and one in it}} \wedge \neg \mathtt{1st}_{[1,2n]}(\mathtt{u})))$$

As stated in Section 3.2, $\mathfrak{w} \models_\bullet \varphi$ examine the prefix of $\mathfrak{w}$ that ends with the first marked symbol. In trees $\mathcal{T}$ encoding $\mathfrak{w}$, this corresponds to the locations that reach every encoding of marked symbols. The idea is then to track the number of these symbols in $\mathcal{T}$. The formula $\mathtt{marks}_n \geq \beta$, defined as $\top$ for $\beta = 0$, shown below is satisfied only by trees with at least β marked symbols. Then, the formula $\#\mathtt{marks}_n(\mathtt{u}) \geq \beta$ checks if the current assignment of $\mathtt{u}$ encodes a symbol that reaches at least (other) β marked locations.

$$\begin{aligned} \mathtt{marks}_n \geq \beta &\stackrel{\text{def}}{=} \exists \mathtt{u}\ (\mathtt{marked}_n(\mathtt{u}) \wedge \overbrace{((\mathtt{size}_{\mathsf{T}+\mathsf{G}} = 1 \wedge \mathtt{alloc}(\mathtt{u})) * \mathtt{marks}_n \geq \beta - 1)}^{\text{by excluding a marked location, at least other } \beta - 1 \text{ such locations can be found}}) \\ \#\mathtt{marks}_n(\mathtt{u}) \geq \beta &\stackrel{\text{def}}{=} \mathtt{symbol}(\mathtt{u}) \wedge ((\mathtt{size}_{\mathsf{T}+\mathsf{G}} = 1 \wedge \mathtt{alloc}(\mathtt{u})) * \mathtt{marks}_n \geq \beta) \end{aligned}$$

At last, we define the translation $\nabla_\beta(\varphi)$, parametrised on the number $\beta \geq 1$ of marked symbols, of a PITL formula φ. $\nabla_\beta(\varphi)$ is homomorphic for Boolean connectives, $\nabla_\beta(\mathtt{a})$ and $\nabla_\beta(\mathtt{pt})$ are respectively $\exists \mathtt{u}\ \mathtt{1st}_{[2\alpha-1,2\alpha]}(\mathtt{u})$ and $\exists \mathtt{u}\ (\mathtt{1st}_{[1,2n]}(\mathtt{u}) \wedge \mathtt{marked}_n(\mathtt{u}))$, whereas $\nabla_\beta(\varphi_1 \mid \varphi_2)$ is

$$\begin{aligned} \exists \mathtt{u}\ \Big(&\mathtt{symbol}(\mathtt{u}) \wedge \big((\mathtt{1st}_{[1,2n]}(\mathtt{u}) \wedge \mathtt{marked}_n(\mathtt{u}) \wedge \nabla_\beta(\varphi_1) \wedge \nabla_\beta(\varphi_2)) \vee \\ &(\mathtt{1st}_{[1,2n]}(\mathtt{u}) \wedge \neg \mathtt{marked}_n(\mathtt{u}) \wedge (\mathtt{size}_{\mathsf{G}} = 1 * (\mathtt{marked}_n(\mathtt{u}) \wedge \nabla_{\beta+1}(\varphi_1))) \wedge \nabla_\beta(\varphi_2)) \vee \\ &(\neg \mathtt{1st}_{[1,2n]}(\mathtt{u}) \wedge \mathtt{marked}_n(\mathtt{u}) \wedge \#\mathtt{marks}_n(\mathtt{u}) \geq \beta - 1 \wedge \nabla_\beta(\varphi_1) \wedge (\mathtt{size}_{\mathsf{G}} = 1 * (\mathtt{1st}_{[1,2n]}(\mathtt{u}) \wedge \nabla_\beta(\varphi_2)))) \vee \\ &(\neg \mathtt{1st}_{[1,2n]}(\mathtt{u}) \wedge \neg \mathtt{marked}_n(\mathtt{u}) \wedge \#\mathtt{marks}_n(\mathtt{u}) \geq \beta \wedge (\mathtt{size}_{\mathsf{G}} = 1 * (\mathtt{marked}_n(\mathtt{u}) \wedge \nabla_{\beta+1}(\varphi_1))) \\ &\wedge (\mathtt{size}_{\mathsf{G}} = 1 * (\mathtt{1st}_{[1,2n]}(\mathtt{u}) \wedge \nabla_\beta(\varphi_2))))\big)\Big). \end{aligned}$$

The translation follows closely the relation $\models_\bullet$. The case for $\nabla_\beta(\varphi_1 \mid \varphi_2)$ is split into four disjuncts, depending on whether or not $\mathtt{u}$ points to a location (1) encoding the first letter of the word and (2) encoding the first marked symbol. For example, in the third disjunct $\mathtt{u}$ points to a location l which is not the first in the main path but that encodes the first marked symbol. $\nabla_\beta(\varphi_1)$ needs then to hold on the current state whereas $\nabla_\beta(\varphi_2)$ must be checked with respect to the portion of tree where l encodes the same marked symbol but is now the first location in the main path. To obtain this, the formula cuts the tree by only removing the child of l that is in the main path. Lemma 2 (whose proof is by induction on the structure of φ) ensures that the translation captures the semantics of the formula. The reduction of PITL with the standard semantics (Theorem 3) then stems from Theorem 1.

▶ **Lemma 2.** *Let $\mathfrak{w} \in \mathbb{\Sigma}^+$ be a marked word with $\beta \geq 1$ marked symbols. Let $(\mathcal{T}, \mathcal{G}, l)$ be a state such that $\mathcal{T}$ encodes $\mathfrak{f}(\mathfrak{w})$. For every* PITL *formula φ, $\mathfrak{w} \models_\bullet \varphi \iff (\mathcal{T}, \mathcal{G}, l) \models \nabla_\beta(\varphi)$.*

▶ **Theorem 3** (PITL to ALT)**.** *Every* PITL *formula φ interpreted on $\models$ is equisatisfiable with*

$$\underbrace{\mathtt{word}_{2n} \wedge \exists \mathtt{u}\ \mathsf{T}(\mathtt{u})}_{\mathcal{T} \text{ encodes a non-empty word}} \wedge \forall \mathtt{u}\ (\mathtt{marked}_n(\mathtt{u}) \Leftrightarrow \underbrace{(\mathsf{T}(\mathtt{u}) \wedge \neg(\top * \mathsf{G}(\mathtt{u})))}_{\mathtt{u} \text{ is interpreted a child of } \rho}) \wedge \nabla_1(\varphi).$$

Complexity results. Even though the translation from PITL to ALT is exponential, the Tower-completeness of PITL [24] ensures that ALT is Tower-hard. It remains to show that

ALT is captured by $\mathsf{1SL}_{\mathtt{R1}}(*, -\!\!*, \mathtt{reach}^+)$. The translation $\tau_{\mathtt{x}}(\varphi)$ of an ALT formula φ provided here is quite straightforward, with the only specificity being the role of $\mathtt{x} \in \mathsf{VAR}$ as the root of the tree. $\tau_{\mathtt{x}}(\varphi)$ is homomorphic for Boolean connectives and $*$ operators, $\tau_{\mathtt{x}}(\mathsf{T}(\mathtt{u})) \stackrel{\text{def}}{=} \mathtt{reach}^+(\mathtt{u}, \mathtt{x})$, $\tau_{\mathtt{x}}(\mathsf{G}(\mathtt{u})) \stackrel{\text{def}}{=} \mathtt{alloc}(\mathtt{u}) \wedge \neg\mathtt{reach}^+(\mathtt{u}, \mathtt{x})$ and $\tau_{\mathtt{x}}(\exists \mathtt{u}\ \varphi) \stackrel{\text{def}}{=} \exists \mathtt{u}\ (\mathtt{u} \neq \mathtt{x} \wedge \tau_{\mathtt{x}}(\varphi))$. Its soundness is proved by induction on the structure of φ (Lemma 4) and implies Theorem 5.

▶ **Lemma 4.** *Let φ be a* ALT *formula and let $\mathtt{x} \in \mathsf{VAR}$. Let $(\mathcal{T}, \mathcal{G}, l)$ be a state and let s be a store such that $s(\mathtt{x}) = \rho$. Then, $(\mathcal{T}, \mathcal{G}, l) \models \varphi$ if and only if $(s, \mathcal{T} + \mathcal{G}, l) \models \tau_{\mathtt{x}}(\varphi)$.*

▶ **Theorem 5.** *Let $\mathtt{x} \in \mathsf{VAR}$. Every* ALT *formula φ is equisat. with $\mathtt{u} \neq \mathtt{x} \wedge \neg\mathtt{alloc}(\mathtt{x}) \wedge \tau_{\mathtt{x}}(\varphi)$.*

As the separating implication only appears inside `alloc` predicates, the formula obtained through the translation satisfies R1. This proves both that $\mathsf{1SL}_{\mathtt{R1}}(*, -\!\!*, \mathtt{reach}^+)$ is TOWER-hard and that ALT is TOWER-complete as it is captured by $\mathsf{1SL}(*, \mathtt{reach}^+, \mathtt{alloc})$, a fragment of first-order separation logic without $-\!\!*$ which is known to be TOWER-complete [4].

4 PSpace-completeness of $\mathsf{1SL}(*, -\!\!*, \mathtt{reach}^+)$ under R1 and R2

We now consider $\mathsf{1SL}^{\mathtt{R1}}_{\mathtt{R2}}(*, -\!\!*, \mathtt{reach}^+)$, the fragment of $\mathsf{1SL}(*, -\!\!*, \mathtt{reach}^+)$ satisfying both R1 and R2. Formulæ of this fragment are built from the non-terminals of the following grammar.

$$\mathcal{C} \coloneqq e_1 = e_2 \mid e_1 \hookrightarrow e_2 \mid \mathtt{emp} \mid \mathcal{C} \wedge \mathcal{C} \mid \neg\mathcal{C} \mid \exists \mathtt{u}\ \mathcal{C} \mid \mathcal{C} * \mathcal{C} \mid \mathcal{A} -\!\!* \mathcal{C}$$
$$\mathcal{A} \coloneqq \mathcal{C} \mid \mathtt{reach}^+(\mathtt{x}, e_1) \mid \mathtt{reach}^+(\mathtt{u}, \mathtt{u}) \mid \mathcal{A} \wedge \mathcal{A} \mid \neg\mathcal{A} \mid \exists \mathtt{u}\ \mathcal{A} \mid \mathcal{A} * \mathcal{A}$$

where $\mathtt{x} \in \mathsf{VAR}$, $e_1, e_2 \in \mathsf{VAR} \cup \{\mathtt{u}\}$. Notice how each *antecedent* of a separating implication $(-\!\!*)$ is in $\mathcal{A}$ whereas its *consequent* is in $\mathcal{C}$. We say that φ is a $\mathcal{A}$-formula (resp. $\mathcal{C}$-formula) whenever it is in the language generated by $\mathcal{A}$ (resp. $\mathcal{C}$). Every $\mathcal{C}$-formula is thus a $\mathcal{A}$-formula.

For this logic, we show that satisfiability can be solved in PSPACE by proving a *small model property*: every satisfiable formula has a polynomial size model. As far as we know, this is the first fragment with reachability predicates in the scope of $-\!\!*$ that is proved decidable. Moreover, it subsumes the logics studied in [12] and [13] which were also found to be PSPACE-complete. The result is shown by extending the *test formulæ technique* introduced by Lozes in [22]: we design a finite index equivalence relation on memory states based on the satisfaction of atomic predicates (the *test formulæ*). Then, we show that any formula of our logic can be expressed as a Boolean combination of test formulæ, effectively replacing quantifiers and spatial connectives. A proof of small model property for Boolean combination of test formulæ thus extends to $\mathsf{1SL}^{\mathtt{R1}}_{\mathtt{R2}}(*, -\!\!*, \mathtt{reach}^+)$. To handle the asymmetry of $\mathcal{A} -\!\!* \mathcal{C}$, the technique is here extended by introducing two families of test formulæ, one for the $\mathcal{C}$ fragment and one for the $\mathcal{A}$ fragment (i.e. respectively the set of every $\mathcal{C}$-formula and the set of every $\mathcal{A}$-formula). The two families are then combined together in order to deduce the complexity of $\mathsf{1SL}^{\mathtt{R1}}_{\mathtt{R2}}(*, -\!\!*, \mathtt{reach}^+)$ (as we show in Section 4.3).

4.1 A family of test formulæ capturing $\mathcal{C}$

In order to define the set of test formulæ for the $\mathcal{C}$ fragment we proceed as follows: (**1**) we introduce a set of syntactical terms and a partition of a memory state. (**2**) We then highlight properties of these objects by introducing the set of test formulæ. (**3**) Lastly, we show that the test formulæ internalise the semantics of separating conjunctions and quantifications. The same steps are carried out, in Section 4.2, for the test formulæ of the $\mathcal{A}$ fragment.

As we are interested in the satisfiability of a given formula, it is natural to consider only the finite set $\mathsf{X} \subseteq_{\mathtt{fin}} \mathsf{VAR}$ of variables appearing in it. The *syntactical terms* $\mathcal{C}\mathsf{Term}_{\mathsf{X}}$ are defined as $\mathsf{X} \cup \mathcal{C}\mathsf{Next}_{\mathsf{X}}$, where $\mathcal{C}\mathsf{Next}_{\mathsf{X}} \stackrel{\text{def}}{=} \{\mathtt{n}(\mathtt{x}) \mid \mathtt{x} \in \mathsf{X}\}$ is the set of *next-point variables*. Given

a memory state (s, h, l), we write $[\![\mathtt{x}]\!]^{\mathsf{X}}_{s,h}$ for $s(\mathtt{x})$ and $[\![\mathtt{n}(\mathtt{x})]\!]^{\mathsf{X}}_{s,h}$ to denote (if it exists) the location $h(s(\mathtt{x}))$. The locations corresponding to terms are said to be *labelled* and their set is denoted with $\mathcal{C}\mathsf{Labels}^{\mathsf{X}}_{s,h}$. Labelled locations correspond to locations for which the logic is able to express particular properties. As such, the test formulæ primarily speak about relationships between these locations, as well as the following subsets of $\mathrm{dom}(h)$:

- $\mathcal{C}\mathsf{Pred}^{\mathsf{X}}_{s,h}(\ell) \stackrel{\text{def}}{=} \{\ell' \in \mathrm{dom}(h) \setminus \mathcal{C}\mathsf{Labels}^{\mathsf{X}}_{s,h} \mid h(\ell') = \ell\}$, for every $\ell \in s(\mathsf{X})$, i.e. the set of unlabelled predecessors of a location corresponding to a program variable.
- $\mathcal{C}\mathsf{Loop}^{\mathsf{X}}_{s,h} \stackrel{\text{def}}{=} \{\ell \in \mathrm{dom}(h) \setminus \mathcal{C}\mathsf{Labels}^{\mathsf{X}}_{s,h} \mid h(\ell) = \ell\}$, i.e. the set of unlabelled self-loops.
- $\mathcal{C}\mathsf{Size}^{\mathsf{X}}_{s,h} \stackrel{\text{def}}{=} \mathrm{dom}(h) \setminus (\mathcal{C}\mathsf{Labels}^{\mathsf{X}}_{s,h} \cup \mathcal{C}\mathsf{Loop}^{\mathsf{X}}_{s,h} \cup \bigcup_{\ell \in s(\mathsf{X})} \mathcal{C}\mathsf{Pred}^{\mathsf{X}}_{s,h}(\ell))$, i.e. the set of unlabelled locations that do not self-loop and are not predecessors of program variables.

