Logical Investigations on Separation Logics (ESSLLI 2015)

Stéphane Demri, Morgan Deters

LOGICAL INVESTIGATIONS ON SEPARATION LOGICS
(draft)

EUROPEAN SUMMER SCHOOL ON LOGIC, LANGUAGE AND INFORMATION
BARCELONA, SPAIN, AUGUST 2015

Stéphane Demri Morgan Deters

Lecture Notes
September 8, 2015

Foreword

In June 2014, with my colleague and friend Morgan Deters, we have submitted an advanced course for ESSLLI'15 about logical investigations on separation logics by focusing on expressiveness, computational complexity of reasoning tasks and decision procedures, some of them based on SMT technology. The submission has been accepted in the fall 2014. In november 2014, while Morgan visited the Laboratoire Spécification and Vérification (ENS Cachan), we discussed further the content of the lectures as well as the plan of the lecture notes. The current document is the fruit of a joint and intense effort to present in a single volume fundamental results about separation logics and to provide numerous bibliographical references for further study. Morgan passed away unexpectedly last january and our project to produce the exact document we had in mind became impossible. Morgan and I wrote several articles about separation logics and we had many discussions about the logical side of separation logics while I have been visiting New York University in 2012–2014. The current document is the outcome of our fruitful collaboration[1]*; it is partly inspired from the material in the papers [DD14, DD15b, DD15a]. All the mistakes are mine.*

To the memory of Morgan Deters.

Stéphane Demri
`demri@lsv.fr`
June 15th, 2015 — Cachan

[1]This work has been partially supported by the EU Seventh Framework Programme under grant agreement No. PIOF-GA-2011-301166 (DATAVERIF).

Contents

INTRODUCTION

Introducing new logics is always an uncertain enterprise since there must be sufficient interest to use new formalisms. In spite of this hurdle, we know several recent success stories. For instance, even though a pioneering work on symbolic modal logic by Lewis appeared in 1918 [Lew18], the first monographs on symbolic modal logic appear about fifty years later, see e.g. [HC68]. Nowadays, modal logic is divided into many distinct branches and remains one of the most active research fields in logic and computer science, see e.g. [BvBW06]. Additionally, the introduction of temporal logic to computer science, due to Pnueli [Pnu77], has been a major step in the development of model-checking techniques, see e.g. [CGP00, BBF+01]. This is now a well-established approach for the formal verification of computer systems: one models the system to be verified by a mathematical structure (typically a directed graph) and expresses behavioral properties in a logical formalism (typically a temporal logic). Verification by model-checking [CGP00] consists of developing algorithms whose goal is to verify whether the logical properties are satisfied by the abstract model. The development of description logics for knowledge representation has also followed a successful path, thanks to a permanent interaction between theoretical works, pushing even further the high complexity and undecidability borders, and more applied works dedicated to the design of new tools and the production of more and more applications, especially in the realm of ontology languages. The wealth of research on description logic is best illustrated by [BCM+03], in which can be found many chapters on theory, implementations, and applications.

It is well-known that modal logic, temporal logic, and description logic have many similarities even though each family has its own research agenda. For instance, models can be (finite or infinite) graphs, the classes of models range from concrete ones to more abstract ones, and any above-mentioned class includes a wide range of logics and fragments. In the present lecture notes, we deal with another class of logics, separation logic, that has been introduced quite recently

(see e.g. [IO01, Rey02]) and is the subject of tremendous interest, leading to many works on theory, tools and applications (mainly for the automatic program analysis). Any resemblance to modal, temporal, or description logic is certainly not purely coincidental—but separation logic also has its own assets.

In the possible-world semantics for modal logic, the connective $\Box$ [resp. $\Diamond$] corresponds to universal [resp. existential] quantification on successor worlds, and these are essential properties to be stated, partly explaining the impact of Kripke's discovery [Kri59, Cop02]. Similarly, the ability to divide a model in two disjoint parts happens to be a very natural property and this might explain the success of separation logic in which disjoint memory states can be considered, providing an elegant means to perform local reasoning. Separation is a key concept that has been already introduced in interval temporal logic ITL [Mos83] with the "chop" connective, and in many other logical formalisms such as in graph logics [Loz04a, DGG07] or in extensions of PDL (see e.g. [BdFV11, BT14a]). Moreover, dependence logic has also a built-in notion of separation, see e.g. [AV11, KMSV14, HLSV14]. Therefore, the development of separation logic can be partly explained by the relevance of the separation concept. Its impressive development can be also justified by the fact that separation logic extends Hoare logic for reasoning about programs with dynamic data structures, meeting also industrial needs as witnessed by the recent acquisition of Monoidics Ltd by Facebook (see e.g. [CDD$^+$15]).

Separation logic has been introduced as an extension of Hoare-Floyd logic (see e.g. [Hoa69, Apt81]) to verify programs with mutable data structures [IO01, Rey02]. A major feature is to be able to reason locally in a modular way, which can be performed thanks to the separating conjunction $*$ that allows one to state properties in disjoint parts of the memory. Moreover, the adjunct implication $-\!\!*$ asserts that whenever a fresh heap satisfies a property, its composition with the current heap satisfies another property. This is particularly useful when a piece of code mutates memory locally, and we want to state some property of the entire memory (such as the preservation of data structure invariants). In a sense, if modal logic is made for reasoning about necessity and possibility, separation logic is made for reasoning about separation and composition. As a taste of separation logic, it is worth observing that models can be finite graphs and the classes of models range from concrete ones (with heaps for instance) to very abstract ones.

Smallfoot was the first implementation to use separation logic, its goal to verify the extent to which proofs and specifications made by hand could be treated automatically [BCO05]. The automatic part is related to the assertion checking, but the user has to provide preconditions, postconditions, and loop invariants. A

major step has been then to show that the method is indeed scalable [YLB+08]. In a sense, the legitimate question about the practical utility of separation logic was quickly answered, leading to a new generation of tools such as Slayer developed by Microsoft Research, Space Invader [DOY06, YLB+08], and Infer [CD11] (still under development at Facebook [CDD+15, Section 4]). Actually, nowadays, many tools support separation logic as an assertion language (see e.g. [MIG14]) and, more importantly, in order to produce interactive proofs with separation logic, several proof assistants encode the logic, see e.g. [Tue11]. Furthermore, there exists also many tools that are dedicated to program verification and closely related to tools explicitly using separation logic, see e.g. a description of the research prototype VeriFast in [VJP15] (typically featherweight VeriFast and Smallfoot share a very similar programming language). Note also that the development of the different tools has been performed progressively; Whereas Smallfoot uses an assertion language for preconditions, postconditions and loop invariants, SmallfootRG [VP07] goes beyond by inferring some loop invariants (which is apart from the introduction of rules for dealing the the magic wand operator). Space Invader [DOY06] extends further the ideas of Smallfoot by determining annotations for unannotated programs.

From the very beginning, the theory of separation logic has been an important research thread even if not always related to automatic verification. This is not very surprising since separation logic can be understood as a concretisation of the logic BI of bunched implications which is a general logic of resource with a nice proof theory [OP99]. More precisely, the logic BI exists in different flavours: its intuitionistic version has additive and multiplicative connectives that behave intuitionistically whereas its Boolean version admits Boolean additive connectives with intuitionistic multiplicative connectives ($*$ and $-\!\!*$), see more details in [LG13]. So, separation logic is rather a concretisation of Boolean BI (see more details in Section 1.3.1).

Besides, as for modal and temporal logics, the relationships between separation logic, and first-order or second-order logics have been the source of many characterisations and works. This is particularly true since the separating connectives are second-order in nature, see e.g. [Loz04a, KR04, CGH05, BDL12]. For instance, separation logic is equivalent to a Boolean propositional logic [Loz04b, Loz04a] if first-order quantifiers are disabled. Similarly, the complexity of satisfiability and model-checking problems for separation logic fragments have been quite studied [COY01, Rey02, CHO+11, AGH+14, BFGN14]. In [COY01], the model-checking and satisfiability problems for propositional separation logic are shown PSPACE-complete; this is done by proving a small model property.

In this course, we would like to emphasise the similarities between separation logic and, modal and temporal logics. Our intention is to pinpoint the common features in terms of models, proof techniques, motivations, decision procedures. Second, we wish to present landmark results about decidability, complexity and expressive power. These are standard themes for studying logics in computer science and we deliberately focus on the logical side of separation logic. Even though our intention is to produce a self-contained document as far as the definitions and results are concerned, we invite the reader to consult surveys on formal verification and separation logic, see e.g., the primer on separation logic in [O'H12], the lecture notes about Hoare logic and separation logic in [Gor14] or [Jen13a, Chapter 7] and [Jen13b]. See also [VJP15] for a detailed description of the tool featherweight VeriFast.

The five lectures are organised as follows and each chapter is dedicated to one lecture.

Lecture 1: First steps in separation logics.

Lecture 2: Propositional separation logics.

Lecture 3: Expressiveness of first-order separation logics.

Lecture 4: Relationships with other logics.

Lecture 5: Decision procedures.

Because of time and space limitations, we had to focus on core separation logic and for the presentation of the main results we adopt a puristic point of view. Namely, most of the logics

- are without data values (by contrast, see e.g. [BDES09, BBL09, MPQ11]),
- use concrete models (by contrast to abstract models considered in [COY07, BK10, LWG10, BV14]),
- are not multi-dimensional extensions of non-classical logics (by contrast, see e.g. [YRSW03, BDL09, CG13]),
- do not provide general inductive predicates (lists, trees, etc.) (by contrast, see e.g. [IRS13, BFGN14]).

However, these extensions shall be introduced and briefly discussed but we shall refer to original articles or surveys for in-depth developments.

Chapter 1

First Steps in Separation Logics

Contents

In this chapter, we provide a brief introduction to separation logic by showing how it is related to formal verification. Later sections give a precise definition and focus on the logical language rather than on the verification process. This means that we adopt a restrictive use of the term 'separation logic' which is understood as an assertion logic, rather than an understanding combining in some way the assertion logic, the programming language and/or the specification logic. Section 1.1.1 is dedicated to Floyd-Hoare logic understood as a proof system made of deduction rules to verify the correctness of programs. Section 1.1.4 recalls how separation logic appears as a way to repair the defects of Floyd-Hoare logic when pointers are involved. Relationships with the logic of bunched implications logic BI are also briefly explained. In Section 1.2.1, we present the syntax and semantics for the versions of separation logics considered in this document. Section 1.2.2 explains how to express properties with formulae from separation logics whereas Section 1.2.4 provides a classification of formulae involving pure, intuitionistic and strictly exact formulae, respectively. Section 1.2.3 presents a few rules in a Floyd-Hoare-like proof system when commands for mutable shared data structures are involved and when the assertion language uses formulae from separation logic. Section 1.2.5 presents some more decision problems for separation logics. Section 1.3 provides first insights about the relationships between separation logics and other non-classical logics. This shall be complemented by material in the subsequent chapters, as it is done, for instance, in Chapter 4.

Highlights of the chapter

1. Definition of separation logics following [IO01, Rey02] in which the set of addresses/values is equal to $\mathbb{N}$.
2. Translation of kSL into weak monadic second-order logic by internalising the semantics.
3. Undecidability proof of 2SL (without separating connectives) by reduction from the finitary satisfiability problem for predicate logic restricted to a unique binary predicate symbol (Theorem 1.3.4) [COY01].

1.1 Floyd-Hoare Logic and Separation Logic

1.1.1 Hoare triples

Hoare logic, proposed in 1969 by Tony Hoare [Hoa69] and inspired by the earlier work of Floyd [Flo67], is a formal system used to show the correctness of programs (see a quite complete survey in [Apt81], the recent lecture notes [Gor14] or the extension to parallel programming languages in [OG76]). This is an axiomatic method that had a substantial impact for the design and the verification of computer programs. Its hallmark, the **Hoare triple**, is composed of assertions φ and ψ and the command $\mathtt{C}$:

$$\{\varphi\}\ \mathtt{C}\ \{\psi\}.$$

Simply put, such a triple means that given a program state where the **precondition** φ holds, the execution of $\mathtt{C}$ yields a state in which the **postcondition** ψ holds. Two commands can be composed:

$$\frac{\{\varphi\}\ \mathtt{C}_1\ \{\psi\} \quad \{\psi\}\ \mathtt{C}_2\ \{\chi\}}{\{\varphi\}\ \mathtt{C}_1;\mathtt{C}_2\ \{\chi\}}\ \text{composition}$$

Similarly, the `skip` has no effect.

$$\frac{}{\{\varphi\}\ \mathtt{skip}\ \{\varphi\}}\ \text{skip}$$

Preconditions can be strengthened and postconditions can be weakened in a natural fashion:

$$\frac{\varphi \Rightarrow \varphi' \quad \{\varphi'\}\ \mathtt{C}\ \{\psi\} \quad \psi \Rightarrow \psi'}{\{\varphi\}\ \mathtt{C}\ \{\psi'\}}\ \text{strengthen/weaken}$$

The expression $\varphi \Rightarrow \varphi'$ can be read as φ entails φ', which amounts to state the logical validity of the formula $\varphi \Rightarrow \varphi'$, when defined in a first-order dialect. An assignment axiom schema is stated simply:

$$\frac{}{\{\varphi[\mathtt{e}/\mathtt{x}]\}\ \mathtt{x} := \mathtt{e}\ \{\varphi\}}\ \text{assignment}$$

In general, atomic commands, such as the assignment, are axiomatised by so-called **small axioms**.

Example 1.1.1. Here are examples of "valid" triples that are related to assignment but that are not instances of the assignment axiom schema.

- $\{\mathtt{x} = 1\}\ \mathtt{x} := \mathtt{x} + 2\ \{\mathtt{x} \geq 2\}$.
- $\{\mathtt{x} = 1\}\ \mathtt{x} := \mathtt{x} + 2\ \{\mathtt{x} = 3\}$.
- $\{\mathtt{x} = 1\}\ \mathtt{x} := \mathtt{x} + 2\ \{\exists\, \mathtt{y}, \mathtt{z}\ (\mathtt{y} \cdot \mathtt{z} > 0 \wedge \mathtt{x} = \mathtt{y} \cdot \mathtt{z} - 1)\}$.

By contrast, the triple below is an instance:

$$\{\mathtt{x} + 2 = 3\}\ \mathtt{x} := \mathtt{x} + 2\ \{\mathtt{x} = 3\}.$$

Here is another deduction rule (that remain sounds even when the new commands introduced below are considered):

$$\frac{\{\varphi\}\ \mathtt{C}\ \{\psi\}}{\{\exists\, \mathtt{u}\ \varphi\}\ \mathtt{C}\ \{\exists\, \mathtt{u}\ \psi\}}\ \text{auxiliary variable elimination}$$

assuming that $\mathtt{u}$ is not free in $\mathtt{C}$.

Additional deduction rules for the command **while** and **if-then-else** in Floyd-Hoare logic proof system can be defined as follows:

$$\frac{\{\varphi \wedge \mathtt{B}\}\ \mathtt{C}\ \{\varphi\}}{\{\varphi\}\ \textbf{while}\ \mathtt{B}\ \textbf{do}\ \mathtt{C}\ \{\varphi \wedge \neg \mathtt{B}\}}\ \text{while rule}$$

$$\frac{\{\varphi \wedge \mathtt{B}\}\ \mathtt{C}_1\ \{\psi\} \quad \{\varphi \wedge \neg \mathtt{B}\}\ \mathtt{C}_2\ \{\psi\}}{\{\varphi\}\ \textbf{if}\ \mathtt{B}\ \textbf{then}\ \mathtt{C}_1\ \textbf{else}\ \mathtt{C}_2\ \{\psi\}}\ \text{conditional rule}$$

Note that $\mathtt{B}$ is a Boolean expression in the programming language but it also belongs to the assertion language for preconditions and postconditions since the deduction rules handles formulae of the form $\varphi \wedge \mathtt{B}$ and $\varphi \wedge \neg \mathtt{B}$.

The **rule of constancy** can be also defined as follows:

$$\frac{\{\varphi\}\ \mathtt{C}\ \{\psi\}}{\{\varphi \wedge \psi'\}\ \mathtt{C}\ \{\psi \wedge \psi'\}}$$

where no variable free in ψ' is modified by $\mathtt{C}$.

An instance of the rule of constancy can be found below:

$$\frac{\{\mathtt{x} = 3\}\ \mathtt{x} := 4; \mathtt{z} := \mathtt{x}\ \{\mathtt{x} = 4\}}{\{\mathtt{x} = 3 \wedge \mathtt{y} = 8\}\ \mathtt{x} := 4; \mathtt{z} := \mathtt{x}\ \{\mathtt{x} = 4 \wedge \mathtt{y} := 8\}}$$

Note that $\mathtt{y}$ does not occur in $\mathtt{x} := 4; \mathtt{z} := \mathtt{x}$.

Given a formal semantics for a simple imperative programming language based on the above command, it is possible to state soundness and completeness properties (see e.g. [Apt81]). We do not provide here a formal semantics in terms of small-step operational semantics for the commands but it can be defined via a binary relation $\mathfrak{s}, \mathtt{C} \leadsto \mathfrak{s}', \mathtt{C}'$ that corresponds to one step in the computation. The expressions $\mathfrak{s}$ and $\mathfrak{s}'$ are assignments for the program variables and executing one step of C leads to $\mathfrak{s}'$ and it remains to compute C. **Termination** is expressed by $\mathfrak{s}, \mathtt{C} \leadsto^* \mathfrak{s}', \mathtt{skip}$ where $\leadsto^*$ denote the reflexive and transitive closure. With this approach, the following simple observations can be made:

1. any command except `skip` can execute in any state (possibly except `halt`),
2. `skip` alone represents the final step of execution of a program,
3. there is no possible runtime error.

Consequently, C does not terminate means that C diverges. When pointers are involved, failed dereference operations are possible and therefore nontermination of a command does not imply necessarily divergence.

Correctness of the proof system involving Hoare triples means that whenever a triple $\{\varphi\}\ \mathtt{C}\ \{\psi\}$ is derived, it is valid. This means that if $\mathfrak{s}, \mathtt{C} \leadsto^* \mathfrak{s}', \mathtt{skip}$ and $\mathfrak{s} \models \varphi$ (the formula from the assertion language is satisfied by the current variable assignment), then $\mathfrak{s}' \models \psi$. This type of correctness is called **partial**. By contrast, **total correctness** requires that if $\mathfrak{s} \models \varphi$, then $\mathfrak{s}, \mathtt{C} \leadsto^* \mathfrak{s}', \mathtt{skip}$ (termination) and $\mathfrak{s}' \models \psi$. Besides, **relative completeness** requires that every valid triple is derivable in the proof system. Of course, all these notions are relative to a programming language, to its semantics, to the assertion language for the pre/post-conditions and to the exact rules of the proof system.

1.1.2 Weakest preconditions

Relative completeness for Hoare logic has been established in [Coo78] by using **weakest preconditions** introduced by E.W. Dijkstra, see e.g. [Dij76]. A weakest precondition $\mathtt{wp}(\mathtt{C}, \psi)$ is a predicate that describes the exact set of states $\mathfrak{s}$ such that when C is started in $\mathfrak{s}$, if it terminates, then it terminates in a state satisfying ψ. In a sense, $\mathtt{wp}(\mathtt{C}, \psi)$ corresponds to the minimal precondition φ that validates $\{\varphi\}\ \mathtt{C}\ \{\psi\}$. So, Hoare logic is complete if the assertion logic Ω can express the weakest preconditions for any C and ψ. The proof for relative completeness from [Coo78] uses weakest preconditions and a structural induction on C.

Note that checking validity of $\{\varphi\}\ \mathtt{C}\ \{\psi\}$ amounts to perform the following tasks:

1. to compute $\mathtt{wp}(\mathtt{C}, \psi)$,
2. to check the validity of $\varphi \Rightarrow \mathtt{wp}(\mathtt{C}, \psi)$.

Of course, this implies that $\{\mathtt{wp}(\mathtt{C}, \psi)\}\ \mathtt{C}\ \{\psi\}$ is valid, which is expected in view of the specification for defining weakest preconditions. So, the completeness result for mono-procedure sequential programs proved in [Coo78] establishes that each triple $\{\mathtt{wp}(\mathtt{C}, \psi)\}\ \mathtt{C}\ \{\psi\}$ is derivable.

By way of example, we brievely explain how weakest preconditions can be defined inductively. Note that this assumes that the assertion logic has sufficient syntactic resources and that it contains appropriate logical connectives and quantifiers.

$$
\begin{array}{lcl}
\mathtt{wp}(\mathtt{skip}, \psi) & \stackrel{\text{def}}{=} & \psi \\
\mathtt{wp}(\mathtt{x} := \mathtt{e}, \psi) & \stackrel{\text{def}}{=} & \psi[\mathtt{e}/\mathtt{x}] \\
\mathtt{wp}(\mathtt{C}_1; \mathtt{C}_2, \psi) & \stackrel{\text{def}}{=} & \mathtt{wp}(\mathtt{C}_1, \mathtt{wp}(\mathtt{C}_2, \psi)) \\
\mathtt{wp}(\textbf{if } \mathtt{B} \textbf{ then } \mathtt{C}_1 \textbf{ else } \mathtt{C}_2, \psi) & \stackrel{\text{def}}{=} & (\mathtt{B} = \top \wedge \mathtt{wp}(\mathtt{C}_1, \psi)) \ \wedge \ (\mathtt{B} = \bot \ \wedge \mathtt{wp}(\mathtt{C}_2, \psi)) \\
\mathtt{wp}(\textbf{while } \mathtt{B} \textbf{ do } \mathtt{C}, \psi) & \stackrel{\text{def}}{=} & I \ \wedge \ \forall\, \mathtt{y}_1, \ldots, \mathtt{y}_k \\
 & & (((\mathtt{B} = \top \wedge I) \Rightarrow \mathtt{wp}(\mathtt{C}, I)) \wedge ((\mathtt{B} = \bot \wedge I) \Rightarrow \psi))[\mathtt{y}_i/\mathtt{x}_i]
\end{array}
$$

($\mathtt{x}_1, \ldots, \mathtt{x}_k$ are the assigned variables in $\mathtt{C}$)

Here, it is worth noting the necessity to annotate the program so that the invariant condition I is known before computing the weakest precondition, which is indeed problematic for a fully automated verification process.

1.1.3 Adding pointers

When pointers are added to the programming language (see an example below), soundness of the rule of constancy is not preserved, as briefly shown below. By way of example, we provide new commands for the manipulation of mutable shared data structures understood as an extension of an imperative programming language.

$\mathtt{x} := \textbf{cons}(\mathtt{e})$	allocation
$\mathtt{x} := [\mathtt{e}]$	lookup
$[\mathtt{e}] := \mathtt{e}'$	mutation
$\textbf{dispose}(\mathtt{e})$	deallocation

Again, we do not provide here a formal semantics in terms of small-step operational semantics for such commands but it is worth mentionning that computational states are extended so that a **store** is a variable assignment and a **heap** is understood as a map from the set of addresses into the set of values. A command of the form $\mathtt{x} := [\mathtt{e}]$ updates the store whereas a command of the form $[\mathtt{e}] := \mathtt{e}'$ updates the heap. Moreover, the domain of the heap is augmented after the execution of the command $\mathtt{x} := \mathbf{cons}(\mathtt{e})$ and it is reduced after the execution of the command $\mathbf{dispose}(\mathtt{e})$. In order to simplify technical developments, in the sequel, we assume that the set of adresses and the set of values are equal to the set of natural numbers. For instance,

$$(\mathfrak{s}, \mathfrak{h}), \mathtt{x} := \mathbf{cons}(\mathtt{e}) \ \leadsto \ (\mathfrak{s}[\mathtt{x} \mapsto n], \mathfrak{h} \uplus \{n \mapsto [\![\mathtt{e}]\!]\}), \mathtt{skip}$$

where $n \mapsto [\![\mathtt{e}]\!]$ is a new memory cell and $[\![\mathtt{e}]\!]$ denotes the interpretation of the expression $\mathtt{e}$ (parameterised by $(\mathfrak{s}, \mathfrak{h})$). This can be generalised to the command $\mathtt{x} := \mathbf{cons}(\mathtt{e}_1, \ldots, \mathtt{e}_s)$ whose effect is to create s new memory cells with consecutive addresses. Similarly, we have

$$(\mathfrak{s}, \mathfrak{h}), \mathbf{dispose}(\mathtt{y}) \ \leadsto \ (\mathfrak{s}, \mathfrak{h} \setminus \{[\![\mathtt{y}]\!] \mapsto m\}), \mathtt{skip}$$

where $[\![\mathtt{y}]\!]$ belongs to the domain of $\mathfrak{h}$, $\mathfrak{h}([\![\mathtt{y}]\!]) = m$, and $\mathfrak{h} \setminus \{[\![\mathtt{y}]\!] \mapsto m\}$ is equal to $\mathfrak{h}$, except that the memory cell $[\![\mathtt{y}]\!] \mapsto m$ is removed.

As noted quite early, original Floyd-Hoare logic has a severe limitation when pointers are involved. Consider the following triple, an instance of the assignment rule:

$$\frac{}{\{\mathtt{y} = 1\}\ \mathtt{x} := 2\ \{\mathtt{y} = 1\}}$$

This essentially states that an assignment of 2 to $\mathtt{x}$ does not affect the value of $\mathtt{y}$ (if it is 1). With many popular imperative programming languages, this is not the case, as $\mathtt{x}$ and $\mathtt{y}$ may in fact be *aliased*, i.e., they may refer to the same or a partially-overlapping region of computer memory.

More precisely, the presence of pointers invalidates the use of the rule of constancy. Indeed, consider the following instance of the rule when pointers are involved:

$$\frac{\{\exists\, \mathtt{u}\ (\mathtt{x} \hookrightarrow \mathtt{u})\}\ [\mathtt{x}] := 4\ \{\mathtt{x} \hookrightarrow 4\}}{\{(\exists\, \mathtt{u}\ (\mathtt{x} \hookrightarrow \mathtt{u})) \wedge \mathtt{y} \hookrightarrow 3\}\ [\mathtt{x}] := 4\ \{\mathtt{x} \hookrightarrow 4 \wedge \mathtt{y} \hookrightarrow 3\}}$$

where $\mathtt{x} \hookrightarrow \mathtt{u}$ is a logical atomic formula stating that the heap has a memory cell with address $\mathtt{x}$ and value $\mathtt{u}$. Unsoundness is due to the possibility that $\mathtt{x}$ is equal to $\mathtt{y}$ (**aliasing**).

This defect will be repaired with the introduction of the **frame rule** below where $*$ is a separating connective so that distinct parts of memory can be reasoned about distinctly.

$$\frac{\{\varphi\}\ \mathsf{C}\ \{\psi\}}{\{\varphi * \psi'\}\ \mathsf{C}\ \{\psi * \psi'\}}\ \text{frame rule}$$

where no variable free in ψ' is modified by C.

Naturally, the problem with pointers was understood early on as a limitation, and aliasing has continued to plague program analysis in the decades since. However, the simplicity and composability of Hoare's proposal was appreciated, and various ways of overcoming this limitation within Hoare's formalism have been sought. Many of these approaches have used some form of *separation,* by which distinct parts of memory can be reasoned about distinctly.

1.1.4 The birth of separation logic

Burstall introduced distinct nonrepeating tree systems in 1972 [Bur72], implicitly appealing to a notion of *separation* to be later enshrined in separation logic. There were, however, limitations of Burstall's approach (see [Rey00] for a full treatment). Fragments of data structures could be asserted as separate, and this invention was important; however, they were not permitted to have internal sharing. This has the effect that the assertion language is limited in its ability to distinguish structures with (unbounded) sharing. Further, the notion of composition was directional, so that mutually-referential data posed a problem.

Recognising these limitations, Reynolds introduced the notion of an "independent conjunction" to Hoare logic, capable of speaking of disjoint structures and thus maintaining some control in the face of the aliasing problem. Its first incarnation, interpreted classically, was flawed, as it assumed monotonicity of interpretations of assertions in extensions of memory states but included an unsound proof rule. This was quickly repaired by coopting an intuitionistic semantics [Rey00].

This intuitionistic version was discovered independently [IO01] by Ishtiaq and O'Hearn. In fact, their efforts (together with Pym) on bunched implication (BI) logics [OP99, Pym02] gave them a somewhat more general perspective, and they recognised Reynolds' assertion language as being an instance of bunched implication that reasons about pointers. Independently, working from Reynolds' earlier classical variant, they developed a version of BI that used Reynolds' independent conjunction, and gave it intuitionistic semantics. Afterward, they considered a classical version, but ended up presenting these in reverse, the intuitionistic as a

variant of the classical; this as a result of the fact that the intuitionistic can be translated into the classical version, and the classical version was useful in reasoning about pointer disposal.

Their paper made two further important contributions. First, they introduced separating implication (the "magic wand") to the logic (this quite naturally came from BI's multiplicative implication). This addition of the magic wand was not merely an afterthought or side effect of the instantiation of bunched implication in this "pointer logic" setting; indeed its addition was justified in its own right when first introduced [IO01]. Despite this, many verification applications have made use of the separating conjunction only and do not employ the magic wand. However, nowadays its use in verification is more recognised; see [LP14, Section 1] and [HCGT14, Section 8] for recent discussions on this topic (see also [TBR14, SS15]).

Second, they introduced the *frame rule,* important for local reasoning [IO01] (see Section 1.1.1). Given a Hoare triple $\{\varphi\}\ \mathtt{C}\ \{\psi\}$ and reasoning about partial computer memories satisfying φ and ψ, one can make conclusions about (disjoint) extensions of those partial memories and, in particular, about how these extensions are unaltered by C. This is at the core of the scalability of separation logic and its ability to handle aliasing.

In all these early versions of separation logic, memory locations were distinct from the integers. Reynolds later offered an extension that takes memory locations to be a (countably infinite) subset of the integers, and made fields of larger units independently addressable. His goal was to adequately model the low-level operation of code and, particularly, address arithmetic. In this document, we adopt such a convention: memory locations are integers. We also adopt modern syntax; before 2002, Reynolds used '&' for separating conjunction. The modern syntax is '$*$' for separating conjunction and '$-\!\!*$' for the separating implication, both taken from bunched implication logic.

1.2 A Core Version of Separation Logic

1.2.1 Basic definitions

Let us start by defining separation logics on concrete models, namely on heaps. Let $\mathtt{PVAR} = \{\mathtt{x}_1, \mathtt{x}_2, \ldots\}$ be a countably infinite set of **program variables** and $\mathtt{FVAR} = \{\mathtt{u}_1, \mathtt{u}_2, \ldots\}$ be a countably infinite set of **quantified variables**. A **memory state** is a pair $(\mathfrak{s}, \mathfrak{h})$ such that

- $\mathfrak{s}$ is a variable valuation of the form $\mathfrak{s} : \mathtt{PVAR} \to \mathbb{N}$ (the **store**),
- A **heap with** $k \geq 1$ **record fields** is a partial function $\mathfrak{h} : \mathbb{N} \rightharpoonup \mathbb{N}^k$ with finite domain. We write $\mathrm{dom}(\mathfrak{h})$ to denote its **domain** and $\mathrm{ran}(\mathfrak{h})$ to denote its **range**.

Usually in models for separation logic(s), memory states have a heap and a store for interpreting program variables, see e.g. [Rey02]. Herein, sometimes, there is no need for program variables (with a store) because we establish hardness results without the help of such program variables. Moreover, for the sake of simplicity, we do not make a distinction between the set of **locations** (domain of $\mathfrak{h}$) and the set of **values** (set of elements from the range of $\mathfrak{h}$).

When $k = 1$, we write $\widetilde{\sharp}\mathfrak{l}$ to denote the cardinal of the set $\{\mathfrak{l}' : \mathfrak{h}(\mathfrak{l}') = \mathfrak{l}\}$ made of **predecessors** of $\mathfrak{l}$ (heap $\mathfrak{h}$ is implicit in the expression $\widetilde{\sharp}\mathfrak{l}$). A location $\mathfrak{l}$ is an **ancestor** of a location $\mathfrak{l}'$ iff there exists $i \geq 0$ such that $\mathfrak{h}^i(\mathfrak{l}) = \mathfrak{l}'$ where $\mathfrak{h}^i(\mathfrak{l})$ is shorthand for $\mathfrak{h}(\mathfrak{h}(\ldots(\mathfrak{h}(\mathfrak{l})\ldots)))$ (i applications of $\mathfrak{h}$ to $\mathfrak{l}$).

Two heaps $\mathfrak{h}_1$ and $\mathfrak{h}_2$ are said to be **disjoint**, noted $\mathfrak{h}_1 \perp \mathfrak{h}_2$, if their domains are disjoint; when this holds, we write $\mathfrak{h}_1 \uplus \mathfrak{h}_2$ to denote the heap corresponding to the disjoint union of the graphs of $\mathfrak{h}_1$ and $\mathfrak{h}_2$, hence $\mathrm{dom}(\mathfrak{h}_1 \uplus \mathfrak{h}_2) = \mathrm{dom}(\mathfrak{h}_1) \uplus \mathrm{dom}(\mathfrak{h}_2)$. When the domains of $\mathfrak{h}_1$ and $\mathfrak{h}_2$ are not disjoint, the composition $\mathfrak{h}_1 \uplus \mathfrak{h}_2$ is not defined even if $\mathfrak{h}_1$ and $\mathfrak{h}_2$ have the same values on $\mathrm{dom}(\mathfrak{h}_1) \cap \mathrm{dom}(\mathfrak{h}_2)$. Moreover, we can also define the disjoint union of the memory states $(\mathfrak{s}_1, \mathfrak{h}_1)$ and $(\mathfrak{s}_2, \mathfrak{h}_2)$ when $\mathfrak{s}_1 = \mathfrak{s}_2$ and $\mathfrak{h}_1 \perp \mathfrak{h}_2$ so that $(\mathfrak{s}_1, \mathfrak{h}_1) \uplus (\mathfrak{s}_2, \mathfrak{h}_2) \stackrel{\text{def}}{=} (\mathfrak{s}_1, \mathfrak{h}_1 \uplus \mathfrak{h}_2)$. We write $\mathfrak{h} \sqsubseteq \mathfrak{h}'$ when the heap $\mathfrak{h}'$ is a **conservative extension** of the heap $\mathfrak{h}$, i.e. $\mathrm{dom}(\mathfrak{h}) \subseteq \mathrm{dom}(\mathfrak{h}')$ and, $\mathfrak{h}$ and $\mathfrak{h}'$ agree on $\mathrm{dom}(\mathfrak{h})$. In Figure 1.1, we illustrate how disjoint memory states are built when there is a unique record field while recalling a standard graphical representation. Each node represents a distinct natural number (the value is not specified in Figure 1.1) and each edge $\mathfrak{l} \to \mathfrak{l}'$ encodes the fact that $\mathfrak{h}(\mathfrak{l}) = \mathfrak{l}'$, assuming that $\mathfrak{h}$ is the heap graphically represented. A variable $\mathtt{x}_i$ just above a node means that its value by the store $\mathfrak{s}$ is precisely that node. In Figure 1.1, the heap on the left of the equality sign (say $\mathfrak{h}$) is equal to the disjoint union of the two heaps on the right of the equality sign (say $\mathfrak{h}_1$, $\mathfrak{h}_2$ from left to right). For example, the self-loop on the node labelled by $\mathtt{x}_3$ encodes that $(\mathfrak{s}, \mathfrak{h}) \models \mathtt{x}_3 \hookrightarrow \mathtt{x}_3$ where $\models$ is the satisfaction relation defined below. Similarly, $(\mathfrak{s}, \mathfrak{h}_1) \models \mathtt{x}_3 \hookrightarrow \mathtt{x}_3$ but not $(\mathfrak{s}, \mathfrak{h}_2) \models \mathtt{x}_3 \hookrightarrow \mathtt{x}_3$. Each edge in the graphical representation of the heap $\mathfrak{h}$ corresponds to a unique edge in the graphical representation of either $\mathfrak{h}_1$ or $\mathfrak{h}_2$.

For every $k \geq 1$, formulae of kSL are built from **expressions** of the form $\mathtt{e} ::= \mathtt{x} \mid \mathtt{u}$ where $\mathtt{x} \in \mathtt{PVAR}$ and $\mathtt{u} \in \mathtt{FVAR}$, and **atomic formulae** of the form

$$\pi ::= \mathtt{e} = \mathtt{e}' \mid \mathtt{e} \hookrightarrow \mathtt{e}_1, \ldots, \mathtt{e}_k \mid \mathtt{emp} \mid \bot \ .$$

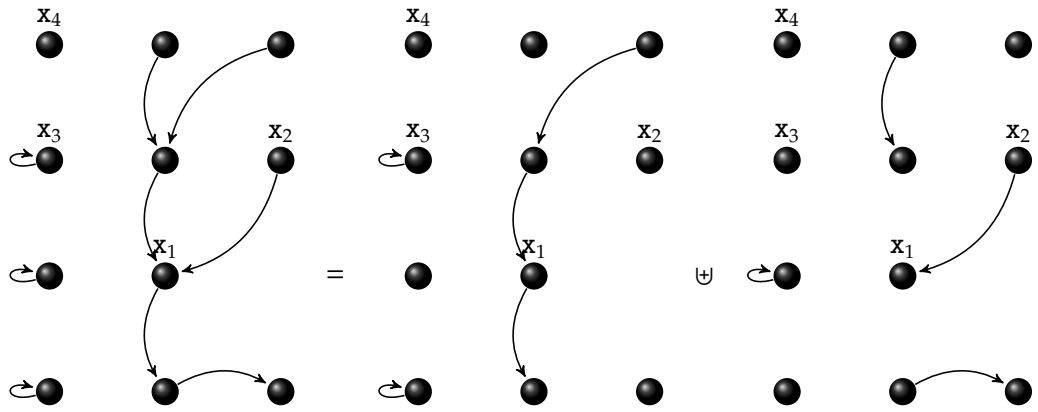


Figure 1.1: Disjoint memory states with one record field.

Formulae are defined by the grammar

$$\varphi, \psi ::= \pi \mid \varphi \wedge \psi \mid \neg\varphi \mid \varphi * \psi \mid \varphi \mathrel{-\!\!*} \psi \mid \exists\, \mathtt{u}\, \varphi$$

where $\mathtt{u} \in \mathtt{FVAR}$. The connective $*$ is **separating conjunction** and $\mathrel{-\!\!*}$ is **separating implication**, usually called the **magic wand**. The use of the magic wand $\mathrel{-\!\!*}$ is due to [IO01]. We also make use of standard notations for derived connectives for this and all logics defined in this document.

As in classical first-order logic, an **assignment** is a map $\mathsf{f} : \mathtt{FVAR} \to \mathbb{N}$. The satisfaction relation $\models$ is parameterised by assignments (obvious clauses for Boolean connectives are omitted):

$(\mathsf{s}, \mathsf{h}) \models_{\mathsf{f}} \mathtt{emp}$	iff	$\mathrm{dom}(\mathsf{h}) = \emptyset$
$(\mathsf{s}, \mathsf{h}) \models_{\mathsf{f}} \mathtt{e} = \mathtt{e}'$	iff	$[\![\mathtt{e}]\!] = [\![\mathtt{e}']\!]$, with $[\![\mathtt{x}]\!] \stackrel{\text{def}}{=} \mathsf{s}(\mathtt{x})$ and $[\![\mathtt{u}]\!] \stackrel{\text{def}}{=} \mathsf{f}(\mathtt{u})$
$(\mathsf{s}, \mathsf{h}) \models_{\mathsf{f}} \mathtt{e} \hookrightarrow \mathtt{e}_1, \ldots, \mathtt{e}_k$	iff	$[\![\mathtt{e}]\!] \in \mathrm{dom}(\mathsf{h})$ and $\mathsf{h}([\![\mathtt{e}]\!]) = ([\![\mathtt{e}_1]\!], \ldots, [\![\mathtt{e}_k]\!])$
$(\mathsf{s}, \mathsf{h}) \models_{\mathsf{f}} \varphi_1 * \varphi_2$	iff	$\mathsf{h} = \mathsf{h}_1 \uplus \mathsf{h}_2$, $(\mathsf{s}, \mathsf{h}_1) \models_{\mathsf{f}} \varphi_1$, $(\mathsf{s}, \mathsf{h}_2) \models_{\mathsf{f}} \varphi_2$ for some $\mathsf{h}_1, \mathsf{h}_2$
$(\mathsf{s}, \mathsf{h}) \models_{\mathsf{f}} \varphi_1 \mathrel{-\!\!*} \varphi_2$	iff	for all h', if $\mathsf{h} \perp \mathsf{h}'$ and $(\mathsf{s}, \mathsf{h}') \models_{\mathsf{f}} \varphi_1$ then $(\mathsf{s}, \mathsf{h} \uplus \mathsf{h}') \models_{\mathsf{f}} \varphi_2$
$(\mathsf{s}, \mathsf{h}) \models_{\mathsf{f}} \exists\, \mathtt{u}\, \varphi$	iff	there is $\mathsf{l} \in \mathbb{N}$ such that $(\mathsf{s}, \mathsf{h}) \models_{\mathsf{f}[\mathtt{u} \mapsto \mathsf{l}]} \varphi$ where $\mathsf{f}[\mathtt{u} \mapsto \mathsf{l}]$ is the assignment equal to f except that $\mathtt{u}$ takes the value l

When φ has no program variables, we also write $\mathsf{h} \models_{\mathsf{f}} \varphi$ to mean that φ is satisfied on the heap h under the assignment f. Furthermore, when φ is a sentence,

we can omit the subscript '$\dagger$' since φ has no free quantified variable. Note also that it is possible to get rid of program variables by viewing them as free quantified variables with rigid interpretation. However, it is sometime useful to distinguish syntactically program variables from quantified variables.

It is worth noting that separating conjunction $*$ has an existential flavour whereas separating implication $-\!\!*$ has a universal flavour. Nonetheless, $-\!\!*$ universally quantifies over an infinite set, namely the set of disjoint heaps. In the literature, an alternative syntax is used where $\mathtt{e} \hookrightarrow \mathtt{e}_1, \ldots, \mathtt{e}_k$ is represented by the conjunction below:

$$\mathtt{e} \overset{1}{\hookrightarrow} \mathtt{e}_1 \ \wedge \ \cdots \ \wedge \ \mathtt{e} \overset{k}{\hookrightarrow} \mathtt{e}_k$$

When pointer arithmetic is allowed, $\mathtt{e} \hookrightarrow \mathtt{e}_1, \ldots, \mathtt{e}_k$ can be also understood as the conjunction below

$$(\mathtt{e} \hookrightarrow \mathtt{e}_1) \ \wedge \ (\mathtt{e}+1 \hookrightarrow \mathtt{e}_2) \ \wedge \ \cdots (\mathtt{e}+(k-1) \hookrightarrow \mathtt{e}_k),$$

which requires some semantical adjustment. Nevertheless, in this document, we stick to $\mathtt{e} \hookrightarrow \mathtt{e}_1, \ldots, \mathtt{e}_k$, as defined above.

The definition of kSL does not provide a special treatment for `nil`. Indeed, it is possible to regain the usual behaviour by requiring that the interpretation of `nil` is not in the heap domain. This can be done in different ways depending on the fragment at hand.

The exact/precise points-to atomic formulae $\mathtt{x} \mapsto \mathtt{y}$ can be defined as abbreviations for $(\mathtt{x} \hookrightarrow \mathtt{y}) \wedge \neg(\neg\mathtt{emp} * \neg\mathtt{emp})$ and states that the domain of the heap is a singleton and the heap contains only the memory cell from $\mathtt{x} \mapsto \mathtt{y}$. The formula $\mathtt{emp} * \neg\mathtt{emp}$) enforces that $\mathrm{card}(\mathrm{dom}(\mathfrak{h})) \geq 2$. It is common to consider $\mathtt{x} \mapsto \mathtt{y}$ as a primitive atomic formula and in the rest of the document, we shall often refer to such an atomic formula. It will be clear from the context, whether $\mathtt{x} \mapsto \mathtt{y}$ should be considered as primitive.

For $k' \geq 0$, we write $k\mathrm{SL}k'$ to denote the fragment of kSL with at most k' quantified variables. So, we write kSL1 to denote the fragment of kSL restricted to a single quantified variable, say $\mathtt{u}$. Moreover, $k\mathrm{SL}k'(-\!\!*)$ [resp. $k\mathrm{SL}k'(*)$] denotes the fragment of $k\mathrm{SL}k'$ without separating conjunction [resp. wihout separating implication]. Note also that kSL can be understood as a syntactic fragment of $(k+1)$SL by simply encoding $\mathtt{e} \hookrightarrow \mathtt{e}_1, \ldots, \mathtt{e}_k$ by $\mathtt{e} \hookrightarrow \mathtt{e}_1, \mathtt{e}_1, \ldots, \mathtt{e}_k$ everywhere (the first expression is repeated twice).

As noted earlier, we do not make a distinction between the (countably infinite) set of locations and the set of values that includes the locations since only the set $\mathbb{N}$ is used to define the stores and heaps.

Let $\mathfrak{L}$ be a logic of the form $k\mathrm{SL}k'$ or one of its fragments or extensions. As usual, the **satisfiability problem** for $\mathfrak{L}$ takes as input a formula φ from $\mathfrak{L}$ and asks whether there is a memory state $(\mathfrak{s}, \mathfrak{h})$ and an assignment $\mathfrak{f}$ such that $(\mathfrak{s}, \mathfrak{h}) \models_{\mathfrak{f}} \varphi$. The **validity problem** is also defined as usual. The **model-checking problem** for $\mathfrak{L}$ takes as input a formula φ from $\mathfrak{L}$, a memory state $(\mathfrak{s}, \mathfrak{h})$ and a finite assignment $\mathfrak{f}$ for free variables from φ and asks whether $(\mathfrak{s}, \mathfrak{h}) \models_{\mathfrak{f}} \varphi$ ($\mathfrak{s}$ is finitely encoded and it is restricted to the program variables occurring in φ). Note that the model-checking problem for first-order logic over finite structures is known to be PSPACE-complete (see e.g. [Var82]) but we cannot conclude a similar statement for fragments of separation logic (even though $\mathfrak{s}$, $\mathfrak{h}$ and $\mathfrak{f}$ can be finitely encoded) because separating implication quantifies over an infinite set of disjoint heaps.

When $k = 1$, observe also that heaps are understood as Kripke frames of the form $(\mathbb{N}, \mathfrak{R})$ where $\mathfrak{R}$ is a finite and functional binary relation. Indeed, $\mathfrak{R} = \{(\mathfrak{l}, \mathfrak{h}(\mathfrak{l})) : \mathfrak{l} \in \mathrm{dom}(\mathfrak{h})\}$ for some heap $\mathfrak{h}$. Furthermore, the locations $\mathfrak{l}$ and $\mathfrak{l}'$ are in the same **connected component** whenever $(\mathfrak{l}, \mathfrak{l}') \in (\mathfrak{R} \cup \mathfrak{R}^{-1})^*$. Usually, connected components are understood as non-singleton components. A finite functional graph $(\mathbb{N}, \mathfrak{R})$ can be made of several maximal connected subgraphs so that each connected subgraph is made of a cycle, possibly with trees attached to it.

Finally, it is well-known that there exists a formal relationship between $*$ and $-\!\!*$ since $-\!\!*$ is the **adjunct** of $*$. This means that $(\varphi * \psi) \Rightarrow \chi$ is valid iff $\varphi \Rightarrow (\psi -\!\!* \chi)$ is valid. Exercise 1.5 is dedicated to this equivalence. This does not imply that the formula $((\varphi * \psi) \Rightarrow \chi) \Leftrightarrow (\varphi \Rightarrow (\psi -\!\!* \chi))$ is valid (otherwise $*$ and $-\!\!*$ would be inter-definable). However, sometimes, we are able to show that we can get rid of one of the separating connectives, see e.g. Chapter 3, without sacrificing the expressive power.

We also introduce so-called **septraction** operator $\overrightarrow{-\!\!*}$: $\varphi \overrightarrow{-\!\!*} \psi$ is defined as the formula $\neg(\varphi -\!\!* \neg\psi)$. As far as we know, its first appearance was in [VP07]. So, $(\mathfrak{s}, \mathfrak{h}) \models_{\mathfrak{f}} \varphi \overrightarrow{-\!\!*} \psi$ iff there is a heap $\mathfrak{h}'$ disjoint from $\mathfrak{h}$ such that $(\mathfrak{s}, \mathfrak{h}') \models_{\mathfrak{f}} \varphi$ and $(\mathfrak{s}, \mathfrak{h} \uplus \mathfrak{h}') \models_{\mathfrak{f}} \psi$. The septraction operator states the existence of a disjoint heap satisfying a formula and for which its addition to the original heap satisfies another formula.

1.2.2 Expressing properties with separation logic

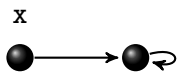
The logic 1SL allows one to express different types of properties on memory states. The examples below indeed illustrate the expressivity of 1SL.

- The domain of the heap has at least α elements: $\neg\mathtt{emp} * \cdots * \neg\mathtt{emp}$ (α times).

- The variable $\mathtt{x}$ is allocated in the heap: $\mathtt{alloc(x)} \stackrel{\text{def}}{=} (\mathtt{x} \hookrightarrow \mathtt{x}) \mathrel{-\!\!*} \bot$. Sometimes, atomic formulae of the form $\mathtt{alloc(x)}$ are primitive in the considered fragments of separation logics since $\mathtt{alloc(x)}$ is a fundamental property to express in a memory state (see Section 5.3).

- The variable $\mathtt{x}$ points to a location that is a self-loop:

$$\exists\, \mathtt{u}\ (\mathtt{x} \hookrightarrow \mathtt{u}) \wedge (\mathtt{u} \hookrightarrow \mathtt{u}).$$



In the following, let $\mathtt{u}$ and $\overline{\mathtt{u}}$ be the variables $\mathtt{u}_1$ and $\mathtt{u}_2$, in either order. Note that any formula $\varphi(\mathtt{u})$ with free variable $\mathtt{u}$ can be turned into an equivalent formula with free variable $\overline{\mathtt{u}}$ by permuting the two variables. Below, we define (standard) formulae and explain which properties they express.

- The domain $\mathrm{dom}(\mathfrak{h})$ has exactly one location:

$$\mathtt{size} = 1 \stackrel{\text{def}}{=} \neg\mathtt{emp} \wedge \neg(\neg\mathtt{emp} * \neg\mathtt{emp}).$$

- The domain $\mathrm{dom}(\mathfrak{h})$ has exactly two locations:

$$\mathtt{size} = 2 \stackrel{\text{def}}{=} (\neg\mathtt{emp} * \neg\mathtt{emp}) \wedge \neg(\neg\mathtt{emp} * \neg\mathtt{emp} * \neg\mathtt{emp}).$$

  It is easy to see that one can also define in 1SL that the heap domain has at least $k \geq 0$ elements (written $\mathtt{size} \geq k$).

- $\mathtt{u}$ has a successor: $\mathtt{alloc(u)} \stackrel{\text{def}}{=} \exists\, \overline{\mathtt{u}}\ \mathtt{u} \hookrightarrow \overline{\mathtt{u}}$.

- $\mathtt{u}$ has at least α predecessors: $\sharp\mathtt{u} \geq \alpha \stackrel{\text{def}}{=} \overbrace{(\exists\, \overline{\mathtt{u}}\ (\overline{\mathtt{u}} \hookrightarrow \mathtt{u})) * \cdots * (\exists\, \overline{\mathtt{u}}\ (\overline{\mathtt{u}} \hookrightarrow \mathtt{u}))}^{\alpha \text{ times}}$.

- $\mathtt{u}$ has at most α predecessors: $\sharp\mathtt{u} \leq \alpha \stackrel{\text{def}}{=} \neg\,(\sharp\mathtt{u} \geq \alpha + 1)$.

- $\mathtt{u}$ has exactly α predecessors: $\sharp\mathtt{u} = \alpha \stackrel{\text{def}}{=} (\sharp\mathtt{u} \geq \alpha) \wedge \neg(\sharp\mathtt{u} \geq \alpha + 1)$.

- There is a non-empty path from $\mathtt{u}$ to $\overline{\mathtt{u}}$ and nothing else except loops that exclude $\overline{\mathtt{u}}$:

$$\begin{aligned}\mathtt{reach}'(\mathtt{u},\overline{\mathtt{u}}) \stackrel{\text{def}}{=}\ & \sharp\mathtt{u} = 0 \wedge \mathtt{alloc}(\mathtt{u}) \wedge \neg\mathtt{alloc}(\overline{\mathtt{u}}) \wedge \\ & \forall\, \overline{\mathtt{u}}\ ((\mathtt{alloc}(\overline{\mathtt{u}}) \wedge \sharp\overline{\mathtt{u}} = 0) \Rightarrow \overline{\mathtt{u}} = \mathtt{u}) \wedge \\ & \forall\, \mathtt{u}\ \Big[(\sharp\mathtt{u} \neq 0 \wedge \mathtt{u} \neq \overline{\mathtt{u}}) \Rightarrow (\sharp\mathtt{u} = 1 \wedge \mathtt{alloc}(\mathtt{u}))\Big].\end{aligned}$$

- There is a (possibly empty) path from $\mathtt{u}$ to $\overline{\mathtt{u}}$:

$$\mathtt{reach}(\mathtt{u},\overline{\mathtt{u}}) \stackrel{\text{def}}{=} \mathtt{u} = \overline{\mathtt{u}} \vee \Big[\top * \mathtt{reach}'(\mathtt{u},\overline{\mathtt{u}})\Big].$$

  One can show that $\mathfrak{h} \models_{\mathfrak{f}} \mathtt{reach}(\mathtt{u},\overline{\mathtt{u}})$ iff there is $i \in \mathbb{N}$ such that $\mathfrak{h}^i(\mathfrak{f}(\mathtt{u})) = \mathfrak{f}(\overline{\mathtt{u}})$. The proof for this property can be found in [BDL12, Lemma 2.4] (a similar property has been established for graph logics in [DGG07]).

- There is a (possibly empty) path from $\mathtt{u}$ to $\overline{\mathtt{u}}$ and nothing else, can be defined as follows:

$$\mathtt{sreach}(\mathtt{u},\overline{\mathtt{u}}) \stackrel{\text{def}}{=} \mathtt{reach}(\mathtt{u},\overline{\mathtt{u}}) \wedge \neg(\neg\mathtt{emp} * \mathtt{reach}(\mathtt{u},\overline{\mathtt{u}}))$$

  $\mathtt{sreach}(\mathtt{u},\overline{\mathtt{u}})$ can be understood as the 'strict' reachability predicate and it is usually written as the segment predicate $\mathtt{ls}(\mathtt{u},\overline{\mathtt{u}})$.

- There is at most a single connected component (and nothing else):

$$\mathtt{1comp} \stackrel{\text{def}}{=} \neg\mathtt{emp} \wedge \exists\, \mathtt{u}\, \forall\, \overline{\mathtt{u}}\ \mathtt{alloc}(\overline{\mathtt{u}}) \Rightarrow \mathtt{reach}(\overline{\mathtt{u}},\mathtt{u}).$$

- There are exactly two components: $\mathtt{2comps} \stackrel{\text{def}}{=} \mathtt{1comp} * \mathtt{1comp}$.

It is also worth noting that the separation logic 1SL is not necessarily minimal, see obvious reasons below. A similar reasoning applies to any separation logic kSL. For instance, in 1SL, the atomic formula $\mathtt{emp}$ is logically equivalent to the following formula using only two quantified variables:

$$\forall\, \mathtt{u}\ \neg(\exists\, \mathtt{u}'\ (\mathtt{u} \hookrightarrow \mathtt{u}')).$$

Alternatively, it is equivalent to the following, which uses only one variable:

$$\forall\, \mathtt{u}\ \neg((\mathtt{u} \hookrightarrow \mathtt{u}) \mathrel{-\!\!*} \bot).$$

Note that $(\mathtt{u} \hookrightarrow \mathtt{u}) \mathrel{-\!\!*} \bot$ is the way to express $\mathtt{alloc}(\mathtt{u})$ in 1SL1 (as shown at the top of this section).

More interestingly, the atomic formula of the form $\mathtt{e} = \mathtt{e}'$ for some expressions $\mathtt{e}$, $\mathtt{e}'$ is logically equivalent to the following formula by using a new quantified variable $\mathtt{u}$ that does not occur in $\mathtt{e} = \mathtt{e}'$:

$$\forall\, \mathtt{u}\, ((\mathtt{u} \hookrightarrow \mathtt{e}) \mathrel{-\!\!*} (\mathtt{u} \hookrightarrow \mathtt{e}')).$$

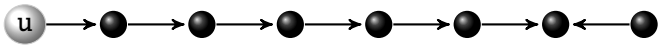
The formula simply states that adding to the heap a memory cell pointing to the location interpreted by $\mathtt{e}$ amounts to adding a memory cell pointing to the location interpreted by $\mathtt{e}'$.

Below, we define (standard) formulae and explain which properties they express.

- For all $\sim \in \{\leq, \geq, =\}$ and $i, \alpha \geq 0$, we define the following formulae:

$$\begin{array}{rcl} \sharp\mathtt{u}^0 \sim \alpha & \stackrel{\text{def}}{=} & \sharp\mathtt{u} \sim \alpha \\ \sharp\mathtt{u}^{i+1} \sim \alpha & \stackrel{\text{def}}{=} & \exists\, \overline{\mathtt{u}}\; \mathtt{u} \hookrightarrow \overline{\mathtt{u}} \wedge \sharp\overline{\mathtt{u}}^{i} \sim \alpha \\ \sharp\mathtt{u}^{-i-1} \sim \alpha & \stackrel{\text{def}}{=} & \exists\, \overline{\mathtt{u}}\; (\overline{\mathtt{u}} \hookrightarrow \mathtt{u}) \wedge \sharp\overline{\mathtt{u}}^{-i} \sim \alpha \end{array}$$

  For instance, $\sharp\mathtt{u}^6 \geq 2$ states that there is a (necessarily unique) location at distance 6 from $\mathtt{u}$ and its number of predecessors is greater than or equal to 2. This is illustrated below.



  Moreover, the formula $\sharp\mathtt{u}^{-5} \leq 2$ states that there is a (not necessarily unique) location at distance -5 from $\mathtt{u}$ and its number of predecessors is not strictly greater than 2. For instance, $\sharp\mathtt{u}^1 \geq 1$ is logically equivalent to $\mathtt{alloc}(\mathtt{u})$.

Remark. The heap is a finite tree with at least two nodes can be expressed by the formula below:

$$\neg\mathtt{emp} \wedge \exists\, \mathtt{u}\; \neg\mathtt{alloc}(\mathtt{u}) \wedge (\forall\, \overline{\mathtt{u}}\; \mathtt{alloc}(\overline{\mathtt{u}}) \Rightarrow \mathtt{reach}(\overline{\mathtt{u}}, \mathtt{u}))$$

Complexity results about two-variable fragments of first-order logic over finite trees can be found in [BBC+13] but we cannot really take advantage of them since we do not use predicate symbols apart from equality and the points-to relation. By contrast, we do admit separating connectives.

1.2.3 Deduction rules in a Floyd-Hoare proof system

In this section, we introduce deduction rules for some Hoare-like proof system in view of the new commands for mutable shared data structures and in view of the assertion language obtained by adding features from separation logic (presence of $*$ and $-\!\!*$).

The mutation command $[\mathtt{e}] := \mathtt{e}'$ leads to the following local rule (small axiom):

$$\frac{}{\{\exists\, \mathtt{u}\ \mathtt{e} \mapsto \mathtt{u}\}\ [\mathtt{e}] := \mathtt{e}'\ \{\mathtt{e} \mapsto \mathtt{e}'\}}\ \text{local mutation}$$

Note that the precondition and the postcondition are now expressed in 1SL. The global rule can be also derived by application of the frame rule.

$$\frac{}{\{(\exists\, \mathtt{u}\ \mathtt{e} \mapsto \mathtt{u}) * \varphi\}\ [\mathtt{e}] := \mathtt{e}'\ \{\mathtt{e} \mapsto \mathtt{e}' * \varphi\}}\ \text{global mutation}$$

Similarly, for performing backward reasoning, one can obtain the following rule:

$$\frac{}{\{(\exists\, \mathtt{u}\ \mathtt{e} \mapsto \mathtt{u}) * (\mathtt{e} \mapsto \mathtt{e}' -\!\!* \varphi)\}\ [\mathtt{e}] := \mathtt{e}'\ \{\varphi\}}\ \text{backwards reasoning mutation}$$

Global rules for deallocation and allocation are provided below:

$$\frac{}{\{(\exists\, \mathtt{u}\ \mathtt{e} \mapsto \mathtt{u}) * \varphi\}\ \mathbf{dispose}(\mathtt{e})\ \{\varphi\}}$$

$$\frac{}{\{\varphi\}\ \mathtt{x} := \mathbf{cons}(\mathtt{e})\ \{(\mathtt{x} \mapsto \mathtt{e}) * \varphi\}}$$

where $\mathtt{x}$ is not free in $\mathtt{e}$ and in φ.

Small axioms for allocation and deallocation are the following ones:

$$\frac{}{\{\exists\, \mathtt{u}\ \mathtt{e} \mapsto \mathtt{u}\}\ \mathbf{dispose}(\mathtt{e})\ \{\mathtt{emp}\}}$$

$$\frac{}{\{\mathtt{emp}\}\ \mathtt{x} := \mathbf{cons}(\mathtt{e})\ \{\mathtt{x} \mapsto \mathtt{e}\}}$$

1.2.4 Classes of formulae

Below, we provide a classification of formulae provided in [Rey02, Section 3], see also [IO01].

Definition 1.2.2. A sentence φ is **pure** if the true value of φ does not depend on the heap, i.e. for all stores $\mathfrak{s}$, and for all heaps $\mathfrak{h}$, $\mathfrak{h}'$, we have $(\mathfrak{s}, \mathfrak{h}) \models \varphi$ iff $(\mathfrak{s}, \mathfrak{h}') \models \varphi$. ∇

For instance, any Boolean combination built over equalities of the form $\mathtt{x} = \mathtt{y}$ is a pure formula. When φ_1 and φ_2 are pure, the formulae below can be proved valid:

1. $(\varphi_1 \wedge \varphi_2) \Leftrightarrow (\varphi_1 * \varphi_2)$.

2. $(\varphi_1 \Rightarrow \varphi_2) \Leftrightarrow (\varphi_1 \mathrel{-\!\!*} \varphi_2)$.

Definition 1.2.3. A sentence φ is **intuitionistic** if for all stores $\mathfrak{s}$, and for all heaps $\mathfrak{h}$, $\mathfrak{h}'$, ($\mathfrak{h}'$ is a conservative extension of $\mathfrak{h}$ and $(\mathfrak{s}, \mathfrak{h}) \models \varphi$) imply $(\mathfrak{s}, \mathfrak{h}') \models \varphi$. ∇

As a rule of thumb, intuitionistic semantics for separation logics is present whenever $(\mathfrak{s}, \mathfrak{h}) \models \varphi$ and $\mathfrak{h} \sqsubseteq \mathfrak{h}'$ imply $(\mathfrak{s}, \mathfrak{h}') \models \varphi$ for a given class of formulae φ.

In particular, any pure formula is intuitionistic. A typical example of intuitionistic formula is $\mathtt{x} \hookrightarrow \mathtt{y}$. Similarly, assuming that φ and ψ are intuitionistic formulae, the formulae below are intuitionistic too:

$$\varphi \wedge \psi \qquad \varphi \vee \psi \qquad \varphi * \psi \qquad \varphi \mathrel{-\!\!*} \psi$$

Moreover, whenever φ is intuitionistic, the formulae below are valid:

- $(\varphi * \top) \Rightarrow \varphi$.

- $(\varphi \Rightarrow (\top \mathrel{-\!\!*} \varphi)$.

Strictly exact formulae have been introduced in [Yan01].

Definition 1.2.4. A sentence φ is **strictly exact** if for all stores $\mathfrak{s}$, and for all heaps $\mathfrak{h}$, $\mathfrak{h}'$, (($\mathfrak{s}, \mathfrak{h}) \models \varphi$ and $(\mathfrak{s}, \mathfrak{h}') \models \varphi$) implies $\mathfrak{h} = \mathfrak{h}'$. ∇

Any formula built over atomic formulae of the form $\mathtt{x} \mapsto \mathtt{y}$ and $*$ are strictly exact (see also Lemma 5.2.3). Strictly exact formulae are helpful to establish the validity of the formula below when φ is strictly exact:

$$((\varphi * \top) \wedge \psi) \Rightarrow (\varphi * (\varphi \mathrel{-\!\!*} \psi)).$$

Strictly exact formulae are clearly domain-exact in the following sense. A sentence φ is **domain-exact** if for all stores $\mathfrak{s}$, and for all heaps $\mathfrak{h}$, $\mathfrak{h}'$, (($\mathfrak{s}, \mathfrak{h}) \models \varphi$ and $(\mathfrak{s}, \mathfrak{h}') \models \varphi$) implies $\mathrm{dom}(\mathfrak{h}) = \mathrm{dom}(\mathfrak{h}')$.

1.2.5 Decision problems

Let $\mathfrak{L}$ be a fragment of the separation logic kSL, $k \geq 1$. Below, we introduce the satisfiability/validity/model-checking/frame inference/abduction problems. Some of the problems have been already mentioned earlier but we prefer to present all of them below, even if repetitions are witnessed.

The **satisfiability problem** for $\mathfrak{L}$ is defined as follows:

Input: A sentence φ in $\mathfrak{L}$.

Question: Is there a memory state $(\mathfrak{s}, \mathfrak{h})$ such that $(\mathfrak{s}, \mathfrak{h}) \models \varphi$?

Similarly, the **validity problem** for $\mathfrak{L}$ is defined as follows:

Input: A sentence φ in $\mathfrak{L}$.

Question: Is it the case that for all memory states $(\mathfrak{s}, \mathfrak{h})$, we have $(\mathfrak{s}, \mathfrak{h}) \models \varphi$?

A variant of the validity problem is the **entailment problem** for $\mathfrak{L}$ that is defined as follows:

Input: Two sentences φ and ψ in $\mathfrak{L}$.

Question: Is it the case that $\varphi \models \psi$? ($\varphi \models \psi$ is a shortcut for: for all memory states $(\mathfrak{s}, \mathfrak{h})$, $(\mathfrak{s}, \mathfrak{h}) \models \varphi$ implies $(\mathfrak{s}, \mathfrak{h}) \models \psi$)

Obviously the validity problem is more general than the entailment problem and such a subproblem makes particularly sense when $\mathfrak{L}$ is not closed under negation, see e.g. the rule strenghtening preconditions in Section 1.1.1. Decidability of the entailement problem implies the decidability of the proof checking in Hoare-style proof systems in which separation logic is used as an assertion language. For example, strenghtening of preconditions or weakening of postconditions can be reduced to instances of the entailment problem. Usually, instances of the other deduction rules and the small axioms can be decided by a simple syntactic analysis. For instance, in tools such as Smallfoot, the proof is reconstructed from partial annotations (e.g., loop invariants) and a calculus of strongest postconditions is used to build verification conditions that are precisely instances of the entailment problem.

The **model-checking problem** for $\mathfrak{L}$ is defined as follows:

Input: A finite memory state $(\mathfrak{s}, \mathfrak{h})$ and a sentence φ in $\mathfrak{L}$.

Question: $(\mathfrak{s}, \mathfrak{h}) \models \varphi$?

As mentioned earlier, the separating implication quantifies over an infinite set of disjoint heaps and therefore finiteness of $(\mathfrak{s}, \mathfrak{h})$ does not imply straightforwardly that the model-checking problem for some kSL0 is decidable. However, the infinite set of disjoint heaps can be sometimes abstracted finitely (see Section 5.3).

Herein, even though we mainly focus on the satisfiability problem, and sometimes on its dual version, the validity problem (or on some of its fragments such as the entailment problem), we present below the **frame inference problem** and the **abduction problem** that are quite specific to separation logics. The **frame inference problem** for $\mathfrak{L}$ is defined as follows:

Input: Two sentences φ and ψ in $\mathfrak{L}$.

Question: Is there a sentence χ in $\mathfrak{L}$ such that $\varphi \models \psi * \chi$?

The **abduction problem** for $\mathfrak{L}$ is defined as follows:

Input: Two sentences φ and ψ in $\mathfrak{L}$.

Question: Is there a sentence χ in $\mathfrak{L}$ such that $\varphi * \chi \models \psi$?

The abduction problem is also called the **anti-frame problem**. Complexity of the abduction problem for symbolic heaps fragment of 1SL0 can be found in [GKO11].

1.3 Relationships with Other Logics

1.3.1 Logic of bunched implications and its Boolean variant

The logic of bunched implications BI has been introduced in [OP99, Pym02] and it combines connectives from intuitionistic logic with connectives from the multiplicative fragment of linear logic [Gir87]. The logic of bunched implications BI interprets formulae as resources that can be shared or separated. As mentioned previously, the works [IO01, OYR04] by O'Hearn, Reynolds and Ishtiaq have used separation to reason about programs with mutable data structures. More precisely, the assertion language of separation logic is a specialisation of the logic of bunched implications BI when the additive connectives ($\wedge$, $\neg$, $\Rightarrow$, $\top$, $\bot$) are classical and the multiplicative connectives ($*$, $-\!\!*$) admit an intuitionistic interpretation, leading to so-called Boolean BI, see e.g. [IO01, GLW06]. By contrast, BI admits an intuitionistic interpretation of the additive connectives.

Below, we recall the Kripke-style semantics for Boolean BI from [GLW06] providing an alternative to Boolean BI-algebras considered in [Pym02]. A Kripke-style semantics for BI can be found in [GMP02] too. Of course, the logic of bunched implications and Boolean BI admit proof-theoretical definitions and remarkable metatheoretical properties, see e.g. [Pym02], but below we focus on the semantics in order to better illustrate how separation logic can be understood as a specialisation of Boolean BI. By way of example, Boolean BI admits several proof systems, such as labelled sequent calculi [HTG13], nested sequent calculi [PSP13], Belnap-style display calculi [Bro12] or Hilbert-style proof system for a hybrid extension of Boolean BI [BV14]. A major difference between propositional separation logic 1SL0 and propositional Boolean BI is certainly that Boolean BI admits an undecidability validity/satisfiability problem [LG13, BK14] whereas the satisfiability problem for 1SL0 is PSPACE-complete (see Chapter 5). Hence, the specialisation of Boolean BI to memory states, and therefore the introduction of that concrete semantics, has a significant advantage computationally.

A **BBI-frame** is a triple $(M, \circ, E)$ such that

- M is a non-empty set,
- $\circ$ is binary function $\circ : M \times M \to \mathcal{P}(M)$ such that $\circ$ is commutative and associative,
- $E \subseteq M$ is the set of neutral elements, i.e. for all $m \in M$, $\{e \circ m : e \in E\} = \{m\}$.

A **BBI-model** [BV14] is a structure $(M, \circ, E, \mathfrak{V})$ such that $(M, \circ, E)$ is a BBI-frame and $\mathfrak{V}$ is a map $\mathfrak{V} : \text{PROP} \to \mathcal{P}(M)$ where $\text{PROP} = \{p_1, p_2, \ldots\}$ is a countably infinite set of atomic propositions.

Let $(\mathfrak{H}\mathfrak{S}_k, \uplus, \mathfrak{U}_k)$ be the triple such that $\mathfrak{H}\mathfrak{S}_k$ is the set of memory states with $k \geq 1$ record fields and $\mathfrak{U}_k$ is the set of memory states of the form $(\mathfrak{s}, \emptyset)$ where $\emptyset$ is the unique heap with empty domain. Note that $(\mathfrak{H}\mathfrak{S}_k, \uplus, \mathfrak{U}_k)$ is a BBI-frame where the non-deterministic monoid $(M, \circ, e)$ is made of memory states such that the heaps have k record fields and the binary function is the set-theoretical version of disjoint union.

The set of formulae for Boolean BI is defined with the following grammar.

$$\varphi, \psi ::= \texttt{emp} \mid p \mid \varphi \wedge \psi \mid \neg\varphi \mid \varphi * \psi \mid \varphi \mathrel{-\!\!*} \psi.$$

Let $m \in M$ and $\mathfrak{V} : \text{PROP} \to \mathcal{P}(M)$ be a valuation, the satisfaction relation $\models$ is defined as follows (we omit the obvious standard clauses for Boolean connectives):

$m \models_{\mathfrak{V}} \mathtt{emp}$	iff	$m \in E$
$m \models_{\mathfrak{V}} p$	iff	$m \in \mathfrak{V}(p)$
$m \models_{\mathfrak{V}} \varphi_1 * \varphi_2$	iff	for some $m_1, m_2 \in M$, we have $m \in m_1 \circ m_2$, $m_1 \models_{\mathfrak{V}} \varphi_1$ and $m_2 \models_{\mathfrak{V}} \varphi_2$
$m \models_{\mathfrak{V}} \varphi_1 \mathrel{-\!\!*} \varphi_2$	iff	for all $m', m'' \in M$ such that $m'' \in m \circ m'$, if $m' \models_{\mathfrak{V}} \varphi_1$ then $m'' \models_{\mathfrak{V}} \varphi_2$.

We keep the constant ‘$\mathtt{emp}$ for Boolean BI but elements of the set E should be understood as units.

A formula φ is **valid** iff for all BBI-models $(M, \circ, E, \mathfrak{V})$ and for all $m \in M$, we have $m \models_{\mathfrak{V}} \varphi$. Satisfiability can be formulated as usually, see e.g. [BdRV01].

The satisfiability problem for kSL0 can be reformulated as the satisfiability problem in the BBI-frame $(\mathfrak{H}\mathfrak{S}_k, \uplus, \mathfrak{U}_k)$ in which atomic propositions are of the form $\mathtt{x}_i \hookrightarrow \mathtt{x}_j$ or $\mathtt{x}_i = \mathtt{x}_j$ and the valuations $\mathfrak{V}$ are constrained in such a way that $(\mathfrak{s}, \mathfrak{h}) \in \mathfrak{V}(\mathtt{x}_i \hookrightarrow \mathtt{x}_j)$ iff $\mathfrak{h}(\mathfrak{s}(\mathtt{x}_i)) = \mathfrak{s}(\mathtt{x}_j)$. Similarly, we require that $(\mathfrak{s}, \mathfrak{h}) \in \mathfrak{V}(\mathtt{x}_i = \mathtt{x}_j)$ iff $\mathfrak{s}(\mathtt{x}_i) = \mathfrak{s}(\mathtt{x}_j)$.

Provability in Boolean BI is obtained by adding the rule “from $\varphi \vdash \neg\neg\psi$ conclude $\varphi \vdash \psi$” to the natural deduction calculus of BI [Pym02]. That is why, Boolean BI is often abbreviated as BI + $\{\neg\neg\psi \Rightarrow \psi\}$. It is also possible to design an Hilbert-tyle proof system for Boolean BI, as done in [GLW06], so that the notion of theorem for Boolean BI is clearly defined, but omitted herein.

Theorem 1.3.1. [GLW06] Theorems of Boolean BI are exactly the formulae valid in the class of BBI-models.

In order to be precise, the BBI-models introduced in [GLW06] assumes that E is a singleton set (single-unit condition) and it is shown in [BV14] that the class of single-unit BBI-models is not definable in Boolean BI. By contrast, it has been shown recently that validity in Boolean BI is not sensitive to the single unit condition [LG14]. Furthermore, we invite the reader to consult [BV14, LG14] for additional comparisons between variants of BBI-models and separation models. By analogy, the modal logic K is complete for the class of irreflexive frames but irreflexivity is not a property that is modally definable, see e.g. [BdRV01].

1.3.2 First-order logic with second-order features

In this section, let us focus on 1SL without program variables. Models for 1SL can be viewed as first-order structures of the form $(\mathbb{N}, \mathfrak{R})$ where $\mathfrak{R}$ is a finite and deterministic binary relation. We have seen in Section 1.2.2 that there is a formula

$\mathtt{reach}(\mathtt{u}, \overline{\mathtt{u}})$ in 1SL2($*$) such that $\mathfrak{h} \models_{\mathfrak{f}} \mathtt{reach}(\mathtt{u}, \overline{\mathtt{u}})$ iff $\mathfrak{f}(\mathtt{u})\mathfrak{R}^\star\mathfrak{f}(\overline{\mathtt{u}})$, where $\mathfrak{R}^\star$ is the reflexive and transitive closure of $\mathfrak{R}$ with

$$\mathfrak{R} \stackrel{\text{def}}{=} \{(\mathfrak{l}, \mathfrak{l}') : \mathfrak{l} \in \mathrm{dom}(\mathfrak{h}),\ \mathfrak{h}(\mathfrak{l}) = \mathfrak{l}'\}.$$

Anyway, 1SL without the separating connectives is clearly a fragment of first-order logic on structures of the form $(\mathbb{N}, \mathfrak{R})$ where $\mathfrak{R}$ is a finite and deterministic binary relation. Adding the separating conjunction provides a little bit of second-order logic, for instance by encoding the reachability relation. Given a binary relation $\mathfrak{R}$, we write $\mathrm{DTC}(\mathfrak{R})$ to denote the deterministic transitive closure of $\mathfrak{R}$ defined as the transitive closure of the relation

$$\mathfrak{R}_{det} = \{(\mathfrak{l}, \mathfrak{l}') \in \mathfrak{R} : \text{there is no } \mathfrak{l}'' \neq \mathfrak{l}' \text{ such that } (\mathfrak{l}, \mathfrak{l}'') \in \mathfrak{R}\}.$$

So, when $(\mathbb{N}, \mathfrak{R})$ is an 1SL model, $\mathrm{DTC}(\mathfrak{R})$ can be defined in 1SL2($*$) itself.

