Separation Logics and Modalities: A Survey

Stéphane Demri* Morgan Deters†

Like modal logic, temporal logic, or description logic, separation logic has become a popular class of logical formalisms in computer science, conceived as assertion languages for Hoare-style proof systems with the goal to perform automatic program analysis. In a broad sense, separation logic is often understood as a programming language, an assertion language and a family of rules involving Hoare triples. In this survey, we present similarities between separation logic as an assertion language and modal and temporal logics. Moreover, we propose a selection of landmark results about decidability, complexity and expressive power.

Keywords: Separation logic, decidability, computational complexity, expressive power, temporal logic, modal logic, first-order logic, second-order logic.

1. Introduction

1.0.0.1 When separation logic joins the club. 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$^+$01]. This is now a well-established approach for the formal verification of computer systems: one models the system to be verified by a mathematical structure (typically a directed graph) and expresses behavioral properties in a logical formalism (typically a temporal logic). Verification by model-checking [CGP00] consists of developing algorithms whose goal is to verify whether the logical properties are satisfied by the abstract model. The development of tools is done in parallel with the design of techniques to optimize the verification process. Apart from the development of model-checkers such as Cadence SMV, SPIN, or Uppaal, the transfer towards industrial applications is also present in research and development units. The development of description logics for knowledge representation has also followed a successful path, thanks to a permanent interaction between theoretical works, pushing ever further the high complexity and undecidability borders, and more applied works dedicated to the design of new tools and the production of more and more applications, especially in the realm of ontology languages. The wealth of research on description logic is best illustrated by [BCM$^+$03], in which can be found many chapters on theory, implementations, and applications. By contrast, Chapter 1 of [BCM$^+$03] provides a gentle introduction to description logics and recalls that its roots can be traced back a few decades. 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 this paper,

*Work partially supported by the EU Seventh Framework Programme under grant agreement No. PIOF-GA-2011-301166 (DATAVERIF). This survey paper follows an invited talk given at AIML'14, Groningen, August 2014.
†Work partially supported by the Air Force Office of Scientific Research under award FA9550-09-1-0596, and the National Science Foundation under grant 0644299.

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 modal operator $\Box$ [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* operator, and in many other logical formalisms such as in graph logics [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.

1.0.0.2 Separation and composition. Separation logic has been introduced as an extension of Hoare logic [Hoa69] 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. No doubt that this type of statement is an oversimplification (apart from the fact that it may appear a bit old-fashioned to most modal logicians) but this may help to get a first picture. 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 (see e.g. cancellative partial commutative monoids in Section 2.4).

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] (see also [SC14] for another list of solvers). Actually, nowadays, many tools support separation logic as an assertion language and, more importantly, in order to produce interactive proofs with separation logic, several proof assistants encode the logic, see e.g. [Tue11].

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 concretization 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 concretization of Boolean BI.

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 characterizations 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.

1.0.0.3 Content of the paper. The goal of this paper is twofold. First, we would like to emphasize 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, and so forth. Second, we wish to present landmark results about decidability, complexity, and expressive power, providing a survey on the theoretical side of separation logic. These are standard themes for studying logics in computer science and we deliberately focus on the logical aspects for separation logic, providing a wealth of bibliographical references for untouched subjects in the paper, such as the design of verification algorithms for tools using separation logic as an assertion language or the presentation of applications. 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.

In Section 2.2, we introduce the plain version for separation logic and then in the rest of Section 2, we present a few variants with abstract memory models or with data values. Even though these logical formalisms provide a partial view on separation logics, we stick to them in the rest of the paper to present all the results, from decidability to expressive power. First-order separation logic 1SL with one record field plays the role of basic separation logic as modal logic K could play the role of basic modal logic [BdRV01]. Section 3 shows connections between separation logics and classical logic augmented with second-order features, weak second-order logic, interval temporal logic ITL [Mos83], modal logic and multi-dimensional modal logics. The concept of separation, made completely explicit in separation logic, happens to be very natural and already present in many logical formalisms such as in interval temporal logics with the chop operator or more recently in separation logic for knowledge representation [Her13]. Moreover, Section 3 recalls why 1SL and variants can be easily viewed as fragments of weak dyadic second-order logic. Relationships with temporal logics with freeze on data words (see e.g. [FS09]) are also presented in Section 4.1.4 whereas some analogy with Presburger arithmetic can be found in Section 5.1 (see also in Section 5.2 the relationships between 1SL and first-order theory of natural numbers with addition and multiplication).