Notice how $\{\mathrm{dom}(h) \cap \mathcal{C}\mathsf{Labels}^{\mathsf{X}}_{s,h}, \mathcal{C}\mathsf{Loop}^{\mathsf{X}}_{s,h}, \mathcal{C}\mathsf{Size}^{\mathsf{X}}_{s,h}\} \cup \{\mathcal{C}\mathsf{Pred}^{\mathsf{X}}_{s,h}(\ell) \mid \ell \in s(\mathsf{X})\}$ partitions $\mathrm{dom}(h)$. The test formulæ $\mathcal{C}\mathsf{TEST}(\mathsf{X}, \alpha)$ are parametric on X and $\alpha \in \mathbb{N}^+$. Here, α is a quantity that roughly express upper-bounds on the capabilities of a $\mathcal{C}$-formula φ to check the sizes of the sets of the partition. In Section 4.3 we show how α is connected to the size of φ. $\mathcal{C}\mathsf{TEST}(\mathsf{X}, \alpha)$ is divided into two sets, a *skeleton* $\mathcal{C}\mathsf{SKEL}(\mathsf{X}, \alpha)$ expressing structural properties that do not depend on the assignment of $\mathtt{u}$, and an *observed* set $\mathcal{C}\mathsf{OBS}(\mathsf{X}, \alpha)$ of relationships between the memory state and the location currently assigned to $\mathtt{u}$.

$$\mathcal{C}\mathsf{SKEL}(\mathsf{X}, \alpha) \stackrel{\text{def}}{=} \left\{ \begin{array}{l|r} \mathtt{t}_1 = \mathtt{t}_2,\ \mathtt{alloc}(\mathtt{t}_1),\ \mathtt{t}_1 \hookrightarrow \mathtt{x},\ \mathtt{t}_1 \hookrightarrow \mathtt{t}_1, & \mathtt{x} \in \mathsf{X},\ \beta \in [1, \alpha] \\ \#\mathtt{pred}^{\mathcal{C}}_{\mathsf{X}}(\mathtt{x}) \geq \beta,\ \#\mathtt{loop}^{1}_{\mathsf{X}} \geq \beta,\ \mathtt{size}^{\mathcal{C}}_{\mathsf{X}} \geq \beta & \text{and } \mathtt{t}_1, \mathtt{t}_2 \in \mathcal{C}\mathsf{Term}_{\mathsf{X}} \end{array} \right\}$$

$$\mathcal{C}\mathsf{OBS}(\mathsf{X}, \alpha) \stackrel{\text{def}}{=} \left\{ \mathtt{u} = \mathtt{t},\ \mathtt{u} \in \mathtt{pred}^{\mathcal{C}}_{\mathsf{X}}(\mathtt{x}),\ \mathtt{u} \in \mathtt{loop}^{1}_{\mathsf{X}},\ \mathtt{u} \in \mathtt{size}^{\mathcal{C}}_{\mathsf{X}} \;\middle|\; \mathtt{x} \in \mathsf{X} \text{ and } \mathtt{t} \in \mathcal{C}\mathsf{Term}_{\mathsf{X}} \right\}$$

The formal semantics of the test formulæ is provided below:

- $(s, h, l) \models \mathtt{t}_1 = \mathtt{t}_2$ if and only if $[\![\mathtt{t}_1]\!]^{\mathsf{X}}_{s,h}$ and $[\![\mathtt{t}_2]\!]^{\mathsf{X}}_{s,h}$ are defined and $[\![\mathtt{t}_1]\!]^{\mathsf{X}}_{s,h} = [\![\mathtt{t}_2]\!]^{\mathsf{X}}_{s,h}$.
- $(s, h, l) \models \mathtt{alloc}(\mathtt{t}_1)$ if and only if $[\![\mathtt{t}_1]\!]^{\mathsf{X}}_{s,h}$ is defined and $[\![\mathtt{t}_1]\!]^{\mathsf{X}}_{s,h} \in \mathrm{dom}(h)$.
- $(s, h, l) \models \mathtt{t}_1 \hookrightarrow \mathtt{x}$ if and only if $[\![\mathtt{t}_1]\!]^{\mathsf{X}}_{s,h}$ is defined and $h([\![\mathtt{t}_1]\!]^{\mathsf{X}}_{s,h}) = s(\mathtt{x})$.
- $(s, h, l) \models \mathtt{t}_1 \hookrightarrow \mathtt{t}_1$ if and only if $[\![\mathtt{t}_1]\!]^{\mathsf{X}}_{s,h}$ is defined and $h([\![\mathtt{t}_1]\!]^{\mathsf{X}}_{s,h}) = [\![\mathtt{t}_1]\!]^{\mathsf{X}}_{s,h}$.
- $(s, h, l) \models \#\mathtt{pred}^{\mathcal{C}}_{\mathsf{X}}(\mathtt{x}) \geq \beta$ if and only if $\mathrm{card}(\mathcal{C}\mathsf{Pred}^{\mathsf{X}}_{s,h}(s(\mathtt{x}))) \geq \beta$.
- $(s, h, l) \models \#\mathtt{loop}^{1}_{\mathsf{X}} \geq \beta$ if and only if $\mathrm{card}(\mathcal{C}\mathsf{Loop}^{\mathsf{X}}_{s,h}) \geq \beta$.
- $(s, h, l) \models \mathtt{size}^{\mathcal{C}}_{\mathsf{X}} \geq \beta$ if and only if $\mathrm{card}(\mathcal{C}\mathsf{Size}^{\mathsf{X}}_{s,h}) \geq \beta$.
- $(s, h, l) \models \mathtt{u} = \mathtt{t}$ if and only if $[\![\mathtt{t}]\!]^{\mathsf{X}}_{s,h}$ is defined and $l = [\![\mathtt{t}]\!]^{\mathsf{X}}_{s,h}$.
- $(s, h, l) \models \mathtt{u} \in \mathtt{pred}^{\mathcal{C}}_{\mathsf{X}}(\mathtt{x})$ if and only if $l \in \mathcal{C}\mathsf{Pred}^{\mathsf{X}}_{s,h}(s(\mathtt{x}))$.
- $(s, h, l) \models \mathtt{u} \in \mathtt{loop}^{1}_{\mathsf{X}}$ if and only if $l \in \mathcal{C}\mathsf{Loop}^{\mathsf{X}}_{s,h}$.
- $(s, h, l) \models \mathtt{u} \in \mathtt{size}^{\mathcal{C}}_{\mathsf{X}}$ if and only if $l \in \mathcal{C}\mathsf{Size}^{\mathsf{X}}_{s,h}$.

In $\mathcal{C}\mathsf{TEST}(\mathsf{X}, \alpha)$, the classical predicates $=$, $\hookrightarrow$ and $\mathtt{alloc}$ are extended to terms of $\mathcal{C}\mathsf{Term}_{\mathsf{X}}$. For instance $\mathtt{n}(\mathtt{x}) \hookrightarrow \mathtt{y}$ is satisfied by only those memory states (s, h, l) where $h(h(s(\mathtt{x}))) = s(\mathtt{y})$. There are some restrictions: all program variables are from X and $\mathtt{t}_2 \hookrightarrow \mathtt{t}_1$ is syntactically constrained so that $\mathtt{t}_2$ is equal to $\mathtt{t}_1$ when the latter is a next-point variable (so for instance $\mathtt{n}(\mathtt{x}) \hookrightarrow \mathtt{n}(\mathtt{y})$ is not a test formula). The test formulæ $\#\mathtt{pred}^{\mathcal{C}}_{\mathsf{X}}(\mathtt{x}) \geq \beta$, $\#\mathtt{loop}^{1}_{\mathsf{X}} \geq \beta$ and $\mathtt{size}^{\mathcal{C}}_{\mathsf{X}} \geq \beta$ are respectively satisfied whenever the sets $\mathcal{C}\mathsf{Pred}^{\mathsf{X}}_{s,h}(s(\mathtt{x}))$, $\mathcal{C}\mathsf{Loop}^{\mathsf{X}}_{s,h}$ and $\mathcal{C}\mathsf{Size}^{\mathsf{X}}_{s,h}$ have size at least β. As β is bounded by α, memory states having both at least α elements in one of these sets satisfy the same test formulæ related to that set. Lastly, the formulæ $\mathtt{u} \in \mathtt{pred}^{\mathcal{C}}_{\mathsf{X}}(\mathtt{x})$, $\mathtt{u} \in \mathtt{loop}^{1}_{\mathsf{X}}$ and $\mathtt{u} \in \mathtt{size}^{\mathcal{C}}_{\mathsf{X}}$ respectively check whether the location currently assigned to $\mathtt{u}$ is in $\mathcal{C}\mathsf{Pred}^{\mathsf{X}}_{s,h}(s(\mathtt{x}))$, $\mathcal{C}\mathsf{Loop}^{\mathsf{X}}_{s,h}$ or $\mathcal{C}\mathsf{Size}^{\mathsf{X}}_{s,h}$.

It is possible to show that each test formula can be expressed with a $\mathcal{C}$-formula. For instance, $\mathtt{u} \in \mathtt{loop}^{1}_{\mathsf{X}}$ is equivalent to $\mathtt{u} \hookrightarrow \mathtt{u} \wedge \bigwedge_{\mathtt{x} \in \mathsf{X}} (\mathtt{u} \neq \mathtt{x} \wedge \neg\, \mathtt{x} \hookrightarrow \mathtt{u})$ and the formula $\#\mathtt{loop}^{1}_{\mathsf{X}} \geq \beta$ is equivalent to $\exists \mathtt{u}\ (\mathtt{u} \in \mathtt{loop}^{1}_{\mathsf{X}} \wedge ((\mathtt{alloc}(\mathtt{u}) \wedge \mathtt{size} = 1) * \#\mathtt{loop}^{1}_{\mathsf{X}} \geq \beta - 1))$,

where $\#\mathtt{loop}^1_{\mathsf{X}} \geq 0 \stackrel{\text{def}}{=} \top$. Although this result ensures that the test formulæ are not more expressive than the logic, we need to show the converse. To do so, we start by defining an indistinguishability relation between memory states, denoted with $(s,h,l) \approx^{\mathcal{C}}_{\mathsf{X},\alpha} (s',h',l')$, that holds if and only if for all $\varphi \in \mathcal{C}\mathsf{TEST}(\mathsf{X},\alpha)$ it holds $(s,h,l) \models \varphi \iff (s',h',l') \models \varphi$.

We then use this relation to show that the expressive power of the test formulæ allows to mimic quantifiers and separating conjunctions, as done in [12] for separation logic with one quantified variable. This result should not be surprising, as the equivalence relation $\simeq_\alpha$ defined in [12] (Def. 3.8) can be shown equivalent to $\approx^{\mathcal{C}}_{\mathsf{X},\alpha}$. We first handle the quantifiers.

▶ **Lemma 6** ($\mathcal{C}$:∃ indistinguishability). *Assume* $(s,h,l) \approx^{\mathcal{C}}_{\mathsf{X},\alpha} (s',h',l')$. *Let* $\ell \in \mathsf{LOC} \setminus \mathsf{L}$ *with* $\mathsf{L} \stackrel{\text{def}}{=} \mathrm{dom}(h') \cup \mathrm{ran}(h') \cup s'(\mathsf{X})$. *For every* $l_1 \in \mathsf{LOC}$ *there is* $l'_1 \in \mathsf{L} \cup \{\ell\}$ *s.t.* $(s,h,l_1) \approx^{\mathcal{C}}_{\mathsf{X},\alpha} (s',h',l'_1)$.

This holds as we show that for every assignment l_1 we can find a location l'_1 so that the formulæ in $\mathcal{C}\mathsf{OBS}(\mathsf{X},\alpha)$ are equisatisfied by both memory states. Indeed, formulæ of $\mathcal{C}\mathsf{SKEL}(\mathsf{X},\alpha)$ do not depend on the assignment of $\mathtt{u}$ but they are key to prove the result. For example, suppose that $l_1 \in \mathcal{C}\mathsf{Loop}^{\mathsf{X}}_{s,h}$. From the equisatisfaction of $\#\mathtt{loop}^1_{\mathsf{X}} \geq 1$, $\mathcal{C}\mathsf{Loop}^{\mathsf{X}}_{s',h'}$ is not empty and we can choose l'_1 to be in this set. Then, (s,h,l_1) and (s',h',l'_1) satisfy the same test formulæ. Notice how l'_1 is taken from a finite set. This is key to prove that the logic is in PSPACE (Theorem 15). Lemma 6 shows that two indistinguishable memory states cannot be distinguished using quantifiers. We show that the same holds for separating conjunctions.

▶ **Lemma 7** ($\mathcal{C}$:∗ indistinguishability). *Let* $\mathsf{X} \subseteq_{\mathtt{fin}} \mathsf{VAR}$ *and* $\alpha, \alpha_1, \alpha_2 \in \mathbb{N}^+$ *with* $\alpha = \alpha_1 + \alpha_2$. *Assume* $(s,h,l) \approx^{\mathcal{C}}_{\mathsf{X},\alpha} (s',h',l')$. *For all heaps* h_1, h_2 *such that* $h = h_1 + h_2$ *there are heaps* h'_1, h'_2 *such that* $h' = h'_1 + h'_2$, $(s,h_1,l) \approx^{\mathcal{C}}_{\mathsf{X},\alpha_1} (s',h'_1,l')$ *and* $(s,h_2,l) \approx^{\mathcal{C}}_{\mathsf{X},\alpha_2} (s',h'_2,l')$.