In fragments of classical logic, the presence of the deterministic transitive closure operator can lead to undecidability where the operator on the binary relation $\mathfrak{R}$ amounts to consider the transitive closure of the deterministic restriction $\mathfrak{R}_{det}$. In [GOR99], it is shown that FO2 (i.e. first-order logic restricted to two quantified variables) augmented with the deterministic transitive closure operator has an undecidable finitary satisfiability problem. By contrast, FO2 has the finite model property and the satisfiability problem is NEXPTIME-complete, see e.g. [GKV97]. Recently, FO2 augmented with the deterministic transitive closure of a single binary relation is shown to have a decidable and EXPSPACE-complete satisfiability problem [CKM14]. The works [GOR99] and [CKM14] contain numerous undecidability results related to the deterministic transitive closure operator but this involves more than one binary relation, whereas the models for 1SL have a unique deterministic binary relation. However, several results presented in [CKM14] are quite optimal with respect to the syntactic resources.

Meanwhile, Yorsh et al. [YRS+06] study a decidable version of first-order logic with reachability; they get decidability by making severe syntactic restrictions on the placement of quantifiers and on the reachability constraints, although the resulting logic is capable of describing useful linked data structures (see also the subsequent works [LQ08, PWZ13]).

1.3.3 Translation into dyadic second-order logic

Next we define weak second-order logic kWSOL, for $k \geq 1$. The sets $\mathtt{PVAR}$ and $\mathtt{FVAR}$ are defined as for kSL as well as the expressions $\mathtt{e}$.

We also consider a family $\mathtt{SVAR} = (\mathtt{SVAR}_i)_{i\geq 1}$ of second-order variables, denoted by $\mathtt{P}, \mathtt{Q}, \mathtt{R}, \ldots$ that are interpreted as finite relations over $\mathbb{N}$. Each variable in $\mathtt{SVAR}_i$ is interpreted as an i-ary relation.

As for kSL, models are memory states with $k \geq 1$ record fields. A second-order assignment $\mathfrak{f}$ is an interpretation of the second-order variables such that for every $\mathtt{P} \in \mathtt{SVAR}_i$, $\mathfrak{f}(\mathtt{P})$ is a finite subset of $\mathbb{N}^i$.

Atomic formulae take the form

$$\pi ::= \mathtt{e} = \mathtt{e}' \mid \mathtt{e} \hookrightarrow \mathtt{e}_1, \ldots, \mathtt{e}_k \mid \mathtt{P}(\mathtt{e}_1, \ldots, \mathtt{e}_n) \mid \mathtt{emp} \mid \bot \,.$$

Formulae of kWSOL are defined by the grammar

$$\varphi, \psi ::= \pi \mid \varphi \wedge \psi \mid \neg\varphi \mid \exists\, \mathtt{u}\, \varphi \mid \exists\, \mathtt{P}\, \varphi$$

where $\mathtt{P} \in \mathtt{SVAR}_n$ for some $n \geq 1$. We write kMSOL (monadic second-order logic) to denote the restriction of kWSOL to second-order variables in $\mathtt{SVAR}_1$ and kDSOL (dyadic second-order logic) to denote its restriction to $\mathtt{SVAR}_2$. Like kSL, models for kWSOL are memory states and quantifications are done over all the possible locations. The satisfaction relation $\models$ is defined as follows ($\mathfrak{f}$ is a hybrid valuation providing interpretation for first-order and second-order variables):

$(\mathfrak{s}, \mathfrak{h}) \models_{\mathfrak{f}} \exists\, \mathtt{P}\, \varphi$	iff	there is a *finite* relation $\mathfrak{R} \subseteq \mathbb{N}^n$ such that $(\mathfrak{s}, \mathfrak{h}) \models_{\mathfrak{f}[\mathtt{P} \mapsto \mathfrak{R}]} \varphi$ where $\mathtt{P} \in \mathtt{SVAR}_n$
$(\mathfrak{s}, \mathfrak{h}) \models_{\mathfrak{f}} \mathtt{P}(\mathtt{e}_1, \ldots, \mathtt{e}_n)$	iff	$(\llbracket \mathtt{e}_1 \rrbracket, \ldots, \llbracket \mathtt{e}_n \rrbracket) \in \mathfrak{f}(\mathtt{P})$.

The satisfiability problem for kWSOL takes as input a sentence φ in kWSOL and asks whether there is a memory state $(\mathfrak{s}, \mathfrak{h})$ such that $(\mathfrak{s}, \mathfrak{h}) \models \varphi$. By Trakhtenbrot's Theorem [Tra63, BGG97], the satisfiability problem for kDSOL (and therefore for kWSOL) is undecidable since finite satisfiability for first-order logic with a unique binary relation symbol is undecidable. Note that a monadic second-order variable can be simulated by a binary second-order variable from $\mathtt{SVAR}_2$, and this can be used to relativise a formula from DSOL in order to check finite satisfiability.

Theorem 1.3.2. [BDL12] kWSOL and kDSOL have the same expressive power.

It is just necessary to show how to reduce kWSOL to kDSOL since kDSOL is a syntactic fragment of kWSOL. Atomic formulae $\mathtt{P}(\mathtt{u})$ with the monadic second-order variable $\mathtt{P}$ are replaced by $\mathtt{P}^{new}(\mathtt{u}, \mathtt{u})$ where $\mathtt{P}^{new}$ is a fresh dyadic second-order variable. Furthermore, $\mathtt{P}(\mathtt{u}_1, \ldots, \mathtt{u}_n)$ with $n > 2$ is substituted by

$$\exists\, \mathtt{u} \bigwedge_{i=1}^{n} \mathtt{P}_i^{new}(\mathtt{u}, \mathtt{u}_i)$$

where $\mathtt{P}_1^{new}, \ldots, \mathtt{P}_n^{new}$ are fresh dyadic second-order variables used only for $\mathtt{P}$. The value k plays no special role here.

It has been recently shown in [DD14] that 1SL2($-\!\!*$) is as expressive as 1WSOL. The proof hinges on the fact that that every sentence from 1DSOL has an equivalent sentence in 1SL2($-\!\!*$), as discussed in Chapter 3. Translation in the other direction concerns us below. Separation logic 1SL can easily be translated into 1DSOL. The presentation given here is by a simple internalisation.

First, some formula definitions useful for the translation.

$$\begin{array}{lcl}
\mathtt{init}(\mathtt{P}) & \stackrel{\text{def}}{=} & \forall\, \mathtt{u}\ \mathtt{v}\ (\mathtt{P}(\mathtt{u},\mathtt{v}) \Leftrightarrow \mathtt{u} \hookrightarrow \mathtt{v}) \\
\mathtt{heap}(\mathtt{P}) & \stackrel{\text{def}}{=} & \forall\, \mathtt{u}\ \mathtt{v}\ \mathtt{w}\ ((\mathtt{P}(\mathtt{u},\mathtt{v}) \wedge \mathtt{P}(\mathtt{u},\mathtt{w})) \Rightarrow \mathtt{v} = \mathtt{w}) \quad \text{(functionality)} \\
\mathtt{P} = \mathtt{Q} \uplus \mathtt{R} & \stackrel{\text{def}}{=} & \forall\, \mathtt{u}\ \mathtt{v}\ ((\mathtt{P}(\mathtt{u},\mathtt{v}) \Leftrightarrow (\mathtt{Q}(\mathtt{u},\mathtt{v}) \vee \mathtt{R}(\mathtt{u},\mathtt{v}))) \wedge \neg(\mathtt{Q}(\mathtt{u},\mathtt{v}) \wedge \mathtt{R}(\mathtt{u},\mathtt{v}))).
\end{array}$$

The formula `init` initialises a binary relation $\mathtt{P}$ to be precisely the heap graph; this is a notational convenience for the top level of the translation. `heap` requires that a relation $\mathtt{P}$ is functional and is used to ensure that subheaps (as interpreted by second-order variables) are in fact heaps. Finally, $\mathtt{P} = \mathtt{Q} \uplus \mathtt{R}$ composes two relations representing subheaps ($\mathtt{Q}$ and $\mathtt{R}$) into one—or alternatively, it can be seen as decomposing $\mathtt{P}$ into two disjoint pieces; it is used in both "directions" in the translation.

Let the top-level translation $t(\varphi) \stackrel{\text{def}}{=} \exists\, \mathtt{P}\ \ (\mathtt{init}(\mathtt{P}) \wedge t_{\mathtt{P}}(\varphi))$, where $t_{\mathtt{P}}$ is the translation with respect to $\mathtt{P}$ as the "current" heap for interpretation. It is homomorphic for Boolean connectives, and otherwise has this definition:

$$\begin{array}{rcl}
t_{\mathtt{P}}(\mathtt{u} \hookrightarrow \mathtt{v}) & \stackrel{\text{def}}{=} & \mathtt{P}(\mathtt{u},\mathtt{v}) \\
t_{\mathtt{P}}(\varphi * \psi) & \stackrel{\text{def}}{=} & \exists\, \mathtt{Q}\ \mathtt{Q}'\ \ (\mathtt{P} = \mathtt{Q} \uplus \mathtt{Q}' \wedge t_{\mathtt{Q}}(\varphi) \wedge t_{\mathtt{Q}'}(\psi)) \\
t_{\mathtt{P}}(\varphi -\!\!* \psi) & \stackrel{\text{def}}{=} & \forall\, \mathtt{Q}\, \Big(((\exists\, \mathtt{Q}'\ \mathtt{heap}(\mathtt{Q}') \wedge \mathtt{Q}' = \mathtt{Q} \uplus \mathtt{P}) \wedge \mathtt{heap}(\mathtt{Q}) \wedge t_{\mathtt{Q}}(\varphi)) \\
 & & \Rightarrow (\exists\, \mathtt{Q}'\ \mathtt{heap}(\mathtt{Q}') \wedge \mathtt{Q}' = \mathtt{Q} \uplus \mathtt{P} \wedge t_{\mathtt{Q}'}(\psi))\Big).
\end{array}$$

Theorem 1.3.3. (see e.g. [BDL12]) There exists a translation t such that for any 1SL sentence φ and for any memory state $(\mathfrak{s}, \mathfrak{h})$, we have $(\mathfrak{s}, \mathfrak{h}) \models \varphi$ in 1SL iff $(\mathfrak{s}, \mathfrak{h}) \models t(\varphi)$ in 1DSOL.

Note that, then, there must also exist a translation from the smaller fragment 1SL2($-\!\!*$) into 1DSOL. This result (along with Theorem 1.3.2 above) will be useful later in showing expressive power results of separation logic.

General inductive predicates Using general inductive predicates provides another means to define second-order properties on heaps and this is a very useful feature to describe the shape of data structures, such as linked lists for instance. Semantics for general inductive predicates using least fixpoint operators can be naturally encoded in second-order logic, see e.g. [QGSM13]. Until very recently, such predicates are hard-coded but new results on the satisfiability and entailment problems for general inductive predicates have been obtained, see e.g. [IRS13, AGH$^+$14, BFGN14]. Whereas, it is shown in [BFGN14] that the satisfiability problem for many standard fragments of separation logic augmented with general inductive predicates is decidable and complexity is characterised (see also [IRS13] for bounded tree-width structures), other fragments have been shown to admit decidable entailment problem [IRS13, AGH$^+$14]. These are general results that are very promising for automatic verification of programs, despite the generality of the defined predicates.

1.3.4 Undecidability

A remarkable result about the decidability status of (first-order) separation logic is stated below and is due to [COY01] (see also [Yan01, Section 8.1] for a related undecidability result).

Theorem 1.3.4. [COY01] The satisfiability problem for 2SL is undecidable.

The proof is based on the fact that finitary satisfiability for classical predicate logic restricted to a single binary predicate symbol is undecidable [Tra63], see also [BGG97]. This means that given a first-order sentence φ built over the binary predicate symbol $\mathtt{R}$, checking whether there is a finite structure $(\mathfrak{D}, \mathfrak{R})$ (a finite directed graph) such that $(\mathfrak{D}, \mathfrak{R}) \models \varphi$ (in the first-order sense) is undecidable. Indeed, any such a structure can be encoded (modulo isomorphism) by some heap $\mathfrak{h}$ and some distinguished location $\mathfrak{l}_0$ such that:

- $\mathfrak{l}_0 \notin \mathrm{dom}(\mathfrak{h})$,
- $\mathfrak{D} = \{\mathfrak{l} \in \mathbb{N} : \mathfrak{h}(\mathfrak{l}) = (\mathfrak{l}_0, \mathfrak{l}_0)\}$,
- $\mathfrak{R} = \{(\mathfrak{l}, \mathfrak{l}') \in \mathfrak{D}^2 :$ there is $\mathfrak{l}''$ such that $\mathfrak{h}(\mathfrak{l}'') = (\mathfrak{l}, \mathfrak{l}')\}$.

Roughly speaking, a pair in $\mathfrak{R}$ is encoded by a memory cell in $\mathfrak{h}$. Let us define the translation T such that φ has a finite model $(\mathfrak{D}, \mathfrak{R})$ iff $T(\varphi)$ is satisfiable in 2SL

with

$$T(\varphi) \stackrel{\text{def}}{=} \exists\, \mathtt{u}, \mathtt{nil}\ \overbrace{(\mathtt{u} \hookrightarrow \mathtt{nil}, \mathtt{nil})}^{\text{"non-empty domain"}} \wedge \overbrace{(\neg \exists \mathtt{u}', \mathtt{u}''\ \mathtt{nil} \hookrightarrow \mathtt{u}', \mathtt{u}'')}^{\text{"nil not in the domain"}} \wedge tr(\varphi)$$

where $tr(\cdot)$ is homomorphic for Boolean connectives and it is defined below:

$$\begin{array}{lcl} tr(\mathtt{u}_i = \mathtt{u}_j) & \stackrel{\text{def}}{=} & (\mathtt{u}_i = \mathtt{u}_j) \wedge (\mathtt{u}_i \hookrightarrow \mathtt{nil}, \mathtt{nil}) \wedge (\mathtt{u}_j \hookrightarrow \mathtt{nil}, \mathtt{nil}) \\ tr(\mathtt{R}(\mathtt{u}_i, \mathtt{u}_j)) & \stackrel{\text{def}}{=} & (\mathtt{u}_i \hookrightarrow \mathtt{nil}, \mathtt{nil}) \wedge (\mathtt{u}_j \hookrightarrow \mathtt{nil}, \mathtt{nil}) \wedge (\exists \mathtt{u}\ (\mathtt{u} \hookrightarrow \mathtt{u}_i, \mathtt{u}_j)) \\ tr(\exists\, \mathtt{u}\ \psi) & \stackrel{\text{def}}{=} & \exists \mathtt{u}\ (\mathtt{u} \hookrightarrow \mathtt{nil}, \mathtt{nil}) \wedge tr(\psi) \\ tr(\forall\, \mathtt{u}\ \psi) & \stackrel{\text{def}}{=} & \forall\, \mathtt{u}\ (\mathtt{u} \hookrightarrow \mathtt{nil}, \mathtt{nil}) \Rightarrow tr(\psi). \end{array}$$

Observe that `nil` is understood as a distinguished variable whose interpretation is not in the heap domain. It is also worth noting that $T(\varphi)$ makes no use of program variables, separating conjunction, or separating implication. In a sense, the undecidability of 2SL, as explained above, is not very much related to separating connectives, but rather to the fact that heaps with two record fields can encode finite binary relations.

Theorem 1.3.5. [COY01] The set of valid formulae for 2SL is not recursively enumerable.

As a consequence, 2SL is not finitely axiomatisable. Indeed, φ is finitely valid iff $\forall \mathtt{u}, \mathtt{nil}\ ((\mathtt{u} \hookrightarrow \mathtt{nil}, \mathtt{nil}) \wedge (\neg \exists \mathtt{u}', \mathtt{u}''\ \mathtt{nil} \hookrightarrow \mathtt{u}', \mathtt{u}'')) \Rightarrow tr(\varphi)$ is 2SL valid. Since this is a logarithmic-space reduction and since the set of finitely valid formulae is not recursively enumerable, this leads to Theorem 1.3.5. It seems that this fact is not so well-known (this is of course mentioned in [COY01], and in a few other places such as in [Web04, Section 5] or in [Qiu13, Chapter 2]) but it has unpleasant consequences for defining proof systems for separation logics with concrete heaps (see e.g., [GM10, LP14, HCGT14, HGT15]). Note also that the result applies to any kSL since 2SL can be viewed then as a syntactic fragment of kSL as soon as $k \geq 2$.

In Chapter 3, we are able to show a similar result with 1SL by using directly first-order theory of natural numbers with addition and multiplication.

1.3.5 Modal logics with updates

The separating connectives $*$ and $-\!\!*$ force the interpretation of subformulae in alternative heaps, which is reminiscent to the destructive aspect of van Benthem's sabotage modal logic [vB05]. Indeed, sabotage modal logic (SML) defined in [vB05]

has the ability to remove states in a transition system. A variant of SML is introduced in [LR03] with the possibility to withdraw transitions, a feature also shared with logics from [PW04, Dem05, Göl07], see also logics of public announcements [Lut06]. The satisfiability problem for that variant is shown undecidable in [LR03] (another variant is shown undecidable in [Roh04] in which deletion of the transitions is done locally to the current state). Other modal logics updating the model while evaluating formulae have been considered in a systematic way in [ABdCH09] and specific instances can be found in [Mer09, AFH14].

1.4 Exercises

Exercise 1.1. Heaps can be understood as canonical elements of equivalence classes. Given a bijection $\sigma : \mathbb{N} \to \mathbb{N}$, we write $\mathfrak{h}' = \mathfrak{h} \circ \sigma$ to denote the heap whose graph is $\{(\sigma(\mathfrak{l}), \sigma(\mathfrak{h}(\mathfrak{l}))) : \mathfrak{l} \in \mathrm{dom}(\mathfrak{h})\}$. Similarly, we write $\mathfrak{f}' = \mathfrak{f} \circ \sigma$ to denote the assignment such that $\mathfrak{f}'(\mathtt{u}_i) = \sigma(\mathfrak{f}(\mathtt{u}_i))$.

a) Show that for all formulae φ in 1SL without program variables, we have $\mathfrak{h} \models_{\mathfrak{f}} \varphi$ iff $\mathfrak{h}' \models_{\mathfrak{f}'} \varphi$.

b) Extend the property when memory states are involved (i.e. with stores).

Exercise 1.2. Show that $*$ is commutative, associative and `emp` is a neutral element, i.e. show that the formulae below are valid.

a) $\varphi_1 * \varphi_2 \Leftrightarrow \varphi_2 * \varphi_1$.

b) $(\varphi_1 * \varphi_2) * \varphi_3 \Leftrightarrow \varphi_1 * (\varphi_2 * \varphi_3)$.

c) $\varphi * \mathtt{emp} \Leftrightarrow \varphi$.

Exercise 1.3. Distributivity laws are best illustrated by showing that the formulae below are valid.

a) $(\varphi_1 \vee \varphi_2) * \psi \Leftrightarrow (\varphi_1 * \psi) \vee (\varphi_2 * \psi)$.

b) $(\varphi_1 \wedge \varphi_2) * \psi \Rightarrow (\varphi_1 * \psi) \wedge (\varphi_2 * \psi)$.

c) $(\exists\, \mathtt{u}\, \varphi) * \psi \Leftrightarrow \exists\, \mathtt{u}\, (\varphi * \psi)$, assuming that $\mathtt{u}$ is not free in ψ.

d) $(\forall\, \mathtt{u}\, \varphi) * \psi \Rightarrow \forall\, \mathtt{u}\, (\varphi * \psi)$, assuming that $\mathtt{u}$ is not free in ψ.

Exercise 1.4. Show that if $\varphi_1 \Rightarrow \varphi_2$ and $\varphi_1' \Rightarrow \varphi_2'$ are valid, then $(\varphi_1 * \varphi_1') \Rightarrow (\varphi_2 * \varphi_2')$ is valid too.

Exercise 1.5. Let $k \geq 1$ and, φ, ψ and χ be formulae in kSL0. Show that $(\varphi * \psi) \Rightarrow \chi$ is valid iff $\varphi \Rightarrow (\psi \mathrel{-\!\!*} \chi)$ is valid.

Exercises 1.2–1.5 are inspired from properties stated in [Rey02, Section 3].

Exercise 1.6. Show that the local mutation rule from Section 1.2.3 is valid.

Exercise 1.7. Let φ be a formula in 1SL0 built without $\hookrightarrow$ and $\mathtt{emp}$. Show that φ is a pure formula.

Exercise 1.8. Let φ_1 and ψ_2 be pure formulae. Show that the formulae below are valid:

a) $(\varphi_1 \wedge \varphi_2) \Leftrightarrow (\varphi_1 * \varphi_2)$.

b) $(\varphi_1 \Rightarrow \varphi_2) \Leftrightarrow (\varphi_1 \mathrel{-\!\!*} \varphi_2)$.

Exercise 1.9. Assuming that φ is intuitionistic, show that $(\varphi * \top) \Rightarrow \varphi$ and $(\varphi \Rightarrow (\top \mathrel{-\!\!*} \varphi)$ are valid formulae.

Exercise 1.10. Let $\mathtt{e}$, $\mathtt{e}'$ be two expressions in 1SL and $\mathtt{u}$ be a variable that does not occur in $\mathtt{e} = \mathtt{e}'$. Show that for all memory states $(\mathfrak{s}, \mathfrak{h})$ and for all assignments $\mathfrak{f}$, we have $[\![\mathtt{e}]\!] = [\![\mathtt{e}']\!]$ iff $(\mathfrak{s}, \mathfrak{h})$ under the assignment $\mathfrak{f}$ satisfies $\forall\, \mathtt{u}\, ((\mathtt{u} \hookrightarrow \mathtt{e}) \mathrel{-\!\!*} (\mathtt{u} \hookrightarrow \mathtt{e}'))$.

Exercise 1.11. Show that the validity of $\varphi \Rightarrow (\psi \mathrel{-\!\!*} \chi)$ and $\varphi' \Rightarrow \psi$ implies the validity of $\varphi * \varphi' \Rightarrow \chi$.

Exercise 1.12. The version of kSL0 defined in this chapter admits an intuitionistic interpretation of the points-to atomic formulae, since whenever $(\mathfrak{s}, \mathfrak{h}) \models_{\mathfrak{f}} \mathtt{e} \hookrightarrow \mathtt{e}'$ and $\mathfrak{h}'$ is a conservative extension of $\mathfrak{h}$ (written $\mathfrak{h} \sqsubseteq \mathfrak{h}'$), we still have $(\mathfrak{s}, \mathfrak{h}') \models_{\mathfrak{f}} \mathtt{e} \hookrightarrow \mathtt{e}'$. Strictly speaking, a classical version of kSL0 should admit atomic formulae of the form $\mathtt{e} \mapsto \mathtt{e}'$ instead, but for kSL0, this does not make a subtantial difference in the classical framework.

Now, let us restrict ourselves to the formulae below:

$$\varphi, \psi ::= \mathtt{e} \hookrightarrow \mathtt{e}' \mid \mathtt{e} = \mathtt{e}' \mid \mathtt{e} \neq \mathtt{e}' \mid \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \varphi \Rightarrow \psi \mid$$

$$\varphi * \psi \mid \varphi \mathrel{-\!\!*} \psi \mid \forall\, \mathtt{u}\, \varphi$$

(no negation, no atomic formula $\mathtt{emp}$). In order to define the intuitionistic version of kSL0, we keep the clauses for the satisfaction relation from classical kSL0 except that we provide an intuitionistic interpretation for the connective $\Rightarrow$: $(\mathfrak{s},\mathfrak{h}) \models_{\mathfrak{i}} \varphi \Rightarrow \psi$ iff for all $\mathfrak{h} \sqsubseteq \mathfrak{h}'$, we have if $(\mathfrak{s},\mathfrak{h}') \models_{\mathfrak{i}} \varphi$, then $(\mathfrak{s},\mathfrak{h}') \models_{\mathfrak{i}} \psi$. Show that the intuitionistic version of kSL0 admits the monotonicity condition: if $(\mathfrak{s},\mathfrak{h}) \models_{\mathfrak{i}} \varphi$ and $\mathfrak{h} \sqsubseteq \mathfrak{h}'$, then $(\mathfrak{s},\mathfrak{h}') \models_{\mathfrak{i}} \varphi$.

Exercise 1.13. Let φ be a sentence in kSL without any program variable. Show that φ is valid iff $(\mathtt{emp} \wedge (\top \mathrel{-\!\!*} \varphi)) * \top$ is satisfiable.

Exercise 1.14. Show that $\mathtt{reach}(\mathtt{u},\overline{\mathtt{u}})$ as defined in the chapter corresponds indeed to the reachability predicate.

Exercise 1.15. Show that the formula below in 1SL2 characterizes the heaps with a unique connected component and with a non-empty domain:

$$\neg\mathtt{emp} \wedge \exists\, \mathtt{u}\ \forall\, \overline{\mathtt{u}}\ \mathtt{alloc}(\overline{\mathtt{u}}) \Rightarrow \mathtt{reach}(\overline{\mathtt{u}},\mathtt{u}).$$

Chapter 2

PROPOSITIONAL SEPARATION LOGICS

Contents

In this chapter, we consider decidability and computational complexity issues for propositional fragments of separation logics of the form kSL0 with $k \geq 1$. The upper bounds are mainly proved in Chapter 5. Section 2.1 is dedicated to the presentation of the PSPACE-completeness for the satisfiability and model-checking problems for kSL0 based on developments from [COY01]. Proofs for PSPACE-hardness are provided. Section 2.2 contains a presentation of the fragments of kSL0 made of symbolic heaps and for which the entailment problem and the satisfiability problem can be solved in polynomial time [CHO+11]. It is very essential that the reasoning for this fragment is tractable since it is used in early tools dealing with separation logic such as Smallfoot [BCO05]. Abstract separation logics are presented in Section 2.3; in such versions of separation logics atomic

propositions are introduced and can be true in any part of the model (contrary to points-to formulae), see also Section 1.3.1. Moreover, models are more abstract and correspond to cancellative partial commutative monoids. We show that the satisfiability problem for a few abstract separation logics is undecidable based on developments from [LG13, BK14, DD15a]. A reduction from the halting problem for Minsky machines is designed.

Highlights of the chapter

1. PSPACE-hardness proof for several fragments of 1SL0 by reduction from QBF as done in [COY01] (straightforward extensions to kSL0 with $k > 1$).
2. Presentation of the symbolic heaps fragment (and variants) that is used in the tool Smallfoot (Section 2.2) as well as recent complexity results.
3. Undecidability proof for propositional separation logic based on memory states (Theorem 2.3.1) inspired from the original proof in [BK10, LWG10] but adapted to avoid any proof-theoretical consideration [DD15b].

2.1 PSPACE-Completeness and Expressive Power

2.1.1 PSPACE-hard fragments of kSL0

Most probably, NP-completeness already implies non-tractability but actually, propositional separation logic of the form kSL0 with $k \geq 1$ can be potentially of even worse complexity, see e.g. [COY01, Rey02].

Theorem 2.1.1. [COY01] For every $k \geq 1$, the satisfiability problem for (propositional) kSL0 is PSPACE-complete.

The proof for the PSPACE upper bound is provided in Section 5.3 in which other results about the expressive power of kSL0 are discussed. Meanwhile, below, we show that 1SL0 is PSPACE-hard by reduction from QBF, following developments from [COY01]. QBF formulae are built from propositional formulae with the addition of propositional quantifications of the form $\forall\ p\ \psi$ and $\exists\ p\ \psi$. Below, without any loss of generality, we consider QBF formulae in prenex normal form. We consider several fragments of 1SL0 in order to pinpoint different causes for PSPACE-hardness.

Let $Q_1\ p_1\ \cdots\ Q_n\ p_n\ \varphi$ be a QBF formula with $\{Q_1, \ldots, Q_n\} \subseteq \{\exists, \forall\}$ and φ is a propositional formula built over the atomic propositions in $\{p_1, \ldots, p_n\}$ and the Boolean connectives $\wedge$, $\vee$ and $\neg$ (only in front of atomic propositions). The formula is said to be in prenex normal form and every QBF formula can be reduced in logarithmic space to an equivalent formula in such a form. We recall that given a propositional valuation $v : \mathrm{PROP} \to \{\bot, \top\}$, we have $v \models \exists\, p\ \varphi$ iff there is $b \in \{\bot, \top\}$ such that $v[p \mapsto b] \models \varphi$. Similarly, $v \models \forall\, p\ \varphi$ iff for all $b \in \{\bot, \top\}$, we have $v[p \mapsto b] \models \varphi$. Satisfiability problem for QBF formulae is known to be PSPACE-complete [Sto77].

In the translation of the formula $Q_1\ p_1\ \cdots\ Q_n\ p_n\ \varphi$, we consider n program variables, say $\mathtt{x}_1$, …, $\mathtt{x}_n$ so that the truth of p_i is encoded by the satisfaction of $\mathtt{alloc}(\mathtt{x}_i)$ (that can be defined by $(\mathtt{x}_i \hookrightarrow \mathtt{x}_i) \mathrel{-\!\!*} \bot$). Similarly, we write $\mathtt{x}_i \mapsto -$ as an abbreviation for $\mathtt{alloc}(\mathtt{x}_i) \wedge \neg(\neg\mathtt{emp} * \neg\mathtt{emp})$. Obviously, $\mathtt{x}_i \mapsto -$ holds true when $\mathtt{x}_i$ is the unique location belonging to the heap domain.

In order to encode independence between the different variables, we enforce that all the program variables have distinct values in the original heap. Moreover, existential quantification over p_i amounts to restrict the current heap either by the empty heap (in that case $\mathtt{alloc}(\mathtt{x}_i)$ holds in the other heap) or by a unique memory cell so that $\mathtt{alloc}(\mathtt{x}_i)$ holds, which allows to simulate quantification. However, it is necessary to enforce in the initial heap that $\mathtt{alloc}(\mathtt{x}_i)$ holds for any program variable $\mathtt{x}_i$, $i \in [1, n]$. Let us define the map tr as follows when tr is homomorphic for Boolean connectives:

$$\begin{array}{lcl} tr(p_i) & \stackrel{\text{def}}{=} & \mathtt{alloc}(\mathtt{x}_i) \\ tr(\exists\, p_i\ \psi) & \stackrel{\text{def}}{=} & (\mathtt{emp} \vee \mathtt{x}_i \mapsto -) * tr(\psi) \\ tr(\forall\, p_i\ \psi) & \stackrel{\text{def}}{=} & \neg((\mathtt{emp} \vee \mathtt{x}_i \mapsto -) * \neg tr(\psi)). \end{array}$$

Lemma 2.1.2. The formula $Q_1\ p_1\ \cdots\ Q_n\ p_n\ \varphi$ is QBF satisfiable iff

$$\chi \stackrel{\text{def}}{=} (\bigwedge_{i \neq j} \mathtt{x}_i \neq \mathtt{x}_j) \wedge (\bigwedge_{i} \mathtt{alloc}(\mathtt{x}_i)) \wedge tr(Q_1\ p_1\ \cdots\ Q_n\ p_n\ \varphi)$$

is 1SL0 satisfiable.

Proof. Let us start by introducing auxiliary definitions. For every $j \in [1, n+1]$, we write φ_j to denote the formula $Q_j\ p_j\ \cdots\ Q_n\ p_n\ \varphi$. So, by definition, we have $\varphi_1 = Q_1\ p_1\ \cdots\ Q_n\ p_n\ \varphi$ and by convention $\varphi_{n+1} = \varphi$. Note also that the atomic propositions in φ_j that are not in the scope of a propositional quantification belongs to the (possibly empty) set $\{p_i : i \in [1, j-1]\}$.

Given a memory state $(\mathfrak{s}, \mathfrak{h})$ and a propositional valuation v, we write $(\mathfrak{s}, \mathfrak{h}) \approx_j v$ to denote the fact that:

- For all $i \neq i' \in [1, n]$, we have $\mathfrak{s}(\mathtt{x}_i) \neq \mathfrak{s}(\mathtt{x}_{i'})$ (this only depends on $\mathfrak{s}$).
- For all $i \in [j, n]$, we have $\mathfrak{s}(\mathtt{x}_i) \in \mathrm{dom}(\mathfrak{h})$ (this only depends on the memory state).
- For all $i \in [1, j-1]$, we have $\mathfrak{s}(\mathtt{x}_i) \in \mathrm{dom}(\mathfrak{h})$ iff $v(p_i) = \top$.

It is easy to establish that $(\mathfrak{s}, \mathfrak{h}) \approx_j v$ for some v is equivalent to $(\mathfrak{s}, \mathfrak{h}) \models (\bigwedge_{i \neq i'} \mathtt{x}_i \neq \mathtt{x}_{i'}) \wedge (\bigwedge_{i \in [j,n]} \mathtt{alloc}(\mathtt{x}_i))$.

By induction on j, we show that for all $j \in [1, n+1]$, if $(\mathfrak{s}, \mathfrak{h}) \approx_j v$, then $(\mathfrak{s}, \mathfrak{h}) \models tr(\varphi_j)$ iff $v \models \varphi_j$. The base case in the induction corresponds to $j = n+1$ and therefore the induction step goes backwards.

Before providing the proof by induction, let us check that this is sufficient to establish the statement in the lemma. If $(\mathfrak{s}, \mathfrak{h}) \approx_1 v$, then $(\mathfrak{s}, \mathfrak{h})$ satisfies $(\bigwedge_{i \neq j} \mathtt{x}_i \neq \mathtt{x}_j) \wedge (\bigwedge_i \mathtt{alloc}(\mathtt{x}_i))$. Suppose that $v \models Q_1\, p_1 \cdots Q_n\, p_n\, \varphi$. Let us consider the memory state $(\mathfrak{s}, \mathfrak{h})$ such that: for all $i \in [1, n]$, we have $\mathfrak{s}(\mathtt{x}_i) = i$ and $\mathfrak{h}(i) = i$. Obviously, $(\mathfrak{s}, \mathfrak{h}) \approx_1 v$ and therefore by the property above, we get $(\mathfrak{s}, \mathfrak{h}) \models tr(\varphi_1)$, that is $(\mathfrak{s}, \mathfrak{h}) \models tr(\varphi)$. Consequently, $(\mathfrak{s}, \mathfrak{h}) \models \chi$. Now suppose that $(\mathfrak{s}, \mathfrak{h}) \models \chi$. Let us take any propositional valuation v. We have $(\mathfrak{s}, \mathfrak{h}) \approx_1 v$ and therefore by the property above, we get $v \models \varphi_1$, that is $v \models \varphi$.

Now let us consider the proof of the above property.

Base case: $j = n+1$.

So, $\varphi_j = \varphi_{n+1} = \varphi$. The proof is by structural induction but the cases in the induction step with $\neg$ (in front of atomic propositions), $\vee$ and $\wedge$ are by an easy verification. We assume that $(\mathfrak{s}, \mathfrak{h}) \approx_{n+1} v$ and let us consider p_i with $i < n+1$. If $v \models p_i$, then $v(p_i) = \top$ and since $(\mathfrak{s}, \mathfrak{h}) \approx_{n+1} v$, we get $\mathfrak{s}(\mathtt{x}_i) \in \mathrm{dom}(\mathfrak{h})$ and $(\mathfrak{s}, \mathfrak{h}) \models \mathtt{alloc}(\mathtt{x}_i)$ $(= tr(p_i))$. Conversely, if $(\mathfrak{s}, \mathfrak{h}) \models tr(p_i)$, then $\mathfrak{s}(\mathtt{x}_i) \in \mathrm{dom}(\mathfrak{h})$ and since $(\mathfrak{s}, \mathfrak{h}) \approx_{n+1} v$ we get $v(p_i) = \top$, whence $v \models p_i$.

For the induction step with $j < n+1$, below we deal with the case $\varphi_j = \exists\, p_j\, \varphi_{j+1}$. The case $\varphi_j = \forall\, p_j\, \varphi_{j+1}$ is omitted since it is very similar. We assume that $(\mathfrak{s}, \mathfrak{h}) \approx_j v$.

First suppose that $v \models \exists\, p_j\, \varphi_{j+1}$. This means that there is $b \in \{\top, \bot\}$ such that $v[p_j \mapsto b] \models \varphi_{j+1}$.

Case 1: $b = \bot$.

Let $\mathfrak{h}', \mathfrak{h}''$ be such that $\mathfrak{h} = \mathfrak{h}' \uplus \mathfrak{h}''$ and $\mathrm{dom}(\mathfrak{h}'') = \{\mathfrak{s}(\mathtt{x}_j)\}$. We know that such

a separation of $\mathfrak{h}$ is possible since $(\mathfrak{s},\mathfrak{h}) \approx_j v$ (in particular $\mathfrak{s}(\mathtt{x}_j) \in \mathrm{dom}(\mathfrak{h})$). We get that $(\mathfrak{s},\mathfrak{h}') \approx_{j+1} v[p_j \mapsto \bot]$ and therefore by the induction hypothesis, we have $(\mathfrak{s},\mathfrak{h}') \models tr(\varphi_{j+1})$. So, $(\mathfrak{s},\mathfrak{h}) \models (\mathtt{x}_j \mapsto -) * tr(\varphi_{j+1})$ and *a fortiori*, $(\mathfrak{s},\mathfrak{h}) \models (\mathtt{emp} \vee (\mathtt{x}_j \mapsto -)) * tr(\varphi_{j+1})$ $(= tr(\varphi_j))$.
Case 2: $b = \top$.
We get that $(\mathfrak{s},\mathfrak{h}) \approx_{j+1} v[p_j \mapsto \top]$ and therefore by induction hypothesis, $(\mathfrak{s},\mathfrak{h}) \models tr(\varphi_{j+1})$. So, $(\mathfrak{s},\mathfrak{h}) \models \mathtt{emp} * tr(\varphi_{j+1})$ and *a fortiori*, $(\mathfrak{s},\mathfrak{h}) \models (\mathtt{emp} \vee (\mathtt{x}_j \mapsto -)) * tr(\varphi_{j+1})$.

Now suppose that $(\mathfrak{s},\mathfrak{h}) \models (\mathtt{emp} \vee (\mathtt{x}_j \mapsto -)) * tr(\varphi_{j+1})$.
Case 1: $(\mathfrak{s},\mathfrak{h}) \models \mathtt{emp} * tr(\varphi_{j+1})$.
So, $(\mathfrak{s},\mathfrak{h}) \models tr(\varphi_{j+1})$ and since $(\mathfrak{s},\mathfrak{h}) \approx_{j+1} v[p_j \mapsto \top]$, by the induction hypothesis, we get $v[p_j \mapsto \top] \models \varphi_{j+1}$, whence $v \models \exists\, p_j\, \varphi_{j+1}$.
Case 2: $(\mathfrak{s},\mathfrak{h}) \models (\mathtt{x}_j \mapsto -) * tr(\varphi_{j+1})$.
So, there are heaps $\mathfrak{h}'$ and $\mathfrak{h}''$ such that $(\mathfrak{s},\mathfrak{h}'') \models \mathtt{x}_j \mapsto -$ and $(\mathfrak{s},\mathfrak{h}') \models tr(\varphi_{j+1})$. Since by construction, $(\mathfrak{s},\mathfrak{h}') \approx_{j+1} v[p_j \mapsto \bot]$, by the induction hypothesis, we get $v[p_j \mapsto \bot] \models \varphi_{j+1}$ and therefore $v \models \exists\, p_j\, \varphi_{j+1}$. **QED**

The above reduction implies that the satisfiability problem for the logic kSL0$(*)$ is PSPACE-hard too, assuming that the atomic formulae are of the form $\mathtt{emp}$, $\mathtt{alloc(x)}$ and $\mathtt{x} = \mathtt{y}$.

Corollary 2.1.3. [COY01] For every $k \geq 1$, the satisfiability and model-checking problems for kSL0 is PSPACE-hard.

Proof. PSPACE-hardness of the satisfiability problem is a direct consequence of Lemma 2.1.2 since QBF is PSPACE-complete. Note that Lemma 2.1.2 is stated for $k = 1$ but it is easy to adapt it to any fixed $k > 1$ since the heap domain can be constrained only by atomic formulae of the form $\mathtt{alloc(x)}$.

Now, it is easy to design a logarithmic-space reduction from QBF into the model-checking for 1SL0. Let $Q_1\, p_1\, \cdots\, Q_n\, p_n\, \varphi$ be a QBF formula. We define the memory state $(\mathfrak{s},\mathfrak{h})$ such that for all $i \in [1,n]$, we have $\mathfrak{s}(\mathtt{x}_i) = i$ and $\mathfrak{h}(i) = i$. Obviously, $(\mathfrak{s},\mathfrak{h}) \models (\bigwedge_{i\neq j} \mathtt{x}_i \neq \mathtt{x}_j) \wedge (\bigwedge_i \mathtt{alloc}(\mathtt{x}_i))$. Let $v_\bot$ be the propositional valuation that returns always the constant value $\bot$. According to previous developments, we have $(\mathfrak{s},\mathfrak{h}) \models tr(Q_1\, p_1\, \cdots\, Q_n\, p_n\, \varphi)$ iff $v_\bot \models Q_1\, p_1\, \cdots\, Q_n\, p_n\, \varphi$. The proof for $k > 1$ is similar since $\mathfrak{h}(i)$ defined above can be adapted to be a k-tuple made of k times the value i. **QED**

A substantial fragment of 1SL0 that admits NP-complete satisfiability and model-checking problems is presented in Exercise 2.4. Other fragments have been

considered in [COY01] by restricting further the use of Boolean or separating connectives. Below we show that decision problems for kSL0($-\!\!*$) are PSPACE-hard too, assuming that the atomic formulae are of the form $\mathtt{alloc(x)}$ and $\mathtt{x = y}$ (see a precise formulation of the fragment in Lemma 2.1.6).

Let $Q_1\ p_1\ \cdots\ Q_n\ p_n\ \varphi$ be a QBF formula with $\{Q_1, \ldots, Q_n\} \subseteq \{\exists, \forall\}$ and φ is a propositional formula built over the atomic propositions in $\{p_1, \ldots, p_n\}$ and the Boolean connectives $\wedge$, $\vee$ and $\neg$ (only in front of atomic propositions). This time, the truth of the atomic proposition p_i is encoded by the truth of $\mathtt{alloc}(\mathtt{x}_i^\top)$ whereas the falsehood of p_i is encoded by the truth of $\mathtt{alloc}(\mathtt{x}_i^\bot)$; $\mathtt{x}_i^\top$ and $\mathtt{x}_i^\bot$ are new program variables associated to p_i. Obviously, $\mathtt{alloc}(\mathtt{x}_i^\top)$ and $\mathtt{alloc}(\mathtt{x}_i^\bot)$ may hold simultaneously, even if $\mathtt{x}_i^\top$ and $\mathtt{x}_i^\bot$ are interpreted differently. That is why, we introduce the formulae $\mathtt{init_i}$ and $\mathtt{ok_i}$ below. The formula $\mathtt{init_i}$ holds true when the encoding of the truth value of p_i is not yet done, i.e. none of $\mathtt{alloc}(\mathtt{x}_i^\bot)$ and $\mathtt{alloc}(\mathtt{x}_i^\top)$ holds true. Similarly, $\mathtt{ok_i}$ holds true when the encoding of the truth value of p_i is done, i.e. exactly one formula among $\mathtt{alloc}(\mathtt{x}_i^\bot)$ and $\mathtt{alloc}(\mathtt{x}_i^\top)$ holds true. This is generalized to sets of indices as defined below.

$$
\begin{array}{lcll}
\mathtt{init_i} & \stackrel{\text{def}}{=} & \neg\mathtt{alloc}(\mathtt{x}_i^\top) \wedge \neg\mathtt{alloc}(\mathtt{x}_i^\bot) & \\
\mathtt{ok_i} & \stackrel{\text{def}}{=} & (\mathtt{alloc}(\mathtt{x}_i^\top) \wedge \neg\mathtt{alloc}(\mathtt{x}_i^\bot)) \vee (\mathtt{alloc}(\mathtt{x}_i^\bot) \wedge \neg\mathtt{alloc}(\mathtt{x}_i^\top)) & \\
\mathtt{init_X} & \stackrel{\text{def}}{=} & \bigwedge_{j\in X} \mathtt{init_j} & (\text{with } X \subseteq [1,n]) \\
\mathtt{ok_X} & \stackrel{\text{def}}{=} & \bigwedge_{j\in X} \mathtt{ok_j} & (\text{with } X \subseteq [1,n]).
\end{array}
$$

The map tr defined below is homomorphic for the Boolean connectives $\wedge$ and $\vee$ and satisfies the following clauses [COY01]:

$$
\begin{array}{lcl}
tr(p_i) & \stackrel{\text{def}}{=} & \mathtt{alloc}(\mathtt{x}_i^\top) \\
tr(\neg p_i) & \stackrel{\text{def}}{=} & \mathtt{alloc}(\mathtt{x}_i^\bot) \\
tr(\forall\ p_i\ \varphi_{i+1}) & \stackrel{\text{def}}{=} & (\mathtt{ok_i} \wedge \mathtt{init}_{[1,n]\setminus\{i\}}) -\!\!* tr(\varphi_{i+1}) \\
tr(\exists\ p_i\ \varphi_{i+1}) & \stackrel{\text{def}}{=} & \sim((\mathtt{ok}_{[1,i-1]} \wedge \mathtt{init}_{[i,n]}) \wedge \sim((\mathtt{ok}_{[1,i]} \wedge \mathtt{init}_{[i+1,n]}) \wedge tr(\varphi_{i+1}))).
\end{array}
$$

with $\sim \psi \stackrel{\text{def}}{=} \psi -\!\!* \bot$. Whereas the encoding of the propositional quantification '$\forall\ p_i$' is rather natural with the help of the separating implication that performs a universal quantification too, the encoding of '$\exists\ p_i$' with a double use of $\sim$ is not immediate and reflects the beauty of the solution given in [COY01].

The translation tr takes advantage of the formulae of the form $\sim \psi$. Below, we present properties that will be helpful in the sequel. Their proof are left as Exercise 2.1. First, we define the relation $\approx_j$ (this slightly different from the one in

the previous PSPACE-hardness proof). Given a memory state $(\mathfrak{s}, \mathfrak{h})$ and a propositional valuation v, we write $(\mathfrak{s}, \mathfrak{h}) \approx_j v$ where the properties below are verified.

- For all $\mathtt{x}$ and $\mathtt{y}$ in $\{\mathtt{x}_i^\top, \mathtt{x}_i^\bot : i \in [1, n]\}$, we have $\mathfrak{s}(\mathtt{x}) \neq \mathfrak{s}(\mathtt{y})$ (this only depends on $\mathfrak{s}$).
- For all $i \in [j, n]$, we have $\{\mathfrak{s}(\mathtt{x}_i^\bot), \mathfrak{s}(\mathtt{x}_i^\top)\} \cap \mathrm{dom}(\mathfrak{h}) = \emptyset$ (this only depends on the memory state).
- For all $i \in [1, j-1]$, we have $(\mathfrak{s}(\mathtt{x}_i^\top) \in \mathrm{dom}(\mathfrak{h})$ and $\mathfrak{s}(\mathtt{x}_i^\bot) \notin \mathrm{dom}(\mathfrak{h})$ and $v(p_i) = \top)$ or $(\mathfrak{s}(\mathtt{x}_i^\bot) \in \mathrm{dom}(\mathfrak{h})$ and $\mathfrak{s}(\mathtt{x}_i^\top) \notin \mathrm{dom}(\mathfrak{h})$ and $v(p_i) = \bot)$.

Consequently, $(\mathfrak{s}, \mathfrak{h}) \approx_j v$ for some v implies $(\mathfrak{s}, \mathfrak{h}) \models \mathtt{ok}_{[1,j-1]} \wedge \mathtt{init}_{[j,n]}$.

Below, we state essential properties about $\sim$.

Lemma 2.1.4. . Let $(\mathfrak{s}, \mathfrak{h})$ be a memory state and φ, ψ be formulae in kSL0.

(I) $(\mathfrak{s}, \mathfrak{h}) \models\!\sim (\varphi \wedge \psi)$ iff for all heaps $\mathfrak{h}'$ disjoint from $\mathfrak{h}$, if $(\mathfrak{s}, \mathfrak{h}') \models \varphi$, then $(\mathfrak{s}, \mathfrak{h}') \not\models \psi$.

(II) $(\mathfrak{s}, \mathfrak{h}) \models\!\sim (\varphi \wedge\, \sim \psi)$ iff for all heaps $\mathfrak{h}'$ disjoint from $\mathfrak{h}$, if $(\mathfrak{s}, \mathfrak{h}') \models \varphi$, then there is a heap $\mathfrak{h}''$ disjoint from $\mathfrak{h}'$ such that $(\mathfrak{s}, \mathfrak{h}'') \models \psi$.

(III) Suppose that $(\mathfrak{s}, \mathfrak{h}) \approx_i v$ for some v. Then, $(\mathfrak{s}, \mathfrak{h}) \models\!\sim ((\mathtt{ok}_{[1,i-1]} \wedge \mathtt{init}_{[i,n]}) \wedge \sim ((\mathtt{ok}_{[1,i]} \wedge \mathtt{init}_{[i+1,n]}) \wedge \varphi))$ iff there is $\mathfrak{h}'$ such that $\mathfrak{h} \sqsubseteq \mathfrak{h}'$, $(\mathfrak{s}, \mathfrak{h}') \approx_{i+1} v'$ for some v' and $(\mathfrak{s}, \mathfrak{h}') \models \varphi$.

Correctness of the translation is stated below.

Lemma 2.1.5. The formula $Q_1\, p_1\, \cdots\, Q_n\, p_n\, \varphi$ is QBF satisfiable iff

$$\chi \stackrel{\text{def}}{=} \Big(\bigwedge_{\mathtt{x} \neq \mathtt{y} \in \{\mathtt{x}_i^\top, \mathtt{x}_i^\bot : i \in [1,n]\}} \mathtt{x} \neq \mathtt{y} \wedge \neg \mathtt{alloc}(\mathtt{x}) \Big) \wedge tr(Q_1\, p_1\, \cdots\, Q_n\, p_n\, \varphi)$$

is 1SL0 satisfiable.

Proof. For every $j \in [1, n{+}1]$, we write φ_j to denote the formula $Q_j\, p_j\, \cdots\, Q_n\, p_n\, \varphi$.

By induction on j, we show that for all $j \in [1, n+1]$, if $(\mathfrak{s}, \mathfrak{h}) \approx_j v$, then $(\mathfrak{s}, \mathfrak{h}) \models tr(\varphi_j)$ iff $v \models \varphi_j$. The base case in the induction corresponds to $j = n + 1$ and therefore the induction step goes backwards.

Before providing the proof by induction, let us check that this is sufficient to establish the statement in the lemma. If $(\mathfrak{s}, \mathfrak{h}) \approx_1 v$, then $(\mathfrak{s}, \mathfrak{h})$ satisfies

$$\Big(\bigwedge_{\mathtt{x}\neq\mathtt{y}\in\{\mathtt{x}_i^\top,\mathtt{x}_i^\bot : i\in[1,n]\}} \mathtt{x} \neq \mathtt{y} \wedge \neg\mathtt{alloc}(\mathtt{x})\Big).$$

Suppose that $v \models Q_1\ p_1 \cdots Q_n\ p_n\ \varphi$. Let us consider the memory state $(\mathfrak{s}, \mathfrak{h})$ such that: for all $i \in [1, n]$, we have $\mathfrak{s}(\mathtt{x}_i^\top) = 2i$, $\mathfrak{s}(\mathtt{x}_i^\bot) = 2i + 1$ and $\mathrm{dom}(\mathfrak{h}) = \emptyset$. Obviously, $(\mathfrak{s}, \mathfrak{h}) \approx_1 v$ and therefore by the property above, we get $(\mathfrak{s}, \mathfrak{h}) \models tr(\varphi_1)$, that is $(\mathfrak{s}, \mathfrak{h}) \models tr(\varphi)$. Consequently, $(\mathfrak{s}, \mathfrak{h}) \models \chi$. Now suppose that $(\mathfrak{s}, \mathfrak{h}) \models \chi$. Let us take any propositional valuation v. We have $(\mathfrak{s}, \mathfrak{h}) \approx_1 v$ and therefore by the property above, we get $v \models \varphi_1$, that is $v \models \varphi$.

Now let us consider the proof of the above property.

Base case: $j = n + 1$.

So, $\varphi_j = \varphi_{n+1} = \varphi$. The proof is by structural induction but the cases in the induction step with $\vee$ and $\wedge$ are by an easy verification. We assume that $(\mathfrak{s}, \mathfrak{h}) \approx_{n+1} v$ and let us consider p_i with $i < n + 1$. If $v \models p_i$, then $v(p_i) = \top$ and since $(\mathfrak{s}, \mathfrak{h}) \approx_{n+1} v$, we get $\mathfrak{s}(\mathtt{x}_i^\top) \in \mathrm{dom}(\mathfrak{h})$ and $(\mathfrak{s}, \mathfrak{h}) \models \mathtt{alloc}(\mathtt{x}_i^\top)$ $(= tr(p_i))$. If $v \models \neg p_i$, then $v(p_i) = \bot$ and since $(\mathfrak{s}, \mathfrak{h}) \approx_{n+1} v$, we get $\mathfrak{s}(\mathtt{x}_i^\bot) \in \mathrm{dom}(\mathfrak{h})$ and $(\mathfrak{s}, \mathfrak{h}) \models \mathtt{alloc}(\mathtt{x}_i^\bot)$ $(= tr(\neg p_i))$. Conversely, if $(\mathfrak{s}, \mathfrak{h}) \models tr(p_i)$, then $\mathfrak{s}(\mathtt{x}_i^\top) \in \mathrm{dom}(\mathfrak{h})$ and since $(\mathfrak{s}, \mathfrak{h}) \approx_{n+1} v$ we get $v(p_i) = \top$, whence $v \models p_i$. Similarly, if $(\mathfrak{s}, \mathfrak{h}) \models tr(\neg p_i)$, then $\mathfrak{s}(\mathtt{x}_i^\bot) \in \mathrm{dom}(\mathfrak{h})$ and since $(\mathfrak{s}, \mathfrak{h}) \approx_{n+1} v$ we get $v(p_i) = \bot$, whence $v \models \neg p_i$.

For the induction step with $j < n + 1$, below we deal with the case $\varphi_j = \exists\ p_j\ \varphi_{j+1}$. The case $\varphi_j = \forall\ p_j\ \varphi_{j+1}$ is omitted and it is left as Exercise 2.2. We assume that $(\mathfrak{s}, \mathfrak{h}) \approx_j v$.

First suppose that $v \models \exists\ p_j\ \varphi_{j+1}$. This means that there is $b \in \{\top, \bot\}$ such that $v[p_j \mapsto b] \models \varphi_{j+1}$. Let $\mathfrak{h}'$ be such that $\mathrm{dom}(\mathfrak{h}') = \{\mathfrak{s}(\mathtt{x}_j^b)\}$ and $\mathfrak{h}'(\mathfrak{s}(\mathtt{x}_j^b)) = 0$ (arbitrary value). Since $(\mathfrak{s}, \mathfrak{h}) \approx_j v$, we obtain $(\mathfrak{s}, \mathfrak{h} \uplus \mathfrak{h}') \approx_{j+1} v[p_j \mapsto \bot]$ and therefore by the induction hypothesis, we have $(\mathfrak{s}, \mathfrak{h} \uplus \mathfrak{h}') \models tr(\varphi_{j+1})$. By Lemma 2.1.4(III), $(\mathfrak{s}, \mathfrak{h}) \models\sim ((\mathtt{ok}_{[1,j-1]} \wedge \mathtt{init}_{[j,n]}) \wedge \sim ((\mathtt{ok}_{[1,j]} \wedge \mathtt{init}_{[j+1,n]}) \wedge tr(\varphi_{j+1})))$ and therefore $(\mathfrak{s}, \mathfrak{h}) \models tr(\varphi_j)$.

Now suppose that $(\mathfrak{s}, \mathfrak{h}) \models\sim ((\mathtt{ok}_{[1,j-1]} \wedge \mathtt{init}_{[j,n]}) \wedge \sim ((\mathtt{ok}_{[1,j]} \wedge \mathtt{init}_{[j+1,n]}) \wedge tr(\varphi_{j+1})))$ and therefore $(\mathfrak{s}, \mathfrak{h}) \models tr(\varphi_j)$. By Lemma 2.1.4(III), there is $\mathfrak{h} \sqsubseteq \mathfrak{h}'$ such that $(\mathfrak{s}, \mathfrak{h}') \approx_{j+1} v'$ for some v' and $(\mathfrak{s}, \mathfrak{h}') \models tr(\varphi_{j+1})$. Note that v and v' agree on the atomic propositions p_i with $i \in [1, j - 1]$. By the induction hypothesis, we have $v' \models \varphi_{j+1}$ and therefore $v \models \exists\ p_j\ \varphi_{j+1}$. **QED**

Corollary 2.1.6. [COY01] For every $k \geq 1$, the satisfiability and model-checking problems for kSL0 restricted to formulae obeying the grammar below

$$\varphi ::= \neg(\mathtt{x} = \mathtt{y}) \mid \mathtt{alloc}(\mathtt{x}) \mid \neg\mathtt{alloc}(\mathtt{x}) \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \varphi \mathrel{-\!\!*} \varphi$$

(with $\mathtt{x}$, $\mathtt{y}$ in $\mathtt{PVAR}$) is PSPACE-hard.

Note that $\bot$ can be defined as $\mathtt{alloc}(\mathtt{x}) \wedge \neg\mathtt{alloc}(\mathtt{x})$ for some program variable $\mathtt{x}$. In the above-mentioned fragment, it is remarkable that negation appears only in front of atomic formulae (to be compared with the formulae involved in Lemma 2.1.2) and the separating conjunction is banished. The proof of Corollary 2.1.6 is similar to the proof of Corollary 2.1.3 and it is left as Exercise 2.3

2.1.2 Boolean formulae for propositional separation logics

In this section, we present a characterisation of the expressive power of propositional separation logic 1SL0, and a similar analysis can be done for any kSL0 with $k > 1$.

Theorem 2.1.7. [Loz04a, Chapter 5] Any formula φ in 1SL0 built over the program variables in $\{\mathtt{x}_1, \ldots, \mathtt{x}_q\}$ is logically equivalent to a Boolean combination of atomic formulae among $\mathtt{size} \geq k$, $\mathtt{alloc}(\mathtt{x}_i)$, $\mathtt{x}_i \hookrightarrow \mathtt{x}_j$ and $\mathtt{x}_i = \mathtt{x}_j$ ($k \in \mathbb{N}$, $i, j \in \{1, \ldots, q\}$).

The formulae of the form $\mathtt{size} \geq k$ and $\mathtt{alloc}(\mathtt{x}_i)$ are introduced in Section 1.2.2 and we recall that $\mathtt{alloc}(\mathtt{x}_i)$ holds when $\mathfrak{s}(\mathtt{x}_i)$ belongs to the heap domain and $\mathtt{size} \geq k$ holds when the cardinal of the heap domain is at least k. By way of example $(\neg\mathtt{emp} * (\mathtt{x}_1 \hookrightarrow \mathtt{x}_2 \mathrel{-\!\!*} \bot))$ is equivalent to $\mathtt{size} \geq 2 \wedge \mathtt{alloc}(\mathtt{x}_1)$. Furthermore, the cardinal of the heap domain without the interpretation of $\mathtt{x}_1$ and $\mathtt{x}_2$ (in the case it belongs to the domain) is at least $k \geq 0$, can be expressed as follows:

$$(\mathtt{alloc}(\mathtt{x}_1) \wedge \mathtt{alloc}(\mathtt{x}_2) \wedge \mathtt{size} \geq k + 2) \vee$$
$$(((\mathtt{alloc}(\mathtt{x}_1) \wedge \neg\mathtt{alloc}(\mathtt{x}_2)) \vee (\neg\mathtt{alloc}(\mathtt{x}_1) \wedge \mathtt{alloc}(\mathtt{x}_2))) \wedge \mathtt{size} \geq k + 1) \vee$$
$$(\neg\mathtt{alloc}(\mathtt{x}_1) \wedge \neg\mathtt{alloc}(\mathtt{x}_2) \wedge \mathtt{size} \geq k).$$

It is clear that such a formula can be generalised to any finite set of program variables. We write $\mathtt{size}_{\overline{q}} \geq k$ to denote the atomic formula such that $(\mathfrak{s}, \mathfrak{h}) \models \mathtt{size}_{\overline{q}} \geq k$ iff $\mathrm{card}(\mathrm{dom}(\mathfrak{h}) \setminus \{\mathfrak{s}(\mathtt{x}_i) : i \in [1, q]\}) \geq k$. The formula $\mathtt{size} \geq k$ can be expressed as follows:

$$(\mathtt{x}_1 \neq \mathtt{x}_2 \wedge \mathtt{alloc}(\mathtt{x}_1) \wedge \mathtt{alloc}(\mathtt{x}_2) \wedge \mathtt{size}_{\overline{2}} \geq k - 2) \vee$$

$$(\mathtt{x}_1 = \mathtt{x}_2 \wedge \mathtt{alloc}(\mathtt{x}_1) \wedge \mathtt{size}_{\overline{2}} \geq k-1) \vee$$
$$(((\mathtt{alloc}(\mathtt{x}_1) \wedge \neg\mathtt{alloc}(\mathtt{x}_2)) \vee (\neg\mathtt{alloc}(\mathtt{x}_1) \wedge \mathtt{alloc}(\mathtt{x}_2))) \wedge \mathtt{size}_{\overline{2}} \geq k-1) \vee$$
$$(\neg\mathtt{alloc}(\mathtt{x}_1) \wedge \neg\mathtt{alloc}(\mathtt{x}_2) \wedge \mathtt{size}_{\overline{2}} \geq k).$$

Such a formula can be generalised to any $q \geq 1$. So using atomic formulae of the form $\mathtt{size} \geq k$ or $\mathtt{size}_{\overline{q}} \geq k$ does not make a substantial difference in terms of expressive power.

Even though Theorem 2.1.7 provides a nice characterisation of the expressive power for 1SL0, several features limit its application. First, Theorem 2.1.7 only deals with the propositional case but we know that this is close to the best we can hope for. Indeed, a similar result is established in [DGLWM14] for 1SL1 by enriching the set of atomic formulae and by polishing and extending material from [Loz04a, BDL09] but the extension to 1SL2 is not possible (see developments in Chapter 3). Moreover, neither Theorem 2.1.7 states how to compute the equivalent formula nor it provides a precise information about the maximal bound k in atomic formulae $\mathtt{size} \geq k$ that are used to build a Boolean combination equivalent to φ in 1SL0 (see Corollary 5.3.12). Actually, one can restrict k to be at most polynomial in the size of φ, assuming that formulae are encoded as finite trees (as opposed to a DAG encoding that would imply an exponential blow-up). This entails a small model property in which the cardinal of the heap domain is bounded, see e.g. [COY01, CGH05] or Section 5.3. This feature is at the core of the translation into first-order logic (with empty signature) designed in [CGH05] and it regains the PSPACE upper bound for the satisfiability problem for 1SL0 (and for 2SL0 too), see e.g. [CGH05, Section 3.4].

Below, let us be a bit more precise about the way to prove Theorem 2.1.7 and to explain the main steps to show the PSPACE upper bound, which is reminiscent to many proofs showing PSPACE upper bound for modal logics by using Ladner-like algorithms, see e.g. [Lad77, Spa93, Dem03]. More details can be found in Section 5.3. Let $q \geq 1$ and $\alpha \in \mathbb{N}$. We write $\mathtt{Test}'(q, \alpha)$ to denote the following set of atomic formulae:

$$\{\mathtt{x}_i = \mathtt{x}_j, \mathtt{x}_i \hookrightarrow \mathtt{x}_j, \mathtt{alloc}(\mathtt{x}_i) : i, j \in [1, q]\} \cup \{\mathtt{size}_{\overline{q}} \geq \beta : \beta \in [0, \alpha]\}.$$

We define an equivalence relation $\approx^q_\alpha$ on the class of memory states, so that two models are in the same equivalence class whenever they cannot be distinguished by any formula in $\mathtt{Test}'(q, \alpha)$: $(\mathfrak{s}, \mathfrak{h}) \approx^q_\alpha (\mathfrak{s}', \mathfrak{h}')$ iff

$$\text{for all } \psi \in \mathtt{Test}'(q, \alpha), \text{ we have } (\mathfrak{s}, \mathfrak{h}) \models \psi \text{ iff } (\mathfrak{s}', \mathfrak{h}') \models \psi.$$

One can show that for any formula φ in 1SL0 with $q \geq 1$ program variables and with size $|\varphi|$ (for some reasonably succinct encoding), for any $\alpha \geq |\varphi|$, if $(\mathfrak{s}, \mathfrak{h}) \approx^{q}_{\alpha} (\mathfrak{s}', \mathfrak{h}')$, then $(\mathfrak{s}, \mathfrak{h}) \models \varphi$ iff $(\mathfrak{s}', \mathfrak{h}') \models \varphi$. This result or some of its variants established in [Loz04a, BDL09, DGLWM14] entails that for checking the satisfaction of φ in some memory state, what matters is really the satisfaction of atomic formulae in $\approx^{q}_{|\varphi|}$. Theorem 2.1.7 is then a direct consequence of this property.

Corollary 2.1.8. [COY01, Yan01] Let φ be a satisfiable formula in 1SL0 with q program variables. Then there is memory state $(\mathfrak{s}, \mathfrak{h})$ such that $(\mathfrak{s}, \mathfrak{h}) \models \varphi$ and $\mathrm{ran}(\mathfrak{s}) \cup \mathrm{dom}(\mathfrak{h}) \cup \mathrm{ran}(\mathfrak{h}) \subseteq [0, q + |\varphi|]$.

PSPACE upper bound for 1SL0 can be pushed a bit further by allowing a unique quantified variable.

Theorem 2.1.9. [DGLWM14] The satisfiability problem for 1SL1 is PSPACE-complete.

PSPACE-hardness is inherited from the PSPACE-hardness of 1SL0 whereas the PSPACE upper bound requires an adequate abstraction. It is open whether 1SL1 extended with reachability predicates can lead to decidable extensions (which would capture some version of separation logic considered in [TBR14]).

2.2 NP and PTIME Fragments

Even though performing reasoning in propositional logic kSL0 (with $k \geq 1$) can be computationally expensive, see above the PSPACE-completeness results for validity and satisfiability, fragments have been designed that are useful for automatic program analysis and hopefully less demanding computationally.

The fragment presented below, has been introduced in [BCO04] and shown decidable by providing a complete proof system. More importantly, the tool Smallfoot has been designed from it, see e.g. [BCO05], and decides the entailment problem for such a fragment, which allows to verify automatically numerous properties. Strangely enough, the precise computational complexity of the entailment problem for such a fragment is not considered in [BCO04] and it is only in [CHO$^+$11, HIOP13] that this problem has been successfully solved.

Let SF ('Smallfoot fragment') be the fragment of 1SL2 defined by the formula φ below, where φ_p defines **pure formulae** (see also Section 1.2.4 for the

introduction of pure formulae semantically) and φ_s defines **spatial formulae**:

$$\varphi_p ::= \bot \mid \top \mid (\mathtt{x}_i = \mathtt{x}_j) \mid \neg(\mathtt{x}_i = \mathtt{x}_j) \mid \varphi_p \wedge \varphi_p$$

$$\varphi_s ::= \mathtt{emp} \mid \top \mid \mathtt{x}_i \mapsto \mathtt{x}_j \mid \mathtt{sreach}(\mathtt{x}_i, \mathtt{x}_j) \mid \varphi_s * \varphi_s \qquad \varphi ::= \varphi_p * \varphi_s$$

where $\mathtt{x}_i$, $\mathtt{x}_j$ are program variables from PVAR. As usually, the formulae are interpreted on memory states with one record field. Obviously, $\mathtt{x}_i \mapsto \mathtt{x}_j$ is interpreted as the exact points-to relation $((\mathfrak{s}, \mathfrak{h}) \models \mathtt{x}_i \mapsto \mathtt{x}_j$ iff $\mathrm{dom}(\mathfrak{h}) = \mathfrak{s}(\mathtt{x}_i)$ and $\mathfrak{h}(\mathfrak{s}(\mathtt{x}_i)) = \mathfrak{s}(\mathtt{x}_j))$ whereas $(\mathfrak{s}, \mathfrak{h}) \models \mathtt{sreach}(\mathtt{x}_i, \mathtt{x}_j)$ holds true iff the heap contains exactly a path from $\mathfrak{s}(\mathtt{x}_i)$ to $\mathfrak{s}(\mathtt{x}_j)$. As shown in Section 1.2.2, $\mathtt{sreach}(\mathtt{x}_i, \mathtt{x}_j)$ (and $\mathtt{reach}(\mathtt{x}_i, \mathtt{x}_j)$ too) can be specified in 1SL2.

We briefly recall that the entailment problem for SF takes as input two SF formulae φ and ψ, and asks whether $\varphi \models \psi$. Note also that the rule for strengthening precedent (SP)

$$\frac{\varphi \Rightarrow \psi' \quad \{\psi'\}\ \mathtt{C}\ \{\psi\}}{\{\varphi\}\ \mathtt{C}\ \{\psi\}}$$

involves entailment checking. This is a building block of the verification process and in particular, proof checking requires that entailment problem is decidable, if not tractable at all.

Whereas a coNP algorithm is provided in [BCO04], the optimal complexity is established in [CHO$^+$11] by using an original approach: to represent formulae as graphs and to search for homomorphisms on these special graphs.

Theorem 2.2.1. [CHO$^+$11, Theorems 16 & 24] (see also [GKO11, Section 4]) The entailment and satisfiability problems for SF can be solved in polynomial time.

Indeed, it is quite surprising that the entailment problem is computationally tractable. A slight extension may easily lead to intractability. For instance considering the variant clause $\varphi ::= \varphi_p * (\varphi_s \wedge \varphi'_s)$ (i.e., allowing a bit of conjunction) already leads to coNP-hardness [CHO$^+$11]. The graph-based algorithm presented in [CHO$^+$11] has been implemented and used for automatic verification, see [HIOP13].

2.3 Undecidable Propositional Separation Logics

2.3.1 A brief introduction to abstract separation logics

Concrete models for separation logic are memory states or heaps as defined earlier, but alternative models exist, for instance heaps with permissions, see e.g. [BCOP05, BK14]. It is also possible to introduce more abstract models with a partial operator for gluing together models that are separate in some sense. We have already seen such abstract models in Section 1.3.1 when BBI-models have been introduced.

This is precisely the approach introduced in [COY07] and investigated in great length in subsequent papers, see e.g. [BK10, LWG10, BV14, HCGT14]. After all, such an abstraction should not come as a surprise since separation logic is understood as an assertion language in a Hoare-style framework that interprets Boolean BI in concrete heaps (see Section 1.3.1). Moreover, sometimes, problems can be easily solved on abstract models because more freedom is allowed (see e.g. [BV14, HCGT14] or Theorem 1.3.5).

The structure $(\mathfrak{H}\mathfrak{S}_k, \uplus, \mathfrak{U}_k)$ satisfies the following properties.

(MON$_{\text{ms}}$) $\uplus$ is a partial binary operation $\uplus : \mathfrak{H}\mathfrak{S}_k \times \mathfrak{H}\mathfrak{S}_k \to \mathfrak{H}\mathfrak{S}_k$ and $\mathfrak{U}_k \subseteq \mathfrak{H}\mathfrak{S}_k$,

(AC$_{\text{ms}}$) $\uplus$ is associative and commutative,

(CAN$_{\text{ms}}$) $\uplus$ is cancellative, i.e. if $(\mathfrak{s}, \mathfrak{h}) \uplus (\mathfrak{s}', \mathfrak{h}')$ is defined and $(\mathfrak{s}, \mathfrak{h}) \uplus (\mathfrak{s}', \mathfrak{h}') = (\mathfrak{s}, \mathfrak{h}) \uplus (\mathfrak{s}'', \mathfrak{h}'')$, then $(\mathfrak{s}', \mathfrak{h}') = (\mathfrak{s}'', \mathfrak{h}'')$,

(U$_{\text{ms}}$) for all $(\mathfrak{s}, \mathfrak{h}) \in \mathfrak{H}\mathfrak{S}_k$, we have $\{(\mathfrak{s}, \mathfrak{h})\} = \{(\mathfrak{s}, \mathfrak{h}) \uplus (\mathfrak{s}', \mathfrak{h}') : (\mathfrak{s}', \mathfrak{h}') \in \mathfrak{U}_k,\ (\mathfrak{s}, \mathfrak{h}) \uplus (\mathfrak{s}', \mathfrak{h}') \text{ is defined}\}$.

A **separation model** defined below satisfies the above properties for the structure $(\mathfrak{H}\mathfrak{S}_k, \uplus, \mathfrak{U}_k)$ by abstracting the essential features and can be viewed as a Kripke frame for a multi-dimensional modal logic with binary modalities, see e.g. [MV97, HCGT14]. A **separation model** is a cancellative partial commutative monoid $(M, \circ, U)$, i.e.

(MON) M is a non-empty set, $\circ$ is a partial binary operation $\circ : M \times M \to M$ and $U \subseteq M$,

(AC) $\circ$ is associative and commutative,

(CAN) $\circ$ is cancellative, i.e. if $m \circ m'$ is defined and $m \circ m' = m \circ m''$, then $m' = m''$,

(U) For all $m \in M$, we have $m \circ U = \{m\}$ where $m \circ U \stackrel{\text{def}}{=} \{m \circ u : u \in U,\ m \circ u \text{ is defined}\}$.

Obviously $(\mathfrak{H}\mathfrak{S}_k, \uplus, \mathfrak{U}_k)$ is a separation model but other memory models can be found in the literature, see e.g. [BK10] for many more examples. For instance, the **RAM-domain model** $(\mathcal{P}_{\text{fin}}(\mathbb{N}), \uplus, \{\emptyset\})$ is a separation model where $\mathcal{P}_{\text{fin}}(\mathbb{N})$ is the set of finite subsets of $\mathbb{N}$ and $X_1 \uplus X_2$ is defined only if $X_1 \cap X_2 = \emptyset$ and then $X_1 \uplus X_2 \stackrel{\text{def}}{=} X_1 \cup X_2$ (disjoint union). This corresponds to the separation model $(\mathfrak{H}\mathfrak{S}_k, \uplus, \mathfrak{U}_k)$ with k equal to zero.

Given a countably infinite set PROP $= \{p_1, p_2, \ldots\}$ of propositional variables, a valuation $\mathfrak{V}$ is a map $\mathfrak{V} : \text{PROP} \to \mathcal{P}(M)$. Semantical structures of the separation model $(M, \circ, U)$ are understood as the separation model itself augmented by a valuation. Hence, the separation logic defined from the separation model $(M, \circ, U)$ has models that can be understood as Kripke models with underlying ternary relation induced by the operation $\circ$ and the interpretation of propositional variables done via $\mathfrak{V}$. The set of formulae is then defined as follows:

$$\varphi, \psi ::= \texttt{emp} \mid p \mid \varphi \wedge \psi \mid \neg\varphi \mid \varphi * \psi \mid \varphi \mathrel{-\!\!*} \psi.$$

Let $m \in M$ and $\mathfrak{V} : \text{PROP} \to \mathcal{P}(M)$ be a valuation, the satisfaction relation $\models$ is defined as follows (we omit the obvious clauses for Boolean connectives).

- $m \models_{\mathfrak{V}} \texttt{emp}$ iff $m \in U$ (we keep the constant emp in the abstract setting but elements of U should be understood as units).
- $m \models_{\mathfrak{V}} p$ iff $m \in \mathfrak{V}(p)$.
- $m \models_{\mathfrak{V}} \varphi_1 * \varphi_2$ iff for some $m_1, m_2 \in M$, we have $m = m_1 \circ m_2$, $m_1 \models_{\mathfrak{V}} \varphi_1$ and $m_2 \models_{\mathfrak{V}} \varphi_2$.
- $m \models_{\mathfrak{V}} \varphi_1 \mathrel{-\!\!*} \varphi_2$ iff for all $m' \in M$ such that $m \circ m'$ is defined, if $m' \models_{\mathfrak{V}} \varphi_1$ then $m \circ m' \models_{\mathfrak{V}} \varphi_2$.

In the above definition for the satisfaction relation, the model $(M, \circ, U)$ is implicit but we also sometimes use the notation $(M, \circ, U), m \models_{\mathfrak{V}} \varphi$ to emphasise the separation model in use. The satisfaction relation on BBI-models is clearly defined following the same schema (see Section 1.3.1).

A formula φ is **valid** in the separation model $(M, \circ, U) \stackrel{\text{def}}{\Leftrightarrow}$ for all $m \in M$ and for all valuations $\mathfrak{V}$, we have $m \models_{\mathfrak{V}} \varphi$. Similarly, a formula φ is **satisfiable** in the separation model $(M, \circ, U) \stackrel{\text{def}}{\Leftrightarrow}$ there exist $m \in M$ and a valuation $\mathfrak{V}$ such

that $m \models_{\mathfrak{V}} \varphi$. We write SL$(M, \circ, U)$ to denote the propositional separation logic defined from the separation model $(M, \circ, U)$ with propositional variables. When C is a class of separation models, we can also define the propositional separation logic SL(C) by admitting a family of separation models instead of a single model. Satisfiability and validity problems are defined accordingly. For instance, φ is satisfiable for SL(C) iff there exist $(M, \circ, U)$ in C, $m \in M$ and a valuation $\mathfrak{V}$ such that $(M, \circ, U), m \models_{\mathfrak{V}} \varphi$.

The satisfiability problem for kSL0 (i.e. kSL without any first-order quantification) can be reformulated as the satisfiability problem in the separation model $(\mathfrak{H}\mathfrak{S}_k, \uplus, \mathfrak{U}_k)$ in which propositional variables are of the form $\mathtt{x}_i \hookrightarrow \mathtt{x}_j$ or $\mathtt{x}_i = \mathtt{x}_j$ and the valuations $\mathfrak{V}$ are constrained in such a way that $(\mathfrak{s}, \mathfrak{h}) \in \mathfrak{V}(\mathtt{x}_i \hookrightarrow \mathtt{x}_j)$ iff $\mathfrak{h}(\mathfrak{s}(\mathtt{x}_i)) = \mathfrak{s}(\mathtt{x}_j)$. Similarly, we require that $(\mathfrak{s}, \mathfrak{h}) \in \mathfrak{V}(\mathtt{x}_i = \mathtt{x}_j)$ iff $\mathfrak{s}(\mathtt{x}_i) = \mathfrak{s}(\mathtt{x}_j)$. Of course, this reformulation assumes that atomic formulae have some structure and it also requires restricting the set of valuations. The set of valuations can be restricted in many other ways, for instance by imposing that a propositional variable holds true only for a finite number of elements of M (see such restrictions in [BK10]).