Section 4 is dedicated to decidability and complexity results. We present the standard proof techniques for several undecidability results and provide the justification for several decidability ones. Finally, we state several complexity results for tractable fragments but also for fragments with non-recursive elementary complexity. Section 5 presents results about the expressive power for propositional separation logics (such as 1SL0) and for first-order separation logics such as 1SL, known to be as expressive as weak-second-order logic [BDL12]. Section 6 concludes the technical part of the paper by providing several pointers about proof systems for separation logics, from sequent-style calculi to decision procedures built on SMT solvers.

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].

2. Separation Logics

2.1 *Basics on separation logic and formal verification*

In this section, we provide a brief introduction to separation logic by showing how it is related to formal verification. Nevertheless, 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.

2.1.1 Hoare logic

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. Its hallmark, the *Hoare triple*, is composed of assertions ϕ and ψ and command C:

$$\{\phi\}\ \mathrm{C}\ \{\psi\}$$

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

$$\frac{\{\phi\}\ \mathrm{C}_1\ \{\psi\} \quad \{\psi\}\ \mathrm{C}_2\ \{\varphi\}}{\{\phi\}\ \mathrm{C}_1;\mathrm{C}_2\ \{\varphi\}}\ \text{composition}$$

Preconditions can be strengthened and postconditions can be weakened in a natural fashion (this is used later in Section 4.3.1):

$$\frac{\phi \Rightarrow \phi' \quad \{\phi'\}\ \mathrm{C}\ \{\psi\} \quad \psi \Rightarrow \psi'}{\{\phi\}\ \mathrm{C}\ \{\psi'\}}\ \text{strengthen/weaken}$$

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

$$\frac{}{\{\phi[\mathrm{x}/\mathrm{e}]\}\ \mathrm{x} := \mathrm{e}\ \{\phi\}}\ \text{assignment}$$

However, this has a severe limitation when pointers are involved. Consider the following triple, an instance of the above schema:

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

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

Naturally, this 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.

2.1.2 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.

Recognizing 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 by Ishtiaq and O'Hearn [IO01]. In fact, their efforts (together with Pym) on bunched implication (BI) logics [OP99, Pym02] gave them a somewhat more general perspective, and they recognized 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 recognized; see [LP14, Section 1] and [HCGT14, Section 8] for recent discussions on this topic (see also [TBR14]).

Second, they introduced the *frame rule*, important for local reasoning [IO01]. Given a Hoare triple $\{\phi\}$ 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. It will be made precise later, after suitable formal definitions have been made.

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 paper, 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.

2.2 *Separation logic on concrete models*

Let us start by defining separation logics on concrete models, namely on heaps. Let $\mathrm{PVAR} = \{\mathrm{x}_1, \mathrm{x}_2, \ldots\}$ be a countably infinite set of *program variables* and $\mathrm{FVAR} = \{\mathrm{u}_1, \mathrm{u}_2, \ldots\}$ 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 $\mathfrak{s} : \mathrm{PVAR} \to \mathbb{N}$ (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, sometime, 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 $\mathfrak{h}$).

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 $(\mathfrak{s}_1, \mathfrak{h}_1) \uplus (\mathfrak{s}_2, \mathfrak{h}_2) \stackrel{\text{def}}{=} (\mathfrak{s}_1, \mathfrak{h}_1 \uplus \mathfrak{h}_2)$. In Figure 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) 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 x_i just above a node means that its value by the store $\mathfrak{s}$ is precisely that node. In Figure 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 x_3 encodes that $(\mathfrak{s}, \mathfrak{h}) \models \mathrm{x}_3 \hookrightarrow \mathrm{x}_3$. Similarly, $(\mathfrak{s}, \mathfrak{h}_1) \models \mathrm{x}_3 \hookrightarrow \mathrm{x}_3$ but not $(\mathfrak{s}, \mathfrak{h}_2) \models \mathrm{x}_3 \hookrightarrow \mathrm{x}_3$.