This result can be proved by looking at (s,h_1,l) and (s,h_2,l) in terms of their partitions

$$\{\mathrm{dom}(h_k) \cap \mathcal{C}\mathsf{Labels}^{\mathsf{X}}_{s,h_k}, \mathcal{C}\mathsf{Loop}^{\mathsf{X}}_{s,h_k}, \mathcal{C}\mathsf{Size}^{\mathsf{X}}_{s,h_k}\} \cup \{\mathcal{C}\mathsf{Pred}^{\mathsf{X}}_{s,h_k}(\ell) \mid \ell \in s(\mathsf{X})\}, \qquad k \in \{1,2\}$$

and showing that it is possible to construct h'_1 and h'_2 by dividing h' in such a way that $(s,h_1,l) \approx^{\mathcal{C}}_{\mathsf{X},\alpha_1} (s',h'_1,l')$ and $(s,h_2,l) \approx^{\mathcal{C}}_{\mathsf{X},\alpha_2} (s',h'_2,l')$. For instance, let us consider the case of unlabelled self-loops. In the following, the index k stands for 1 or 2. We define $\mathsf{L}_k \stackrel{\text{def}}{=} \mathcal{C}\mathsf{Loop}^{\mathsf{X}}_{s,h} \cap \mathrm{dom}(h_k)$, i.e. the set of unlabelled self-loops of h assigned to h_k. We partially construct h'_1 and h'_2 by defining two disjoint sets $\mathsf{L}'_1 \subseteq \mathrm{dom}(h'_1)$ and $\mathsf{L}'_2 \subseteq \mathrm{dom}(h'_2)$ so that:

$$\mathsf{L}'_1 \cup \mathsf{L}'_2 = \mathcal{C}\mathsf{Loop}^{\mathsf{X}}_{s',h'} \qquad \min(\alpha_k, \mathrm{card}(\mathsf{L}_k)) = \min(\alpha_k, \mathrm{card}(\mathsf{L}'_k)) \qquad l \in \mathsf{L}_k \Leftrightarrow l' \in \mathsf{L}'_k$$

This can be done as, by $(s,h,l) \approx^{\mathcal{C}}_{\mathsf{X},\alpha} (s',h',l')$, it follows that $l \in \mathcal{C}\mathsf{Loop}^{\mathsf{X}}_{s,h} \Leftrightarrow l' \in \mathcal{C}\mathsf{Loop}^{\mathsf{X}}_{s',h'}$ and $\min(\alpha, \mathrm{card}(\mathcal{C}\mathsf{Loop}^{\mathsf{X}}_{s,h})) = \min(\alpha, \mathrm{card}(\mathcal{C}\mathsf{Loop}^{\mathsf{X}}_{s',h'}))$. The construction goes by cases:

- if $\mathrm{card}(\mathsf{L}_1) < \alpha_1$ then select $\mathrm{card}(\mathsf{L}_1)$ locations from $\mathcal{C}\mathsf{Loop}^{\mathsf{X}}_{s',h'}$. If $l \in \mathsf{L}_1$ then l' is one of the selected locations. These locations constitute L'_1. Then, L'_2 is the set $\mathsf{L}' \setminus \mathsf{L}'_1$;
- else if $\mathrm{card}(\mathsf{L}_2) < \alpha_2$ then select $\mathrm{card}(\mathsf{L}_2)$ locations from $\mathcal{C}\mathsf{Loop}^{\mathsf{X}}_{s',h'}$. If $l \in \mathsf{L}_2$ then l' is one of the selected locations. These locations constitute L'_2. Then, $\mathsf{L}'_1 \stackrel{\text{def}}{=} \mathsf{L}' \setminus \mathsf{L}'_2$;
- otherwise $\mathrm{card}(\mathsf{L}_1) \geq \alpha_1$ and $\mathrm{card}(\mathsf{L}_2) \geq \alpha_2$. Select α_1 locations from $\mathcal{C}\mathsf{Loop}^{\mathsf{X}}_{s',h'}$. If $l \in \mathsf{L}_1$ then l' is one of the selected locations. These locations constitute L'_1. Then, $\mathsf{L}'_2 \stackrel{\text{def}}{=} \mathsf{L}' \setminus \mathsf{L}'_1$.

Suppose now that, by considering all the other elements of the partitions of h_1 and h_2, we can complete the construction of h'_1 and h'_2 so that $l \in \mathcal{C}\mathsf{Loop}^{\mathsf{X}}_{s,h_k} \setminus \mathsf{L}_k \Leftrightarrow l' \in \mathcal{C}\mathsf{Loop}^{\mathsf{X}}_{s',h'_k} \setminus \mathsf{L}'_k$ and $\min(\alpha_k, \mathrm{card}(\mathcal{C}\mathsf{Loop}^{\mathsf{X}}_{s,h_k} \setminus \mathsf{L}_k)) = \min(\alpha_k, \mathrm{card}(\mathcal{C}\mathsf{Loop}^{\mathsf{X}}_{s',h'_k} \setminus \mathsf{L}'_k))$ holds. From the properties of L'_k ensured by construction it then holds that

$$l \in \mathcal{C}\mathsf{Loop}^{\mathsf{X}}_{s,h} \Leftrightarrow l' \in \mathcal{C}\mathsf{Loop}^{\mathsf{X}}_{s',h'} \qquad \min(\alpha_k, \mathrm{card}(\mathcal{C}\mathsf{Loop}^{\mathsf{X}}_{s,h_k})) = \min(\alpha_k, \mathrm{card}(\mathcal{C}\mathsf{Loop}^{\mathsf{X}}_{s',h'_k}))$$

By semantics of the test formulæ we then conclude the following equisatisfiability results:

- $(s, h_k, l) \models \mathtt{u} \in \mathtt{loop}^1_{\mathsf{X}}$ if and only if $(s', h'_k, l') \models \mathtt{u} \in \mathtt{loop}^1_{\mathsf{X}}$;
- for each $\beta \in [1, \alpha_k]$, $(s, h_k, l) \models \#\mathtt{loop}^1_{\mathsf{X}} \geq \beta$ if and only if $(s', h'_k, l') \models \#\mathtt{loop}^1_{\mathsf{X}} \geq \beta$.

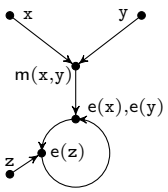
In order to complete the proof of Lemma 7, a reasoning similar to what we presented here for unlabelled self-loops is applied to each element of the partition and every test formula. This concludes the study of the family of test formulæ for the $\mathcal{C}$ fragment.

4.2 A family of test formulæ capturing $\mathcal{A}$

We now consider the $\mathcal{A}$ fragment and follow the same steps of the last section. Let $\mathsf{X} \subseteq_{\mathtt{fin}} \mathsf{VAR}$. $\mathcal{A}\mathsf{Term}_{\mathsf{X}}$ is the set of *syntactical terms* $\mathsf{X} \cup \mathcal{A}\mathsf{Meet}_{\mathsf{X}} \cup \mathcal{A}\mathsf{End}_{\mathsf{X}}$, where $\mathcal{A}\mathsf{Meet}_{\mathsf{X}} \stackrel{\text{def}}{=} \{\mathsf{m}(\mathtt{x}, \mathtt{y}) \mid \mathtt{x}, \mathtt{y} \in \mathsf{X}\}$ and $\mathcal{A}\mathsf{End}_{\mathsf{X}} \stackrel{\text{def}}{=} \{\mathsf{e}(\mathtt{x}) \mid \mathtt{x} \in \mathsf{X}\}$ are respectively the set of *meet-point* and *end-point variables*. The interpretation of terms $[\![.]\!]^{\mathsf{X}}_{s,h}$ of the last section is extended on the new terms:

- $[\![\mathsf{m}(\mathtt{x}, \mathtt{y})]\!]^{\mathsf{X}}_{s,h} = \ell \stackrel{\text{def}}{\Leftrightarrow} h^{\delta_1}(s(\mathtt{x})) = h^{\delta_2}(s(\mathtt{y})) = \ell$ for some $\delta_1, \delta_2 \geq 1$ and for each $0 \leq \delta'_1 \leq \delta_1$, $0 \leq \delta'_2 \leq \delta_2$, if $\delta'_1 \neq \delta_1$ or $\delta'_2 \neq \delta_2$ then $h^{\delta'_1}(s(\mathtt{x})) \neq h^{\delta'_2}(s(\mathtt{y}))$. For every $\delta' \geq 1$, $h^{\delta'}(\ell) \neq \ell$.
- $[\![\mathsf{e}(\mathtt{x})]\!]^{\mathsf{X}}_{s,h} = \ell \stackrel{\text{def}}{\Leftrightarrow}$ there is $\delta \geq 1$ such that $h^{\delta}(s(\mathtt{x})) = \ell$ and if $\ell \in \mathrm{dom}(h)$ then $h^{\delta'}(\ell) = \ell$ for some $\delta' \geq 1$ and for all $0 \leq \delta_1 < \delta$ there no $\delta_2 \geq 1$ is such that $h^{\delta_2}(h^{\delta_1}(s(\mathtt{x}))) = h^{\delta_1}(s(\mathtt{x}))$.

$[\![\mathsf{m}(\mathtt{x}, \mathtt{y})]\!]^{\mathsf{X}}_{s,h}$ is the first location ℓ that is reached by both $s(\mathtt{x})$ and $s(\mathtt{y})$. Moreover, for $[\![\mathsf{m}(\mathtt{x}, \mathtt{y})]\!]^{\mathsf{X}}_{s,h}$ to be defined, $s(\mathtt{x})$ and $s(\mathtt{y})$ cannot reach each other and ℓ cannot be inside a cycle. Instead, $[\![\mathsf{e}(\mathtt{x})]\!]^{\mathsf{X}}_{s,h}$ is defined if $s(\mathtt{x})$ is a memory cell that is not inside a loop, as the first location reachable from $s(\mathtt{x})$ that is inside a loop or is not in $\mathrm{dom}(h)$. We extend the notion of *labelled location* to the locations corresponding to terms of $\mathcal{A}\mathsf{Term}_{\mathsf{X}}$ and denote with $\mathcal{A}\mathsf{Labels}^{\mathsf{X}}_{s,h}$ their set. The picture on the right provide an example of the labelled locations in a memory state. As done for $\mathcal{C}$, the test formulæ of the $\mathcal{A}$ fragment express conditions on the labelled locations and on a specific partition of a memory state. Let $\alpha \in \mathbb{N}^+$. We define,



- $\mathcal{A}\mathsf{Pred}^{\mathsf{X}}_{s,h}(\ell) \stackrel{\text{def}}{=} \{\ell' \in \mathrm{dom}(h) \mid h(\ell') = \ell$ and $h^{\delta}(s(\mathtt{y})) \neq \ell'$ for every $\mathtt{y} \in \mathsf{X}$ and $\delta \geq 0\}$ for each $\ell \in s(\mathsf{X})$, i.e. the set of predecessors of a location corresponding to a program variable that are not reached by any location corresponding to a program variable;
- $\mathcal{A}\mathsf{Path}^{\mathsf{X}}_{s,h}(\ell) \stackrel{\text{def}}{=} \{\ell' \in \mathrm{dom}(h) \mid \exists \delta \geq 0\ h^{\delta}(\ell) = \ell'$ and $h^{\delta'}(\ell) \notin \mathcal{A}\mathsf{Labels}^{\mathsf{X}}_{s,h}$ for every $0 < \delta' \leq \delta\}$ for each $\ell \in \mathcal{A}\mathsf{Labels}^{\mathsf{X}}_{s,h}$. This is the set of memory cells that are reachable from the labelled location ℓ without passing through any labelled location different from ℓ;
- $\mathcal{A}\mathsf{Loop}^{\mathsf{X}}_{s,h}(\beta) \stackrel{\text{def}}{=} \big\{\{\ell_0, \cdots, \ell_{\beta-1}\} \subseteq \mathrm{dom}(h) \mid h(\ell_i) = \ell_{(i+1 \mod \beta)}$ and $\ell_i \notin \mathcal{A}\mathsf{Labels}^{\mathsf{X}}_{s,h}$, for every $i \in [0, \beta - 1]\big\}$ for each $\beta \in [1, \alpha]$, i.e. the set of unlabelled cycles of size β;
- $\mathcal{A}\mathsf{Loop}{\Uparrow}^{\mathsf{X}}_{s,h}(\alpha) \stackrel{\text{def}}{=} \big\{\{\ell_0, \cdots, \ell_\gamma\} \subseteq \mathrm{dom}(h) \mid (h(\ell_i) = \ell_{(i+1 \mod \gamma)}$ and $\ell_i \notin \mathcal{A}\mathsf{Labels}^{\mathsf{X}}_{s,h})$, for each $i \in [0, \gamma - 1]$ with $\gamma > \alpha\big\}$, i.e. the set of unlabelled cycles of size at least $\alpha + 1$;
- $\mathcal{A}\mathsf{Size}^{\mathsf{X}}_{s,h}(\alpha)$ is the set of locations in $\mathrm{dom}(h)$ that are not in any of the sets defined above.

It is straightforward to see that these sets constitute a partition of $\mathrm{dom}(h)$. Notice that any $\ell \in \mathcal{A}\mathsf{Labels}^{\mathsf{X}}_{s,h}$ is in $\mathrm{dom}(h)$ if and only if $\mathcal{A}\mathsf{Path}^{\mathsf{X}}_{s,h}(\ell) \neq \emptyset$. From the interpretation of terms, if $\mathcal{A}\mathsf{Path}^{\mathsf{X}}_{s,h}(\ell) \neq \emptyset$ then there is exactly one location ℓ' in this set that points to a labelled location. Then, we denote with $\mathcal{A}\mathsf{seen}^{\mathsf{X}}_{s,h}(\ell)$ the location $h(\ell')$, i.e. the first labelled location reachable from ℓ in at least one step. As in the previous section, the set of test formulæ $\mathcal{A}\mathsf{TEST}(\mathsf{X}, \alpha)$ is split into a *skeleton* $\mathcal{A}\mathsf{SKEL}(\mathsf{X}, \alpha)$ and an *observed* set $\mathcal{A}\mathsf{OBS}(\mathsf{X}, \alpha)$.

$$\mathcal{A}\mathsf{SKEL}(\mathsf{X}, \alpha) \stackrel{\text{def}}{=} \left\{ \begin{array}{l} \mathtt{t}_1 = \mathtt{t}_2,\ \mathtt{sees}_{\mathsf{X}}(\mathtt{t}_1, \mathtt{t}_2) \geq \beta^{\wr} \\ \#\mathtt{loop}_{\mathsf{X}}(\beta) \geq \beta^{\circlearrowleft},\ \#\mathtt{loop}^{\Uparrow}_{\mathsf{X}} \geq \beta^{\circlearrowleft} \\ \#\mathtt{pred}^{\mathcal{A}}_{\mathsf{X}}(\mathtt{x}) \geq \beta,\ \mathtt{size}^{\mathcal{A}}_{\mathsf{X}} \geq \beta \end{array} \middle| \begin{array}{r} \beta^{\wr} \in \left[1, \frac{1}{6}(\alpha+1)(\alpha+2)(\alpha+3)\right] \\ \beta^{\circlearrowleft} \in \left[1, \frac{1}{2}\alpha(\alpha+3) - 1\right],\ \beta \in [1, \alpha] \\ \mathtt{x} \in \mathsf{X},\ \mathtt{t}_1, \mathtt{t}_2 \in \mathcal{A}\mathsf{Term}_{\mathsf{X}} \end{array} \right\}$$

$$\mathcal{A}\mathsf{OBS}(\mathsf{X},\alpha) \stackrel{\text{def}}{=} \left\{ \begin{array}{l} \mathtt{u}\in\mathtt{sees}_{\mathsf{X}}(\mathtt{t}_1,\mathtt{t}_2) \geq (\overleftarrow{\beta},\overrightarrow{\beta}) \\ \mathtt{u}=\mathtt{t}_1,\ \mathtt{u}\in\mathtt{loop}_{\mathsf{X}}(\beta),\ \mathtt{u}\in\mathtt{loop}^{\Uparrow}_{\mathsf{X}} \\ \mathtt{u}\in\mathtt{pred}^{\mathcal{A}}_{\mathsf{X}}(\mathtt{x}),\ \mathtt{u}\in\mathtt{size}^{\mathcal{A}}_{\mathsf{X}} \end{array} \middle| \begin{array}{r} \overleftarrow{\beta} \in \left[1, \frac{1}{6}\alpha(\alpha+1)(\alpha+2)+1\right] \\ \overrightarrow{\beta} \in \left[1, \frac{1}{2}\alpha(\alpha+3)\right],\ \beta\in[1,\alpha] \\ \mathtt{x}\in\mathsf{X},\ \mathtt{t}_1,\mathtt{t}_2\in\mathcal{A}\mathsf{Term}_{\mathsf{X}} \end{array} \right\}$$