2.3.2 Encoding runs of Minsky machines

Whereas the satisfiability problem for any propositional fragment kSL0 is decidable and indeed PSPACE-complete (see Section 2.1), propositional versions of abstract separation logic with propositional variables are easily shown undecidable.

Theorem 2.3.1.
[BK10, LWG10] The satisfiability problems for SL$(\mathcal{P}_{\text{fin}}(\mathbb{N}), \uplus, \{\emptyset\})$ and for SL$(\mathfrak{H}\mathfrak{S}_k, \uplus, \mathfrak{U}_k)$ –$k \geq 1$– are undecidable.

Actually, results in [BK10, LWG10] are much more general. Herein, we limit ourselves to two separation models that are obviously related to concrete heaps. Below, by way of example, we provide the undecidability proof for the logic SL$(\mathcal{P}_{\text{fin}}(\mathbb{N}), \uplus, \{\emptyset\})$ by simple semantical arguments (and without using any proof-theoretical arguments, unlike what is done in [BK10, LWG10]).

Before presenting the undecidability proof, let us mention the equivalence of the statements below:

1. φ is valid in SL$(\mathcal{P}_{\text{fin}}(\mathbb{N}), \uplus, \{\emptyset\})$.

2. $\neg\varphi$ is not satisfiable in $\mathrm{SL}(\mathcal{P}_{\mathrm{fin}}(\mathbb{N}), \uplus, \{\emptyset\})$.

3. $\neg\varphi$ is not satisfiable in $\mathrm{SL}(\mathfrak{H}\mathfrak{S}_k, \uplus, \mathfrak{U}_k)$ (for any $k \geq 1$).

4. φ is valid in $\mathrm{SL}(\mathfrak{H}\mathfrak{S}_k, \uplus, \mathfrak{U}_k)$ (for any $k \geq 1$).

Whereas the equivalences between instances for validity and satisfiability are standard (thanks to negation in the logical language), the equivalences related to distinct separation models are simply due to the fact, in such logics, composition of heaps only requires that the domain are disjoint, independently of the range of the heaps. Note also that $\mathrm{SL}(\mathcal{P}_{\mathrm{fin}}(\mathbb{N}), \uplus, \{\emptyset\})$ can be understood as the logic $\mathrm{SL}(\mathfrak{H}\mathfrak{S}_k, \uplus, \mathfrak{U}_k)$ with k equal to zero.

Let $\mathtt{M}$ be a Minsky machine with $\alpha \geq 1$ instructions, 1 is the initial instruction and α is the halting instruction [Min67]. Machine $\mathtt{M}$ has two counters c_1 and c_2 and the instructions are of the following types ($j \in [1,2]$, $I \in [1, \alpha - 1]$, $J, J_1, J_2 \in [1, \alpha]$):

1. I: $\mathsf{c}_j := \mathsf{c}_j + 1$; goto J.

2. I: if $\mathsf{c}_j = 0$ then goto J_1 else ($\mathsf{c}_j := \mathsf{c}_j - 1$; goto J_2).

3. α: halt.

Machine $\mathtt{M}$ halts if there is a run of the form $(I_0, c_0^1, c_0^2), (I_1, c_1^1, c_1^2), \ldots, (I_L, c_L^1, c_L^2)$ such that $(I_i, c_i^1, c_i^2) \in [1, \alpha] \times \mathbb{N}^2$ ($i \in [1, L]$), the succession of configurations respects the instructions (in the obvious way), $I_0 = 1$, $I_L = \alpha$, and $c_0^1 = c_0^2 = 0$. The halting problem consists in checking whether a machine halts and it is known to be undecidable, see e.g. [Min67]. Indeed, Minsky machines are Turing-complete.

By way of example, the Minsky machine

1: $\mathsf{c}_1 := \mathsf{c}_1 + 1$; goto 2.

2: $\mathsf{c}_2 := \mathsf{c}_2 + 1$; goto 1.

3: halt.

has a unique computation

$$(1,0,0) \to (2,1,0) \to (1,1,1) \to (2,2,1) \to (1,2,2) \to (2,3,2) \ldots$$

We build a formula $\varphi_{\mathtt{M}}$ such that $\mathtt{M}$ halts iff $\varphi_{\mathtt{M}}$ is valid in $\mathrm{SL}(\mathcal{P}_{\mathrm{fin}}(\mathbb{N}), \uplus, \{\emptyset\})$, which entails the undecidability of the satisfiability problem for $\mathrm{SL}(\mathcal{P}_{\mathrm{fin}}(\mathbb{N}), \uplus, \{\emptyset\})$.

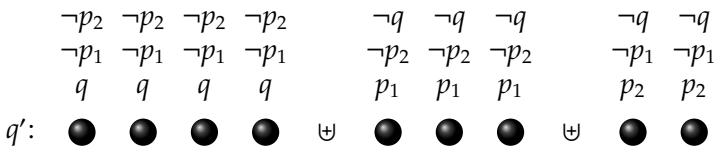


Figure 2.1: Set encoding of the configuration $(4, 3, 2)$

The formula $\varphi_{\mathtt{M}}$ is built over the propositional variables q, q', p_1 and p_2. Given a valuation $\mathfrak{V}$, a configuration (I, c_1, c_2) of $\mathtt{M}$ is encoded by some set $X \in \mathcal{P}_{\text{fin}}(\mathbb{N})$ such that

- $X \in \mathfrak{V}(q')$ (meaning X encodes a configuration),
- $X = X_0 \uplus X_1 \uplus X_2$ (X can be decomposed so that there are disjoint parts about the instruction counter, the first counter and the second counter),
- $\text{card}(X_0) = I$, $\text{card}(X_1) = c_1$ and $\text{card}(X_2) = c_2$,
- for all $\emptyset \neq Y \subseteq X_0$, $Y \in \mathfrak{V}(q) \setminus (\mathfrak{V}(p_1) \cup \mathfrak{V}(p_2))$,
- for all $\emptyset \neq Y \subseteq X_1$, $Y \in \mathfrak{V}(p_1) \setminus (\mathfrak{V}(p_2) \cup \mathfrak{V}(q))$,
- for all $\emptyset \neq Y \subseteq X_2$, $Y \in \mathfrak{V}(p_2) \setminus (\mathfrak{V}(p_1) \cup \mathfrak{V}(q))$.

In that case, we write $X \approx_{\mathfrak{V}} (I, c_1, c_2)$. The basic idea is that the atomic proposition p_j identifies the sets that contribute to the value for the counter $\mathtt{c}_j$ whereas the atomic proposition q identifies the sets that contribute to the value of the instruction counter. Furthermore, we require more than that:

1. $X \in \mathfrak{V}(p)$ implies that none of the strict non-empty subsets of X belongs to $\mathfrak{V}(p')$ with $p' \neq p$ and all its strict non-empty subsets belongs to $\mathfrak{V}(p)$.
2. The empty set is the only one satisfying both p_1 and p_2, which should not come as a surprise since both counters can take the zero value.

Figure 2.1 illustrates how the configuration $(4, 3, 2)$ can be encoded as a set with the corresponding valuation.

The formula $\varphi_{\mathtt{M}}$ has the following form:

$$((\mathtt{emp} \wedge p_1 \wedge p_2 \wedge \neg q \wedge \neg q') \wedge \mathtt{closure}) \Rightarrow (\top \mathrel{-\!\!\!\overline{\ast}} (q' \wedge (p_1 * p_2 * (\mathtt{size} = \alpha \wedge q))))$$

The formula $\mathtt{closure}$ guarantees that for any configuration (I, c_1, c_2) reachable from the initial configuration $(1, 0, 0)$, there is some $X \in \mathcal{P}_{\text{fin}}(\mathbb{N})$ such that $X \approx_{\mathfrak{V}} (I, c_1, c_2)$ (in that case, note that $\text{card}(X) = I + c_1 + c_2$).

The formula $\top \mathrel{-\!\!\!\!\!\!\rightarrow\!\!\!\!\!\!\rightarrow} (q' \wedge (p_1 * p_2 * (\mathtt{size} = \alpha \wedge q)))$ states that there is $X \in \mathcal{P}_{\text{fin}}(\mathbb{N})$ such that $X = X_0 \uplus X_1 \uplus X_2$, $\text{card}(X_0) = \alpha$, X_1 encodes the first counter and X_2 encodes the second counter.

In order to define the formula $\mathtt{closure}$, we introduce the universal modalities $\langle \mathtt{U} \rangle$ and $[\mathtt{U}]$. Let $\langle \mathtt{U} \rangle\ \psi$ be an abbreviation for $\top \mathrel{-\!\!\!\!\!\!\rightarrow\!\!\!\!\!\!\rightarrow} \psi$ and $[\mathtt{U}]\ \psi$ be an abbreviation for $\top \mathrel{-\!\!*} \psi$, following an obvious analogy with the universal modality in Kripke models, see e.g. [GP92, Hem96]. The formula $\mathtt{closure}$ is defined as the conjunction of the following formulae:

- $\langle \mathtt{U} \rangle\ (\mathtt{size} = 1 \wedge q \wedge q')$. There is some set X encoding the configuration $(1, 0, 0)$.
- $[\mathtt{U}](p_1 \Rightarrow (\neg((\neg p_1 \wedge \neg \mathtt{emp}) * \top) \wedge (\neg \mathtt{emp} \Rightarrow \neg p_2) \wedge \neg q \wedge \neg q')$.
- $[\mathtt{U}](p_2 \Rightarrow (\neg((\neg p_2 \wedge \neg \mathtt{emp}) * \top) \wedge (\neg \mathtt{emp} \Rightarrow \neg p_1) \wedge \neg q \wedge \neg q')$.
- $[\mathtt{U}](q \Rightarrow (\neg((\neg q \wedge \neg \mathtt{emp}) * \top)) \wedge \neg p_1 \wedge \neg p_2)$.

In the sequel, the modalities $\langle \mathtt{U} \rangle$ and $[\mathtt{U}]$ are used at the outermost level only and therefore they are evaluated only on the empty set. More generally, the universal modality $[\mathtt{U}]$ can be defined $[\mathtt{U}]\varphi \stackrel{\text{def}}{=} (\mathtt{emp} \wedge (\top \mathrel{-\!\!*} \varphi)) * \top$ (see also Exercise 1.13). Consequently, whenever $X \models_{\mathfrak{V}} (q \wedge \mathtt{size} = I) * p_1 * p_2$ for some $I \in [1, \alpha]$, there is no $I' \neq I$ such that $X \models_{\mathfrak{V}} (q \wedge \mathtt{size} = I') * p_1 * p_2$. Moreover, there are unique X_0, X_1 and X_2 such that $X = X_0 \uplus X_1 \uplus X_2$, $X_0 \models_{\mathfrak{V}} (q \wedge \mathtt{size} = I)$, $X_1 \models_{\mathfrak{V}} p_1$ and $X_2 \models_{\mathfrak{V}} p_2$. We add to $\mathtt{closure}$ the following formulae:

- For all instructions of the form I: $\mathtt{c}_1 := \mathtt{c}_1 + 1$; goto J, we consider

$$[\mathtt{U}]((((q \ \wedge\ \mathtt{size} = I) * p_1 * p_2) \wedge q') \Rightarrow$$

$$(q \ \wedge\ \mathtt{size} = I) * (((q \ \wedge\ \mathtt{size} = J) * (\mathtt{size} = 1 \wedge p_1)) \mathrel{-\!\!\!\!\!\!\rightarrow\!\!\!\!\!\!\rightarrow}$$

$$(((q \ \wedge\ \mathtt{size} = J) * p_1 * p_2) \wedge q')))$$

- Formulae for instructions of the form I: $\mathtt{c}_2 := \mathtt{c}_2 + 1$; goto J are defined similarly.

- For all instructions of the form I: if $\mathtt{c}_1 = 0$ then goto J_1 else ($\mathtt{c}_1 := \mathtt{c}_1 - 1$; goto J_2), we consider

$$[\mathtt{U}]((((q \wedge \mathtt{size} = I) * p_2) \wedge q') \Rightarrow ((q \wedge \mathtt{size} = I) * ((q \wedge \mathtt{size} = J_1) \mathrel{-\!\!\!\!\!\!\rightarrow} q'))) \wedge$$

$$[\mathtt{U}]((((q \wedge \mathtt{size} = I) * (p_1 \wedge \neg \mathtt{emp}) * p_2) \wedge q') \Rightarrow$$
$$(((q \wedge \mathtt{size} = I) * (p_1 \wedge \mathtt{size} = 1)) * ((q \wedge \mathtt{size} = J_2) \mathrel{-\!\!\!\!\!\!\rightarrow} q')))$$

- Formulae for instructions of the form I: if $\mathtt{c}_2 = 0$ then goto J_1 else ($\mathtt{c}_2 := \mathtt{c}_2 - 1$; goto J_2), are defined similarly.

Here is the crucial property about the formula `closure`.

Lemma 2.3.2. Let $\mathfrak{V}$ be a valuation such that $\emptyset \models_{\mathfrak{V}} ((\mathtt{emp} \wedge p_1 \wedge p_2 \wedge \neg q \wedge \neg q') \wedge \mathtt{closure})$ and $X \approx_{\mathfrak{V}} (I, c_1, c_2)$. If $(I, c_1, c_2) \rightarrow (I', c_1', c_2')$ in $\mathtt{M}$, then there is a finite subset X' of $\mathbb{N}$ such that $X' \approx_{\mathfrak{V}} (I', c_1', c_2')$.

The proof of Lemma 2.3.2 is left as Exercise 2.9. Consequently, if $\emptyset \models_{\mathfrak{V}} ((\mathtt{emp} \wedge p_1 \wedge p_2 \wedge \neg q \wedge \neg q') \wedge \mathtt{closure})$, then all the reachable configurations from $(1, 0, 0)$ have an encoding by a set in the separation model.

The correctness proof works as follows. Suppose that the machine $\mathtt{M}$ halts. This means that for any valuation $\mathfrak{V}$, if $\emptyset \models_{\mathfrak{V}} (\mathtt{emp} \wedge p_1 \wedge p_2 \wedge \neg q \wedge \neg q') \wedge \mathtt{closure}$, then by Lemma 2.3.2, there is some $X \in \mathcal{P}_{\text{fin}}(\mathbb{N})$ such that $X \approx_{\mathfrak{V}} (\alpha, c_1, c_2)$ for some $c_1, c_2 \in \mathbb{N}$, i.e. there is some $X \in \mathcal{P}_{\text{fin}}(\mathbb{N})$ such that $X \models_{\mathfrak{V}} (p_1 * p_2 * (\mathtt{size} = \alpha \wedge q)) \wedge q'$, which is equivalent to $\emptyset \models_{\mathfrak{V}} (\top \mathrel{-\!\!\!\!\!\!\rightarrow} ((p_1 * p_2 * (\mathtt{size} = \alpha \wedge q))) \wedge q')$. Now suppose that the machine $\mathtt{M}$ does not halt, this means that there is no configuration of the form (α, c_1, c_2) reachable from the initial configuration $(1, 0, 0)$. Let us define the following valuation $\mathfrak{V}_0$:

- $\mathfrak{V}_0(q) \stackrel{\text{def}}{=} \mathcal{P}_{\text{fin}}([1, \alpha - 1]) \setminus \{\emptyset\}$.
- $\mathfrak{V}_0(p_1) \stackrel{\text{def}}{=} \mathcal{P}_{\text{fin}}(\{\alpha + 2k + 1 : k \in \mathbb{N}\})$, $\mathfrak{V}_0(p_2) \stackrel{\text{def}}{=} \mathcal{P}_{\text{fin}}(\{\alpha + 2k : k \in \mathbb{N}\})$. So, $X \in \mathfrak{V}_0(p_1)$ and $X' \in \mathfrak{V}_0(p_2)$ imply that $X \cap X' = \emptyset$.
- $\mathfrak{V}_0(q')$ is equal to the set below:

$$\{X \in \mathcal{P}_{\text{fin}}(\mathbb{N}) : (I, c_1, c_2) \text{ reachable from } (1, 0, 0),\ X = X_0 \uplus X_1 \uplus X_2,$$
$$\text{card}(X_0) = I, X_0 \in \mathcal{P}_{\text{fin}}([1, \alpha - 1]), \text{card}(X_1) = c_1, X_1 \in \mathcal{P}_{\text{fin}}(\{\alpha + 2k + 1 : k \in \mathbb{N}\}),$$
$$\text{card}(X_2) = c_2, X_2 \in \mathcal{P}_{\text{fin}}(\{\alpha + 2k : k \in \mathbb{N}\})\}.$$

One can check that

1. for every configuration (I, c_1, c_2), we have (I, c_1, c_2) is reachable from $(1, 0, 0)$ iff there is X such that $X \approx_{\mathfrak{B}_0} (I, c_1, c_2)$,
2. $\emptyset \models_{\mathfrak{B}_0} (\mathtt{emp} \wedge p_1 \wedge p_2 \wedge \neg q \wedge \neg q') \wedge \mathtt{closure}$,
3. there is no $X \in \mathcal{P}_{\text{fin}}(\mathbb{N})$ such that $X \models_{\mathfrak{B}_0} (p_1 * p_2 * (\mathtt{size} = \alpha \wedge q)) \wedge q'$.

Consequently, $\varphi_{\mathtt{M}}$ is not valid in $\mathrm{SL}(\mathcal{P}_{\text{fin}}(\mathbb{N}), \uplus, \{\emptyset\})$. This concludes the proof of Theorem 2.3.1. It is worth noting that this undecidability proof uses only semantical arguments.

2.4 Exercises

Exercise 2.1. Prove Lemma 2.1.4.

Exercise 2.2. Complete the proof of 2.1.5 with the case $\varphi_j = \exists\, p_j\, \varphi_{j+1}$ in the induction step.

Exercise 2.3. Prove Corollary 2.1.6.

Exercise 2.4. [COY01, Section 5] Let $\mathfrak{L}$ be the fragment of 1SL0 defined by the grammar below:

$$\varphi, \psi ::= \mathtt{alloc(x)} \mid \neg\mathtt{alloc(x)} \mid \mathtt{emp} \mid \neg\mathtt{emp} \mid \mathtt{x} \hookrightarrow \mathtt{y} \mid \neg(\mathtt{x} \hookrightarrow \mathtt{y}) \mid \mathtt{x} \mapsto \mathtt{y} \mid$$
$$\neg(\mathtt{x} \mapsto \mathtt{y}) \mid \mathtt{x} = \mathtt{y} \mid \neg(\mathtt{x} = \mathtt{y}) \mid \top \mid \bot \mid \varphi \vee \psi \mid \varphi \wedge \psi \mid \varphi * \psi$$

where $\mathtt{x}, \mathtt{y} \in \mathtt{PVAR}$.

a) Explain why the satisfiability problem for $\mathfrak{L}$ is NP-hard.

b) Given a propositional formula φ built over atomic propositions in $\{p_1, \ldots, p_n\}$ in which negation occurs only in front of atomic propositions, we write $tr(\varphi)$ to denote the formula in $\mathfrak{L}$ obtained from φ by substituting every occurrence of p_i by the atomic formula $\mathtt{alloc}(\mathtt{x}_i)$. We define the memory state $(\mathfrak{s}, \mathfrak{h})$ such that for all $i \in [1, n]$, we have $\mathfrak{s}(\mathtt{x}_i) \stackrel{\text{def}}{=} i$ and $\mathfrak{h}(i) \stackrel{\text{def}}{=} i$. Show that $(\mathfrak{s}, \mathfrak{h}) \models tr(\varphi) * \top$ iff φ is satisfiable. Conclude that the model-checking problem for $\mathfrak{L}$ is NP-hard.

c) By using Corollary 2.1.8, show that if φ in $\mathfrak{L}$ is satisfiable, then φ holds true on a memory state $(\mathfrak{s}, \mathfrak{h})$ with $\mathrm{ran}(\mathfrak{s}) \cup \mathrm{dom}(\mathfrak{h}) \cup \mathrm{ran}(\mathfrak{h}) \subseteq [0, p(|\varphi|)]$ for some polynomial $p(\cdot)$.

d) Given a memory state $(\mathfrak{s}, \mathfrak{h})$ and a formula φ in $\mathfrak{L}$, define a witness of polynomial size so that checking whether the witness guarantees that $(\mathfrak{s}, \mathfrak{h}) \models \varphi$ can be done in polynomial time. Conclude that the model-checking problem for $\mathfrak{L}$ is NP-complete.

e) Explain why the satisfiability problem for $\mathfrak{L}$ is NP-complete.

Exercise 2.5. By using Theorem 2.1.7, show that there is no formula in 1SL0 equivalent to the formula $\exists\, \mathtt{u}\, (\mathtt{x}_1 \hookrightarrow \mathtt{u}) \wedge (\mathtt{u} \hookrightarrow \mathtt{x}_2)$.

Exercise 2.6. Prove that for all memory states $(\mathfrak{s}, \mathfrak{h})$, we have $(\mathfrak{s}, \mathfrak{h}) \models (\neg \mathtt{emp} * (\mathtt{x}_1 \hookrightarrow \mathtt{x}_2 \mathrel{-\!\!*} \bot))$ iff $(\mathfrak{s}, \mathfrak{h}) \models \mathtt{size} \geq 2 \wedge \mathtt{alloc}(\mathtt{x}_1)$.

Exercise 2.7. Show that the formula φ is valid in $\mathrm{SL}(\mathcal{P}_{\mathrm{fin}}(\mathbb{N}), \uplus, \{\emptyset\})$ iff φ is valid in $\mathrm{SL}(\mathfrak{H}\mathfrak{S}_k, \uplus, \mathfrak{U}_k)$ –$k \geq 1$.

Exercise 2.8. Check that $(\mathcal{P}_{\mathrm{fin}}(\mathbb{N}), \uplus, \{\emptyset\})$ is a separation model.

Exercise 2.9. Prove Lemma 2.3.2.

Chapter 3

EXPRESSIVENESS OF SEPARATION LOGICS

Contents

In this chapter, we take care of the expressiveness of first-order separation logics; for the sake of simplicity, we consider fragments without program variables and therefore no need to consider stores in memory states (the models are restricted to heaps). It is worth recalling a few well-known results about expressive power of modal or temporal logics. For instance, linear-time temporal logic LTL

is known to be as expressive as first-order logic by Kamp's Theorem [Kam68] (see also [HR05, Rab14]). More references about bibliographical references for expressiveness of non-classical logics can be found in Section 3.6.

Section 3.1 shows how data words can be encoded as heaps by using formulae in 1SL2. In Section 3.2, we explain how to encode arithmetical constraints comparing the numbers of predecessors of two locations within 1SL2. Details are provided when the two locations belong to a specific class of heaps. By using the formulae built so far, in Section 3.3 we show that the satisfiability problem for 1SL2 is undecidable by reduction from the halting problem for Minsky machines. In Section 3.4, we explain how formulae in 1DSOL (without program variables) can be translated into 1SL2($-\!\!*$) (without program variables) by respecting faithfully the semantics. A reduction with program variables or with $k > 1$ record fields is possible but it is not presented in that section.

Highlights of the chapter

1. Presentation of an encoding of data words as heaps that can be specified in 1SL2 (Section 3.1) [DD15b].
2. Undecidability proof of 1SL2 by reduction from the halting problem for Minsky machines (Theorem 3.3.7) [DD15b].
3. Proof that 1SL2($-\!\!*$) is as expressive as weak second-order logic (Theorem 3.4.14). We use first principles from [BDL12] and the encodings from [DD14].

3.1 Encoding Data Words in 1SL2

In this section, we present a simple encoding of data words with multiple attributes into heaps that will be useful in the rest of the chapter. Finite data words [Bou02] are ubiquitous structures that include timed words [AD94], runs of Minsky machines, and runs of concurrent programs with an unbounded number of processes. These are finite words in which every position carries a label from a finite alphabet and a finite tuple of data values from some infinite alphabet. A wealth of specification formalisms for data words (and slight variants) has been introduced stemming from automata (see e.g. [KF94, NSV04, BL10, Fig10]) to adequate logical languages such as first-order logic [BDM$^+$11, Dav09, SZ12] and temporal logics [Fig10, DHLT14].

A **data word** of dimension β is a finite non-empty sequence in $([1,\alpha] \times \mathbb{N}^\beta)^+$ for some $\alpha \geq 1$ and $\beta \geq 0$. The set $[1,\alpha]$ is understood as a finite alphabet of cardinal α whereas $\mathbb{N}$ is the infinite data domain. Data words of dimension zero are simply finite words over a finite alphabet whereas data words of dimension one correspond to data words in the sense introduced in [Bou02]. Finite runs of Minsky machines (with two counters) can be viewed as data words of dimension two over the alphabet $[1,\alpha]$ assuming that the Minsky machine has α distinct instructions (see Section 2.3.2). In full generality, the set $\mathbb{N}$ could be replaced by any infinite data domain $\mathfrak{D}$ (for instance by $\mathbb{R}$ to define timed words); however, we do not need to be so general, and in this chapter, we focus on the infinite domain $\mathbb{N}$.

Let $\mathfrak{dw} = (a^1, \mathfrak{v}_1^1, \ldots, \mathfrak{v}_\beta^1) \cdots (a^L, \mathfrak{v}_1^L, \ldots, \mathfrak{v}_\beta^L)$ be a data word in $([1,\alpha] \times \mathbb{N}^\beta)^+$, i.e. $\mathfrak{dw}$ is of dimension β and its underlying alphabet has cardinal $\alpha \geq 1$. The data word $\mathfrak{dw}$ shall be encoded by the heap $\mathfrak{h}_{\mathfrak{dw}}$ containing a path of the form below:

$$\mathfrak{l}_0^1 \to \mathfrak{l}_1^1 \to \cdots \to \mathfrak{l}_\beta^1 \to \cdots \to \mathfrak{l}_0^L \to \mathfrak{l}_1^L \to \cdots \to \mathfrak{l}_\beta^L$$

where

- for every $i \in [1,L]$, $\mathfrak{l}_0^i$ has $a^i + 2$ predecessors,
- for all $i \in [1,L]$ and all $j \in [1,\beta]$, $\mathfrak{l}_j^i$ has $\mathfrak{v}_j^i + \alpha + 3$ predecessors,
- every location in the heap domain is either on that path or points to a location on that path.

Such a path from $\mathfrak{l}_0^1$ to $\mathfrak{l}_\beta^L$ is called the **main path**, and $\mathfrak{h}_{\mathfrak{dw}}^{(\beta+1)L-1}(\mathfrak{l}_0^1) = \mathfrak{l}_\beta^L$. Other simple encodings are possible (for instance without shifting the values from the finite alphabet or from the infinite domain) but the current one is well-suited for all the developments made in this chapter. In particular, the encoding allows us to know easily whether a location encodes a letter from the finite alphabet or an element from the infinite domain. Note also that $\mathfrak{h}_{\mathfrak{dw}}$ is not uniquely specified, and we understand it modulo isomorphism, see Exercise 1.1.

Figure 3.1 presents the encoding of the data word $\mathfrak{dw}_0 = (2,1)(1,2)(2,2)$ of dimension 1 with $\alpha = 2$ with its representation of the heap $\mathfrak{h}_{\mathfrak{dw}}$ in which the predecessors of the locations on the main path are provided schematically.

The heap $\mathfrak{h}_{\mathfrak{dw}}$ looks like a fishbone. Let us make this precise. A heap $\mathfrak{h}$ is a **fishbone** $\overset{\text{def}}{\Leftrightarrow}$

(fb1) $\mathrm{dom}(\mathfrak{h}) \neq \emptyset$,

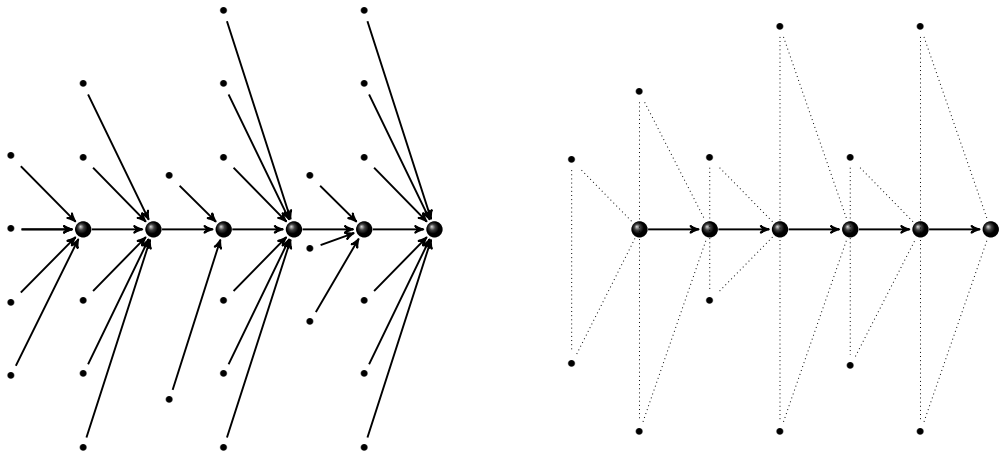


Figure 3.1: The heap for data word $\mathfrak{dw}_0 = (2,1)(1,2)(2,2)$.

(fb2) there is a location reachable from all the locations of $\mathrm{dom}(\mathfrak{h})$ that is not in $\mathrm{dom}(\mathfrak{h})$, and

(fb3) there are no distinct locations $\mathfrak{l}_1, \mathfrak{l}_2, \mathfrak{l}_3, \mathfrak{l}_4, \mathfrak{l}_5$ such that $\mathfrak{l}_1 \to \mathfrak{l}_2 \to \mathfrak{l}_3 \leftarrow \mathfrak{l}_4 \leftarrow \mathfrak{l}_5$ in the heap $\mathfrak{h}$.

When $\mathfrak{h}$ is a fishbone, it has a tree-like structure (when looking at the edges backward), equipped with a root (the unique location from (fb2)), but additionally, one can recognise the locations on the main path as those locations with at least one predecessor. The existence of such a main path is guaranteed by (fb3). The first location on the main path satisfies the formula

$$\mathtt{first}(\mathtt{u}) \stackrel{\text{def}}{=} (\sharp\mathtt{u} \geq 1) \wedge \neg(\sharp\mathtt{u}^{-1} \geq 1)$$

and the last location on the main path satisfies precisely the formula

$$\mathtt{last}(\mathtt{u}) \stackrel{\text{def}}{=} (\sharp\mathtt{u} \geq 1) \wedge \neg\mathtt{alloc}(\mathtt{u})$$

Let φ_{fb} be the formula below:

$$\overbrace{\neg\mathtt{emp}}^{\text{(fb1)}} \wedge \overbrace{(\exists\, \mathtt{u}\ \neg\mathtt{alloc}(\mathtt{u}) \wedge (\forall\, \overline{\mathtt{u}}\ \mathtt{alloc}(\overline{\mathtt{u}}) \Rightarrow \mathtt{reach}(\overline{\mathtt{u}}, \mathtt{u})))}^{\text{(fb2)}} \wedge$$

$$\overbrace{\neg(\exists\, \mathtt{u}\ (\sharp \mathtt{u}^{-2} \geq 0) * (\sharp \mathtt{u}^{-2} \geq 0))}^{\text{(fb3)}}.$$

The formulae of the form $\sharp \mathtt{u}^{-i} \sim k$ and $\mathtt{reach}(\overline{\mathtt{u}}, \mathtt{u})$ can be found in Section 1.2.2.

Lemma 3.1.1. Let $\mathfrak{h}$ be a heap. We have $\mathfrak{h} \models \varphi_{\text{fb}}$ iff $\mathfrak{h}$ is a fishbone.

The proof for Lemma 3.1.1 is by an easy verification. Now, let us refine the notion of a fishbone heap so that it takes into account constraints on numbers of predecessors. An (α, β)**-fishbone** is a fishbone heap such that

(C1) the first location on the main path has a number of predecessors in $[3, \alpha+2]$,

(C2) on the main path, a location with a number of predecessors in $[3, \alpha + 2]$, is followed by β locations with at least $\alpha + 3$ predecessors, and

(C3) the number of locations on the main path is a multiple of $\beta + 1$.

It is easy to check that the formulae φ_{C1}, φ_{C2} and φ_{C3} in 1SL2$(*)$ defined below are able to express the conditions (C1), (C2) and (C3), respectively. This assumes that the heap is already known to be a fishbone, which is equivalent to the satisfaction of φ_{fb} (by Lemma 3.1.1).

$$\varphi_{(C1)} \stackrel{\text{def}}{=} \exists\, \mathtt{u}\ \mathtt{first}(\mathtt{u}) \wedge (3 \leq \sharp \mathtt{u} \leq \alpha + 2)$$

$$\varphi_{(C2)} \stackrel{\text{def}}{=} \forall\, \mathtt{u}\ (3 \leq \sharp \mathtt{u} \leq \alpha + 2) \Rightarrow \bigwedge_{i \in [1,\beta]} \sharp \mathtt{u}^{+i} \geq \alpha + 3$$

$$\varphi_{(C3)} \stackrel{\text{def}}{=} \forall\, \mathtt{u}\ (3 \leq \sharp \mathtt{u} \leq \alpha + 2) \Rightarrow ((\neg \sharp \mathtt{u}^{+(\beta+1)} \geq 0) \vee (3 \leq \sharp \mathtt{u}^{+(\beta+1)} \leq \alpha + 2)).$$

We write $\mathbf{dw}(\alpha, \beta)$ to denote the formula $\varphi_{\text{fb}} \wedge \varphi_{(C1)} \wedge \varphi_{(C2)} \wedge \varphi_{(C3)}$. It specifies the shape of the encoding of data words in $([1, \alpha] \times \mathbb{N}^{\beta})^+$ as stated below.

Lemma 3.1.2. Let $\mathfrak{h}$ be a heap. We have $\mathfrak{h} \models \mathbf{dw}(\alpha, \beta)$ iff $\mathfrak{h}$ is an (α, β)-fishbone.

Again, the proof is by an easy verification by using Lemma 3.1.1 and the correspondence between the condition (Ci) and the formula $\varphi_{(Ci)}$.

Given a data word $\mathfrak{dw} = (a^1, \mathfrak{v}_1^1, \ldots, \mathfrak{v}_\beta^1) \cdots (a^L, \mathfrak{v}_1^L, \ldots, \mathfrak{v}_\beta^L)$, we can associate a (α, β)-fishbone $\mathfrak{h}_{\mathfrak{dw}}$ with $(1 + \beta) \times L$ locations on the main path, say

$$\mathfrak{l}_0^1 \to \mathfrak{l}_1^1 \to \cdots \to \mathfrak{l}_\beta^1 \to \cdots \to \mathfrak{l}_0^L \to \mathfrak{l}_1^L \to \cdots \to \mathfrak{l}_\beta^L$$

such that

- for every $i \in [1, L]$, $\widetilde{\sharp \mathfrak{l}_0^i} = a^i + 2$,
- for all $i \in [1, L]$ and all $j \in [1, \beta]$, $\widetilde{\sharp \mathfrak{l}_j^i} = \mathfrak{v}_j^i + \alpha + 3$.

We recall that $\widetilde{\sharp \mathfrak{l}}$ denotes the number of predecessors of the location $\mathfrak{l}$ (given an implicit current heap), see also Section 1.2.1.

The heap $\mathfrak{h}_{\mathfrak{dw}}$ is unique modulo isomorphism. This natural encoding generalises the encoding of finite words by heaps in [BDL12, Section 3] (see also a related encoding in [BBL09]) while providing a much more concise representation. Note also that the encoding by itself is of no use since it is essential to be able to operate on it with the logical language at hand.

Conversely, given a (α, β)-fishbone $\mathfrak{h}$ with $(1 + \beta) \times L$ locations on the main path, say

$$\mathfrak{l}_0^1 \to \mathfrak{l}_1^1 \to \cdots \to \mathfrak{l}_\beta^1 \to \cdots \to \mathfrak{l}_0^L \to \mathfrak{l}_1^L \to \cdots \to \mathfrak{l}_\beta^L$$

we associate a (unique) data word $\mathfrak{dw}_\mathfrak{h} = (a^1, \mathfrak{v}_1^1, \ldots, \mathfrak{v}_\beta^1) \cdots (a^L, \mathfrak{v}_1^L, \ldots, \mathfrak{v}_\beta^L)$ such that

- for every $i \in [1, L]$, $a^i \stackrel{\text{def}}{=} \widetilde{\sharp \mathfrak{l}_0^i} - 2$ and,
- for all $i \in [1, L]$ and all $j \in [1, \beta]$, $\mathfrak{v}_j^i \stackrel{\text{def}}{=} \widetilde{\sharp \mathfrak{l}_j^i} - \alpha - 3$.

Lemma 3.1.3. There is a one-to-one map between data words in $([1, \alpha] \times \mathbb{N}^\beta)^+$ and (α, β)-fishbone heaps (modulo isomorphism).

The proof is then by an easy verification. So, we have seen that finite words can be encoded in 1SL2(∗), which allows us to establish that 1SL2(∗) is NEXPTIME-hard since first-order logic restricted to two quantified variables on finite words (written $\mathrm{FO2}_{\alpha,0}(<, +1, =)$ herein) is NEXPTIME-complete [EVW97]. Indeed, consider a sentence φ in that fragment of first-order logic. Let us define $tr(\varphi)$ such that φ is satisfiable iff $\mathbf{dw}(\alpha, 0) \wedge tr(\varphi)$ is satisfiable in 1SL2(∗). The logarithmic-space translation tr is homomorphic for Boolean connectives and is further defined as follows ($i, j \in \{1, 2\}$).

$$\begin{array}{lcl}
tr(\mathtt{u}_i = \mathtt{u}_j) & \stackrel{\text{def}}{=} & \mathtt{u}_i = \mathtt{u}_j \\
tr(a(\mathtt{u}_i)) & \stackrel{\text{def}}{=} & (\sharp \mathtt{u}_i = a + 2) \\
tr(\mathtt{u}_i = 1 + (\mathtt{u}_j)) & \stackrel{\text{def}}{=} & \mathtt{u}_j \hookrightarrow \mathtt{u}_i \\
tr(\mathtt{u}_i < \mathtt{u}_j) & \stackrel{\text{def}}{=} & \mathtt{reach}(\mathtt{u}_i, \mathtt{u}_j) \wedge \mathtt{u}_i \neq \mathtt{u}_j \\
tr(\exists\, \mathtt{u}_i\, \varphi) & \stackrel{\text{def}}{=} & \exists\, \mathtt{u}_i\, (\sharp \mathtt{u}_i \geq 1) \wedge tr(\varphi).
\end{array}$$

Note that $\mathrm{FO2}_{\alpha,0}(<,+1,=)$ and 1SL2$(*)$ share the same number of quantified variables and $\mathtt{reach}(\mathtt{u}_i,\mathtt{u}_j)$ can be expressed in 1SL2$(*)$ (see Section 1.2.2). We do not provide the correctness proof herein since we can do much better than NEXPTIME-hardness by making a strong connection with Moszkowski's Interval Temporal Logic ITL (with the locality condition), see Section 4.2.

3.2 Encoding Arithmetical Constraints in 1SL2

Below, we show how to express in 1SL2 the constraints $\sharp\mathtt{u} = \sharp\overline{\mathtt{u}}$, $\sharp\mathtt{u} = \sharp\overline{\mathtt{u}} + 1$ and $\sharp\overline{\mathtt{u}} = \sharp\mathtt{u} + 1$, when $\mathtt{u}$ and $\overline{\mathtt{u}}$ are interpreted by locations on the main path of $(\alpha, 2)$-fishbone heaps.

We shall use the fact that $N \leq N'$ $(N, N' \in \mathbb{N})$ iff for every $n \geq 0$, we have $N' \leq n$ implies $N \leq n$. Quantification over the set of natural numbers will be simulated by quantification over disjoint heaps in which n is related to the cardinal of their heap domains. Such quantification is performed thanks to the magic wand operator.

A **fork** in $\mathfrak{h}$ is a sequence of distinct locations $\mathfrak{l}, \mathfrak{l}_0, \mathfrak{l}_1, \mathfrak{l}_2$ such that $\mathfrak{h}(\mathfrak{l}_0) = \mathfrak{l}$, $\widetilde{\sharp\mathfrak{l}_0} = 2$, $\mathfrak{h}(\mathfrak{l}_1) = \mathfrak{l}_0$, $\mathfrak{h}(\mathfrak{l}_2) = \mathfrak{l}_0$ and $\widetilde{\sharp\mathfrak{l}_1} = \widetilde{\sharp\mathfrak{l}_2} = 0$. The **endpoint** of the fork is $\mathfrak{l}$. Similarly, a **knife** in $\mathfrak{h}$ is a sequence of distinct locations $\mathfrak{l}, \mathfrak{l}_0, \mathfrak{l}_1$ such that $\mathfrak{h}(\mathfrak{l}_0) = \mathfrak{l}$, $\widetilde{\sharp\mathfrak{l}_0} = 1$, $\mathfrak{h}(\mathfrak{l}_1) = \mathfrak{l}_0$ and $\widetilde{\sharp\mathfrak{l}_1} = 0$. The **endpoint** of the knife is $\mathfrak{l}$. By way of example, the heap of Figure 3.2 contains three knives, two forks and four endpoints (identified by '$\star$').

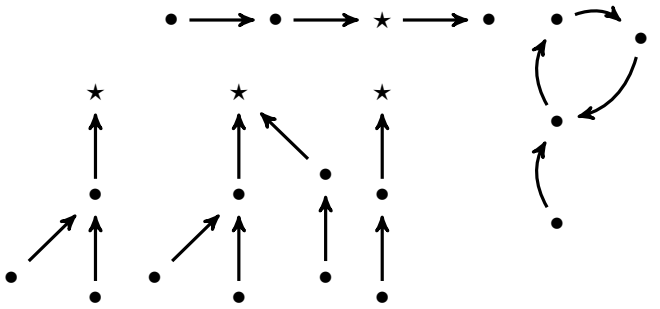


Figure 3.2: A heap with three knives, two forks and four endpoints.

Lemma 3.2.1. Let $\mathfrak{h}$ be a (α, β)-fishbone heap with $\alpha \geq 1$ and $\beta \geq 0$. Then, $\mathfrak{h}$ has no knife and no fork.

Indeed, in such heaps, any allocated location has no predecessor or at least three predecessors.

A heap $\mathfrak{h}$ is a **collection of knives** $\overset{\text{def}}{\Leftrightarrow}$ there is no location in $\mathrm{dom}(\mathfrak{h})$ that does not belong to a knife and no distinct knives share the same endpoint. A heap $\mathfrak{h}$ is **segmented** whenever $\mathrm{dom}(\mathfrak{h}) \cap \mathrm{ran}(\mathfrak{h}) = \emptyset$ and no location has strictly more than one predecessor.

Lemma 3.2.2. Let $\mathfrak{h}$ be a (α, β)-fishbone heap with $\alpha \geq 1$, $\beta \geq 0$ and $\mathfrak{h}'$ be a segmented heap disjoint from $\mathfrak{h}$. Then, $\mathfrak{h} \uplus \mathfrak{h}'$ has no fork.

Being segmented can be naturally expressed in 1SL2:

$$\mathtt{seg} \overset{\text{def}}{=} \forall\, \mathtt{u}\, \overline{\mathtt{u}}\, (\mathtt{u} \hookrightarrow \overline{\mathtt{u}} \Rightarrow ((\sharp\overline{\mathtt{u}} = 1) \wedge (\sharp\mathtt{u} = 0) \wedge \neg\mathtt{alloc}(\overline{\mathtt{u}}))).$$

The statement below is counterpart to [BDL12, Lemma 5.2] with simplified properties and with simpler formulae but using only two quantified variables.

Lemma 3.2.3. There are formulae $\mathtt{forky}(\mathtt{u})$, $\mathtt{KS}$ and $\mathtt{KS1F}$ in 1SL2 such that for every heap $\mathfrak{h}$,

(I) $\mathfrak{h} \models_{\mathfrak{f}} \mathtt{forky}(\mathtt{u})$ iff all the predecessors of $\mathfrak{f}(\mathtt{u})$ are endpoints of forks,

(II) $\mathfrak{h} \models \mathtt{KS}$ iff $\mathfrak{h}$ is a collection of knives,

(III) $\mathfrak{h} \models \mathtt{KS1F}$ iff there are $\mathfrak{h}_1, \mathfrak{h}_2$ such that $\mathfrak{h} = \mathfrak{h}_1 \uplus \mathfrak{h}_2$, $\mathfrak{h}_1$ is a collection of knives and $\mathfrak{h}_2$ is made of a unique fork such that its unique endpoint is not in the range of $\mathfrak{h}_1$.

Proof. $\mathtt{forky}(\mathtt{u})$ is equal to:

$$\forall\, \overline{\mathtt{u}}\, (\overline{\mathtt{u}} \hookrightarrow \mathtt{u}) \Rightarrow (\exists\, \mathtt{u}\, (\mathtt{u} \hookrightarrow \overline{\mathtt{u}}) \wedge (\sharp\mathtt{u} = 2) \wedge \neg(\sharp\mathtt{u}^{-1} \geq 1)).$$

A knife is made of two consecutive memory cells that can be respectively called part 1 and part 2 as shown in $\mathfrak{l} \xrightarrow{\text{part 1}} \mathfrak{l}' \xrightarrow{\text{part 2}} \mathfrak{l}''$.

$$\mathtt{KS} \overset{\text{def}}{=} \forall\, \mathtt{u}\, \mathtt{alloc}(\mathtt{u}) \Rightarrow (\varphi_{\text{part1}}(\mathtt{u}) \vee \varphi_{\text{part2}}(\mathtt{u}))$$

where

$$\varphi_{\text{part1}}(\mathtt{u}) \overset{\text{def}}{=} (\sharp\mathtt{u} = 0) \wedge (\sharp\mathtt{u}^{+1} = 1) \wedge (\sharp\mathtt{u}^{+2} = 1) \wedge \neg(\sharp\mathtt{u}^{+3} \geq 0)$$

$$\varphi_{\text{part2}}(\mathtt{u}) \overset{\text{def}}{=} (\sharp\mathtt{u} = 1) \wedge (\sharp\mathtt{u}^{-1} = 0) \wedge (\sharp\mathtt{u}^{+1} = 1) \wedge \neg(\sharp\mathtt{u}^{+2} \geq 0).$$

$$\mathtt{KS1F} \stackrel{\text{def}}{=}$$

$$\overbrace{[\exists\, \mathtt{u}\ (\sharp\mathtt{u} = 2) \wedge (\sharp\mathtt{u}^{+1} = 1) \wedge \neg(\sharp\mathtt{u}^{+2} \geq 0) \wedge \neg(\sharp\mathtt{u}^{-1} \geq 1) \wedge \neg(\exists\, \overline{\mathtt{u}}\ (\mathtt{u} \neq \overline{\mathtt{u}}) \wedge (\sharp\overline{\mathtt{u}} = 2))]}^{\text{unique fork}} \wedge$$

$$[\forall\, \mathtt{u}\ \mathtt{alloc}(\mathtt{u}) \Rightarrow$$

$$(\underbrace{\varphi_{\text{part1}}(\mathtt{u}) \vee \varphi_{\text{part2}}(\mathtt{u})}_{\text{part with knifes}} \vee \underbrace{((\sharp\mathtt{u} = 0) \wedge (\sharp\mathtt{u}^{+1} = 2)) \vee (\sharp\mathtt{u} = 2))}_{\text{part with one fork}}]$$

In our proof, we use the idea of augmenting the heap with a segmented heap, then augmenting it further with knives to form forks whose endpoints are predecessors of $\mathtt{u}$; this is borrowed from [BDL12]. As it is, this would not be sufficient to express arithmetical constraints on fishbone heaps since only two quantified variables are allowed. This restriction is not considered in [BDL12]—the formulae there use strictly more than two quantified variables. This is why we had to provide specific developments that are well-tailored to fishbone heaps while taking into account our limited amount of syntactic resources (this can be generalised to any heap in [DD14]). Simplifications have also been made in order to focus on undecidability rather than on questions of expressive power. Note also that there are versions of separation logics in which arithmetical constraints are built-in, for instance about the length of lists, see e.g. [BIP10].

Lemma 3.2.4. Let $\mathfrak{h}$ be a heap with $\mathfrak{h} = \mathfrak{h}_1 \uplus \mathfrak{h}_2$ and $\mathfrak{f}$ be an assignment such that $\mathfrak{h}_1$ is a $(\alpha, 2)$-fishbone $\mathfrak{f}(\mathtt{u})$ is on the main path of $\mathfrak{h}_1$, $\mathfrak{h}_2 \models_{\mathfrak{f}} \mathtt{seg} \wedge \sharp\mathtt{u} = 0$, $n = \mathrm{card}(\mathrm{ran}(\mathfrak{h}_2) \setminus \mathrm{dom}(\mathfrak{h}_1))$ and m is the number of predecessors of $\mathfrak{f}(\mathtt{u})$ in $\mathfrak{h}_1$. We have the following properties:

(I) $\mathfrak{h} \models_{\mathfrak{f}} \neg(\mathtt{KS} \mathrel{-\!\!*} \neg\mathtt{forky}(\mathtt{u}))$ iff $n \geq m$.

(II) $\mathfrak{h} \models_{\mathfrak{f}} \neg(\mathtt{KS1F} \mathrel{-\!\!*} \neg\mathtt{forky}(\mathtt{u}))$ iff $n \geq m - 1$.

In Figure 3.3, we present three heaps obtained by combining a segmented heap $\mathfrak{h}_2$ with collections of knives (corresponding to $\mathfrak{h}_3$ in the proof of Lemma 3.2.4). Edges labelled by '1' are part of a fishbone heap $\mathfrak{h}_1$ (partially represented) whereas edges labelled by '2' are part of a segmented heap $\mathfrak{h}_2$ so that no edge points to $\mathfrak{f}(\mathtt{u})$ or to $\mathfrak{f}(\overline{\mathtt{u}})$. The heap on the left (corresponding to $\mathfrak{h}_1 \uplus \mathfrak{h}_2$ in Lemma 3.2.4) is obtained by adding a segmented heap $\mathfrak{h}_2$ whereas the heap in the middle (say $\mathfrak{h}_1 \uplus \mathfrak{h}_2 \uplus \mathfrak{h}_3$) is obtained then by adding a collection of knives $\mathfrak{h}_3$ so that every predecessor of $\mathfrak{f}(\mathtt{u})$ is the endpoint of a fork. Note that not all edges of the segmented

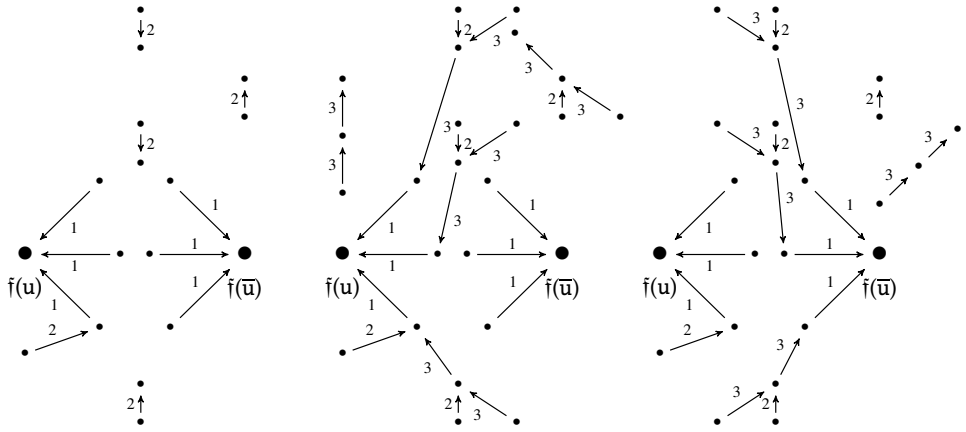


Figure 3.3: A segmented heap and collections of knives.

heap are used to build forks. Similarly, the heap on the right (say $\mathfrak{h}_1 \uplus \mathfrak{h}_2 \uplus \mathfrak{h}'_3$) is obtained then by adding a collection of knives $\mathfrak{h}'_3$ to the heap $\mathfrak{h}_1 \uplus \mathfrak{h}_2$ on the very left so that every predecessor of $\mathsf{f}(\overline{\mathsf{u}})$ is the endpoint of a fork.

Proof. Let us provide the proof for (I). (The proof for (II), being analogous, is omitted.) So, let $\mathfrak{h}$ be a heap with $\mathfrak{h} = \mathfrak{h}_1 \uplus \mathfrak{h}_2$ such that $\mathfrak{h}_1$ is an $(\alpha, 2)$-fishbone heap, and $\mathfrak{h}_2 \models_{\mathsf{f}} \mathtt{seg} \wedge \sharp\mathsf{u} = 0$. Moreover, $\mathsf{f}(\mathsf{u})$ is on the main path, which entails that $\mathfrak{h}(\mathsf{f}(\mathsf{u})) \neq \mathsf{f}(\mathsf{u})$ (if $\mathfrak{h}(\mathsf{f}(\mathsf{u}))$ is defined at all) and $\mathsf{f}(\mathsf{u})$ has at least one predecessor.

One can make the following (obvious) observations.

(O1) The heap $\mathfrak{h}_1$ has no knives and, $\mathfrak{h}_1$ and $\mathfrak{h}_1 \uplus \mathfrak{h}_2$ have no forks. (see Lemma 3.2.1 and Lemma 3.2.2).

(O2) $\mathfrak{h}_1 \uplus \mathfrak{h}_2$ may not be a $(\alpha, 2)$-fishbone heap but this is fine since we only need to focus on the number of predecessors of $\mathsf{f}(\mathsf{u})$ (i.e., on the value m). Indeed, $\mathfrak{h}_1 \uplus \mathfrak{h}_2$ may contain knives (see the left heap in Figure 3.3). A knife $\mathsf{l}_1 \to \mathsf{l}_2 \to \mathsf{l}_3$ in $\mathfrak{h}_1 \uplus \mathfrak{h}_2$ is made of $\mathsf{l}_1 \in \mathrm{dom}(\mathfrak{h}_2)$ and of $\mathsf{l}_2 \in \mathrm{dom}(\mathfrak{h}_1)$. This observation is not really used below but, hopefully, it could be helpful to better grasp how the heaps $\mathfrak{h}_1$ and $\mathfrak{h}_2$ are combined.

(O3) $\mathsf{f}(\mathsf{u})$ has the same number of predecessors in $\mathfrak{h}_1$ and in $\mathfrak{h}_1 \uplus \mathfrak{h}_2$. This is due to the fact that $\mathfrak{h}_2 \models_{\mathsf{f}} \sharp\mathsf{u} = 0$.

(O4) For every $n \geq 0$, there is a disjoint heap $\mathfrak{h}'_2$ such that $\mathfrak{h}' = \mathfrak{h}_1 \uplus \mathfrak{h}'_2$, $\mathfrak{h}'_2 \models_{\mathsf{f}} \mathtt{seg} \wedge \sharp\mathsf{u} = 0$ and $\mathrm{card}(\mathrm{ran}(\mathfrak{h}'_2) \setminus \mathrm{dom}(\mathfrak{h}_1)) = n$. See the left heap in Figure 3.3

with $\mathrm{card}(\mathrm{dom}(\mathfrak{h}_2)) = 5$ and $\mathrm{card}(\mathrm{ran}(\mathfrak{h}_2) \setminus \mathrm{dom}(\mathfrak{h}_1)) = 4$ (look at edges labelled by '2'). Once more, this observation is not used below but it will be in the proof of Theorem 3.2.5.

First, let us suppose that $\mathfrak{h} \models_{\mathfrak{f}} \neg(\mathtt{KS} \mathrel{-\!\!*} \neg\mathtt{forky}(\mathtt{u}))$, i.e., (†) there is a heap $\mathfrak{h}_3$, disjoint from $\mathfrak{h}_1 \uplus \mathfrak{h}_2$, such that $(\mathfrak{h}_1 \uplus \mathfrak{h}_2) \uplus \mathfrak{h}_3 \models_{\mathfrak{f}} \mathtt{forky}(\mathtt{u})$ and $\mathfrak{h}_3 \models_{\mathfrak{f}} \mathtt{KS}$. Let us make additional observations.

- The only forks in $\mathfrak{h}_1 \uplus \mathfrak{h}_2 \uplus \mathfrak{h}_3$ whose endpoints are predecessors of $\mathfrak{f}(\mathtt{u})$ are those obtained with $\mathfrak{l}_1 \to \mathfrak{l}_2$ such that $\mathfrak{l}_1 \in \mathrm{dom}(\mathfrak{h}_2)$ (so $\mathfrak{h}_2(\mathfrak{l}_1) = \mathfrak{l}_2$), $\mathfrak{l}_2 \notin \mathrm{dom}(\mathfrak{h}_1)$, and $\mathfrak{l}'_1 \to \mathfrak{l}_2 \to \mathfrak{l}'_3$ is a knife from $\mathfrak{h}_3$. This is due to (O1) and to the fact that all the predecessors of $\mathfrak{f}(\mathtt{u})$ in $\mathfrak{h}$ have no predecessors since $\mathfrak{f}(\mathtt{u})$ is on the main path of $\mathfrak{h}$.

- The number of forks in $\mathfrak{h}_1 \uplus \mathfrak{h}_2 \uplus \mathfrak{h}_3$ whose endpoints are predecessors of $\mathfrak{f}(\mathtt{u})$ is therefore less of equal to $\mathrm{card}(\mathrm{ran}(\mathfrak{h}_2) \setminus \mathrm{dom}(\mathfrak{h}_1))$.

- The number of predecessors of $\mathfrak{f}(\mathtt{u})$ in $\mathfrak{h}_1 \uplus \mathfrak{h}_2 \uplus \mathfrak{h}_3$ is greater or equal to the number of its predecessors in $\mathfrak{h}_1$ (by using (O3)). So, if $\mathfrak{h}_1 \uplus \mathfrak{h}_2 \uplus \mathfrak{h}_3 \models_{\mathfrak{f}} \mathtt{forky}(\mathtt{u})$, then the number of predecessors of $\mathfrak{f}(\mathtt{u})$ in $\mathfrak{h}_1$ is smaller or equal to $\mathrm{card}(\mathrm{ran}(\mathfrak{h}_2) \setminus \mathrm{dom}(\mathfrak{h}_1)) = n$, i.e. $n \geq m$.

Now, let us establish the other direction and let us suppose that $n \geq m$ and the predecessors of $\mathfrak{f}(\mathtt{u})$ are $\mathfrak{p}_1, \ldots, \mathfrak{p}_m$. Let $\mathfrak{l}^1_1, \mathfrak{l}^2_1, \ldots, \mathfrak{l}^1_n, \mathfrak{l}^2_n$ be locations such that $\{\mathfrak{l}^1_1, \ldots, \mathfrak{l}^1_n\} = \mathrm{ran}(\mathfrak{h}_2) \setminus \mathrm{dom}(\mathfrak{h}_1)$ and for every $i \in [1, n]$, we have $\mathfrak{h}_2(\mathfrak{l}^2_i) = \mathfrak{l}^1_i$. Let us build $\mathfrak{h}_3$ so that it satisfies (†), which is quite easy to realise. Let $\mathfrak{l}^{\mathrm{new}}_1, \ldots, \mathfrak{l}^{\mathrm{new}}_m$ be (new) locations that are not in $\mathrm{dom}(\mathfrak{h}_1 \uplus \mathfrak{h}_2) \cup \mathrm{ran}(\mathfrak{h}_1 \uplus \mathfrak{h}_2)$. We define $\mathfrak{h}_3$ so that it contains exactly m knives whose endpoints are exactly all the predecessors of $\mathfrak{f}(\mathtt{u})$. For every $i \in [1, m]$, we define $\mathfrak{h}_3(\mathfrak{l}^{\mathrm{new}}_i) \stackrel{\mathrm{def}}{=} \mathfrak{l}^1_i$ and $\mathfrak{h}_3(\mathfrak{l}^1_i) \stackrel{\mathrm{def}}{=} \mathfrak{p}_i$ (well, that is possible because $\mathfrak{l}^1_i \notin \mathrm{dom}(\mathfrak{h}_1 \uplus \mathfrak{h}_2)$). It is easy to check that $\mathfrak{h}_3$ satisfies (†).

Consequently, $\mathfrak{h} \models_{\mathfrak{f}} \neg(\mathtt{KS} \mathrel{-\!\!*} \neg\mathtt{forky}(\mathtt{u}))$ iff $n \geq m$. **QED**

Now, we are able to state the main proposition of this section that allows us to compare the numbers of predecessors for two locations on the main path of a fishbone heap. Let us introduce the following abbreviations:

$$\begin{array}{lcl} \chi_1(\mathtt{u}, \overline{\mathtt{u}}) & \stackrel{\mathrm{def}}{=} & \mathtt{seg} \wedge \sharp\mathtt{u} = 0 \wedge \sharp\overline{\mathtt{u}} = 0 \\ \chi_2(\mathtt{u}) & \stackrel{\mathrm{def}}{=} & \neg(\mathtt{KS} \mathrel{-\!\!*} \neg\mathtt{forky}(\mathtt{u})) \\ \chi_3(\mathtt{u}) & \stackrel{\mathrm{def}}{=} & \neg(\mathtt{KS1F} \mathrel{-\!\!*} \neg\mathtt{forky}(\mathtt{u})). \end{array}$$

Theorem 3.2.5. [DD15b] Suppose $\mathfrak{h}_1$ is a $(\alpha, 2)$-fishbone heap and, $\mathfrak{f}(\mathsf{u})$ and $\mathfrak{f}(\overline{\mathsf{u}})$ are on the main path of $\mathfrak{h}_1$. We have the following equivalences:

- $\mathfrak{h}_1 \models_{\mathfrak{f}} \chi_1(\mathsf{u}, \overline{\mathsf{u}}) \mathrel{-\!\!*} (\chi_2(\overline{\mathsf{u}}) \Rightarrow \chi_2(\mathsf{u}))$ iff $\widetilde{\sharp\mathsf{u}} \leq \widetilde{\sharp\overline{\mathsf{u}}}$.
- $\mathfrak{h}_1 \models_{\mathfrak{f}} \chi_1(\mathsf{u}, \overline{\mathsf{u}}) \mathrel{-\!\!*} (\chi_2(\overline{\mathsf{u}}) \Rightarrow \chi_3(\mathsf{u}))$ iff $\widetilde{\sharp\mathsf{u}} \leq \widetilde{\sharp\overline{\mathsf{u}}} + 1$.
- $\mathfrak{h}_1 \models_{\mathfrak{f}} \chi_1(\mathsf{u}, \overline{\mathsf{u}}) \mathrel{-\!\!*} (\chi_3(\overline{\mathsf{u}}) \Rightarrow \chi_2(\mathsf{u}))$ iff $\widetilde{\sharp\mathsf{u}} \leq \widetilde{\sharp\overline{\mathsf{u}}} - 1$.

Proof. By way of example, let us show the second property. The other cases are proved in a similar fashion. Let $\mathfrak{h}_1$ be a $(\alpha, 2)$-fishbone heap. The statements below are equivalent.

1. $\mathfrak{h}_1 \models_{\mathfrak{f}} (\chi_1(\mathsf{u}, \overline{\mathsf{u}}) \mathrel{-\!\!*} (\chi_2(\overline{\mathsf{u}}) \Rightarrow \chi_3(\mathsf{u})))$.
2. For every disjoint heap $\mathfrak{h}_2$ such that $\mathfrak{h}_2 \models_{\mathfrak{f}} \chi_1(\mathsf{u}, \overline{\mathsf{u}})$, if $\mathfrak{h}_1 \uplus \mathfrak{h}_2 \models_{\mathfrak{f}} \chi_2(\overline{\mathsf{u}})$, then $\mathfrak{h}_1 \uplus \mathfrak{h}_2 \models_{\mathfrak{f}} \models \chi_3(\mathsf{u})$. (by definition of $\models_{\mathfrak{f}}$)
3. For every $n \geq 0$, there is a disjoint heap $\mathfrak{h}_2$ with $\mathrm{card}(\mathrm{ran}(\mathfrak{h}_2) \setminus \mathrm{dom}(\mathfrak{h}_1)) = n$ such that $\mathfrak{h}_2 \models_{\mathfrak{f}} \chi_1(\mathsf{u}, \overline{\mathsf{u}})$ and if $\mathfrak{h}_1 \uplus \mathfrak{h}_2 \models_{\mathfrak{f}} \chi_2(\overline{\mathsf{u}})$, then $\mathfrak{h}_1 \uplus \mathfrak{h}_2 \models_{\mathfrak{f}} \chi_3(\mathsf{u})$ (see (O4) in the proof of Lemma 3.2.4). This is possible by using the fact that one can add a segmented heap so that the resulting heap has n isolated memory cells. Indeed, given the heap $\mathfrak{h}_1$, let us build a disjoint heap $\mathfrak{h}_2$ such that $\mathfrak{h}_2 \models_{\mathfrak{f}} \chi_1(\mathsf{u}, \overline{\mathsf{u}})$ and $\mathrm{dom}(\mathfrak{h}_2) = n$ for any fixed $n \geq 0$. Since $X = \mathrm{dom}(\mathfrak{h}_1) \cup \mathrm{ran}(\mathfrak{h}_2) \cup \{\mathfrak{f}(\mathsf{u}), \mathfrak{f}(\overline{\mathsf{u}})\}$ is a finite subset of $\mathbb{N}$, there are $2n$ distinct locations $\mathfrak{l}^1_1, \mathfrak{l}^2_1, \ldots, \mathfrak{l}^1_n, \mathfrak{l}^2_n$ in $\mathbb{N} \setminus X$. We simply need to define $\mathfrak{h}_2$ such that $\mathrm{dom}(\mathfrak{h}_2) \stackrel{\text{def}}{=} \{\mathfrak{l}^1_1, \ldots, \mathfrak{l}^1_n\}$, $\mathrm{ran}(\mathfrak{h}_2) \stackrel{\text{def}}{=} \{\mathfrak{l}^2_1, \ldots, \mathfrak{l}^2_n\}$ and for all $i \in [1, n]$, we set $\mathfrak{h}_2(\mathfrak{l}^1_i) \stackrel{\text{def}}{=} \mathfrak{l}^2_i$.
4. for every $n \geq 0$, we have $n \geq \widetilde{\sharp\overline{\mathsf{u}}}$ in $\mathfrak{h}_1$ implies $n \geq \widetilde{\sharp\mathsf{u}} - 1$ in $\mathfrak{h}_1$. (by Lemma 3.2.4)
5. $\widetilde{\sharp\mathsf{u}} \leq \widetilde{\sharp\overline{\mathsf{u}}} + 1$. **QED**

Theorem 3.2.5 can be generalised by restricting ourselves to formulae in 1SL2($\mathrel{-\!\!*}$) and without assuming any peculiar property on the heap $\mathfrak{h}_1$.

Theorem 3.2.6. [DD14] There are formulae in 1SL2($\mathrel{-\!\!*}$) that can express the properties $\widetilde{\sharp\mathsf{u}} \leq \widetilde{\sharp\overline{\mathsf{u}}}$, $\widetilde{\sharp\mathsf{u}} \leq \widetilde{\sharp\overline{\mathsf{u}}} + 1$ or $\widetilde{\sharp\mathsf{u}} \leq \widetilde{\sharp\overline{\mathsf{u}}} - 1$ (whatever the heap).

3.3 Undecidabillity of 1SL2

3.3.1 Constraints between locations at distance three

This section is quite technical and it can be skipped for a first reading. Actually, Theorem 3.3.4 is used in forthcoming Section 3.3.2 in which runs of Minsky machines are encoded as $(\alpha, 2)$-fishbone heaps. So, we should be able to compare the respective numbers of predecessors for two locations on the main path whose distance is three (number of edges to reach one location from the other one). Indeed, this is required to guarantee that the encoding of the counter values respects the instructions of the Minsky machine.

The goal of this section is the following: given a formula $\chi(\mathrm{u}, \overline{\mathrm{u}})$ equal to either $\sharp\mathrm{u} = \sharp\overline{\mathrm{u}}$ or $\sharp\mathrm{u} = \sharp\overline{\mathrm{u}} + 1$ (in particular, this means that $\chi(\mathrm{u}, \overline{\mathrm{u}})$ only deals with numbers of predecessors and Section 3.2 explains how to define these formulae in 1SL2), we show how to define a formula in 1SL2, say $\chi^{+3}(\mathrm{u})$, such that

$$\mathfrak{h} \models_{\mathfrak{f}} \chi^{+3}(\mathrm{u}) \text{ iff } \mathfrak{h} \models_{\mathfrak{f}[\overline{\mathrm{u}} \mapsto \mathfrak{h}^3(\mathfrak{f}(\mathrm{u}))]} \chi(\mathrm{u}, \overline{\mathrm{u}}),$$

assuming that $\mathfrak{h}^3(\mathfrak{f}(\mathrm{u}))$ is defined, $\mathfrak{h} \models_{\mathfrak{f}} \mathbf{dw}(\alpha, 2) \wedge (\sharp\mathrm{u} \geq \alpha + 3))$ and $\mathfrak{f}(\mathrm{u})$ is on the main path. When $\chi(\mathrm{u}, \overline{\mathrm{u}})$ is equal to $\sharp\mathrm{u} = \sharp\overline{\mathrm{u}}$ [resp. $\sharp\mathrm{u} = \sharp\overline{\mathrm{u}} + 1$], we write $\sharp\mathrm{u} = \sharp\mathrm{u}^{+3}$ [resp. $\sharp\mathrm{u} = \sharp\mathrm{u}^{+3} + 1$] instead of $\chi^{+3}(\mathrm{u})$. Note that if we had three quantified variables, defining $\chi^{+3}(\mathrm{u})$ would not require much work since the formula below does the job:

$$\exists\, \mathrm{u}'\, (\mathrm{u} \hookrightarrow \mathrm{u}' \wedge \exists\, \overline{\mathrm{u}}\, (\mathrm{u}' \hookrightarrow \overline{\mathrm{u}} \wedge \exists\, \mathrm{u}'\, (\overline{\mathrm{u}} \hookrightarrow \mathrm{u}' \wedge \chi(\mathrm{u}, \mathrm{u}')))).$$

Let us start our construction. To do so, let $\mathfrak{h}$ be a heap and $\mathfrak{f}$ be an assignment such that $\mathfrak{h} \models_{\mathfrak{f}} \mathbf{dw}(\alpha, 2) \wedge (\sharp\mathrm{u}^{+3} \geq 0) \wedge (\sharp\mathrm{u} \geq \alpha + 3))$. In the statements below, this property is always satisfied.

The **u-3cut** of $\mathfrak{h}$ is the minimal subheap $\mathfrak{h}_{3cut}$ of $\mathfrak{h}$ (with respect to set inclusion of the domain and therefore $\mathfrak{h}_{3cut} \sqsubseteq \mathfrak{h}$) such that all the ancestors of $\mathrm{l}' = \mathfrak{h}^3(\mathfrak{f}(\mathrm{u}))$ in $\mathrm{dom}(\mathfrak{h})$ are also ancestors of l' in $\mathfrak{h}_{3cut}$. As a consequence, $\mathfrak{f}(\mathrm{u})$ and l' have the same amount of predecessors in $\mathfrak{h}$ and in the u-3cut heap.

In Figure 3.4, the bottom left heap is the u-3cut of the heap at the top. When $\mathfrak{h} \models_{\mathfrak{f}} \sharp\mathrm{u}^{+4} \geq 0$, the **almost u-3cut** of $\mathfrak{h}$ is the minimal subheap of $\mathfrak{h}$ containing the u-3cut heap and such that $\sharp\mathrm{u}^{+4} = 1$ holds true. The almost u-3cut of $\mathfrak{h}$ contains the edge from l' which is the only predecessor of the interpretation of u^{+4}. In Figure 3.4, the middle left heap is the almost u-3cut of the heap at the top. Below, we explain how to obtain the u-3cut of some heap, possibly via the construction of the almost u-3cut, if it exists.

Lemma 3.3.1 below states that all we need to define $\chi^{+3}(\mathtt{u})$ is to be able to express a property in its $\mathtt{u}$-3cut. In particular, the only location that is unallocated and on the main path is $\mathfrak{h}^3(\mathfrak{f}(\mathtt{u}))$.

Lemma 3.3.1. Let $\mathfrak{h} \models_{\mathfrak{f}} \mathbf{dw}(\alpha, 2) \wedge (\sharp\mathtt{u}^{+3} \geq 0) \wedge (\sharp\mathtt{u} \geq \alpha + 3))$ and $\mathfrak{h}'$ be its $\mathtt{u}$-3cut heap. Then, $\mathfrak{h} \models_{\mathfrak{f}[\overline{\mathtt{u}} \mapsto \mathfrak{h}^3(\mathfrak{f}(\mathtt{u}))]} \chi(\mathtt{u}, \overline{\mathtt{u}})$ iff $\mathfrak{h}' \models_{\mathfrak{f}} (\exists\, \overline{\mathtt{u}}\ \neg\mathtt{alloc}(\overline{\mathtt{u}}) \wedge \sharp\overline{\mathtt{u}} \geq 1 \wedge \chi(\mathtt{u}, \overline{\mathtt{u}}))$.

Proof. Let $\mathtt{l}' = \mathfrak{h}^3(\mathfrak{f}(\mathtt{u}))$ and $\mathtt{l} = \mathfrak{f}(\mathtt{u})$. Let $\mathfrak{h}'$ be the $\mathtt{u}$-3cut heap of $\mathfrak{h}$. We have (†) $\widetilde{\sharp\mathtt{l}}$ in $\mathfrak{h}$ is equal to $\sharp\mathtt{l}$ in $\mathfrak{h}'$ and $\widetilde{\sharp\mathtt{l}'}$ in $\mathfrak{h}$ is equal to $\sharp\mathtt{l}'$ in $\mathfrak{h}'$. Indeed, the $\mathtt{u}$-3cut heap $\mathfrak{h}'$ is a subheap of $\mathfrak{h}$ such that all the ancestors of $\mathtt{l}'$ in $\mathfrak{h}$ are also ancestors of $\mathtt{l}'$ in $\mathfrak{h}'$ and $\mathtt{l}$ is an ancestor of $\mathtt{l}'$ in $\mathfrak{h}$. Note also that $\mathtt{l}'$ is the unique location such that $\mathfrak{h}' \models_{[\overline{\mathtt{u}} \mapsto \mathtt{l}']} \neg\mathtt{alloc}(\overline{\mathtt{u}}) \wedge \sharp\overline{\mathtt{u}} \geq 1$. So, $\mathfrak{h}' \models_{\mathfrak{f}} (\exists\, \overline{\mathtt{u}}\ \neg\mathtt{alloc}(\overline{\mathtt{u}}) \wedge \sharp\overline{\mathtt{u}} \geq 1 \wedge \chi(\mathtt{u}, \overline{\mathtt{u}}))$ iff $\mathfrak{h}' \models_{\mathfrak{f}[\overline{\mathtt{u}} \mapsto \mathtt{l}']} \chi(\mathtt{u}, \overline{\mathtt{u}}))$ iff $\mathfrak{h} \models_{\mathfrak{f}[\overline{\mathtt{u}} \mapsto \mathfrak{h}^3(\mathfrak{f}(\mathtt{u}))]} \chi(\mathtt{u}, \overline{\mathtt{u}})$ by (†). Note that we use the fact that $\chi(\mathtt{u}, \overline{\mathtt{u}})$ specifies a property about the numbers of predecessors. **QED**

When $\mathfrak{h}$ is equal to its $\mathtt{u}$-3cut, i.e. when $(\sharp\mathtt{u}^{+4} \geq 0)$ does not hold, we have $\mathfrak{h} \models_{\mathfrak{f}[\overline{\mathtt{u}} \mapsto \mathfrak{h}^3(\mathfrak{f}(\mathtt{u}))]} \chi(\mathtt{u}, \overline{\mathtt{u}})$ iff $\mathfrak{h} \models_{\mathfrak{f}} \varphi_{\mathrm{UC}}(\mathtt{u})$ with

$$\varphi_{\mathrm{UC}}(\mathtt{u}) \stackrel{\text{def}}{=} (\exists\, \overline{\mathtt{u}}\ \neg\mathtt{alloc}(\overline{\mathtt{u}}) \wedge \sharp\overline{\mathtt{u}} \geq 1 \wedge \chi(\mathtt{u}, \overline{\mathtt{u}}))$$

Now, let us consider the case when $\mathfrak{h}$ is not equal to its $\mathtt{u}$-3cut (probably, the most common situation) and let us show how to separate the current heap so that we can isolate the $\mathtt{u}$-3cut heap.

Lemma 3.3.2. Let $\mathfrak{h} \models_{\mathfrak{f}} \mathbf{dw}(\alpha, 2) \wedge (\sharp\mathtt{u}^{+4} \geq 0) \wedge (\sharp\mathtt{u} \geq \alpha + 3))$ and $\varphi(\mathtt{u})$ be an arbitrary formula. Then, $\mathfrak{h} \models_{\mathfrak{f}} \mathtt{1comp} * (\mathtt{1comp} \wedge (\sharp\mathtt{u}^{+4} = 1) \wedge \neg(\sharp\mathtt{u}^{+5} \geq 0) \wedge \varphi(\mathtt{u}))$ iff the almost $\mathtt{u}$-3cut of $\mathfrak{h}$, say $\mathfrak{h}'$, satisfies: $\mathfrak{h}' \models_{\mathfrak{f}} \varphi(\mathtt{u})$.

The formula $\mathtt{1comp}$ was introduced in Section 1.2.2, and it states that the heap is made of a unique connected component (see also Exercise 1.15). The way $\mathfrak{h}$ has to be divided to satisfy the formula is illustrated by the two heaps in the middle of Figure 3.4.