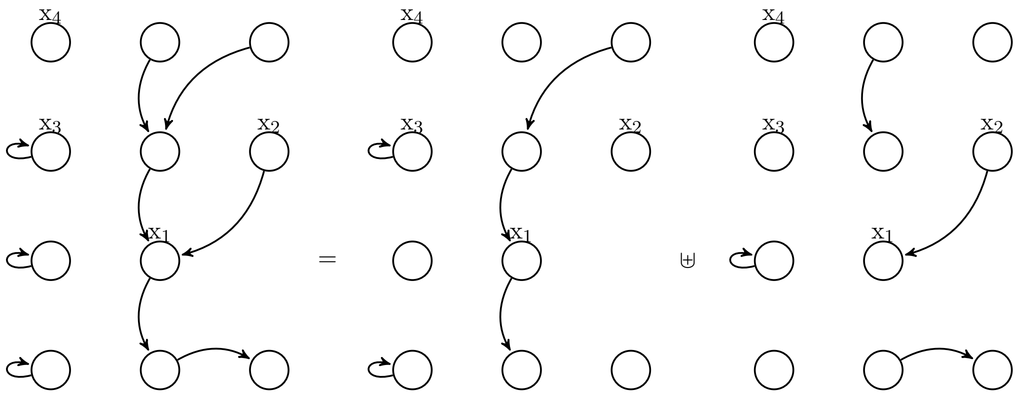

Figure 1. Disjoint memory states with one record field

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 $\mathrm{e} ::= \mathrm{x} \mid \mathrm{u}$ where $\mathrm{x} \in \mathrm{PVAR}$ and $\mathrm{u} \in \mathrm{FVAR}$, and *atomic formulae* of the form

$$\pi ::= \mathrm{e} = \mathrm{e}' \mid \mathrm{e} \hookrightarrow \mathrm{e}_1, \ldots, \mathrm{e}_k \mid \mathrm{emp} \mid \perp$$

Formulae are defined by the grammar

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

where $\mathrm{u} \in \mathrm{FVAR}$. The connective $*$ is *separating conjunction* and $\mathrel{-\!\!*}$ is *separating implication*, usually called the *magic wand*. We make use of standard notations for derived connectives for this and all logics defined in this paper. As in classical first-order logic, an *assignment* is a map $\mathfrak{f} : \mathrm{FVAR} \to \mathbb{N}$. The satisfaction relation $\models$ is parameterized by assignments (obvious clauses for Boolean connectives are omitted):