The formal semantics of the test formulæ is provided below:

- $(s,h,l) \models \mathtt{t}_1 = \mathtt{t}_2$ if and only if $[\![\mathtt{t}_1]\!]^{\mathsf{X}}_{s,h}$ and $[\![\mathtt{t}_2]\!]^{\mathsf{X}}_{s,h}$ are both defined, $[\![\mathtt{t}_1]\!]^{\mathsf{X}}_{s,h} = [\![\mathtt{t}_2]\!]^{\mathsf{X}}_{s,h}$.
- $(s,h,l) \models \mathtt{sees}_{\mathsf{X}}(\mathtt{t}_1,\mathtt{t}_2) \geq \beta$ iff $\mathrm{card}(\mathcal{A}\mathsf{Path}^{\mathsf{X}}_{s,h}([\![\mathtt{t}_1]\!]^{\mathsf{X}}_{s,h})) \geq \beta$, $[\![\mathtt{t}_2]\!]^{\mathsf{X}}_{s,h} = \mathcal{A}\mathsf{seen}^{\mathsf{X}}_{s,h}([\![\mathtt{t}_1]\!]^{\mathsf{X}}_{s,h})$.
- $(s,h,l) \models \#\mathtt{loop}_{\mathsf{X}}(\beta_1) \geq \beta_2$ if and only if $\mathcal{A}\mathsf{Loop}^{\mathsf{X}}_{s,h}(\beta_1)$ has at least β_2 cycles.
- $(s,h,l) \models \#\mathtt{loop}^{\Uparrow}_{\mathsf{X}} \geq \beta$ if and only if $\mathcal{A}\mathsf{Loop}{\Uparrow}^{\mathsf{X}}_{s,h}(\alpha)$ has at least β cycles.
- $(s,h,l) \models \#\mathtt{pred}^{\mathcal{A}}_{\mathsf{X}}(\mathtt{x}) \geq \beta$ if and only if $\mathcal{A}\mathsf{Pred}^{\mathsf{X}}(s(\mathtt{x}))$ has at least β elements.
- $(s,h,l) \models \mathtt{size}^{\mathcal{A}}_{\mathsf{X}} \geq \beta$ if and only if $\mathcal{A}\mathsf{Size}^{\mathsf{X}}_{s,h}(\alpha)$ has at least β elements.
- $(s,h,l) \models \mathtt{u} = \mathtt{t}$ if and only if $l = [\![\mathtt{t}]\!]^{\mathsf{X}}_{s,h}$;
- $(s,h,l) \models \mathtt{u}\in\mathtt{sees}_{\mathsf{X}}(\mathtt{t}_1,\mathtt{t}_2) \geq (\beta_1,\beta_2)$ if and only if there are $\delta_1 \geq \beta_1$ and $\delta_2 \geq \beta_2$ such that $\delta_1 + \delta_2 = \mathrm{card}(\mathcal{A}\mathsf{Path}^{\mathsf{X}}_{s,h}([\![\mathtt{t}_1]\!]^{\mathsf{X}}_{s,h}))$ and $h^{\delta_1}([\![\mathtt{t}_1]\!]^{\mathsf{X}}_{s,h}) = l$ and $h^{\delta_2}(l) = [\![\mathtt{t}_2]\!]^{\mathsf{X}}_{s,h}$;
- $(s,h,l) \models \mathtt{u}\in\mathtt{loop}_{\mathsf{X}}(\beta)$ if and only if there is a set $\mathsf{L} \in \mathcal{A}\mathsf{Loop}^{\mathsf{X}}_{s,h}(\beta)$ such that $l \in \mathsf{L}$;
- $(s,h,l) \models \mathtt{u}\in\mathtt{loop}^{\Uparrow}_{\mathsf{X}}$ if and only if there is a set $\mathsf{L} \in \mathcal{A}\mathsf{Loop}{\Uparrow}^{\mathsf{X}}_{s,h}(\alpha)$ such that $l \in \mathsf{L}$;
- $(s,h,l) \models \mathtt{u}\in\mathtt{pred}^{\mathcal{A}}_{\mathsf{X}}(\mathtt{x})$ if and only if $l \in \mathcal{A}\mathsf{Pred}^{\mathsf{X}}_{s,h}(s(\mathtt{x}))$;
- $(s,h,l) \models \mathtt{u}\in\mathtt{size}^{\mathcal{A}}_{\mathsf{X}}$ if and only if $l \in \mathcal{A}\mathsf{Size}^{\mathsf{X}}_{s,h}(\alpha)$.

As done for the $\mathcal{C}$ fragment, these test formulæ follow closely the partition defined above. For example, $\mathtt{sees}_{\mathsf{X}}(\mathtt{t}_1,\mathtt{t}_2) \geq \beta$ states that $\mathcal{A}\mathsf{Path}^{\mathsf{X}}_{s,h}([\![\mathtt{t}_1]\!]^{\mathsf{X}}_{s,h})$ describe a path of length at least β from the location corresponding to $\mathtt{t}_1$ to its nearest labelled location $[\![\mathtt{t}_2]\!]^{\mathsf{X}}_{s,h}$. Then, formula $\mathtt{u}\in\mathtt{sees}_{\mathsf{X}}(\mathtt{t}_1,\mathtt{t}_2) \geq (\beta_1,\beta_2)$ state that the current assignment of $\mathtt{u}$ is in this path and is reached from $[\![\mathtt{t}_1]\!]^{\mathsf{X}}_{s,h}$ after at least β_1 steps whereas it reaches $[\![\mathtt{t}_2]\!]^{\mathsf{X}}_{s,h}$ in at least β_2 steps.

The interesting aspect of $\mathcal{A}\mathsf{TEST}(\mathsf{X},\alpha)$ lies on the upper-bounds given to the formulæ, e.g. the bound $\frac{1}{2}\alpha(\alpha+3)-1$ on β for $\#\mathtt{loop}^{\Uparrow}_{\mathsf{X}} \geq \beta$ formulaæ. These non-trivial upper-bounds arise as we internalise the separating conjunction so that Lemma 8 (see below) holds. Since these bounds are novelty in the test formulæ proof technique, as an example we informally show how to derive the bound $\frac{1}{2}\alpha(\alpha+3)-1$ on β for $\#\mathtt{loop}^{\Uparrow}_{\mathsf{X}} \geq \beta$ formulæ. Let (s,h,l) be a memory state, $h_1 + h_2 = h$ and $\alpha,\alpha_1,\alpha_2 \in \mathbb{N}^+$ so that $\alpha = \alpha_1 + \alpha_2$ (as in Lemma 8) and (w.l.o.g.) $\alpha_1 \geq \alpha_2$. We study how the satisfaction of the test formulæ changes when the heap h is split into h_1 and h_2 by looking at the possible partitions of these two heaps.

In our simple case, for each loop $\mathsf{P} \in \mathcal{A}\mathsf{Loop}{\Uparrow}^{\mathsf{X}}_{s,h}(\alpha)$ one of the following must hold: $\mathsf{P} \subseteq \mathrm{dom}(h_1)$, $\mathsf{P} \subseteq \mathrm{dom}(h_2)$ or P is divided into two non-empty sets $\mathsf{P}_1 \subseteq \mathrm{dom}(h_1)$ and $\mathsf{P}_2 \subseteq \mathrm{dom}(h_2)$. By definition of the partition, in the first two cases we have $\mathsf{P} \in \mathcal{A}\mathsf{Loop}{\Uparrow}^{\mathsf{X}}_{s,h_k}(\alpha_k)$, for $k \in \{1,2\}$, whereas for the third case we have $\mathsf{P}_1 \subseteq \mathcal{A}\mathsf{Size}^{\mathsf{X}}_{s,h_1}(\alpha_1)$ and $\mathsf{P}_2 \subseteq \mathcal{A}\mathsf{Size}^{\mathsf{X}}_{s,h_2}(\alpha_2)$. Then, these locations affects the formulæ $\#\mathtt{loop}^{\Uparrow}_{\mathsf{X}} \geq \beta_1$ and $\mathtt{size}^{\mathcal{A}}_{\mathsf{X}} \geq \beta_2$ of $\mathcal{A}\mathsf{TEST}(\mathsf{X},\alpha_1)$ and $\mathcal{A}\mathsf{TEST}(\mathsf{X},\alpha_2)$. For $\mathcal{A}\mathsf{TEST}(\mathsf{X},\alpha)$, we denote with $\mathcal{L}(\alpha)$ the (desired) upper-bound on β for $\#\mathtt{loop}^{\Uparrow}_{\mathsf{X}} \geq \beta$ formulæ and with $\mathcal{S}(\alpha)$ the one for $\mathtt{size}^{\mathcal{A}}_{\mathsf{X}} \geq \beta'$ formulæ. For the sake of brevity, suppose we derived $\mathcal{S}(\alpha) = \alpha$ with similar arguments to the ones herein described. To effectively internalise the separating conjunct, $\mathcal{L}(\alpha)$ must be at least the sum of the upper-bounds $\mathcal{L}(\alpha_1)$ and $\mathcal{L}(\alpha_2)$ of $\mathcal{A}\mathsf{TEST}(\mathsf{X},\alpha_1)$ and $\mathcal{A}\mathsf{TEST}(\mathsf{X},\alpha_2)$ respectively, plus the upper-bound $\mathcal{S}(\alpha_1)$ of $\mathcal{A}\mathsf{TEST}(\mathsf{X},\alpha_1)$ (as $\alpha_1 \geq \alpha_2$). Then, we need to satisfy the inequality $\mathcal{L}(\alpha) \geq \max_{\alpha=\alpha_1+\alpha_2}(\mathcal{L}(\alpha_1) + \mathcal{L}(\alpha_2) + \mathcal{S}(\alpha_1) + 1)$, where the last addend 1 is introduced to handle the quantified variable $\mathtt{u}$. As $\mathcal{L}(\alpha_1) + \mathcal{L}(\alpha_2) + \mathcal{S}(\alpha_1) + 1$ is maximal for $\alpha_1 = \alpha - 1$, we solve the recurrence system $\{\mathcal{L}(1) = 1,\ \mathcal{L}(\alpha+1) = \mathcal{L}(\alpha) + \mathcal{L}(1) + \alpha + 1\}$ (here, $\mathcal{L}(1) = 1$ refers to the base-case of $\alpha = 1$) and obtain the upper-bound $\mathcal{L}(\alpha) = \frac{1}{2}\alpha(\alpha+3) - 1$ that satisfies the inequality above. Similarly, we derive all the upper-bounds of $\mathcal{A}\mathsf{TEST}(\mathsf{X},\alpha)$ *(Appendix A provides details of these derivations).*

We say that two memory states are indistinguishable, written $(s,h,l) \approx^{\mathcal{A}}_{\mathsf{X},\alpha} (s',h',l')$, for the $\mathcal{A}$ fragment if and only if for all $\varphi \in \mathcal{A}\mathsf{TEST}(\mathsf{X},\alpha)$ $(s,h,l) \models \varphi \iff (s',h',l') \models \varphi$. As for the test formulæ of the $\mathcal{C}$ fragment (Lemma 7), we can show that two indistinguishable memory states cannot be distinguished using separating conjunctions.

▶ **Lemma 8** ($\mathcal{A}$:$*$ indistinguishability). *Let* $\mathsf{X} \subseteq_{\mathtt{fin}} \mathsf{VAR}$ *and* $\alpha, \alpha_1, \alpha_2 \in \mathbb{N}^+$ *with* $\alpha = \alpha_1 + \alpha_2$. *Assume* $(s,h,l) \approx^{\mathcal{A}}_{\mathsf{X},\alpha} (s',h',l')$. *For all heaps* h_1, h_2 *such that* $h = h_1 + h_2$ *there are heaps* h'_1, h'_2 *such that* $h' = h'_1 + h'_2$, $(s,h_1,l) \approx^{\mathcal{A}}_{\mathsf{X},\alpha_1} (s',h'_1,l')$ *and* $(s,h_2,l) \approx^{\mathcal{A}}_{\mathsf{X},\alpha_2} (s',h'_2,l')$.

As done for Lemma 7, this lemma can be proved by looking at (s,h_1,l) and (s,h_2,l) in terms of their partitions and showing that it is possible to construct h'_1 and h'_2 by dividing h' in such a way that $(s,h_1,l) \approx^{\mathcal{A}}_{\mathsf{X},\alpha_1} (s',h'_1,l')$ and $(s,h_2,l) \approx^{\mathcal{A}}_{\mathsf{X},\alpha_2} (s',h'_2,l')$. In order to relate this result to the observations done for Lemma 7 and show the role of the upper-bounds introduced in this section, let us consider the case of loops of size greater than α. In the following, the index k stands for 1 or 2. For h_1 and h_2, we define the three following sets

$$\mathsf{L}_1 \stackrel{\text{def}}{=} \{\mathsf{P} \in \mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s,h}(\alpha) \mid \mathsf{P} \subseteq \mathrm{dom}(h_1)\} \qquad \mathsf{L}_2 \stackrel{\text{def}}{=} \{\mathsf{P} \in \mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s,h}(\alpha) \mid \mathsf{P} \subseteq \mathrm{dom}(h_2)\}$$

$$\mathsf{S} \stackrel{\text{def}}{=} \{(\mathsf{P}_1,\mathsf{P}_2) \mid \mathsf{P}_1 \cup \mathsf{P}_2 \in \mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s,h}(\alpha),\ \mathsf{P}_1 \neq \emptyset \neq \mathsf{P}_2 \text{ and for each } k \in \{1,2\}\ \mathsf{P}_k \subseteq \mathrm{dom}(h_k)\}$$

Then, L_k is the set of loops in $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s,h}(\alpha)$ that are completely assigned to h_k whereas S contains the loops of $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s,h}(\alpha)$ that are split between h_1 and h_2. As introduced above, for $\mathcal{A}\mathsf{TEST}(\mathsf{X},\alpha)$ we denote with $\mathcal{L}(\alpha) \stackrel{\text{def}}{=} \frac{1}{2}\alpha(\alpha+3)-1$ the upper-bound for $\#\mathtt{loop}^{\Uparrow}_{\mathsf{X}} \geq \beta$ formulæ and with $\mathcal{S}(\alpha) \stackrel{\text{def}}{=} \alpha$ the one for $\mathtt{size}^{\mathcal{A}}_{\mathsf{X}} \geq \beta'$ formulæ. Moreover, $\mathcal{P}(.)$ denotes the powerset operator whereas $\pi_k(.)$ denotes the k-th projection of a tuple. We partially construct h'_1 and h'_2 by defining three sets $\mathsf{L}'_1 \subseteq \mathcal{P}(\mathrm{dom}(h'_1))$, $\mathsf{L}'_2 \subseteq \mathcal{P}(\mathrm{dom}(h'_2))$ and $\mathsf{S}' \subseteq \mathcal{P}(\mathrm{dom}(h'_1)) \times \mathcal{P}(\mathrm{dom}(h'_2))$ satisfying the following properties:

- L'_k is the set of loops in $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha)$ that are completely assigned to h'_k
- S' contains the loops of $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha)$ that are split between h'_1 and h'_2.
- there is $\mathsf{P} \in \mathsf{L}_k$ with $l \in \mathsf{P}$ if and only if there is $\mathsf{P}' \in \mathsf{L}'_k$ with $l' \in \mathsf{P}'$;
- there is $\mathsf{P} \in \mathsf{S}$ with $l \in \pi_k(\mathsf{P})$ if and only if there is $\mathsf{P}' \in \mathsf{S}'$ with $l' \in \pi_k(\mathsf{P}')$;
- $\min(\mathcal{L}(\alpha_k), \mathrm{card}(\mathsf{L}_k)) = \min(\mathcal{L}(\alpha_k), \mathrm{card}(\mathsf{L}'_k))$;
- $\min(\mathcal{S}(\alpha_k), \mathrm{card}(\bigcup_{\mathsf{P}\in\mathsf{S}} \pi_k(\mathsf{P}))) = \min(\mathcal{S}(\alpha_k), \mathrm{card}(\bigcup_{\mathsf{P}'\in\mathsf{S}'} \pi_k(\mathsf{P}')))$.