Proof. Let $\mathfrak{h}$ be heap such that $\mathfrak{h} \models_{\mathfrak{f}} \mathbf{dw}(\alpha, 2) \wedge (\sharp\mathtt{u}^{+4} \geq 0) \wedge (\sharp\mathtt{u} \geq \alpha + 3)$. Let $\mathfrak{h}'$ be the almost $\mathtt{u}$-3cut heap of $\mathfrak{h}$ and $\mathfrak{h}''$ be the heap such that $\mathfrak{h} = \mathfrak{h}' \uplus \mathfrak{h}''$. By construction of $\mathfrak{h}'$, it is easy to check that $\mathfrak{h}' \models_{\mathfrak{f}} \mathtt{1comp} \wedge (\sharp\mathtt{u}^{+4} = 1) \wedge \neg(\sharp\mathtt{u}^{+5} \geq 0)$. Similarly, $\mathfrak{h}'' \models_{\mathfrak{f}} \mathtt{1comp}$. This implies that $\mathfrak{h} \models_{\mathfrak{f}} \mathtt{1comp} * (\mathtt{1comp} \wedge (\sharp\mathtt{u}^{+4} = 1) \wedge \neg(\sharp\mathtt{u}^{+5} \geq 0))$.

So, suppose that the almost $\mathtt{u}$-3cut heap of $\mathfrak{h}$ satisfies: $\mathfrak{h}' \models_{\mathfrak{f}} \varphi(\mathtt{u})$. This means that $\mathfrak{h}'' \models_{\mathfrak{f}} \mathtt{1comp}$ and $\mathfrak{h}' \models_{\mathfrak{f}} (\mathtt{1comp} \wedge (\sharp\mathtt{u}^{+4} = 1) \wedge \neg(\sharp\mathtt{u}^{+5} \geq 0) \wedge \varphi(\mathtt{u}))$. Hence, $\mathfrak{h} \models_{\mathfrak{f}} \mathtt{1comp} * (\mathtt{1comp} \wedge (\sharp\mathtt{u}^{+4} = 1) \wedge \neg(\sharp\mathtt{u}^{+5} \geq 0) \wedge \varphi(\mathtt{u}))$.

Now, suppose that $\mathfrak{h} \models_{\mathfrak{f}} \mathtt{1comp} * (\mathtt{1comp} \wedge (\sharp \mathtt{u}^{+4} = 1) \wedge \neg(\sharp \mathtt{u}^{+5} \geq 0) \wedge \varphi(\mathtt{u}))$. There are heaps $\mathfrak{h}_1$ and $\mathfrak{h}_2$ such that $\mathfrak{h}_2 \models \mathtt{1comp}$ and $\mathfrak{h}_1 \models_{\mathfrak{f}} (\mathtt{1comp} \wedge (\sharp \mathtt{u}^{+4} = 1) \wedge \neg(\sharp \mathtt{u}^{+5} \geq 0) \wedge \varphi(\mathtt{u}))$. In particular, this means that $\mathfrak{h}_1 \models_{\mathfrak{f}} \mathtt{1comp} \wedge (\sharp \mathtt{u}^{+4} = 1) \wedge \neg(\sharp \mathtt{u}^{+5} \geq 0)$.

Let us show that there is a unique pair $(\mathfrak{h}_1, \mathfrak{h}_2)$ of heaps satisfying that property and $\mathfrak{h}_1 = \mathfrak{h}'$, which will entail that $\mathfrak{h}' \models_{\mathfrak{f}} \varphi(\mathtt{u})$. First note that

$$\{\mathfrak{f}(\mathtt{u}), \mathfrak{h}(\mathfrak{f}(\mathtt{u})), \mathfrak{h}^2(\mathfrak{f}(\mathtt{u})), \mathfrak{h}^3(\mathfrak{f}(\mathtt{u})), \mathfrak{h}^4(\mathfrak{f}(\mathtt{u}))\} \subseteq \mathrm{dom}(\mathfrak{h}_1) \quad \mathfrak{h}^5(\mathfrak{f}(\mathtt{u})) \notin \mathrm{dom}(\mathfrak{h}_1)$$

Since $\mathfrak{h}_1 \models_{\mathfrak{f}} (\sharp \mathtt{u}^{+4} = 1)$, all the predecessors of $\mathfrak{h}^4(\mathfrak{f}(\mathtt{u}))$, apart from $\mathfrak{h}^3(\mathfrak{f}(\mathtt{u}))$, are in $\mathrm{dom}(\mathfrak{h}_2)$ and there are more than two such predecessors since $\mathfrak{h}^4(\mathfrak{f}(\mathtt{u}))$ is on the main path of $\mathfrak{h}$ and therefore has at least three predecessors in $\mathfrak{h}$.

Hence, $\mathfrak{h}_1$ contains also all the ancestors of $\mathfrak{h}^3(\mathfrak{f}(\mathtt{u}))$, otherwise $\mathfrak{h}_2$ would have at least two distinct connected components. So, the $\mathtt{u}$-3cut of $\mathfrak{h}$ is also a subheap of $\mathfrak{h}_1$.

Now, it is easy to check that if any location in $\mathrm{dom}(\mathfrak{h}'')$ that is not a predecessor of $\mathfrak{h}^4(\mathfrak{f}(\mathtt{u}))$ were in $\mathrm{dom}(\mathfrak{h}_1)$, then $\mathfrak{h}_1$ would have more than two connected components. Hence, $\mathfrak{h}_1$ is the almost $\mathtt{u}$-3cut heap of $\mathfrak{h}$ and therefore $\mathfrak{h}' \models_{\mathfrak{f}} \varphi(\mathtt{u})$. **QED**

Let us build on Lemma 3.3.2 so as to be able to specify properties on the $\mathtt{u}$-3cut heap.

Lemma 3.3.3. Let $\mathfrak{h} \models_{\mathfrak{f}} \mathbf{dw}(\alpha, 2) \wedge (\sharp \mathtt{u}^{+4} \geq 0) \wedge (\sharp \mathtt{u} \geq \alpha + 3))$ and $\varphi(\mathtt{u})$ be the formula $(\mathtt{size} = 1) * (\neg(\sharp \mathtt{u}^{+4} \geq 0) \wedge \varphi_{\mathrm{UC}}(\mathtt{u}))$. Then, $\mathfrak{h} \models_{\mathfrak{f}} \mathtt{1comp} * (\mathtt{1comp} \wedge (\sharp \mathtt{u}^{+4} = 1) \wedge \neg(\sharp \mathtt{u}^{+5} \geq 0) \wedge \varphi(\mathtt{u}))$ iff the $\mathtt{u}$-3cut of $\mathfrak{h}$, say $\mathfrak{h}'$, satisfies: $\mathfrak{h}' \models_{\mathfrak{f}} \varphi_{\mathrm{UC}}(\mathtt{u})$.

Below, the auxiliary formulae $\varphi(\mathtt{u}))$ and $\varphi_{\mathrm{AUC}}(\mathtt{u})$ (in 1SL2):

$$\begin{array}{lcl} \varphi(\mathtt{u}) & \stackrel{\mathrm{def}}{=} & (\mathtt{size} = 1) * (\neg(\sharp \mathtt{u}^{+4} \geq 0) \wedge \varphi_{\mathrm{UC}}(\mathtt{u})) \\ \varphi_{\mathrm{AUC}}(\mathtt{u}) & \stackrel{\mathrm{def}}{=} & \mathtt{1comp} * (\mathtt{1comp} \wedge (\sharp \mathtt{u}^{+4} = 1) \wedge \neg(\sharp \mathtt{u}^{+5} \geq 0) \wedge \varphi(\mathtt{u})) \end{array}$$

The proof for Lemma 3.3.3 is also by an easy verification by observing that an almost $\mathtt{u}$-3cut heap is equal to the $\mathtt{u}$-3cut plus one memory cell (see Figure 3.4).

By combining Lemma 3.3.1–3.3.3, we get the following proposition by performing a case analysis depending whether $\sharp \mathtt{u}^{+4} \geq 0$ holds true or not on the heap $\mathfrak{h}$.

Theorem 3.3.4. [DD15b] Let $\mathfrak{h}$ be a heap and $\mathfrak{f}$ be an assignment such that $\mathfrak{h} \models_{\mathfrak{f}} \mathbf{dw}(\alpha, 2) \wedge (\sharp \mathtt{u}^{+3} \geq 0) \wedge (\sharp \mathtt{u} \geq \alpha + 3))$. We have $\mathfrak{h} \models_{\mathfrak{f}[\overline{\mathtt{u}} \mapsto \mathfrak{h}^3(\mathfrak{f}(\mathtt{u}))]} \chi(\mathtt{u}, \overline{\mathtt{u}})$ iff $\mathfrak{h} \models_{\mathfrak{f}} \chi^{+3}(\mathtt{u})$ with the formula $\chi^{+3}(\mathtt{u})$ defined below:

$$\chi^{+3}(\mathtt{u}) \stackrel{\mathrm{def}}{=} (\neg(\sharp \mathtt{u}^{+4} \geq 0) \wedge \varphi_{\mathrm{UC}}(\mathtt{u})) \vee ((\sharp \mathtt{u}^{+4} \geq 0) \wedge \varphi_{\mathrm{AUC}}(\mathtt{u})).$$

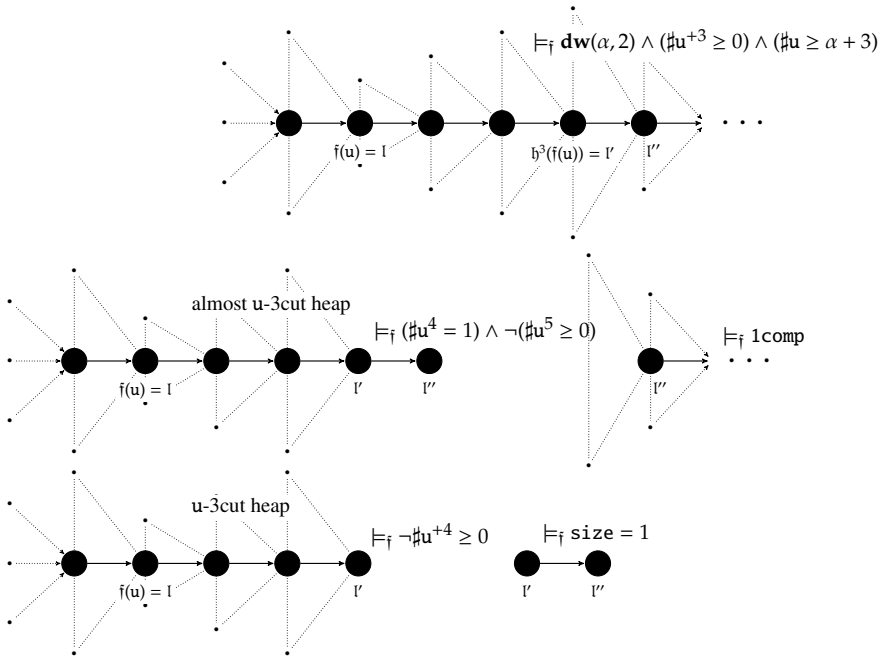


Figure 3.4: How to get a $\mathfrak{u}$-3cut – Decomposition in two stages.

Proof. We distinguish two cases depending whether $\mathfrak{h}$ is itself its $\mathfrak{u}$-3cut or not.
Case 1: $\neg(\sharp \mathfrak{u}^{+4} \geq 0)$, i.e. $\mathfrak{h}$ is its own $\mathfrak{u}$-3cut heap. By Lemma 3.3.1, if $\mathfrak{h} \models_{\mathfrak{f}[\overline{\mathfrak{u}} \mapsto \mathfrak{h}^3(\mathfrak{f}(\mathfrak{u}))]} \chi(\mathfrak{u}, \overline{\mathfrak{u}})$, then $\mathfrak{h} \models_{\mathfrak{f}} \varphi_{\mathrm{UC}}(\mathfrak{u})$ and therefore $\mathfrak{h} \models_{\mathfrak{f}} \chi^{+3}(\mathfrak{u})$. Conversely, if $\mathfrak{h} \models_{\mathfrak{f}} \chi^{+3}(\mathfrak{u})$, then $\mathfrak{h} \models_{\mathfrak{f}} \varphi_{\mathrm{UC}}(\mathfrak{u})$ since $(\sharp \mathfrak{u}^{+4} \geq 0)$ does not hold on $\mathfrak{h}$. Again, by Lemma 3.3.1, we get that $\mathfrak{h} \models_{\mathfrak{f}[\overline{\mathfrak{u}} \mapsto \mathfrak{h}^3(\mathfrak{f}(\mathfrak{u}))]} \chi(\mathfrak{u}, \overline{\mathfrak{u}})$.
Case 2: $(\sharp \mathfrak{u}^{+4} \geq 0)$. By Lemma 3.3.1, if $\mathfrak{h} \models_{\mathfrak{f}[\overline{\mathfrak{u}} \mapsto \mathfrak{h}^3(\mathfrak{f}(\mathfrak{u}))]} \chi(\mathfrak{u}, \overline{\mathfrak{u}})$, then $\mathfrak{h}' \models_{\mathfrak{f}} \varphi_{\mathrm{UC}}(\mathfrak{u})$ where $\mathfrak{h}'$ is the $\mathfrak{u}$-3cut of $\mathfrak{h}$. By Lemma 3.3.3, this implies that $\mathfrak{h} \models_{\mathfrak{f}} \varphi_{\mathrm{AUC}}(\mathfrak{u})$ and therefore $\mathfrak{h} \models_{\mathfrak{f}} \chi^{+3}(\mathfrak{u})$ (thanks to its second disjunction). Conversely, if $\mathfrak{h} \models_{\mathfrak{f}} \chi^{+3}(\mathfrak{u})$, then $\mathfrak{h} \models_{\mathfrak{f}} \varphi_{\mathrm{AUC}}(\mathfrak{u})$ since $(\sharp \mathfrak{u}^{+4} \geq 0)$ holds on $\mathfrak{h}$. Again, by Lemma 3.3.3, we get that $\mathfrak{h}' \models_{\mathfrak{f}[\overline{\mathfrak{u}} \mapsto \mathfrak{h}^3(\mathfrak{f}(\mathfrak{u}))]} \varphi_{\mathrm{UC}}(\mathfrak{u})$ and by Lemma 3.3.1, we conclude $\mathfrak{h} \models_{\mathfrak{f}[\overline{\mathfrak{u}} \mapsto \mathfrak{h}^3(\mathfrak{f}(\mathfrak{u}))]} \chi(\mathfrak{u}, \overline{\mathfrak{u}})$. **QED**

Note that the reasoning performed in this section cannot be extended to an arbitrary formula $\chi(\mathfrak{u}, \overline{\mathfrak{u}})$ since taking a $\mathfrak{u}$-3cut or an almost $\mathfrak{u}$-3cut preserves the number of predecessors of $\mathfrak{f}(\mathfrak{u})$ and $\mathfrak{h}^3(\mathfrak{f}(\mathfrak{u}))$ but may not preserve more general properties. Nevertheless, this is sufficient for our needs in Section 3.3.2.

3.3.2 Reduction from the halting problem for Minsky machines

Let $\mathtt{M}$ be a Minsky machine with $\alpha \geq 1$ instructions, where 1 is the initial instruction and α is the halting instruction. We recall that a machine $\mathtt{M}$ has two counters $\mathtt{c}_1$ and $\mathtt{c}_2$ and the instructions are of the following types:

1. I: $\mathtt{c}_j := \mathtt{c}_j + 1$; goto J.
2. I: if $\mathtt{c}_j = 0$ then goto J_1 else ($\mathtt{c}_j := \mathtt{c}_j - 1$; goto J_2).
3. α: halt.

When a Minsky machine $\mathtt{M}$ has $\alpha \geq 1$ instructions, any run starting from the initial instruction 1 and ending by the halting instruction α (there is a single such run since $\mathtt{M}$ is deterministic) is a data word of dimension two over the finite alphabet $[1, \alpha]$. We have seen that $(\alpha, 2)$-fishbone heaps can be characterised thanks to the formula $\mathbf{dw}(\alpha, 2)$. Obviously, more constraints need to be expressed, typically those related to the first instruction and those related to the halting instruction. Let us start by specifying the limit conditions thanks to the formulae φ_{first} and φ_{last} below.

- The first three locations on the main path have 3, $\alpha + 3$, and $\alpha + 3$ predecessors respectively:

$$\varphi_{\text{first}} \stackrel{\text{def}}{=} \exists\, \mathtt{u}\ \mathtt{first}(\mathtt{u}) \wedge (\sharp \mathtt{u} = 3) \wedge (\sharp \mathtt{u}^{+1} = \alpha + 3) \wedge (\sharp \mathtt{u}^{+2} = \alpha + 3).$$

- The main path encoding the run ends by a configuration with the halting instruction:

$$\varphi_{\text{last}} \stackrel{\text{def}}{=} \exists\, \mathtt{u}\ ((\sharp \mathtt{u} = \alpha + 2) \wedge (\sharp \mathtt{u}^{+2} \geq 0) \wedge \neg(\sharp \mathtt{u}^{+3} \geq 0)).$$

Let us call $\varphi^\star$ the conjunction of $\mathbf{dw}(\alpha, 2) \wedge \varphi_{\text{first}} \wedge \varphi_{\text{last}}$. It specifies the shape of the encoding of the run without taking care of the constraints about counter values and instruction counter.

Lemma 3.3.5. Let $\mathfrak{h}$ be a heap. $\mathfrak{h} \models \varphi^\star$ iff $\mathfrak{h}$ encodes a data word

$$\mathfrak{dw} = (a^1, \mathfrak{v}_1^1, \mathfrak{v}_2^1) \cdots (a^L, \mathfrak{v}_1^L, \mathfrak{v}_2^L)$$

such that $a^1 = 1$, $a^L = \alpha$, and $\mathfrak{v}_1^1 = \mathfrak{v}_2^1 = 0$.

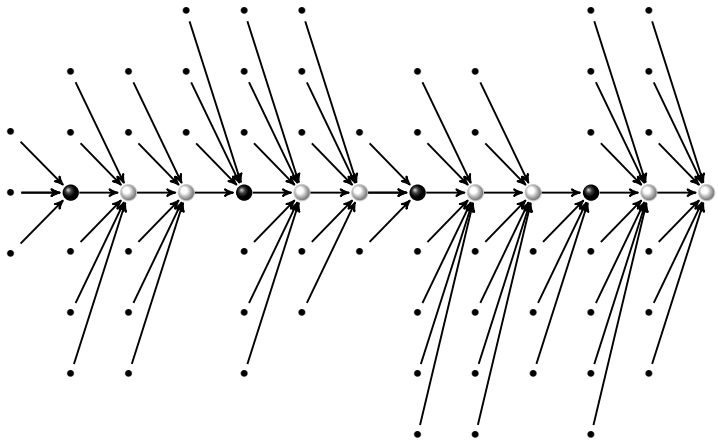


Figure 3.5: A $(3,2)$-fishbone heap encoding $(1,0,0) \rightarrow (2,1,0) \rightarrow (1,1,1) \rightarrow (2,2,1)$

Figure 3.5 presents an encoding of the sequence of configurations $(1,0,0) \rightarrow (2,1,0) \rightarrow (1,1,1) \rightarrow (2,2,1)$ when $\alpha = 3$. Note that the $(3,2)$-fishbone heap satisfies $\varphi^\star$.

We have provided formulae for basic properties about the encoding of the runs, but this is insufficient. Indeed, three consecutive locations on the main path encode a configuration of the Minsky machine $\mathtt{M}$. In order to check that two consecutive configurations correspond to a step that is valid for $\mathtt{M}$, we need to compare numbers of predecessors for locations on the main path at distance three from each other. To do so, we use formulae of the form $\chi^{+3}(\mathsf{u})$ when $\chi(\mathsf{u},\overline{\mathsf{u}})$ expresses one of the following arithmetical constraints: $\sharp\mathsf{u} = \sharp\overline{\mathsf{u}}$, $\sharp\mathsf{u} = \sharp\overline{\mathsf{u}} + 1$ and $\sharp\overline{\mathsf{u}} = \sharp\mathsf{u} + 1$ (see Section 3.3.1). For each instruction $I \in [1, \alpha - 1]$, we build a formula φ_I so that the Minsky machine $\mathtt{M}$ halts iff the formula

$$\varphi^\star \wedge \bigwedge_{I \in [1,\alpha-1]} \chi_I$$

is satisfiable in 1SL2. It remains to define χ_I for each instruction I.

If the instruction I is of the form "I: $\mathsf{c}_j := \mathsf{c}_j + 1$; goto J" then we need to check the following properties:

1. If a location $\mathfrak{l}$ encodes the instruction I on the main path (i.e. $\widetilde{\sharp\mathfrak{l}} = I + 2$) and $\mathfrak{h}^3(\mathfrak{l})$ is defined, then the location $\mathfrak{h}^3(\mathfrak{l})$ encodes the instruction J.

2. If a location $\mathfrak{l}$ encodes the value for the counter c_j in a configuration with instruction I (i.e., $\widetilde{\sharp\mathfrak{l}} \geq \alpha + 3$ and the jth ancestor of $\mathfrak{l}$ has $I + 2$ predecessors) and $\mathfrak{h}^3(\mathfrak{l})$ is defined, then $\widetilde{\sharp\mathfrak{l}} + 1 = \widetilde{\sharp\mathfrak{h}^3(\mathfrak{l})}$.

3. Similarly, if a location $\mathfrak{l}$ encodes the value for the counter c_{3-j} (i.e., the counter c_{3-J} is not updated after instruction I) in a configuration with instruction I (i.e., $\widetilde{\sharp\mathfrak{l}} \geq \alpha + 3$ and the jth ancestor of $\mathfrak{l}$ has $I + 2$ predecessors) and $\mathfrak{h}^3(\mathfrak{l})$ is defined, then $\widetilde{\sharp\mathfrak{l}} = \widetilde{\sharp\mathfrak{h}^3(\mathfrak{l})}$.

The properties can be expressed by the formula χ_I below:

$$\forall\, \mathfrak{u}\, (\sharp\mathfrak{u}^{+3} \geq 0) \Rightarrow [\overbrace{((\sharp\mathfrak{u} = I + 2) \Rightarrow (\sharp\mathfrak{u}^{+3} = J + 2))}^{(1)} \wedge$$
$$\overbrace{((\sharp\mathfrak{u} \geq \alpha + 3) \wedge (\sharp\mathfrak{u}^{-j} = I + 2) \Rightarrow (\sharp\mathfrak{u} = \sharp\mathfrak{u}^{+3} - 1))}^{(2)} \wedge$$
$$\overbrace{((\sharp\mathfrak{u} \geq \alpha + 3) \wedge (\sharp\mathfrak{u}^{j-3} = I + 2) \Rightarrow (\sharp\mathfrak{u} = \sharp\mathfrak{u}^{+3}))}^{(3)}].$$

Each subformula decorated by a curly bracket with (i) expresses exactly the property (i) above. Note that $\sharp\mathfrak{u}^{+3} = J + 2$ states that the number of predecessors of $\mathfrak{h}^3(\mathfrak{f}(\mathfrak{u}))$ is $J + 2$, which is quite easy to express in 1SL2 (see Section 1.2.2). By contrast, the formula $\sharp\mathfrak{u} = \sharp\mathfrak{u}^{+3} - 1$ states that the number of predecessors of $\mathfrak{h}^3(\mathfrak{f}(\mathfrak{u}))$ is equal to the number of predecessors of $\mathfrak{f}(\mathfrak{u})$ plus one, which requires the more sophisticated formulae introduced in Section 3.2 and in Section 3.3.1.

Similarly, let I be the instruction "I: if $\mathsf{c}_j = 0$ then goto J_1 else ($\mathsf{c}_j := \mathsf{c}_j - 1$; goto J_2)" then χ_I is defined as follows:

$$\forall\, \mathfrak{u}\, (\sharp\mathfrak{u}^{+3} \geq 0) \Rightarrow [\overbrace{((\sharp\mathfrak{u} = I + 2) \wedge (\sharp\mathfrak{u}^{+j} = \alpha + 3) \Rightarrow (\sharp\mathfrak{u}^{+3} = J_1 + 2))}^{(4)} \wedge$$
$$\overbrace{((\sharp\mathfrak{u} = I + 2) \wedge (\sharp\mathfrak{u}^{+j} > \alpha + 3) \Rightarrow (\sharp\mathfrak{u}^{+3} = J_2 + 2))}^{(5)} \wedge$$
$$\overbrace{((\sharp\mathfrak{u} > \alpha + 3) \wedge (\sharp\mathfrak{u}^{-j} = I + 2) \Rightarrow (\sharp\mathfrak{u}^{+3} = \sharp\mathfrak{u} - 1))}^{(6)} \wedge$$

$$\overbrace{((\sharp \mathsf{u} = \alpha + 3) \wedge (\sharp \mathsf{u}^{-j} = I + 2) \Rightarrow (\sharp \mathsf{u}^{+3} = \alpha + 3))}^{(7)} \wedge$$
$$\overbrace{((\sharp \mathsf{u} \geq \alpha + 3) \wedge (\sharp \mathsf{u}^{j-3} = I + 2) \Rightarrow (\sharp \mathsf{u} = \sharp \mathsf{u}^{+3}))}^{(8)}].$$

The subformula decorated by a curly bracket with (4) states that if a location $\mathfrak{l}$ encodes the instruction I and $\mathfrak{h}^j(\mathfrak{l})$ has $\alpha+3$ predecessors (i.e., counter c_j has value zero), then the location $\mathfrak{h}^3(\mathfrak{l})$ has $J_1 + 2$ predecessors (i.e., the next instruction is J_1). Similarly, the subformula decorated by a curly bracket with (5) states that if a location $\mathfrak{l}$ encodes the instruction I and $\mathfrak{h}^j(\mathfrak{l})$ has strictly more than $\alpha + 3$ predecessors (i.e., counter c_j has non-zero value), then the location $\mathfrak{h}^3(\mathfrak{l})$ has J_2+2 predecessors (i.e., the next instruction is J_2). Moreover, the subformula decorated by a curly bracket with (6) states that if a location $\mathfrak{l}$ has at least $\alpha + 3$ predecessors and its jth ancestor has $I + 2$ predecessors (i.e., counter c_j has non-zero value and we are really dealing with instruction I), then the number of predecessors of $\mathfrak{h}^3(\mathfrak{l})$ is equal to the number of predecessors of $\mathfrak{l}$ minus one, which corresponds to encode a decrement on counter c_j. Subformulae (7) and (8) admit a similar reading.

It is now easy to show the following lemma since we have seen that all the constraints between consecutive configurations can be encoded in 1SL2, assuming that the heap encodes a data word in $([1, \alpha] \times \mathbb{N}^2)^+$.

Lemma 3.3.6. $\mathtt{M}$ has a halting run iff

$$\mathbf{dw}(\alpha, 2) \ \wedge \ \varphi_{first} \ \wedge \ \varphi_{last} \ \wedge \ \bigwedge_{I \in [1,\alpha]} \chi_I$$

is satisfiable in 1SL2.

Below, we conclude by a major undecidability result.

Theorem 3.3.7. [DD15b] 1SL2 satisfiability problem is undecidable.

We know that if the number of quantified variables is not restricted, 1SL($\mathrel{-\!\!*}$) is undecidable too [BDL12] and recently the satisfiability problem for 1SL2($\mathrel{-\!\!*}$) has been shown undecidable as well [DD14], but this requires a far more complex proof passing via an equivalence to weak second-order logic (see the main steps of the proof in forthcoming Section 3.4).

3.4 Expressive Completeness

In Section 1.3.3, we have seen how 1SL2 can be translated into 1DSOL. Below, we provide the translation in the other direction with the fragment 1SL2($-\!\!*$), which is much more complicated as explained below. Material from this section is quite involved and can be skipped for a first reading; it is mainly taken from the submitted paper [DD14]. A reduction from 1DSOL into 1SL($-\!\!*$) can be also found in [BDL12].

In order to express sentences in DSOL by sentences in 1SL2($-\!\!*$), a hybrid valuation is encoded in the heap by building a disjoint **valuation heap** that takes care of pairs of locations (for interpretation of second-order variables) and that takes care of locations (for interpretation of first-order variables). In principle, this makes sense since every heap has a finite domain and therefore there is always an infinite set of locations that is not in its domain. This leaves enough room to encode a finite amount of information such as the interpretation of second-order variables when they are interpreted by finite sets. We can easily add to the original heap with the magic wand; this permits us to create and update the valuation heap. However, we then must always be able to distinguish between the original heap and the valuation heap.

The main idea to build such a valuation heap rests on the fact that a pair of locations $(\mathfrak{l}, \mathfrak{l}')$ belongs to the interpretation of a second-order variable P_i whenever $\mathfrak{l}$ and $\mathfrak{l}'$ can be identified in the valuation heap by special patterns involving $\mathfrak{l}$ and $\mathfrak{l}'$ that uniquely characterise the interpretation by P_i. Similarly, a location $\mathfrak{l}$ is the interpretation of a first-order variable whenever $\mathfrak{l}$ can be identified in the valuation heap thanks to some dedicated pattern around $\mathfrak{l}$.

Before explaining further the general principles, let us first provide more information about the above-mentioned patterns. An **entry of degree** $d \geq 2$ is a sequence of distinct locations $\mathfrak{l}_1, \ldots, \mathfrak{l}_d, \mathfrak{l}_{\mathfrak{ind}}, \mathfrak{l}$ such that

- $\mathfrak{h}(\mathfrak{l}_1) = \cdots = \mathfrak{h}(\mathfrak{l}_d) = \mathfrak{l}_{\mathfrak{ind}}$,
- $\widetilde{\sharp \mathfrak{l}_{\mathfrak{ind}}} = d$,
- $\widetilde{\sharp \mathfrak{l}_1} = \cdots = \widetilde{\sharp \mathfrak{l}_d} = 0$, and
- $\mathfrak{h}(\mathfrak{l}_{\mathfrak{ind}}) = \mathfrak{l}$.

The location $\mathfrak{l}$ is called the **element**, $\mathfrak{l}_{\mathfrak{ind}}$ the **index** and the locations $\mathfrak{l}_1, \ldots, \mathfrak{l}_d$, the **pins**. Entries generalise the notions of forks and large forks from Section 3.2

and are called markers in [BDL12]. See an entry of degree 4 in the middle of Figure 3.6. So, the pair of locations $(\mathrm{l}, \mathrm{l}')$ is identified as part of the interpretation of P_i when l and l' are elements of entries with very large degree. The above-mentioned special patterns are therefore entries, but we require that the degree of the respective entries for l and l' satisfy some arithmetical constraints, which is possible thanks to Theorem 3.2.5, and which allows us to relate l with l'.

Then, the principle of the translation consists in building the valuation heap on demand (typically when a quantification appears) and to find special patterns involving entries with large degree whenever an atomic formula needs to be evaluated.

These principles have been introduced in [BDL12] to translate 1DSOL formulae into 1SL($\mathrel{-\!\!*}$) formulae. However, because we are restricted to two first-order variables and because we also require that the separating conjunction is banished, we present below a different way to apply these principles so that we can show that 1SL2($\mathrel{-\!\!*}$) is expressively equivalent to DSOL (and therefore to 1WSOL).

This high-level description of the formula translation and of the encoding of some hybrid valuation in the heap hides many of the details, which can be found below. However, before explaining how we apply these principles within 1SL2($\mathrel{-\!\!*}$), let us emphasise the most obvious and difficult problems to be solved:

- we must be able to distinguish the pairs of locations from distinct second-order variables,
- we also need to encode first-order valuations, and
- the main problem is certainly to access the original heap properly without interference from the valuation heap.

3.4.1 Left and right parentheses

We introduce variants of entries that are used as delimiters.

A **left j-parenthesis of degree $d \geq 3$ with $j \geq 0$** is a sequence of distinct locations $\mathrm{l}'_{j+1}, \ldots, \mathrm{l}'_1, \mathrm{l}_1, \ldots, \mathrm{l}_d, \mathrm{l}_{\mathrm{ind}}$ such that

(u) $\mathfrak{h}(\mathrm{l}_1) = \cdots = \mathfrak{h}(\mathrm{l}_d) = \mathrm{l}_{\mathrm{ind}}$; $\widetilde{\sharp \mathrm{l}_{\mathrm{ind}}} = d$; $\widetilde{\sharp \mathrm{l}'_{j+1}} = \widetilde{\sharp \mathrm{l}_3} = \widetilde{\sharp \mathrm{l}_4} = \cdots = \widetilde{\sharp \mathrm{l}_d} = 0$,

(v) $\mathrm{l}_{\mathrm{ind}} \notin \mathrm{dom}(\mathfrak{h})$; $\mathrm{l}'_{j+1} \to \mathrm{l}'_j \to \mathrm{l}'_{j-1} \to \cdots \to \mathrm{l}'_1 \to \mathrm{l}_1$; $\widetilde{\sharp \mathrm{l}'_j} = \widetilde{\sharp \mathrm{l}'_{j-1}} = \cdots = \widetilde{\sharp \mathrm{l}'_1} = \widetilde{\sharp \mathrm{l}_1} = 1$, and

(w) $\widetilde{\sharp \mathfrak{l}_2} = 0$.

The location $\mathfrak{l}_{\mathfrak{ind}}$ is called the **index**. The heap at the left of Figure 3.6 presents a left j-parenthesis of degree 3.

A **right j-parenthesis of degree $d \geq 3$ with $j \geq 0$** is a sequence of distinct locations $\mathfrak{l}'_{j+1}, \ldots, \mathfrak{l}'_1, \mathfrak{l}''_{j+1}, \ldots, \mathfrak{l}''_1, \mathfrak{l}_1, \ldots, \mathfrak{l}_d, \mathfrak{l}_{\mathfrak{ind}}$ such that (u), (v), and

- $\widetilde{\sharp \mathfrak{l}''_{j+1}} = 0$,
- $\widetilde{\sharp \mathfrak{l}''_j} = \widetilde{\sharp \mathfrak{l}''_{j-1}} = \cdots = \widetilde{\sharp \mathfrak{l}''_1} = \widetilde{\sharp \mathfrak{l}_2} = 1$, and
- $\mathfrak{l}''_{j+1} \to \mathfrak{l}''_j \to \mathfrak{l}''_{j-1} \to \cdots \to \mathfrak{l}''_1 \to \mathfrak{l}_2$.

The location $\mathfrak{l}_{\mathfrak{ind}}$ is also called the **index**. The heap at the right of Figure 3.6 presents a right j-parenthesis of degree 5. A j-parenthesis can be understood as an entry, except that the index location is not allocated, and containing one or two paths of length $j+1$, depending whether it is a left or a right parenthesis.

Lemma 3.4.1. For all $j \geq 0$, there is a formula $\mathtt{lp}_j(\mathtt{u})$ [resp. $\mathtt{rp}_j(\mathtt{u})$] in 1SL2($\mathrel{-\!\!*}$) such that for all heaps $\mathfrak{h}$ and valuations $\mathfrak{f}$, we have $\mathfrak{h} \models_{\mathfrak{f}} \mathtt{lp}_j(\mathtt{u})$ [resp. $\mathfrak{h} \models_{\mathfrak{f}} \mathtt{rp}_j(\mathtt{u})$] iff $\mathfrak{f}(\mathtt{u})$ is the index of some left [resp. right] j-parenthesis in $\mathfrak{h}$.

Proof. Let us start by defining formulae for backward paths of length $j+1$:

- $\mathtt{bpath}(1, \mathtt{u}) \stackrel{\text{def}}{=} (\sharp \mathtt{u} = 1) \wedge \forall\, \overline{\mathtt{u}}\, (\overline{\mathtt{u}} \hookrightarrow \mathtt{u}) \Rightarrow \sharp \overline{\mathtt{u}} = 0$.
- $\mathtt{bpath}(j+1, \mathtt{u}) \stackrel{\text{def}}{=} (\sharp \mathtt{u} = 1) \wedge \exists\, \overline{\mathtt{u}}\, (\overline{\mathtt{u}} \hookrightarrow \mathtt{u}) \wedge \mathtt{bpath}(j, \overline{\mathtt{u}})$.

So, whenever $j \geq 0$, we have $\mathfrak{h} \models_{\mathfrak{f}} \mathtt{bpath}(j+1, \mathtt{u})$ iff there are $\mathfrak{l}_0, \ldots, \mathfrak{l}_j$ such that $\mathfrak{l}_0 \to \mathfrak{l}_1 \to \cdots \to \mathfrak{l}_j \hookrightarrow \mathfrak{f}(\mathtt{u})$ and $\widetilde{\sharp \mathfrak{l}_0} = 0$, for every $k \in [1, j]$ $\widetilde{\sharp \mathfrak{l}_k} = 1$ and $\widetilde{\sharp \mathfrak{f}(\mathtt{u})} = 1$.

The formula below characterises the locations such that the predecessors either have no predecessor or have a backward path of length $j+1$ exactly:

$$\chi_{j+1}(\mathtt{u}) \stackrel{\text{def}}{=} \forall\, \overline{\mathtt{u}}\, (\overline{\mathtt{u}} \hookrightarrow \mathtt{u}) \Rightarrow (\sharp \overline{\mathtt{u}} = 0 \vee \mathtt{bpath}(j+1, \overline{\mathtt{u}})).$$

Then, the formulae $\mathtt{lp}_j(\mathtt{u})$ and $\mathtt{rp}_j(\mathtt{u})$ are defined as follows:

- $\mathtt{lp}_j(\mathtt{u}) \stackrel{\text{def}}{=} \neg \mathtt{alloc}(\mathtt{u}) \wedge \chi_{j+1}(\mathtt{u}) \wedge (\sharp \mathtt{u} \geq 3) \wedge (\exists\, \overline{\mathtt{u}}\, (\overline{\mathtt{u}} \hookrightarrow \mathtt{u}) \wedge \mathtt{bpath}(j+1, \overline{\mathtt{u}})) \wedge ((\mathtt{size} = 1) \mathrel{-\!\!\!\!\!\!\rightarrow\!\!\!*} \chi_{j+2}(\mathtt{u}))$.

- $\mathtt{rp}_j(\mathsf{u}) \stackrel{\text{def}}{=} \neg\mathtt{alloc}(\mathsf{u}) \wedge \chi_{j+1}(\mathsf{u}) \wedge (\sharp\mathsf{u} \geq 3) \wedge (\exists\, \overline{\mathsf{u}}\, (\overline{\mathsf{u}} \hookrightarrow \mathsf{u}) \wedge \mathtt{bpath}(j+1, \overline{\mathsf{u}})) \wedge \neg((\mathtt{size} = 1) \mathrel{\overrightarrow{-\!\!*}} \chi_{j+2}(\mathsf{u})) \wedge ((\mathtt{size} = 1) \mathrel{\overrightarrow{-\!\!*}} ((\mathtt{size} = 1) \mathrel{\overrightarrow{-\!\!*}} \chi_{j+2}(\mathsf{u})))$.

The formula $\mathtt{lp}_j(\mathsf{u})$ states that $\mathfrak{f}(\mathsf{u})$ is not allocated, it has at least three predecessors and any predecessor of $\mathfrak{f}(\mathsf{u})$ either has no predecessor or has a backward path of length $j+1$. Moreover, there is at least one predecessor of $\mathfrak{f}(\mathsf{u})$ that has a backward path of length $j+1$ thanks to the satisfaction of the subformula $(\exists\, \overline{\mathsf{u}}\, (\overline{\mathsf{u}} \hookrightarrow \mathsf{u}) \wedge \mathtt{bpath}(j+1, \overline{\mathsf{u}}))$. Satisfaction of the subformula $(\mathtt{size} = 1) \mathrel{\overrightarrow{-\!\!*}} \chi_{j+2}(\mathsf{u})$ entails that there is only one such backward path of length $j+1$. A similar analysis can be performed with the formula $\mathtt{rp}_j(\mathsf{u})$ with the exception that it is required to guarantee that there are exactly two predecessors of $\mathfrak{f}(\mathsf{u})$ that have a backward path of length $j+1$. **QED**

In several places, we need to identify the indices from entries as well as their pins. Let $\mathtt{eindex}(\mathsf{u})$ be defined as follows:

$$\mathtt{eindex}(\mathsf{u}) \stackrel{\text{def}}{=} (\sharp_z\mathsf{u} \geq 2) \wedge \mathtt{allzpred}(\mathsf{u}) \wedge \exists\, \overline{\mathsf{u}}\, \mathsf{u} \hookrightarrow \overline{\mathsf{u}}$$

that characterises indices from entries. The formula $\sharp_z\mathsf{u} \geq 2$ states that the number of predecessors of u with zero predecessor is at least 2 and $\mathtt{allzpred}(\mathsf{u})$ holds true when all the predecessors of u have no predecessor, see Exercise 3.5. Let $\mathtt{epin}(\mathsf{u})$ characterise pins from entries:

$$\mathtt{epin}(\mathsf{u}) \stackrel{\text{def}}{=} \exists\, \overline{\mathsf{u}}\, \mathsf{u} \hookrightarrow \overline{\mathsf{u}} \;\wedge\; \mathtt{eindex}(\overline{\mathsf{u}}).$$

Similarly, we need to characterise the locations from parentheses. We already know how to identify their indices (Lemma 3.4.1). It remains to identify the other locations via the formula $\mathtt{onpar}_i(\mathsf{u})$ to characterise the locations on some i-parenthesis: roughly speaking, such locations are exactly those that can reach the index of some i-parenthesis in less than $i+2$ steps. Let $\mathtt{onpar}_i(\mathsf{u})$ be the formula

$$\mathtt{onpar}_i(\mathsf{u}) \stackrel{\text{def}}{=} \bigvee_{j=0}^{i+2} \mathtt{dist}_i(j, \mathsf{u})$$

with $\mathtt{dist}_i(0, \mathsf{u}) \stackrel{\text{def}}{=} \mathtt{lp}_i(\mathsf{u}) \vee \mathtt{rp}_i(\mathsf{u})$, and $\mathtt{dist}_i(j+1, \mathsf{u}) \stackrel{\text{def}}{=} \exists\, \overline{\mathsf{u}}\, (\mathsf{u} \hookrightarrow \overline{\mathsf{u}}) \wedge \mathtt{dist}_i(j, \overline{\mathsf{u}})$ for all $j \geq 0$.

Lemma 3.4.2. Let $\mathfrak{h}$ be a heap, $\mathfrak{f}$ be a valuation and $i \geq 0$. Then, $\mathfrak{h} \models_{\mathfrak{f}} \mathtt{onpar}_i(\mathsf{u})$ iff $\mathfrak{f}(\mathsf{u})$ is on some left or right i-parenthesis in $\mathfrak{h}$.

Proof. The proof takes advantage of the following properties.

- $\mathfrak{h} \models_{\mathfrak{f}} \mathtt{dist}_i(j, \mathtt{u})$ iff $\mathfrak{f}(\mathtt{u})$ can reach an index location from a left i-parenthesis or from a right i-parenthesis, in j steps for some $j \geq 0$. The proof is obvious, by induction on j.

- If $\mathfrak{f}(\mathtt{u})$ can reach an index location from an i-parenthesis, then $\mathfrak{f}(\mathtt{u})$ is necessarily on an i-parenthesis.

- Every location on an i-parenthesis can reach its index in less than $i + 2$ steps.

As a conclusion, $\mathfrak{f}(\mathtt{u})$ is on an i-parenthesis iff it can reach the index of some i-parenthesis in less than $i + 2$ steps, which is exactly the way $\mathtt{onpar}_i(\mathtt{u})$ is defined with the help of the generalised disjunction. **QED**

3.4.2 The role of parentheses

Before explaining the role of parentheses, we introduce the interval of variable indices $[1, K]$ ($K \in \mathbb{N} \setminus \{0\}$) assuming that for each $j \in [1, K]$, either $\mathtt{P}_j$ or $\mathtt{u}_j$ occurs in the 1DSOL formula to be translated (but not both of them). So, the developments below are relative to a finite set of first-order and second-order variables and this is concretised by the interval $[1, K]$ (always possible since a formula has a finite number of variables).

Let us come back to parentheses and assume that X is a subset of $[0, K]$. In an X-well-formed heap $\mathfrak{h}$ (see Definition 3.4.9 below), the parentheses play the following role. For each $j \in X$, we have the index location $\mathfrak{lp}_j$ from a distinguished left j-parenthesis and the index location $\mathfrak{rp}_j$ from a distinguished right j-parenthesis. Moreover, let $d^l_j = \widetilde{\sharp \mathfrak{lp}_j}$ and $d^r_j = \widetilde{\sharp \mathfrak{rp}_j}$ (in $\mathfrak{h}$). When $j \in X$ is related to a first-order variable, we require that $d^r_j = d^l_j + 2$ and there is an entry of degree $d^l_j + 1$ such that its element is understood as the interpretation of the variable $\mathtt{u}_j$ (see Figure 3.6 with $d^r_j = 5$ and $d^l_j = 3$). That explains why the parentheses are viewed as delimiters.

Similarly, let $\{(\mathfrak{l}_1, \mathfrak{l}'_1), \ldots, (\mathfrak{l}_\beta, \mathfrak{l}'_\beta)\}$ be a finite set of pairs of locations, understood as the interpretation of a second-order variable $\mathtt{P}_j$ with $j \in X$. In $\mathfrak{h}$, there are 2β entries whose respective degrees are exactly $\{d^l_j + 3(i-1) + 1, d^l_j + 3(i-1) + 2 : i \in [1, \beta]\}$ with $d^r_j = d^l_j + 3\beta + 1$. A pair of entries of respective degrees $d^l_j + 3(i-1) + 1$ and $d^l_j + 3(i-1) + 2$ have exactly as elements $\mathfrak{l}_i$ and $\mathfrak{l}'_i$ respectively, which allows

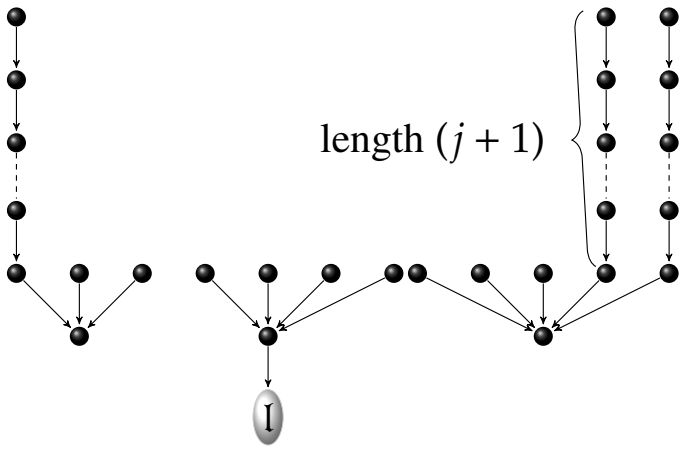


Figure 3.6: Encoding $[\mathfrak{u}_j \mapsto \mathfrak{l}]$.

to encode the pair $(\mathfrak{l}_i, \mathfrak{l}'_i)$. All this underlying encoding makes sense only if the left and right parentheses as well as the entries whose degrees are related to their degrees are uniquely determined (see Condition (1) in Definition 3.4.5, below).

For this reason, we introduce a left 0-parenthesis and a right 0-parenthesis with $d^r_0 = d^l_0 + 1$ (0 is not a variable index), the degree d^l_0 is strictly greater than the degree of any location in the original heap, all degrees d^l_j with $j \neq 0$ are strictly greater than d^l_0 and finally, the above-mentioned entries and parentheses are the only ones with their respective degrees. This guarantees that any entry from a pair of entries with successive degrees serving for the interpretation of a second-order variable, cannot serve twice for another pair or for another variable. Below, we provide the technical developments.

We say that a heap $\mathfrak{h}$ is **made of entries and parentheses only** $\overset{\text{def}}{\Leftrightarrow}$ every location in dom($\mathfrak{h}$) belongs either to a left i-parenthesis for some $i \geq 0$, to a right i-parenthesis for some $i \geq 0$, or to an entry. Given a heap $\mathfrak{h}$ made of entries and parentheses only, we define the set *indspect*($\mathfrak{h}$) as follows:

$$\mathit{indspect}(\mathfrak{h}) \overset{\text{def}}{=} \{\widetilde{\sharp}\mathfrak{l} : \mathfrak{l} \textit{ is the index of some entry or parenthesis in } \mathfrak{h}\}$$

This set *indspect*($\mathfrak{h}$) is called the **index spectrum** of $\mathfrak{h}$.

Let $\mathfrak{h}_B$ be a heap such that $\alpha = max(\{\widetilde{\sharp}\mathfrak{l} : \mathfrak{l} \in \mathbb{N}\})$. For instance, if $\mathfrak{h}_B$ has empty domain, then $\alpha = 0$.

We can now find locations in a heap with a maximal number of predecessors,

and we conclude this section with a definition useful in later constructions. Let us introduce the formula $\mathtt{maxdeg}(\mathtt{u})$:

$$\mathtt{maxdeg}(\mathtt{u}) \stackrel{\text{def}}{=} \neg\exists\, \overline{\mathtt{u}}\ \sharp\overline{\mathtt{u}} > \sharp\mathtt{u}$$

Corollary 3.4.3. For all heaps $\mathfrak{h}$ and valuations $\mathfrak{f}$, we have $\mathfrak{h} \models_{\mathfrak{f}} \mathtt{maxdeg}(\mathtt{u})$ iff $\widetilde{\sharp\mathfrak{f}(\mathtt{u})} = \mathit{max}(\{\widetilde{\sharp\mathfrak{l}} : \mathfrak{l} \in \mathbb{N}\})$.

A valuation heap $\mathfrak{h}_V$ for $\mathfrak{h}_B$ is made of entries and parentheses only whose degrees are greater than $\mathit{max}(3, \alpha + 1)$. The heap $\mathfrak{h}_V$ satisfies the following simple conditions (more constraints will follow): $\mathit{min}(\mathit{indspect}(\mathfrak{h}_V))$ is greater than the value $\mathit{max}(3, \alpha + 1)$ and it is witnessed by the degree of some left 0-parenthesis; each degree in $\mathit{indspect}(\mathfrak{h}_V)$ is witnessed by exactly one entry or parenthesis. Formula $\mathtt{indmin}(\mathtt{u})$ below is satisfied in $\mathfrak{h} = \mathfrak{h}_B \uplus \mathfrak{h}_V$ by a location $\mathfrak{l}$ witnessing the minimal value in $\mathit{indspect}(\mathfrak{h}_V)$:

$$\mathtt{indmin}(\mathtt{u}) \stackrel{\text{def}}{=} \mathtt{lp}_0(\mathtt{u}) \wedge (\forall\, \overline{\mathtt{u}}\ ((\overline{\mathtt{u}} \neq \mathtt{u}) \wedge \mathtt{lp}_0(\overline{\mathtt{u}})) \Rightarrow \sharp\overline{\mathtt{u}} < \sharp\mathtt{u}).$$

Thanks to Section 3.2, we know that it is possible to compare numbers of predecessors as expressed above. So, $\mathtt{indmin}(\mathtt{u})$ holds when $\mathfrak{f}(\mathtt{u})$ is the unique location that is the index of some left 0-parenthesis with greatest degree.

Lemma 3.4.4. Let $\mathfrak{f}$ be a valuation and $\mathfrak{h}$ be a heap. We have $\mathfrak{h} \models_{\mathfrak{f}} \mathtt{indmin}(\mathtt{u})$ iff $\mathfrak{f}(\mathtt{u})$ is an index of some left 0-parenthesis and there is no other location $\mathfrak{l} \neq \mathfrak{f}(\mathtt{u})$ such that $\widetilde{\sharp\mathfrak{l}} \geq \widetilde{\sharp\mathfrak{f}(\mathtt{u})}$ and $\mathfrak{l}$ is the index of some left 0-parenthesis.

The proof is by an easy verification by using Lemma 3.4.1. Once a heap $\mathfrak{h}$ satisfies the formula $\exists\, \mathtt{u}\ \mathtt{indmin}(\mathtt{u})$, the unique location $\mathfrak{l}_0$ such that $\mathfrak{h} \models_{[\mathtt{u}\mapsto\mathfrak{l}_0]} \mathtt{indmin}(\mathtt{u})$ (say with $\widetilde{\sharp\mathfrak{l}_0} = d_0$) plays the role of a delimiter between the original heap and the part of the heap that encodes the hybrid valuation.

We have seen that an index spectrum is defined for heaps made of entries and parentheses only. This is fine, but below we adapt the definition to heaps $\mathfrak{h}$ satisfying $\exists\, \mathtt{u}\ \mathtt{indmin}(\mathtt{u})$. Let us define the set $\mathit{spect}(\mathfrak{h})$ as follows:

$$\mathit{spect}(\mathfrak{h}) \stackrel{\text{def}}{=} \{\widetilde{\sharp\mathfrak{l}} : \mathfrak{l} \textit{ is an index of some entry or parenthesis in } \mathfrak{h}\} \cap [d_0, +\infty[.$$

The set $\mathit{spect}(\mathfrak{h})$ is called the **spectrum** of $\mathfrak{h}$. This illustrates how the location $\mathfrak{l}_0$ and the degree $\widetilde{\sharp\mathfrak{l}_0} = d_0$ play the role of separator between the original heap and the valuation heap.

The subheap encoding the valuation is made of parentheses and entries and we shall need to identify the indices of such patterns. The formula $\mathtt{Lindex}(\mathtt{u})$ defined below suffices for this purpose:

$$\mathtt{Lindex}(\mathtt{u}) \stackrel{\text{def}}{=} (\exists\, \overline{\mathtt{u}}\ \mathtt{indmin}(\overline{\mathtt{u}}) \wedge \sharp\overline{\mathtt{u}} \leq \sharp\mathtt{u}) \wedge \Big(\big(\bigvee_{i\in[0,K]} (\mathtt{lp}_i(\mathtt{u}) \vee \mathtt{rp}_i(\mathtt{u}))\big) \vee \mathtt{eindex}(\mathtt{u})\Big)$$

($\mathtt{u}$ is interpreted as a **large index**). Given $X \subseteq [0, K]$, we shall use also the following formula:

$$\mathtt{Lindex}_X(\mathtt{u}) \stackrel{\text{def}}{=} (\exists\, \overline{\mathtt{u}}\ \mathtt{indmin}(\overline{\mathtt{u}}) \wedge \sharp\overline{\mathtt{u}} \leq \sharp\mathtt{u}) \wedge \Big(\big(\bigvee_{i\in X} (\mathtt{lp}_i(\mathtt{u}) \vee \mathtt{rp}_i(\mathtt{u}))\big) \vee \mathtt{eindex}(\mathtt{u})\Big).$$

Entries and parentheses with large indices are also called large entries and parentheses, respectively. It is easy to define a large index that is also the index of a left [resp. right] parenthesis. Let $\mathtt{llp}_i(\mathtt{u}) \stackrel{\text{def}}{=} \mathtt{Lindex}(\mathtt{u}) \wedge \mathtt{lp}_i(\mathtt{u})$ and $\mathtt{lrp}_i(\mathtt{u}) \stackrel{\text{def}}{=} \mathtt{Lindex}(\mathtt{u}) \wedge \mathtt{rp}_i(\mathtt{u})$ (see Lemma 3.4.1). The large index with a maximal degree can be also characterised as follows:

$$\mathtt{maxLindex}(\mathtt{u}) \stackrel{\text{def}}{=} (\forall\, \overline{\mathtt{u}}\ \mathtt{Lindex}(\overline{\mathtt{u}}) \Rightarrow (\sharp\overline{\mathtt{u}} \leq \sharp\mathtt{u})) \wedge \mathtt{Lindex}(\mathtt{u}).$$

Below, we state how the parentheses are organised.

Definition 3.4.5. Let $X = \{i_0, \ldots, i_s\} \subseteq [0, K]$ with $0 = i_0 < i_1 < \cdots < i_s$. A heap $\mathfrak{h}$ is X**-almost-well-formed** $\stackrel{\text{def}}{\Leftrightarrow}$

1. For every $j \in [0, s]$, there is a unique location $\mathfrak{l}^l_j$ [resp. $\mathfrak{l}^r_j$] such that $\mathfrak{h} \models_{[\mathtt{u}\mapsto\mathfrak{l}^l_j]} \mathtt{llp}_{i_j}(\mathtt{u})$ [resp. $\mathfrak{h} \models_{[\mathtt{u}\mapsto\mathfrak{l}^r_j]} \mathtt{lrp}_{i_j}(\mathtt{u})$].
2. For every $j \in [0, s]$, $\widetilde{\sharp\mathfrak{l}^l_j} < \widetilde{\sharp\mathfrak{l}^r_j}$, and $\widetilde{\sharp\mathfrak{l}^r_0} = \widetilde{\sharp\mathfrak{l}^l_0} + 1$.
3. For every $j \in [1, s]$, we have $\widetilde{\sharp\mathfrak{l}^l_j} = \widetilde{\sharp\mathfrak{l}^r_{j-1}} + 1$.
4. $\mathfrak{h} \models_{[\mathtt{u}\mapsto\mathfrak{l}^r_s]} \mathtt{maxLindex}(\mathtt{u})$.
5. For every $j \in [1, s]$, if i_j is the index of a first-order variable, then $\widetilde{\sharp\mathfrak{l}^l_j} = \widetilde{\sharp\mathfrak{l}^r_j} - 2$ (see Figure 3.6).
6. For every $j \in ([1, K] \setminus X)$, there is no location $\mathfrak{l}$ such that $\mathfrak{h} \models_{[\mathtt{u}\mapsto\mathfrak{l}]} \mathtt{llp}_j(\mathtt{u}) \vee \mathtt{lrp}_j(\mathtt{u})$. ∇

The definition for X-almost-well-formed heaps mainly specifies the existence of j-parentheses with $j \in X$ and how their respective degrees are related. The degrees are organised as follows and they all belong to the spectrum of $\mathfrak{h}$ (below we let $d^l_j = \widetilde{\sharp \mathfrak{l}^l_j}$ and $d^r_j = \widetilde{\sharp \mathfrak{l}^r_j}$).

$$\begin{array}{ccccccccccccccc}
\models \mathtt{indmin}(\mathtt{u}) & & & & & & & & & & & & & & \models \mathtt{maxLindex}(\mathtt{u}) \\
d^l_0 & < & d^r_0 & < & d^l_1 & < & d^r_1 & < & d^l_2 & < & d^r_2 & < \cdots < & d^l_s & < & d^r_s \\
 & & \| & & \| & & & & \| & & & & \| & & \\
 & & d^l_0 + 1 & & d^r_0 + 1 & & & & d^r_1 + 1 & & & & d^r_{s-1} + 1 & &
\end{array}$$

Moreover, when i_j is the index of a first-order variable, we have $d^r_j = d^l_j + 2$.

Lemma 3.4.6. There exists a formula $\mathtt{awfh}_X$ in 1SL2($-\!\!*$) such that $\mathfrak{h} \models \mathtt{awfh}_X$ iff $\mathfrak{h}$ is X-almost-well-formed.

The proof is left as Exercise 3.1.

Let $\mathfrak{h}$ be an X-almost-well-formed heap for some $\{0\} \subseteq X \subseteq [0, K]$ and $i \in X$. We write $\mathtt{vind}_i(\mathtt{u})$ to denote

$$\mathtt{Lindex}(\mathtt{u}) \wedge \mathtt{eindex}(\mathtt{u}) \wedge (\exists\, \overline{\mathtt{u}}\ \mathtt{llp}_i(\overline{\mathtt{u}}) \wedge \sharp\overline{\mathtt{u}} < \sharp\mathtt{u}) \wedge (\exists\, \overline{\mathtt{u}}\ \mathtt{lrp}_i(\overline{\mathtt{u}}) \wedge \sharp\overline{\mathtt{u}} > \sharp\mathtt{u})$$

It characterises indices whose degree is strictly between the degree of some large left i-parenthesis and the degree of some large right i-parenthesis. We write $\mathit{degrees}(i, \mathfrak{h})$ to denote the set:

$$\mathit{degrees}(i, \mathfrak{h}) \stackrel{\text{def}}{=} \{\widetilde{\sharp \mathfrak{l}} \in \mathbb{N} : \mathfrak{h} \models_{[\mathtt{u} \mapsto \mathfrak{l}]} \mathtt{vind}_i(\mathtt{u}),\ \mathfrak{l} \in \mathbb{N}\}.$$

Lemma 3.4.7. Let $\mathfrak{h}$ be a heap such that $\mathfrak{h} \models \exists\, \mathtt{u}\ \mathtt{indmin}(\mathtt{u})$ and $i \geq 0$ be such that there are unique locations $\mathfrak{lp}$ and $\mathfrak{rp}$ with $\mathfrak{h} \models_{[\mathtt{u} \mapsto \mathfrak{lp}]} \mathtt{llp}_i(\mathtt{u})$ and $\mathfrak{h} \models_{[\mathtt{u} \mapsto \mathfrak{rp}]} \mathtt{lrp}_i(\mathtt{u})$. For every $\mathfrak{l} \in \mathbb{N}$, we have $\mathfrak{h} \models_{[\mathtt{u} \mapsto \mathfrak{l}]} \mathtt{vind}_i(\mathtt{u})$ iff $\mathfrak{l}$ is the index of some entry and $\widetilde{\sharp \mathfrak{lp}} < \widetilde{\sharp \mathfrak{l}} < \widetilde{\sharp \mathfrak{rp}}$.

The proof of Lemma 3.4.7 is left as Exercise 3.2.

The formula $\mathtt{elt}_j(\mathtt{u})$ defined below holds true when $\mathtt{u}$ is interpreted as the element of the unique entry attached to the first-order variable $\mathtt{u}_j$.

$$\mathtt{elt}_j(\mathtt{u}) \stackrel{\text{def}}{=} \exists\, \overline{\mathtt{u}}\ (\overline{\mathtt{u}} \hookrightarrow \mathtt{u}) \wedge \mathtt{vind}_j(\overline{\mathtt{u}})$$

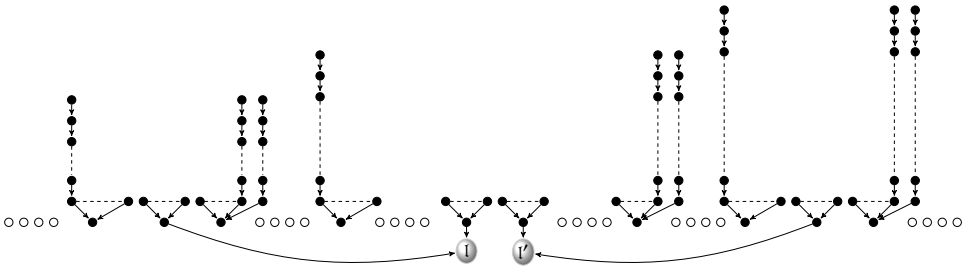


Figure 3.7: How the translation of $\mathtt{P}_i(\mathtt{u}_j, \mathtt{u}_k)$ works ($j < i < k$): $(\mathfrak{l}, \mathfrak{l}') \in \mathfrak{V}_{\mathfrak{h}}(\mathtt{P}_i)$.

Then, the translation of $\mathtt{P}_i(\mathtt{u}_j, \mathtt{u}_k)$ can be designed as follows:

$$\exists\, \mathtt{u}\, (\mathtt{elt}_j(\mathtt{u}) \;\wedge\; \exists\, \overline{\mathtt{u}}\, (\overline{\mathtt{u}} \hookrightarrow \mathtt{u} \;\wedge\; \mathtt{vind}_i(\overline{\mathtt{u}}) \wedge$$

$$\exists\, \mathtt{u}\, (\sharp\mathtt{u} = \sharp\overline{\mathtt{u}} + 1 \;\wedge\; \mathtt{vind}_i(\mathtt{u}) \;\wedge\; \exists\, \overline{\mathtt{u}}\, (\mathtt{u} \hookrightarrow \overline{\mathtt{u}} \;\wedge\; \mathtt{elt}_k(\overline{\mathtt{u}}))))).$$

These definitions take advantage of the fact that there are unique large left and right parentheses for each variable index. Figure 3.7 illustrates the constraints satisfied by the formula when $j < i < k$. From left to right, the figure represents explicitly a left j-parenthesis, then a right j-parenthesis, then a left i-parenthesis, a right i-parenthesis and a left k-parenthesis, followed finally by a right k-parenthesis. Other entries and parentheses are present in the figure, but they are represented by dots in order to focus on the memory cells relevant to evaluate the formula obtained by translation of $\mathtt{P}_i(\mathtt{u}_j, \mathtt{u}_k)$. The degrees of parentheses and entries increase from left to right.

3.4.3 Taking care of valuations

Now that we have a way of identifying that part of the heap that encodes our valuation, we turn our attention to encoding the valuation itself. Below, we introduce a condition for a subheap to be "glued" to an existing valuation. We distinguish three cases.

- A **local** 0**-valuation** is a heap made of a left 0-parenthesis of degree d and a right 0-parenthesis of degree $d + 1$ only, for some $d \geq 3$.
- Let $i \in [1, K]$ be the index of some first-order variable. A **local** i**-valuation** is a heap made of a left i-parenthesis of degree d, an entry of degree $d + 1$ and a right i-parenthesis of degree $d + 2$ only, for some $d \geq 3$.

- Let $i \in [1, K]$ be the index of some second-order variable. A **local i-valuation** is a heap $\mathfrak{h}$ such that

  1. every location $\mathfrak{l}$ in $\mathrm{dom}(\mathfrak{h})$ belongs either to a left i-parenthesis, to a right i-parenthesis, or to an entry,
  2. $\mathfrak{h}$ contains a unique left [resp. right] i-parenthesis,
  3. $min(indspect(\mathfrak{h}))$ is the degree of some left i-parenthesis,
  4. $max(indspect(\mathfrak{h}))$ is the degree of some right i-parenthesis,
  5. $indspect(\mathfrak{h})$ is of the form below for some $\alpha \geq 3$, $\beta \geq 0$,

     $$\{\alpha\} \cup \{\alpha + 3(i-1) + 1, \alpha + 3(i-1) + 2 : i \in [1, \beta]\} \cup \{\alpha + 3\beta + 1\}$$

     (when $\beta = 0$, $indspect(\mathfrak{h})$ is equal to $\{\alpha, \alpha + 1\}$),
  6. there are no two distinct indices with the same degree.

Since local i-valuations are typically heaps that are added to the current heap to encode the interpretation of a variable, it is essential to be able to characterise them by 1SL2($\mathrel{-\!\!*}$) formulae. This is the purpose of the result below.

Lemma 3.4.8. Let $i \in [0, K]$. There is a formula $\mathtt{localval}_i(\mathtt{u})$ in 1SL2($\mathrel{-\!\!*}$) such that $\mathfrak{h} \models_{\mathfrak{f}} \mathtt{localval}_i(\mathtt{u})$ iff $\mathfrak{h}$ is a local i-valuation and $\mathfrak{f}(\mathtt{u})$ is the index of its left i-parenthesis.

Proof. For characterising local 0-valuations, it is sufficient to express the properties below:

1. any location in the domain is on some left or on some right 0-parenthesis,
2. there is exactly one left 0-parenthesis whose index is $\mathfrak{f}(\mathtt{u})$,
3. there is exactly one right 0-parenthesis,
4. $\widetilde{\sharp \mathfrak{l}} = \widetilde{\sharp \mathfrak{f}(\mathtt{u})} + 1$ where $\mathfrak{l}$ is the index of the unique right 0-parenthesis.

(1)-(4) can be expressed by the formula below:

$$(\forall\, \mathtt{u}\ \mathtt{alloc}(\mathtt{u}) \Rightarrow \mathtt{onpar}_0(\mathtt{u})) \wedge (\mathtt{lp}_0(\mathtt{u}) \wedge \neg(\exists\, \overline{\mathtt{u}}\ \mathtt{lp}_0(\overline{\mathtt{u}}) \wedge \mathtt{u} \neq \overline{\mathtt{u}})) \wedge$$
$$(\exists\, \overline{\mathtt{u}}\ (\mathtt{rp}_0(\overline{\mathtt{u}}) \wedge \neg(\exists\, \mathtt{u}\ \mathtt{rp}_0(\mathtt{u}) \wedge \mathtt{u} \neq \overline{\mathtt{u}})) \wedge (\sharp\overline{\mathtt{u}} = \sharp\mathtt{u} + 1))$$

Formulae of the form $\mathtt{lp}_i(\mathtt{u})$ and $\mathtt{rp}_i(\mathtt{u})$ are provided in the proof of Lemma 3.4.1 whereas formulae of the form $\mathtt{onpar}_i(\mathtt{u})$ are provided before Lemma 3.4.2.

For characterising local i-valuations for some first-order variable $\mathtt{u}_i$, it is sufficient to express the properties below:

1. any location in the domain is on some left i-parenthesis, or on some right i-parenthesis or on some entry,
2. there is exactly one left i-parenthesis whose index is $\mathtt{f}(\mathtt{u})$,
3. there is exactly one right i-parenthesis,
4. there is a unique entry, whose degree is d, such that $\widetilde{\sharp \mathtt{l}} = \widetilde{\sharp \mathtt{f}(\mathtt{u})} + 2$ and $\widetilde{\sharp \mathtt{f}(\mathtt{u})} = d - 1$ where $\mathtt{l}$ is the index of the unique right i-parenthesis.

(1)-(4) can be expressed by the formula below:

$$(\forall\, \mathtt{u}\ \mathtt{alloc}(\mathtt{u}) \Rightarrow \mathtt{onpar}_i(\mathtt{u}) \vee \mathtt{epin}(\mathtt{u}) \vee \mathtt{eindex}(\mathtt{u})) \wedge$$
$$(\mathtt{lp}_i(\mathtt{u}) \wedge \neg(\exists\, \overline{\mathtt{u}}\ \mathtt{lp}_i(\overline{\mathtt{u}}) \wedge \mathtt{u} \neq \overline{\mathtt{u}})) \wedge (\exists\, \overline{\mathtt{u}}\ (\mathtt{rp}_i(\overline{\mathtt{u}}) \wedge \neg(\exists\, \mathtt{u}\ \mathtt{rp}_i(\mathtt{u}) \wedge \mathtt{u} \neq \overline{\mathtt{u}}) \wedge (\sharp\overline{\mathtt{u}} = \sharp\mathtt{u}+2)) \wedge$$
$$(\exists\, \overline{\mathtt{u}}\ \mathtt{eindex}(\overline{\mathtt{u}}) \wedge (\neg\exists \mathtt{u}\ (\mathtt{u} \neq \overline{\mathtt{u}}) \wedge \mathtt{eindex}(\mathtt{u})) \wedge (\sharp\overline{\mathtt{u}} = \sharp\mathtt{u} + 1))$$

For characterising local i-valuations for some second-order variable $\mathtt{P}_i$, it is sufficient to express the properties below:

1. any location in the domain is on some left i-parenthesis, or on some right i-parenthesis or on some entry,
2. there is exactly one left i-parenthesis whose index is $\mathtt{f}(\mathtt{u})$,
3. there is exactly one right i-parenthesis whose index is the location $\mathtt{l}$,
4. any entry has degree in $[\widetilde{\sharp \mathtt{f}(\mathtt{u})} + 1, \widetilde{\sharp \mathtt{l}} - 1]$ and its index is the unique one with that degree,
5. $\widetilde{\sharp \mathtt{l}} > \widetilde{\sharp \mathtt{f}(\mathtt{u})}$,
6. if $\widetilde{\sharp \mathtt{l}} > \widetilde{\sharp \mathtt{f}(\mathtt{u})} + 1$, then
   1. there is an entry with degree $\widetilde{\sharp \mathtt{f}(\mathtt{u})} + 1$,
   2. there is an entry with degree $\widetilde{\sharp \mathtt{f}(\mathtt{u})} + 2$,
   3. if there are entries with respective degree d and $d+1$, then there is no entry or right i-parenthesis of degree $d + 2$,
   4. if there are entries with respective degree d and $d + 1$ and $d + 3 < \widetilde{\sharp \mathtt{l}}$, then there are entries of respective degree $d + 3$ and $d + 4$.

(1)–(5) can be expressed by the formula below:

$$(\forall\, \mathtt{u}\ \mathtt{alloc}(\mathtt{u}) \Rightarrow \mathtt{onpar}_i(\mathtt{u}) \vee \mathtt{epin}(\mathtt{u}) \vee \mathtt{eindex}(\mathtt{u})) \wedge$$

$$(\mathtt{lp}_i(\mathtt{u}) \wedge \neg(\exists\, \overline{\mathtt{u}}\ \mathtt{lp}_i(\overline{\mathtt{u}}) \wedge \mathtt{u} \neq \overline{\mathtt{u}})) \wedge (\exists\, \overline{\mathtt{u}}\ (\mathtt{rp}_i(\overline{\mathtt{u}}) \wedge \neg(\exists\, \mathtt{u}\ \mathtt{rp}_i(\mathtt{u}) \wedge \mathtt{u} \neq \overline{\mathtt{u}}) \wedge \sharp\overline{\mathtt{u}} > \sharp\mathtt{u}))$$

$$\forall\, \mathtt{u}\ \mathtt{eindex}(\mathtt{u}) \Rightarrow \neg(\exists\, \overline{\mathtt{u}}\ ((\mathtt{eindex}(\overline{\mathtt{u}}) \vee \mathtt{lp}_i(\overline{\mathtt{u}}) \vee \mathtt{rp}_i(\overline{\mathtt{u}})) \wedge \sharp\mathtt{u} = \sharp\overline{\mathtt{u}})$$

$$(\exists\, \overline{\mathtt{u}}\ \mathtt{rp}_i(\overline{\mathtt{u}}) \wedge (\forall\, \mathtt{u}\ \mathtt{eindex}(\mathtt{u}) \Rightarrow \sharp\mathtt{u} < \sharp\overline{\mathtt{u}} - 1)) \wedge (\forall\, \overline{\mathtt{u}}\ \mathtt{eindex}(\overline{\mathtt{u}}) \Rightarrow \sharp\overline{\mathtt{u}} > \sharp\mathtt{u})$$

(6a)–(6d) can be expressed by the formula below:

$$(\exists\, \overline{\mathtt{u}}\ \mathtt{rp}_i(\overline{\mathtt{u}}) \wedge \sharp\overline{\mathtt{u}} > \sharp\mathtt{u} + 1) \Rightarrow$$

$$(\exists\, \overline{\mathtt{u}}\ \mathtt{eindex}(\overline{\mathtt{u}}) \wedge \sharp\overline{\mathtt{u}} = \sharp\mathtt{u} + 1) \wedge (\exists\, \overline{\mathtt{u}}\ \mathtt{eindex}(\overline{\mathtt{u}}) \wedge \sharp\overline{\mathtt{u}} = \sharp\mathtt{u} + 2) \wedge$$

$$\forall\, \mathtt{u}\ (\mathtt{eindex}(\mathtt{u}) \wedge (\exists\, \overline{\mathtt{u}}\ \mathtt{eindex}(\overline{\mathtt{u}}) \wedge \sharp\overline{\mathtt{u}} = \sharp\mathtt{u}+1)) \Rightarrow \neg(\exists\, \overline{\mathtt{u}}\ (\mathtt{eindex}(\overline{\mathtt{u}}) \vee \mathtt{rp}_i(\overline{\mathtt{u}})) \wedge \sharp\overline{\mathtt{u}} = \sharp\mathtt{u}+2) \wedge$$

$$\forall\, \mathtt{u}\ (\mathtt{eindex}(\mathtt{u}) \wedge (\exists\, \overline{\mathtt{u}}\ \mathtt{eindex}(\overline{\mathtt{u}}) \wedge \sharp\overline{\mathtt{u}} = \sharp\mathtt{u} + 1) \wedge (\exists\, \overline{\mathtt{u}}\ \mathtt{rp}_i(\overline{\mathtt{u}}) \wedge \sharp\overline{\mathtt{u}} > \sharp\mathtt{u} + 3)) \Rightarrow$$

$$((\exists\, \overline{\mathtt{u}}\ \mathtt{eindex}(\overline{\mathtt{u}}) \wedge \sharp\overline{\mathtt{u}} = \sharp\mathtt{u} + 3) \wedge (\exists\, \overline{\mathtt{u}}\ \mathtt{eindex}(\overline{\mathtt{u}}) \wedge \sharp\overline{\mathtt{u}} = \sharp\mathtt{u} + 4))$$

QED

The definition for X-almost-well-formed heaps mainly takes care of parentheses. In Definition 3.4.9, constraints on the degrees of large indices are specified.

Definition 3.4.9. Let $X = \{i_0, \ldots, i_s\} \subseteq [0, K]$ with $0 = i_0 < i_1 < \cdots < i_s$. A heap $\mathfrak{h}$ is **X-well-formed** $\overset{\text{def}}{\Leftrightarrow}$ the following conditions hold:

1. $\mathfrak{h}$ is X-almost-well-formed,
2. for every $j \in [1, s]$, if i_j is the index of a first-order variable, then $\mathit{degrees}(i_j, \mathfrak{h})$ is a singleton,
3. for every $j \in [1, s]$, if i_j is the index of a second-order variable, then $\mathit{degrees}(i_j, \mathfrak{h})$ is the set below for some $\alpha_j \geq 3$, $\beta_j \geq 0$:

$$\{\alpha_j + 3(i-1) + 1, \alpha_j + 3(i-1) + 2 : i \in [1, \beta_j]\},$$

4. for every location $\mathfrak{l}$ such that $\mathfrak{h} \models_{[\mathtt{u} \mapsto \mathfrak{l}]} \mathtt{Lindex}(\mathtt{u})$, there is no $\mathfrak{l}' \neq \mathfrak{l}$ such that $\mathfrak{h} \models_{[\mathtt{u} \mapsto \mathfrak{l}']} \mathtt{Lindex}(\mathtt{u})$ and $\widetilde{\sharp}\mathfrak{l} = \widetilde{\sharp}\mathfrak{l}'$.
5. If a location has degree greater than the degree of the unique large left 0-parenthesis, then it is a large index.