- $(\mathfrak{s}, \mathfrak{h}) \models_{\mathfrak{f}} \mathrm{emp}$ iff $\mathrm{dom}(\mathfrak{h}) = \emptyset$.
- $(\mathfrak{s}, \mathfrak{h}) \models_{\mathfrak{f}} \mathrm{e} = \mathrm{e}'$ iff $[\mathrm{e}] = [\mathrm{e}']$, with $[\mathrm{x}] \stackrel{\text{def}}{=} \mathfrak{s}(\mathrm{x})$ and $[\mathrm{u}] \stackrel{\text{def}}{=} \mathfrak{f}(\mathrm{u})$.
- $(\mathfrak{s}, \mathfrak{h}) \models_{\mathfrak{f}} \mathrm{e} \hookrightarrow \mathrm{e}_1, \ldots, \mathrm{e}_k$ iff $[\mathrm{e}] \in \mathrm{dom}(\mathfrak{h})$ and $\mathfrak{h}([\mathrm{e}]) = ([\mathrm{e}_1], \ldots, [\mathrm{e}_k])$.
- $(\mathfrak{s}, \mathfrak{h}) \models_{\mathfrak{f}} \phi_1 * \phi_2$ iff $\mathfrak{h} = \mathfrak{h}_1 \uplus \mathfrak{h}_2$, $(\mathfrak{s}, \mathfrak{h}_1) \models_{\mathfrak{f}} \phi_1$, $(\mathfrak{s}, \mathfrak{h}_2) \models_{\mathfrak{f}} \phi_2$ for some $\mathfrak{h}_1, \mathfrak{h}_2$.
- $(\mathfrak{s}, \mathfrak{h}) \models_{\mathfrak{f}} \phi_1 \mathrel{-\!\!*} \phi_2$ iff for all $\mathfrak{h}'$, if $\mathfrak{h} \perp \mathfrak{h}'$ and $(\mathfrak{s}, \mathfrak{h}') \models_{\mathfrak{f}} \phi_1$ then $(\mathfrak{s}, \mathfrak{h} \uplus \mathfrak{h}') \models_{\mathfrak{f}} \phi_2$.
- $(\mathfrak{s}, \mathfrak{h}) \models_{\mathfrak{f}} \exists\, \mathrm{u}\, \phi$ iff there is $\mathfrak{l} \in \mathbb{N}$ such that $(\mathfrak{s}, \mathfrak{h}) \models_{\mathfrak{f}[\mathrm{u} \mapsto \mathfrak{l}]} \phi$ where $\mathfrak{f}[\mathrm{u} \mapsto \mathfrak{l}]$ is the assignment equal to $\mathfrak{f}$ except that u takes the value $\mathfrak{l}$.

When ϕ has no program variables, we also write $\mathfrak{h} \models_{\mathfrak{f}} \phi$ to mean that ϕ is satisfied on the heap $\mathfrak{h}$ under the assignment $\mathfrak{f}$.

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

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

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

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

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

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.

For $k' \geq 0$, we write kSLk' 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 u. Moreover, kSL$k'(\mathrel{-\!\!*})$ [resp. kSL$k'(*)$] denotes the fragment of kSLk' 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 $\mathrm{e} \hookrightarrow \mathrm{e}_1, \ldots, \mathrm{e}_k$ by $\mathrm{e} \hookrightarrow \mathrm{e}_1, \mathrm{e}_1, \ldots, \mathrm{e}_k$ 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 kSLk' 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 $(\mathfrak{s}, \mathfrak{h}) \models_{\mathfrak{f}} \phi$. 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 $(\mathfrak{s}, \mathfrak{h}) \models_{\mathfrak{f}} \phi$ ($\mathfrak{s}$ 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 $\mathrel{-\!\!*}$ since $\mathrel{-\!\!*}$ is the *adjunct* of $*$. This means that $(\phi * \psi) \Rightarrow \varphi$ is valid iff $\phi \Rightarrow (\psi \mathrel{-\!\!*} \varphi)$ is valid. Note that this does not imply that the formula $((\phi * \psi) \Rightarrow \varphi) \Leftrightarrow (\phi \Rightarrow (\psi \mathrel{-\!\!*} \varphi))$ is valid (otherwise $*$ and $\mathrel{-\!\!*}$ 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. Section 5.2, without sacrificing the expressive power.

We also introduce so-called *septraction* operator $\overset{\neg}{\mathrel{-\!\!*}}$: $\phi \overset{\neg}{\mathrel{-\!\!*}} \psi$ is defined as the formula $\neg(\phi \mathrel{-\!\!*} \neg\psi)$. As far as we know, its first appearance was in [VP07]. So, $(\mathfrak{s}, \mathfrak{h}) \models_{\mathfrak{f}} \phi \overset{\neg}{\mathrel{-\!\!*}} \psi$ iff there is a heap $\mathfrak{h}'$ disjoint from $\mathfrak{h}$ such that $(\mathfrak{s}, \mathfrak{h}') \models_{\mathfrak{f}} \phi$ 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.