By $(s,h,l) \approx^{\mathcal{A}}_{\mathsf{X},\alpha} (s',h',l')$ we are guaranteed to find a construction satisfying all these properties, as it implies that $\min(\mathcal{L}(\alpha), \mathrm{card}(\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s,h}(\alpha))) = \min(\mathcal{L}(\alpha), \mathrm{card}(\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha)))$ and $l \in \mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s,h}(\alpha) \Leftrightarrow l' \in \mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha)$. Recall that, by definition of the upper-bounds, $\mathcal{L}(\alpha) \geq \mathcal{L}(\alpha_1) + \mathcal{L}(\alpha_2) + \mathcal{S}(\max(\alpha_1,\alpha_2)) + 1$. Then, similarly to the proof of Lemma 7, the construction goes by cases depending on whether or not the cardinalities of L_1, L_2 and S are less than $\mathcal{L}(\alpha_1)$, $\mathcal{L}(\alpha_2)$ and $\mathcal{S}(\max(\alpha_1,\alpha_2))$ respectively and on whether or not l belongs to a set in $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s,h}(\alpha)$ *(Appendix B provides the details of this step of the construction).*

Suppose now that, by considering all the other elements of the partitions of h_1 and h_2, we can complete the construction of h'_1 and h'_2 so that both heaps enjoy the following properties:

$$l \in \mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s,h_k}(\alpha_k) \setminus \mathsf{L}_k \Leftrightarrow l' \in \mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'_k}(\alpha_k) \setminus \mathsf{L}'_k$$

$$l \in \mathcal{A}\mathsf{Size}^{\mathsf{X}}_{s,h}(\alpha_k) \setminus \bigcup_{\mathsf{P}\in\mathsf{S}} \pi_k(\mathsf{P}) \Leftrightarrow l' \in \mathcal{A}\mathsf{Size}^{\mathsf{X}}_{s',h'}(\alpha_k) \setminus \bigcup_{\mathsf{P}'\in\mathsf{S}'} \pi_k(\mathsf{P}')$$

$$\min(\mathcal{L}(\alpha_k), \mathrm{card}(\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s,h_k}(\alpha_k) \setminus \mathsf{L}_k)) = \min(\mathcal{L}(\alpha_k), \mathrm{card}(\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'_k}(\alpha_k) \setminus \mathsf{L}'_k))$$

$$\min(\mathcal{S}(\alpha_k), \mathrm{card}(\mathcal{A}\mathsf{Size}^{\mathsf{X}}_{s,h_k}(\alpha_k) \setminus \bigcup_{\mathsf{P}\in\mathsf{S}} \pi_k(\mathsf{P}))) = \min(\mathcal{S}(\alpha_k), \mathrm{card}(\mathcal{A}\mathsf{Size}^{\mathsf{X}}_{s',h'_k}(\alpha_k) \setminus \bigcup_{\mathsf{P}'\in\mathsf{S}'} \pi_k(\mathsf{P}')))$$

Then, as previously done for Lemma 7, by semantics of the test formulæ and from the properties of L'_k and S' ensured by construction we obtain the following equisatisfiability results:

- $(s, h_k, l) \models \mathtt{u} \in \mathtt{loop}^{\Uparrow}_{\mathsf{X}}$ if and only if $(s', h'_k, l') \models \mathtt{u} \in \mathtt{loop}^{\Uparrow}_{\mathsf{X}}$;
- $(s, h_k, l) \models \mathtt{u} \in \mathtt{size}^{\mathcal{A}}_{\mathsf{X}}$ if and only if $(s', h'_k, l') \models \mathtt{u} \in \mathtt{size}^{\mathcal{A}}_{\mathsf{X}}$;
- for each $\beta \in [1, \mathcal{L}(\alpha_k)]$, $(s, h_k, l) \models \#\mathtt{loop}^{\Uparrow}_{\mathsf{X}} \geq \beta$ if and only if $(s', h'_k, l') \models \#\mathtt{loop}^{\Uparrow}_{\mathsf{X}} \geq \beta$.
- for each $\beta \in [1, \mathcal{S}(\alpha_k)]$, $(s, h_k, l) \models \mathtt{size}^{\mathcal{A}}_{\mathsf{X}} \geq \beta$ if and only if $(s', h'_k, l') \models \mathtt{size}^{\mathcal{A}}_{\mathsf{X}} \geq \beta$.

In order to complete the proof of Lemma 8, a similar reasoning is applied to each element of the partition and every test formula. We conclude this section by showing that $\mathcal{A}\mathsf{TEST}(\mathsf{X}, \alpha)$ also enjoys quantification indistinguishability. The proof goes, as for Lemma 6, by checking how the satisfaction of formulæ in $\mathcal{A}\mathsf{OBS}(\mathsf{X}, \alpha)$ changes as the quantified variable is reassigned.

▶ **Lemma 9** ($\mathcal{A}$:∃ indistinguishability). *Assume* $(s, h, l) \approx^{\mathcal{A}}_{\mathsf{X},\alpha} (s', h', l')$. *Let* $\ell \in \mathsf{LOC} \setminus \mathsf{L}$ *with* $\mathsf{L} \stackrel{\text{def}}{=} \mathrm{dom}(h') \cup \mathrm{ran}(h') \cup s'(\mathsf{X})$. *For every* $l_1 \in \mathsf{LOC}$ *there is* $l'_1 \in \mathsf{L} \cup \{\ell\}$ *s.t.* $(s, h, l_1) \approx^{\mathcal{A}}_{\mathsf{X},\alpha} (s', h', l'_1)$.

4.3 Connecting the two families of test formulæ

We are now ready to show that the two indistinguishability relations introduced for the $\mathcal{C}$ and $\mathcal{A}$ fragment allow us to mimic the $\mathcal{A} \mathrel{-\!\!*} \mathcal{C}$ separating implication using the test formulæ.

▶ **Lemma 10** ($\mathcal{A} \mathrel{-\!\!*} \mathcal{C}$ indistinguishability). *Let* $(s, h, l) \approx^{\mathcal{C}}_{\mathsf{X},\alpha+\mathrm{card}(\mathsf{X})} (s', h', l')$ *for some* $\alpha \in \mathbb{N}^+$ *and* $\mathsf{X} \subseteq_{\mathtt{fin}} \mathsf{VAR}$. *For all heaps* h_1 *disjoint from* h *there exists* h'_1 *disjoint from* h' *such that:*

1. $\mathrm{card}(\mathrm{dom}(h'_1))$ *is bounded by a polynomial* $\mathfrak{P}(\mathrm{card}(\mathsf{X}), \alpha)$ *in* $\mathcal{O}(\mathrm{card}(\mathsf{X})^3 \alpha^4)$,

2. $(s, h_1, l) \approx^{\mathcal{A}}_{\mathsf{X},\alpha} (s', h'_1, l')$ *and* **3.** $(s, h + h_1, l) \approx^{\mathcal{C}}_{\mathsf{X},\alpha} (s', h' + h'_1, l')$.

The proof is achieved by defining a "small" heap h'_1 so that the first two points of the lemma are guaranteed by construction. In doing so, we need to carefully handle the labelled locations so that the third point also holds. For instance, suppose that in h_1 there is a path from $s(\mathtt{x})$ to a location corresponding to the term $\mathtt{t}$. Moreover, in this path the location $h_1(s(\mathtt{x}))$ corresponds to $\mathsf{n}(\mathtt{y})$ in h, as represented below (on the left). Then, $(s, h + h_1, l) \models \mathsf{n}(\mathtt{x}) = \mathsf{n}(\mathtt{y})$.

We then construct (see the memory state on the right) h'_1 so that there is a path from $s'(\mathtt{x})$ to a location corresponding to the term $\mathtt{t}$ where $h'_1(s'(\mathtt{x}))$ is the location corresponding to the therm $\mathsf{n}(\mathtt{y})$ in h'. The hypothesis $(s, h, l) \approx^{\mathcal{C}}_{\mathsf{X},\alpha+\mathrm{card}(\mathsf{X})} (s', h', l')$ guarantees that this can always be done correctly. The third point of the lemma is then achieved by noticing that the second point implies $(s, h_1, l) \approx^{\mathcal{C}}_{\mathsf{X},\alpha} (s', h', l')$. Indeed this holds from the following result.

▶ **Lemma 11.** *Let* $\mathsf{X} \subseteq_{\mathtt{fin}} \mathsf{VAR}$ *and* $\alpha \in \mathbb{N}^+$. *It holds that* $\approx^{\mathcal{A}}_{\mathsf{X},\alpha} \subseteq \approx^{\mathcal{C}}_{\mathsf{X},\alpha}$.

To relate the equisatisfaction of φ in $1\mathsf{SL}^{\mathtt{R1}}_{\mathtt{R2}}(*, \mathrel{-\!\!*}, \mathtt{reach}^+)$ to the indistinguishability relations, we define its *memory size* $|\varphi|_{\mathtt{m}}$ as: 1 for atomic formulæ, $|\varphi_1 \wedge \varphi_2|_{\mathtt{m}} \stackrel{\text{def}}{=} \max(|\varphi_1|_{\mathtt{m}}, |\varphi_2|_{\mathtt{m}})$, $|\neg\varphi|_{\mathtt{m}} \stackrel{\text{def}}{=} |\varphi|_{\mathtt{m}}$, $|\varphi_1 * \varphi_2|_{\mathtt{m}} \stackrel{\text{def}}{=} |\varphi_1|_{\mathtt{m}} + |\varphi_2|_{\mathtt{m}}$ and $|\varphi_1 \mathrel{-\!\!*} \varphi_2|_{\mathtt{m}} \stackrel{\text{def}}{=} \mathsf{fv}(\varphi_1 \mathrel{-\!\!*} \varphi_2) + \max(|\varphi_1|_{\mathtt{m}}, |\varphi_2|_{\mathtt{m}})$.

▶ **Lemma 12.** *Let* φ *be a* $\mathcal{A}$*-formula such that* $\mathsf{fv}(\varphi) \subseteq \mathsf{X} \subseteq_{\mathtt{fin}} \mathsf{VAR}$ *and* $|\varphi|_{\mathtt{m}} \leq \alpha \in \mathbb{N}^+$. *Let* (s, h, l), (s', h', l') *be two memory states.* $(s, h, l) \models \varphi \iff (s', h', l') \models \varphi$ *whenever*

1. $(s, h, l) \approx^{\mathcal{A}}_{\mathsf{X},\alpha} (s', h', l')$, *or* **2.** $(s, h, l) \approx^{\mathcal{C}}_{\mathsf{X},\alpha} (s', h', l')$ *and* φ *is a* $\mathcal{C}$*-formula.*

The proof is by structural induction on φ. The basic cases require to translate every atomic $\mathcal{C}$-formula and $\mathtt{reach}^+(e_1, e_2)$ into a boolean combination of respectively $\mathcal{C}\mathsf{TEST}(\mathsf{X}, 1)$ and $\mathcal{A}\mathsf{TEST}(\mathsf{X}, 1)$ formulæ. The inductive cases for Boolean connectives are immediate, whereas for $*$, $\mathrel{-\!\!*}$, $\exists$ we use the various indistinguishability lemmata. Then, the following result holds.

▶ **Theorem 13** ($\mathcal{A}$ captures the logic). *Every* $1\mathsf{SL}^{\mathtt{R1}}_{\mathtt{R2}}(*, -\!\!*, \mathtt{reach}^+)$ *formula* φ *is logically equivalent to a Boolean combination of test formulæ from* $\mathcal{A}\mathsf{TEST}(\mathsf{fv}(\varphi), |\varphi|_{\mathsf{m}})$.

For the proof, we first show that $(s,h,l) \approx^{\mathcal{A}}_{\mathsf{fv}(\varphi),|\varphi|_{\mathsf{m}}} (s',h',l')$ and $(s',h',l') \models \bigwedge_{\psi\in\mathsf{LIT}(s,h,l)} \psi$ are equivalent, where $\mathsf{LIT}(s,h,l)$ is the finite set of literals

$$\{\psi \in \mathcal{A}\mathsf{TEST}(\mathsf{fv}(\varphi), |\varphi|_{\mathsf{m}}) \mid (s,h,l) \models \psi\} \cup \{\neg\psi \mid (s,h,l) \not\models \psi \text{ and } \psi \in \mathcal{A}\mathsf{TEST}(\mathsf{fv}(\varphi), |\varphi|_{\mathsf{m}})\}$$

Then, the expression $\bigvee_{(s,h,l)\models\varphi} \bigwedge_{\psi\in\mathsf{LIT}(s,h,l)} \psi$ is equivalent to a Boolean combination χ of $\mathcal{A}\mathsf{TEST}(\mathsf{fv}(\varphi), |\varphi|_{\mathsf{m}})$ formulæ. Using Lemma 12 we conclude that φ is logically equivalent to χ.

Complexity upper bounds. Analogously to what is done in [12, 13], following Theorem 13 we can establish a small model property for $1\mathsf{SL}^{\mathtt{R1}}_{\mathtt{R2}}(*, -\!\!*, \mathtt{reach}^+)$, whose proof relies on the upper-bounds on $\mathcal{A}\mathsf{TEST}(\mathsf{X}, \alpha)$ formulæ discussed in the previous section. Let $|\mathsf{X}_\varphi| \stackrel{\text{def}}{=} \mathrm{card}(\mathsf{fv}(\varphi))$.

▶ **Theorem 14** (small model). *Every satisfiable* φ *in* $1\mathsf{SL}^{\mathtt{R1}}_{\mathtt{R2}}(*, -\!\!*, \mathtt{reach}^+)$ *is satisfied by a state* (s,h,l) *such that* $\mathrm{card}(\mathrm{dom}(h))$ *is bounded by a polynomial* $\mathcal{Q}(|\mathsf{X}_\varphi|, |\varphi|_{\mathsf{m}})$ *in* $\mathcal{O}(|\mathsf{X}_\varphi|^3 |\varphi|^4_{\mathsf{m}})$.