When $\mathfrak{h}$ is X-well-formed, we write $\mathfrak{h} = \mathfrak{h}_B \uplus \mathfrak{h}_V$ such that $\mathrm{dom}(\mathfrak{h}_V)$ is made of entries and parentheses of degree $d \geq \sharp\mathfrak{l}_0$ for some $\mathfrak{l}_0 \in \mathbb{N}$ such that $\mathfrak{h} \models_{[\mathfrak{u}\mapsto\mathfrak{l}_0]} \mathtt{indmin}(\mathfrak{u})$ (i.e., $\mathfrak{l}_0$ is the index of the left 0-parenthesis with the maximal degree). By Definition 3.4.9, we have $spect(\mathfrak{h}) = indspect(\mathfrak{h}_V)$ and clearly the decomposition is unique since $\mathfrak{l}_0$ is unique.

Again, well-formed heaps can be characterised by formulae in 1SL2($\mathrel{-\!\!*}$) whose size is linear in K.

Lemma 3.4.10. Given $\{0\} \subseteq X \subseteq [0, K]$, there is a formula $\mathtt{wfh}_X$ in 1SL2($\mathrel{-\!\!*}$) such that $\mathfrak{h} \models \mathtt{wfh}_X$ iff $\mathfrak{h}$ is X-well-formed.

Proof. We consider the conjunction of the formulae below, each of them deals with one of the four conditions. Condition (1) is obviously taken care by the formula $\mathtt{awfh}_X$ (see the proof of Lemma 3.4.6). Condition (2) is dealt with the formula below:

$$\bigwedge_{j\in[1,s],\ \mathrm{FO}\ i_j} \exists\, \mathtt{u}\ \mathtt{vind}_{i_j}(\mathtt{u})$$

Note that since the heap is already X-almost-well-formed, at most one location can satisfy the above existential quantification for each FO variable index i_j. Similarly, Condition (3) is taken care by the formula below (see the proof of Lemma 3.4.8 and more specifically Condition (6) in that proof with indices from second-order variables):

$$\bigwedge_{j\in[1,s],\ \mathrm{SO}\ i_j} (\exists\, \mathtt{u}\ \exists\, \overline{\mathtt{u}}\ \mathtt{llp}_{i_j}(\mathtt{u}) \wedge \mathtt{lrp}_{i_j}(\overline{\mathtt{u}}) \wedge (\sharp\overline{\mathtt{u}} > \sharp\mathtt{u} + 1)) \Rightarrow$$

$$\overbrace{[(\exists\, \mathtt{u}\ \exists\, \overline{\mathtt{u}}\ \mathtt{llp}_{i_j}(\mathtt{u}) \wedge \mathtt{vind}_{i_j}(\overline{\mathtt{u}}) \wedge \sharp\overline{\mathtt{u}} = \sharp\mathtt{u} + 1)}^{(a)} \wedge$$

$$\overbrace{(\exists\, \mathtt{u}\ \exists\, \overline{\mathtt{u}}\ \mathtt{llp}_{i_j}(\mathtt{u}) \wedge \mathtt{vind}_{i_j}(\overline{\mathtt{u}}) \wedge \sharp\overline{\mathtt{u}} = \sharp\mathtt{u} + 2)}^{(b)} \wedge$$

$$\overbrace{\forall\, \mathtt{u}\ (\mathtt{vind}_{i_j}(\mathtt{u}) \wedge (\exists\, \overline{\mathtt{u}}\ \mathtt{vind}_{i_j}(\overline{\mathtt{u}}) \wedge \sharp\overline{\mathtt{u}} = \sharp\mathtt{u} + 1))\ \Rightarrow}^{(c)}$$

$$\overbrace{\neg(\exists\, \overline{\mathtt{u}}\ (\mathtt{vind}_{i_j}(\overline{\mathtt{u}}) \vee \mathtt{rp}_{i_j}(\overline{\mathtt{u}})) \wedge \sharp\overline{\mathtt{u}} = \sharp\mathtt{u} + 2)}^{(c)} \wedge$$

$$\overbrace{\forall\, \mathtt{u}\, (\mathtt{vind}_{i_j}(\mathtt{u}) \wedge (\exists\, \overline{\mathtt{u}}\, \mathtt{vind}_{i_j}(\overline{\mathtt{u}}) \wedge \sharp\overline{\mathtt{u}} = \sharp\mathtt{u} + 1) \wedge (\exists\, \overline{\mathtt{u}}\, \mathtt{lrp}_{i_j}(\overline{\mathtt{u}}) \wedge \sharp\overline{\mathtt{u}} > \sharp\mathtt{u} + 3)) \Rightarrow}^{\text{(d)}}$$
$$((\exists\, \overline{\mathtt{u}}\, \mathtt{vind}_{i_j}(\overline{\mathtt{u}}) \wedge \sharp\overline{\mathtt{u}} = \sharp\mathtt{u} + 3) \wedge (\exists\, \overline{\mathtt{u}}\, \mathtt{vind}_{i_j}(\overline{\mathtt{u}}) \wedge \sharp\overline{\mathtt{u}} = \sharp\mathtt{u} + 4))].$$

The above formula expresses the conditions below, mimicking Condition (6) from the proof of Lemma 3.4.8:

(a) there is an entry with degree $d^l + 1$ where d^l is the degree of the unique left i_j-parenthesis,

(b) there is an entry with degree $d^l + 2$,

(c) if there are entries with respective degree d and $d + 1$ in $\mathit{degrees}(i_j, \mathfrak{h})$, then there is no entry or right i_j-parenthesis of degree $d + 2$ in $\mathit{degrees}(i_j, \mathfrak{h})$,

(d) if there are entries with respective degree d and $d + 1$ in $\mathit{degrees}(i_j, \mathfrak{h})$ and $d + 3 < d^r$ where d^r is the degree of the unique right i_j-parenthesis, then there are entries of respective degree $d + 3$ and $d + 4$ in $\mathit{degrees}(i_j, \mathfrak{h})$. **QED**

Condition (4) is expressed as follows by simply internalising the condition in 1SL2(⇁):

$$\forall\, \mathtt{u}\, \mathtt{Lindex}(\mathtt{u}) \Rightarrow \neg(\exists\, \overline{\mathtt{u}}\, \mathtt{Lindex}(\overline{\mathtt{u}}) \wedge (\mathtt{u} \neq \overline{\mathtt{u}}) \wedge \sharp\overline{\mathtt{u}} = \sharp\mathtt{u}).$$

Condition (5) can be expressed as follows:

$$\forall\, \mathtt{u}\, (\exists\, \overline{\mathtt{u}}\, \mathtt{indmin}(\overline{\mathtt{u}}) \wedge \sharp\mathtt{u} \geq \sharp\overline{\mathtt{u}}) \Rightarrow \mathtt{Lindex}_X(\mathtt{u}).$$

Let us define formally a valuation from a valuation heap.

Definition 3.4.11. Let $\mathfrak{h}$ be an X-well-formed heap for some $\{0\} \subseteq X \subseteq [0, K]$.

- For every second-order $i \in X$, we define

$$\mathfrak{V}_{\mathfrak{h}}(\mathtt{P}_i) \stackrel{\text{def}}{=} \{(\mathfrak{h}_V(\mathtt{l}), \mathfrak{h}_V(\mathtt{l}')) : \widetilde{\sharp\mathtt{l}'} = \widetilde{\sharp\mathtt{l}}+1,\ \widetilde{\sharp\mathtt{l}}, \widetilde{\sharp\mathtt{l}'} \in \mathit{degrees}(i, \mathfrak{h}), \mathtt{l}, \mathtt{l}' \text{ are index locations}\}$$

- For every first-order $i \in X$, $\mathfrak{V}_{\mathfrak{h}}(\mathtt{u}_i) \stackrel{\text{def}}{=} \mathfrak{h}_V(\mathtt{l})$ where $\mathtt{l}$ is the unique index location such that $\widetilde{\sharp\mathtt{l}} \in \mathit{degrees}(i, \mathfrak{h})$.

We say that $\mathfrak{V}_{\mathfrak{h}}$ is the valuation **extracted** from $\mathfrak{h}$. ∇

Below, we present an essential technical result stating how heaps can be composed when a new variable needs to be interpreted. The formulae involved to compose the X-well-formed heap $\mathfrak{h}$ and the local i-valuation heap $\mathfrak{h}'$ are directly used in the translation of quantified formulae (see Section 3.4.4). Lemma 3.4.12 is used in the proof of Lemma 3.4.13.

Lemma 3.4.12 (Composition). Let $\dagger$ be a valuation, $\mathfrak{h}$ be an X-well-formed heap with $\{0\} \subseteq X \subseteq [0, K]$, $i \in [1, K] \setminus X$ with $i > max(X)$, and $\mathfrak{h}'$ be a disjoint heap such that:

(I) $\mathfrak{h} \models_\dagger \mathtt{indmin}(\mathtt{u}) \wedge \mathtt{isoloc}(\overline{\mathtt{u}})$,

(II) $\mathfrak{h}' \models_\dagger \mathtt{localval}_i(\overline{\mathtt{u}})$,

(III) $\mathfrak{h} \uplus \mathfrak{h}' \models_\dagger \mathtt{wfh}_{X\cup\{i\}} \wedge \mathtt{indmin}(\mathtt{u}) \wedge \mathtt{llp}_i(\overline{\mathtt{u}})$.

Then, $spect(\mathfrak{h} \uplus \mathfrak{h}') = spect(\mathfrak{h}) \uplus indspect(\mathfrak{h}')$.

Roughly speaking, Lemma 3.4.12 states that given an X-well-formed heap $\mathfrak{h}$, adding a disjoint local i-valuation $\mathfrak{h}'$ with $i \notin X$, leads to an $(X \cup \{i\})$-well-formed heap so that the interpretation of variables with variable indices in X from the extracted valuation, is the same with $\mathfrak{h}$ and with $\mathfrak{h} \uplus \mathfrak{h}'$. The heap $\mathfrak{h}'$ can be then understood as a *conservative* extension of the heap $\mathfrak{h}$.

The proof of Lemma 3.4.12 is quite combinatorial and this is the place where we check that the original heap cannot be confused with the valuation heap (and the other way around). It is important to guarantee, as the proof does, that adding a new part of the valuation does not destroy what has been built so far.

3.4.4 A reduction from DSOL into 1SL2(−∗)

Below, we define a translation from a sentence φ in 1DSOL into a sentence in 1SL2(−∗) that uses only logarithmic space. Without any loss of generality, we assume that

1. two occurrences of quantified variables in φ have distinct variable indices (e.g., $\mathtt{P}_4$ and $\mathtt{u}_4$ cannot both occur in φ and "$\forall\, \mathtt{u}_4$" cannot occur more than once) and

2. if $\exists\, \mathtt{u}_i\ \psi_1$ is a subformula of $\exists\, \mathtt{u}_j\ \psi_2$, then $i > j$ and this holds for any combination of first-order/second-order variables.

At the outset, we may rename variables so that these simple conditions are satisfied. We assume that the variable indices for (first-order or second-order) variables are among $[1, K]$.

The translation of the formula φ, written $\mathtt{T}(\varphi)$, first applies a top-level translation $t_{top}(\cdot)$ which takes care of initialising the valuation heap; then, a recursive map $t(\cdot)$ is applied. So, $\mathtt{T}(\varphi) \stackrel{\text{def}}{=} t_{top}(\varphi)$ where $t_{top}(\varphi)$ is defined as follows:

$$t_{top}(\varphi) \stackrel{\text{def}}{=} \exists\, \mathtt{u}\ \mathtt{isoloc}(\mathtt{u}) \wedge (\mathtt{localval}_0(\mathtt{u}) \mathrel{-\!\!\!\!\!\rightarrow\!\!\!\!\!*} (\mathtt{wfh}_{\{0\}} \wedge$$

$$\mathtt{indmin}(\mathtt{u}) \wedge (\forall\, \overline{\mathtt{u}}\ ((\mathtt{u} \neq \overline{\mathtt{u}}) \wedge \neg \mathtt{lrp}_0(\overline{\mathtt{u}})) \Rightarrow (\sharp\overline{\mathtt{u}} < \sharp\mathtt{u})) \wedge t(\{0\}, \varphi)))$$

The first step of the translation consists in adding 0-parentheses so that the heap that evaluates $t(\{0\}, \varphi)$ is $\{0\}$-well-formed. The translation map $t(\cdot)$ has two arguments: the formula to be transformed and the set of variable indices for variables that have been quantified so far. The map $t(\cdot)$ is inductively defined as follows ($X \subseteq [0, K]$, ψ subformula of φ) and it is homomorphic for Boolean connectives:

$$\begin{array}{lcl}
t\big(X, \mathtt{u}_i = \mathtt{u}_j\big) & \stackrel{\text{def}}{=} & \exists\, \mathtt{u}\ \mathtt{elt}_i(\mathtt{u}) \wedge \mathtt{elt}_j(\mathtt{u}) \\
t\big(X, \mathtt{u}_i \hookrightarrow \mathtt{u}_j\big) & \stackrel{\text{def}}{=} & \exists\, \mathtt{u}\, \exists\, \overline{\mathtt{u}}\ (\mathtt{elt}_i(\mathtt{u}) \wedge \mathtt{elt}_j(\overline{\mathtt{u}}) \wedge \mathtt{u} \hookrightarrow \overline{\mathtt{u}}) \\
t\big(X, \mathtt{P}_i(\mathtt{u}_j, \mathtt{u}_k)\big) & \stackrel{\text{def}}{=} & \exists\, \mathtt{u}\ (\mathtt{elt}_j(\mathtt{u}) \wedge \exists\, \overline{\mathtt{u}}\ (\overline{\mathtt{u}} \hookrightarrow \mathtt{u} \wedge \mathtt{vind}_i(\overline{\mathtt{u}}) \wedge \\
 & & \exists\, \mathtt{u}\ (\sharp\mathtt{u} = \sharp\overline{\mathtt{u}} + 1 \wedge \mathtt{vind}_i(\mathtt{u}) \wedge \exists\, \overline{\mathtt{u}}\ (\mathtt{u} \hookrightarrow \overline{\mathtt{u}} \wedge \mathtt{elt}_k(\overline{\mathtt{u}}))))) \\
t(X, \exists\, \mathtt{u}_i\ \psi) & \stackrel{\text{def}}{=} & \exists\, \mathtt{u}\, \exists\, \overline{\mathtt{u}}\ ((\mathtt{indmin}(\mathtt{u}) \wedge \mathtt{isoloc}(\overline{\mathtt{u}})) \wedge (\mathtt{localval}_i(\overline{\mathtt{u}}) \mathrel{-\!\!\!\!\!\rightarrow\!\!\!\!\!*} \\
 & & (\mathtt{wfh}_{X \cup \{i\}} \wedge \mathtt{indmin}(\mathtt{u}) \wedge \mathtt{llp}_i(\overline{\mathtt{u}}) \wedge t(X \cup \{i\}, \psi)))) \\
t(X, \exists\, \mathtt{P}_i\ \psi) & \stackrel{\text{def}}{=} & \exists\, \mathtt{u}\, \exists\, \overline{\mathtt{u}}\ ((\mathtt{indmin}(\mathtt{u}) \wedge \mathtt{isoloc}(\overline{\mathtt{u}})) \wedge (\mathtt{localval}_i(\overline{\mathtt{u}}) \mathrel{-\!\!\!\!\!\rightarrow\!\!\!\!\!*} \\
 & & (\mathtt{wfh}_{X \cup \{i\}} \wedge \mathtt{indmin}(\mathtt{u}) \wedge \mathtt{llp}_i(\overline{\mathtt{u}}) \wedge t(X \cup \{i\}, \psi)))).
\end{array}$$

Every subformula $t(X, \psi)$ has no free variable from $\mathtt{free}(\psi) \subseteq X$ where $\mathtt{free}(\psi)$ denotes the set of variable indices in ψ from either first-order or second-order *free* variables.

Below, we state the correctness lemma that allows us to get Theorem 3.4.14 (the proof is by structural induction).

Lemma 3.4.13 (Correctness). Let φ be a DSOL sentence of the above form, ψ be one of its subformulae and $(\mathtt{free}(\psi) \cup \{0\}) \subseteq X \subseteq [0, K]$. Let $\mathfrak{h} = \mathfrak{h}_B \uplus \mathfrak{h}_V$ be a X-well-formed heap and $\mathfrak{V}_{\mathfrak{h}}$ be the valuation extracted from $\mathfrak{h}$. Then, $\mathfrak{h}_B \models_{\mathfrak{V}_{\mathfrak{h}}} \psi$ iff $\mathfrak{h} \models t(X, \psi)$.

Proof. The proof is by structural induction.
Base case 1: ψ is equal to $\mathtt{u}_i = \mathtt{u}_j$.
Since $\mathfrak{h}$ is X-well-formed and $i, j \in X$, $\mathfrak{V}_{\mathfrak{h}}(\mathtt{u}_i)$ is equal to $\mathfrak{h}_V(\mathtt{l})$ where $\mathtt{l}$ is the unique index location such that $\widetilde{\sharp\mathtt{l}} \in \mathit{degrees}(i, \mathfrak{h})$. Similarly, $\mathfrak{V}_{\mathfrak{h}}(\mathtt{u}_j)$ is equal to $\mathfrak{h}_V(\mathtt{l}')$ where $\mathtt{l}'$ is the unique index location such that $\widetilde{\sharp\mathtt{l}'} \in \mathit{degrees}(j, \mathfrak{h})$. Uniqueness is a consequence of Definition 3.4.9(2).

Let us recall that $\mathtt{elt}_i(\mathtt{u}) = \exists\, \overline{\mathtt{u}}\ (\overline{\mathtt{u}} \hookrightarrow \mathtt{u}) \wedge \mathtt{vind}_i(\overline{\mathtt{u}})$ with $\mathtt{vind}_i(\mathtt{u}) = \mathtt{Lindex}(\mathtt{u}) \wedge \mathtt{eindex}(\mathtt{u}) \wedge (\exists\, \overline{\mathtt{u}}\ \mathtt{llp}_i(\overline{\mathtt{u}}) \wedge \sharp\overline{\mathtt{u}} < \sharp\mathtt{u}) \wedge (\exists\, \overline{\mathtt{u}}\ \mathtt{lrp}_i(\overline{\mathtt{u}}) \wedge \sharp\overline{\mathtt{u}} > \sharp\mathtt{u})$. Similarly, $\mathtt{elt}_j(\mathtt{u}) = \exists\, \overline{\mathtt{u}}\ (\overline{\mathtt{u}} \hookrightarrow \mathtt{u}) \wedge \mathtt{vind}_j(\overline{\mathtt{u}})$.

First, let us suppose that $\mathfrak{h}_B \models_{\mathfrak{V}_{\mathfrak{h}}} \mathtt{u}_i = \mathtt{u}_j$. This means that $\mathfrak{V}_{\mathfrak{h}}(\mathtt{u}_i)$ and $\mathfrak{V}_{\mathfrak{h}}(\mathtt{u}_j)$ are equal, say to $\mathtt{l}'$ and therefore there is a unique index location $\mathtt{l}_i$ such that $\mathfrak{h}_V(\mathtt{l}_i) = \mathtt{l}'$ and $\widetilde{\sharp\mathtt{l}_i} \in \mathit{degrees}(i, \mathfrak{h})$. Moreover, there is a unique index location $\mathtt{l}_j$ such that $\mathfrak{h}_V(\mathtt{l}_j) = \mathtt{l}'$ and $\widetilde{\sharp\mathtt{l}_j} \in \mathit{degrees}(j, \mathfrak{h})$. Since $\widetilde{\sharp\mathtt{l}_i}$ and $\widetilde{\sharp\mathtt{l}_j}$ belong to the index spectrum of $\mathfrak{h}_V$, the locations $\mathtt{l}_i$ and $\mathtt{l}_j$, have the same number of predecessors in $\mathfrak{h}$ and in $\mathfrak{h}_V$ and they are also indices in $\mathfrak{h}$ (see Lemma 3.4.12). Consequently, $\mathfrak{h} \models_{[\mathtt{u}\mapsto\mathtt{l}']} \mathtt{elt}_i(\mathtt{u}) \wedge \mathtt{elt}_j(\mathtt{u})$ (see also Lemma 3.4.7), whence $\mathfrak{h} \models \exists\, \mathtt{u}\ \mathtt{elt}_i(\mathtt{u}) \wedge \mathtt{elt}_j(\mathtt{u})$.

Now suppose that $\mathfrak{h} \models \exists\, \mathtt{u}\ \mathtt{elt}_i(\mathtt{u}) \wedge \mathtt{elt}_j(\mathtt{u})$. There is a location $\mathtt{l}'$ such that $\mathfrak{h} \models_{[\mathtt{u}\mapsto\mathtt{l}']} \mathtt{elt}_i(\mathtt{u}) \wedge \mathtt{elt}_j(\mathtt{u})$. Therefore there are index locations $\mathtt{l}_i$ and $\mathtt{l}_j$ such that $\widetilde{\sharp\mathtt{l}_i} \in \mathit{degrees}(i, \mathfrak{h})$, $\widetilde{\sharp\mathtt{l}_j} \in \mathit{degrees}(j, \mathfrak{h})$, $\mathfrak{h}(\mathtt{l}_i) = \mathtt{l}'$ and $\mathfrak{h}(\mathtt{l}_j) = \mathtt{l}'$. By Lemma 3.4.12, $\mathfrak{h}_V(\mathtt{l}_i) = \mathtt{l}'$, $\widetilde{\sharp\mathtt{l}_i} \in \mathit{indspect}(\mathfrak{h}_V)$, $\mathfrak{h}_V(\mathtt{l}_j) = \mathtt{l}'$ and $\widetilde{\sharp\mathtt{l}_j} \in \mathit{indspect}(\mathfrak{h}_V)$. By definition of $\mathfrak{V}_{\mathfrak{h}}$ (see Definition 3.4.11), this implies that $\mathfrak{V}_{\mathfrak{h}}(\mathtt{u}_i) = \mathfrak{V}_{\mathfrak{h}}(\mathtt{u}_j)$ and therefore $\mathfrak{h}_B \models_{\mathfrak{V}_{\mathfrak{h}}} \mathtt{u}_i = \mathtt{u}_j$.

Base case 2: ψ is equal to $\mathtt{u}_i \hookrightarrow \mathtt{u}_j$.
Again, since $\mathfrak{h}$ is X-well-formed and $i, j \in X$, $\mathfrak{V}_{\mathfrak{h}}(\mathtt{u}_i)$ is equal to $\mathfrak{h}_V(\mathtt{l})$ where $\mathtt{l}$ is the unique location such that $\widetilde{\sharp\mathtt{l}} \in \mathit{degrees}(i, \mathfrak{h})$. Similarly, $\mathfrak{V}_{\mathfrak{h}}(\mathtt{u}_j)$ is equal to $\mathfrak{h}_V(\mathtt{l})$ where $\mathtt{l}$ is the unique location such that $\widetilde{\sharp\mathtt{l}} \in \mathit{degrees}(j, \mathfrak{h})$.

First, let us suppose that $\mathfrak{h}_B \models_{\mathfrak{V}_{\mathfrak{h}}} \mathtt{u}_i \hookrightarrow \mathtt{u}_j$. This means that $\mathfrak{h}_B(\mathfrak{V}_{\mathfrak{h}}(\mathtt{u}_i)) = \mathfrak{V}_{\mathfrak{h}}(\mathtt{u}_j)$, and there is a unique location $\mathtt{l}_i$ such that $\mathfrak{h}_V(\mathtt{l}_i) = \mathfrak{V}_{\mathfrak{h}}(\mathtt{u}_i)$ and $\widetilde{\sharp\mathtt{l}_i} \in \mathit{degrees}(i, \mathfrak{h})$. There is also a unique location $\mathtt{l}_j$ such that $\mathfrak{h}_V(\mathtt{l}_j) = \mathfrak{V}_{\mathfrak{h}}(\mathtt{u}_j)$ and $\widetilde{\sharp\mathtt{l}_j} \in \mathit{degrees}(j, \mathfrak{h})$. Since $\widetilde{\sharp\mathtt{l}_i}$ and $\widetilde{\sharp\mathtt{l}_j}$ belong to the index spectrum of $\mathfrak{h}_V$, the locations $\mathtt{l}_i$ and $\mathtt{l}_j$, have the same number of predecessors in $\mathfrak{h}$ and in $\mathfrak{h}_V$ and they are also indices in $\mathfrak{h}$ (see Lemma 3.4.12). There are index locations $\mathtt{l}', \mathtt{l}''$ such that $\mathfrak{h} \models_{[\mathtt{u}\mapsto\mathtt{l}', \overline{\mathtt{u}}\mapsto\mathtt{l}'']} \mathtt{elt}_i(\mathtt{u}) \wedge \mathtt{elt}_j(\overline{\mathtt{u}}) \wedge \mathtt{u} \hookrightarrow \overline{\mathtt{u}}$. Note that $\mathfrak{h}_B(\mathfrak{V}_{\mathfrak{h}}(\mathtt{u}_i)) = \mathfrak{V}_{\mathfrak{h}}(\mathtt{u}_j)$ implies $\mathfrak{h}(\mathfrak{V}_{\mathfrak{h}}(\mathtt{u}_i)) = \mathfrak{V}_{\mathfrak{h}}(\mathtt{u}_j)$ since $\mathfrak{h}_B$ is a subheap of $\mathfrak{h}$. Consequently,

$\mathfrak{h} \models \exists\, \mathtt{u}\, \exists\, \overline{\mathtt{u}}\, \mathtt{elt}_i(\mathtt{u}) \wedge \mathtt{elt}_j(\overline{\mathtt{u}}) \wedge \mathtt{u} \hookrightarrow \overline{\mathtt{u}}$.

Now suppose that $\mathfrak{h} \models \exists\, \mathtt{u}\, \exists\, \overline{\mathtt{u}}\, \mathtt{elt}_i(\mathtt{u}) \wedge \mathtt{elt}_j(\overline{\mathtt{u}}) \wedge \mathtt{u} \hookrightarrow \overline{\mathtt{u}}$. Consequently, there are locations $\mathfrak{l}'$ and $\mathfrak{l}''$ such that $\mathfrak{h} \models_{[\mathtt{u} \mapsto \mathfrak{l}', \overline{\mathtt{u}} \mapsto \mathfrak{l}'']} \mathtt{elt}_i(\mathtt{u}) \wedge \mathtt{elt}_j(\overline{\mathtt{u}}) \wedge \mathtt{u} \hookrightarrow \overline{\mathtt{u}}$. Therefore there are locations $\mathfrak{l}_i$ and $\mathfrak{l}_j$ such that $\widetilde{\sharp}\mathfrak{l}_i \in \mathit{degrees}(i, \mathfrak{h})$, $\widetilde{\sharp}\mathfrak{l}_j \in \mathit{degrees}(j, \mathfrak{h})$, $\mathfrak{h}(\mathfrak{l}_i) = \mathfrak{l}'$, $\mathfrak{h}(\mathfrak{l}_j) = \mathfrak{l}''$ and of course $\mathfrak{h}(\mathfrak{l}') = \mathfrak{l}''$. By Lemma 3.4.12 and since $\mathit{indspect}(\mathfrak{h}_V) = \mathit{spect}(\mathfrak{h})$, $\mathfrak{h}_V(\mathfrak{l}_i) = \mathfrak{l}'$, $\widetilde{\sharp}\mathfrak{l}_i \in \mathit{indspect}(\mathfrak{h}_V)$, $\mathfrak{h}_V(\mathfrak{l}_j) = \mathfrak{l}''$ and $\widetilde{\sharp}\mathfrak{l}_j \in \mathit{indspect}(\mathfrak{h}_V)$. By construction of $\mathfrak{V}_\mathfrak{h}$ (see Definition 3.4.11), this implies that $\mathfrak{h}(\mathfrak{V}_\mathfrak{h}(\mathtt{u}_i)) = \mathfrak{V}_\mathfrak{h}(\mathtt{u}_j)$ and therefore $\mathfrak{h}_B \models_{\mathfrak{V}_\mathfrak{h}} \mathtt{u}_i \hookrightarrow \mathtt{u}_j$. Note that $\mathfrak{V}_\mathfrak{h}(\mathtt{u}_i) \notin \mathrm{dom}(\mathfrak{h}_V)$ since $\mathfrak{h}$ is X-well-formed.

Base case 3: ψ is equal to $\mathtt{P}_i(\mathtt{u}_j, \mathtt{u}_k)$.
Suppose that $\mathfrak{h}_B \models_{\mathfrak{V}_\mathfrak{h}} \mathtt{P}_i(\mathtt{u}_j, \mathtt{u}_k)$. By definition of the satisfaction relation $\models$, this is equivalent to $(\mathfrak{V}_\mathfrak{h}(\mathtt{u}_j), \mathfrak{V}_\mathfrak{h}(\mathtt{u}_k)) \in \mathfrak{V}_\mathfrak{h}(\mathtt{P}_i)$. By definition of $\mathfrak{V}_\mathfrak{h}$, $\mathfrak{V}_\mathfrak{h}(\mathtt{u}_j)$ is equal to $\mathfrak{h}_V(\mathfrak{l}_j)$ where $\mathfrak{l}_j$ is the unique index location such that $\widetilde{\sharp}\mathfrak{l}_j \in \mathit{degrees}(j, \mathfrak{h})$. Similarly, $\mathfrak{V}_\mathfrak{h}(\mathtt{u}_k)$ is equal to $\mathfrak{h}_V(\mathfrak{l}_k)$ where $\mathfrak{l}_k$ is the unique index location such that $\widetilde{\sharp}\mathfrak{l}_k \in \mathit{degrees}(k, \mathfrak{h})$. So, $\mathfrak{h} \models_{[\mathtt{u} \mapsto \mathfrak{l}_j]} \mathtt{vind}_j(\mathtt{u})$ and $\mathfrak{h} \models_{[\overline{\mathtt{u}} \mapsto \mathfrak{l}_k]} \mathtt{vind}_k(\overline{\mathtt{u}})$. Moreover, by definition of $\mathfrak{V}_\mathfrak{h}$, there are index locations $\mathfrak{l}_i$ and $\mathfrak{l}'_i$ such that

1. $\widetilde{\sharp}\mathfrak{l}'_i = \widetilde{\sharp}\mathfrak{l}_i + 1$,
2. $\widetilde{\sharp}\mathfrak{l}_i, \widetilde{\sharp}\mathfrak{l}'_i \in \mathit{degrees}(i, \mathfrak{h})$,
3. $\mathfrak{h}(\mathfrak{l}_i) = \mathfrak{V}_\mathfrak{h}(\mathtt{u}_j)$ and $\mathfrak{h}(\mathfrak{l}'_i) = \mathfrak{V}_\mathfrak{h}(\mathtt{u}_k)$,
4. $\mathfrak{h} \models_{[\overline{\mathtt{u}} \mapsto \mathfrak{l}_i]} \mathtt{vind}_i(\overline{\mathtt{u}})$ and $\mathfrak{h} \models_{[\mathtt{u} \mapsto \mathfrak{l}'_i]} \mathtt{vind}_i(\mathtt{u})$.

Finally, the formulae $\mathtt{elt}_j(\mathtt{u})$ and $\mathtt{elt}_k(\overline{\mathtt{u}})$ are defined so that $\mathfrak{h} \models_{[\mathtt{u} \mapsto \mathfrak{V}_\mathfrak{h}(\mathtt{u}_j)]} \mathtt{elt}_j(\mathtt{u})$ and $\mathfrak{h} \models_{[\overline{\mathtt{u}} \mapsto \mathfrak{V}_\mathfrak{h}(\mathtt{u}_k)]} \mathtt{elt}_k(\overline{\mathtt{u}})$. So, we have

- $\mathfrak{l}_i \to \mathfrak{V}_\mathfrak{h}(\mathtt{u}_j)$,
- $\widetilde{\sharp}\mathfrak{l}'_i = \widetilde{\sharp}\mathfrak{l}_i + 1$,
- $\mathfrak{l}'_i \to \mathfrak{V}_\mathfrak{h}(\mathtt{u}_k)$.

This guarantees the satisfaction of

$$\mathfrak{h} \models \exists\, \mathtt{u} \Big(\mathtt{elt}_j(\mathtt{u}) \wedge \exists\, \overline{\mathtt{u}} \Big(\overline{\mathtt{u}} \hookrightarrow \mathtt{u} \ \wedge\ \mathtt{vind}_i(\overline{\mathtt{u}}) \wedge$$

$$\exists\, \mathtt{u}\ (\sharp\mathtt{u} = \sharp\overline{\mathtt{u}} + 1 \wedge \mathtt{vind}_i(\mathtt{u}) \wedge \exists\, \overline{\mathtt{u}}\ (\mathtt{u} \hookrightarrow \overline{\mathtt{u}} \wedge \mathtt{elt}_k(\overline{\mathtt{u}})))\Big)\Big)$$

The proof in the other direction is by an easy verification and similar since all of the above implications are indeed equivalences.

Induction step. The induction hypothesis is the following: for every subformula ψ' of size strictly less than the size of ψ, for every $\mathtt{free}(\psi') \subseteq X' \subseteq [0, K]$, we have $\mathfrak{h}_B \models_{\mathfrak{V}_\mathfrak{h}} \psi'$ iff $\mathfrak{h} \models t(X', \psi')$. The cases when the outermost connective is Boolean are by an easy verification.

Case 1: ψ is equal to $\exists\, \mathtt{u}_i\ \psi'$. Suppose that $\mathfrak{h}_B \models_{\mathfrak{V}_\mathfrak{h}} \exists\, \mathtt{u}_i\ \psi'$. By definition of the satisfaction relation $\models$, there is $\mathfrak{l} \in \mathbb{N}$ such that $\mathfrak{h}_B \models_{\mathfrak{V}_\mathfrak{h}[\mathtt{u}_i \mapsto \mathfrak{l}]} \psi'$. In case $\mathfrak{l}$ belongs to the set Y defined below,

$$Y = \mathrm{dom}(\mathfrak{h}_V) \cup \{\mathfrak{l} \in \mathbb{N} : \mathfrak{h} \models_{[\mathtt{u} \mapsto \mathfrak{l}]} \bigvee_{j \in X} (\mathtt{llp}_j(\mathtt{u}) \vee \mathtt{lrp}_j(\mathtt{u}))\}$$

(and therefore $\mathfrak{l}$ is an isolated location in $\mathfrak{h}_B$), we pick another location $\mathfrak{l}'$ that does not belong to Y and that is also isolated in $\mathfrak{h}_B$. It is then easy to show that $\mathfrak{h}_B \models_{\mathfrak{V}_\mathfrak{h}[\mathtt{u}_i \mapsto \mathfrak{l}]} \psi'$ iff $\mathfrak{h}_B \models_{\mathfrak{V}_\mathfrak{h}[\mathtt{u}_i \mapsto \mathfrak{l}']} \psi'$. So, without any loss of generality, below we assume that $\mathfrak{l}$ does not belong to Y.

Let us build $\mathfrak{h}^i_V$ and an assignment $\mathfrak{f}$ such that:

1. $\mathfrak{h}^i_V \models_\mathfrak{f} \mathtt{localval}_i(\overline{\mathtt{u}})$,
2. $\mathfrak{h} \models_\mathfrak{f} \mathtt{indmin}(\mathtt{u}) \wedge \mathtt{isoloc}(\overline{\mathtt{u}})$,
3. $\mathfrak{h} \uplus \mathfrak{h}^i_V \models_\mathfrak{f} \mathtt{wfh}_{X \cup \{i\}} \wedge \mathtt{indmin}(\mathtt{u}) \wedge \mathtt{llp}_i(\overline{\mathtt{u}})$.

Assume that $max(X) = j$ and m be the degree of the right j-parenthesis with greatest degree. It is easy to define a local i-valuation $\mathfrak{h}^i_V$ disjoint from $\mathfrak{h}$ such that the degree of the left i-parenthesis is $m+1$, the degree of the right i-parenthesis is $m+3$, the degree of the unique entry is $m+2$, its element is precisely $\mathfrak{l}$ and all the locations in its domain are isolated in $\mathfrak{h}$ (always possible since $\mathrm{dom}(\mathfrak{h}) \cup \mathrm{ran}(\mathfrak{h})$ is finite).

It is not difficult to check that $\mathfrak{h}^i_V$ and $\mathfrak{f}$ satisfy the above conditions. Since $\mathfrak{h} \uplus \mathfrak{h}^i_V$ is $(X \cup \{i\})$-well-formed by construction, by Lemma 3.4.12, we have $\mathfrak{V}_\mathfrak{h}[\mathtt{u}_i \mapsto \mathfrak{l}]$ equal to $\mathfrak{V}_{\mathfrak{h} \uplus \mathfrak{h}^i_V}$. Hence, $\mathfrak{h}_B \models_{\mathfrak{V}_{\mathfrak{h} \uplus \mathfrak{h}^i_V}} \psi'$ and by the induction hypothesis, we get $\mathfrak{h} \uplus \mathfrak{h}^i_V \models t(X \cup \{i\}, \psi')$. However, it is easy to conclude then that $\mathfrak{h} \models t(X, \psi')$. Indeed, $\mathfrak{h}$ satisfies the formula below

$$\exists\, \mathtt{u}\ \exists\, \overline{\mathtt{u}}\ (\mathtt{indmin}(\mathtt{u}) \wedge \mathtt{isoloc}(\overline{\mathtt{u}}) \wedge (\mathtt{localval}_i(\overline{\mathtt{u}}) \mathrel{\overrightarrow{-\!\!*}}$$

$$(\mathtt{wfh}_{X\cup\{i\}} \wedge \mathtt{indmin}(\mathtt{u}) \wedge \mathtt{llp}_i(\overline{\mathtt{u}}) \wedge t(X \cup \{i\}, \psi))))$$

whenever there are locations $\mathfrak{l}^\star$ and $\mathfrak{l}^{\star\star}$ and a disjoint heap $\mathfrak{h}^\star$ such that:

1. $\mathfrak{l}^\star$ is the unique large 0-parenthesis in $\mathfrak{h}$ and $\mathfrak{l}^{\star\star}$ is isolated in $\mathfrak{h}$,
2. $\mathfrak{h}^\star$ is an i-local valuation such that the index of the left i-parenthesis is $\mathfrak{l}^{\star\star}$, $\mathfrak{h} \uplus \mathfrak{h}^\star$ is $(X \cup \{i\})$-well-formed,
3. $\mathfrak{l}^\star$ is the left 0-parenthesis in $\mathfrak{h} \uplus \mathfrak{h}^\star$ and $\mathfrak{l}^{\star\star}$ is the left i-parenthesis in $\mathfrak{h} \uplus \mathfrak{h}^\star$,
4. $\mathfrak{h} \uplus \mathfrak{h}^\star \models_{[\mathtt{u}\mapsto\mathfrak{l}^\star, \overline{\mathtt{u}}\mapsto\mathfrak{l}^{\star\star}]} t(X \cup \{i\}, \psi)$.

It is clear that such objects exist by considering the above construction.

The proof in the other direction (i.e. $\mathfrak{h} \models t(X, \psi)$ implies $\mathfrak{h}_B \models_{\mathfrak{V}_\mathfrak{h}} \exists\, \mathtt{u}_i\ \psi'$) is actually very similar since most of the above implications are indeed equivalences.

Case 2: ψ is equal to $\exists\, \mathtt{P}_i\ \psi'$. Suppose that $\mathfrak{h}_B \models_{\mathfrak{V}_\mathfrak{h}} \exists\, \mathtt{P}_i\ \psi'$. By definition of the satisfaction relation $\models$, there is a finite binary relation $R \subseteq \mathbb{N}^2$ such that $\mathfrak{h}_B \models_{\mathfrak{V}_\mathfrak{h}[\mathtt{P}_i\mapsto R]} \psi'$. In case R involves locations in Y defined below,

$$Y = \mathrm{dom}(\mathfrak{h}_V) \cup \{\mathfrak{l} \in \mathbb{N} : \mathfrak{h} \models_{[\mathtt{u}\mapsto\mathfrak{l}]} \bigvee_{j\in X} (\mathtt{llp}_j(\mathtt{u}) \vee \mathtt{lrp}_j(\mathtt{u}))\}$$

(and therefore R involves some isolated locations in $\mathfrak{h}_B$), we pick another R' (of same cardinality β) that does not involve locations in Y. It is then easy to show that $\mathfrak{h}_B \models_{\mathfrak{V}_\mathfrak{h}[\mathtt{P}_i\mapsto R]} \psi'$ iff $\mathfrak{h}_B \models_{\mathfrak{V}_\mathfrak{h}[\mathtt{P}_i\mapsto R']} \psi'$. So, without any loss of generality, below we assume that R does not involve locations in Y (see also [BDL12, Lemma 2.1]).

Let us build $\mathfrak{h}^i_V$ and an assignment $\mathfrak{f}$ such that:

1. $\mathfrak{h}^i_V \models_\mathfrak{f} \mathtt{localval}_i(\overline{\mathtt{u}})$,
2. $\mathfrak{h} \models_\mathfrak{f} \mathtt{indmin}(\mathtt{u}) \wedge \mathtt{isoloc}(\overline{\mathtt{u}})$,
3. $\mathfrak{h} \uplus \mathfrak{h}^i_V \models_\mathfrak{f} \mathtt{wfh}_{X\cup\{i\}} \wedge \mathtt{indmin}(\mathtt{u}) \wedge \mathtt{llp}_i(\overline{\mathtt{u}})$.

Assume that $max(X) = j$ and m be the degree of the right j-parenthesis with greatest degree. It is easy to define a local i-valuation $\mathfrak{h}^i_V$ disjoint from $\mathfrak{h}$ such that

1. the degree of the left i-parenthesis is $m + 1$,
2. the degree of the right i-parenthesis is $(m + 1) + 3\beta + 1$ for some $\beta \geq 0$,

3. there are 2β entries,

4. for every pair $(\mathfrak{l}, \mathfrak{l}')$ in R, there are two entries of consecutive degrees whose elements are $\mathfrak{l}$ and $\mathfrak{l}'$ respectively.

This is always possible since $\mathrm{dom}(\mathfrak{h}) \cup \mathrm{ran}(\mathfrak{h})$ and R are finite.

It is not difficult to check that $\mathfrak{h}^i_V$ and $\mathfrak{f}$ satisfy the above conditions. Since $\mathfrak{h} \uplus \mathfrak{h}^i_V$ is $(X \cup \{i\})$-well-formed by construction, by Lemma 3.4.12, we have $\mathfrak{V}_{\mathfrak{h}}[\mathtt{P}_i \mapsto R]$ equal to $\mathfrak{V}_{\mathfrak{h}\uplus\mathfrak{h}^i_V}$. Hence, $\mathfrak{h}_B \models_{\mathfrak{V}_{\mathfrak{h}\uplus\mathfrak{h}^i_V}} \psi'$ and by the induction hypothesis, we get $\mathfrak{h} \uplus \mathfrak{h}^i_V \models t(X \cup \{i\}, \psi')$. However, it is easy to conclude then that $\mathfrak{h} \models t(X, \psi')$. Indeed, $\mathfrak{h}$ satisfies the formula below

$$\exists\, \mathtt{u}\, \exists\, \overline{\mathtt{u}}\, (\mathtt{indmin}(\mathtt{u}) \wedge \mathtt{isoloc}(\overline{\mathtt{u}}) \wedge (\mathtt{localval}_i(\overline{\mathtt{u}}) \mathrel{\overrightarrow{-\!\!*}}$$

$$(\mathtt{wfh}_{X\cup\{i\}} \wedge \mathtt{indmin}(\mathtt{u}) \wedge \mathtt{llp}_i(\overline{\mathtt{u}}) \wedge t(X \cup \{i\}, \psi))))$$

whenever there are locations $\mathfrak{l}^\star$ and $\mathfrak{l}^{\star\star}$ and a disjoint heap $\mathfrak{h}^\star$ such that:

1. $\mathfrak{l}^\star$ is the unique large 0-parenthesis in $\mathfrak{h}$ and $\mathfrak{l}^{\star\star}$ is isolated in $\mathfrak{h}$,

2. $\mathfrak{h}^\star$ is an i-local valuation such that the index of the left i-parenthesis is $\mathfrak{l}^{\star\star}$, $\mathfrak{h} \uplus \mathfrak{h}^\star$ is $(X \cup \{i\})$-well-formed,

3. $\mathfrak{l}^\star$ is the left 0-parenthesis in $\mathfrak{h} \uplus \mathfrak{h}^\star$ and $\mathfrak{l}^{\star\star}$ is the left i-parenthesis in $\mathfrak{h} \uplus \mathfrak{h}^\star$,

4. $\mathfrak{h} \uplus \mathfrak{h}^\star \models_{[\mathtt{u}\mapsto\mathfrak{l}^\star, \overline{\mathtt{u}}\mapsto\mathfrak{l}^{\star\star}]} t(X \cup \{i\}, \psi)$.

It is clear that such objects exist by considering the above construction.

The proof in the other direction (i.e. $\mathfrak{h} \models t(X, \psi)$ implies $\mathfrak{h}_B \models_{\mathfrak{V}_{\mathfrak{h}}} \exists\, \mathtt{P}_i\ \psi'$) is actually very similar since most of the above implications are indeed equivalences. **QED**

Here is the major expressiveness result.

Theorem 3.4.14. For every sentence φ in 1DSOL, for every heap $\mathfrak{h}$, we have $\mathfrak{h} \models \varphi$ iff $\mathfrak{h} \models \mathtt{T}(\varphi)$, so WSOL and 1SL2($\mathrel{\overrightarrow{-\!\!*}}$) have the same expressive power.

The proof of Theorem 3.4.14 is left as Exercise 3.3.

Observe that $\mathtt{T}(\varphi)$ can be computed in logarithmic space in the size of φ (to do this, one must also check the size of all the formulae built in the previous proofs). So, the restriction to two variables in 1SL2($\mathrel{\overrightarrow{-\!\!*}}$) does not reduce the expressive power, unlike restrictions in [Ven91, EVW97] but we know also other

logics restricted to two variables that are expressively complete, see e.g. [LSW01, MdR05].

We get the ultimate undecidability result below (no separating conjunction, two quantified variables, one record field).

Corollary 3.4.15. [DD14] The satisfiability problem for 1SL2($-\!\!*$) is undecidable.

The absence of program variables in the logic 1SL2($-\!\!*$) makes the proof of Corollary 3.4.15 even more difficult to design, which is perfect to obtain the sharpest undecidability result. An expressiveness result with program variables is possible and it is left as Exercise 3.6.

Theorem 3.4.16. The set of valid formulae in 1SL2($-\!\!*$) is not recursively enumerable.

Indeed, finitary validity for classical predicate logic restricted to a single binary predicate is not recursively enumerable, which implies a similar property for DSOL and therefore for 1SL2($-\!\!*$) by Theorem 3.4.14.

A quick argument for proving Theorem 3.4.16 consists in noting that second-order logic is not finitely axiomatizable and 1SL2($-\!\!*$) is equivalent to it, but this would be too sloppy since there are so many variants of second-order logic, and some of them are indeed finitely axiomatizable. In order to be more precise and to show Theorem 3.4.16, a more direct proof consists in combining the following arguments.

- First-order theory of natural numbers with addition and multiplication is not recursively enumerable by Gödel's first incompleteness theorem.
- There is a logarithmic-space reduction tr_1 such that for any formula φ from first-order arithmetic, φ is valid iff $tr_1(\varphi)$ is valid in 1WSOL. To show this, it is sufficient to represent natural numbers by the cardinals of finite sets and to deal with addition and multiplication by performing equality tests between finite set cardinalities. This can be done by using dyadic or ternary predicate symbols, for instance to state the existence of some bijection between two finite sets (see Theorem 1.3.2). By way of example, the atomic formula $\mathtt{u}_1 \times \mathtt{u}_2 = \mathtt{u}_3$ amounts to check whether the product set made of the interpretation of the monadic second-order variables $\mathtt{P}_1$ and $\mathtt{P}_2$ has the same cardinality as the interpretation of the monadic second-order variable $\mathtt{P}_3$. Obviously, this assumes that each variable $\mathtt{u}_i$ has a unique corresponding monadic second-order variable $\mathtt{P}_i$. So the formula $\mathtt{u}_1 \times \mathtt{u}_2 = \mathtt{u}_3$ can be encoded by:

$$\exists\, \mathtt{P}\ \mathtt{PRODUCT}(\mathtt{P}, \mathtt{P}_1 \times \mathtt{P}_2) \wedge \mathtt{EQCARD}(\mathtt{P}, \mathtt{P}_3)$$

where $\mathtt{PRODUCT}(\mathtt{P}, \mathtt{P}_1 \times \mathtt{P}_2) \stackrel{\text{def}}{=} \forall\, \mathtt{u}, \mathtt{u}'\ \mathtt{P}(\mathtt{u}, \mathtt{u}') \Leftrightarrow \mathtt{P}_1(\mathtt{u}) \wedge \mathtt{P}_2(\mathtt{u}')$. The formula $\mathtt{EQCARD}(\mathtt{P}, \mathtt{P}_3)$ stating that the interpretation of the binary second-order variable has the same cardinality as the interpretation of the unary second-order variable can be defined similarly, but this requires to introduce a ternary second-order variable specified as a bijection between the two sets.

3.5 Exercises

Exercise 3.1. Prove Lemma 3.4.6.

Exercise 3.2. Prove Lemma 3.4.7.

Exercise 3.3. Prove Theorem 3.4.14.

Exercise 3.4. Let $1\mathrm{SL}^{\infty}$ be the variant of 1SL in which the heap domain can be either finite or infinite (in 1SL, the heap domain is necessarily finite).

a) Show that the set of valid formulae for $1\mathrm{SL}^{\infty}$ without separating connectives is recursively enumerable.

b) Define a formula $\mathtt{seg}$ in $1\mathrm{SL}^{\infty}$ that characterises the **segmented** heaps, i.e. those heaps $\mathfrak{h}$ such that $\mathrm{dom}(\mathfrak{h}) \cap \mathrm{ran}(\mathfrak{h}) = \emptyset$ and no location has strictly more than one predecessor.

c) Show that for any heap for $1\mathrm{SL}^{\infty}$, $\mathrm{dom}(\mathfrak{h})$ is infinite iff $\mathfrak{h}$ satisfies $\mathtt{seg} \mathrel{\overline{-\!\!*}} \forall\, \mathtt{u}\ \mathtt{alloc}(\mathtt{u})$.

d) Conclude that $1\mathrm{SL}^{\infty}$ (with separating connectives) does not admit a recursively enumerable set of valid formulae.

Exercise 3.5.

a) Define a formula in 1SL2 that states that the number of predecessors of $\mathtt{u}$ with zero precedessor is at least 2.

b) Define a formula $\mathtt{allzpred}(\mathtt{u})$ in $1\mathrm{SL2}(-\!\!*)$ that holds true exactly when all the predecessors of $\mathtt{u}$ have no predecessor.

c) Define a formula $(\sharp_z \mathtt{u} = 0)$ in $1\mathrm{SL2}(-\!\!*)$ that holds true exactly when all the predecessors of $\mathtt{u}$ have at least one predecessor.

d) Define a formula $\mathtt{size} = 1$ in 1SL2($-\!\!*$) that holds true exactly when the heap domain has exactly one location (the separating implication $-\!\!*$ is not even needed).

e) Assuming that φ charaterises the heaps such that the number of predecessors of $\mathtt{u}$ that have no predecessor is at most $k-1$, define a formula that characterises the heaps such that the number of predecessors of $\mathtt{u}$ that have no predecessor is at most k (by using $\mathtt{size} = 1$, φ and the septraction operator).

f) Show that for all $\bowtie \in \{\leq, \geq, <, >\}$ and $k \geq 2$, there is a formula in 1SL2($-\!\!*$) that holds true exactly that the number n of predecessors of $\mathtt{u}$ that have no predecessor verifies $n \bowtie k$.

Exercise 3.6. . Extend the translation provided in Section 3.4 to deal with program variables.

3.6 Bibliographical References on Expressiveness

Expressive completeness. The literature is rich with results comparing the expressive power of non-classical logics with most standard logics such as first- or second-order logic. For instance, the celebrated Kamp's Theorem [Kam68, Rab14] amounts to stating that linear-time temporal logic (LTL) is equal in expressive power to first-order logic. More generally, we know the expressive completeness of Stavi connectives for general linear time, see e.g. [GHR94]. This has been refined to the restriction to two variables, leading to the equivalence between unary LTL and FO2, see e.g. [EVW97, Wei11]. Monadic second-order logic (MSO) is another yardstick logic and, for instance, it is well-known that ω-regular languages (those definable by Büchi automata) are exactly those definable in MSO, see e.g. [Str94]. Similarly, extended temporal logic ETL, defined in [Wol83] and extending LTL, is also known to be equally expressive with MSO. This applies also to linear μ-calculus [Var88] or to PSL [Lan07], to quote a few more examples. On non-linear structures, bisimulation invariant fragment of MSO and modal μ-calculus have been shown equivalent [JW96]. In addition, there is a wealth of results relating first-order logic with two variables and non-classical logics, providing a neat characterisation of the expressive power of many formalisms since first-order logic and second-order logic are queen logics. For instance, Boolean modal logic with converse and identity is as expressive as first-order logic with two quantified variables (FO2) [LSW01]. Sometimes, a

third variable is needed to get expressive completeness. For instance, in [Ven91] it is proved that interval logic with connectives Chop, D and T is expressively complete over linear flows of time with respect to first-order logic restricted to three quantified variables. In the realm of interval temporal logics, we also know expressive completeness of metric propositional neighborhood logic with respect to the two-variable fragment of first-order logic for linear orders with successor function, interpreted over natural numbers [BDG$^+$10].

Expressiveness of separation logics It is known since [COY01] that first-order separation logic with two record fields (herein called 2SL) is undecidable (see also Section 1.3.4) and this is sharpened in [BDL12] by showing that 1SL is also undecidable, as a consequence of the expressive equivalence between 1SL and weak second-order logic. More recently, 1SL restricted to two variables (1SL2) is shown undecidable too [DD15b] (see Section 3.3). From the very beginning, the relationships between separation logic and second-order logic have been quite puzzling (see e.g. an interesting answer with infinite arbitrary structures in [KR04]). Moreover, comparisons of fragments have been also studied, for instance 1SL($*$) has been established strictly less expressive than MSO in [AD09] (see also the related work [Mar06] or Section 4.4). So, in this chapter, we have shown that first-order separation logic with one record field, two quantified variables, and no separating conjunction is as expressive as weak second-order logic on heaps; in short, 1SL2($-\!\!*$) $\equiv$ 1WSOL [DD14]. Because we forbid ourselves the use of many syntactic resources, this underlines even further the power of the magic wand. By way of comparison with [GOR99, IRR$^+$04], we show undecidability of a two-variable logic with second-order features. Our main undecidability result cannot be derived from [GOR99, IRR$^+$04] since in 1SL models, we deal with a single functional binary relation, namely the finite heap.

Chapter 4

RELATIONSHIPS TO OTHER LOGICS

Contents

In this chapter, we investigate further how to reduce the satisfiability problem for non-classical logics into similar problems for separation logics. In Section 4.1, we introduce several versions of separation logics with data values and we show undecidability by reduction from the satisfiability problem for Freeze LTL. In that section, we also explain how to obtain undecidability of 1SL2 by reducing first-order logics on data words; this provides an alternative undecidability

proof for 1SL2 (see also Section 3.3). Section 4.2 is dedicated to interval temporal logic PITL and we establish that the satisfiability problem for 1SL2(∗) is non-elementary by reduction from the satisfiability problem for PITL. Section 4.3 introduces a modal logic for heaps MLH and shows the decidability of MLH(∗) by translation into 1SL2(∗) by using a translation similar to the one from the modal logic K into FO2. The non-elementarity of the satisfiability problem for MLH(∗) is also established. Section 4.4 is about the decidability status of 1MSOL and 1SL(∗) and their difference of expressivenesss. 1SL(∗) is an important fragment to consider—and an important one to show decidable—as some verification applications do not require the use of the separating implication.

Highlights of the chapter

1. The satisfiability problem for the logic 1SL3(∗)[$\mathbb{Z}$, =] with data values in $\mathbb{Z}$ is undecidable (Theorem 4.1.1) [BBL09, Theorem 3] . The proof is obtained by reducing an undecidable version of linear-time temporal logic LTL with the freeze operator [FS09].

2. The satisfiability problem for 1SL2(∗) has non-elementary complexity (Theorem 4.2.7). This is shown by reduction from propositional interval temporal logic with the locality condition [DD15b].

4.1 Data Logics

4.1.1 Separation logic with data

In memory states defined in Section 1.2.1, heaps are of the form $\mathbb{N} \rightharpoonup \mathbb{N}^k$ when k record fields are involved. No record field is really distinguished and the logic kSL mainly allows to reason about the shape properties of the heap (and not so much on functional correctness). However, it is often important to be able to reason about data values, a typical example would be to consider programs that produce sorted lists. In that case, we would like to specify that the values occurring in a list are linearly ordered. Pointer arithmetic is another means to reason about data values when the set of locations (herein, represented by the set $\mathbb{N}$) is equipped with relations other than equality. Even though it is well-known that adding data domains easily leads to undecidable logics, see e.g. [DD07, BMS$^+$06], there exist several successful examples of logics able to reason about heap structures and data values, while having decidable reasoning tasks, see e.g. [BHJS07, BBL09,

BDES09, MPQ11, Bro13]. Reasoning about data values mainly means to be able to distinguish at least one record field dedicated to data values and to express data constraints in the formulae. That is why, in full generality, data domains need to be introduced in the semantics.

A **data domain** is a pair $(\mathfrak{D}, (\mathfrak{R}_i)_{i\in I})$ where $\mathfrak{D}$ is a non-empty set, I is an index set and each $\mathfrak{R}_i$ is a relation of arity $a(i)$ on $\mathfrak{D}$, that is, it is a subset of $\mathfrak{D}^{a(i)}$. A typical example of data domain is $(\mathbb{Z}, <, =)$. The index set I is not necessarily finite and below, we assume that $\mathfrak{D}$ is infinite and the family $(\mathfrak{R}_i)_{i\in I}$ contains the diagonal relation on $\mathfrak{D}$ so that equality tests between data values can be expressed in the logic. Indeed, this makes the data domain all the more interesting and non-trivial. A **memory state with data** (with respect to the data domain $(\mathfrak{D}, (\mathfrak{R}_i)_{i\in I})$) is a triple $(\mathfrak{s}, \mathfrak{h}, \mathfrak{d})$ such that $(\mathfrak{s}, \mathfrak{h})$ is a memory state and $\mathfrak{d}$ is a partial function $\mathbb{N} \rightharpoonup \mathfrak{D}$. Below, we also assume that $\mathrm{dom}(\mathfrak{h}) = \mathrm{dom}(\mathfrak{d})$ in order to have correspondences between partial functions of the form $\mathbb{N} \rightharpoonup \mathfrak{D} \times \mathbb{N}^k$ (the first record field is therefore dedicated to data values) and pairs of the form $(\mathfrak{h}, \mathfrak{d})$ when $\mathfrak{h}$ is a heap with k record fields. This is analogous to what is defined in [BBL09], a pioneering work for separation logic with data values but other options are possible, even though not investigated below.

The logic $k\mathrm{SL}[\mathfrak{D}, (\mathfrak{R}_i)_{i\in I}]$ is defined as kSL except that atomic formulae of the form $\mathtt{R}_i(\mathtt{e}_1, \ldots, \mathtt{e}_{a(i)})$ are added for each $i \in I$ and the models are memory states with data with respect to $(\mathfrak{D}, (\mathfrak{R}_i)_{i\in I})$. In order to avoid confusion with equality between locations (by contrast to equality between their data values, if any), we write $\mathtt{e} \sim \mathtt{e}'$ to denote the equality formula between two data values, following a similar convention from [BMS+06]. For instance, linear ordered data domains have been considered in [DD07, ST11] with LTL-like logics or in [BBL09] with separation logic where $1\mathrm{SL}[\mathbb{Z}, \leq, =]$ has been investigated. The satisfaction relation is extended in order to cope with the new atomic formulae:

- $(\mathfrak{s}, \mathfrak{h}, \mathfrak{d}) \models_\dagger \mathtt{R}_i(\mathtt{e}_1, \ldots, \mathtt{e}_{a(i)}) \stackrel{\text{def}}{\Leftrightarrow} \mathfrak{d}([\![\mathtt{e}_1]\!]), \ldots, \mathfrak{d}([\![\mathtt{e}_{a(i)}]\!])$ are defined and $\mathfrak{R}_i(\mathfrak{d}([\![\mathtt{e}_1]\!]), \ldots, \mathfrak{d}([\![\mathtt{e}_{a(i)}]\!]))$.

In the logic $k\mathrm{SL}[\mathfrak{D}, (\mathfrak{R}_i)_{i\in I}]$, there is no quantification on data values but this would be possible by defining a multi-sorted separation logic to distinguish locations from data values, as done in [BBL09]. Similarly, the logics of the form kSL happen to be quite expressive (see Chapter 3) and adding the ability to reason about data can only increase the computational complexity of the reasoning tasks. That is why, most of the variants of separation logic with data considered in [BBL09] banish the magic wand operator and provide even further restrictions. Let $1\mathrm{SL}^{\mathrm{sdc}}_<$ be the fragment of $1\mathrm{SL}(*)[\mathbb{Z}, \leq, =]$ (without magic wand) such that the

atomic formulae about data values can only occur in subformulae of the one of the forms below:

$$\mathtt{e} \hookrightarrow \mathtt{e}' \wedge (\mathtt{e} \leq \mathtt{e}') \qquad \mathtt{e} \hookrightarrow \mathtt{e}' \wedge (\mathtt{e}' \leq \mathtt{e})$$

As noted in [BBL09], only **short-distance comparisons** are possible in $1SL^{sdc}_{<}$ and this may allow to specify sorted lists. For example, the formula below specifies that the list between $\mathtt{x}_1$ and $\mathtt{x}_2$ (assuming that $\mathtt{reach}(\mathtt{x}_1, \mathtt{x}_2) \wedge \mathtt{alloc}(\mathtt{x}_1) \wedge \mathtt{alloc}(\mathtt{x}_2)$ holds true) is sorted:

$$\forall \mathtt{u}, \mathtt{u}' \, ((\mathtt{reach}(\mathtt{x}_1, \mathtt{u}) \wedge \mathtt{reach}(\mathtt{u}, \mathtt{u}') \wedge \mathtt{reach}(\mathtt{u}', \mathtt{x}_2)) \Rightarrow \mathtt{u} \leq \mathtt{u}')$$

where $\leq$ is "less than or equal relation" that is definable in the data domain $(\mathbb{Z}, <, =)$.

Other types of logics with data have been considered in the literature. One of the most prominent ones is the logic STRAND introduced [MPQ11] that can state constraints on the heap structures but also on the data. Recursive structures are defined thanks to monadic second-order logic whereas the use of data constraints is significantly limited. The very combination of the two types of properties allow to reason with heap-manipulation programs using deductive verification and SMT solvers, such as Z3 [dMB08]. Another related logic is the one introduced in [BDES09] for which a quite general framework is proposed to reason about heap structures and data values.

4.1.2 Undecidability for separation logic with data

The logic 1SL(∗) happens to be decidable (see forthcoming Section 4.4). Below, we show that 1SL3(∗)[$\mathbb{Z}$, =], i.e. 1SL3(∗) augmented with data values in which only equality tests are possible is undecidable. The definition of 1SL(∗)[$\mathbb{Z}$, =] can be found in Section 4.1.1. We provide a reduction from the satisfiability problem for some temporal logic with the freeze operator, see e.g. [DL09, FS09] whereas the original proof in [BBL09] uses a first-order logic on data words.

Below, we only consider data words of dimension 0. In Section 3.1, we have seen that there is a formula $\mathbf{dw}(\alpha, 0)$ such that $\mathfrak{h} \models \mathbf{dw}(\alpha, 0)$ iff $\mathfrak{h}$ is an $(\alpha, 0)$-fishbone. Let us consider a subclass of $(\alpha, 0)$-fishbone heaps so that the main path has at least two locations. We write $\mathbf{dw}'(\alpha, 0)$ to denote the formulae characterizing such heaps. There is also a formula $\mathtt{mp}(\mathtt{u})$ in 1SL2(∗) such that for any $(\alpha, 0)$-fishbone heap $\mathfrak{h}$ with the above definition, $\mathfrak{h} \models_{\mathfrak{f}} \mathtt{mp}(\mathtt{u})$ iff $\mathfrak{f}(\mathtt{u})$ is on the main path and it has exactly one predecessor.

Now, let us show that the satisfiability problem for 1SL3(∗)[$\mathbb{Z}$, =] is undecidable by designing a reduction from an undecidable logic whose models are data words. We have already explained how 1SL3(∗)[$\mathbb{Z}$, =] can encode data words. As mentioned earlier, there exist many formalisms to specify properties about data words; among them can be found temporal logics with the freeze operator [DL09, FS09]. Let $\mathrm{LTL}^{\alpha}_{\downarrow}(\mathtt{F},\mathtt{F}^{-1})$ be the set of formulae defined as follows:

$$\varphi ::= a \mid \uparrow \mid \downarrow \varphi \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathtt{F}\varphi \mid \mathtt{F}^{-1}\varphi$$

with $a \in [1,\alpha]$. The operators $\mathtt{F}$ and $\mathtt{F}^{-1}$ are the standard (non-strict) temporal operators stating the existence of some future [resp. past] position satisfying a given property. The atomic formula $\uparrow$ and the freeze operator $\downarrow$ are interpreted as in hybrid logics [ABM01] except that instead of storing a node address, a data value is stored. Formulae in $\mathrm{LTL}^{\alpha}_{\downarrow}(\mathtt{F},\mathtt{F}^{-1})$ are interpreted over data words $\mathfrak{dw} = (a^1,\mathfrak{v}^1)\cdots(a^L,\mathfrak{v}^L)$ in $([1,\alpha]\times\mathbb{N})^+$ via the satisfaction relation $\models_{\mathfrak{v}}$ (Boolean clauses are omitted and $i \in [1,L]$):

$$\begin{array}{lcl}
\mathfrak{dw}, i \models_{\mathfrak{v}} a & \overset{\text{def}}{\Leftrightarrow} & a^i = a \\
\mathfrak{dw}, i \models_{\mathfrak{v}} \uparrow & \overset{\text{def}}{\Leftrightarrow} & \mathfrak{v}^i = \mathfrak{v} \\
\mathfrak{dw}, i \models_{\mathfrak{v}} \downarrow \varphi & \overset{\text{def}}{\Leftrightarrow} & \mathfrak{dw}, i \models_{\mathfrak{v}^i} \varphi \\
\mathfrak{dw}, i \models_{\mathfrak{v}} \mathtt{F}\varphi & \overset{\text{def}}{\Leftrightarrow} & \text{there is } i' \in [i,L] \text{ such that } \mathfrak{dw}, i' \models_{\mathfrak{v}^i} \varphi \\
\mathfrak{dw}, i \models_{\mathfrak{v}} \mathtt{F}^{-1}\varphi & \overset{\text{def}}{\Leftrightarrow} & \text{there is } i' \in [1,i] \text{ such that } \mathfrak{dw}, i' \models_{\mathfrak{v}^i} \varphi.
\end{array}$$

A sentence is satisfiable $\overset{\text{def}}{\Leftrightarrow}$ there is a data word $\mathfrak{dw}$ in $([1,\alpha]\times\mathbb{N})^+$ such that $\mathfrak{dw}, 1 \models \varphi$ (no need to specify a data value since φ is closed). The satisfiability problem for $\mathrm{LTL}^{\alpha}_{\downarrow}(\mathtt{F},\mathtt{F}^{-1})$ is known to be undecidable [FS09, Theorem 4].

Let us define $T(\varphi)$ as follows:

$$T(\varphi) \overset{\text{def}}{=} \mathbf{dw}'(\alpha, 0) \wedge tr(\mathtt{u}_0, \varphi) \wedge \mathtt{mp}(\mathtt{u}_0) \wedge \neg\exists\, \mathtt{u}_1\ (\mathtt{u}_1 \hookrightarrow \mathtt{u}_0 \wedge \mathtt{mp}(\mathtt{u}_1))$$

We aim at satisfying that φ is satisfiable iff $T(\varphi)$ is satisfiable in 1SL3(∗)[$\mathbb{Z}$, =]. The map tr takes two arguments: a quantified variable among $\{\mathtt{u}_0, \mathtt{u}_1\}$ (variables are indeed recycled, see e.g. [Gab81]) and a formula. A third variable $\mathtt{u}_2$ is used but its purpose is to store a data value because of the presence of the freeze operator.

We define the logarithmic-space translation tr as follows ($i \in \{0,1\}$) where tr is homomorphic for Boolean connectives:

$$
\begin{array}{lcl}
tr(\mathtt{u}_i, \uparrow) & \stackrel{\text{def}}{=} & (\mathtt{u}_2 \sim \mathtt{u}_i) \\
tr(\mathtt{u}_i, a) & \stackrel{\text{def}}{=} & (\sharp \mathtt{u}_i = a + 2) \\
tr(\mathtt{u}_i, \downarrow \psi) & \stackrel{\text{def}}{=} & \exists\, \mathtt{u}_2\, ((\mathtt{u}_2 \sim \mathtt{u}_i) \wedge tr(\mathtt{u}_i, \psi)) \\
tr(\mathtt{u}_i, \mathtt{F}\psi) & \stackrel{\text{def}}{=} & \exists\, \mathtt{u}_{1-i}\, (tr(\mathtt{u}_{1-i}, \psi) \wedge \mathtt{mp}(\mathtt{u}_{1-i}) \wedge \mathtt{last}(\mathtt{u}_{1-i}) \wedge \mathtt{reach}(\mathtt{u}_i, \mathtt{u}_{1-i})) \\
tr(\mathtt{u}_i, \mathtt{F}^{-1}\psi) & \stackrel{\text{def}}{=} & \exists\, \mathtt{u}_{1-i}\, (tr(\mathtt{u}_{1-i}, \psi) \wedge \mathtt{mp}(\mathtt{u}_{1-i}) \wedge \mathtt{last}(\mathtt{u}_{1-i}) \wedge \mathtt{reach}(\mathtt{u}_{1-i}, \mathtt{u}_i)).
\end{array}
$$

We recall that $\mathtt{u}_2 \sim \mathtt{u}_i$ holds true when $\mathtt{u}_2$ and $\mathtt{u}_i$ are allocated and have the same data value.

We have already seen how to define the formulae $\sharp \mathtt{u}_i = a + 2$, $\mathtt{reach}(\mathtt{u}_i, \mathtt{u}_{1-i})$, $\mathtt{mp}(\mathtt{u}_{1-i})$ and $\mathbf{dw}'(\alpha, 0)$. It is easy to check that φ is satisfiable iff $T(\varphi)$ is satisfiable in 1SL3$(*)[\mathbb{Z}, =]$ since the map tr only internalizes the semantics of $\mathrm{LTL}^{\alpha}_{\downarrow}(\mathtt{F},\mathtt{F}^{-1})$ in 1SL3$(*)[\mathbb{Z}, =]$.

Theorem 4.1.1. [BBL09, Theorem 3]
The satisfiability problem for 1SL3$(*)[\mathbb{Z}, =]$ is undecidable.

Other undecidability results about separation logics with data values can be found in [BBL09].

4.1.3 A decidable fragment

In Section 4.1.1, we introduce an extension of 1SL$(*)$ in which data interpreted in $\mathbb{Z}$ are added and can be compared only locally. The translation from 1SL$(*)$ to 1MSOL can be extended to $\mathrm{1SL}^{\mathrm{sdc}}_{<}$, which provides a quite strong new decidability result.

Proposition 4.1.2. [BBL09, Corollary 1] The satisfiability problem for $\mathrm{1SL}^{\mathrm{sdc}}_{<}$ is decidable.

It remains open to characterize a significant class of data domains for which the extension of 1SL$(*)$ with data from those data domains would lead to decidability too. Other decidability results about 1SL$(*)$ extended with data values can be found in [BBL09, Section 5.2].

4.1.4 First-order data logics

As mentioned earlier, there exist many formalisms to specify properties about data words; among them can be found first-order languages. Below, we recall a few

standard definitions. Finally, we sketch the proof of a reduction from an undecidable variant of first-order logic on data words into 1SL2. These results show interesting relationships between first-order logics on data words and separation logics.

Let us present the first-order language $\mathrm{FO2}_{\alpha,\beta}(<, +1, =, \sim, \prec)$ to interpret data words in $([1,\alpha] \times \mathbb{N}^{\beta})^{+}$ following developments from [BDM+11]. Most of the time, a fragment of the full language is needed, but it is helpful to provide the most general definition once and uniformly.

Let $\mathrm{FO2}_{\alpha,\beta}(<, +1, =, \sim, \prec)$ be the set of formulae defined below:

$$\varphi \ ::= \ a(\mathtt{v}) \mid \mathtt{v} \sim_j \mathtt{v} \mid \mathtt{v} \prec_j \mathtt{v} \mid \mathtt{v} < \mathtt{v} \mid \mathtt{v} = 1{+}(\mathtt{v}) \mid \mathtt{v} = \mathtt{v} \mid \neg\varphi \mid \varphi \wedge \varphi \mid \exists\, \mathtt{v}\, \varphi$$

with $\mathtt{v} ::= \mathtt{u}_1 \mid \mathtt{u}_2$, $j \in [1,\beta]$ and $a \in [1,\alpha]$. When $\beta = 0$, this implies that there is no atomic formula using $\sim_j$ or $\prec_j$. We write $\mathrm{FO2}_{\alpha,\beta}(<, +1, =, \sim)$ to denote the restriction of $\mathrm{FO2}_{\alpha,\beta}(<, +1, =, \sim, \prec)$ without $\prec$. Formulae in $\mathrm{FO2}_{\alpha,\beta}(<, +1, =, \sim, \prec)$ are interpreted over data words

$$\mathfrak{dw} = (a^1, \mathfrak{v}_1^1, \ldots, \mathfrak{v}_\beta^1) \cdots (a^L, \mathfrak{v}_1^L, \ldots, \mathfrak{v}_\beta^L)$$

in $([1,\alpha] \times \mathbb{N}^{\beta})^{+}$ via the satisfaction relation $\models_{\mathfrak{f}}$ parameterised by $\mathfrak{f} : \{\mathtt{u}_1, \mathtt{u}_2\} \to [1, L]$ (Boolean clauses are omitted, and $i, i' \in \{1, 2\}$):

$$\begin{array}{lcl}
\mathfrak{dw} \models_{\mathfrak{f}} a(\mathtt{u}_i) & \overset{\text{def}}{\Leftrightarrow} & a^{\mathfrak{f}(\mathtt{u}_i)} = a \\
\mathfrak{dw} \models_{\mathfrak{f}} \mathtt{u}_i \sim_j \mathtt{u}_{i'} & \overset{\text{def}}{\Leftrightarrow} & \mathfrak{v}_j^{\mathfrak{f}(\mathtt{u}_i)} = \mathfrak{v}_j^{\mathfrak{f}(\mathtt{u}_{i'})} \\
\mathfrak{dw} \models_{\mathfrak{f}} \mathtt{u}_i \prec_j \mathtt{u}_{i'} & \overset{\text{def}}{\Leftrightarrow} & \mathfrak{v}_j^{\mathfrak{f}(\mathtt{u}_i)} < \mathfrak{v}_j^{\mathfrak{f}(\mathtt{u}_{i'})} \\
\mathfrak{dw} \models_{\mathfrak{f}} \mathtt{u}_i = \mathtt{u}_{i'} & \overset{\text{def}}{\Leftrightarrow} & \mathfrak{f}(\mathtt{u}_i) = \mathfrak{f}(\mathtt{u}_{i'}) \\
\mathfrak{dw} \models_{\mathfrak{f}} \mathtt{u}_i = 1 + (\mathtt{u}_{i'}) & \overset{\text{def}}{\Leftrightarrow} & \mathfrak{f}(\mathtt{u}_i) = \mathfrak{f}(\mathtt{u}_{i'}) + 1 \\
\mathfrak{dw} \models_{\mathfrak{f}} \mathtt{u}_i < \mathtt{u}_{i'} & \overset{\text{def}}{\Leftrightarrow} & \mathfrak{f}(\mathtt{u}_i) < \mathfrak{f}(\mathtt{u}_{i'}) \\
\mathfrak{dw} \models_{\mathfrak{f}} \exists\, \mathtt{u}_i\, \varphi & \overset{\text{def}}{\Leftrightarrow} & \text{there is } p \in [1, L] \text{ such that } \mathfrak{dw} \models_{\mathfrak{f}[\mathtt{u}_i \mapsto p]} \varphi.
\end{array}$$

A sentence φ in $\mathrm{FO2}_{\alpha,\beta}(<, +1, =, \sim, \prec)$ is satisfiable $\overset{\text{def}}{\Leftrightarrow}$ there is a data word $\mathfrak{dw}$ in $([1,\alpha] \times \mathbb{N}^{\beta})^{+}$ such that $\mathfrak{dw} \models \varphi$ (no need to specify a variable assignment since φ is closed).

Let us recall major results about FO2 on data words.

Proposition 4.1.3.

(I) The satisfiability problem for $\bigcup_{\alpha \geq 1} \mathrm{FO2}_{\alpha,0}(<, +1, =)$ is NEXPTIME-complete [EVW97] (see also [Wei11, Corollary 2.2.4]).

(II) The satisfiability problem for $\bigcup_{\alpha\geq 1} \mathrm{FO2}_{\alpha,1}(<,+1,=,\sim)$ is decidable and closely related to the reachability problem for Petri nets [BDM$^+$11, Dav09, Theorem 3].

(III) The satisfiability problem for $\bigcup_{\alpha\geq 1} \mathrm{FO2}_{\alpha,2}(<,+1,=,\sim)$ is undecidable [BDM$^+$11, Proposition 27][Dav09].

(IV) The satisfiability problem for $\bigcup_{\alpha\geq 1} \mathrm{FO2}_{\alpha,1}(<,+1,=,\sim,\prec)$ is undecidable [BDM$^+$11, Dav09].

Proposition 4.1.3(IV) shall be used in this section but decidability can be regained, as shown in [SZ12], where finite satisfiability of FO2 over data words with a linear order on the positions and a linear order and a corresponding successor relation on the data values shown in ExpSpace [SZ12].