2.3 *A bunch of properties stated in 1SL*

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: $\neg\mathrm{emp} * \cdots * \neg\mathrm{emp}$ (α times).
- The variable x is allocated in the heap: $\mathrm{alloc}(\mathrm{x}) \overset{\mathrm{def}}{=} (\mathrm{x} \hookrightarrow \mathrm{x}) \mathrel{-\!\!*} \perp$.

- The variable x points to a location that is a self-loop:

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

In the following, let u and $\overline{\mathrm{u}}$ be the variables u_1 and u_2, in either order. Note that any formula $\phi(\mathrm{u})$ with free variable u can be turned into an equivalent formula with free variable $\overline{\mathrm{u}}$ 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:

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

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

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

  It is easy to see that one can also define in 1SL that the heap domain has at least $k \geq 0$ elements (written $\mathrm{size} \geq k$).
- u has a successor: $\mathrm{alloc}(\mathrm{u}) \stackrel{\text{def}}{=} \exists\, \overline{\mathrm{u}}\ \mathrm{u} \hookrightarrow \overline{\mathrm{u}}$
- u has at least α predecessors: $\sharp\mathrm{u} \geq \alpha \stackrel{\text{def}}{=} \overbrace{(\exists\, \overline{\mathrm{u}}\ (\overline{\mathrm{u}} \hookrightarrow \mathrm{u})) * \cdots * (\exists\, \overline{\mathrm{u}}\ (\overline{\mathrm{u}} \hookrightarrow \mathrm{u}))}^{\alpha \text{ times}}$
- u has at most α predecessors: $\sharp\mathrm{u} \leq \alpha \stackrel{\text{def}}{=} \neg\,(\sharp\mathrm{u} \geq \alpha + 1)$
- u has exactly α predecessors: $\sharp\mathrm{u} = \alpha \stackrel{\text{def}}{=} (\sharp\mathrm{u} \geq \alpha) \wedge \neg(\sharp\mathrm{u} \geq \alpha + 1)$
- There is a non-empty path from u to $\overline{\mathrm{u}}$ and nothing else except loops that exclude $\overline{\mathrm{u}}$:

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

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

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

  One can show that $\mathfrak{h} \models_{\mathfrak{f}} \mathrm{reach}(\mathrm{u}, \overline{\mathrm{u}})$ iff there is $i \in \mathbb{N}$ such that $\mathfrak{h}^i(\mathfrak{f}(\mathrm{u})) = \mathfrak{f}(\overline{\mathrm{u}})$. 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 u to $\overline{\mathrm{u}}$ and nothing else can be defined as follows:

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

  $\mathrm{sreach}(\mathrm{u}, \overline{\mathrm{u}})$ can be understood as the 'strict' reachability predicate and it is usually written as the segment predicate $\mathrm{ls}(\mathrm{u}, \overline{\mathrm{u}})$.
- There is at most a single connected component (and nothing else):

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

- There are exactly two components: $2\mathrm{comps} \overset{\mathrm{def}}{=} 1\mathrm{comp} * 1\mathrm{comp}$

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 emp is logically equivalent to the following formula using only two quantified variables:

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

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

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

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

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

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

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

2.4 *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. This is 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 BI in concrete heaps. Moreover, sometimes, problems can be easily solved on abstract models because more freedom is allowed (see e.g. [BV14, HCGT14] or Theorem 4.2).

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

(MON$_{\mathrm{ms}}$) $\uplus$ 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$_{\mathrm{ms}}$) $\uplus$ is associative and commutative,
(CAN$_{\mathrm{ms}}$) $\uplus$ 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$_{\mathrm{ms}}$) 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 $(\mathfrak{H}\mathfrak{S}_k, \uplus, \mathfrak{U}_k)$ by abstracting the essential features and can be viewed as a Kripke frame for a multi-

*See also the ANR project DYNRES at `http://anr-dynres.loria.fr/` partly dedicated to abstract separation logics to reason about resources.

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 $m \circ U = \{m\}$.