The satisfiability problem of $1\mathsf{SL}^{\mathtt{R1}}_{\mathtt{R2}}(*, -\!\!*, \mathtt{reach}^+)$ is then PSPACE-complete (the hardness is inherited from propositional separation logic [9]). Indeed, as $|\varphi|_{\mathsf{m}}$ is at most $|\mathsf{X}_\varphi| \times |\varphi|$, where $|\varphi|$ is the number of symbols in φ, Theorem 14 ensures a polynomial bound $\mathcal{Q}(|\mathsf{X}_\varphi|, |\mathsf{X}_\varphi| \times |\varphi|)$ on the heap of the smallest memory state that satisfies φ. Then, the non-deterministic PSPACE algorithm (leading to a PSPACE upper-bound by Savitch Theorem [27]) first guess a heap h, a store s restricted to $\mathsf{fv}(\varphi)$ and $l \in \mathsf{LOC}$ such that $\mathrm{dom}(h) \cup \mathrm{ran}(h) \cup \mathrm{ran}(s) \cup \{l\}$ has at most $2 \times \mathcal{Q}(|\mathsf{X}_\varphi|, |\mathsf{X}_\varphi| \times |\varphi|) + |\mathsf{X}_\varphi| + 1$ locations (in the worst case all these sets are disjoint). It then checks that $(s,h,l) \models \varphi$ by using a linear-depth recursive algorithm that internalises the semantics of φ (see e.g. [8]). The various indistinguishability lemmata ensure that only a polynomial amount of locations ever needs to be considered. For instance, $(s,h,l) \not\models \varphi_1 -\!\!* \varphi_2$ if and only if we can guess h_1 so that $(s,h_1,l) \models \varphi_1$, $(s,h+h_1,l) \not\models \varphi_2$ and (by Lemma 10) $\mathrm{dom}(h_1) \cup \mathrm{ran}(h_2)$ has at most $2 \times \mathfrak{P}(|\mathsf{X}_\varphi|, |\mathsf{X}_\varphi| \times |\varphi|)$ locations.

▶ **Theorem 15.** *The satisfiability problem of* $1\mathsf{SL}^{\mathtt{R1}}_{\mathtt{R2}}(*, -\!\!*, \mathtt{reach}^+)$ *can be decided in* PSPACE.

5 Conclusions

We studied $1\mathsf{SL}^{\mathtt{R1}}_{\mathtt{R2}}(*, -\!\!*, \mathtt{reach}^+)$, an extension of propositional separation logic involving one quantified variable and reachability predicates whose satisfiability problem is PSPACE-complete. We discussed how modest extensions of the logic entail TOWER-hardness.

As far as we know, $1\mathsf{SL}^{\mathtt{R1}}_{\mathtt{R2}}(*, -\!\!*, \mathtt{reach}^+)$ is the largest decidable fragment in which the separating implication cohabits with reachability predicates and quantified variables, subsuming the logics introduced in [12] and [13] which were also found to be PSPACE-complete. Moreover, crucial robustness properties lying outside the expressive power of many fragments of separation logic can be directly expressed in $1\mathsf{SL}^{\mathtt{R1}}_{\mathtt{R2}}(*, -\!\!*, \mathtt{reach}^+)$ and checked in PSPACE. Then, a procedure to decide robustness properties for this logic could perhaps be implemented inside tools using any fragment of $1\mathsf{SL}^{\mathtt{R1}}_{\mathtt{R2}}(*, -\!\!*, \mathtt{reach}^+)$ as their assertion logic (see e.g. [15]).

To prove the PSPACE upper-bound we relied on the widely used proof technique of the test formulæ, here extended to capture asymmetric spatial connectives. The work presented in [10] show a possible relation between test formulæ and games, paving a way toward better understanding this technique and generalising it even further. In particular, by also using the recurrence systems introduced here in Section 4.2, it seems possible to use this approach to tackle in a new way extensions of separation logic with user-defined inductive predicates.

Acknowledgements. I would like to thank S. Demri and E. Lozes for their feedback.

References

1 Timos Antonopoulos, Nikos Gorogiannis, Christoph Haase, Max I. Kanovich, and Joël Ouaknine. Foundations for decision problems in separation logic with general inductive predicates. In *FoSSaCS*, volume 8412 of *Lecture Notes in Computer Science*, pages 411–425. Springer, 2014.

2 Josh Berdine, Cristiano Calcagno, and Peter W. O'Hearn. Smallfoot: Modular automatic assertion checking with separation logic. In *FMCO*, volume 4111 of *Lecture Notes in Computer Science*, pages 115–137. Springer, 2005.

3 Josh Berdine, Byron Cook, and Samin Ishtiaq. Slayer: Memory safety for systems-level code. In *CAV*, volume 6806 of *Lecture Notes in Computer Science*, pages 178–183. Springer, 2011.

4 Rémi Brochenin, Stéphane Demri, and Étienne Lozes. On the almighty wand. *Inf. Comput.*, 211:106–137, 2012.

5 Luís Caires and Luca Cardelli. A spatial logic for concurrency. In *TACS*, volume 2215 of *Lecture Notes in Computer Science*, pages 1–37. Springer, 2001.

6 Cristiano Calcagno and Dino Distefano. Infer: An automatic program verifier for memory safety of C programs. In *NASA Formal Methods*, volume 6617 of *Lecture Notes in Computer Science*, pages 459–465. Springer, 2011.

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

8 Cristiano Calcagno, Hongseok Yang, and Peter W. O'Hearn. Computability and complexity results for a spatial assertion language for data structures. In *FSTTCS*, volume 2245 of *Lecture Notes in Computer Science*, pages 108–119. Springer, 2001.

9 Byron Cook, Christoph Haase, Joël Ouaknine, Matthew J. Parkinson, and James Worrell. Tractable reasoning in a fragment of separation logic. In *CONCUR*, volume 6901 of *Lecture Notes in Computer Science*, pages 235–249. Springer, 2011.

10 Anuj Dawar, Philippa Gardner, and Giorgio Ghelli. Adjunct elimination through games in static ambient logic. In *FSTTCS*, volume 3328 of *Lecture Notes in Computer Science*, pages 211–223. Springer, 2004.

11 Stéphane Demri and Morgan Deters. Two-variable separation logic and its inner circle. *ACM Trans. Comput. Log.*, 16(2):15:1–15:36, 2015.

12 Stéphane Demri, Didier Galmiche, Dominique Larchey-Wendling, and Daniel Méry. Separation logic with one quantified variable. *Theory Comput. Syst.*, 61(2):371–461, 2017.

13 Stéphane Demri, Étienne Lozes, and Alessio Mansutti. The effects of adding reachability predicates in propositional separation logic. In *FoSSaCS*, volume 10803 of *Lecture Notes in Computer Science*, pages 476–493. Springer, 2018.

14 Kamil Dudka, Petr Peringer, and Tomás Vojnar. Predator: A practical tool for checking manipulation of dynamic data structures using separation logic. In *CAV*, volume 6806 of *Lecture Notes in Computer Science*, pages 372–378. Springer, 2011.

15 Christoph Haase, Samin Ishtiaq, Joël Ouaknine, and Matthew J. Parkinson. Seloger: A tool for graph-based reasoning in separation logic. In *CAV*, volume 8044 of *Lecture Notes in Computer Science*, pages 790–795. Springer, 2013.

16 C. A. R. Hoare. An axiomatic basis for computer programming. *Commun. ACM*, 12(10):576–580, 1969.

17 Radu Iosif, Adam Rogalewicz, and Jirí Simácek. The tree width of separation logic with recursive definitions. In *CADE*, volume 7898 of *Lecture Notes in Computer Science*, pages 21–38. Springer, 2013.

18 Bart Jacobs, Jan Smans, Pieter Philippaerts, Frédéric Vogels, Willem Penninckx, and Frank Piessens. Verifast: A powerful, sound, predictable, fast verifier for C and java. In *NASA*

Formal Methods, volume 6617 of *Lecture Notes in Computer Science*, pages 41–55. Springer, 2011.

19 Christina Jansen, Jens Katelaan, Christoph Matheja, Thomas Noll, and Florian Zuleger. Unified reasoning about robustness properties of symbolic-heap separation logic. In *ESOP*, volume 10201 of *Lecture Notes in Computer Science*, pages 611–638. Springer, 2017.

20 Jens Katelaan, Dejan Jovanovic, and Georg Weissenbacher. A separation logic with data: Small models and automation. In *IJCAR*, volume 10900 of *Lecture Notes in Computer Science*, pages 455–471. Springer, 2018.

21 Quang Loc Le, Makoto Tatsuta, Jun Sun, and Wei-Ngan Chin. A decidable fragment in separation logic with inductive predicates and arithmetic. In *CAV (2)*, volume 10427 of *Lecture Notes in Computer Science*, pages 495–517. Springer, 2017.

22 Étienne Lozes. Adjuncts elimination in the static ambient logic. *Electr. Notes Theor. Comput. Sci.*, 96:51–72, 2004.

23 Stephen Magill, Ming-Hsien Tsai, Peter Lee, and Yih-Kuen Tsay. THOR: A tool for reasoning about shape and arithmetic. In *CAV*, volume 5123 of *Lecture Notes in Computer Science*, pages 428–432. Springer, 2008.

24 Ben C. Moszkowski. A hierarchical completeness proof for propositional interval temporal logic with finite time. *Journal of Applied Non-Classical Logics*, 14(1-2):55–104, 2004.

25 Benjamin Charles Moszkowski. *Reasoning About Digital Circuits*. PhD thesis, Stanford, CA, USA, 1983.

26 John C. Reynolds. Separation logic: A logic for shared mutable data structures. In *LICS*, pages 55–74. IEEE Computer Society, 2002.

27 Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *J. Comput. Syst. Sci.*, 4(2):177–192, 1970.

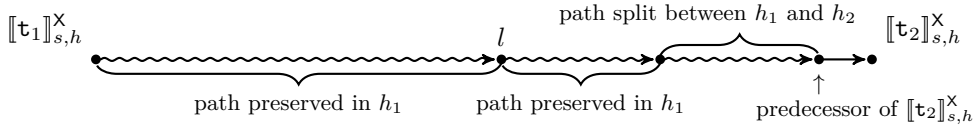
28 Sylvain Schmitz. Complexity hierarchies beyond elementary. *TOCT*, 8(1):3:1–3:36, 2016.

29 Hongseok Yang, Oukseh Lee, Josh Berdine, Cristiano Calcagno, Byron Cook, Dino Distefano, and Peter W. O'Hearn. Scalable shape analysis for systems code. In *CAV*, volume 5123 of *Lecture Notes in Computer Science*, pages 385–398. Springer, 2008.

A Details on the upper-bounds of $\mathcal{A}\mathsf{TEST}(\mathsf{X},\alpha)$ formulæ

In this section we provide the inequalities and recurrence systems that have been used to compute the upper-bounds for the test formulæ $\mathcal{A}\mathsf{TEST}(\mathsf{X},\alpha)$ introduced in Section 4.2. Note that the recurrence systems considered are generally more constrained that the inequalities that we want to satisfy. This is however not a problem, as we just need to find a possible solution to the inequalities, and our recurrence systems provide exactly that. For the cases of $\#\mathtt{loop}_{\mathsf{X}}(\beta_1) \geq \beta_2$ and $\#\mathtt{loop}^{\Uparrow}_{\mathsf{X}} \geq \beta_2$ we refer the reader to the body of the paper. Let (s,h,l) be a memory state, $h_1 + h_2 = h$ and $\alpha, \alpha_1, \alpha_2 \in \mathbb{N}^+$ s.t. $\alpha = \alpha_1 + \alpha_2$ (as in Lemma 8).

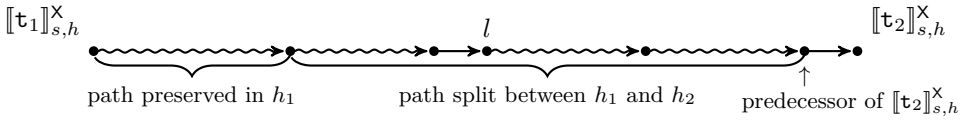
- Bound for $\mathtt{size}^{\mathcal{A}}_{\mathsf{X}} \geq \beta$ formulæ. For $\mathcal{A}\mathsf{TEST}(\mathsf{X},\alpha)$, let us call this upper-bound $\mathcal{S}(\alpha)$, as done in the body of the paper.
  - Inequality: $\mathcal{S}(\alpha) \geq \max_{\alpha=\alpha_1+\alpha_2}(\mathcal{S}(\alpha_1) + \mathcal{S}(\alpha_2))$.
  - Recurrence system: $\{\mathcal{S}(1) = 1,\ \mathcal{S}(\alpha+1) = \mathcal{S}(\alpha) + 1\}$.
  - Solution: $\mathcal{S}(\alpha) = \alpha$.
  - Informal explanation: every location in $\mathcal{A}\mathsf{Size}^{\mathsf{X}}_{s,h}(\alpha)$ can only appear in $\mathcal{A}\mathsf{Size}^{\mathsf{X}}_{s,h_1}(\alpha_1)$ or $\mathcal{A}\mathsf{Size}^{\mathsf{X}}_{s,h_2}(\alpha_2)$.
- Bound for $\#\mathtt{pred}^{\mathcal{A}}_{\mathsf{X}} \geq \beta$ formulæ. For $\mathcal{A}\mathsf{TEST}(\mathsf{X},\alpha)$, let us call this upper-bound $\mathsf{Pred}(\alpha)$.
  - Inequality: $\mathsf{Pred}(\alpha) \geq \max_{\alpha=\alpha_1+\alpha_2}(\mathsf{Pred}(\alpha_1) + \mathsf{Pred}(\alpha_2))$.
  - Recurrence system: $\{\mathsf{Pred}(1) = 1,\ \mathsf{Pred}(\alpha+1) = \mathsf{Pred}(\alpha) + 1\}$.
  - Solution: $\mathsf{Pred}(\alpha) = \alpha$.
  - Informal explanation: similarly to the previous case, every location in $\mathcal{A}\mathsf{Pred}^{\mathsf{X}}_{s,h}(\ell)$ can only appear in $\mathcal{A}\mathsf{Pred}^{\mathsf{X}}_{s,h_1}(\ell)$ or $\mathcal{A}\mathsf{Pred}^{\mathsf{X}}_{s,h_2}(\ell)$, for every $\ell \in s(\mathsf{X})$.
- Bound for $\overrightarrow{\beta}$ in $\mathtt{u} \in \mathtt{sees}_{\mathsf{X}}(\mathtt{t}_1,\mathtt{t}_2) \geq (\overleftarrow{\beta}, \overrightarrow{\beta})$ formulæ. For $\mathcal{A}\mathsf{TEST}(\mathsf{X},\alpha)$, let us call this upper-bound $\mathsf{Right}(\alpha)$.
  - Inequality: $\mathsf{Right}(\alpha) \geq \max_{\alpha=\alpha_1+\alpha_2}(\mathsf{Right}(\max(\alpha_1,\alpha_2)) + \mathcal{S}(\alpha_1) + \mathcal{S}(\alpha_2) + 1)$.
  - Recurrence system: $\{\mathsf{Right}(1) = 2,\ \mathsf{Right}(\alpha+1) = \mathsf{Right}(\alpha) + (\alpha+1) + 1\}$.
  - Solution: $\mathsf{Right}(\alpha) = \frac{1}{2}\alpha(\alpha+3)$.
  - Informal explanation: the semantics of the formula $\mathtt{u} \in \mathtt{sees}_{\mathsf{X}}(\mathtt{t}_1,\mathtt{t}_2) \geq (\overleftarrow{\beta}, \overrightarrow{\beta})$ implies that there is a path from $[\![\mathtt{t}_1]\!]^{\mathsf{X}}_{s,h}$ to $[\![\mathtt{t}_2]\!]^{\mathsf{X}}_{s,h}$ that goes through l and is such that between l and $[\![\mathtt{t}_2]\!]^{\mathsf{X}}_{s,h}$ there are at least $\overrightarrow{\beta}$ locations. Suppose $\alpha_1 \geq \alpha_2$ (therefore, we focus our attention on h_1). Then, when the heap h is split into h_1 and h_2 it can hold that:
    - $[\![\mathtt{t}_1]\!]^{\mathsf{X}}_{s,h}$ is a labelled location of (s,h_1,l);
    - in h_1, the location l is still reachable from $[\![\mathtt{t}_1]\!]^{\mathsf{X}}_{s,h}$, as shown in the picture below.