A slightly simpler undecidability proof for 1SL2 can be also obtained from the undecidability of the satisfiability problem for $\bigcup_{\alpha\geq 1} \mathrm{FO2}_{\alpha,1}(<,+1,=,\sim,\prec)$ on data words [BDM$^+$11] (see Theorem 4.1.3(IV)). Let us briefly provide the main ingredients for such a proof. We define a logarithmic-space translation tr as follows. A position $\mathtt{u}$ in the data word corresponds to a location on the main path of the fishbone encoding the same position but for the (unique) part related to the (unique) datum. In the translation process, we freely use macros defined earlier ($i, j \in \{1,2\}$) and tr is homomorphic for Boolean connectives:

$$
\begin{array}{lcl}
tr(\mathtt{u}_i = \mathtt{u}_j) & \stackrel{\text{def}}{=} & \mathtt{u}_i = \mathtt{u}_j \\
tr(\mathtt{u}_i < \mathtt{u}_j) & \stackrel{\text{def}}{=} & \mathtt{reach}(\mathtt{u}_i, \mathtt{u}_j) \wedge \mathtt{u}_i \neq \mathtt{u}_j \\
tr(\mathtt{u}_j = 1 + (\mathtt{u}_i)) & \stackrel{\text{def}}{=} & \top * (\mathtt{reach}'(\mathtt{u}_i, \mathtt{u}_j) \wedge (\sharp \mathtt{u}_i^{+2} = 1) \wedge \neg(\sharp \mathtt{u}_i^{+3} \geq 0)) \\
tr(a(\mathtt{u}_i)) & \stackrel{\text{def}}{=} & \exists\, \mathtt{u}_{3-i}\ (\mathtt{u}_{3-i} \hookrightarrow \mathtt{u}_i) \wedge (\sharp \mathtt{u}_{3-i} = a + 2) \\
tr(\mathtt{u}_i \sim_1 \mathtt{u}_j) & \stackrel{\text{def}}{=} & \sharp \mathtt{u}_i = \sharp \mathtt{u}_j \\
tr(\mathtt{u}_i \prec_1 \mathtt{u}_j) & \stackrel{\text{def}}{=} & \sharp \mathtt{u}_i + 1 \leq \sharp \mathtt{u}_j \\
tr(\exists\, \mathtt{u}_i\ \varphi) & \stackrel{\text{def}}{=} & \exists\, \mathtt{u}_i\ (\sharp \mathtt{u}_i \geq \alpha + 3) \wedge tr(\varphi).
\end{array}
$$

The arithmetical constraints of the form $\sharp \mathtt{u}_i = \sharp \mathtt{u}_j$ and $\sharp \mathtt{u}_i + 1 \leq \sharp \mathtt{u}_j$ are those defined in Section 3.2 when the quantified variables are interpreted by locations on the main path of some $(\alpha, 1)$-fishbone heap.

Lemma 4.1.4. Let φ be a formula in $\mathrm{FO2}_{\alpha,1}(<,+1,=,\sim,\prec)$.

(I) For every data word $\mathfrak{dw}$ in $([1,\alpha]\times\mathbb{N})^+$, $\mathfrak{dw} \models \varphi$ iff $\mathfrak{h}_{\mathfrak{dw}} \models \mathbf{dw}(\alpha,1) \wedge tr(\varphi)$.

(II) Let $\mathfrak{h}$ be a heap such that $\mathfrak{h} \models \mathbf{dw}(\alpha, 1) \wedge tr(\varphi)$, then there is a data word $\mathfrak{dw}$ in $([1, \alpha] \times \mathbb{N})^+$ such that $\mathfrak{h}$ and $\mathfrak{h}_{\mathfrak{dw}}$ are isomorphic and $\mathfrak{dw} \models \varphi$.

As a corollary, the satisfiability problem for 1SL2 is undecidable.

4.2 Interval Temporal Logics

Interval-based temporal logics admit time intervals as first-class objects (instead of time points), and an early and classical study for reasoning about intervals can be found in [All83]. One of the most prominent interval-based logics is Propositional Interval Temporal Logic (PITL), introduced by Ben Moszkowski in [Mos83] for the verification of hardware components. It contains the so-called 'chop' operation that consists of chopping an interval into two subintervals. This is of course reminiscent of separating conjunction in separation logic, and in this section we make a formal statement about this correspondence. Before doing so, it is worth noting that even though most standard point-based temporal logics used in computer science are decidable (CTL, CTL*, ECTL*, etc.), undecidability is much more common in the realm of interval-based temporal logics (see e.g. [BMG$^+$14]). Below, we consider PITL in which propositional variables are interpreted under the **locality condition** and for which decidability is guaranteed but computational complexity is very high. This will allow us to derive similar bounds for 1SL2($*$).

Below, we recall the main definitions about PITL under the locality condition and we explain why formulae from PITL can be faithfully translated into formulae in 1SL2($*$), leading to insights about both formalisms and new complexity results.

4.2.1 The logic PITL

Given $\alpha \geq 1$, we consider the finite alphabet $\Sigma = [1, \alpha]$ and we write PITL_Σ to denote propositional interval temporal logic in which the models are non-empty finite words in Σ^+. We write PITL instead of PITL_Σ when the finite alphabet Σ is clear from the context. Formulae for PITL_Σ are defined according to the following abstract grammar:

$$\varphi \quad ::= \quad a \mid \mathtt{pt} \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \, \mathbf{C} \, \varphi$$

with $a \in \Sigma$. Even though elements of Σ are natural numbers (for the sake of technical convenience), we write a to denote such an arbitrary element in order to

emphasise that a is a letter from a finite alphabet. Roughly speaking, a holds true at word $\mathfrak{w}$ when a is the first letter of $\mathfrak{w}$. Similarly, the atomic formula $\mathtt{pt}$ holds true at a word $\mathfrak{w}$ when the word $\mathfrak{w}$ is only a single letter. The connective $\mathbf{C}$ is the *chop* operator, which chops a word.

Formally, we have a nonempty word $\mathfrak{w} \in \Sigma^+$, its length $|\mathfrak{w}|$, extractions of the ith letter $\mathfrak{w}_i$ where $1 \leq i \leq |\mathfrak{w}|$, and extractions of nonempty subwords $\mathfrak{w}_{i..j} = \mathfrak{w}_i\mathfrak{w}_{i+1}..\mathfrak{w}_j$, where $1 \leq i \leq j \leq |\mathfrak{w}|$. We define a ternary relation chops on words:

$$\mathsf{chops} \stackrel{\text{def}}{=} \{(\mathfrak{w}_1, \mathfrak{w}_2, \mathfrak{w}_3) \mid \exists\, a, \mathfrak{w}', \mathfrak{w}'' \text{ s.t. } \mathfrak{w}_1 = \mathfrak{w}' a \mathfrak{w}'', \mathfrak{w}_2 = \mathfrak{w}' a, \mathfrak{w}_3 = a\mathfrak{w}''\}\,.$$

Observe that when a word $\mathfrak{w}_1$ is chopped into two subwords $\mathfrak{w}_2$ and $\mathfrak{w}_3$, there is an overlap between the last letter of $\mathfrak{w}_2$ and the first letter of $\mathfrak{w}_3$. For instance, $(abb, ab, bb) \in \mathsf{chops}$ but $(ab, a, b) \notin \mathsf{chops}$.

Let us define the satisfaction relation $\models$ for PITL_Σ between a word $\mathfrak{w} \in \Sigma^+$ and a formula φ:

$\mathfrak{w} \models a$	$\stackrel{\text{def}}{\Leftrightarrow}$	the first letter of $\mathfrak{w}$ is a
$\mathfrak{w} \models \mathtt{pt}$	$\stackrel{\text{def}}{\Leftrightarrow}$	the length of $\mathfrak{w}$ is equal to one
$\mathfrak{w} \models \neg\varphi$	$\stackrel{\text{def}}{\Leftrightarrow}$	$\mathfrak{w} \not\models \varphi$
$\mathfrak{w} \models \varphi \wedge \psi$	$\stackrel{\text{def}}{\Leftrightarrow}$	$\mathfrak{w} \models \varphi$ and $\mathfrak{w} \models \psi$
$\mathfrak{w} \models \varphi\,\mathbf{C}\,\psi$	$\stackrel{\text{def}}{\Leftrightarrow}$	there exist words $\mathfrak{w}_1, \mathfrak{w}_2$ such that $\mathsf{chops}(\mathfrak{w}, \mathfrak{w}_1, \mathfrak{w}_2)$, $\mathfrak{w}_1 \models \varphi$ and $\mathfrak{w}_2 \models \psi$.

The satisfiability problem for PITL_Σ consists in checking whether a PITL_Σ formula admits a model satisfying it. Note that the models are *nonempty, finite* words and the satisfaction of a letter on a word depends only on its first letter (the locality condition).

Two examples Consider the alphabet Σ with two distinct letters a and b and the PITL_Σ formula below:

$$(b\,\mathbf{C}\,a)\,\mathbf{C}\,\neg\mathtt{pt}$$

This formula is satisfiable; many words satisfy this formula, for example the word "*bab*"—the top-level chop is satisfied since $ba \models b\,\mathbf{C}\,a$ and $ab \models \neg\mathtt{pt}$. This gives insight on how to specify a lower bound on word length, by applying sufficiently many chops and $\neg\mathtt{pt}$ to force a particular (minimum) length. Of course, $b\,\mathbf{C}\,a$ also enforces a minimum word length (of 2), but constrains also the word content.

Consider another example:

$$\mathtt{pt} \wedge (a\,\mathbf{C}\,b)\,.$$

For this formula to be satisfiable, there must exist a word $\mathfrak{w}$ for which both $\mathfrak{w} \models \mathtt{pt}$ and $\mathfrak{w} \models a \,\mathbf{C}\, b$. This is impossible, as the first implies $|\mathfrak{w}| = 1$, and there is no way to chop a single-letter word into subwords that satisfy both a and b; the formula is unsatisfiable.

Proposition 4.2.1. (see e.g. [Mos83, Mos04]) Given $\alpha \geq 1$ and $\Sigma = [1, \alpha]$, the satisfiability problem for PITL_Σ is decidable, but with $\alpha \geq 2$ is not elementary recursive.

4.2.2 A correspondence between words and heaps

From now on, we use the data word representation from Section 3.1. Thanks to Lemma 3.1.3, we know there is a fishbone heap $\mathfrak{h}_\mathfrak{w}$ corresponding to each nonempty word $\mathfrak{w} \in \Sigma^+$. Let us define a relation $\sim$ that establishes this correspondence between words and their fishbone representations, adding also a correspondence between the empty word and the empty heap:

$$\sim \stackrel{\text{def}}{=} \{(\mathfrak{w}, \mathfrak{h}_\mathfrak{w}) \mid \mathfrak{w} \in \Sigma^+\} \cup \{(\varepsilon, \emptyset)\}\,.$$

Here, observe that:

1. $\sim$ is a bijection between the set of finite words in Σ^* and the set of (equivalence classes of isomorphic) $(\alpha, 0)$-fishbone heaps augmented with the empty heap;
2. so, every word $\mathfrak{w}$ is in $\mathrm{dom}(\sim)$;
3. so, every $(\alpha, 0)$-fishbone heap is in $\mathrm{ran}(\sim)$;
4. so, if $\mathfrak{w} \sim \mathfrak{h}$, $\mathfrak{h}$ is either empty or an $(\alpha, 0)$-fishbone heap; and
5. if $\mathfrak{w} \sim \mathfrak{h}$, then $\mathfrak{w}$ is empty iff $\mathrm{dom}(\mathfrak{h})$ is empty.

In this section, we will only employ $(\alpha, 0)$-fishbone heaps, with $\alpha = \mathrm{card}(\Sigma)$.

The correspondence between finite words in Σ^+ and $(\alpha, 0)$-fishbone heaps satisfies a nice property as far as splitting a word into two disjoint subwords is concerned (which is a slight variant of chopping). Before making a formal statement, let us introduce the following notion.

A **clean cut** of a $(\alpha, 0)$-fishbone heap $\mathfrak{h}$ is a pair of $(\alpha, 0)$-fishbone heaps $(\mathfrak{h}_1, \mathfrak{h}_2)$ such that $\mathfrak{h} = \mathfrak{h}_1 \uplus \mathfrak{h}_2$, and for some words $\mathfrak{w}_1 \sim \mathfrak{h}_1$ and $\mathfrak{w}_2 \sim \mathfrak{h}_2$, we have $\mathfrak{w}_1\mathfrak{w}_2 \sim \mathfrak{h}$. That is, a clean cut is one that neatly cleaves a heap representation of a word

into two subheaps in correspondence with two subwords. Figure 4.1 illustrates examples of a clean cut and a non-clean cut on a fishbone heap. Clockwise from left: the original $(\alpha, 0)$-fishbone heap; a clean cut of the original heap; a non-clean cut of the original heap. Note that clean cuts must result in two $(\alpha, 0)$-fishbone heaps. A non-clean cut may or may not do so; the figure depicts a non-clean cut that does result in two $(\alpha, 0)$-fishbone heaps. Informally, a non-clean cut is one that either results in one subheap (or both) no longer satisfying the $(\alpha, 0)$-fishbone conditions, or that results in subheaps that don't preserve predecessor counts and thus don't represent subwords of the original.

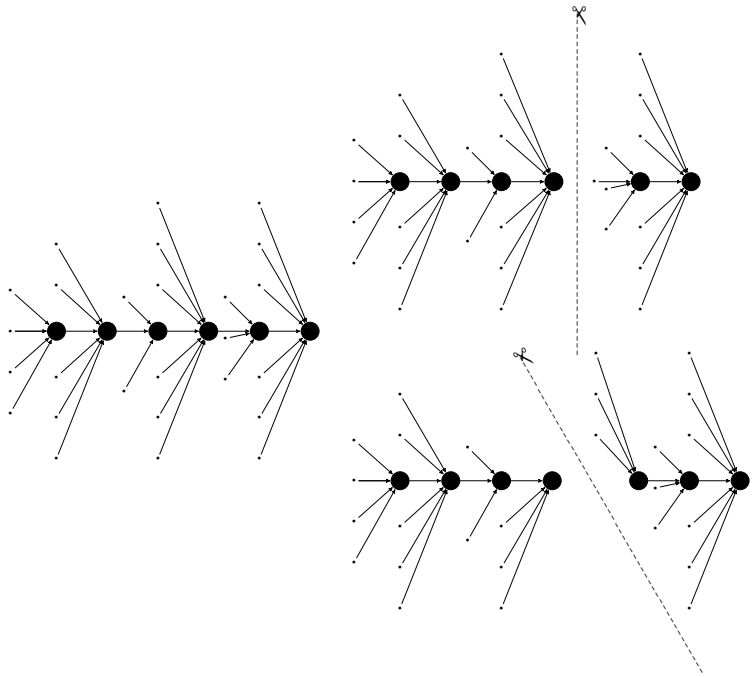


Figure 4.1: A visual depiction of *clean* and *non-clean cuts*.

Lemma 4.2.2. Let $\mathfrak{w} \sim \mathfrak{h}$ with $\mathfrak{w} = \mathfrak{w}_1\mathfrak{w}_2 \in \Sigma^*$. There exist heaps $\mathfrak{h}_1$ and $\mathfrak{h}_2$ such that $\mathfrak{h} = \mathfrak{h}_1 \uplus \mathfrak{h}_2$, $\mathfrak{w}_1 \sim \mathfrak{h}_1$, and $\mathfrak{w}_2 \sim \mathfrak{h}_2$.

Proof. Suppose that $\mathfrak{w} \sim \mathfrak{h}$ and $\mathfrak{w} = \mathfrak{w}_1\mathfrak{w}_2 \in \Sigma^*$. If $\mathfrak{w}_1 = \varepsilon$ or $\mathfrak{w}_2 = \varepsilon$, then the proof is by an easy verification with $\mathfrak{h}$ equal to $\mathfrak{h}_1$ or $\mathfrak{h}_2$ respectively. In particular, if $\mathfrak{w} = \varepsilon$, then $\mathfrak{h}$ is the empty heap and therefore $\mathfrak{w}_1 = \mathfrak{w}_2 = \varepsilon$ and $\mathfrak{h}_1 = \mathfrak{h}_2 = \emptyset$, which satisfies the statement.

Otherwise suppose that $\mathfrak{w} = a_1 \cdots a_K \in \Sigma^+$, $\mathfrak{w}_1 = a_1 \cdots a_{K'} \in \Sigma^+$, $\mathfrak{w}_2 = a_{K'+1} \cdots a_K \in \Sigma^+$ $(K > K')$. Since $\mathfrak{w}$ is nonempty and $\mathfrak{w} \sim \mathfrak{h}$, $\mathfrak{h}$ is a fishbone heap and the main path of $\mathfrak{h}$ is of the form $\mathfrak{l}_1 \to \mathfrak{l}_2 \to \cdots \to \mathfrak{l}_K$ and for every $i \in [1, K]$, $\widetilde{\sharp \mathfrak{l}_i} = a_i + 2$. Let $\mathfrak{h}_1$ be the subheap of $\mathfrak{h}$ whose domain is $\{\mathfrak{l}' \in \mathbb{N} : \mathfrak{l}' \text{ is an ancestor of } \mathfrak{l}_{K'} \text{ in } \mathfrak{h}\}$, and let $\mathfrak{h}_2$ be the unique heap such that $\mathfrak{h} = \mathfrak{h}_1 \uplus \mathfrak{h}_2$. It is easy to show that $\mathfrak{w}_1 \sim \mathfrak{h}_1$ and $\mathfrak{w}_2 \sim \mathfrak{h}_2$. Moreover, it is not difficult to see that $(\mathfrak{h}_1, \mathfrak{h}_2)$ is a clean cut of $\mathfrak{h}$. **QED**

Lemma 4.2.2 entails the following lemma, that will be useful to show the correctness of our reduction from PITL_Σ into 1SL2(∗). It is tailored to the semantics of the chop operator in PITL_Σ.

Lemma 4.2.3. For all letters $a, b \in \Sigma$, words $\mathfrak{w} \in \Sigma^+$ and $\mathfrak{w}', \mathfrak{w}'' \in \Sigma^*$, and heaps $\mathfrak{h}$ such that $\mathfrak{w} \sim \mathfrak{h}$ and $\mathsf{chops}(a\mathfrak{w}, a\mathfrak{w}'b, b\mathfrak{w}'')$, there exist heaps $\mathfrak{h}_1, \mathfrak{h}_2$ such that $\mathfrak{w}'b \sim \mathfrak{h}_1$, $\mathfrak{w}'' \sim \mathfrak{h}_2$, and $\mathfrak{h} = \mathfrak{h}_1 \uplus \mathfrak{h}_2$.

4.2.3 A reduction and its three ways to chop

In this section, we present a satisfiability-preserving translation of PITL_Σ into 1SL2(∗). This translation hinges on the insight that the chop operation is very similar to the separating conjunction in separation logic. However, the correspondence is not an exact one: the connective **C** of PITL_Σ does not cut into disjoint pieces, but rather preserves one letter on both sides, in a sense "duplicating" the letter upon which the chop operates.

To handle this discrepancy, our translation uses the standard separating conjunction on heaps, but internally carries a "ghost letter" (a parameter to the translation) on one side to represent this "lost" letter. In the translation, we denote this ghost letter parameter $a \in \Sigma$. Figure 4.2 illustrates how a chop operation on words is translated into a separation on heaps. It is worth noting that we must always obtain a clean cut from the original heap.

Before presenting the formal definition of the translation, let us present a formula that allows us to perform a clean cut for which one of the subheaps contains all the ancestors of $\mathsf{f}(\mathsf{u})$. Such a formula will be used in the translation and this is the purpose of Lemma 4.2.4 below.

Lemma 4.2.4. Given a fishbone heap $\mathfrak{h}$ and a word $\mathfrak{w}$ such that $\mathfrak{w} \sim \mathfrak{h}$, and an assignment f such that $\mathsf{f}(\mathsf{u})$ is a location on the main path of $\mathfrak{h}$ with $\mathfrak{h} \models_{\mathsf{f}} \mathtt{alloc}(\mathsf{u})$, any pair of heaps $(\mathfrak{h}_1, \mathfrak{h}_2)$ such that $\mathfrak{h} = \mathfrak{h}_1 \uplus \mathfrak{h}_2$, $\mathfrak{h}_1 \models_{\mathsf{f}} \mathbf{dw}(\alpha, 0) \wedge \neg \mathtt{alloc}(\mathsf{u})$, and $\mathfrak{h}_2 \models_{\mathsf{f}} \mathbf{dw}(\alpha, 0) \wedge \sharp \mathsf{u} = 0$, is a clean cut of $\mathfrak{h}$.

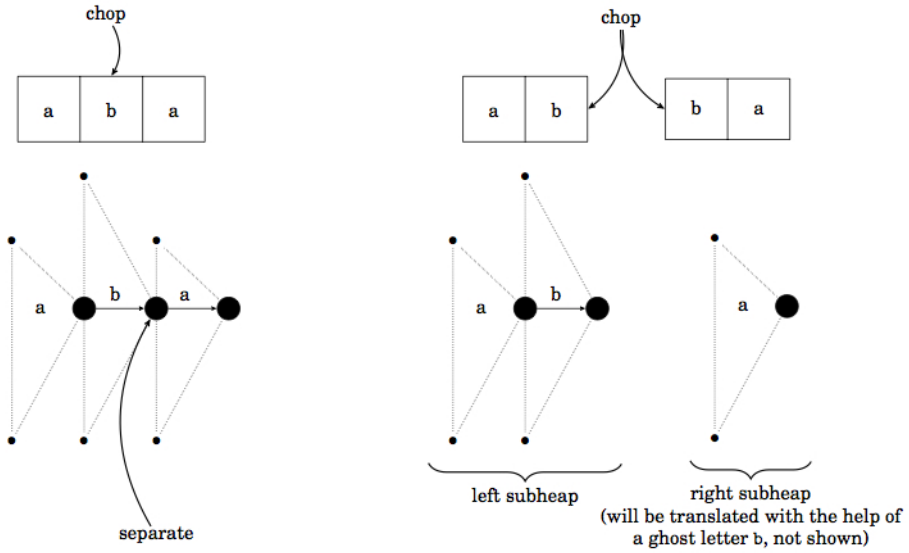


Figure 4.2: The correspondence between PITL's chop '**C**' and separation logic's separating conjunction '$*$' (before and after).

Proof. Since $\mathfrak{h}_1 \models_f \mathbf{dw}(\alpha, 0)$ and $\mathfrak{h}_2 \models_f \mathbf{dw}(\alpha, 0)$, we know the heaps $\mathfrak{h}_1$ and $\mathfrak{h}_2$ are $(\alpha, 0)$-fishbones. Fishbones are single components, so we know that $\mathfrak{h}$ must be separated into exactly two connected components. It remains to analyse precisely how $\mathfrak{h}$ can be separated into two fishbones, and to show that it must be a clean cut.

We know $\mathfrak{l} = f(\mathsf{u})$ is on the main path of $\mathfrak{h}$, so that means $\mathfrak{h} \models_f \sharp\mathsf{u} > 0$. Since $\mathfrak{h}_2 \models_f \sharp\mathsf{u} = 0$, that means $\mathfrak{l}$ must have the same number of predecessors in $\mathfrak{h}$ as it does in $\mathfrak{h}_1$. We know $\mathfrak{h}_1 \models_f \neg\mathtt{alloc}(\mathsf{u})$, and we know $\mathfrak{l}$ has at least one predecessor in $\mathfrak{h}_1$. Therefore, $\mathfrak{l}$ is on the main path of $\mathfrak{h}_1$. We know $\mathfrak{h}_2 \models_f \sharp\mathsf{u} = 0$, so $\mathfrak{l}$ is not on the main path of $\mathfrak{h}_2$. However, it is allocated (since $\mathfrak{h} \models_f \mathtt{alloc}(\mathsf{u})$ and $\mathfrak{h}_1 \models_f \neg\mathtt{alloc}(\mathsf{u})$), so its successor (call the location $\mathfrak{l}'$) is on the main path of $\mathfrak{h}_2$. Let $f' = f[\overline{\mathsf{u}} \mapsto \mathfrak{l}']$. Now, note that $\mathfrak{h} \models_{f'} \mathsf{u} \hookrightarrow \overline{\mathsf{u}}$ and $\mathfrak{h}_1 \not\models_{f'} \mathsf{u} \hookrightarrow \overline{\mathsf{u}}$, and recall that that on a fishbone, no two predecessors of an element can both have predecessors (fb3). Therefore, $\mathfrak{l}'$ must have the same number of predecessors in $\mathfrak{h}_2$ as it did in $\mathfrak{h}$, and none of these predecessors can be on the main path.

Thus $\mathfrak{l}$ is the final location on the main path of $\mathfrak{h}_1$, and $\mathfrak{l}'$ is the first location on the main path of $\mathfrak{h}_2$. Further, $\mathfrak{l}$ has 0 predecessors in $\mathfrak{h}_2$ and the same number of predecessors in $\mathfrak{h}$ and $\mathfrak{h}_1$. $\mathfrak{l}'$ has 0 predecessors in $\mathfrak{h}_1$ and the same number of predecessors in $\mathfrak{h}$ and $\mathfrak{h}_2$.

Putting the above together, since $\mathfrak{l}$ and $\mathfrak{l}'$ are positions on the main path of $\mathfrak{h}$ such that $\mathfrak{h}(\mathfrak{l}) = \mathfrak{l}'$, and since $\mathfrak{l} \notin \mathrm{dom}(\mathfrak{h}_1)$, $\mathfrak{l} \in \mathrm{dom}(\mathfrak{h}_2)$, $\mathfrak{l} \in \mathrm{ran}(\mathfrak{h}_1)$, $\mathfrak{l} \notin \mathrm{ran}(\mathfrak{h}_2)$, we must have a clean cut. **QED**

We reduce a PITL_Σ formula φ to a 1SL2($*$) formula $t(\varphi)$ with the help of the main translation $t(\cdot)$. We use the auxiliary translation map $t_a(\cdot)$ parameterised by a ghost letter a. The three disjuncts in the translation of $\varphi \,\mathbf{C}\, \psi$ correspond to three types of chopping of $\mathfrak{w}$ that leads to three ways of separating the heap $\mathfrak{h}$ (assuming that $\mathfrak{w} \sim \mathfrak{h}$):

1. When $(\mathfrak{w}, a\mathfrak{w}_1 b, b\mathfrak{w}_2) \in \mathsf{chops}$ and the ghost letter is a, the heap $\mathfrak{h}$ is separated into the heap $\mathfrak{h}_1$ with $\mathfrak{w}_1 b \sim \mathfrak{h}_1$ (with ghost letter a) and into the heap $\mathfrak{h}_2$ with $\mathfrak{w}_2 \sim \mathfrak{h}_2$ (with ghost letter b).

2. When $(\mathfrak{w}, \mathfrak{w}, b) \in \mathsf{chops}$ and the ghost letter is a, the heap $\mathfrak{h}$ is separated into itself (again with ghost letter a) and into the empty heap (with ghost letter b).

3. When $(\mathfrak{w}, a, \mathfrak{w}) \in \mathsf{chops}$ and the ghost letter is a, the heap $\mathfrak{h}$ is separated into the empty heap (with ghost letter a) and into itself (again with ghost letter a).

These are the three possible cases and the rest of the translation is quite straightforward.

$$t(\varphi) \stackrel{\text{def}}{=} \Big(\mathbf{dw}(\alpha, 0) \vee \mathtt{emp}\Big) \wedge \bigvee_{a \in \Sigma} t_a(\varphi)$$

The map $t_a(\cdot)$ is homomorphic for Boolean connectives and it is defined as follows for the remaining cases:

$$\begin{array}{lcl}
t_a(b) & \stackrel{\text{def}}{=} & \top \text{ if } b = a \\
t_a(b) & \stackrel{\text{def}}{=} & \bot \text{ if } b \neq a \\
t_a(\mathtt{pt}) & \stackrel{\text{def}}{=} & \mathtt{emp} \\
t_a(\varphi \,\mathbf{C}\, \psi) & \stackrel{\text{def}}{=} & \mathsf{chop1}_a \vee \mathsf{chop2}_a \vee \mathsf{chop3}_a \text{ (see below)} \\
\mathsf{chop1}_a & \stackrel{\text{def}}{=} & \bigvee_{b \in \Sigma} \exists\, \mathtt{u}(\sharp \mathtt{u} = b + 2 \wedge \\
 & & [\mathbf{dw}(\alpha, 0) \wedge \neg \mathtt{alloc}(\mathtt{u}) \wedge t_a(\varphi) \;*\; \mathbf{dw}(\alpha, 0) \wedge \sharp \mathtt{u} = 0 \wedge t_b(\psi)]) \\
\mathsf{chop2}_a & \stackrel{\text{def}}{=} & \bigvee_{b \in \Sigma} (\exists\, \mathtt{u}\ \mathtt{last}(\mathtt{u}) \wedge \sharp \mathtt{u} = b + 2) \wedge [t_a(\varphi) \;*\; (\mathtt{emp} \wedge t_b(\psi))] \\
\mathsf{chop3}_a & \stackrel{\text{def}}{=} & (\mathtt{emp} \wedge t_a(\varphi)) \;*\; t_a(\psi)
\end{array}$$

In full generality, $t_a(\cdot)$ is also parameterised by the alphabet Σ (see the clause for formulae with outermost chop operator $\mathbf{C}$) and the formulae $\mathsf{chop1}_a$, $\mathsf{chop2}_a$, and $\mathsf{chop3}_a$ are parameterised by $\varphi\,\mathbf{C}\,\psi$. Clearly the translation $t(\cdot)$ can only produce 1SL2($*$) formulae, as the right-hand side of each translation step above is in 1SL2($*$). Note also that $t_a(\varphi)$ always produces a closed formula (i.e., without free occurrences of individual variables).

The correctness of the translation is stated below, making completely explicit the role of the ghost letter in the translation process.

Lemma 4.2.5. Let $a \in \Sigma$, $\mathfrak{w} \in \Sigma^*$, and $\mathfrak{h}$ be a heap such that $\mathfrak{w} \sim \mathfrak{h}$. For every PITL_Σ formula φ, we have $a\mathfrak{w} \models \varphi$ iff $\mathfrak{h} \models t_a(\varphi)$.

Since $t_a(\varphi)$ has no free occurrences of individual variables, in Lemma 4.2.5, there is no need to specify what the assignments are. The proof is by structural induction and it is left as Exercise 4.3.

As a result, we obtain a reduction between the satisfiability problems, as stated below.

Lemma 4.2.6. Given $\alpha \geq 1$ and $\Sigma = [1, \alpha]$, a PITL_Σ formula φ is satisfiable if and only if the 1SL2($*$) formula $t(\varphi)$ is satisfiable.

Proof. ($\Rightarrow$) Suppose that φ is satisfiable. This means that there exists a nonempty word $\mathfrak{w}$ such that $\mathfrak{w} \models \varphi$. The word $\mathfrak{w}$ can be written in the form $\mathfrak{w} = a\mathfrak{w}'$ for some letter a. If $\mathfrak{w}' = \varepsilon$, we have $\mathfrak{w}' \sim \emptyset$ and by Lemma 4.2.5, we have $\emptyset \models \mathtt{emp} \wedge t_a(\varphi)$. So $\emptyset \models t(\varphi)$ and therefore $t(\varphi)$ is satisfiable. If $\mathfrak{w}' \neq \varepsilon$, then there is a $(\alpha, 0)$-fishbone heap $\mathfrak{h}'$ such that $\mathfrak{w}' \sim \mathfrak{h}'$. By Lemma 4.2.5, we have $\mathfrak{h}' \models \mathbf{dw}(\alpha, 0) \wedge t_a(\varphi)$. So $\mathfrak{h}' \models t(\varphi)$ and therefore $t(\varphi)$ is satisfiable.

($\Leftarrow$) If $t(\varphi)$ is satisfiable, then there exists a heap $\mathfrak{h}$ such that

$$\mathfrak{h} \models (\mathbf{dw}(\alpha, 0) \vee \mathtt{emp}) \wedge \bigvee_{a \in \Sigma} t_a(\varphi)\,.$$

If $\mathfrak{h} \models \mathtt{emp} \wedge t_a(\varphi)$ for some letter a, then by Lemma 4.2.5, we have $a \models \varphi$. Otherwise, if $\mathfrak{h} \models \mathbf{dw}(\alpha, 0) \wedge t_a(\varphi)$, then $\mathfrak{h}$ is an $(\alpha, 0)$-fishbone heap by Lemma 3.1.2 and then there is a word $\mathfrak{w}$ such that $\mathfrak{w} \sim \mathfrak{h}$ such that by Lemma 4.2.5, we have $a\mathfrak{w} \models \varphi$. In both cases, φ is a satisfiable formula in PITL_Σ. **QED**

Theorem 4.2.7. The satisfiability problem for 1SL2($*$) is decidable but not elementary recursive.

Proof. Satisfiability for PITL_Σ is known to be decidable with non-elementary complexity when Σ has at least two elements, see e.g. [Mos83, Mos04], and 1SL($*$) is decidable [BDL12] (see also Section 4.4). From the correctness of our translation $t(\cdot)$ of PITL_Σ to 1SL2($*$) (Lemma 4.2.6), we then conclude that 1SL2($*$) is decidable but not elementary recursive. Note that the map $t(\cdot)$ may require exponential time and space in the size of the input formula in the worst-case but this is still fine to establish that 1SL2($*$) is not elementary recursive, since this adds only a single exponential. **QED**

As mentioned earlier, Theorem 4.2.7 refines the non-elementarity result for 1SL($*$) established in [BDL12]. So, solving the satisfiability problem for 1SL2($*$) requires time bounded below by towers of exponentials of height that depend on the formula size, see e.g. [Sch15].

Remark. The reduction from PITL to 1SL2($*$) [DD15b] allows us to underline the common features of both formalisms. To our knowledge, this is the first time that the similarity has been turned into a concrete, interesting result. The possibility to relate separation logic and interval temporal logic has been already envisaged by Tony Hoare, see e.g. [Zho08] (We thank Ben Moszkowski for pointing us to this work.)

However, non-elementarity of 1SL2($*$) can be established in a slightly different way as explained below. First, non-elementarity of PITL is due to Dexter Kozen (see e.g. [Mos04, Appendix A.3]) (We thank Ben Moszkowski for pointing us to this fact.), and the proof is by reduction from the nonemptiness problem of regular expressions built over a binary alphabet with union, concatenation and complement [Sto74]. Nonelementarity of 1SL2($*$) can be obtained by defining a similar reduction, but this is of course less insightful to understand the relationships between interval temporal logic and separation logic. Alternatively, it is also possible to consider the variant of PITL in which the chop operator does not share a letter, since this variant is of identical expressive power and complexity. In that way, we may avoid the introduction of the ghost letter but at the cost of introducing empty models (which may occur when chopping has no sharing) and of using a less standard interval temporal logic. So, the current reduction from PITL is quite an attractive option to relate the logics. Finally, as noted in [Mos04, Appendix A], complexity results about PITL presented in [Mos83] were obtained in collaboration with Joseph Halpern.

In Section 4.3.2 below, we establish an even stronger result about 1SL2($*$) (see Theorem 4.3.5). The proof uses the same principles as for the proof of Theorem 4.2.7 and we only need to express the properties in *modal lingua*.

4.3 Modal Logics

4.3.1 A modal logic for heaps

Let us introduce a new modal logic that is closely related to 1SL2. Modal Logic for Heaps (MLH) is a multimodal logic in which models are exactly heap graphs and it does not contain propositional variables (as 1SL does not contain unary predicate symbols). In a sense, it is similar to Hennessy-Milner logic HML [HM80] in which the only atomic formulae are truth constants. However, the language contains modal operators and separating connectives, which is a feature shared with the logics defined in [CG13]. We define below the formulae of the modal logic MLH.

$$\varphi \quad ::= \quad \bot \mid \neg\varphi \mid \varphi \wedge \varphi \mid \Diamond\varphi \mid \Diamond^{-1}\varphi \mid \langle\neq\rangle\varphi \mid \langle\star\rangle\varphi \mid \varphi * \varphi \mid \varphi \mathrel{-\!\!*} \varphi.$$

There are no quantified variables involved in formulae, which is a feature shared with most known propositional modal logics, see e.g. [BdRV01]. We write MLH($*$) to denote the fragment of MLH without the magic wand operator $\mathrel{-\!\!*}$.

A **model for MLH** $\mathfrak{M}$ is a pair $(\mathbb{N}, \Re)$ such that $\Re$ is a binary relation on $\mathbb{N}$ that is finite and functional. Otherwise said, the models for MLH are heap graphs (when the heaps encode a unique record field, i.e. $k = 1$). Models for MLH could be alternatively defined as heaps but we prefer to stick to the most usual presentation for modal logics with frames. The satisfaction relation $\models$ is defined below and it provides a standard semantics for the modal operators and separating connectives (we omit the clauses for Boolean connectives):

never $\mathfrak{M}, \mathfrak{l} \models \bot$

$\mathfrak{M}, \mathfrak{l} \models \Diamond\varphi \quad \overset{\text{def}}{\Leftrightarrow}$ there is $\mathfrak{l}'$ such that $(\mathfrak{l}, \mathfrak{l}') \in \Re$ and $\mathfrak{M}, \mathfrak{l}' \models \varphi$

$\mathfrak{M}, \mathfrak{l} \models \Diamond^{-1}\varphi \quad \overset{\text{def}}{\Leftrightarrow}$ there is $\mathfrak{l}'$ such that $(\mathfrak{l}', \mathfrak{l}) \in \Re$ and $\mathfrak{M}, \mathfrak{l}' \models \varphi$

$\mathfrak{M}, \mathfrak{l} \models \langle\star\rangle\varphi \quad \overset{\text{def}}{\Leftrightarrow}$ there is $\mathfrak{l}'$ such that $(\mathfrak{l}, \mathfrak{l}') \in \Re^*$ and $\mathfrak{M}, \mathfrak{l}' \models \varphi$
($\Re^*$ is the reflexive and transitive closure of $\Re$)

$\mathfrak{M}, \mathfrak{l} \models \langle\neq\rangle\varphi \quad \overset{\text{def}}{\Leftrightarrow}$ there is $\mathfrak{l}' \neq \mathfrak{l}$ such that $\mathfrak{M}, \mathfrak{l}' \models \varphi$

$\mathfrak{M}, \mathfrak{l} \models \varphi_1 * \varphi_2 \quad \overset{\text{def}}{\Leftrightarrow}$ $(\mathbb{N}, \Re_1), \mathfrak{l} \models \varphi_1$ and $(\mathbb{N}, \Re_2), \mathfrak{l} \models \varphi_2$
for some partition $\{\Re_1, \Re_2\}$ of $\Re$

$\mathfrak{M}, \mathfrak{l} \models \varphi_1 \mathrel{-\!\!*} \varphi_2 \quad \overset{\text{def}}{\Leftrightarrow}$ for all models $\mathfrak{M}' = (\mathbb{N}, \Re')$ such that $\Re \cap \Re' = \emptyset$
and $\Re \cup \Re'$ is functional,
$\mathfrak{M}', \mathfrak{l} \models \varphi_1$ implies $(\mathbb{N}, \Re \cup \Re'), \mathfrak{l} \models \varphi_2$

We use the following standard abbreviations:

$$
\begin{array}{lcl}
\langle U\rangle\varphi & \stackrel{\text{def}}{=} & \varphi \vee \langle\neq\rangle\, \varphi \\
[U]\varphi & \stackrel{\text{def}}{=} & \neg\langle U\rangle\neg\varphi \\
[\neq]\varphi & \stackrel{\text{def}}{=} & \neg\langle\neq\rangle\neg\varphi \\
\Diamond^{-1}_{\geq k}\top & \stackrel{\text{def}}{=} & \Diamond^{-1}\top * \cdots * \Diamond^{-1}\top \ (k \geq 1 \text{ times}) \\
\Diamond^{-1}_{\leq k-1}\top & \stackrel{\text{def}}{=} & \neg\Diamond^{-1}_{\geq k}\top \\
\Diamond^{-1}_{[k_1,k_2]}\top & \stackrel{\text{def}}{=} & \Diamond^{-1}_{\geq k_1}\top \wedge \Diamond^{-1}_{\leq k_2}\top \\
\Diamond^{-1}_{=k}\top & \stackrel{\text{def}}{=} & \Diamond^{-1}_{\geq k}\top \wedge \Diamond^{-1}_{\leq k}\top
\end{array}
$$

Whenever φ is already in MLH($*$), these abbreviations allow to remain in MLH($*$).

A formula φ is satisfiable whenever there is a model $\mathfrak{M}$ and a location $\mathfrak{l}$ such that $\mathfrak{M}, \mathfrak{l} \models \varphi$. The satisfiability problem for MLH is therefore defined as any such problem for modal logics.

Note that MLH has forward and backward modalities as in Prior's tense logic (see e.g. [Pri67]), the inequality modal operator (see e.g. [dR92]) and the transitive closure operator as in PDL (see e.g. [HKT00]). The most non-standard feature of MLH is certainly the presence of the separating connectives.

It is possible to design a relational translation from MLH formulae into 1SL2 formulae by recycling variables (only $\mathtt{u}_1$ and $\mathtt{u}_2$ are used, so $i \in \{1, 2\}$) and tr is homomorphic for the connectives $\neg$, $\wedge$, $*$ and $-\!\!*$:

$$
\begin{array}{lcl}
tr(\bot, \mathtt{u}_i) & \stackrel{\text{def}}{=} & \bot \\
tr(\Diamond\, \varphi, \mathtt{u}_i) & \stackrel{\text{def}}{=} & \exists\, \mathtt{u}_{3-i}\, (\mathtt{u}_i \hookrightarrow \mathtt{u}_{3-i}) \wedge tr(\varphi, \mathtt{u}_{3-i}) \\
tr(\Diamond^{-1}\, \varphi, \mathtt{u}_i) & \stackrel{\text{def}}{=} & \exists\, \mathtt{u}_{3-i}\, (\mathtt{u}_{3-i} \hookrightarrow \mathtt{u}_i) \wedge tr(\varphi, \mathtt{u}_{3-i}) \\
tr(\langle\neq\rangle\, \varphi, \mathtt{u}_i) & \stackrel{\text{def}}{=} & \exists\, \mathtt{u}_{3-i}\, (\mathtt{u}_i \neq \mathtt{u}_{3-i}) \wedge tr(\varphi, \mathtt{u}_{3-i}) \\
tr(\langle\star\rangle\, \varphi, \mathtt{u}_i) & \stackrel{\text{def}}{=} & \exists\, \mathtt{u}_{3-i}\, \mathtt{reach}(\mathtt{u}_i, \mathtt{u}_{3-i}) \wedge tr(\varphi, \mathtt{u}_{3-i}).
\end{array}
$$

The formulae of the form $\mathtt{reach}(\mathtt{u}_i, \mathtt{u}_{i'})$ have been introduced in Section 1.2.2 and states the reachability of $\mathtt{u}_{i'}$ from $\mathtt{u}_i$.

Lemma 4.3.1. A formula φ in MLH is satisfiable iff $\exists\, \mathtt{u}_1\ tr(\varphi, \mathtt{u}_1)$ is satisfiable in 1SL2. Moreover, if φ is in MLH($*$), then $\exists\, \mathtt{u}_1\ tr(\varphi, \mathtt{u}_1)$ is in 1SL2($*$).

Proof. (sketch) The proof is obtained as an obvious adaptation of the proof for the relational translation from modal logic K into FO2, see e.g. [Mor76, vB76, BdRV01]. Indeed, the models $(\mathbb{N}, \mathfrak{R})$ for MLH are heap graphs and therefore formulae in 1SL2 can be equivalently interpreted on MLH models; for instance, we get $(\mathbb{N}, \mathfrak{R}) \models_{\mathfrak{f}} \mathtt{u}_1 \hookrightarrow \mathtt{u}_2$ iff $(\mathfrak{f}(\mathtt{u}_1), \mathfrak{f}(\mathtt{u}_2)) \in \mathfrak{R}$. Similarly, $(\mathbb{N}, \mathfrak{R}) \models_{\mathfrak{f}} \varphi_1 -\!\!* \varphi_2$

iff for all MLH models $(\mathbb{N}, \mathfrak{R}')$ such that $(\mathbb{N}, \mathfrak{R} \cup \mathfrak{R}')$ is an MLH model too and $(\mathbb{N}, \mathfrak{R}') \models_{\dagger} \varphi_1$, we have $(\mathbb{N}, \mathfrak{R} \cup \mathfrak{R}') \models_{\dagger} \varphi_2$.

Note that $\mathtt{u}_j$ is the only free variable in $tr(\varphi, \mathtt{u}_j)$. The standard translation tr is semantically faithful in the following sense: for all MLH models $(\mathbb{N}, \mathfrak{R})$, $\mathfrak{l} \in \mathbb{N}$ and formulae φ in MLH, we have $(\mathbb{N}, \mathfrak{R}), \mathfrak{l} \models \varphi$ iff $(\mathbb{N}, \mathfrak{R}) \models_{[\mathtt{u}_1 \mapsto \mathfrak{l}]} tr(\varphi, \mathtt{u}_1)$. This is sufficient to establish Lemma 4.3.1.

We show that for all $i \in \{1, 2\}$, for all formulae ψ in MLH, for all MLH models $\mathfrak{M} = (\mathbb{N}, \mathfrak{R})$ and for $\mathfrak{l} \in \mathbb{N}$, we have $\mathfrak{M}, \mathfrak{l} \models \psi$ iff $\mathfrak{M} \models_{[\mathtt{u}_i \mapsto \mathfrak{l}]} tr(\psi, \mathtt{u}_i)$. The proof is by structural induction. The base case for $\bot$ and the cases in the induction step for the Boolean connectives are straightforward. By way of example, let us provide the cases in the induction step for $\psi = \langle \star \rangle \psi'$ and for $\psi = \psi_1 * \psi_2$. The proof for the other cases is similar and quite standard.
Case $\psi = \langle \star \rangle \psi'$. The following are equivalent:

- $\mathfrak{M}, \mathfrak{l} \models \psi$,
- $\mathfrak{M}, \mathfrak{l}' \models \psi'$ for some $\mathfrak{l}' \in \mathfrak{R}^*(\mathfrak{l})$ (by definition of $\models$),
- $\mathfrak{M} \models_{[\mathtt{u}_{3-i} \mapsto \mathfrak{l}']} tr(\psi', \mathtt{x}_{3-i})$ for some $\mathfrak{l}' \in \mathbb{N}$ such that $\mathfrak{l}' \in \mathfrak{R}^*(\mathfrak{l})$ (by the induction hypothesis),
- $\mathfrak{M} \models_{[\mathtt{u}_i \mapsto \mathfrak{l}]} \exists\, \mathtt{u}_{3-i}\ \mathtt{reach}(\mathtt{u}_i, \mathtt{u}_{3-i}) \wedge tr(\psi', \mathtt{x}_{3-i})$ (by definition of $\models$ in 1SL2 and by the fact that `reach` is the reachability predicate),
- $\mathfrak{M} \models_{[\mathtt{u}_i \mapsto \mathfrak{l}]} tr(\psi, \mathtt{u}_i)$ (by definition of tr).

Case $\psi = \psi_1 * \psi_2$. The following are equivalent:

- $\mathfrak{M}, \mathfrak{l} \models \psi$,
- $(\mathbb{N}, \mathfrak{R}_1), \mathfrak{l} \models \psi_1$ and $(\mathbb{N}, \mathfrak{R}_2), \mathfrak{l} \models \psi_2$ for some partition $\{\mathfrak{R}_1, \mathfrak{R}_2\}$ of $\mathfrak{R}$, (by definition of $\models$ in MLH),
- $(\mathbb{N}, \mathfrak{R}_1) \models_{[\mathtt{u}_i \mapsto \mathfrak{l}]} tr(\psi_1, \mathtt{u}_i)$ and $(\mathbb{N}, \mathfrak{R}_2) \models_{[\mathtt{u}_i \mapsto \mathfrak{l}]} tr(\psi_2, \mathtt{u}_i)$ for some partition $\{\mathfrak{R}_1, \mathfrak{R}_2\}$ of $\mathfrak{R}$, (by the induction hypothesis),
- $\mathfrak{M} \models_{[\mathtt{u}_i \mapsto \mathfrak{l}]} tr(\psi_1, \mathtt{u}_i) * tr(\psi_2, \mathtt{u}_i)$ (by definition of the satisfaction relation in 1SL2)
- $\mathfrak{M} \models_{[\mathtt{u}_i \mapsto \mathfrak{l}]} tr(\psi, \mathtt{u}_i)$ (by definition of tr). **QED**

Modal logic MLH can be viewed as a fragment of 1SL2. Any formula $\psi_1 * \psi_2$ [resp. $\psi_1 \mathrel{-\!\!*} \psi_2$] in $tr(\varphi, \mathtt{u}_1)$ has at most one free variable. A similar restriction can be found in monodic fragments for first-order temporal logics, see e.g. [DFL02].

Since MLH($*$) can be translated into 1SL2($*$) and 1SL($*$) is decidable [BDL12, Corollary 3.3] (see also Section 4.4), we get decidability of MLH($*$) as a corollary.

Corollary 4.3.2. The satisfiability problem for MLH($*$) is decidable.

Note that to be more uniform, we could have added to the modal language the converse operators $\langle \neq \rangle^{-1}$ and $\langle \star \rangle^{-1}$. However, since the inequality relation is symmetric, $\langle \neq \rangle^{-1}\varphi$ is logically equivalent to $\langle \neq \rangle\varphi$. The above translation can be obviously extended with the modal operator $\langle \star \rangle^{-1}$ and therefore decidability holds also for this extension. However, we have introduced MLH mainly to establish non-elementarity of MLH($*$) (shown below), refining the result for 1SL2($*$). We did not include $\langle \star \rangle^{-1}$ because the proof of non-elementarity result does not require it. By contrast, we do not know whether the satisfiability problem for MLH is decidable. As far as we know, the characterisation of the computational complexity of MLH without separating connectives is open too. This corresponds to a fragment of deterministic PDL with (restricted) graded modalities and inequality modality.

4.3.2 A refinement with the modal fragment of 1SL2($*$)

In this section, we show that the satisfiability problem for MLH($*$) is decidable but it is not elementary recursive. Decidability is due to the fact that the standard translation leads to formulae in 1SL2($*$), see Section 4.3.1. In order to establish the lower bound, we express in MLH($*$) all the properties that were useful to translate PITL_Σ formulae into 1SL2($*$). For instance, note that the empty heap is the only heap validating the formula $([U]\neg\Diamond\top)$. Similarly, a location with at least one predecessor and with no successor (for instance, last location on the main path in a fishbone heap) satisfies the formula $(\Diamond^{-1}\top \wedge \neg\Diamond\top)$.

More interestingly, the formula in 1SL2($*$) characterising the (α, β)-fishbone heaps has a modal counterpart. Let us consider the following formulae.

- The formula $\varphi^{\Box}_{\mathrm{fb}}$ defined below is designed exactly as the formula φ_{fb} (see Section 3.1).

$$(\langle U \rangle \Diamond \top) \wedge$$

$$\langle U \rangle((\Diamond^{-1}\top \wedge \neg\Diamond\top) \wedge [\neq]\neg(\Diamond^{-1}\top \wedge \neg\Diamond\top)) \wedge [U](\Diamond\top \Rightarrow \langle \star \rangle(\Diamond^{-1}\top \wedge \neg\Diamond\top)) \wedge$$

$$(\neg\langle U\rangle(\Diamond^{-1}\Diamond^{-1}\top * \Diamond^{-1}\Diamond^{-1}\top)).$$

This is a faithful translation except that we use the specification language MLH(∗).

- The formula $\varphi^{\Box}_{(C1)}$ defined below is also designed exactly as the formula $\varphi_{(C1)}$ (see Section 3.1).

$$\langle U\rangle((\Diamond^{-1}\top) \wedge (\neg\Diamond^{-1}\Diamond^{-1}\top) \wedge \Diamond^{-1}_{[3,\alpha+3]}\top).$$

- The formula $\varphi^{\Box}_{(C2)}$ is equal to $[U](\Diamond^{-1}_{[3,\alpha+3]}\top \Rightarrow \bigwedge_{i\in[1,\beta]} \overbrace{\Diamond\cdots\Diamond}^{i \text{ times}} \Diamond^{-1}_{\geq\alpha+3}\top)$.

- The formula $\varphi^{\Box}_{(C3)}$ is defined below:

$$[U](\Diamond^{-1}_{[3,\alpha+3]}\top \Rightarrow (\neg \overbrace{\Diamond\cdots\Diamond}^{\beta+1 \text{ times}} \top) \vee \overbrace{\Diamond\cdots\Diamond}^{\beta+1 \text{ times}}(\Diamond^{-1}_{[3,\alpha+3]}\top)).$$

We write $\mathbf{dw}^{\Box}(\alpha,\beta)$ to denote the formula $\varphi^{\Box}_{\text{fb}} \wedge \varphi^{\Box}_{(C1)} \wedge \varphi^{\Box}_{(C2)} \wedge \varphi^{\Box}_{(C3)}$. It specifies the shape of the encoding of data words in $([1,\alpha] \times \mathbb{N}^{\beta})^{+}$ as stated below. Note that since $\mathbf{dw}^{\Box}(\alpha,\beta)$ is a Boolean combination of formulae whose outermost connectives are $[U]$ or $\langle U\rangle$, then $\mathbf{dw}^{\Box}(\alpha,\beta)$ holds true at some location iff $\mathbf{dw}^{\Box}(\alpha,\beta)$ holds true at any location.

Lemma 4.3.3. Let $\mathfrak{M} = (\mathbb{N}, \mathfrak{R})$ be a model for MLH. $\mathfrak{M}, \mathfrak{l} \models \mathbf{dw}^{\Box}(\alpha,\beta)$ for some location $\mathfrak{l}$ iff $\mathfrak{M}$ is the graph of an (α,β)-fishbone heap.

Again, the proof is by an easy verification by using Lemma 3.1.1 and the correspondence between condition (Ci) and the formula $\varphi^{\Box}_{(Ci)}$. In the rest of this section we are back to the case $\beta = 0$.

Given a formula φ in PITL_{Σ} with $\Sigma = [1,\alpha]$, we define a modal formula $t(\varphi)^{\Box}$ such that φ is satisfiable iff $t(\varphi)^{\Box}$ is satisfiable. Actually, the modal formula $t(\varphi)^{\Box}$ will express exactly the same properties as in the translation into 1SL2(∗). For instance, $t(\varphi)^{\Box}$ is precisely the formula below:

$$(\mathbf{dw}^{\Box}(\alpha,0) \vee ([U]\neg\Diamond\top)) \wedge (\bigvee_{a\in\Sigma} t(\varphi)^{\Box}_{a})$$

The formula $t(\varphi)^{\Box}_{a}$ is defined inductively as follows.

- $t(a)^{\Box}_a \stackrel{\text{def}}{=} \top$ and $t(b)^{\Box}_a \stackrel{\text{def}}{=} \bot$ for every letter $b \in \Sigma \setminus \{a\}$.
- $t(\cdot)^{\Box}_a$ is homomorphic for Boolean connectives.
- $t(\mathtt{pt})^{\Box}_a \stackrel{\text{def}}{=} ([U]\neg\Diamond\top)$.
- The formula $t(\varphi \,\mathbf{C}\, \psi)^{\Box}_a$ is defined as $\mathsf{chop1}^{\Box}_a \vee \mathsf{chop2}^{\Box}_a \vee \mathsf{chop3}^{\Box}_a$ where:
  - $\mathsf{chop1}^{\Box}_a \stackrel{\text{def}}{=} \bigvee_{b\in\Sigma}\langle U\rangle((\Diamond^{-1}_{=b+2}\top \wedge \mathbf{dw}^{\Box}(\alpha,0) \wedge \neg\Diamond\top \wedge t(\varphi)^{\Box}_a) * (\mathbf{dw}^{\Box}(\alpha,0) \wedge \neg\Diamond^{-1}\top \wedge t(\psi)^{\Box}_b))$,
  - $\mathsf{chop2}^{\Box}_a \stackrel{\text{def}}{=} (\bigvee_{b\in\Sigma}\langle U\rangle\ ((\Diamond^{-1}\top \wedge \neg\Diamond\top) \wedge \Diamond^{-1}_{=b+2}\top) \wedge (t(\varphi)^{\Box}_a * (t(\psi)^{\Box}_b \wedge ([U]\neg\Diamond\top))))$,
  - $\mathsf{chop3}^{\Box}_a \stackrel{\text{def}}{=} ((t(\varphi)^{\Box}_a \wedge ([U]\neg\Diamond\top)) * t(\psi)^{\Box}_a)$.

Lemma 4.3.4. Let $\alpha \geq 1$, $\Sigma = [1,\alpha]$, φ be a PITL_Σ formula and $t(\varphi)^{\Box}$ be its translation in MLH. We have φ is satisfiable iff $t(\varphi)^{\Box}$ is satisfiable.

The proof goes as in the case for the direct translation into 1SL2(∗) since the modal subformulae express exactly the same properties. Therefore, we can refine Theorem 4.2.7 as follows.

Theorem 4.3.5. The satisfiability problem for MLH(∗) is decidable but not elementary recursive.

Interestingly, we do not know the decidability status for full MLH (i.e., with the magic wand operator).

4.4 Monadic Second-Order Logic

We have seen that 1SL(∗) is decidable with non-elementary recursive complexity (actually two variables suffice). Below, we briefly explain why decidability comes from the decidability of 1MSOL and we provide a property that can separate the expressive power of 1SL(∗) and 1MSOL.

First, note that 1MSOL on memory states with one record field is decidable by taking advantage of [Rab69, BGG97]. Indeed, the weak monadic second-order theory of unary functions is the theory over structures of the form $(\mathfrak{D}, \mathsf{f}, =)$ where $\mathfrak{D}$ is a countable domain, f is a unary function, and = is equality. Weakness means that quantifications are over finite sets.

This theory is decidable, see e.g. [BGG97, Corollary 7.2.11]. Since it is possible to express that $\mathfrak{D}$ is infinite and to simulate that $\mathfrak{f}$ is a partial function with finite domain, one can specify that $(\mathfrak{D}, \mathfrak{f}, =)$ augmented with a first-order valuation is isomorphic to a heap. It is then possible to define a simple translation $tr_{\mathtt{P}}(.)$, computable in logarithmic space, such that a 1MSOL sentence φ is satisfiable iff

$$\overbrace{(\neg\exists\, \mathtt{P}\ \forall \mathtt{u}\ \mathtt{P}(\mathtt{u}))}^{\text{infinity}} \wedge \exists\, \mathtt{P}\ tr_{\mathtt{P}}(\varphi)$$

is satisfiable in the weak monadic second-order theory of one unary function. See details in [BDL12]. Using a similar technique, it is possible to translate 1SL(∗) into 1MSOL. Any formula φ in 1SL(∗) is satisfiable iff

$$\exists\, \mathtt{P}\ (\forall \mathtt{u}\ \mathtt{P}(\mathtt{u}) \Leftrightarrow\ (\exists \mathtt{u}'\ \mathtt{u} \hookrightarrow \mathtt{u}')) \wedge tr_{\mathtt{P}}(\varphi)$$

is satisfiable where $tr_{\mathtt{P}}(\cdot)$ is defined with the following clauses:

$$\begin{array}{lcl} tr_{\mathtt{P}}(\mathtt{u} \hookrightarrow \mathtt{u}') & \stackrel{\text{def}}{=} & \mathtt{P}(\mathtt{u}) \wedge \mathtt{u} \hookrightarrow \mathtt{u}' \\ tr_{\mathtt{P}}(\mathtt{u} = \mathtt{u}') & \stackrel{\text{def}}{=} & \mathtt{u} = \mathtt{u}' \\ tr_{\mathtt{P}}(\varphi * \psi) & \stackrel{\text{def}}{=} & \exists\, \mathtt{Q}, \mathtt{Q}'\ (\mathtt{P} = \mathtt{Q} \uplus \mathtt{Q}') \wedge tr_{\mathtt{Q}}(\varphi) \wedge tr_{\mathtt{Q}'}(\psi) \end{array}$$

where $\mathtt{P} = \mathtt{Q} \uplus \mathtt{Q}'$ is an abbreviation for $\forall \mathtt{u}\, (\mathtt{P}(\mathtt{u}) \Leftrightarrow (\mathtt{Q}(\mathtt{u}) \vee \mathtt{Q}'(\mathtt{u}))) \wedge \neg(\mathtt{Q}(\mathtt{u}) \wedge \mathtt{Q}'(\mathtt{u}))$. We leave out the Boolean connectives and first-order quantification, for which $tr_{\mathtt{P}}$ is homomorphic.

Theorem 4.4.1. [BDL12, Corollary 3.3] The satisfiability problem for 1SL(∗) is decidable.

As conjectured in [BDL08], we have the following separation result.

Proposition 4.4.2. [AD09, Corollary 5.3] (see also [Ant10]) 1SL(∗) is strictly less expressive than 1MSOL.

A standard tool to show non-expressibility in first-order logic or in second-order logic is to use Ehrenfeucht-Fraïssé games, see e.g. [Lib04] (called **EF-games** in the sequel). These games have been adapted for some versions of separation logic, see e.g. [AD09, Ant10] based on similar games on spatial logics [DGG04, Mar06, DGG07]. In [AD09, Ant10], using EF-games, it is shown that there is no formula in 1SL(∗) that characterises the forests of binary trees such that there is one binary tree whose number of leaves is a multiple of 3.

Even though the principle of the method with EF-games is standard, the proof is quite complex and tedious, since it requires designing two families of heaps, to define an adequate strategy for the `Duplicator` player and to show that the strategy does the job, see e.g. [Ant10] for most of the details as well as for additional bibliographical references. In particular, `Duplicator` has a winning strategy for a game on $(\mathfrak{h}_1, \mathfrak{h}_2)$ with rank r iff $\mathfrak{h}_1$ and $\mathfrak{h}_2$ agree on all formulae of 1SL$(*)$ of rank r. See [Ant10] for more details about the notion of game and rank, for instance.

By way of example, we explain below why the above-mentioned property can be expressed in 1MSOL. First, let us express in 1SL$(*)$ that the heap is a forest of binary trees, which entails that this can be stated in 1MSOL too. It is sufficient to state that every location has at most two predecessors and every location $\mathtt{l}$ can reach a non-allocated location (the root of the tree to which $\mathtt{l}$ belongs too if $\mathtt{l}$ is in the heap domain):

$$\forall\, \mathtt{u}\ (\sharp \mathtt{u} \leq 2 \wedge \exists\, \overline{\mathtt{u}}\ (\mathtt{reach}(\mathtt{u}, \overline{\mathtt{u}}) \wedge \neg \mathtt{alloc}(\overline{\mathtt{u}}))).$$

Note that the quantification above is fine even when $\mathtt{u}$ is not in the heap domain (take the value for $\mathtt{u}$ to witness the satisfaction of $\exists\, \overline{\mathtt{u}}\ (\mathtt{reach}(\mathtt{u}, \overline{\mathtt{u}}) \wedge \neg \mathtt{alloc}(\overline{\mathtt{u}}))$). In order to express in 1MSOL that there is a binary tree whose number of leaves is a multiple of 3, we first identify the locations of the tree (via the second-order variable $\mathtt{P}$), we label each location of the tree by either $\mathtt{P}_0$, $\mathtt{P}_1$ and $\mathtt{P}_2$ (depending on the number of leaves (modulo 3) below the location) and we state consistency constraints (obviously simulating the behavior of some bottom-up three-state tree automaton) and finally we require that the root of the tree is labelled by $\mathtt{P}_0$. The formula defined below assumes that the heap is already known as a forest of binary trees.

$$\exists\ \mathtt{u}\ (\neg \mathtt{alloc}(\mathtt{u}) \wedge \sharp \mathtt{u} \geq 1) \wedge$$

$$\exists\ \mathtt{P}, \mathtt{P}_0, \mathtt{P}_1, \mathtt{P}_2\ ((\forall\, \overline{\mathtt{u}}\, \mathtt{P}(\overline{\mathtt{u}}) \Leftrightarrow \mathtt{reach}(\overline{\mathtt{u}}, \mathtt{u})) \wedge (\mathtt{P} = \mathtt{P}_0 \uplus \mathtt{P}_1 \uplus \mathtt{P}_2)\ \wedge\ \mathtt{P}_0(\mathtt{u}) \wedge$$

$$(\forall\, \overline{\mathtt{u}}\ (\mathtt{P}(\overline{\mathtt{u}}) \wedge \sharp \overline{\mathtt{u}} = 0) \Rightarrow \mathtt{P}_1(\overline{\mathtt{u}})) \wedge$$

$$(\forall\, \overline{\mathtt{u}}, \overline{\mathtt{u}}'\ ((\overline{\mathtt{u}} \hookrightarrow \overline{\mathtt{u}}') \wedge \mathtt{P}(\overline{\mathtt{u}}) \wedge \sharp \overline{\mathtt{u}}' = 1) \Rightarrow \bigwedge_{i=0}^{2} (\mathtt{P}_i(\overline{\mathtt{u}}) \Leftrightarrow \mathtt{P}_i(\overline{\mathtt{u}}'))) \wedge$$

$$\bigwedge_{i,j,k \in \{0,1,2\}, i \equiv_3 j+k} (\forall\, \overline{\mathtt{u}}, \overline{\mathtt{u}}', \overline{\mathtt{u}}''(\mathtt{P}_j(\overline{\mathtt{u}}') \wedge \mathtt{P}_k(\overline{\mathtt{u}}'') \wedge (\overline{\mathtt{u}}' \neq \overline{\mathtt{u}}'') \wedge (\overline{\mathtt{u}}' \hookrightarrow \overline{\mathtt{u}}) \wedge (\overline{\mathtt{u}}'' \hookrightarrow \overline{\mathtt{u}})) \Rightarrow \mathtt{P}_i(\overline{\mathtt{u}})))$$

Note that we used a shortcut formula $(\mathtt{P} = \mathtt{P}_0 \uplus \mathtt{P}_1 \uplus \mathtt{P}_2)$ to state that the interpretation of $\mathtt{P}$ is the disjoint union of the interpretation of $\mathtt{P}_0$, $\mathtt{P}_1$ and $\mathtt{P}_2$, details are omitted.

4.5 Exercises

Exercise 4.1. Given $\alpha \geq 1$, define a formula $\mathtt{mp}(\mathtt{u})$ that holds true on $(\alpha, 1)$-fishbone heaps whenever $\mathtt{u}$ is interpreted as a location on the main path.

Exercise 4.2. Assuming that $\mathrm{card}(\Sigma) \geq 2$, show that computing $t(\varphi)$ (in Section 4.2.3) may require exponential time in the worst-case.

Exercise 4.3. Show Lemma 4.2.5.

Exercise 4.4. With the abbreviation $[U]$ introduced in Section 4.3, show that $\mathfrak{M}, \mathfrak{l} \models [U]\varphi$ iff for all $\mathfrak{l}' \in \mathbb{N}$, we have $\mathfrak{M}, \mathfrak{l}' \models \varphi$.

Exercise 4.5. Show that the formula below holds true on heaps that are made of a (finite) collection of trees in which each node has branching-degree at most two.

$$\forall\, \mathtt{u}\ (\sharp\mathtt{u} \leq 2 \wedge \exists\, \overline{\mathtt{u}}\ (\mathtt{reach}(\mathtt{u}, \overline{\mathtt{u}}) \wedge \neg \mathtt{alloc}(\overline{\mathtt{u}}))).$$

Exercise 4.6. Design a formula in 1MSOL that characterises the forests of binary trees such that there is one tree whose number of leaves is a multiple of 5.

Exercise 4.7. Let $\mathrm{SL}'(*)$ be the extension of $1\mathrm{SL}(*)$ in which the separating implication can used but only in a very restricted way, typically in formulae of the form $((\mathtt{size} \leq k) \wedge \varphi) \mathrel{-\!\!*} \varphi'$. Show that the satisfiability problem for $\mathrm{SL}'(*)$ is decidable by using the decidability of $1\mathrm{SL}(*)$.

Chapter 5

DECISION PROCEDURES

Contents

Numerous decision procedures have been designed for fragments of separation logics, from analytic methods [GM10, HCGT14] to translation to theories handled by SMT solvers [PWZ13, PR13, BRK+15], passing via graph-based algorithms [HIOP13]. However, the framework of satisfiability modulo theories

(SMT), strongly related to the mechanisation of the problems SAT and QBF, remains probably the most promising one to develop decision procedures dedicated to reasoning tasks for separation logics. This chapter is dedicated to present several distinct ways to decide fragments of separation logics.

Section 5.1 briefly presents the standard dichotomy between the direct approach and the translation-based approach to decide non-classical logics. In Section 5.2 we present a translation from a symbolic heaps fragment of separation logic into the logic GRASS, following developments in [PWZ13]. In Section 5.3, we establish that the satisfiability and model-checking problems for 1SL0 can be solved in polynomial space by using a non-deterministic algorithm. An equivalence relation on memory states with finite index is designed so that infinity involved in the interpretation of the magic wand operator can be tamed finitely. Then, we characterise precisely the expressiveness of 1SL0 and it is the key step to show a small heap property. Section 5.4 presents a translation from 1SL0 into QBF and into a PSPACE fragment of first-order logic by encoding the quantifications in the algorithm from Section 5.3. The chapter concludes by Section 5.5 in which bibliographical references about the design of proof systems are provided.

Highlights of the chapter

1. Presentation of the NP upper bound for the satisfiability problem for SLL$\mathbb{B}$ relying on an SMT-based translation into the logic GRASS defined from the combination of two logical theories [PWZ13] (Corollary 5.2.8).
2. Characterisation of the expressiveness of 1SL0 by using Boolean combinations of test formulae (Theorem 5.3.7). This follows and refines developments from [Yan01, Loz04a, DGLWM14].
3. Presentation of the PSPACE upper bound for the satisfiability and model-checking problems for 1SL0 (Theorem 5.3.11) based on techniques developped in [Yan01, COY01, Loz04a].

5.1 Direct Versus Translation Approach

5.1.1 Direct approach versus translation for deciding modal logics

In order to mechanise modal logics, there exist at least two main approaches with well-identified motivations. The direct approach consists in building specialised proof systems for the logics and requires building new theorem provers but, it has the advantage to design fine-tuned tools and to propose plenty of optimizations. The development of tableaux-based provers for modal logics following the seminal work [Fit83] perfectly illustrates this trend (see also [GHSS14]). By contrast, the translation approach consists in reducing decision problems for the original logics to similar problems for logics that have already well-established theorem provers (see e.g. [dNSH00]). Its main advantage is to use existing tools and therefore to focus only on the translations, that are usually much simpler to implement. For example, translation of modal logics into first-order logic, with the explicit goal to mechanise such logics is an approach that has been introduced in [Mor76] (see also [Fin75, vB76, Moo77]) and it has been intensively developed over the years, see e.g. [ONdRG01] for an overview.

5.1.2 Translation versus specialised algorithms for separation logic

Despite its young age, one can observe that the mechanisation of separation logic follows a similar dichotomy. This is all the more obvious nowadays since there are a lot of activities to develop verification methods with decision procedures for fragments of practical use, see e.g. [CHO+11]. Many decision procedures have been designed for fragments of separation logics or abstract variants, from analytic methods [GM10, HCGT14] to translation to theories handled by SMT solvers [LQ08, PWZ13], passing via graph-based algorithms [HIOP13]. The translation approach has been already advocated in [CGH05] and in [Hag04, Chapter 8] in which propositional kSL0 is translated into a fragment of classical logic that can be decided in polynomial space (see Section 5.4 for an alternative presentation of that result). However, the framework of satisfiability modulo theories (SMT) remains probably the most promising one to develop decision procedures dedicated to reasoning tasks for separation logics, see e.g [BPS09, RBHC07, PR13, PWZ13]. It is worth noting that the verification of programs

that manipulate linked-list structures can be also done with a SAT solver, when the assertions are written in some restricted logical formalism, see a remarkable example in [IBI+13]. In Section 5.4, we illustrate why QBF solvers (see e.g. [LB10, HSB14]) can be useful too.

5.1.3 The SMT framework

Deciding logical formulae within a given logical theory is ubiquitous in computer science and the works around Satisfiability Modulo Theories (SMT) are dedicated to solve this problem by providing methods, proof systems and solvers in order to be able to decide as much theories as possible, as well as their combination (see e.g. [BT14b]). Nowadays, SMT solvers are essential for most tools that formally verify programs, from bounded model-checking to abstraction-based model-checking (actually the number of applications seems unbounded). Roughly speaking, decision problems for many verification problems are reduced to the satisfiability of formulae in specific first-order/quantifier-free theories that can be handled by SMT solvers. Typical theories are quantifier-free Presburger arithmetic (also known as linear integer arithmetic LIA) or the theory of equality over uninterpreted functions (EUF).

A nice feature of such solvers is their ability to combine distinct theories allowing to express richer statements. As advocated in [PR13, PWZ13], being able to integrate decidable fragments of separation logic in some SMT solver not only allows to decide satisfiability or entailment problems by taking advantage of the technology behind SMT solvers but also it provides an efficient way to combine separation logics with other theories, such as linear arithmetic LIA. Actually, the seminal paper [PWZ13] provides a translation of SLL$\mathbb{B}$ (see also Section 5.2) into a decidable fragment of first-order logic and a decision procedure has been implemented in an SMT solver. This provides an important step to integrate reasoning about separation logic into SMT solvers. The paper [BRK+15] goes even beyond since it presents how SMT solvers can be used systematically to obtain decision procedures for a class of theories that are decidable by using a finite instantiation of (quantified) axioms.

Some strongly related work can be also found in [PR13] in which the constraints added to list segments are much more general than pure equalities. Besides, the first competition of solvers for several fragments of separation logic was held recently, see e.g. [SC14], witnessing how promising appears this research direction. In the article [SC14] reporting the competition SL-COMP 2014, more can be found about the list of solvers that competed as well as the fragments of sep-

aration logic that have been considered (roughly, symbolic heaps with recursive definitions, see also Sections 2.2 and 1.3.3). By way of examples, we list below solvers for separation logics involving SMT-based or SAT-based tools [SC14].

1. Asterix [PR13] is a tool for solving entailment problem for symbolic heaps fragment augmented with theories richer than the one with pure equalities only that relies on SMT solving technology (Z3). This provides a generalization to what has been done in [PWZ13], even though results in [PWZ13] are not restricted to the theory of pure equalities.

2. SLSAT [BFGN14, SC14] is a tool for solving the satisfiability problem for the so-called SLRD+ fragment with general defined predicates.

3. SPEN [ESS13] is a solver that deals with satisfiability and entailment problems for SLRD+ for a subclass of recursive definitions. The solver MiniSAT is also used to resolve Boolean abstraction of separation logic formulae.

In order to solve decision problems for separation logics, there is indeed a great diversity of techniques. As we have seen, there are reductions to SAT or SMT problems [PR13, BFGN14, SC14, ESS13] but other approaches exist, for instance by reduction into tree automata membership/inclusion problems [ESS13, IRV14]. In the rest of the chapter, two approaches are presented in details.

5.2 Translation Into a Reachability Logic

In this section, we present a reduction from the satisfiability problem for SLL^{1}B into the logic GRASS, following the results from [PWZ13, PWZ14]. Decidability and even NP-completeness of the satisfiability problem for GRASS are obtained by a reduction to a slight variant of the combination of two theories, following a standard approach in SMT, see e.g. [BT14b]. Interestingly, the decision procedure uses two standard techniques related to SMT: the combination of theories (see e.g. [NO79, TZ04]) and the handling of quantifiers (see e.g. [BRK$^+$15] for local theory extensions). Other examples of translations can be found in [Hag04, CGH05, CHO$^+$11, HIOP13, SC14] but because of lack of space, we focus on the current one that is very promising to use further SMT solvers in order to decide decision problems for fragments of separation logics. Actually, the work [PWZ13] goes much beyond what is presented in this section.

5.2.1 A target logic combining reachability and sets

The logic of graph reachability and stratified sets GRASS [PWZ13] is defined below as the disjoint combination of a theory of reachability in function graphs (see e.g. [LQ08, WMK11]) and a theory of stratified sets (see e.g. [Zar03]). This means that, on the top of a theory of elements, a theory of sets is defined so that the elements satisfied another theory. In that way, the theory of stratified sets is two-level. Presently, the theory of elements is a theory about reachability. Consequently, the logic GRASS contains two sorts: a sort for locations and a sort for sets of locations. GRASS is presented as the quantifier-free kernel of a many-sorted first-order logic, following an approach that has been at heart of modern SMT solvers (see more definitions in [PWZ13, BT14b]). For the ease of presentation, and since we do not need the full generality of many-sorted first-order logic, our presentation of GRASS semantics is quite ad-hoc but the reader should keep in mind that a more orthodox presentation based on standard features of SMT can be found in [PWZ13].

Below, we present a translation from SLL$\mathbb{B}$ into the logic GRASS, following the results from [PWZ13]. The tool GRASShopper [PWZ14] implements a decision procedure for GRASS on top of the SMT solver Z3 [dMB08].

Formulae in GRASS are built from terms T denoting locations and terms S denoting sets of locations. Atomic formulae of type A state properties between locations whereas atomic formulae of type B state properties between locations and sets of locations. Finally, formulae of type R are Boolean combinations of atomic formulae of type A. These syntactic objects are defined following the grammars below:

$$
\begin{array}{lll}
T & ::= & \mathtt{x} \mid \mathtt{f}(T) \\
A & ::= & T = T \mid T \overset{\setminus T}{\rightarrow} T \\
R & ::= & A \mid \neg R \mid R \wedge R \\
S & ::= & \mathtt{X} \mid \emptyset \mid S \setminus S \mid S \cup S \mid S \cap S \mid \{\mathtt{x}.R\} \\
 & & \textit{proviso:}\ \mathtt{x}\ \text{does not occur below}\ \mathtt{f}\ \text{in}\ R \\
B & ::= & S = S \mid T \in S \\
\varphi & ::= & A \mid B \mid \neg\varphi \mid \varphi \wedge \varphi
\end{array}
$$

The proviso in the definition of set comprehensions guarantees decidability of the logic GRASS. The term $\mathtt{x}$ is a variable from a countably infinite set of (program) variables whereas the term $\mathtt{X}$ is a set variable from an unspecified countably infinite set of such variables. By contrast, there is a single function symbol $\mathtt{f}$.

Example 5.2.1. A formula of type R is presented below:

$$(\mathtt{x}_1 = \mathtt{f}(\mathtt{x}_2)) \wedge \mathtt{x}_1 \overset{\setminus \mathtt{x}_2}{\rightarrow} \mathtt{x}_3.$$

An atomic formula of type B is presented below:

$$\mathtt{f}(\mathtt{f}(\mathtt{x}_1)) \in \{\mathtt{x}.\ (\mathtt{x} = \mathtt{x}_1)\} \cap \{\mathtt{x}'.\ (\mathtt{x}' \overset{\setminus \mathtt{x}_2}{\rightarrow} \mathtt{x}_3)\}.$$

Now, let us define the models for the logic GRASS and let us explain how the formulae are interpreted on such semantical structures. A **GRASS-model** is a structure of the form $\mathcal{A} = (\mathfrak{L}, \mathfrak{h})$ such that

1. $\mathfrak{L}$ is a non-empty set; for obvious reasons elements of $\mathfrak{L}$ are called **locations**.
2. $\mathfrak{h}$ is a map $\mathfrak{L} \rightarrow \mathfrak{L}$ satisfying the conditions below.
   1. For all $\mathfrak{l} \in \mathfrak{L}$, the set $\{\mathfrak{l}' \mid (\mathfrak{l}, \mathfrak{l}') \in \mathfrak{h}^*\}$ is finite where $\mathfrak{h}^*$ is the reflexive and transitive closure of the graph relation induced by the map $\mathfrak{h}$.
   2. Similarly, for all $\mathfrak{l} \in \mathfrak{L}$, the set $\{\mathfrak{l}' \mid (\mathfrak{l}', \mathfrak{l}) \in \mathfrak{h}^*\}$ is finite.

   The map $\mathfrak{h}$ corresponds to the interpretation of the function symbol $\mathtt{f}$.

Even though the symbol '$\mathfrak{h}$' is also used to denote heaps, the map $\mathfrak{h}$ in a GRASS-model is not strictly speaking a heap (in the sense provided in Section 1.2.1). For instance, $\mathfrak{L}$ is not necessarily equal to $\mathbb{N}$ and $\mathfrak{h}$ is a total function. Nevertheless, $\mathfrak{h}$ satisfies two finiteness conditions. However, the way the logic GRASS is defined and the semantics is presented allows to use results about the combination of theories to establish the NP upper bound of the satisfiability problem. Herein, we do not present the details of the NP upper bound for deciding GRASS since we view such a decision procedure as a blackbox. Indeed, in the translation-based approach, the essential work is to design a translation into a logical theory for which decidability/complexity is already established. However, it would be definitely interesting to present how works the decidability proof for GRASS satisfiability, but this is currently beyond the scope of this document.

The terms T are interpreted as elements of $\mathfrak{L}$ whereas the terms S are interpreted as subsets of $\mathfrak{L}$. More precisely, for any term T, $[\![T]\!]_{\mathcal{A},\mathfrak{f}}$ is an element of $\mathfrak{L}$ following the clauses below ($\mathfrak{f}$ is a variable assignment):

$$\begin{array}{lcl} [\![\mathtt{x}]\!]_{\mathcal{A},\mathfrak{f}} & \overset{\text{def}}{=} & \mathfrak{f}(\mathtt{x}) \\ [\![\mathtt{f}(T)]\!]_{\mathcal{A},\mathfrak{f}} & \overset{\text{def}}{=} & \mathfrak{h}([\![T]\!]_{\mathcal{A},\mathfrak{f}}). \end{array}$$

Similarly, for any term S, $[\![S]\!]_{\mathcal{A},\mathfrak{f}}$ is a subset of $\mathfrak{L}$ following the clauses below:

$$
\begin{array}{lcl}
[\![\mathtt{X}]\!]_{\mathcal{A},\mathfrak{f}} & \stackrel{\text{def}}{=} & \mathfrak{f}(\mathtt{X}) \\
[\![\emptyset]\!]_{\mathcal{A},\mathfrak{f}} & \stackrel{\text{def}}{=} & \emptyset \text{ (no syntactic difference between the constant '}\emptyset\text{' and the empty set)} \\
[\![S_1 \ \mathtt{O} \ S_2]\!]_{\mathcal{A},\mathfrak{f}} & \stackrel{\text{def}}{=} & [\![S_1]\!]_{\mathcal{A},\mathfrak{f}} \ \mathtt{O} \ [\![S_2]\!]_{\mathcal{A},\mathfrak{f}} \ (\mathtt{O} \in \{\setminus, \cup, \cap\}) \\
[\![\{\mathtt{x}.R\}]\!]_{\mathcal{A},\mathfrak{f}} & \stackrel{\text{def}}{=} & \{\mathrm{l} \mid \mathcal{A} \models_{\mathfrak{f}[\mathtt{x}\mapsto \mathrm{l}]} R\} \\
 & & \text{where the satisfaction relation } \models \text{ is defined below.}
\end{array}
$$

Despite the fact that the definition of $\models$ uses the semantics map $[\![\cdot]\!]_{\mathcal{A},\mathfrak{f}}$ and the other way around for set comprehension, these mutually recursive definitions are well-founded. So, given a variable assignment $\mathfrak{f}$, the satisfaction relation $\models$ for GRASS is defined as follows:

$$
\begin{array}{lll}
\mathcal{A} \models_{\mathfrak{f}} T_1 = T_2 & \text{iff} & [\![T_1]\!]_{\mathcal{A},\mathfrak{f}} = [\![T_2]\!]_{\mathcal{A},\mathfrak{f}} \\
\mathcal{A} \models_{\mathfrak{f}} T_1 \stackrel{\setminus T_2}{\rightarrow} T_3 & \text{iff} & ([\![T_1]\!]_{\mathcal{A},\mathfrak{f}}, [\![T_3]\!]_{\mathcal{A},\mathfrak{f}}) \in \mathfrak{R}^* \text{ where} \\
 & & \mathfrak{R} = \{(\mathrm{l}, \mathfrak{h}(\mathrm{l})) \mid \mathrm{l} \in \mathfrak{L},\ \mathrm{l} \neq [\![T_2]\!]_{\mathcal{A},\mathfrak{f}}\} \\
\mathcal{A} \models_{\mathfrak{f}} S_1 = S_2 & \text{iff} & [\![S_1]\!]_{\mathcal{A}} = [\![S_2]\!]_{\mathcal{A}} \\
\mathcal{A} \models_{\mathfrak{f}} T \in S & \text{iff} & [\![T]\!]_{\mathcal{A},\mathfrak{f}} \in [\![S]\!]_{\mathcal{A},\mathfrak{f}} \\
\mathcal{A} \models_{\mathfrak{f}} \neg\varphi & \text{iff} & \text{not } \mathcal{A} \models_{\mathfrak{f}} \varphi \\
\mathcal{A} \models_{\mathfrak{f}} \varphi_1 \wedge \varphi_2 & \text{iff} & \mathcal{A} \models_{\mathfrak{f}} \varphi_1 \text{ and } \mathcal{A} \models_{\mathfrak{f}} \varphi_2.
\end{array}
$$

Clauses for equalities and for Boolean connectives are completely standard and therefore the only clause that may deserve a bit of explanations if for atomic formulae of the form $T_1 \stackrel{\setminus T_2}{\rightarrow} T_3$. Indeed, it holds true whenever $[\![T_3]\!]_{\mathcal{A},\mathfrak{f}}$ can be reached from $[\![T_1]\!]_{\mathcal{A},\mathfrak{f}}$ by using the graph of $\mathfrak{h}$, with the proviso that the path does not visit an edge starting by the location $[\![T_2]\!]_{\mathcal{A},\mathfrak{f}}$.

The satisfiability problem for GRASS takes as input a formula φ and asks for the existence of a model $\mathcal{A}$ and a variable assignment $\mathfrak{f}$ such that $\mathcal{A} \models_{\mathfrak{f}} \varphi$.

Proposition 5.2.2. [PWZ13] The satisfiability problem for GRASS is NP-complete.

The underlying theory of reachability in function graphs is not first-order definable not only because of the finiteness constraints on $\mathfrak{h}$ but also because transitive closure of relations is not first-order definable. Nevertheless, satisfiability problem for the quantifier-free fragment of that fragment can be shown in NP, see details in [TW13, Section 5]. Using Nelson-Oppen combination of the decision procedures for the two theories [NO79] (see also [TZ04]), the fact that both of

them are stably infinite with respect to the location sort and other properties, one can obtain an NP decision procedure for GRASS; again we omit the details that can be found in [PWZ13, Appendix A]. So, the decision procedure for GRASS is obtained from the combination of the theory of reachability and the theory of stratified sets, but some additional work is required, for instance to get rid of set comprehension, see e.g. [PWZ13, Appendix A] (see also [BRK+15] for a more general approach using a lazy and finite instantiation of quantified axioms with the help of E-matching).

5.2.2 A variant separation logic interpreted on GRASS-models

Before designing the translation into GRASS, we introduce SLLB in which spatial formulae are interpreted precisely and equalities and disequalities are spatial formulae too. Spatial formulae φ_s are defined as follows:

$$\varphi_s ::= (\mathtt{x}_i = \mathtt{x}_j) \mid \neg(\mathtt{x}_i = \mathtt{x}_j) \mid \mathtt{x}_i \mapsto \mathtt{x}_j \mid \mathtt{sreach}(\mathtt{x}_i, \mathtt{x}_j) \mid \varphi_s * \varphi_s$$

Formulae of SLLB are simply Boolean combinations of spatial formulae.

Models of SLLB are GRASS-models and as we have seen earlier, these are not exactly heaps. Probably, the major difference is due to the fact that the map $\mathfrak{h}$ in a GRASS-model is a total function and therefore to regain the semantics from separation logic, the finite domain is encoded by the interpretation of a set variable. Finiteness is guaranteed because the satisfaction relation $\models^{\mathtt{X}}_{\mathfrak{f}}$ (defined below) encodes a precise semantics parameterised by the set variable $\mathtt{X}$. Below, we provide the definition of $\models^{\mathtt{X}}_{\mathfrak{f}}$; we omit the obvious clauses for the Boolean combinations of spatial formulae.

$\mathcal{A} \models^{\mathtt{X}}_{\mathfrak{f}} \mathtt{x}_i = \mathtt{x}_j$	iff	$\mathfrak{f}(\mathtt{x}_i) = \mathfrak{f}(\mathtt{x}_j)$ and $\mathfrak{f}(\mathtt{X}) = \emptyset$
$\mathcal{A} \models^{\mathtt{X}}_{\mathfrak{f}} \neg(\mathtt{x}_i = \mathtt{x}_j)$	iff	$\mathfrak{f}(\mathtt{x}_i) \neq \mathfrak{f}(\mathtt{x}_j)$ and $\mathfrak{f}(\mathtt{X}) = \emptyset$
$\mathcal{A} \models^{\mathtt{X}}_{\mathfrak{f}} \mathtt{x}_i \mapsto \mathtt{x}_j$	iff	$\mathfrak{h}(\mathfrak{f}(\mathtt{x}_i)) = \mathfrak{f}(\mathtt{x}_j)$ and $\mathfrak{f}(\mathtt{X}) = \{\mathfrak{f}(\mathtt{x}_i)\}$
$\mathcal{A} \models^{\mathtt{X}}_{\mathfrak{f}} \varphi^1_s * \varphi^2_s$	iff	there are X_1, X_2 such that $\mathfrak{f}(\mathtt{X}) = X_1 \uplus X_2$, $\mathcal{A} \models^{\mathtt{X}}_{\mathfrak{f}[\mathtt{X} \mapsto X_1]} \varphi^1_s$ and $\mathcal{A} \models^{\mathtt{X}}_{\mathfrak{f}[\mathtt{X} \mapsto X_2]} \varphi^2_s$
$\mathcal{A} \models^{\mathtt{X}}_{\mathfrak{f}} \mathtt{sreach}(\mathtt{x}_i, \mathtt{x}_j)$	iff	either $\mathfrak{f}(\mathtt{x}_i) = \mathfrak{f}(\mathtt{x}_j)$ and $\mathfrak{f}(\mathtt{X}) = \emptyset$, or there is $n \geq 1$ such that $\mathfrak{h}^n(\mathfrak{f}(\mathtt{x}_i)) = \mathfrak{f}(\mathtt{x}_j)$ and $\mathfrak{f}(\mathtt{X}) = \{\mathfrak{f}(\mathtt{x}_i), \mathfrak{h}(\mathfrak{f}(\mathtt{x}_i)), \ldots, \mathfrak{h}^{n-1}(\mathfrak{f}(\mathtt{x}_i))\}$.

The (possibly empty) set of locations $\mathfrak{f}(\mathtt{X})$ is called the **footprint** of φ when $\mathcal{A} \models^{\mathtt{X}}_{\mathfrak{f}} \varphi$ and it corresponds to the (unique) domain of the underlying heap. For instance, the formula $(\mathtt{x}_1 = \mathtt{x}_3) * (\mathtt{x}_2 \mapsto \mathtt{x}_2)$ is satisfiable unlike $(\mathtt{x}_1 = \mathtt{x}_3) \wedge (\mathtt{x}_2 \mapsto \mathtt{x}_2)$. Lemma 5.2.3 below guarantees finiteness and unicity of the footprint.

Lemma 5.2.3. Let φ_s be a spatial formula, $\mathcal{A}$ be a GRASS-model and $\mathfrak{f}$,$\mathfrak{f}'$ be variable assignments that differ at most for the set variable $\mathtt{X}$. If $\mathcal{A} \models^{\mathtt{X}}_{\mathfrak{f}} \varphi_s$ and $\mathcal{A} \models^{\mathtt{X}}_{\mathfrak{f}'} \varphi_s$ then $\mathfrak{f} = \mathfrak{f}'$ and $\mathfrak{f}(\mathtt{X})$ is finite.

The proof is left as Exercise 5.3. So, in this section, we have followed a precise interpretation of the spatial formulae and a memory state $(\mathfrak{s}, \mathfrak{h})$ in the usual sense in separation logic is encoded by a structure $(\mathbb{N}, \mathfrak{h}', \mathfrak{f}, \mathtt{X})$ where

1. $(\mathbb{N}, \mathfrak{h}')$ is a GRASS-model,
2. $\mathfrak{s}$ and $\mathfrak{f}$ agree on the program variables and $\mathfrak{f}(\mathtt{X}) = \mathrm{dom}(\mathfrak{h})$,
3. $\mathfrak{h}'$ restricted to $\mathfrak{f}(\mathtt{X})$ is equal to $\mathfrak{h}$.

We write GRASS-FO to denote the first-order extension of quantifier-free GRASS by adding existential quantification over set variables: $\mathcal{A} \models_{\mathfrak{f}} \exists\, \mathtt{Y}\ \varphi$ iff there is $X \subseteq \mathfrak{L}$ such that $\mathcal{A} \models_{\mathfrak{f}[\mathtt{Y} \mapsto X]} \varphi$. Actually, in the translation described below, existential quantifications shall occur only in front of the formulae obtained by translation and therefore its satisfiability status is identical to the satisfiability status of the formulae obtained by removing the existential quantifications. So, the formulae in GRASS-FO are purely instrumental in this presentation and the logic GRASS-FO is not studied for itself.

In Lemma 5.2.4 below, we state a few properties that are useful to show the correctness of the forthcoming translation but these are quite standard in first-order logic and therefore the proofs are left as Exercise 5.4.

Lemma 5.2.4.

(I) Let φ_1 and φ_2 be GRASS-FO formulae so that $\mathtt{Y}_1$ is not free in φ_2 and $\mathtt{Y}_2$ is not free in φ_1. Then, $(\exists\, \mathtt{Y}_1 \varphi_1)\ \mathtt{O}\ (\exists\, \mathtt{Y}_2\ \varphi_2)$ is logically equivalent to $\exists\, \mathtt{Y}_1, \mathtt{Y}_2\ (\varphi_1\ \mathtt{O}\ \varphi_2)$ with $\mathtt{O} \in \{\wedge, \vee\}$.

(II) Given a GRASS-FO formula of the form $\exists\, \mathtt{Y}\ \varphi$, we have φ is satisfiable iff $\exists\, \mathtt{Y}\ \varphi$ is satisfiable.

5.2.3 A logarithmic-space translation

Given an atomic formula φ and a (fresh) set variable $\mathtt{Y}$, we introduce in the table below the formulae ψ_1 and $\psi_2(\mathtt{Y})$ that are used in the translation from SLLB into GRASS (implicitly these formulae are parameterised by φ). The formula $\psi_1(\mathtt{Y})$ encodes the reachability constraints expressed by φ whereas $\psi_2(\mathtt{Y})$ takes care of the footprint.

Atomic formula φ	ψ_1	$\psi_2(\mathtt{Y})$
$\mathtt{x}_i = \mathtt{x}_j$	$\mathtt{x}_i = \mathtt{x}_j$	$\mathtt{Y} = \emptyset$
$\mathtt{x}_i \neq \mathtt{x}_j$	$\mathtt{x}_i \neq \mathtt{x}_j$	$\mathtt{Y} = \emptyset$
$\mathtt{x}_i \mapsto \mathtt{x}_j$	$\mathtt{f}(\mathtt{x}_i) = \mathtt{x}_j$	$\mathtt{Y} = \{\mathtt{y}.\ (\mathtt{x}_i = \mathtt{y})\}$
$\mathtt{sreach}(\mathtt{x}_i, \mathtt{x}_j)$	$\mathtt{x}_i \overset{\setminus \mathtt{x}_j}{\rightarrow} \mathtt{x}_j$	$\mathtt{Y} = \{\mathtt{x}.\ (\mathtt{x}_i \overset{\setminus \mathtt{x}_j}{\rightarrow} \mathtt{x}) \ \wedge\ \mathtt{x} \neq \mathtt{x}_j\}$

Let $\mathtt{X}$ be a distinguished set variable that serves for the footprint. The translation tr defined below from SLL$\mathbb{B}$ formulae into GRASS-FO formulae is implicitly parameterised by $\mathtt{X}$. Without any loss of generality, we can assume that the SLL$\mathbb{B}$ formulae are in negation normal form (binary connectives are $\wedge$ and $\vee$, and negation occurs only in front of spatial formulae). The map tr is homomorphic for $\vee$ and $\wedge$ and its definition is completed by the clauses below:

$$tr(\varphi_s^1 * \cdots * \varphi_s^n) \overset{\text{def}}{=} \exists\, \mathtt{Y}_1\ \cdots\ \mathtt{Y}_n\ ((\psi_1^1 \wedge \cdots \wedge \psi_1^n \wedge \bigwedge_{i \neq i'} \mathtt{Y}_i \cap \mathtt{Y}_{i'} = \emptyset) \wedge$$

$$((\psi_2^1(\mathtt{Y}_1) \wedge \cdots \wedge \psi_2^n(\mathtt{Y}_n) \wedge \mathtt{X} = \mathtt{Y}_1 \cup \cdots \cup \mathtt{Y}_n),$$

where the formulae $\psi_1^i(\mathtt{Y}_i)$ and $\psi_2^i(\mathtt{Y}_i)$ are defined from the previous table with the atomic formula φ_s^i, and $\mathtt{Y}_1, \ldots, \mathtt{Y}_n$ are fresh set variables. Similarly,

$$tr(\neg(\varphi_s^1 * \cdots * \varphi_s^n)) \overset{\text{def}}{=} \exists\, \mathtt{Y}_1\ \cdots\ \mathtt{Y}_n\ \neg((\psi_1^1 \wedge \cdots \wedge \psi_1^n \wedge \bigwedge_{i \neq i'} \mathtt{Y}_i \cap \mathtt{Y}_{i'} = \emptyset) \wedge$$

$$((\psi_2^1(\mathtt{Y}_1) \wedge \cdots \wedge \psi_2^n(\mathtt{Y}_n) \wedge \mathtt{X} = \mathtt{Y}_1 \cup \cdots \cup \mathtt{Y}_n).$$

So both translations quantify existentially over set variables and the only difference is the presence of the negation in front of one of the main conjuncts. This requires some explanation, which is provided below.

It is worth observing that for all GRASS-models $\mathcal{A}$ and for all variable assignments $\mathfrak{f}$, there is exactly one tuple $(X_1, \ldots, X_n)$ in $\mathfrak{L}^n$ such that $\mathcal{A} \models_{\mathfrak{g}} (\psi_2^1(\mathtt{Y}_1) \wedge \cdots \wedge \psi_2^n(\mathtt{Y}_n) \wedge \mathtt{X} = \mathtt{Y}_1 \cup \cdots \cup \mathtt{Y}_n$ where $\mathfrak{g} = \mathfrak{f}[\mathtt{Y}_1 \mapsto X_1, \ldots, \mathtt{Y}_n \mapsto X_n]$. Thanks to that property that is closely related to preciseness, we can get advantage of the property below.

Lemma 5.2.5. Let χ_1 and $\chi_2(\mathtt{X}, \mathtt{Y}_1, \ldots, \mathtt{Y}_n)$ be GRASS formulae such that for all GRASS-models $\mathcal{A}$ and for all variable assignments $\mathfrak{f}$, there is exactly one tuple $(X_1, \ldots, X_n)$ in $\mathfrak{L}^n$ such that $\mathcal{A} \models_{\mathfrak{g}} \chi_2(\mathtt{X}, \mathtt{Y}_1, \ldots, \mathtt{Y}_n)$ where $\mathfrak{g} = \mathfrak{f}[\mathtt{Y}_1 \mapsto X_1, \ldots, \mathtt{Y}_n \mapsto X_n]$. Then, for all GRASS-models $\mathcal{A}$ and all variable assignments $\mathfrak{f}$, the statements below are equivalent:

(I) $\mathcal{A} \models_{\dagger} \neg(\exists\, \mathtt{Y}_1 \cdots \mathtt{Y}_n\, \chi_1 \wedge \chi_2(\mathtt{X}, \mathtt{Y}_1, \ldots, \mathtt{Y}_n))$,

(II) $\mathcal{A} \models_{\dagger} (\exists\, \mathtt{Y}_1 \cdots \mathtt{Y}_n\, \neg\chi_1 \wedge \chi_2(\mathtt{X}, \mathtt{Y}_1, \ldots, \mathtt{Y}_n))$.

The proof of Lemma 5.2.5 is left as Exercise 5.5 and requires standard arguments from first-order logic. Consequently, $tr(\neg(\varphi_s^1 * \cdots * \varphi_s^n))$ is logically equivalent to the negation of $tr(\varphi_s^1 * \cdots * \varphi_s^n)$ even though both translations involve existential quantifications. This is a key property established in [PWZ13]. Let $T(\varphi)$ be the GRASS formula obtained from $tr(\varphi)$ by removing the sequences of existential quantifications of the form $\exists\, \mathtt{Y}_1 \cdots \mathtt{Y}_n$.

Example 5.2.6. The formula $\varphi = (\mathtt{x}_1 \mapsto \mathtt{x}_2) \wedge \mathtt{x}_1 = \mathtt{x}_3$ is not satisfiable and its translation $T(\varphi)$ in GRASS is given below:

$$(\mathtt{f}(\mathtt{x}_1) = \mathtt{x}_2 \wedge \mathtt{Y}_1 = \{\mathtt{y}.(\mathtt{x}_1 = \mathtt{y})\} \wedge \mathtt{X} = \mathtt{Y}_1) \wedge (\mathtt{x}_1 = \mathtt{x}_3 \wedge \mathtt{Y}_2 = \emptyset \wedge \mathtt{X} = \mathtt{Y}_2).$$

By contrast, the formula $\varphi' = (\mathtt{x}_1 \neq \mathtt{x}_3) * (\mathtt{x}_1 \mapsto \mathtt{x}_3) * (\mathtt{x}_3 \mapsto \mathtt{x}_3)$ is satisfiable and its translation $T(\varphi')$ is the following:

$$(\mathtt{x}_1 \neq \mathtt{x}_3 \wedge \mathtt{f}(\mathtt{x}_1) = \mathtt{x}_3 \wedge \mathtt{f}(\mathtt{x}_3) = \mathtt{x}_3 \wedge (\mathtt{Y}_1 \cap \mathtt{Y}_2) = \emptyset \wedge (\mathtt{Y}_1 \cap \mathtt{Y}_3) = \emptyset \wedge (\mathtt{Y}_2 \cap \mathtt{Y}_3) = \emptyset) \wedge$$

$$\mathtt{Y}_1 = \emptyset \wedge \mathtt{Y}_2 = \{\mathtt{y}.(\mathtt{x}_1 = \mathtt{y})\} \wedge \mathtt{Y}_3 = \{\mathtt{y}.(\mathtt{x}_3 = \mathtt{y})\} \wedge (\mathtt{X} = \mathtt{Y}_1 \cup \mathtt{Y}_2 \cup \mathtt{Y}_3).$$

By Lemma 5.2.4, $T(\varphi)$ is satisfiable iff $tr(\varphi)$ is satisfiable. It remains to show that the satisfiability status of $tr(\varphi)$ is equivalent to the satisfiability status of φ.

Lemma 5.2.7. Let φ be an SLL$\mathbb{B}$ formula in negation normal form, $\mathcal{A}$ be a GRASS-model, $\dagger$ be a variable assignment and $\mathtt{X}$ be a distinguished set variable. Then $\mathcal{A} \models_{\dagger}^{\mathtt{X}} \varphi$ iff $\mathcal{A} \models_{\dagger} tr(\varphi)$.

Consequently, there is a logarithmic-space reduction from the satisfiability problem for SLL$\mathbb{B}$ into the satisfiability problem for GRASS. Given an SLL$\mathbb{B}$ formula φ, we perform the following operations:

1. Compute φ' obtained from φ by pushing the negations inwards as much as possible so that φ' is an equivalent formula in negation normal form.

2. Compute $T(\varphi')$ by using the recursive definition for $tr(\varphi')$ but we omit the prefixes of the form $\exists\, \mathtt{Y}_1 \cdots \mathtt{Y}_n$ on-the-fly.

The proof of Lemma 5.2.7 is left as Exercise 5.7 and it can be done by a standard structural induction. In the induction step, the cases for Boolean connectives $\wedge$ and $\vee$ are by an easy verification whereas the base case for atomic formulae of the form $\neg(\varphi_s^1 * \cdots * \varphi_s^n)$ follows immediately from the base case for atomic formulae of the form $\varphi_s^1 * \cdots * \varphi_s^n$ thanks to Lemma 5.2.5. So, the main difficulty in the proof is to consider the translation for atomic formulae of the form $(\varphi_s^1 * \cdots * \varphi_s^n)$ but this is not very difficult since then the translation simply internalises the SLL$\mathbb{B}$ precise semantics into GRASS-FO.

Corollary 5.2.8. The satisfiability, validity and entailment problems for SLL$\mathbb{B}$ can be solved in NP.

NP-hardness is an obvious consequence that SLL$\mathbb{B}$ contains equality constraints and it is closed under Boolean connectives.

We have seen that the logics SLL$\mathbb{B}$ and GRASS have quite a lot of similarities and it is legitimate to wonder why to bother to introduce the logic GRASS as it has been done in [PWZ13]. Indeed, the models are quite similar, the encoding of the reachability constraints and footprint is quite clear. However, the decidability proof for GRASS (omitted in this document) invokes results about the combination of theories along the lines of the seminal work of Nelson-Oppen (stably infinity of the theories) but also about elimination of implicit quantifiers or set comprehensions. That is why, the way GRASS has been designed allows to use material about the decidability of combined theories (we recall that the details can be found in [PWZ13]) and to take advantage of the SMT framework for combinations with other theories. Alternatively, the developments in this section can be viewed as a piece of evidence that SLL$\mathbb{B}$ can be rephrased to fit the SMT framework, as far as several decision problems are concerned (satisfiability, abduction, etc).

5.3 Direct Approach: An Example

In this section, we show that the satisfiability and model-checking problems for 1SL0 (the propositional version of 1SL without quantifiers and quantified variables) can be solved in polynomial space. An equivalence relation on memory states with finite index is designed so that infinity involved in the interpretation of the magic wand operator can be tamed finitely [Yan01, Loz04b]. This allows to characterise precisely the expressiveness of 1SL0 and it is the key step to show a small heap property. The developments made in this section are mainly inspired

from material and results in [Yan01, COY01, Loz04a] (see also some subsequent developments in [DGLWM14]).

5.3.1 Expressiveness

Given $q \geq 1$ and $\alpha \in \mathbb{N}$, we write $\mathtt{Test}(q, \alpha)$ to denote the following set of atomic formulae:

1. $\mathtt{x}_i = \mathtt{x}_j$, $\mathtt{x}_i \hookrightarrow \mathtt{x}_j$ with $i, j \in [1, q]$.
2. $\mathtt{alloc}(\mathtt{x}_i)$ with $i \in [1, q]$.
3. $\mathtt{size} \geq \beta$ with $\beta \in [0, \alpha]$.

Here, $\mathtt{alloc}(\mathtt{x}_i)$ and $\mathtt{size} \geq \beta$ are not anymore shortcuts as defined in Section 1.2.2 but these expressions should be understood as primitive formulae (at least to build Boolean combinations from it). We recall that $\mathtt{size} \geq \beta$ holds true when the heap domain has cardinal at least β. Hence, formulae from $\mathtt{Test}(q, \alpha)$ state very basic properties on memory states.

We can also use $\mathtt{size}_{\overline{q}} \geq \beta$ as an abbreviation for the Boolean combination of test formulae defined below that characterises the memory states such that the cardinal of the heap domain is at least β even if we remove from it the locations that are interpreted by a program variable among $\mathtt{x}_1, \ldots, \mathtt{x}_q$:

$$\bigvee_{X \subseteq \{\mathtt{x}_1, \ldots, \mathtt{x}_q\}} \Big(\bigwedge_{\mathtt{x} \in X} \mathtt{alloc}(\mathtt{x}) \Big) \wedge \Big(\bigwedge_{\mathtt{x} \in \{\mathtt{x}_1, \ldots, \mathtt{x}_q\} \setminus X} \neg \mathtt{alloc}(\mathtt{x}) \Big) \wedge \mathtt{size} \geq (\mathrm{card}(X) + \beta).$$

Alternatively, we write $\mathtt{Test}'(q, \alpha)$ to denote the variant set of $\mathtt{Test}(q, \alpha)$ in which test formulae of the form $\mathtt{size} \geq \beta$ are replaced by $\mathtt{size}_{\overline{q}} \geq \beta$ (understood as primitive formulae this time).

Lemma 5.3.1. Let $(\mathfrak{s}, \mathfrak{h})$ be a memory state and $\mathfrak{h} \sqsubseteq \mathfrak{h}'$ (i.e. $\mathfrak{h}'$ is a conservative extension of $\mathfrak{h}$). For all $\psi \in \mathtt{Test}'(q, \alpha)$, $(\mathfrak{s}, \mathfrak{h}) \models \psi$ implies $(\mathfrak{s}, \mathfrak{h}') \models \psi$.

The proof is left as Exercise 5.8. We write $(\mathfrak{s}, \mathfrak{h}) \approx^q_\alpha (\mathfrak{s}', \mathfrak{h}')$ ($q \geq 1$, $\alpha \in \mathbb{N}$) whenever the memory states $(\mathfrak{s}, \mathfrak{h})$ and $(\mathfrak{s}', \mathfrak{h}')$ agree on the satisfaction of the test formulae in $\mathtt{Test}'(q, \alpha)$. The equivalence relation is actually an indistinguishability relation with respect to a finite set of formulae parameterised by syntactic resources based on q and α. Note that, above $\mathtt{size}_{\overline{q}} \geq \beta$ is used instead of $\mathtt{size} \geq \beta$, which shall simplify a few forthcoming technical developments (this is therefore a bit different from what has been done in [Yan01, Loz04a, BDL09] for instance). However, this is irrelevant for expressiveness, see also Exercise 5.13.

Lemma 5.3.2. Let $q \geq 1$ and $\alpha \in \mathbb{N}$. Then $\approx^q_{\alpha+1} \subseteq \approx^q_\alpha$ and $\approx^{q+1}_\alpha \subseteq \approx^q_\alpha$.

The proof is left as Exercise 5.9. Below, we establish two technical lemmas that garantee that $\approx^q_\alpha$ behave properly. These are essential properties to establish forthcoming Theorem 5.3.6.

Lemma 5.3.3. Let $\alpha, \alpha_1, \alpha_2 \in \mathbb{N}$ with $\alpha = \alpha_1 + \alpha_2$ and $(\mathfrak{s}, \mathfrak{h})$, $(\mathfrak{s}', \mathfrak{h}')$ be memory states such that $(\mathfrak{s}, \mathfrak{h}) \approx^q_\alpha (\mathfrak{s}', \mathfrak{h}')$. For all heaps $\mathfrak{h}_1$, $\mathfrak{h}_2$ such that $\mathfrak{h} = \mathfrak{h}_1 \uplus \mathfrak{h}_2$, there are heaps $\mathfrak{h}'_1$, $\mathfrak{h}'_2$ such that $\mathfrak{h}' = \mathfrak{h}'_1 \uplus \mathfrak{h}'_2$, $(\mathfrak{s}, \mathfrak{h}_1) \approx^q_{\alpha_1} (\mathfrak{s}', \mathfrak{h}'_1)$ and $(\mathfrak{s}, \mathfrak{h}_2) \approx^q_{\alpha_2} (\mathfrak{s}', \mathfrak{h}'_2)$.

Proof. Let $\alpha, \alpha_1, \alpha_2$, $(\mathfrak{s}, \mathfrak{h})$, $(\mathfrak{s}', \mathfrak{h}')$ and $\mathfrak{h}_1$, $\mathfrak{h}_2$ be defined as in the statement. In order to build $\mathfrak{h}'_1$ and $\mathfrak{h}'_2$, let us make a few basic observations.

1. Since $(\mathfrak{s}, \mathfrak{h}) \approx^q_\alpha (\mathfrak{s}', \mathfrak{h}')$, for all $i \in [1, q]$, $\mathfrak{s}(\mathbf{x}_i) \in \mathrm{dom}(\mathfrak{h})$ iff $\mathfrak{s}'(\mathbf{x}_i) \in \mathrm{dom}(\mathfrak{h}')$.
2. Let $\mathsf{C}_{\overline{q}}(\mathfrak{s}, \mathfrak{h})$ be the cardinal of the set $\mathrm{dom}(\mathfrak{h}) \setminus \{\mathfrak{s}(\mathbf{x}_1), \ldots, \mathfrak{s}(\mathbf{x}_q)\}$. We have $\mathsf{C}_{\overline{q}}(\mathfrak{s}, \mathfrak{h}) = \mathsf{C}_{\overline{q}}(\mathfrak{s}, \mathfrak{h}_1) + \mathsf{C}_{\overline{q}}(\mathfrak{s}, \mathfrak{h}_2)$.

Let us define explicitly the heap $\mathfrak{h}'_1$ and therefore $\mathfrak{h}'_2$ is defined as the complement heap with respect to the heap $\mathfrak{h}'$. Moreover, we only need to specify explicitly the domain of $\mathfrak{h}'_1$ since the images are those from $\mathfrak{h}'$. There is only one exception in the Case 4 below in which $\mathfrak{h}'_2$ is defined explicitly instead.

- For every $i \in [1, q]$, if $\mathfrak{s}(\mathbf{x}_i) \in \mathrm{dom}(\mathfrak{h}_1)$, then $\mathfrak{s}'(\mathbf{x}'_i) \in \mathrm{dom}(\mathfrak{h}'_1)$ by definition.
- So, for every $i \in [1, q]$, $\mathfrak{s}'(\mathbf{x}'_i) \in \mathrm{dom}(\mathfrak{h}'_1)$ iff $\mathfrak{s}(\mathbf{x}_i) \in \mathrm{dom}(\mathfrak{h}_1)$ and therefore $\mathfrak{s}'(\mathbf{x}'_i) \in \mathrm{dom}(\mathfrak{h}'_2)$ iff $\mathfrak{s}(\mathbf{x}_i) \in \mathrm{dom}(\mathfrak{h}_2)$.

The heap $\mathfrak{h}'_1$ is further populated depending on cardinality constraints.

Case 1: $\mathsf{C}_{\overline{q}}(\mathfrak{s}, \mathfrak{h}) \leq \alpha$.
Since $(\mathfrak{s}, \mathfrak{h}) \approx^q_\alpha (\mathfrak{s}', \mathfrak{h}')$, we have $\mathsf{C}_{\overline{q}}(\mathfrak{s}', \mathfrak{h}') = \mathsf{C}_{\overline{q}}(\mathfrak{s}, \mathfrak{h}) \leq \alpha$ too. Let $\beta_1 = \mathsf{C}_{\overline{q}}(\mathfrak{s}, \mathfrak{h}_1)$. Since $\mathsf{C}_{\overline{q}}(\mathfrak{s}', \mathfrak{h}') = \mathsf{C}_{\overline{q}}(\mathfrak{s}, \mathfrak{h})$, and $\beta_1 \leq \mathsf{C}_{\overline{q}}(\mathfrak{s}', \mathfrak{h}')$, there are β_1 locations $\mathfrak{l}_1, \cdots, \mathfrak{l}_{\beta_1}$ in $\mathrm{dom}(\mathfrak{h}') \setminus \{\mathfrak{s}'(\mathbf{x}_1), \ldots, \mathfrak{s}'(\mathbf{x}_q)\}$. Then $\{\mathfrak{l}_1, \ldots, \mathfrak{l}_{\beta_1}\} \subseteq \mathrm{dom}(\mathfrak{h}'_1)$ by definition. This concludes the construction of $\mathfrak{h}'_1$.
Case 2: $\mathsf{C}_{\overline{q}}(\mathfrak{s}, \mathfrak{h}) > \alpha$, $\mathsf{C}_{\overline{q}}(\mathfrak{s}, \mathfrak{h}_1) \geq \alpha_1$ *and* $\mathsf{C}_{\overline{q}}(\mathfrak{s}, \mathfrak{h}_2) \geq \alpha_2$.
Let $\beta_1 = min(\alpha_1, \mathsf{C}_{\overline{q}}(\mathfrak{s}, \mathfrak{h}_1)) = \alpha_1$. Since $min(\alpha, \mathsf{C}_{\overline{q}}(\mathfrak{s}, \mathfrak{h})) = min(\alpha, \mathsf{C}_{\overline{q}}(\mathfrak{s}', \mathfrak{h}')) = \alpha$ and $\beta_1 \leq min(\alpha, \mathsf{C}_{\overline{q}}(\mathfrak{s}, \mathfrak{h}))$, then $\beta_1 \leq min(\alpha, \mathsf{C}_{\overline{q}}(\mathfrak{s}', \mathfrak{h}'))$. Let $\mathfrak{l}_1 < \cdots < \mathfrak{l}_{\beta_1}$ be the β_1 smallest locations in $\mathrm{dom}(\mathfrak{h}') \setminus \{\mathfrak{s}'(\mathbf{x}_1), \ldots, \mathfrak{s}'(\mathbf{x}_q)\}$ (for the usual linear ordering on $\mathbb{N}$). Then $\{\mathfrak{l}_1, \ldots, \mathfrak{l}_{\beta_1}\} \subseteq \mathrm{dom}(\mathfrak{h}'_1)$ by definition. This concludes the construction of $\mathfrak{h}'_1$.

Case 3: $\mathsf{C}_{\overline{q}}(\mathfrak{s},\mathfrak{h}) > \alpha$, $\mathsf{C}_{\overline{q}}(\mathfrak{s},\mathfrak{h}_1) < \alpha_1$ *and* $\mathsf{C}_{\overline{q}}(\mathfrak{s},\mathfrak{h}_2) > \alpha_2$.
Let $\beta_1 = \mathsf{C}_{\overline{q}}(\mathfrak{s},\mathfrak{h}_1)$. Since $min(\alpha, \mathsf{C}_{\overline{q}}(\mathfrak{s},\mathfrak{h})) = min(\alpha, \mathsf{C}_{\overline{q}}(\mathfrak{s}',\mathfrak{h}')) = \alpha$, $\beta_1 \leq \alpha$ and $\beta_1 \leq min(\alpha, \mathsf{C}_{\overline{q}}(\mathfrak{s}',\mathfrak{h}'))$, let $\mathrm{l}_1 < \cdots < \mathrm{l}_{\beta_1}$ be the β_1 smallest locations in $\mathrm{dom}(\mathfrak{h}') \setminus \{\mathfrak{s}'(\mathtt{x}_1), \ldots, \mathfrak{s}'(\mathtt{x}_q)\}$. Then $\{\mathrm{l}_1, \ldots, \mathrm{l}_{\beta_1}\} \subseteq \mathrm{dom}(\mathfrak{h}'_1)$ by definition. This concludes the construction of $\mathfrak{h}'_1$.
Case 4: $\mathsf{C}_{\overline{q}}(\mathfrak{s},\mathfrak{h}) > \alpha$, $\mathsf{C}_{\overline{q}}(\mathfrak{s},\mathfrak{h}_1) \geq \alpha_1$ *and* $\mathsf{C}_{\overline{q}}(\mathfrak{s},\mathfrak{h}_2) < \alpha_2$.
This is symmetrical to Case 3 and the only case for which we define explicitly the heap $\mathfrak{h}'_2$ (and therefore $\mathfrak{h}'_1$ by complementation). Let $\beta_2 = \mathsf{C}_{\overline{q}}(\mathfrak{s},\mathfrak{h}_2)$. Since $min(\alpha, \mathsf{C}_{\overline{q}}(\mathfrak{s},\mathfrak{h})) = min(\alpha, \mathsf{C}_{\overline{q}}(\mathfrak{s}',\mathfrak{h}')) = \alpha$, $\beta_2 \leq \alpha$ and $\beta_2 \leq min(\alpha, \mathsf{C}_{\overline{q}}(\mathfrak{s}',\mathfrak{h}'))$, let $\mathrm{l}_1 < \cdots < \mathrm{l}_{\beta_2}$ be the β_2 smallest locations in $\mathrm{dom}(\mathfrak{h}') \setminus \{\mathfrak{s}'(\mathtt{x}_1), \ldots, \mathfrak{s}'(\mathtt{x}_q)\}$. Then $\{\mathrm{l}_1, \ldots, \mathrm{l}_{\beta_2}\} \subseteq \mathrm{dom}(\mathfrak{h}'_2)$ by definition. This concludes the construction of $\mathfrak{h}'_2$.

Note that after the last three cases, the case $\mathsf{C}_{\overline{q}}(\mathfrak{s},\mathfrak{h}_1) < \alpha_1$, $\mathsf{C}_{\overline{q}}(\mathfrak{s},\mathfrak{h}_2) < \alpha_2$ is excluded because $\alpha = \alpha_1 + \alpha_2$ and $\mathsf{C}_{\overline{q}}(\mathfrak{s},\mathfrak{h}) > \alpha$.

It remains to show that $(\mathfrak{s},\mathfrak{h}_1) \approx^q_{\alpha_1} (\mathfrak{s}',\mathfrak{h}'_1)$ and $(\mathfrak{s},\mathfrak{h}_2) \approx^q_{\alpha_2} (\mathfrak{s}',\mathfrak{h}'_2)$. For the satisfaction of the test formulae of the form $\mathtt{x}_i = \mathtt{x}_j$, $\mathtt{alloc}(\mathtt{x}_i)$ and $\mathtt{x}_i \hookrightarrow \mathtt{x}_j$, the proof is by an easy verification. It remains to check the satisfiability status of the formulae $\mathtt{size}_{\overline{q}} \geq \beta$ with $\beta \in [0, \alpha]$. This amounts to check that $min(\alpha_1, \mathsf{C}_{\overline{q}}(\mathfrak{s},\mathfrak{h}_1)) = min(\alpha_1, \mathsf{C}_{\overline{q}}(\mathfrak{s},\mathfrak{h}'_1))$ and $min(\alpha_2, \mathsf{C}_{\overline{q}}(\mathfrak{s},\mathfrak{h}_2)) = min(\alpha_2, \mathsf{C}_{\overline{q}}(\mathfrak{s},\mathfrak{h}'_2))$. The Case 1 is by an easy verification since $\mathsf{C}_{\overline{q}}(\mathfrak{s},\mathfrak{h}_1) = \mathsf{C}_{\overline{q}}(\mathfrak{s}',\mathfrak{h}'_1)$ and $\mathsf{C}_{\overline{q}}(\mathfrak{s},\mathfrak{h}_2) = \mathsf{C}_{\overline{q}}(\mathfrak{s}',\mathfrak{h}'_2)$ whereas the Case 2 is quite immediate because $min(\alpha_1, \mathsf{C}_{\overline{q}}(\mathfrak{s},\mathfrak{h}_1)) = min(\alpha_1, \mathsf{C}_{\overline{q}}(\mathfrak{s}',\mathfrak{h}'_1)) = \alpha_1$ by construction, and $\mathsf{C}_{\overline{q}}(\mathfrak{s}',\mathfrak{h}') - \alpha_1 > \alpha_2$ and therefore $min(\alpha_2, \mathsf{C}_{\overline{q}}(\mathfrak{s}',\mathfrak{h}'_2)) = \alpha_2$, i.e. $min(\alpha_2, \mathsf{C}_{\overline{q}}(\mathfrak{s}',\mathfrak{h}'_2))$ is equal to $min(\alpha_2, \mathsf{C}_{\overline{q}}(\mathfrak{s},\mathfrak{h}_2))$.

Below we analyse the Case 3 (we omit the Case 4 since its treatment is symmetrical). $\mathsf{C}_{\overline{q}}(\mathfrak{s},\mathfrak{h}_1) = \mathsf{C}_{\overline{q}}(\mathfrak{s}',\mathfrak{h}'_1) < \alpha_1$ and therefore $\mathsf{C}_{\overline{q}}(\mathfrak{s}',\mathfrak{h}') - \mathsf{C}_{\overline{q}}(\mathfrak{s}',\mathfrak{h}'_1) > \alpha_2$. Consequently, $\mathsf{C}_{\overline{q}}(\mathfrak{s}',\mathfrak{h}'_2) > \alpha_2$, whence $min(\alpha_2, \mathsf{C}_{\overline{q}}(\mathfrak{s}',\mathfrak{h}'_2)) = min(\alpha_2, \mathsf{C}_{\overline{q}}(\mathfrak{s},\mathfrak{h}_2)) = \alpha_2$. **QED**

Given a memory state $(\mathfrak{s},\mathfrak{h})$, we write $\mathtt{maxval}(\mathfrak{s},\mathfrak{h})$ to denote the maximal value $max(\mathrm{ran}(\mathfrak{s}) \cup \mathrm{dom}(\mathfrak{h}) \cup \mathrm{ran}(\mathfrak{h}))$.

Lemma 5.3.4. Let $\alpha \in \mathbb{N}$ and $(\mathfrak{s},\mathfrak{h})$, $(\mathfrak{s}',\mathfrak{h}')$ be memory states such that $(\mathfrak{s},\mathfrak{h}) \approx^q_\alpha (\mathfrak{s}',\mathfrak{h}')$. For any heap $\mathfrak{h}_1$ disjoint from $\mathfrak{h}$, there is a heap $\mathfrak{h}'_1$ disjoint from $\mathfrak{h}'$ such that the conditions below hold:

(I) $(\mathfrak{s},\mathfrak{h}_1) \approx^q_\alpha (\mathfrak{s}',\mathfrak{h}'_1)$.

(II) $(\mathfrak{s},\mathfrak{h} \uplus \mathfrak{h}_1) \approx^q_\alpha (\mathfrak{s}',\mathfrak{h}' \uplus \mathfrak{h}'_1)$.

(III) $\mathtt{maxval}(\mathfrak{s}',\mathfrak{h}'_1) \leq \mathtt{maxval}(\mathfrak{s}',\mathfrak{h}') + \alpha$.

If we give up the condition (III), the condition (I) can be strengthened by: $(\mathfrak{s}, \mathfrak{h}_1) \approx^q_\beta (\mathfrak{s}', \mathfrak{h}'_1)$ for all $\beta \geq 0$ (i.e. $\mathrm{dom}(\mathfrak{h}_1)$ and $\mathrm{dom}(\mathfrak{h}'_1)$ have the same cardinality).

Proof. Let α, $(\mathfrak{s}, \mathfrak{h})$, $(\mathfrak{s}', \mathfrak{h}')$ and $\mathfrak{h}_1$ be defined as in the statement. Let us explain how to build the heap $\mathfrak{h}'_1$ and then we show that it satisfies the right properties.The heap $\mathfrak{h}'_1$ is built incrementally and therefore its initial value is naturally the empty heap.

- For every $\mathfrak{s}(\mathtt{x}_i) \in \mathrm{dom}(\mathfrak{h}_1)$ with $i \in [1, q]$, if $\mathfrak{h}_1(\mathfrak{s}(\mathtt{x}_i)) = \mathfrak{s}(\mathtt{x}_j)$ for some $j \in [1, q]$, then $\mathfrak{h}'_1(\mathfrak{s}'(\mathtt{x}_i)) \stackrel{\text{def}}{=} \mathfrak{s}'(\mathtt{x}_j)$, otherwise $\mathfrak{h}'_1(\mathfrak{s}'(\mathtt{x}_i)) \stackrel{\text{def}}{=} max\{\mathfrak{s}'(\mathtt{x}_j) \mid j \in [1, q]\} + 1$. Since $(\mathfrak{s}, \mathfrak{h}) \approx^q_\alpha (\mathfrak{s}', \mathfrak{h}')$ and, $\mathfrak{h}$ and $\mathfrak{h}_1$ are disjoint, this implies that the current value of $\mathfrak{h}'_1$ after the addition of all the new memory cells (according to the above definition) is disjoint from $\mathfrak{h}'$. Moreover, $(\mathfrak{s}, \mathfrak{h}_1)$ and $(\mathfrak{s}', \mathfrak{h}'_1)$ agree on test formulae of the form $\mathtt{x}_i = \mathtt{x}_j$, $\mathtt{x}_i \hookrightarrow \mathtt{x}_j$ and $\mathtt{alloc}(\mathtt{x}_i)$.

- Let $\beta_1 = min(\alpha, \mathsf{C}_{\overline{q}}(\mathfrak{s}, \mathfrak{h}_1))$. For every $i \in [1, \beta_1]$, we extend $\mathfrak{h}'_1$ so that

$$\mathfrak{h}'_1(max\{\mathfrak{s}'(\mathtt{x}_j) \mid j \in [1, q]\} + i) \stackrel{\text{def}}{=} max\{\mathfrak{s}'(\mathtt{x}_j) \mid j \in [1, q]\} + i.$$

  Consequently, $min(\alpha, \mathsf{C}_{\overline{q}}(\mathfrak{s}, \mathfrak{h}_1)) = min(\alpha, \mathsf{C}_{\overline{q}}(\mathfrak{s}', \mathfrak{h}'_1))$ and therefore this implies that $(\mathfrak{s}, \mathfrak{h}_1)$ and $(\mathfrak{s}', \mathfrak{h}'_1)$ agree on the test formulae of the form $\mathtt{size}_{\overline{q}} \geq \beta$ with $\beta \in [0, \alpha]$.

From the construction of $\mathfrak{h}'_1$, we can conclude that $(\mathfrak{s}, \mathfrak{h}_1) \approx^q_\alpha (\mathfrak{s}', \mathfrak{h}'_1)$ and

$$\mathtt{maxval}(\mathfrak{s}', \mathfrak{h}'_1) \leq \mathtt{maxval}(\mathfrak{s}', \mathfrak{h}') + \alpha.$$

It remains to show that $(\mathfrak{s}, \mathfrak{h} \uplus \mathfrak{h}_1) \approx^q_\alpha (\mathfrak{s}', \mathfrak{h}' \uplus \mathfrak{h}'_1)$. The satisfaction of the test formulae $\mathtt{x}_i = \mathtt{x}_j$ is inherited from $(\mathfrak{s}, \mathfrak{h}) \approx^q_\alpha (\mathfrak{s}', \mathfrak{h}')$ (because the stores are unchanged) whereas the satisfaction of the test formulae $\mathtt{x}_i \hookrightarrow \mathtt{x}_j$ and $\mathtt{alloc}(\mathtt{x}_i)$ requires simple arguments by distinguishing four cases:

1. $\mathfrak{s}(\mathtt{x}_i) \in \mathrm{dom}(\mathfrak{h})$,
2. $\mathfrak{s}(\mathtt{x}_i) \notin \mathrm{dom}(\mathfrak{h} \uplus \mathfrak{h}_1)$,
3. $\mathfrak{s}(\mathtt{x}_i) \in \mathrm{dom}(\mathfrak{h}_1)$ and $\mathfrak{h}_1(\mathfrak{s}(\mathtt{x}_i)) \notin \{\mathfrak{s}(\mathtt{x}_j) \mid j \in [1, q]\}$ and finally
4. $\mathfrak{s}(\mathtt{x}_i) \in \mathrm{dom}(\mathfrak{h}_1)$ and $\mathfrak{h}_1(\mathfrak{s}(\mathtt{x}_i)) \in \{\mathfrak{s}(\mathtt{x}_j) \mid j \in [1, q]\}$.

Details are omitted. In order to check that $(\mathfrak{s}, \mathfrak{h} \uplus \mathfrak{h}_1)$ and $(\mathfrak{s}', \mathfrak{h}' \uplus \mathfrak{h}'_1)$ agree on test formulae of the form $\mathtt{size}_{\overline{q}} \geq \beta$ with $\beta \in [0, \alpha]$, it is sufficient to observe that $min(\alpha, \mathsf{C}_{\overline{q}}(\mathfrak{s}, \mathfrak{h})) = min(\alpha, \mathsf{C}_{\overline{q}}(\mathfrak{s}', \mathfrak{h}'))$ and $min(\alpha, \mathsf{C}_{\overline{q}}(\mathfrak{s}, \mathfrak{h}_1)) = min(\alpha, \mathsf{C}_{\overline{q}}(\mathfrak{s}', \mathfrak{h}'_1))$ imply

$$min(\alpha, \mathsf{C}_{\overline{q}}(\mathfrak{s}, \mathfrak{h} \uplus \mathfrak{h}_1)) = min(\alpha, \mathsf{C}_{\overline{q}}(\mathfrak{s}', \mathfrak{h}' \uplus \mathfrak{h}'_1))$$

since $\mathsf{C}_{\overline{q}}(\mathfrak{s}, \mathfrak{h} \uplus \mathfrak{h}_1) = \mathsf{C}_{\overline{q}}(\mathfrak{s}, \mathfrak{h}) + \mathsf{C}_{\overline{q}}(\mathfrak{s}, \mathfrak{h}_1)$ and $\mathsf{C}_{\overline{q}}(\mathfrak{s}', \mathfrak{h}' \uplus \mathfrak{h}'_1) = \mathsf{C}_{\overline{q}}(\mathfrak{s}', \mathfrak{h}') + \mathsf{C}_{\overline{q}}(\mathfrak{s}', \mathfrak{h}'_1)$. **QED**

For each formula φ, we define its **memory size** $\mathtt{msize}(\varphi)$ following the clauses below [Yan01]. This is just a refinement of the size of φ when formulae are represented by their syntactic trees.

$$\begin{array}{lcl}
\mathtt{msize}(\mathtt{x}_i \hookrightarrow \mathtt{x}_{i'}) & \stackrel{\text{def}}{=} & 1 \\
\mathtt{msize}(\mathtt{x}_i = \mathtt{x}_{i'}) & \stackrel{\text{def}}{=} & 0 \\
\mathtt{msize}(\mathtt{emp}) & \stackrel{\text{def}}{=} & 1 \\
\mathtt{msize}(\neg\psi) & \stackrel{\text{def}}{=} & \mathtt{msize}(\psi) \\
\mathtt{msize}(\psi_1 \wedge \psi_2) & \stackrel{\text{def}}{=} & max(\mathtt{msize}(\psi_1), \mathtt{msize}(\psi_2)) \\
\mathtt{msize}(\psi_1 * \psi_2) & \stackrel{\text{def}}{=} & \mathtt{msize}(\psi_1) + \mathtt{msize}(\psi_2) \\
\mathtt{msize}(\psi_1 \mathrel{-\!\!*} \psi_2) & \stackrel{\text{def}}{=} & max(\mathtt{msize}(\psi_1), \mathtt{msize}(\psi_2)).
\end{array}$$

For instance $\mathtt{msize}(\mathtt{size} = 3) = 4$ when $\mathtt{size} = 3$ is defined in 1SL0 by the formula below:

$$((\neg\mathtt{emp}) * (\neg\mathtt{emp}) * (\neg\mathtt{emp})) \wedge \neg((\neg\mathtt{emp}) * (\neg\mathtt{emp}) * (\neg\mathtt{emp}) * (\neg\mathtt{emp})).$$

The memory size is bounded above by the size when formulae are encoded as trees.

Lemma 5.3.5. For every φ in 1SL0, $\mathtt{msize}(\varphi)$ is smaller that the size of φ, assuming that formulae are encoded as trees.

Below, we state the main result about the equivalence relation $\approx^q_\alpha$: two memory states in the relation cannot be distinguished by formulae of memory size smaller than α.

Theorem 5.3.6. Let φ be a formula in 1SL0 built over the program variables $\mathtt{x}_1$, ..., $\mathtt{x}_q$. For any $\alpha \in \mathbb{N}$ such that $\mathtt{msize}(\varphi) \leq \alpha$, and for all memory states $(\mathfrak{s}, \mathfrak{h})$, $(\mathfrak{s}', \mathfrak{h}')$ such that $(\mathfrak{s}, \mathfrak{h}) \approx^q_\alpha (\mathfrak{s}', \mathfrak{h}')$, we have $(\mathfrak{s}, \mathfrak{h}) \models \varphi$ iff $(\mathfrak{s}', \mathfrak{h}') \models \varphi$.

Proof. Suppose that $(\mathfrak{s},\mathfrak{h}) \approx^{q}_{\alpha} (\mathfrak{s}',\mathfrak{h}')$ and φ be a formula with $\mathtt{msize}(\varphi) \leq \alpha$. By structural induction, we show that $(\mathfrak{s},\mathfrak{h}) \models \varphi$ if and only if $(\mathfrak{s}',\mathfrak{h}') \models \varphi$. It is sufficient to establish one direction of the equivalence thanks to the symmetry.

The base case with the atomic formulae of the form $\mathtt{x}_i \hookrightarrow \mathtt{x}_j$, $\mathtt{x}_i = \mathtt{x}_j$ or $\mathtt{emp}$ is by an easy verification due to the very definition of the test formulae. Indeed, $\mathtt{x}_i \hookrightarrow \mathtt{x}_j$ and $\mathtt{x}_i = \mathtt{x}_j$ are already test formulae in $\mathtt{Test}'(q,\alpha)$ whereas $\mathtt{emp}$ is logically equivalent to

$$(\bigwedge_{i\in[1,q]} \neg\mathtt{alloc}(\mathtt{x}_i)) \wedge \neg(\mathtt{size}_{\overline{q}} \geq 1).$$

In the induction step, the cases with Boolean connectives are even more straightforward to prove.
Case 1: $\psi = \psi_1 * \psi_2$.
Suppose that $(\mathfrak{s},\mathfrak{h}) \models \psi_1 * \psi_2$ and $\mathtt{msize}(\psi_1 * \psi_2) \leq \alpha$. There are heaps $\mathfrak{h}_1$ and $\mathfrak{h}_2$ such that $\mathfrak{h} = \mathfrak{h}_1 \uplus \mathfrak{h}_2$, $(\mathfrak{s},\mathfrak{h}_1) \models \psi_1$ and $(\mathfrak{s},\mathfrak{h}_2) \models \psi_2$. As $\alpha \geq \mathtt{msize}(\psi_1 * \psi_2) = \mathtt{msize}(\psi_1) + \mathtt{msize}(\psi_2)$, there exist α_1 and α_2 such that $\alpha = \alpha_1 + \alpha_2$, $\alpha_1 \geq \mathtt{msize}(\psi_1)$ and $\alpha_2 \geq \mathtt{msize}(\psi_2)$. By Lemma 5.3.3, there exist heaps $\mathfrak{h}'_1$ and $\mathfrak{h}'_2$ such that $\mathfrak{h}' = \mathfrak{h}'_1 \uplus \mathfrak{h}'_2$, $(\mathfrak{s},\mathfrak{h}_1) \approx^{q}_{\alpha_1} (\mathfrak{s}',\mathfrak{h}'_1)$ and $(\mathfrak{s},\mathfrak{h}_2) \approx^{q}_{\alpha_2} (\mathfrak{s}',\mathfrak{h}'_2)$. By the induction hypothesis, we get $(\mathfrak{s}',\mathfrak{h}'_1) \models \psi_1$ and $(\mathfrak{s}',\mathfrak{h}'_2) \models \psi_2$. Consequently, we obtain $(\mathfrak{s}',\mathfrak{h}') \models \psi_1 * \psi_2$.
Case 2: $\psi = \psi_1 \mathrel{-\!\!*} \psi_2$.
Suppose that $(\mathfrak{s},\mathfrak{h}) \models \psi_1 \mathrel{-\!\!*} \psi_2$ and $\mathtt{msize}(\psi_1 \mathrel{-\!\!*} \psi_2) \leq \alpha$. Since $\mathtt{msize}(\psi_1 \mathrel{-\!\!*} \psi_2) = \mathtt{msize}(\psi_2)$, we also get that $\mathtt{msize}(\psi_1), \mathtt{msize}(\psi_2) \leq \alpha$.

Let us prove that $(\mathfrak{s}',\mathfrak{h}') \models \psi_1 \mathrel{-\!\!*} \psi_2$. Let $\mathfrak{h}'_1$ be a heap disjoint from $\mathfrak{h}'$ such that $(\mathfrak{s}',\mathfrak{h}'_1) \models \psi_1$. By Lemma 5.3.4, there is a heap $\mathfrak{h}_1$ disjoint from $\mathfrak{h}$ such that $(\mathfrak{s},\mathfrak{h}_1) \approx^{q}_{\alpha} (\mathfrak{s}',\mathfrak{h}'_1)$ and $(\mathfrak{s},\mathfrak{h} \uplus \mathfrak{h}_1) \approx^{q}_{\alpha} (\mathfrak{s}',\mathfrak{h}' \uplus \mathfrak{h}'_1)$. By the induction hypothesis, we conclude that $(\mathfrak{s},\mathfrak{h}_1) \models \psi_1$. Since $(\mathfrak{s},\mathfrak{h}) \models \psi_1 \mathrel{-\!\!*} \psi_2$, this implies that $(\mathfrak{s},\mathfrak{h} \uplus \mathfrak{h}_1) \models \psi_2$. By the induction hypothesis, we conclude that $(\mathfrak{s}',\mathfrak{h}' \uplus \mathfrak{h}'_1) \models \psi_2$. Since $\mathfrak{h}'_1$ is an arbitrary disjoint heap from $\mathfrak{h}'$, we obtain $(\mathfrak{s}',\mathfrak{h}') \models \psi_1 \mathrel{-\!\!*} \psi_2$. **QED**

Theorem 5.3.7. Let φ be a formula in 1SL0 built over the variables in $\mathtt{x}_1, \ldots, \mathtt{x}_q$. The formula φ is logically equivalent to a Boolean combination of test formulae from $\mathtt{Test}(q, q + \mathtt{msize}(\varphi))$.

Theorem 5.3.7 can be viewed as a means to eliminate separating connectives $*$ and $\mathrel{-\!\!*}$ and this is analogous to quantifier elimination in Presburger arithmetic [Pre29] for which periodicity constraints need to be introduced in order to eliminate the quantifiers (see e.g. [Coo72]). Similarly, the atomic formulae

$\mathtt{size} \geq k$ and $\mathtt{alloc}(\mathtt{x}_i)$ would require the use of the separating connectives to be properly defined in 1SL0 but in the Boolean combinations, these formulae are understood as primitive.

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

$$\{\chi \in \mathtt{Test}'(q, \alpha) \mid (\mathfrak{s}, \mathfrak{h}) \models \chi\} \cup \{\neg\chi \mid (\mathfrak{s}, \mathfrak{h}) \not\models \chi \text{ with } \chi \in \mathtt{Test}'(q, \alpha)\}.$$

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

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

The expression below

$$\psi' \stackrel{\text{def}}{=} \bigvee_{(\mathfrak{s},\mathfrak{h}) \models \varphi} \Big(\bigwedge_{\psi \in \mathrm{LIT}(\mathfrak{s},\mathfrak{h})} \psi \Big)$$

is equivalent to a Boolean combination φ' of formulae from $\mathtt{Test}'(q, \alpha)$ because $\mathrm{LIT}(\mathfrak{s}, \mathfrak{h})$ ranges over the finite set of elements from $\mathtt{Test}'(q, \alpha)$ (just select a finite amount of disjuncts). By Theorem 5.3.6, the formula φ is logically equivalent to φ', which concludes the proof since any formula of the form $\mathtt{size}_{\overline{q}} \geq \beta$ with $\beta \leq \alpha$ is logically equivalent to a Boolean combination of test formulae from $\mathtt{Test}(q, q + \alpha)$.

Indeed, suppose that $(\mathfrak{s}, \mathfrak{h}) \models \varphi$. Obviously, this implies that $(\mathfrak{s}, \mathfrak{h}) \models \bigwedge_{\psi \in \mathrm{LIT}(\mathfrak{s},\mathfrak{h})} \psi$ and therefore $(\mathfrak{s}, \mathfrak{h}) \models \varphi'$. Conversely, suppose that $(\mathfrak{s}, \mathfrak{h}) \models \varphi'$. This means that there is a memory state $(\mathfrak{s}', \mathfrak{h}')$ such that $(\mathfrak{s}', \mathfrak{h}') \models \varphi$ and $(\mathfrak{s}, \mathfrak{h}) \models \bigwedge_{\psi \in \mathrm{LIT}(\mathfrak{s},\mathfrak{h})} \psi$. Since $(\mathfrak{s}, \mathfrak{h}) \approx^{q}_{\alpha} (\mathfrak{s}', \mathfrak{h}')$, $\mathtt{msize}(\varphi) \leq \alpha$ and $(\mathfrak{s}', \mathfrak{h}') \models \varphi$, by Theorem 5.3.6 we get $(\mathfrak{s}, \mathfrak{h}) \models \varphi$. **QED**

Now, it is possible to establish the small model property.

Corollary 5.3.8. Let φ be a satisfiable 1SL0 formula built over $\mathtt{x}_1, \ldots, \mathtt{x}_q$. There is a memory state $(\mathfrak{s}, \mathfrak{h})$ such that $(\mathfrak{s}, \mathfrak{h}) \models \varphi$ and $\mathtt{maxval}(\mathfrak{s}, \mathfrak{h}) \leq q + \mathtt{msize}(\varphi)$.

The proof is left as Exercise 5.10.

5.3.2 A model-checking decision procedure

In order to check the satisfiability status of φ, only the truth value of formulae in $\mathtt{Test}'(q, \mathtt{msize}(\varphi))$ matters. That is why, instead of operating on memory states to check satisfiability, it is sufficient to operate on its abstractions (with respect to the basic properties induced by $\mathtt{Test}'(q, \mathtt{msize}(\varphi))$), whence the introduction of symbolic memory states below. A **symbolic memory state** $\mathtt{sms}$ over (q, α) is a finite structure $(P, A, H, \mathfrak{n})$ such that:

1. P is a partition of $\{\mathtt{x}_1, \ldots, \mathtt{x}_q\}$ (encoding equalities) and $A \subseteq P$ (encoding the subset in the domain).

2. H is a functional relation on P such that $\mathrm{dom}(H) = A$.

3. $\mathfrak{n} \in [0, \alpha]$ and this corresponds to the number of locations in the heap domain that are not equal to the interpretation of some program variables in $\{\mathtt{x}_1, \ldots, \mathtt{x}_q\}$. For values strictly greater than α, a truncation is considered.

Given $q \geq 1$ and $\alpha \in \mathbb{N}$, the number of symbolic memory states over (q, α) is "only" exponential in $q + \alpha$. Given a memory state $(\mathfrak{s}, \mathfrak{h})$, we define its **abstraction** $\mathtt{Symb}[\mathfrak{s}, \mathfrak{h}]$ over (q, α) as the symbolic memory state $(P, A, H, \mathfrak{n})$ such that

- $\mathfrak{n} = min(\alpha, \mathrm{card}(\mathrm{dom}(\mathfrak{h}) \setminus \{\mathfrak{s}(\mathtt{x}_i) \mid i \in [1, q]\}))$.

- P is a partition of $\{\mathtt{x}_1, \ldots, \mathtt{x}_q\}$ so that for all $\mathtt{x}, \mathtt{x}'$, we have $\mathfrak{s}(\mathtt{x}) = \mathfrak{s}(\mathtt{x}')$ iff $\mathtt{x}$ and $\mathtt{x}'$ belong to the same set in P.

- $A = \{X \in P \mid \text{there is } \mathtt{x} \in X,\ \mathfrak{s}(\mathtt{x}) \in \mathrm{dom}(\mathfrak{h})\}$.

- $X\ H\ X'$ iff there are $\mathtt{x} \in X$ and $\mathtt{x}' \in X'$ such that $\mathfrak{h}(\mathfrak{s}(\mathtt{x})) = \mathfrak{s}(\mathtt{x}')$.

Note that given a symbolic memory state $\mathtt{sms}$ over (q, α), there exists always a memory state $(\mathfrak{s}, \mathfrak{h})$ such that $\mathtt{Symb}[\mathfrak{s}, \mathfrak{h}]$ is equal to $\mathtt{sms}$. Not only every symbolic memory state has always a concretisation but also symbolic memory states are the right way to abstract memory states when the language 1SL0 is involved, which can be formally stated as follows: $(\mathfrak{s}, \mathfrak{h}) \approx^q_\alpha (\mathfrak{s}', \mathfrak{h}')$ iff $\mathtt{Symb}[\mathfrak{s}, \mathfrak{h}] = \mathtt{Symb}[\mathfrak{s}', \mathfrak{h}']$.

Definition 5.3.9. Given symbolic memory states $\mathtt{sms}$, $\mathtt{sms}_1$ and $\mathtt{sms}_2$, we write $*_{\mathfrak{s}}(\mathtt{sms}, \mathtt{sms}_1, \mathtt{sms}_2)$ whenever there exist a store $\mathfrak{s}$ and disjoint heaps $\mathfrak{h}_1$ and $\mathfrak{h}_2$ such that $\mathtt{Symb}[\mathfrak{s}, \mathfrak{h}_1 \uplus \mathfrak{h}_2] = \mathtt{sms}$, $\mathtt{Symb}[\mathfrak{s}, \mathfrak{h}_1] = \mathtt{sms}_1$ and $\mathtt{Symb}[\mathfrak{s}, \mathfrak{h}_2] = \mathtt{sms}_2$. ∇

1. **if** ψ is atomic **then** return AMC($\mathtt{sms}, \psi$);
2. **if** $\psi = \neg\psi_1$ **then** return not MC($\mathtt{sms}, \psi_1$);
3. **if** $\psi = \psi_1 \wedge \psi_2$ **then** return (MC($\mathtt{sms}, \psi_1$) and MC($\mathtt{sms}, \psi_2$));
4. **if** $\psi = \psi_1 * \psi_2$ **then** return $\top$ iff there are $\mathtt{sms}_1$ and $\mathtt{sms}_2$ such that $*_s(\mathtt{sms}, \mathtt{sms}_1, \mathtt{sms}_2)$ and MC($\mathtt{sms}_1, \psi_1$) = MC($\mathtt{sms}_2, \psi_2$) = $\top$;
5. **if** $\psi = \psi_1 \mathrel{-\!\!*} \psi_2$ **then** return $\bot$ iff for some $\mathtt{sms}'$ and $\mathtt{sms}''$ such that $*_s(\mathtt{sms}'', \mathtt{sms}', \mathtt{sms})$, MC($\mathtt{sms}', \psi_1$) = $\top$ and MC($\mathtt{sms}'', \psi_2$) = $\bot$;

Figure 5.1: Function MC($\mathtt{sms}, \psi$)

Given $q \geq 1$ and $\alpha \in \mathbb{N}$, the ternary relation $*_s$ can be decided in polynomial time in $q + log(\alpha)$ for all the symbolic memory states built over (q, α). Indeed, assuming that $\mathtt{sms} = (P, A, H, \mathfrak{n})$ and $\mathtt{sms}_i = (P_i, A_i, H_i, \mathfrak{n}_i)$, the relation $*_s(\mathtt{sms}, \mathtt{sms}_1, \mathtt{sms}_2)$ holds exactly when the conditions below are satisfied:

1. $P = P_1 = P_2$,
2. $A = A_1 \cup A_2$ and $A_1 \cap A_2 = \emptyset$,
3. $H = H_1 \cup H_2$,
4. $\mathfrak{n} = \max(\alpha, \mathfrak{n}_1 + \mathfrak{n}_2)$.

A formal proof is left as Exercise 5.12. Note that there is no need to specify that $H_1 \cap H_2 = \emptyset$ since this is a consequence of $A_1 \cap A_2 = \emptyset$, $\mathrm{dom}(H_1) \subseteq A_1$ and $\mathrm{dom}(H_2) \subseteq A_2$.

Figure 5.1 presents a procedure MC($\mathtt{sms}, \psi$) returning a Boolean value in $\{\bot, \top\}$ and taking as arguments, a symbolic memory state over (q, α) and a formula ψ whose size is bounded above by α. All the quantifications over symbolic memory states are done over (q, α). A case analysis is provided depending on the outermost connective of the input formula. Its structure is standard and mimicks faithfully the semantics for 1SL0 *except* that we deal with symbolic memory states. The auxiliary function AMC($\mathtt{sms}, \psi$) also returns a Boolean value in $\{\bot, \top\}$, makes no recursive calls and is dedicated to atomic formulae (see Figure 5.2). The design of MC is similar to nondeterministic polynomial-space procedures, see e.g. [Lad77, Spa93, COY01, Dem03].

1. **if** ψ is $\mathtt{emp}$ **then** return $\top$ iff $A = \emptyset$ and $\mathfrak{n} = 0$;
2. **if** ψ is $\mathtt{x}_i = \mathtt{x}_j$ **then** return $\top$ iff $\mathtt{x}_i, \mathtt{x}_j \in X$, for some $X \in P$;
3. **if** ψ is $\mathtt{x}_i \hookrightarrow \mathtt{x}_j$ **then** return $\top$ iff $(X, X') \in H$ where $\mathtt{x}_i \in X \in P$ and $\mathtt{x}_j \in X' \in P$;

Figure 5.2: Function AMC($\mathtt{sms}, \psi$)

Lemma 5.3.10. Let $q \geq 1$, $\alpha \in \mathbb{N}$, $\mathtt{sms}$ be a symbolic memory state over (q, α) and φ be in 1SL0 built over $\mathtt{x}_1, \ldots, \mathtt{x}_q$ such that $\mathtt{msize}(\varphi) \leq \alpha$. We have MC($\mathtt{sms}, \varphi$) returns $\top$ iff there exists $(\mathfrak{s}, \mathfrak{h})$ such that $\mathtt{Symb}[\mathfrak{s}, \mathfrak{h}] = \mathtt{sms}$ and $(\mathfrak{s}, \mathfrak{h}) \models \varphi$.

Proof. Let us show the equivalence by structural induction. We omit below the base case with atomic formulae and the case with negated subformulae.

First, let us make a basic observation. Indeed, given a symbolic memory state $\mathtt{sms}$, the statements below are equivalent:

- there exists $(\mathfrak{s}, \mathfrak{h})$ such that $\mathtt{Symb}[\mathfrak{s}, \mathfrak{h}] = \mathtt{sms}$ and $(\mathfrak{s}, \mathfrak{h}) \models \varphi$,
- for all memory states $(\mathfrak{s}, \mathfrak{h})$ such that $\mathtt{Symb}[\mathfrak{s}, \mathfrak{h}] = \mathtt{sms}$, we have $(\mathfrak{s}, \mathfrak{h}) \models \varphi$.

Indeed, $\mathtt{Symb}[\mathfrak{s}, \mathfrak{h}] = \mathtt{Symb}[\mathfrak{s}', \mathfrak{h}']$ implies $(\mathfrak{s}, \mathfrak{h}) \approx^q_\alpha (\mathfrak{s}', \mathfrak{h}')$ and therefore we conclude $(\mathfrak{s}, \mathfrak{h}) \approx^q_{\mathtt{msize}(\varphi)} (\mathfrak{s}', \mathfrak{h}')$ by Lemma 5.3.2 since $\mathtt{msize}(\varphi) \leq \mathtt{msize}(\varphi) \leq \alpha$. By Theorem 5.3.6, we get $(\mathfrak{s}, \mathfrak{h}) \models \varphi$ iff $(\mathfrak{s}', \mathfrak{h}') \models \varphi$.
Case 1: $\varphi = \varphi_1 \wedge \varphi_2$.
Suppose that $(\mathfrak{s}, \mathfrak{h}) \models \varphi$ and $\mathtt{Symb}[\mathfrak{s}, \mathfrak{h}] = \mathtt{sms}$. So, $(\mathfrak{s}, \mathfrak{h}) \models \varphi_1$ and $(\mathfrak{s}, \mathfrak{h}) \models \varphi_2$, and by the induction hypothesis, both MC($\mathtt{sms}, \varphi_1$) and MC($\mathtt{sms}, \varphi_2$) return $\top$ ($\mathtt{msize}(\varphi_1), \mathtt{msize}(\varphi_2) \leq \alpha$ too). Hence, MC($\mathtt{sms}, \varphi$) returns $\top$.