In this case, the set of locations that are reachable in h_1 from l still counts for the satisfaction of a $\mathtt{u} \in \mathtt{sees}_{\mathsf{X}}(\mathtt{t}_1,\mathtt{t}') \geq (\overleftarrow{\beta}, \overrightarrow{\beta})$ predicate, again with respect to $\overrightarrow{\beta}$. This justifies $\mathsf{Right}(\max(\alpha_1,\alpha_2))$ addend of the inequality above. In the figure, notice how there there is a part of the path whose locations are split between h_1 and h_2. These locations can only appear in $\mathcal{A}\mathsf{Size}^{\mathsf{X}}_{s,h_1}(\alpha_1)$ or $\mathcal{A}\mathsf{Size}^{\mathsf{X}}_{s,h_2}(\alpha_2)$ and justify the addition of $\mathcal{S}(\alpha_1) + \mathcal{S}(\alpha_2)$ to the inequality. Lastly, if $[\![\mathtt{t}_2]\!]^{\mathsf{X}}_{s,h}$ corresponds to a program variable in X then its predecessor appears in $\mathcal{A}\mathsf{Pred}^{\mathsf{X}}_{s,h_1}([\![\mathtt{t}_2]\!]^{\mathsf{X}}_{s,h})$ or in $\mathcal{A}\mathsf{Pred}^{\mathsf{X}}_{s,h_2}([\![\mathtt{t}_2]\!]^{\mathsf{X}}_{s,h})$. For this reason, the inequality above contains a $+1$ addend. Then, the inequality is

maximal for $\alpha_1 = \alpha - 1$ and $\alpha_2 = 1$ or vice-versa (recall that we do not admit α_1 or α_2 to be equal to 0). From $\mathcal{S}(\alpha) = \alpha$ we obtain the recurrence system above with solution $\mathsf{Right}(\alpha) = \frac{1}{2}\alpha(\alpha+3)$. Notice that the base case for the recurrence system is $\mathsf{Right}(1) = 2$. Under the condition that $[\![\mathtt{t}_2]\!]^{\mathsf{X}}_{s,h} = s(\mathtt{x})$ for some $\mathtt{x} \in \mathsf{X}$, the test formulæ can always check if the path from l to $s(\mathtt{x})$ has at least length 2 with the conjunction $\mathtt{u} \in \mathtt{sees}_{\mathsf{X}}(\mathtt{t}_1, \mathtt{x}) \geq (1,1) \wedge \neg \mathtt{u} \in \#\mathtt{pred}^{\mathcal{A}}_{\mathsf{X}}(\mathtt{x})$. We therefore require $\mathsf{Right}(1)$ to be at least 2 so that the formula above can be simply expressed in $\mathcal{A}\mathsf{TEST}(\mathsf{X}, 1)$ as $\mathtt{u} \in \mathtt{sees}_{\mathsf{X}}(\mathtt{t}_1, \mathtt{x}) \geq (1,2)$. This makes the proof of Lemma 8 easier as, with respect to $\overrightarrow{\beta}$, we can handle the test formulæ $\mathtt{u} \in \mathtt{sees}_{\mathsf{X}}(\mathtt{t}_1, \mathtt{t}_2) \geq (\overleftarrow{\beta}, \overrightarrow{\beta})$ without looking at the satisfaction of other test formulæ.

- Bounds for $\mathtt{sees}_{\mathsf{X}}(\mathtt{t}_1, \mathtt{t}_2) \geq \beta$ formulæ and $\overleftarrow{\beta}$ in $\mathtt{u} \in \mathtt{sees}_{\mathsf{X}}(\mathtt{t}_1, \mathtt{t}_2) \geq (\overleftarrow{\beta}, \overrightarrow{\beta})$ formulæ. For $\mathcal{A}\mathsf{TEST}(\mathsf{X}, \alpha)$, we denote the first upper-bound with $\mathsf{Sees}(\alpha)$ whereas the second upper-bound is called $\mathsf{Left}(\alpha)$.
  - Inequalities: $\mathsf{Sees}(\alpha) \geq \mathsf{Left}(\alpha) + \mathsf{Right}(\alpha)$;
    $$\mathsf{Left}(\alpha) + \mathsf{Right}(\alpha) \geq \max_{\alpha=\alpha_1+\alpha_2}(\mathsf{Sees}(\max(\alpha_1+\alpha_2)) + 1 + \mathcal{S}(\alpha_1) + \mathcal{S}(\alpha_2) + 1)$$
  - Recurrence system:
    $$\left\{\begin{array}{l} \mathsf{Left}(1) = 2,\ \mathsf{Left}(\alpha+1) = \mathsf{Sees}(\alpha) + 1, \\ \mathsf{Sees}(\alpha) = \mathsf{Left}(\alpha) + \mathsf{Right}(\alpha) \end{array}\right\}$$
  - Solution: $\mathsf{Left}(\alpha) = \frac{1}{6}\alpha(\alpha+1)(\alpha+2)+1$ and $\mathsf{Sees}(\alpha) = \frac{1}{6}(\alpha+1)(\alpha+2)(\alpha+3)$.
  - Informal explanation: first of, it is easy to see that $\mathsf{Sees}(\alpha) \geq \mathsf{Left}(\alpha) + \mathsf{Right}(\alpha)$ must hold. Indeed if a memory state satisfies the test formula $\mathtt{u} \in \mathtt{sees}_{\mathsf{X}}(\mathtt{t}_1, \mathtt{t}_2) \geq (\overleftarrow{\beta}, \overrightarrow{\beta})$ then by definition it also satisfies $\mathtt{sees}_{\mathsf{X}}(\mathtt{t}_1, \mathtt{t}_2) \geq \overleftarrow{\beta} + \overrightarrow{\beta}$. Then, the latter formula needs to be in $\mathcal{A}\mathsf{TEST}(\mathsf{X}, \alpha)$ (otherwise, for instance, Lemma 9 does not hold). The first inequality takes care of this. In the following, we strength this inequality to $\mathsf{Sees}(\alpha) = \mathsf{Left}(\alpha) + \mathsf{Right}(\alpha)$, as done for the recurrence system.
    For the second inequality, the semantics of the formula $\mathtt{u} \in \mathtt{sees}_{\mathsf{X}}(\mathtt{t}_1, \mathtt{t}_2) \geq (\overleftarrow{\beta}, \overrightarrow{\beta})$ implies that there is a path from $[\![\mathtt{t}_1]\!]^{\mathsf{X}}_{s,h}$ to $[\![\mathtt{t}_2]\!]^{\mathsf{X}}_{s,h}$ that goes through l and is such that between $[\![\mathtt{t}_1]\!]^{\mathsf{X}}_{s,h}$ and l there are at least $\overleftarrow{\beta}$ locations. Suppose $\alpha_1 \geq \alpha_2$ (therefore, we focus our attention on h_1). When the heap h is split into h_1 and h_2 it can hold that:
    * $[\![\mathtt{t}_1]\!]^{\mathsf{X}}_{s,h}$ is a labelled location of (s, h_1, l) and is such that $[\![\mathtt{t}_1]\!]^{\mathsf{X}}_{s,h_1} \in \mathrm{dom}(h_1)$;
    * in h_1, l is no longer reachable from $[\![\mathtt{t}_1]\!]^{\mathsf{X}}_{s,h}$, as shown in the picture below.



    In this case, the set of locations that are reachable in h_1 from $[\![\mathtt{t}_1]\!]^{\mathsf{X}}_{s,h}$ counts for the satisfaction of a $\mathtt{sees}_{\mathsf{X}}(\mathtt{t}_1, \mathtt{t}_2) \geq \beta$ predicate. This justifies $\mathsf{Sees}(\max(\alpha_1, \alpha_2))$ addend of the second inequality. In the figure, notice how there there is a part of the path whose locations are split between h_1 and h_2. These locations can only appear in $\mathcal{A}\mathsf{Size}^{\mathsf{X}}_{s,h_1}(\alpha_1)$ or $\mathcal{A}\mathsf{Size}^{\mathsf{X}}_{s,h_2}(\alpha_2)$ and justify the addition of $\mathcal{S}(\alpha_1) + \mathcal{S}(\alpha_2)$ to the inequality. If $[\![\mathtt{t}_2]\!]^{\mathsf{X}}_{s,h}$ corresponds to a program variable in X then its predecessor appears in $\mathcal{A}\mathsf{Pred}^{\mathsf{X}}_{s,h_1}([\![\mathtt{t}_2]\!]^{\mathsf{X}}_{s,h})$ or in $\mathcal{A}\mathsf{Pred}^{\mathsf{X}}_{s,h_2}([\![\mathtt{t}_2]\!]^{\mathsf{X}}_{s,h})$. For this reason, the inequality above contains a $+1$ addend. A last $+1$ is instead added to handle a corner case, regarding l, in the proof of Lemma 8. Then, the second inequality is maximal for

$\alpha_1 = \alpha - 1$ and $\alpha_2 = 1$ or vice versa (recall that we do not admit α_1 or α_2 to be equal to 0). We then obtain

$$\mathsf{Left}(\alpha) + \mathsf{Right}(\alpha) \geq \mathsf{Sees}(\alpha - 1) + 1 + \mathcal{S}(\alpha) + 1$$

As $\mathsf{Right}(\alpha) \geq \mathcal{S}(\alpha) + 1$, to find a solution is then sufficient consider the recurrence system above. Notice that the base case for the recurrence system is $\mathsf{Left}(1) = 2$. As for the previous case, this is done to ease the proof of Lemma 8, as we can then handle the $\mathtt{u} \in \mathtt{sees}_{\mathsf{X}}(\mathtt{t}_1, \mathtt{t}_2) \geq (\overleftarrow{\beta}, \overrightarrow{\beta})$ formulæ without looking at the satisfaction of other test formulæ.

B Lemma 8: the case of $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s,h}(\alpha)$

In this section we complete the construction started in Section 4.2. Let (s, h, l) and (s', h', l') be two memory states. Moreover let $\alpha, \alpha_1, \alpha_2 \in \mathbb{N}^+$ so that $\alpha = \alpha_1 + \alpha_2$ and $h = h_1 + h_2$. In the following, the index k stands for 1 or 2. For h_1 and h_2, we define the three following sets

$$\mathsf{L}_1 \stackrel{\text{def}}{=} \{\mathsf{P} \in \mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s,h}(\alpha) \mid \mathsf{P} \subseteq \mathrm{dom}(h_1)\} \qquad \mathsf{L}_2 \stackrel{\text{def}}{=} \{\mathsf{P} \in \mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s,h}(\alpha) \mid \mathsf{P} \subseteq \mathrm{dom}(h_2)\}$$

$$\mathsf{S} \stackrel{\text{def}}{=} \{(\mathsf{P}_1, \mathsf{P}_2) \mid \mathsf{P}_1 \cup \mathsf{P}_2 \in \mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s,h}(\alpha),\ \mathsf{P}_1 \neq \emptyset \neq \mathsf{P}_2 \text{ and for each } k \in \{1,2\}\ \mathsf{P}_k \subseteq \mathrm{dom}(h_k)\}$$

Then, L_k is the set of loops in $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s,h}(\alpha)$ that are completely assigned to h_k whereas S contains the loops of $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s,h}(\alpha)$ that are split between h_1 and h_2. Lastly, we denote with $\mathcal{L}(\alpha) \stackrel{\text{def}}{=} \frac{1}{2}\alpha(\alpha+3) - 1$ the upper-bound for $\#\mathtt{loop}^{\Uparrow}_{\mathsf{X}} \geq \beta$ formulæ and with $\mathcal{S}(\alpha) \stackrel{\text{def}}{=} \alpha$ the one for $\mathtt{size}^{\mathcal{A}}_{\mathsf{X}} \geq \beta'$ formulæ, as introduced in Section 4.2.

The result that we want to prove is the following:

Hypothesis:

H1. $\min(\mathcal{L}(\alpha), \mathrm{card}(\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s,h}(\alpha))) = \min(\mathcal{L}(\alpha), \mathrm{card}(\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha)))$

H2. $l \in \mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s,h}(\alpha) \Leftrightarrow l' \in \mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha)$.

Thesis:

there are h'_1, h'_2 such that $h' = h'_1 + h'_2$ such that it is possible to define three sets $\mathsf{L}'_1 \subseteq \mathcal{P}(\mathrm{dom}(h'_1))$, $\mathsf{L}'_2 \subseteq \mathcal{P}(\mathrm{dom}(h'_2))$ and $\mathsf{S}' \subseteq \mathcal{P}(\mathrm{dom}(h'_1)) \times \mathcal{P}(\mathrm{dom}(h'_2))$ satisfying the following properties:

- L'_k is the set of loops in $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha)$ that are completely assigned to h'_k
- S' contains the loops of $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha)$ that are split between h'_1 and h'_2.
- there is $\mathsf{P} \in \mathsf{L}_k$ with $l \in \mathsf{P}$ if and only if there is $\mathsf{P}' \in \mathsf{L}'_k$ with $l' \in \mathsf{P}'$;
- there is $\mathsf{P} \in \mathsf{S}$ with $l \in \pi_k(\mathsf{P})$ if and only if there is $\mathsf{P}' \in \mathsf{S}'$ with $l' \in \pi_k(\mathsf{P}')$;
- $\min(\mathcal{L}(\alpha_k), \mathrm{card}(\mathsf{L}_k)) = \min(\mathcal{L}(\alpha_k), \mathrm{card}(\mathsf{L}'_k))$;
- $\min(\mathcal{S}(\alpha_k), \mathrm{card}(\bigcup_{\mathsf{P}\in\mathsf{S}} \pi_k(\mathsf{P}))) = \min(\mathcal{S}(\alpha_k), \mathrm{card}(\bigcup_{\mathsf{P}'\in\mathsf{S}'} \pi_k(\mathsf{P}')))$.