Conversely, suppose that MC($\mathtt{sms}, \varphi$) returns $\top$. By definition of MC, we have MC($\mathtt{sms}, \varphi_1$) and MC($\mathtt{sms}, \varphi_2$). For all memory states $(\mathfrak{s}', \mathfrak{h}')$ such that $\mathtt{Symb}[\mathfrak{s}', \mathfrak{h}'] = \mathtt{sms}$, we have $(\mathfrak{s}', \mathfrak{h}') \models \varphi_1$ and $(\mathfrak{s}', \mathfrak{h}') \models \varphi_2$. We have seen that every symbolic memory state admits a concretisation. Consequently, there is a memory state $(\mathfrak{s}, \mathfrak{h})$ such that $\mathtt{Symb}[\mathfrak{s}, \mathfrak{h}] = \mathtt{sms}$. By the induction hypothesis, $(\mathfrak{s}, \mathfrak{h}) \models \varphi_1$ and $(\mathfrak{s}, \mathfrak{h}) \models \varphi_2$ and therefore $(\mathfrak{s}, \mathfrak{h}) \models \varphi$.
Case 2: $\varphi = \varphi_1 * \varphi_2$.
Suppose that $(\mathfrak{s}, \mathfrak{h}) \models \varphi$ and $\mathtt{Symb}[\mathfrak{s}, \mathfrak{h}] = \mathtt{sms}$. So, there are subheaps $\mathfrak{h}_1$ and $\mathfrak{h}_2$ such that $\mathfrak{h} = \mathfrak{h}_1 \uplus \mathfrak{h}_2$, $(\mathfrak{s}, \mathfrak{h}_1) \models \varphi_1$ and $(\mathfrak{s}, \mathfrak{h}_2) \models \varphi_2$. By the induction hypothesis,

$\mathrm{MC}(\mathtt{Symb}[\mathfrak{s},\mathfrak{h}_1],\varphi_1)$ and $\mathrm{MC}(\mathtt{Symb}[\mathfrak{s},\mathfrak{h}_2],\varphi_2)$ return $\top$. By definition of $*_s$, we have

$$*_s(\mathtt{Symb}[\mathfrak{s},\mathfrak{h}],\mathtt{Symb}[\mathfrak{s},\mathfrak{h}_1],\mathtt{Symb}[\mathfrak{s},\mathfrak{h}_2])$$

and therefore there are $\mathtt{sms}_1$ and $\mathtt{sms}_2$ such that $*_s(\mathtt{sms},\mathtt{sms}_1,\mathtt{sms}_2)$ and $\mathrm{MC}(\mathtt{sms}_1,\varphi_1)$ $= \mathrm{MC}(\mathtt{sms}_2,\varphi_2) = \top$, whence $\mathrm{MC}(\mathtt{sms},\varphi)$ returns $\top$.

Conversely we assume $\mathtt{sms}_1$, $\mathtt{sms}_2$ such that

$$*_s(\mathtt{sms},\mathtt{sms}_1,\mathtt{sms}_2) \text{ and } \mathrm{MC}(\mathtt{sms}_1,\varphi_1) = \mathrm{MC}(\mathtt{sms}_2,\varphi_2) = \top.$$

By definition of $*_s$, there exist a store $\mathfrak{s}$ and disjoint heaps $\mathfrak{h}_1$ and $\mathfrak{h}_2$ such that $\mathtt{Symb}[\mathfrak{s},\mathfrak{h}_1 \uplus \mathfrak{h}_2] = \mathtt{sms}$, $\mathtt{Symb}[\mathfrak{s},\mathfrak{h}_1] = \mathtt{sms}_1$, $\mathtt{Symb}[\mathfrak{s},\mathfrak{h}_2] = \mathtt{sms}_2$. By the induction hypothesis, we get that $(\mathfrak{s},\mathfrak{h}_1) \models \varphi_1$ and $(\mathfrak{s},\mathfrak{h}_2) \models \varphi_2$. Consequently, $(\mathfrak{s},\mathfrak{h}) \models \varphi$.
Case 3: $\varphi = \varphi_1 \mathrel{-\!\!*} \varphi_2$. Suppose that $(\mathfrak{s},\mathfrak{h}) \not\models \varphi$ and $\mathtt{Symb}[\mathfrak{s},\mathfrak{h}] = \mathtt{sms}$. There is a heap $\mathfrak{h}'$ disjoint from $\mathfrak{h}$ such that $(\mathfrak{s},\mathfrak{h}') \models \varphi_1$ and $(\mathfrak{s},\mathfrak{h} \uplus \mathfrak{h}') \not\models \varphi_2$. By definition of $*_s$, we have $*_s(\mathtt{Symb}[\mathfrak{s},\mathfrak{h} \uplus \mathfrak{h}'],\mathtt{Symb}[\mathfrak{s},\mathfrak{h}],\mathtt{Symb}[\mathfrak{s},\mathfrak{h}'])$. By the induction hypothesis, we have $\mathrm{MC}(\mathtt{Symb}[\mathfrak{s},\mathfrak{h}'],\varphi_1)$ returns $\top$ and $\mathrm{MC}(\mathtt{Symb}[\mathfrak{s},\mathfrak{h} \uplus \mathfrak{h}'],\varphi_2)$ returns $\bot$. So, for some $\mathtt{sms}'$ and $\mathtt{sms}''$ such that $*_s(\mathtt{sms}'',\mathtt{sms}',\mathtt{sms})$, we get $\mathrm{MC}(\mathtt{sms}',\varphi_1)$ $= \top$ and $\mathrm{MC}(\mathtt{sms}'',\varphi_2) = \bot$, whence $\mathrm{MC}(\mathtt{sms},\varphi)$ returns $\bot$.

Conversely, suppose that $\mathrm{MC}(\mathtt{sms},\varphi)$ returns $\bot$. For some $\mathtt{sms}'$ and $\mathtt{sms}''$ such that $*_s(\mathtt{sms}'',\mathtt{sms}',\mathtt{sms})$ and $\mathrm{MC}(\mathtt{sms}',\varphi_1) = \top$ and $\mathrm{MC}(\mathtt{sms}'',\varphi_2) = \bot$. By definition of $*_s$, there exist a store $\mathfrak{s}$ and disjoint heaps $\mathfrak{h}_1$ and $\mathfrak{h}_2$ such that $\mathtt{Symb}[\mathfrak{s},\mathfrak{h}_1 \uplus \mathfrak{h}_2] = \mathtt{sms}''$, $\mathtt{Symb}[\mathfrak{s},\mathfrak{h}_1] = \mathtt{sms}$, $\mathtt{Symb}[\mathfrak{s},\mathfrak{h}_2] = \mathtt{sms}'$. By the induction hypothesis and by using that preliminary equivalence, we have $(\mathfrak{s},\mathfrak{h}_1 \uplus \mathfrak{h}_2) \not\models \varphi_2$ and $(\mathfrak{s},\mathfrak{h}_2) \models \varphi_1$ whence $(\mathfrak{s},\mathfrak{h}) \not\models \varphi$.

For the correction of the proof of the last case, we crucially need to use that $\mathtt{msize}(\varphi_1 \mathrel{-\!\!*} \varphi_2) \leq \alpha$ implies that $\mathtt{msize}(\varphi_2) \leq \alpha$ but also $\mathtt{msize}(\varphi_1) \leq \alpha$ –this last inequality would not necessarily hold with the size function $\mathtt{msize}(\cdot)$. **QED**

Consequently, we get the following complexity characterisation

Theorem 5.3.11. [COY01] Model-checking and satisfiability problems for 1SL0 are in PSPACE.

Proof. PSPACE upper bound is a consequence of Lemma 5.3.10 by recalling that φ is satisfiable iff there is a memory state $(\mathfrak{s},\mathfrak{h})$ such that $(\mathfrak{s},\mathfrak{h}) \models \varphi$ iff there is a symbolic memory state $\mathtt{sms}$ over $(q,\mathtt{msize}(\varphi))$ such that $\mathrm{MC}(\mathtt{sms},\varphi) = \top$. It is sufficient to guess $\mathtt{sms}$ and then check whether $\mathrm{MC}(\mathtt{sms},\varphi)$ returns $\top$. All of this can be done in non-deterministic polynomial space and by Savitch's Theorem [Sav70], we get the PSPACE upper bound.

We observe that MC($\mathtt{sms}, \varphi$) runs in space $\mathcal{O}(d(q + log(\alpha)))$ where d is the depth of syntactic tree for φ since the quantifications are over symbolic memory states over (q, α), there is an exponential amount in $q + \alpha$ and each symbolic memory state can be encoded with space in $\mathcal{O}(q+log(\alpha))$. Moreover, the recursion depth of MC is linear in d, which is itself linear in the size of φ. Hence, all the symbolic memory states considered in the algorithm are of polynomial size in the size of φ.

For the model-checking problem, it is sufficient to call MC with the abstraction of the memory state. Indeed, $(\mathfrak{s}, \mathfrak{h}) \models \varphi$ iff MC($\mathtt{Symb}[\mathfrak{s}, \mathfrak{h}], \varphi$) returns true with $\alpha = \mathtt{msize}(\varphi)$ (by Lemma 5.3.10). Now, $\mathtt{Symb}[\mathfrak{s}, \mathfrak{h}]$ and α can be constructed in polynomial space and MC runs in polynomial space too. **QED**

PSPACE-hardness has been presented in Section 2.1.2. An alternative way to obtain the PSPACE upper bound has been proposed in [Hag04, CGH05] by a logarithmic-space reduction to the first-order theory of equality shown in PSPACE in [Sto77] (see also Section 5.4).

Corollary 5.3.12. Given a formula φ in 1SL0, computing a Boolean combination of atomic formulae from $\mathtt{Test}(q, q + \mathtt{msize}(\varphi))$ logically equivalent to φ can be done in polynomial space.

Proof. This is actually a quite direct consequence of the proof of Theorem 5.3.7. Let φ be a 1SL0 formula with $\alpha = \mathtt{msize}(\varphi)$. By the proof of Theorem 5.3.7 and Corollary 5.3.8, φ is logically equivalent to the formula below

$$\bigvee \{ (\bigwedge_{\psi \in \mathrm{LIT}(\mathfrak{s},\mathfrak{h})} \psi) \mid \mathrm{MC}(\mathtt{Symb}[\mathfrak{s}, \mathfrak{h}], \varphi) = \top \text{ and } \mathtt{maxval}(\mathfrak{s}, \mathfrak{h}) \leq q + \alpha \}.$$

Note that $\mathtt{Symb}[\mathfrak{s}, \mathfrak{h}]$ is computed with q and $\mathtt{msize}(\varphi)$.

The non-isomorphic copies of memory states $(\mathfrak{s}, \mathfrak{h})$ such that $\mathtt{maxval}(\mathfrak{s}, \mathfrak{h}) \leq q + \alpha$ can be enumerated in polynomial space. Moreover, the model-checking problem for 1SL0 is in PSPACE; so the above formula can be built in polynomial space (but its size may be exponential in the size of φ) by taking advantage of the fact that $\bigwedge_{\psi \in \mathrm{LIT}(\mathfrak{s},\mathfrak{h})}$ can be constructed in polynomial space. **QED**

As by-product, we can also establish the existence of a family of PTIME fragments of 1SL0.

Corollary 5.3.13. Let $q \geq 1$. The satisfiability problem for 1SL0 restricted to formulae with at most q program variables can be solved in polynomial time.

Proof. Let φ be a formula in 1SL0 with at most q program variables. When q is fixed, the number of symbolic memory states over $(q, \mathtt{msize}(\varphi))$ is polynomial in the size of φ. To check the satisfiability status of the formula φ, for each symbolic memory state $\mathtt{sms}$, we check whether $\mathrm{MC}(\mathtt{sms}, \varphi)$ returns $\top$. To do so, we use standard principles from dynamic programming and we maintain a table $\mathtt{A}[\mathtt{sms}, \psi]$ taking values in $\{\mathtt{unknown}, \top, \bot\}$ to memorize the value returned by $\mathrm{MC}(\mathtt{sms}, \psi)$ for each subformula of ψ and for each symbolic memory state $\mathtt{sms}$ over $(q, \mathtt{msize}(\varphi))$. All the initial values are equal to $\mathtt{unknown}$. We launch a new call to $\mathrm{MC}(\mathtt{sms}, \psi)$ only when $\mathtt{A}[\mathtt{sms}, \psi] = \mathtt{unknown}$. Moreover, before returning a value with MC, we update the table $\mathtt{A}$ accordingly. Since the table has a polynomial number of entries in the size of φ, we have that φ is satisfiable iff there is a symbolic memory state such that $\mathtt{A}[\mathtt{sms}, \varphi] = \top$, which can be verified in polynomial time in the size of φ. **QED**

All the above results can be extended to $k > 1$ (see Exercise 5.14) by adequately adapting the previous developments.

5.4 Translation into QBF

Symbolic memory states abstract memory states and their encoding requires a polynomial amount of bits. In this section, we show how symbolic memory states can be encoded by propositional variables and then how the model-checking algorithm can be expressed as a QBF formula where the quantification over symbolic memory states in MC (from Section 5.3) can be straighforwardly encoded by quantification in QBF. This allows us to reduce easily 1SL0 satisfiability to QBF satisfiability, and then to recall how QBF can be translated into a PSPACE fragment of first-order logic [Sto77]. In that way, instead of using an SMT solver as done in Section 5.2 or an ad-hoc algorithm MC based on symbolic memory states, it becomes possible to use either a QBF solver (see e.g. [LB10, HSB14]) or a theorem prover for first-order logic (see e.g. [WDF+09, KV13]) to solve the satisfiability/validity problem for 1SL0, as advocated in [Hag04, CGH05]. Nevertheless, it is fair to observe that the translation into QBF presented below takes advantage of the abstraction made in Section 5.3. It is worth mentioning that an alternative approach has been developped in [BF12] in which concepts from separation logic are directly imported into first-order logic in order to perform program verification.

In order to encode symbolic memory states over (q, α) of the form $(P, A, H, \mathfrak{n})$, we consider the following atomic propositions.

- $EQ(i, j)$ $(i, j \in [1, q])$ for encoding whether the program variables $\mathtt{x}_i$ and $\mathtt{x}_j$ belong to the same set $X \in P$.

- $A(i)$ $(i \in [1, q])$ for encoding whether there is $P \in A$ such that $\mathtt{x}_i \in P$.

- $H(i, j)$ $(i, j \in [1, q])$ for encoding whether there is a pair (X, X') such that $\mathtt{x}_i \in X$, $\mathtt{x}_j \in X'$ and (X, X') belongs to H.

- $N(\beta)$ $(\beta \in [0, \alpha])$ for encoding whether $\mathfrak{n} = \beta$. (Of course, a binary encoding is possible but not needed to illustrate the main ideas.)

We write $\mathtt{X}$ to denote the above set of atomic propositions (that is parameterised by q and α). So, each symbolic memory state comes with its set of atomic propositions. We write $\mathtt{X}'$ to denote the set of atomic propositions defined as $\mathtt{X}$ except that the atomic propositions are primed. The set $\mathtt{X}''$ is defined similarly as well as other variants that decorate differently the atomic propositions.

Given a propositional valuation v built over the sets $\mathtt{X}^1, \ldots, \mathtt{X}^n$, we write $\mathtt{Symb}[v]$ to denote the unique n-tuple of symbolic memory states, if it is defined, such that the ith element of the tuple is equal to the symbolic memory state corresponding to the interpretation of the atomic propositions from $\mathtt{X}^i$. Given a symbolic memory state $(P, A, H, \mathfrak{n})$, one can easily design a propositional valuation that encodes it by the truth value of the atomic propositions in $\mathtt{X}$. Similarly, existence of a symbolic memory state corresponding to a propositional valuation over $\mathtt{X}$ can be specified by a propositional formula of polynomial size.

Lemma 5.4.1. Let $\mathtt{X}$ be a set of atomic propositions possibly encoding a symbolic memory state. There is a propositional formula $SMS(\mathtt{X})$ built over $\mathtt{X}$ such that for all propositional valuations v, we have $v \models SMS(\mathtt{X})$ iff there is a symbolic memory state $\mathtt{sms}$ such that $\mathtt{Symb}[v] = \mathtt{sms}$.

The proof is left as Exercise 5.16. $SMS(\mathtt{X})$ is a conjunction of formulae expressing simple properties. By way of example, there is a conjunct that states that exactly one atomic proposition among $N(0), \ldots, N(\alpha)$ holds true.

Lemma 5.4.2. Let $\mathtt{X}$, $\mathtt{X}'$ and $\mathtt{X}''$ be three sets of atomic propositions possibly encoding symbolic memory states. There is a propositional formula $*_p(\mathtt{X}, \mathtt{X}', \mathtt{X}'')$ such that for all propositional valuations v, we have $v \models *_p(\mathtt{X}, \mathtt{X}', \mathtt{X}'')$ iff there are symbolic memory states $\mathtt{sms}$, $\mathtt{sms}'$ and $\mathtt{sms}''$ such that $*_s(\mathtt{sms}, \mathtt{sms}', \mathtt{sms}'')$ and $\mathtt{Symb}[v] = (\mathtt{sms}, \mathtt{sms}', \mathtt{sms}'')$.

Proof. (sketch) The formula $*_p(\mathtt{X}, \mathtt{X}', \mathtt{X}'')$ is defined as the conjunction of the propositional formulae below:

1. $\mathtt{X}$, $\mathtt{X}'$ and $\mathtt{X}''$ encode a symbolic memory state:

$$SMS(\mathtt{X}) \wedge SMS(\mathtt{X}') \wedge SMS(\mathtt{X}'').$$

2. Encoding of '$P = P_1 = P_2$':

$$\bigwedge_{i,j \in [1,q]} (EQ(i,j) \Leftrightarrow EQ'(i,j)) \wedge (EQ'(i,j) \Leftrightarrow EQ''(i,j)).$$

3. Encoding of '$A = A_1 \cup A_2$ and $A_1 \cap A_2 = \emptyset$':

$$\bigwedge_{i \in [1,q]} (A(i) \Leftrightarrow (A'(i) \vee A''(i))) \wedge \neg(A'(i) \wedge A''(i)).$$

4. Encoding of '$H = H_1 \cup H_2$':

$$\bigwedge_{i,j \in [1,q]} (H(i,j) \Leftrightarrow (H'(i,j) \vee H''(i,j))) \wedge \neg(H'(i,j) \wedge H''(i,j)).$$

5. Encoding of '$\mathfrak{n} = max(\alpha, \mathfrak{n}_1 + \mathfrak{n}_2)$':

$$\bigwedge_{\beta,\beta' \in [0,\alpha]} (N'(\beta) \wedge N''(\beta')) \Rightarrow N(min(\alpha, \beta + \beta')).$$

The translation below is a simplification of the translation from Section 1.3.3 by taking into account the structure of the symbolic memory states involved in the algorithm MC from Section 5.3. The map tr is homomorphic for Boolean connectives ($\mathtt{X}$, $\mathtt{X}'$ and $\mathtt{X}''$are sets of atomic propositions possibly encoding symbolic memory states).

$$\begin{array}{lcl}
tr(\mathtt{emp}, \mathtt{X}) & \stackrel{\text{def}}{=} & N(0) \wedge \neg A(1) \wedge \cdots \wedge \neg A(q) \\
tr(\mathtt{x}_i \hookrightarrow \mathtt{x}_j, \mathtt{X}) & \stackrel{\text{def}}{=} & H(i,j) \\
tr(\mathtt{x}_i = \mathtt{x}_j, \mathtt{X}) & \stackrel{\text{def}}{=} & EQ(i,j) \\
tr(\psi_1 * \psi_2, \mathtt{X}) & \stackrel{\text{def}}{=} & \exists\, \mathtt{X}',\ \mathtt{X}''\ *_p (\mathtt{X}, \mathtt{X}', \mathtt{X}'') \wedge tr(\psi_1, \mathtt{X}') \wedge tr(\psi_2, \mathtt{X}'') \\
tr(\psi_1 \mathrel{-\!\!*} \psi_2, \mathtt{X}) & \stackrel{\text{def}}{=} & \forall\, \mathtt{X}'',\ (\exists \mathtt{X}'\ *_p (\mathtt{X}'', \mathtt{X}, \mathtt{X}')) \Rightarrow \\
 & & (\exists \mathtt{X}'\ *_p (\mathtt{X}'', \mathtt{X}, \mathtt{X}') \wedge (tr(\psi_1, \mathtt{X}') \Rightarrow tr(\psi_2, \mathtt{X}'')).
\end{array}$$

All the quantifications involved in the translation introduce new atomic propositions and $\exists\, \{p_1, \ldots, p_m\}\ \psi$ is understood as a shortcut for $\exists\, p_1, \ldots, \exists\, p_m\ \psi$. Moreover, the atomic propositions $N(0)$, $A(1)$, $\ldots$, $A(q)$, $H(i, j)$ and $EQ(i, j)$ mentioned above are those from the set $\mathtt{X}$, which is one argument of the translation.

Lemma 5.4.3. Let φ be a formula built over the program variables $\mathtt{x}_1, \ldots, \mathtt{x}_q$ and $\alpha = \mathtt{msize}(\varphi)$. $\exists\, \mathtt{X}\ \mathit{SMS}(\mathtt{X}) \wedge \mathit{tr}(\varphi, \mathtt{X})$ is QBF satisfiable iff there is symbolic memory state $\mathtt{sms}$ over (q, α) such that MC($\mathtt{sms}$,φ) returns $\top$.

Again, the proof is omitted but one can establish a correspondence between the QBF quantifications and the quantifications in MC. As a corollary, φ in 1SL0 is satisfiable iff $\exists\, \mathtt{X}\ \mathit{SMS}(\mathtt{X}) \wedge \mathit{tr}(\varphi, \mathtt{X})$ is QBF satisfiable. Since the translation requires only logarithmic space, this provides a PSPACE upper bound for 1SL0. More importantly, we have presented a technique to decide separation logic via a QBF solver and therefore we can take advantage of all the breakthroughs made recently, see e.g. [LB10, HSB14].

Furthermore, we can also regain a translation into first-order logic restricted to the equality predicate as done in [Hag04, CGH05]. Actually, the previous translation composed with the final translation presented below leads again to a polynomial-space decision procedure. However, we believe that we have simplified the reduction while we have been able to provide an intermediate step in QBF that has not been explored and investigated so far.

Let $\chi = Q_1\ p_1\ \cdots\ Q_n\ p_n\ \varphi$ be a QBF formula with $\{Q_1, \ldots, Q_n\} \subseteq \{\exists, \forall\}$ and φ is a propositional formula built over the atomic propositions in $\{p_1, \ldots, p_n\}$

Let us consider the translated formula below and the translation tr' is homomorphic for Boolean connectives.

$$\exists\, \mathtt{x}_0,\ \mathtt{x}_1\ (\mathtt{x}_0 \neq \mathtt{x}_1) \wedge \mathit{tr}'(\chi)$$

$$\begin{array}{lcl}
\mathit{tr}'(\exists\, p\ \psi) & \stackrel{\text{def}}{=} & \exists\, \mathtt{x}_p\ (\mathtt{x}_p = \mathtt{x}_0 \vee \mathtt{x}_p = \mathtt{x}_1) \wedge \mathit{tr}'(\psi) \\
 & & \mathtt{x}_p \text{ is a fresh variable} \\
\mathit{tr}'(\forall\, p\ \psi) & \stackrel{\text{def}}{=} & \forall\, \mathtt{x}_p\ (\mathtt{x}_p = \mathtt{x}_0 \vee \mathtt{x}_p = \mathtt{x}_1) \Rightarrow \mathit{tr}'(\psi) \\
 & & \mathtt{x}_p \text{ is a fresh variable} \\
\mathit{tr}'(p) & \stackrel{\text{def}}{=} & (\mathtt{x}_p = \mathtt{x}_1).
\end{array}$$

It is easy to establish the result below.

Lemma 5.4.4. χ is QBF satisfiable iff $\exists\, \mathtt{x}_0,\ \mathtt{x}_1\ (\mathtt{x}_0 \neq \mathtt{x}_1) \wedge \mathit{tr}'(\chi)$ is satisfiable in first-order logic restricted to the equality predicate.

As promised, we have regained the existence of a simple translation from 1SL0 into a PSPACE fragment of first-order logic. Actually, this is essentially the translation designed in [CGH05] but presented in a different way and with an intermediate step in QBF (see also [Cha04, Chapter 4]). Alternatively, matching logic, that is a first-order dialect with patterns built-in, can easily encode separation logic, see the survey paper [Ros15].

Most probably, it would be possible to design a (labelled) sequent-style calculus for 1SL0 that mimicks a sequent-style calculus for QBF [Egl12], by taking into account the correspondence between propositional valuations and symbolic memory states. This may add some theoretical value but we doubt that in practice, this provides decision procedures more efficient than the ones obtained by translation into QBF and then use an QBF solver.

5.5 Bibliographical References about Proof Systems

In the previous sections, we have seen that for any $k \geq 1$, the set of valid formulae for kSL is not recursively enumerable and therefore there is no hope to design finite axiomatization for kSL and to design nice sequent-style proof systems. Nevertheless, calculi exist for abstract separation logics, mostly because first-order conditions are involved in separation models, see e.g. the conditions in [HCGT14]. Similarly, display calculi for bunched logics can be found in [Bro12]. The recent work [HGT15] also presents a sound (but necessarily incomplete) labelled sequent calculus for the logic 2SL that repairs the deficiencies of previous calculi.

Hilbert-style axiomatizations can be also found in [BV14] by using nominals but again this involves mainly abstract separation models and does not deal with concrete heaps as in kSL. Still, it is possible to design complete proof systems for propositional logics, such as the tableaux-style calculus for 2SL0 in [GM10] but completeness for full 2SL is not possible (see Theorem 1.3.5). The literature contains also a few attempts to design complete proof systems for some kSL. Graph-based decision procedures can be found in [HIOP13], which goes beyond 1SL0 (see also an NP-complete fragment of separation logic that can be decided using a model-theoretical decision procedure [ESS13]).

5.6 Exercises

Exercise 5.1. Let $\mathcal{A} = (\mathbb{N}, \mathfrak{h})$ be the structure such that $\mathfrak{h} : \mathbb{N} \to \mathbb{N}$, $\mathfrak{h}(0) = 1$ and for all $i \geq 1$, $\mathfrak{h}(i) = i$.

a) Check that $\mathcal{A}$ is a GRASS-model.

b) Given a variable assignment $\mathfrak{f}$, characterise the set $[\![\{\mathtt{x}.\mathtt{x} = \mathtt{x}\} \setminus S]\!]_{\mathcal{A},\mathfrak{f}}$ in terms of $[\![S]\!]_{\mathcal{A},\mathfrak{f}}$.

c) Given terms $T_1, \ldots, T_N$, design a term S such that for all $\mathcal{A}$ and $\mathfrak{f}$, we have

$$\{[\![T_1]\!]_{\mathcal{A},\mathfrak{f}}, \ldots, [\![T_N]\!]_{\mathcal{A},\mathfrak{f}}\} = [\![S]\!]_{\mathcal{A},\mathfrak{f}}.$$

Exercise 5.2. Let $\varphi_s = \varphi_s^1 * \varphi_s^2$ be a spatial formula in SLLB such that φ_s^1 is a non-empty separating conjunction of equalities or inequalities and φ_s^2 is a non-empty separating conjunction of reachability atomic formulae. Show that φ_s is satisfiable in SLLB iff $\varphi_s^1 \wedge \varphi_s^2$ is satisfiable in 1SL0 augmented with the strict reachability predicate $\mathtt{sreach}$.

Exercise 5.3. Prove Lemma 5.2.3.

Exercise 5.4. Show the statements in Lemma 5.2.4.

Exercise 5.5. Show Lemma 5.2.5.

Exercise 5.6. Construct the translation of the SLLB formula $\neg(\mathtt{x}_1 \mapsto \mathtt{x}_2 * \mathtt{x}_3 \mapsto \mathtt{x}_4)$ into GRASS (by eliminating also adequatly the existential quantifications over set variables).

Exercise 5.7. Prove Lemma 5.2.7.

Exercise 5.8. Show Lemma 5.3.1.

Exercise 5.9. Proof Lemma 5.3.2.

Exercise 5.10. Show Corollary 5.3.8.

Exercise 5.11. Compute the abstraction over (q, α) with $q = 4$ and $\alpha = 3$ for the three memory states presented in Figure 1.1.

Exercise 5.12. Given symbolic memory states $\mathtt{sms}$, $\mathtt{sms}_1$ and $\mathtt{sms}_2$, show that $*_s(\mathtt{sms}, \mathtt{sms}_1, \mathtt{sms}_2)$ iff the conditions below hold:

a) $P = P_1 = P_2$,

b) $A = A_1 \cup A_2$ and $A_1 \cap A_2 = \emptyset$,

c) $H = H_1 \cup H_2$,

d) $\mathfrak{n} = min(\alpha, \mathfrak{n}_1 + \mathfrak{n}_2)$.

Exercise 5.13. Let $\beta \geq 0$. Define a Boolean combination of test formulae from $\mathtt{Test}(q, \beta)$ that is logically equivalent to $\mathtt{size}_{\bar{q}} \geq \beta$.

Exercise 5.14. For every $k > 1$, show that the satisfiability and model-checking problems for kSL0 can be solved in PSPACE.

Exercise 5.15. Explain why the satisfiability and model-checking problems for 1SL0 remains in PSPACE even if the formulae are encoded as DAGs (instead of trees, as it is implicitly assumed all over the chapter).

Exercise 5.16. Prove Lemma 5.4.1. For instance, the formula $SMS(\mathtt{X})$ should enforce that 'EQ' is an equivalence relation, 'EQ' is a congruence for 'A' and 'H', etc.

Chapter 6

CONCLUSION

In this document, we have presented a puristic version of separation logics in which the set of locations (resp. values) is equal to the set of natural numbers and most of the time, the fragments satisfy simple syntactic closure properties. Our main object of study are fragments of the form $k\mathrm{SL}k'$, i.e. first-order separation logic with k record fields and at most k' quantified variables in formulae. Several proof techniques have been introduced to establish decidability, undecidability, computational complexity characterisation and expressiveness (see Figure 6.1 for a partial overview of results). Some of the results are quite standard and have been proved at the early age of separation logics (see e.g. [Yan01, COY01, Loz04a]) whereas we have also presented more recent results about expressiveness (see e.g. [BDL12, DD14]) and decision procedures, for instance those based on the SMT framework (see e.g. [PWZ13]) or QBF. Moreover, we have provided formal relationships with other classes of logics introduced with possibly different motivations such as, logic of bunched implications [OP99, Pym02], data logics [BMS+06, FS09], interval temporal logics [Mos83], modal logics [DD15b] or first-order logic [CGH05]. The material in the different chapters has been designed so that unified notations are used and pointers to the literature are provided as much as possible.

Nevertheless, even though the document deals with separation logics, it focuses on a selection of logical investigations and does not say much about other aspects of such logics; the format of an ESSLLI course justifies however the necessity of such a choice. References to analytic proof systems for (abstract or concrete) separation logics are provided in Section 5.5 but this topic would deserve more developments. Besides, the document provides motivations about separation logics related to formal verification by extending Floyd-Hoare logic (see

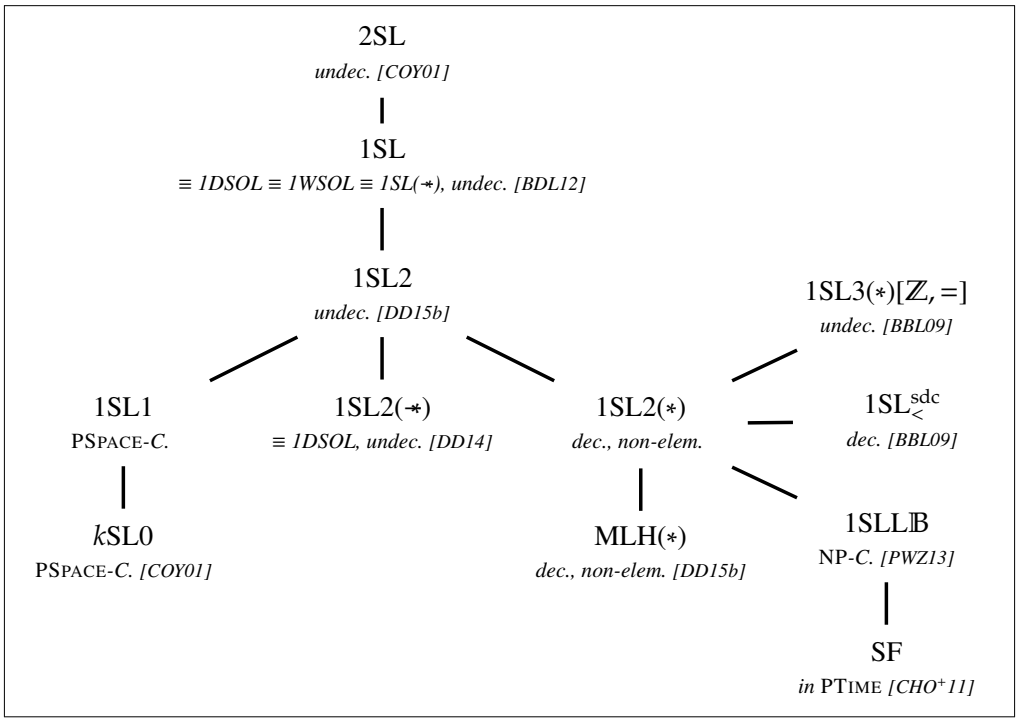


Figure 6.1: A few results about decidability and complexity

e.g. Section 1.1). Certainly, the document could be completed along these lines by proving program properties by using separation logics and its proof systems. We invite the reader to consult [O'H12, Gor14, Jen13a] to have first-class developments about formal verification with separation logics. Similarly, the emerging use of SMT solvers to decide separation logics (see e.g. Section 5.1.3) witnesses the power of such a framework (Section 5.2 is nevertheless related to it). Other aspects about program verification have been quickly presented in order to focus instead on fundamental properties of separation logics. However, in full generality and in order to illustrate the practical use of separation logics, more material about data values could be considered (see e.g. [BDES09, BBL09, MPQ11]) as well as about the use of inductive predicates such as those related to lists, trees, etc. (see e.g. [IRS13, BFGN14]). We also refer the reader to the slides by P. O'Hearn from the invited talk given at the "SIGPLAN Programming Languages Mentoring Workshop (PLMW)", Roma 2013, for current trends in the mechanisation of proofs for formal verification.

As conclusion, let us mention a few research directions related to logical investigations of separation logics. First, for the first time, SMT-COMP 2014 run a competition with SMT solvers for separation logic as an "off" event, see e.g. [SC14]. A promising direction consists in developing further SMT-based decision procedures for separation logics allowing even more combinations with other logical theories. Besides, there is still some need and interest to design even more tractable fragments useful for formal verification. As far as we know, it is open whether 1SL0 + `sreach` is decidable (both separating connectives $*$ and $-\!\!*$ belong to such a fragment), see e.g. its use in [TBR14]. Finally, designing proof systems for separation logics from which decision procedures can be designed (when possible) remains quite open. For instance, as far as we know, no label-free sequent-style calculus exists for 1SL0 that can lead to a decision procedure that runs in polynomial space. These are only a few possible directions.

Bibliography

[ABdCH09] G. Aucher, Ph. Balbiani, L. Fariñas del Cerro, and A. Herzig. Global and local graph modifiers. *Electronic Notes in Theoretical Computer Science*, 231:293–307, 2009. (Cited on page 40)

[ABM01] C. Areces, P. Blackburn, and M. Marx. Hybrid logics: characterization, interpolation and complexity. *The Journal of Symbolic Logic*, 66(3):977–1010, 2001. (Cited on page 115)

[AD94] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994. (Cited on page 66)

[AD09] T. Antonopoulos and A. Dawar. Separating graph logic from MSO. In *FOSSACS'09*, volume 5504 of *Lecture Notes in Computer Science*, pages 63–77. Springer, 2009. (Cited on pages 110, 134)

[AFH14] C. Areces, R. Fervari, and G. Hoffmann. Swap logic. *Logic Journal of the IGPL*, 22(2):309–332, 2014. (Cited on page 40)

[AGH+14] T. Antonopoulos, N. Gorogiannis, C. Haase, M. Kanovich, and J. Ouaknine. Foundations for decision problems in separation logic with general inductive predicates. In *FOSSACS'14*, volume 8412 of *Lecture Notes in Computer Science*, pages 411–425. Springer, 2014. (Cited on pages 11, 38)

[All83] J. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983. (Cited on page 119)

[Ant10] T. Antonopoulos. *Expressive Power of Query Languages*. PhD thesis, University of Cambridge, 2010. (Cited on pages 134, 135)

[Apt81] K. Apt. Ten Years of Hoare's Logic. *ACM Transactions on Programming Languages and Systems*, 3(4):431–483, 1981. (Cited on pages 10, 15, 17)

[AV11] S. Abramsky and J. Väänänen. From IF to BI: a tale of dependence and separation. *CoRR*, abs/1102.1388, 2011. (Cited on page 10)

[BBC+13] S. Benaim, M. Benedikt, W. Charatonik, E. Kieroński, R. Lenhardt, F. Mazowiecki, and J. Worrell. Complexity of two-variable logic over finite trees. In *ICALP'13*, volume 7966 of *Lecture Notes in Computer Science*, pages 74–88. Springer, 2013. (Cited on page 28)

[BBF+01] B. Bérard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, and Ph. Schnoebelen. *Systems and Software Verification, Model-Checking Techniques and Tools*. Springer, 2001. (Cited on page 9)

[BBL09] K. Bansal, R. Brochenin, and E. Lozes. Beyond shapes: Lists with ordered data. In *FOSSACS'09*, volume 5504 of *Lecture Notes in Computer Science*, pages 425–439. Springer, 2009. (Cited on pages 12, 70, 112, 113, 114, 116, 170, 171)

[BCM+03] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2003. (Cited on page 9)

[BCO04] J. Berdine, C. Calcagno, and P. O'Hearn. A decidable fragment of separation logic. In *FSTTCS'04*, volume 3328 of *Lecture Notes in Computer Science*, pages 97–109. Springer, 2004. (Cited on pages 53, 54)

[BCO05] J. Berdine, C. Calcagno, and P. O'Hearn. Smallfoot: Modular automatic assertion checking with separation logic. In *FMCO'05*, volume 4111 of *Lecture Notes in Computer Science*, pages 115–137. Springer, 2005. (Cited on pages 10, 43, 53)

[BCOP05] R. Bornat, C. Calcagno, P. O'Hearn, and M. Parkinson. Permission accounting in separation logic. In *POPL'05*, pages 259–270. ACM, 2005. (Cited on page 55)

[BDES09] A. Bouajjani, C. Dragoi, C. Enea, and M. Sighireanu. A logic-based framework for reasoning about composite data structures. In *CONCUR'09*, volume 5710 of *Lecture Notes in Computer Science*, pages 178–195, 2009. (Cited on pages 12, 112, 114, 171)

[BdFV11] M. Benevides, R. de Freitas, and J. Viana. Propositional dynamic logic with storing, recovering and parallel composition. *Electronic Notes in Theoretical Computer Science*, 269:95–107, 2011. (Cited on page 10)

[BDG+10] D. Bresolin, D. Della Monica, V. Goranko, A. Montanari, and G. Sciavicco. Metric propositional neighborhood logics: Expressiveness, decidability, and undecidability. In *ECAI'10*, volume 215 of *Frontiers in Artificial Intelligence and Applications*, pages 695–700, 2010. (Cited on page 110)

[BDL08] R. Brochenin, S. Demri, and E. Lozes. On the almighty wand. In *CSL'08*, volume 5213 of *Lecture Notes in Computer Science*, pages 322–337. Springer, 2008. (Cited on page 134)

[BDL09] R. Brochenin, S. Demri, and E. Lozes. Reasoning about sequences of memory states. *Annals of Pure and Applied Logic*, 161(3):305–323, 2009. (Cited on pages 12, 52, 53, 150)

[BDL12] R. Brochenin, S. Demri, and E. Lozes. On the almighty wand. *Information and Computation*, 211:106–137, 2012. (Cited on pages 11, 27, 36, 37, 66, 70, 72, 73, 84, 85, 86, 105, 110, 127, 131, 134, 169, 170)

[BDM+11] M. Bojańczyk, C. David, A. Muscholl, Th. Schwentick, and L. Segoufin. Two-variable logic on data words. *ACM Transactions on Computational Logic*, 12(4):27, 2011. (Cited on pages 66, 117, 118)

[BdRV01] P. Blackburn, M. de Rijke, and Y. Venema. *Modal Logic*. Cambridge University Press, 2001. (Cited on pages 34, 128, 129)

[BF12] F. Bobot and J.-C. Filliâtre. Separation predicates: A taste of separation logic in first-order logic. In *Formal Methods and Software Engineering - 14th International Conference on Formal Engineering Methods, ICFEM 2012*, pages 167–181, 2012. (Cited on page 162)

[BFGN14] J. Brotherston, C. Fuhs, N. Gorogiannis, and J. Navarro Perez. A decision procedure for satisfiability in separation logic with inductive predicates. In *CSL-LICS'14*, 2014. (Cited on pages 11, 12, 38, 141, 171)

[BGG97] E. Börger, E. Grädel, and Y. Gurevich. *The Classical Decision Problem*. Perspectives in Mathematical Logic. Springer, 1997. (Cited on pages 36, 38, 133, 134)

[BHJS07] A. Bouajjani, P. Habermehl, Y. Jurski, and M. Sighireanu. Rewriting systems with data. In *FCT'07*, volume 4639 of *Lecture Notes in Computer Science*, pages 1–22. Springer, 2007. (Cited on page 112)

[BIP10] M. Bozga, R. Iosif, and S. Perarnau. Quantitative separation logic and programs with lists. *Journal of Automated Reasoning*, 45(2):131–156, 2010. (Cited on page 73)

[BK10] J. Brotherston and M. Kanovich. Undecidability of propositional separation logic and its neighbours. In *LICS'10*, pages 130–139. IEEE, 2010. (Cited on pages 12, 44, 55, 56, 57)

[BK14] J. Brotherston and M. Kanovich. Undecidability of propositional separation logic and its neighbours. *Journal of the Association for Computing Machinery*, 61(2), 2014. (Cited on pages 33, 44, 55)

[BL10] M. Bojańczyk and S. Lasota. An extension of data automata that captures XPath. In *LICS'10*, pages 243–252. IEEE, 2010. (Cited on page 66)

[BMG+14] D. Bresolin, D. Della Monica, V. Goranko, A. Montanari, and G. Sciavicco. The dark side of interval temporal logic: marking the undecidability border. *Annals of Mathematics and Artificial Intelligence*, 71(1-3):41–83, 2014. (Cited on page 119)

[BMS+06] M. Bojańczyk, A. Muscholl, Th. Schwentick, L. Segoufin, and C. David. Two-variable logic on words with data. In *LICS'06*, pages 7–16. IEEE, 2006. (Cited on pages 112, 113, 169)

[Bou02] P. Bouyer. A logical characterization of data languages. *Information Processing Letters*, 84(2):75–85, 2002. (Cited on pages 66, 67)

[BPS09] M. Botincan, M. Parkinson, and W. Schulte. Separation logic verification of C programs with an SMT solver. *Electronic Notes in Theoretical Computer Science*, 254:5–23, 2009. (Cited on page 139)

[BRK+15] K. Bansal, A. Reynolds, T. King, C. Barrett, and Th. Wies. Deciding local theory extensions via E-matching. In *CAV'15*, volume 9207 of *Lecture Notes in Computer Science*, pages 87–105. Springer, 2015. (Cited on pages 137, 140, 141, 145)

[Bro12] J. Brotherston. Bunched logics displayed. *Studia Logica*, 100(6):1223–1254, 2012. (Cited on pages 33, 166)

[Bro13] R. Brochenin. *Separation Logic: Expressiveness, Complexity, Temporal Extension*. PhD thesis, LSV, ENS Cachan, Sep. 2013. (Cited on page 112)

[BT14a] Ph. Balbiani and T. Tinchev. Decidability and computability in PRSPDL. In *AIML'14*, pages 16–33. College Publications, 2014. (Cited on page 10)

[BT14b] C. Barrett and C. Tinelli. Satisfiability Modulo Theories. In *Handbook of Model Checking*. Springer, 2014. In preparation. (Cited on pages 140, 141, 142)

[Bur72] R. Burstall. Some techniques for proving correctness of programs which alter data structures. *Machine Intelligence*, 7:23–50, 1972. (Cited on page 20)

[BV14] J. Brotherston and J. Villard. Parametric completeness for separation theories. In *POPL'14*, pages 453–464. ACM, 2014. (Cited on pages 12, 33, 34, 55, 166)

[BvBW06] P. Blackburn, J.F. van Benthem, and F. Wolter, editors. *Handbook of modal logic (Vol. 3)*. Elsevier, 2006. (Cited on page 9)

[CD11] C. Calcagno and D. 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. (Cited on page 11)

[CDD+15] C. Calcagno, D. Distefano, J. Dubreil, D. Gabi, P. Hooimeijer, M. Luca, P. O'Hearn, I. Papakonstantinou, J. Purbrick, and D. Rodriguez. Moving fast with software verification. In *NASA Formal Methods - 7th International Symposium, NFM*, volume 9058 of *Lecture Notes in Computer Science*, pages 3–11. Springer, 2015. (Cited on pages 10, 11)

[CG13] J.-R. Courtault and D. Galmiche. A modal BI logic for dynamic resource properties. In *LFCS'13*, volume 7734 of *Lecture Notes in Computer Science*, pages 134–148. Springer, 2013. (Cited on pages 12, 128)

[CGH05] C. Calcagno, Ph. Gardner, and M. Hague. From separation logic to first-order logic. In *FOSSACS'05*, volume 3441 of *Lecture Notes in Computer Science*, pages 395–409. Springer, 2005. (Cited on pages 11, 52, 139, 141, 161, 162, 165, 166, 169)

[CGP00] E. Clarke, O. Grumberg, and D. Peled. *Model checking*. The MIT Press Books, 2000. (Cited on page 9)

[Cha04] A. Chawdhary. Translating separation logic to first order logic. Master's thesis, Imperial College, 2004. (Cited on page 166)

[CHO+11] B. Cook, C. Haase, J. Ouaknine, M. Parkinson, and J. Worrell. Tractable reasoning in a fragment of separation logic. In *CONCUR'11*, volume 6901 of *Lecture Notes in Computer Science*, pages 235–249. Springer, 2011. (Cited on pages 11, 43, 53, 54, 139, 141, 170)

[CKM14] W. Charatonik, E. Kieroński, and F. Mazowiecki. Decidability of Weak Logics with Deterministic Transitive Closure. In *CSL-LICS'14*. ACM, 2014. (Cited on page 35)

[Coo72] D. Cooper. Theorem proving in arithmetic without multiplication. *Machine Learning*, 7:91–99, 1972. (Cited on page 155)

[Coo78] S. Cook. Soundness and completeness of an axiom system for program verification. *SIAM Journal of Computing*, 7(1), 1978. (Cited on pages 17, 18)

[Cop02] J. Copeland. The genesis of possible worlds semantics. *Journal of Philosophical Logic*, 31(1):99–137, 2002. (Cited on page 10)

[COY01] C. Calcagno, P. O'Hearn, and H. Yang. Computability and complexity results for a spatial assertion language for data structures. In *FSTTCS'01*, volume 2245 of *Lecture Notes in Computer Science*, pages 108–119. Springer, 2001. (Cited on pages 11, 14, 38, 39, 43, 44, 47, 48, 50, 52, 53, 62, 110, 138, 150, 158, 160, 169, 170)

[COY07] C. Calcagno, P. O'Hearn, and H. Yang. Local action and abstract separation logic. In *LICS'07*, pages 366–378. IEEE, 2007. (Cited on pages 12, 55)

[Dav09] C. David. *Analyse de XML avec données non-bornées*. PhD thesis, LIAFA, Université Paris VII, 2009. (Cited on pages 66, 118)

[DD07] S. Demri and D. D'Souza. An automata-theoretic approach to constraint LTL. *Information and Computation*, 205(3):380–415, 2007. (Cited on pages 112, 113)

[DD14] S. Demri and M. Deters. Expressive completeness of separation logic with two variables and no separating conjunction. In *CSL-LICS'14*. ACM, 2014. Extended version currently submitted to a journal. (Cited on pages 3, 37, 66, 73, 76, 84, 85, 107, 110, 169, 170)

[DD15a] S. Demri and M. Deters. Separation logics and modalities: A survey. *Journal of Applied Non-Classical Logics*, 25(1):50–99, 2015. (Cited on pages 3, 44)

[DD15b] S. Demri and M. Deters. Two-variable separation logic and its inner circle. *ACM Transactions on Computational Logic*, 2(16), 2015. (Cited on pages 3, 44, 66, 76, 79, 84, 110, 112, 127, 169, 170)

[Dem03] S. Demri. A polynomial space construction of tree-like models for logics with local chains of modal connectives. *Theoretical Computer Science*, 300(1–3):235–258, 2003. (Cited on pages 52, 158)

[Dem05] S. Demri. A reduction from DLP to PDL. *Journal of Logic and Computation*, 15(5):767–785, 2005. (Cited on page 40)

[DFL02] A. Degtyarev, M. Fisher, and A. Lisitsa. Equality and monodic first-order temporal logic. *Studia Logica*, 72(2):147–156, 2002. (Cited on page 131)

[DGG04] A. Dawar, Ph. Gardner, and G. Ghelli. Adjunct elimination through games in static ambient logic. In *FST & TCS'04*, volume 3328 of *Lecture Notes in Computer Science*, pages 211–223. Springer, 2004. (Cited on page 134)

[DGG07] A. Dawar, Ph. Gardner, and G. Ghelli. Expressiveness and complexity of graph logic. *Information and Computation*, 205(3):263–310, 2007. (Cited on pages 10, 27, 134)

[DGLWM14] S. Demri, D. Galmiche, D. Larchey-Wendling, and D. Mery. Separation logic with one quantified variable. In *CSR'14*, volume 8476 of *Lecture Notes in Computer Science*, pages 125–138. Springer, 2014. Extended version currently submitted to a journal. (Cited on pages 52, 53, 138, 150)

[DHLT14] N. Decker, P. Habermehl, M. Leucker, and D. Thoma. Ordered navigation on multi-attributed data words. In *CONCUR'14*, volume 8704 of *Lecture Notes in Computer Science*, pages 497–511, 2014. (Cited on page 66)

[Dij76] E.W. Dijkstra. *A Discipline of Programming*. Prentice-Hall, 1976. (Cited on page 17)

[DL09] S. Demri and R. Lazić. LTL with the freeze quantifier and register automata. *ACM Transactions on Computational Logic*, 10(3), 2009. (Cited on pages 114, 115)

[dMB08] L. de Moura and N. Björner. Z3: An Efficient SMT Solver. In *TACAS'08*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer, 2008. (Cited on pages 114, 142)

[dNSH00] H. de Nivelle, R. Schmidt, and U. Hustadt. Resolution-based methods for modal logics. *Logic Journal of the IGPL*, 8(3):265–292, 2000. (Cited on page 139)

[DOY06] D. Distefano, P. O'Hearn, and H. Yang. A local shape analysis based on separation logic. In *TACAS'06*, volume 3920 of *Lecture Notes in Computer Science*, pages 287–302. Springer, 2006. (Cited on page 11)

[dR92] M. de Rijke. The modal logic of inequality. *The Journal of Symbolic Logic*, 57(2):566–584, 1992. (Cited on page 129)

[Egl12] U. Egly. On sequent systems and resolution for QBFs. In *SAT'12*, volume 7317 of *Lecture Notes in Computer Science*, pages 100–113. Springer, 2012. (Cited on page 166)

[ESS13] C. Enea, V. Saveluc, and M. Sighireanu. Compositional invariant checking for overlaid and nested linked lists. In *ESOP'13*, volume 7792 of *Lecture Notes in Computer Science*, pages 129–148. Springer, 2013. (Cited on pages 141, 166)

[EVW97] K. Etessami, M. Vardi, and Th. Wilke. First-order logic with two variables and unary temporal logics. In *LICS'97*, pages 228–235. IEEE, 1997. (Cited on pages 70, 106, 109, 117)

[Fig10] D. Figueira. *Reasoning on words and trees with data*. PhD thesis, ENS Cachan, December 2010. (Cited on page 66)

[Fin75] K. Fine. Some connections between elementary and modal logic. In *3rd Scandinavian Logic Symposium*, pages 15–31. North Holland, 1975. (Cited on page 139)

[Fit83] M. Fitting. *Proof methods for modal and intuitionistic logics*. D. Reidel Publishing Co., 1983. (Cited on page 139)

[Flo67] R. Floyd. Assigning meanings to programs. *Proceedings of Symposia in Applied Mathematics*, 19:19–32, 1967. (Cited on page 15)

[FS09] D. Figueira and L. Segoufin. Future-looking logics on data words and trees. In *MFCS'09*, volume 5734 of *Lecture Notes in Computer Science*, pages 331–343. Springer, 2009. (Cited on pages 112, 114, 115, 169)

[Gab81] D. Gabbay. Expressive functional completeness in tense logic. In *Aspects of Philosophical Logic*, pages 91–117. Reidel, 1981. (Cited on page 115)

[GHR94] D. Gabbay, I. Hodkinson, and M. Reynolds. *Temporal Logic: Mathematical Foundations and Computational Aspects*, volume 1. Oxford University Press, 1994. (Cited on page 109)

[GHSS14] O. Gasquet, A. Herzig, B. Said, and F. Schwarzentruber. *Kripke's Worlds - An Introduction to Modal Logics via Tableaux*. Studies in Universal Logic. Birkhäuser, 2014. (Cited on page 139)

[Gir87] J.Y. Girard. Linear logic. *Theoretical Computer Science*, pages 1–101, 1987. (Cited on page 32)

[GKO11] N. Gorogiannis, M. Kanovich, and P. O'Hearn. The complexity of abduction for separated heap abstractions. In *SAS'11*, volume 6887 of *Lecture Notes in Computer Science*, pages 25–42. Springer, 2011. (Cited on pages 32, 54)

[GKV97] E. Grädel, Ph. Kolaitis, and M. Vardi. On the decision problem for two-variable first-order logic. *Bulletin of Symbolic Logic*, 3(1):53–69, 1997. (Cited on page 35)

[GLW06] D. Galmiche and D. Larchey-Wending. Expressivity properties of boolean BI through relational models. In *FST&TCS'06*, volume 4337 of *Lecture Notes in Computer Science*, pages 358–369. Springer, 2006. (Cited on pages 32, 33, 34)

[GM10] D. Galmiche and D. Méry. Tableaux and resource graphs for separation logic. *Journal of Logic and Computation*, 20(1):189–231, 2010. (Cited on pages 39, 137, 139, 166)

[GMP02] D. Galmiche, D. Mery, and D. Pym. Resource tableaux (extended abstract). In *CSL'02*, volume 2471 of *Lecture Notes in Computer Science*, pages 183–199. Springer, 2002. (Cited on page 33)

[Göl07] S. Göller. On the complexity of reasoning about dynamic policies. In *CSL'07*, volume 4646 of *Lecture Notes in Computer Science*, pages 358–373. Springer, 2007. (Cited on page 40)

[GOR99] E. Grädel, M. Otto, and E. Rosen. Undecidability results on two-variable logics. *Archive for Mathematical Logic*, 38(4–5):313–354, 1999. (Cited on pages 35, 110)

[Gor14] M. Gordon. Hoare Logic. Lecture notes, February 2014. Available at `http://www.cl.cam.ac.uk/~mjcg/HoareLogic/`. (Cited on pages 12, 15, 171)

[GP92] V. Goranko and S. Passy. Using the universal modality: gains and questions. *Journal of Logic and Computation*, 2(1):5–30, 1992. (Cited on page 60)

[Hag04] M. Hague. Static checkers for tree structures and heaps. Master's thesis, Imperial College, 2004. (Cited on pages 139, 141, 161, 162, 165)

[HC68] G. Hughes and M. Cresswell. *An introduction to modal logic*. Methuen and Co., 1968. (Cited on page 9)

[HCGT14] Z. Hou, R. Clouston, R. Goré, and A. Tiu. Proof search for propositional abstract separation logics via labelled sequents. In *POPL'14*, pages 465–476. ACM, 2014. (Cited on pages 21, 39, 55, 137, 139, 166)

[Hem96] E. Hemaspaandra. The price of universality. *Notre Dame Journal of Formal Logic*, 37(2):173–203, 1996. (Cited on page 60)

[HGT15] Z. Hou, R. Goré, and A. Tiu. Automated theorem proving for assertions in separation logic with all connectives. In *CADE'15*, volume 9195 of *Lecture Notes in Computer Science*, pages 501–516. Springer, 2015. (Cited on pages 39, 166)

[HIOP13] C. Haase, S. Ishtiaq, J. Ouaknine, and M. Parkinson. SeLoger: A tool for graph-based reasoning in separation logic. In *CAV'13*, volume 8044 of *Lecture Notes in Computer Science*, pages 790–795. Springer, 2013. (Cited on pages 53, 54, 137, 139, 141, 166)

[HKT00] D. Harel, D. Kozen, and J. Tiuryn. *Dynamic Logic*. MIT Press, 2000. (Cited on page 129)

[HLSV14] L. Hella, K. Luosto, K. Sano, and J. Virtema. The expressive power of modal dependence logic. In *AIML'14*, pages 294–312. College Publications, 2014. (Cited on page 10)

[HM80] M. Hennessy and R. Milner. On observing nondeterminism and concurrency. In *ICALP'80*, volume 85 of *Lecture Notes in Computer Science*, pages 299–309. Springer, 1980. (Cited on page 128)

[Hoa69] C.A.R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, 1969. (Cited on pages 10, 15)

[HR05] I. Hodkinson and M. Reynolds. Separation – past, present, and future. In S. Artemov, H. Barringer, A. d'Avila Garcez, L. Lamb, and J. Woods, editors, *We will show them! (Essays in honour of Dov Gabbay on his 60th birthday)*, volume 2, pages 117–142. College Publications, 2005. (Cited on page 66)

[HSB14] M. Heule, M. Seidl, and A. Biere. A unified proof system for QBF preprocessing. In *IJCAR'14*, volume 8562 of *Lecture Notes in Computer Science*, pages 91–106. Springer, 2014. (Cited on pages 140, 162, 165)

[HTG13] Z. Hou, A. Tiu, and R. Goré. A labelled sequent calculus for BBI: proof theory and proof search. In *TABLEAUX'13*, volume 8123 of *Lecture Notes in Computer Science*, pages 172–187. Springer, 2013. (Cited on page 33)

[IBI+13] Sh. Itzhaky, A. Banerjee, N. Immerman, A. Nanevski, and M. Sagiv. Effectively-propositional reasoning about reachability in linked data structures. In *CAV'13*, volume 8044 of *Lecture Notes in Computer Science*, pages 756–772. Springer, 2013. (Cited on page 140)

[IO01] S. Ishtiaq and P. O'Hearn. BI as an assertion language for mutable data structures. In *POPL'01*, pages 14–26. ACM, 2001. (Cited on pages 10, 14, 20, 21, 23, 29, 32)

[IRR+04] N. Immerman, A. Rabinovich, Th. Reps, M. Sagiv, and G. Yorsh. The boundary between decidability and undecidability for transitive-closure logics. In *CSL'04*, volume 3210 of *Lecture Notes in Computer Science*, pages 160–174. Springer, 2004. (Cited on page 110)

[IRS13] R. Iosif, A. Rogalewicz, and J. Simacek. The tree width of separation logic with recursive definitions. In *CADE'13*, volume 7898 of *Lecture Notes in Computer Science*, pages 21–38. Springer, 2013. (Cited on pages 12, 38, 171)

[IRV14] R. Iosif, A. Rogalewicz, and Th. Vojnar. Deciding entailments in inductive separation logic with tree automata. ArXiv, 2014. (Cited on page 141)

[Jen13a] J. Jensen. *Enabling Concise and Modular Specifications in Separation Logic*. PhD thesis, IT University of Copenhagen, 2013. (Cited on pages 12, 171)

[Jen13b] J. Jensen. Techniques for model construction in separation logic, September 2013. (Cited on page 12)

[JW96] D. Janin and I. Walukiewicz. On the expressive completeness of the propositional mu-calculus with respect to monadic second order logic. In *CONCUR'96*, volume 1119 of *Lecture Notes in Computer Science*, pages 263–277, 1996. (Cited on page 109)

[Kam68] J. Kamp. *Tense Logic and the Theory of Linear Order*. PhD thesis, UCLA, USA, 1968. (Cited on pages 66, 109)

[KF94] M. Kaminski and N. Francez. Finite-memory automata. *Theoretical Computer Science*, 134(2):329–363, 1994. (Cited on page 66)

[KMSV14] J. Kontinen, J.-S. Müller, H. Schnoor, and H. Vollmer. Modal independence logic. In *AIML'14*, pages 353–372. College Publications, 2014. (Cited on page 10)

[KR04] V. Kuncak and M. Rinard. On spatial conjunction as second-order logic. Technical Report MIT–CSAIL–TR–2004–067, MIT CSAIL, October 2004. (Cited on pages 11, 110)

[Kri59] S. Kripke. A completeness theorem in modal logic. *The Journal of Symbolic Logic*, 24(1):1–14, 1959. (Cited on page 10)

[KV13] L. Kovács and A. Voronkov. First-order theorem proving and Vampire. In *CAV'13*, volume 8044 of *Lecture Notes in Computer Science*, pages 1–35. Springer, 2013. (Cited on page 162)

[Lad77] R. Ladner. The computational complexity of provability in systems of modal propositional logic. *SIAM Journal of Computing*, 6(3):467–480, 1977. (Cited on pages 52, 158)

[Lan07] M. Lange. Linear time logics around PSL: Complexity, expressiveness, and a little bit of succinctness. In *CONCUR'07*, volume 4703 of *Lecture Notes in Computer Science*, pages 90–104. Springer, 2007. (Cited on page 109)

[LB10] F. Lonsing and A. Biere. DepQBF: A dependency-aware QBF solver. *Journal on Satisfiability, Boolean Modeling and Computation*, 7(2–3):71–76, 2010. (Cited on pages 140, 162, 165)

[Lew18] C. I. Lewis. *A survey of symbolic logic*. University of California Press, Berkeley, 1918. (Cited on page 9)

[LG13] D. Larchey-Wendling and D. Galmiche. Nondeterministic phase semantics and the undecidability of boolean BI. *ACM Transactions on Computational Logic*, 14(1), 2013. (Cited on pages 11, 33, 44)

[LG14] D. Larchey-Wendling and D. Galmiche. Looking at separation algebras with boolean BI-eyes. In *Theoretical Computer Science - 8th IFIP TC 1/WG 2.2 International Conference, TCS 2014*, volume 8705 of *Lecture*

Notes in Computer Science, pages 326–340. Springer, 2014. (Cited on page 34)

[Lib04] L. Libkin. *Elements of Finite Model Theory*. Springer, 2004. (Cited on page 134)

[Loz04a] E. Lozes. *Expressivité des Logiques Spatiales*. Phd thesis, ENS Lyon, 2004. (Cited on pages 10, 11, 51, 52, 53, 138, 150, 169)

[Loz04b] E. Lozes. Separation logic preserves the expressive power of classical logic. In *SPACE'04*, 2004. (Cited on pages 11, 149)

[LP14] W. Lee and S. Park. A proof system for separation logic with magic wand. In *POPL'14*, pages 477–490. ACM, 2014. (Cited on pages 21, 39)

[LQ08] S. Lahiri and S. Qadeer. Back to the future: revisiting precise program verification using SMT solvers. In *POPL'08*, pages 171–182. ACM, 2008. (Cited on pages 35, 139, 142)

[LR03] Ch. Löding and Ph. Rohde. Model checking and satisfiability for sabotage modal logic. In *FSTTCS'03*, volume 2914 of *Lecture Notes in Computer Science*, pages 302–313. Springer, 2003. (Cited on page 40)

[LSW01] C. Lutz, U. Sattler, and F. Wolter. Modal logic and the two-variable fragment. In *CSL'01*, volume 2142 of *Lecture Notes in Computer Science*, pages 247–261. Springer, 2001. (Cited on pages 107, 109)

[Lut06] C. Lutz. Complexity and succinctness of public announcement logic. In *AAMAS'06*, pages 137–143. ACM, 2006. (Cited on page 40)

[LWG10] D. Larchey-Wendling and D. Galmiche. The undecidability of Boolean BI through phase semantics. In *LICS'10*, pages 140–149. IEEE, 2010. (Cited on pages 12, 44, 55, 57)

[Mar06] J. Marcinkowski. On the expressive power of graph logic. In *CSL'06*, volume 4207 of *Lecture Notes in Computer Science*, pages 486–500. Springer, 2006. (Cited on pages 110, 134)

[MdR05] M. Marx and M. de Rijke. Semantic characterizations of navigational XPath. *SIGMOD Record*, 34(2):41–46, 2005. (Cited on page 107)

[Mer09] S. Mera. *Modal Memory Logics*. PhD thesis, LORIA & U. of Buenos Aires, 2009. (Cited on page 40)

[MIG14] E. Maclean, A. Ireland, and G. Grov. Proof automation for functional correctness in separation logic. *Journal of Logic and Computation*, 2014. to appear. (Cited on page 11)

[Min67] M. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, 1967. (Cited on page 58)

[Moo77] R. C. Moore. Reasoning about knowledge and action. In *IJCAI-5*, pages 223–227, 1977. (Cited on page 139)

[Mor76] Ch. Morgan. Methods for automated theorem proving in non classical logics. *IEEE Transactions on Computers*, 25(8):852–862, 1976. (Cited on pages 129, 139)

[Mos83] B. Moszkowski. Reasoning about digital circuits. Technical Report STAN-CS-83-970, Dept. of Computer Science, Stanford University, Stanford, CA, 1983. (Cited on pages 10, 119, 121, 127, 169)

[Mos04] B. 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. (Cited on pages 121, 127)

[MPQ11] P. Madhusudan, G. Parlato, and X. Qiu. Decidable logics combining heap structures and data. In *POPL'11*, pages 611–622. ACM, 2011. (Cited on pages 12, 112, 114, 171)

[MV97] M. Marx and Y. Venema. *Multi-Dimensional Modal Logic*. Applied Logic. Kluwer, 1997. (Cited on page 55)

[NO79] G. Nelson and D. C. Oppen. Simplification by cooperating decision procedures. *ACM Trans. Program. Lang. Syst.*, 1(2):245–257, 1979. (Cited on pages 141, 144)

[NSV04] F. Neven, T. Schwentick, and V. Vianu. Finite state machines for strings over infinite alphabets. *ACM Transactions on Computational Logic*, 5(3):403–435, 2004. (Cited on page 66)

[OG76] S. Owicki and D. Gries. An axiomatic proof technique for parall programs I. *Acta Informatica*, 6:319–340, 1976. (Cited on page 15)

[O'H12] P. O'Hearn. A primer on separation logic. In *Software Safety and Security: Tools for Analysis and Verification*, volume 33 of *NATO Science for Peace and Security Series*, pages 286–318, 2012. (Cited on pages 12, 171)

[ONdRG01] H.J. Ohlbach, A. Nonnengart, M. de Rijke, and D. Gabbay. Encoding two-valued non-classical logics in classical logic. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, pages 1403–1486. Elsevier Science Publishers B.V., 2001. (Cited on page 139)

[OP99] P. O'Hearn and D. Pym. The logic of bunched implications. *Bulletin of Symbolic Logic*, 5(2):215–244, 1999. (Cited on pages 11, 20, 32, 169)

[OYR04] P. O'Hearn, H. Yang, and J. Reynolds. Separation and information hiding. In *POPL'04*, pages 268–280. ACM, 2004. (Cited on page 32)

[Pnu77] A. Pnueli. The temporal logic of programs. In *FOCS'77*, pages 46–57. IEEE, 1977. (Cited on page 9)

[PR13] J. Navarro Pérez and A. Rybalchenko. Separation Logic Modulo Theories. In *APLAS'13*, volume 8301 of *Lecture Notes in Computer Science*, pages 90–106, 2013. (Cited on pages 137, 139, 140, 141)

[Pre29] M. Presburger. Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. In *Comptes Rendus du premier congrès de mathématiciens des Pays Slaves, Warszawa*, pages 92–101, 1929. (Cited on page 155)

[Pri67] A. Prior. *Past, Present and Future*. Oxford University Press, 1967. (Cited on page 129)

[PSP13] J. Park, J. Seo, and S. Park. A theorem prover for Boolean BI. In *POPL'13*, pages 219–232. ACM, 2013. (Cited on page 33)

[PW04] R. Pucella and V. Weissman. Reasoning about dynamic policies. In *FOSSACS'04*, volume 2987 of *Lecture Notes in Computer Science*, pages 453–467, 2004. (Cited on page 40)

[PWZ13] R. Piskac, Th. Wies, and D. Zufferey. Automating separation logic using SMT. In *CAV'13*, volume 8044 of *Lecture Notes in Computer Science*, pages 773–789. Springer, 2013. (Cited on pages 35, 137, 138, 139, 140, 141, 142, 144, 145, 148, 149, 169, 170)

[PWZ14] R. Piskac, Th. Wies, and D. Zufferey. GRASShopper - complete heap verification with mixed specifications. In *TACAS'14*, volume 8413 of *Lecture Notes in Computer Science*, pages 124–139. Springer, 2014. (Cited on pages 141, 142)

[Pym02] D. Pym. *The Semantics and Proof Theory of the Logic of Bunched Implications*, volume 26 of *Applied Logic*. Kluwer Academic Publishers, 2002. (Cited on pages 20, 32, 33, 34, 169)

[QGSM13] X. Qiu, P. Garg, A. Stefanescu, and P. Madhusudan. Natural proofs for structure, data, and separation. In *PLDI'13*, pages 231–242. ACM, 2013. (Cited on page 38)

[Qiu13] X. Qiu. *Automatic techniques for proving correctness of heap-manipulating programs*. PhD thesis, University of Illinois, 2013. (Cited on page 39)

[Rab69] M. Rabin. Decidability of second-order theories and automata on infinite trees. *Transactions of the American Mathematical Society*, 41:1–35, 1969. (Cited on page 133)

[Rab14] A. Rabinovich. A Proof of Kamp's theorem. *LMCS*, 10(1), 2014. (Cited on pages 66, 109)

[RBHC07] Z. Rakamaric, R. Bruttomesso, A. Hu, and A. Cimatti. Verifying heap-manipulating programs in an SMT framework. In *ATVA'07*, volume 4762 of *Lecture Notes in Computer Science*, pages 237–252. Springer, 2007. (Cited on page 139)

[Rey00] J.C. Reynolds. Intuitionistic reasoning about shared mutable data structure. In *Millennial Perspectives in Computer Science*, pages 303–321. Palgrave Macmillan Limited, 2000. (Cited on page 20)

[Rey02] J.C. Reynolds. Separation logic: a logic for shared mutable data structures. In *LICS'02*, pages 55–74. IEEE, 2002. (Cited on pages 10, 11, 14, 22, 29, 41, 44)

[Roh04] Ph. Rohde. Moving in a crumbling network: The balanced case. In *CSL'04*, volume 3210 of *Lecture Notes in Computer Science*, pages 310–324. Springer, 2004. (Cited on page 40)

[Ros15] G. Rosu. Matching Logic – Extended Abstract. In *RTA'15*, pages 5–21. Leibniz-Zentrum für Informatik, LIPICS, 2015. (Cited on page 166)

[Sav70] W.J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177–192, 1970. (Cited on page 160)

[SC14] M. Sighireanu and D. Cok. Report on SL-COMP 2014. *Journal of Satisfiability, Boolean Modeling and Computation*, 2014. To appear. (Cited on pages 140, 141, 171)

[Sch15] S. Schmitz. Complexity hierarchies beyond Elementary. *ACM Transactions on Computation Theory*, 2015. To appear. (Cited on page 127)

[Spa93] E. Spaan. The complexity of propositional tense logics. In M. de Rijke, editor, *Diamonds and Defaults*, pages 287–309. Kluwer Academic Publishers, Series Studies in Pure and Applied Intensional Logic, Volume 229, 1993. (Cited on pages 52, 158)

[SS15] M. Schwerhoff and A. Summers. Lightweight support for magic wands in an automatic verifier. In *ECOOP'15*, pages 999–1023. Leibniz-Zentrum für Informatik, LIPICS, 2015. (Cited on page 21)

[ST11] L. Segoufin and S. Torunczyk. Automata based verification over linearly ordered data domains. In *STACS'11*, pages 81–92, 2011. (Cited on page 113)

[Sto74] L. Stockmeyer. *The complexity of decision problems in automata theory and logic*. PhD thesis, Department of Electrical Engineering, MIT, 1974. (Cited on page 127)

[Sto77] L. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3:1–21, 1977. (Cited on pages 45, 161, 162)

[Str94] H. Straubing. *Finite Automata, Formal Logic, and Circuit Complexity*. Progress in Theoretical Computer Science. Birkhäuser, 1994. (Cited on page 109)

[SZ12] Th. Schwentick and Th. Zeume. Two-variable logic and two order relations. *LMCS*, 8:1–27, 2012. (Cited on pages 66, 118)

[TBR14] A. Thakur, J. Breck, and T. Reps. Satisfiability modulo abstraction for separation logic with linked lists. In *SPIN'14*, pages 58–67. ACM, 2014. (Cited on pages 21, 53, 171)

[Tra63] B. Trakhtenbrot. Impossibility of an algorithm for the decision problem in finite classes. *AMS Translations, Series 2*, 23:1–5, 1963. (Cited on pages 36, 38)

[Tue11] Th. Tuerk. *A separation logic fragment for HOL*. PhD thesis, University of Cambridge, 2011. (Cited on page 11)

[TW13] N. Totla and T. Wies. Complete instantiation-based interpolation. In *POPL'13*, pages 537–548. ACM, 2013. (Cited on page 144)

[TZ04] C. Tinelli and C. Zarba. Combining decision procedures for sorted theories. In *JELIA'04*, volume 3229 of *Lecture Notes in Computer Science*, pages 641–653. Springer, 2004. (Cited on pages 141, 144)

[Var82] M. Vardi. The complexity of relational query languages. In *14th Annual ACM Symposium on Theory of Computing*, pages 137–146. ACM, 1982. (Cited on page 25)

[Var88] M. Vardi. A temporal fixpoint calculus. In *15th Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages, San Diego*, pages 250–259. ACM, 1988. (Cited on page 109)

[vB76] J. van Benthem. *Correspondence Theory*. PhD thesis, Mathematical Institute, University of Amsterdam, 1976. (Cited on pages 129, 139)

[vB05] J. van Benthem. An Essay on Sabotage and Obstruction. In *Mechanizing Mathematical Reasoning, Essays in Honor of Jörg Siekmann on the Occasion of his 69th Birthday*, pages 268–276. Springer Verlag, 2005. (Cited on page 39)

[Ven91] Y. Venema. A modal logic for chopping intervals. *Journal of Logic and Computation*, 1(4):453–476, 1991. (Cited on pages 106, 110)

[VJP15] F. Vogels, B. Jacobs, and F. Piessens. Featherweight VeriFast. *CoRR*, 2015. (Cited on pages 11, 12)

[VP07] V. Vafeiadis and M. Parkinson. A Marriage of Rely/Guarantee and Separation Logic. In *CONCUR'07*, volume 4703 of *Lecture Notes in Computer Science*, pages 256–271. Springer, 2007. (Cited on pages 11, 25)

[WDF+09] Ch. Weidenbach, D. Dimova, A. Fietzke, R. Kumar, M. Suda, and P. Wischnewski. SPASS version 3.5. In *CADE'09*, volume 5663 of *Lecture Notes in Computer Science*, pages 140–145. Springer, 2009. (Cited on page 162)

[Web04] T. Weber. Towards mechanized program verification with separation logic. In *CSL'04*, volume 3210 of *Lecture Notes in Computer Science*, pages 250–264, 2004. (Cited on page 39)

[Wei11] Ph. Weis. *Expressiveness and Succinctness of First-Order Logic on Finite Words*. PhD thesis, University of Massachussetts, 2011. (Cited on pages 109, 117)

[WMK11] T. Wies, M. Muñiz, and V. Kuncak. An efficient decision procedure for imperative tree data structures. In *CADE'11*, volume 6803 of *Lecture Notes in Computer Science*, pages 476–491. Springer, 2011. (Cited on page 142)

[Wol83] P. Wolper. Temporal logic can be more expressive. *Information and Computation*, 56:72–99, 1983. (Cited on page 109)

[Yan01] H. Yang. *Local Reasoning for Stateful Programs*. PhD thesis, University of Illinois, Urbana-Champaign, 2001. (Cited on pages 30, 38, 53, 138, 149, 150, 154, 169)

[YLB+08] H. Yang, O. Lee, J. Berdine, C. Calcagno, B. Cook, D. Distefano, and P. O'Hearn. Scalable shape analysis for systems code. In *CAV'08*, volume 5123 of *Lecture Notes in Computer Science*, pages 385–398. Springer, 2008. (Cited on page 11)

[YRS+06] G. Yorsh, A. Rabinovich, M. Sagiv, A. Meyer, and A. Bouajjani. A logic of reachable patterns in linked data-structures. In *FOSSACS'06*, volume 3921 of *Lecture Notes in Computer Science*, pages 94–110. Springer, 2006. (Cited on page 35)

[YRSW03] E. Yahav, Th. Reps, M. Sagiv, and R. Wilhelm. Verifying temporal heap properties specified via evolution logic. In *ESOP'03*, volume 2618 of *Lecture Notes in Computer Science*, pages 204–222. Springer, 2003. (Cited on page 12)

[Zar03] C. Zarba. Combining sets with elements. In *Verification: Theory and Practice, Essays Dedicated to Zohar Manna on the Occasion of His 64th Birthday*, volume 2772 of *Lecture Notes in Computer Science*, pages 762–782. Springer, 2003. (Cited on page 142)

[Zho08] Zhou Chaochen. In D. Bjorner and M. Henson, editors, *Logics of Specification Languages*, chapter Reviews on "Duration Calculus", pages 609–611. Springer, 2008. Monographs in Theoretical Computer Science. An EATCS Series. (Cited on page 127)

Index