Recall that, by definition of the upper-bounds, $\mathcal{L}(\alpha) \geq \mathcal{L}(\alpha_1) + \mathcal{L}(\alpha_2) + \mathcal{S}(\max(\alpha_1, \alpha_2)) + 1$. Moreover, note that from $\alpha = \alpha_1 + \alpha_2$ with $\alpha_1, \alpha_2 \in \mathbb{N}^+$ we obtain that $\alpha \geq 2$ and every set in $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s,h}(\alpha)$ and $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha)$ contains at least three locations. The construction goes by cases depending on whether or not the cardinalities of L_1, L_2 and S are less than $\mathcal{L}(\alpha_1)$, $\mathcal{L}(\alpha_2)$ and $\mathcal{S}(\max(\alpha_1, \alpha_2))$ respectively. For each case we also need to deal with whether or not l belongs to a set in $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s,h}(\alpha)$. When we define a set among L'_1, L'_2 and S' we implicitly assume that their locations are assigned to h'_1 and h'_2 accordingly to the definitions of these sets (i.e. L'_1 and $\pi_1(\mathsf{S}')$ only contain locations of h'_1 whereas L'_2 and $\pi_2(\mathsf{S}')$ only contain locations of h'_2).

1. **Case:** $\mathrm{card}(\mathsf{L}_1) < \mathcal{L}(\alpha_1)$, $\mathrm{card}(\mathsf{L}_2) < \mathcal{L}(\alpha_2)$ and $\mathrm{card}(\mathsf{S}) < \mathcal{S}(\max(\alpha_1, \alpha_2))$.
   By hypothesis **H1** and the definition of $\mathcal{L}(\alpha)$, it holds that
   $$\mathrm{card}(\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s,h}(\alpha)) = \mathrm{card}(\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha)).$$
   - Select $\mathrm{card}(\mathsf{L}_1)$ sets from $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha)$. If l appears in one of the sets of L_1, then by hypothesis **H2** l' belongs to one the sets of $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha)$ and we require that set to be among the selected ones. These sets constitute L'_1.
   - Select $\mathrm{card}(\mathsf{L}_2)$ sets from $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha) \setminus \mathsf{L}'_1$. If l appears in one of the sets of L_2, then by hypothesis **H2** l' belongs to one the sets of $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha)$ and we require that set to be among the selected ones. These sets constitute L'_2.
   - By $\mathrm{card}(\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s,h}(\alpha)) = \mathrm{card}(\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha))$ it follows that
     $$\mathrm{card}(\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha) \setminus (\mathsf{L}'_1 \cup \mathsf{L}'_2)) = \mathrm{card}(\mathsf{S}).$$

   Let $\mathfrak{f}$ be an injection between S and $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha) \setminus (\mathsf{L}'_1 \cup \mathsf{L}'_2)$ such that if there is $\mathsf{P} \in \mathsf{S}$ with $l \in \mathsf{P}$ then $l' \in \mathfrak{f}(\mathsf{P})$. This constraint on $\mathfrak{f}$ can be always satisfied thanks to **H2**. For each $\mathsf{P} \in \mathsf{S}$, divide $\mathfrak{f}(\mathsf{P})$ into two sets P'_1 and P'_2, as follows:
   - if $\mathrm{card}(\pi_1(\mathsf{P})) < \mathcal{S}(\alpha_1)$ then select $\mathrm{card}(\pi_1(\mathsf{P}))$ locations from $\mathfrak{f}(\mathsf{P})$. If $l \in \pi_1(\mathsf{P})$ then l' is one of the selected locations. These locations form P'_1. Then, $\mathsf{P}'_2 \stackrel{\text{def}}{=} \mathfrak{f}(\mathsf{P}) \setminus \mathsf{P}'_1$;
   - else if $\mathrm{card}(\pi_2(\mathsf{P})) < \mathcal{S}(\alpha_2)$ then select $\mathrm{card}(\pi_2(\mathsf{P}))$ locations from $\mathfrak{f}(\mathsf{P})$. If $l \in \pi_2(\mathsf{P})$ then l' is one of the selected locations. These locations form P'_2. Then, $\mathsf{P}'_1 \stackrel{\text{def}}{=} \mathfrak{f}(\mathsf{P}) \setminus \mathsf{P}'_2$;
   - otherwise $\mathrm{card}(\pi_1(\mathsf{P})) \geq \mathcal{S}(\alpha_1)$ and $\mathrm{card}(\pi_2(\mathsf{P})) \geq \mathcal{S}(\alpha_2)$. Select $\mathcal{S}(\alpha_1)$ locations from $\mathfrak{f}(\mathsf{P})$. If $l \in \pi_1(\mathsf{P})$ then l' is one of the selected locations. These locations form P'_1. Then, $\mathsf{P}'_2 \stackrel{\text{def}}{=} \mathfrak{f}(\mathsf{P}) \setminus \mathsf{P}'_1$.

   Iteratively add $(\mathsf{P}'_1, \mathsf{P}'_2)$ to S'.
2. **Case:** $\mathrm{card}(\mathsf{L}_1) < \mathcal{L}(\alpha_1)$, $\mathrm{card}(\mathsf{L}_2) < \mathcal{L}(\alpha_2)$ and $\mathrm{card}(\mathsf{S}) \geq \mathcal{S}(\max(\alpha_1, \alpha_2))$.
   - Select $\mathrm{card}(\mathsf{L}_1)$ sets from $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha)$. If l appears in one of the sets of L_1, then by hypothesis **H2** l' belongs to one the sets of $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha)$ and we require that set to be among the selected ones. These sets constitute L'_1.
   - Select $\mathrm{card}(\mathsf{L}_2)$ sets from $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha) \setminus \mathsf{L}'_1$. If l appears in one of the sets of L_2, then by hypothesis **H2** l' belongs to one the sets of $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha)$ and we require that set to be among the selected ones. These sets constitute L'_2.
   - By **H1**, $\mathrm{card}(\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha) \setminus (\mathsf{L}'_1 \cup \mathsf{L}'_2)) \geq \mathcal{S}(\max(\alpha_1, \alpha_2))$. For each set $\{\ell_1, \ell_2, \cdots, \ell_n\}$ of $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha) \setminus (\mathsf{L}'_1 \cup \mathsf{L}'_2)$, iteratively add $(\{\ell_1\}, \{\ell_2, \cdots, \ell_n\})$ to S' while being careful that if l' is in the set then $l' = \ell_1$ if and only if there is $\mathsf{P} \in \mathsf{S}$ such that $l \in \pi_1(\mathsf{P})$. This constraint is always satisfiable thanks to **H2**. Notice how, after this step, S' has at least $\mathcal{S}(\max(\alpha_1, \alpha_2))$ elements.
3. **Case:** $\mathrm{card}(\mathsf{L}_1) < \mathcal{L}(\alpha_1)$, $\mathrm{card}(\mathsf{L}_2) \geq \mathcal{L}(\alpha_2)$ and $\mathrm{card}(\mathsf{S}) < \mathcal{S}(\max(\alpha_1, \alpha_2))$.
   - Select $\mathrm{card}(\mathsf{L}_1)$ sets from $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha)$. If l appears in one of the sets of L_1, then by hypothesis **H2** l' belongs to one the sets of $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha)$ and we require that set to be among the selected ones. These sets constitute L'_1.
   - Let $\mathfrak{f}$ be an injection between S and $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha) \setminus \mathsf{L}'_1$ such that if there is $\mathsf{P} \in \mathsf{S}$ with $l \in \mathsf{P}$ then $l' \in \mathfrak{f}(\mathsf{P})$. This constraint on $\mathfrak{f}$ can be always satisfied thanks to **H2**. For each $\mathsf{P} \in \mathsf{S}$, divide $\mathfrak{f}(\mathsf{P})$ into two sets P'_1 and P'_2 as shown in the third step of the 1st case of the construction. Iteratively add $(\mathsf{P}'_1, \mathsf{P}'_2)$ to S'.
   - The set $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha) \setminus (\mathsf{L}'_1 \cup \mathsf{S}')$ constitutes L'_2.
4. **Case:** $\mathrm{card}(\mathsf{L}_1) < \mathcal{L}(\alpha_1)$, $\mathrm{card}(\mathsf{L}_2) \geq \mathcal{L}(\alpha_2)$ and $\mathrm{card}(\mathsf{S}) \geq \mathcal{S}(\max(\alpha_1, \alpha_2))$.

- Select $\mathrm{card}(\mathsf{L}_1)$ sets from $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha)$. If l appears in one of the sets of L_1, then by hypothesis **H2** l' belongs to one the sets of $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha)$ and we require that set to be among the selected ones. These sets constitute L'_1.
- Select $\mathcal{L}(\alpha_2)$ sets from $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha) \setminus \mathsf{L}'_1$. If l appears in one of the sets of L_2, then by hypothesis **H2** l' belongs to one the sets of $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha)$ and we require that set to be among the selected ones. These sets constitute L'_2.
- By **H1**, $\mathrm{card}(\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha) \setminus (\mathsf{L}'_1 \cup \mathsf{L}'_2)) \geq \mathcal{S}(\max(\alpha_1, \alpha_2))$. For each set $\{\ell_1, \ell_2, \cdots, \ell_n\}$ of $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha) \setminus (\mathsf{L}'_1 \cup \mathsf{L}'_2)$, iteratively add $(\{\ell_1\}, \{\ell_2, \cdots, \ell_n\})$ to S' while being careful that if l' is in the set then $l' = \ell_1$ if and only if there is $\mathsf{P} \in \mathsf{S}$ such that $l \in \pi_1(\mathsf{P})$. This constraint is always satisfiable thanks to **H2**. Notice how, after this step, S' has at least $\mathcal{S}(\max(\alpha_1, \alpha_2))$ elements.

5. **Case:** $\mathrm{card}(\mathsf{L}_1) \geq \mathcal{L}(\alpha_1)$, $\mathrm{card}(\mathsf{L}_2) < \mathcal{L}(\alpha_2)$ and $\mathrm{card}(\mathsf{S}) < \mathcal{S}(\max(\alpha_1, \alpha_2))$. Symmetrical to the 3rd case with respect to the indexes 1 and 2.
6. **Case:** $\mathrm{card}(\mathsf{L}_1) \geq \mathcal{L}(\alpha_1)$, $\mathrm{card}(\mathsf{L}_2) < \mathcal{L}(\alpha_2)$ and $\mathrm{card}(\mathsf{S}) \geq \mathcal{S}(\max(\alpha_1, \alpha_2))$. Symmetrical to the 4rd case with respect to the indexes 1 and 2.
7. **Case:** $\mathrm{card}(\mathsf{L}_1) \geq \mathcal{L}(\alpha_1)$, $\mathrm{card}(\mathsf{L}_2) \geq \mathcal{L}(\alpha_2)$ and $\mathrm{card}(\mathsf{S}) < \mathcal{S}(\max(\alpha_1, \alpha_2))$.
   - Select $\mathcal{L}(\alpha_1)$ sets from $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha)$. If l appears in one of the sets of L_1, then by hypothesis **H2** l' belongs to one the sets of $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha)$ and we require that set to be among the selected ones. These sets constitute L'_1.
   - Let $\mathfrak{f}$ be an injection between S and $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha) \setminus \mathsf{L}'_1$ such that if there is $\mathsf{P} \in \mathsf{S}$ with $l \in \mathsf{P}$ then $l' \in \mathfrak{f}(\mathsf{P})$. This constraint on $\mathfrak{f}$ can be always satisfied thanks to **H2**. For each $\mathsf{P} \in \mathsf{S}$, divide $\mathfrak{f}(\mathsf{P})$ into two sets P'_1 and P'_2 as shown in the third step of the 1st case of the construction. Iteratively add $(\mathsf{P}'_1, \mathsf{P}'_2)$ to S'.
   - The set $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha) \setminus (\mathsf{L}'_1 \cup \mathsf{S}')$ constitutes L'_2.
8. Case: $\mathrm{card}(\mathsf{L}_1) \geq \mathcal{L}(\alpha_1)$, $\mathrm{card}(\mathsf{L}_2) \geq \mathcal{L}(\alpha_2)$ and $\mathrm{card}(\mathsf{S}) \geq \mathcal{S}(\max(\alpha_1, \alpha_2))$.
   - Select $\mathcal{L}(\alpha_1)$ sets from $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha)$. If l appears in one of the sets of L_1, then by hypothesis **H2** l' belongs to one the sets of $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha)$ and we require that set to be among the selected ones. These sets constitute L'_1.
   - Select $\mathcal{L}(\alpha_2)$ sets from $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha) \setminus \mathsf{L}'_1$. If l appears in one of the sets of L_2, then by hypothesis **H2** l' belongs to one the sets of $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha)$ and we require that set to be among the selected ones. These sets constitute L'_2.
   - By **H1**, $\mathrm{card}(\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha) \setminus (\mathsf{L}'_1 \cup \mathsf{L}'_2)) \geq \mathcal{S}(\max(\alpha_1, \alpha_2))$. For each set $\{\ell_1, \ell_2, \cdots, \ell_n\}$ of $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s',h'}(\alpha) \setminus (\mathsf{L}'_1 \cup \mathsf{L}'_2)$, iteratively add $(\{\ell_1\}, \{\ell_2, \cdots, \ell_n\})$ to S' while being careful that if l' is in the set then $l' = \ell_1$ if and only if there is $\mathsf{P} \in \mathsf{S}$ such that $l \in \pi_1(\mathsf{P})$. This constraint is always satisfiable thanks to **H2**. Notice how, after this step, S' has at least $\mathcal{S}(\max(\alpha_1, \alpha_2))$ elements.

It is easy to see that all the properties are always satisfied for any of the cases above. In particular, for the cases where $\mathrm{card}(\mathsf{S}) \geq \mathcal{S}(\max(\alpha_1, \alpha_2))$, i.e. cases 2, 4, 6 and 8, the property

$$\min(\mathcal{S}(\alpha_k), \mathrm{card}(\textstyle\bigcup_{\mathsf{P} \in \mathsf{S}} \pi_k(\mathsf{P}))) = \min(\mathcal{S}(\alpha_k), \mathrm{card}(\textstyle\bigcup_{\mathsf{P}' \in \mathsf{S}'} \pi_k(\mathsf{P}')))$$

is implied by $\mathrm{card}(\mathsf{S}) \geq \mathcal{S}(\max(\alpha_1, \alpha_2))$ and $\mathrm{card}(\mathsf{S}') \geq \mathcal{S}(\max(\alpha_1, \alpha_2))$. To prove that this property holds also for the cases where $\mathrm{card}(\mathsf{S}) < \mathcal{S}(\max(\alpha_1, \alpha_2))$, i.e. cases 1, 3, 5 and 7, we use the fact that, for every $\mathsf{P} \in \mathsf{S}$ and $k \in \{1, 2\}$, $\pi_k(\mathsf{P}) < \mathcal{S}(\alpha_k)$ implies $\pi_{3-k}(\mathsf{P}) \geq \mathcal{S}(\alpha_{3-k})$. Indeed, this holds as all the loops in $\mathcal{A}\mathsf{Loop}\Uparrow^{\mathsf{X}}_{s,h}(\alpha)$ have size greater than $\alpha = \mathcal{S}(\alpha) = \mathcal{S}(\alpha_1) + \mathcal{S}(\alpha_2)$.