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 $\text{PROP} = \{p_1, p_2, \ldots\}$ 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 interpretation of propositional variables done via $\mathfrak{V}$. The set of formulae is then defined as follows (note that it is a propositional language):

$$\phi, \psi ::= \text{emp} \mid p \mid \phi \wedge \psi \mid \neg\phi \mid \phi * \psi \mid \phi \mathrel{-\!\!*} \psi$$

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):

- $m \models_{\mathfrak{V}} \text{emp}$ 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 $m \in \mathfrak{V}(p)$.
- $m \models_{\mathfrak{V}} \phi_1 * \phi_2$ iff for some $m_1, m_2 \in M$, we have $m = m_1 \circ m_2$, $m_1 \models_{\mathfrak{V}} \phi_1$ and $m_2 \models_{\mathfrak{V}} \phi_2$.
- $m \models_{\mathfrak{V}} \phi_1 \mathrel{-\!\!*} \phi_2$ iff for all $m' \in M$ such that $m \circ m'$ is defined, if $m' \models_{\mathfrak{V}} \phi_1$ then $m \circ m' \models_{\mathfrak{V}} \phi_2$.

In the above definition for the satisfaction relation, the model $(M, \circ, U)$ is implicit but we also sometimes use the notation $(M, \circ, U), m \models_{\mathfrak{V}} \phi$ to emphasize the separation model in use.

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 $m \models_{\mathfrak{V}} \phi$. 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 $m \models_{\mathfrak{V}} \phi$. We write SL$(M, \circ, U)$ to denote the propositional separation logic defined from the separation model $(M, \circ, U)$ with propositional variables. When $\mathcal{C}$ is a class of separation models, we can also define the propositional separation logic SL$(\mathcal{C})$ 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$(\mathcal{C})$ iff there exist $(M, \circ, U)$ in $\mathcal{C}$, $m \in M$ and a valuation $\mathfrak{V}$ such that $(M, \circ, U), m \models_{\mathfrak{V}} \phi$.

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. Note that 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]).

2.5 *Separation logic with data*

In memory states defined in Section 2.2, 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. [BBL09, BDES09, MPQ11]. 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 would be 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 $\mathrm{R}_i(\mathrm{e}_1, \ldots, \mathrm{e}_{a(i)})$ 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 $\mathrm{e} \sim \mathrm{e}'$ 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:

- $(\mathfrak{s}, \mathfrak{h}, \mathfrak{d}) \models_{\mathfrak{f}} \mathrm{R}_i(\mathrm{e}_1, \ldots, \mathrm{e}_{a(i)}) \overset{\text{def}}{\Leftrightarrow} \mathfrak{d}([\mathrm{e}_1]), \ldots, \mathfrak{d}([\mathrm{e}_{a(i)}])$ are defined and $\mathfrak{R}_i(\mathfrak{d}([\mathrm{e}_1]), \ldots, \mathfrak{d}([\mathrm{e}_{a(i)}]))$.

Note that in $k\mathrm{SL}[\mathfrak{D}, (\mathfrak{R}_i)_{i\in I}]$, 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 (see e.g., Section 2.2) happen to be quite expressive (see forthcoming developments) 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 $1SL^{sdc}_{<}$ be the fragment of $1SL(*)[\mathbb{Z}, \leq, =]$ (without magic wand) such that the atomic formulae can only occur in subformulae of the form either $\mathrm{e} \hookrightarrow \mathrm{e}' \wedge (\mathrm{e} \leq \mathrm{e}')$ or $\mathrm{e} \hookrightarrow \mathrm{e}' \wedge (\mathrm{e}' \leq \mathrm{e})$. 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 x_1 and x_2 (assuming that $\mathrm{reach}(\mathrm{x}_1, \mathrm{x}_2) \wedge \mathrm{alloc}(\mathrm{x}_1) \wedge \mathrm{alloc}(\mathrm{x}_2)$ holds true) is sorted:

$$\forall \mathrm{u}, \mathrm{u}' \ ((\mathrm{reach}(\mathrm{x}_1, \mathrm{u}) \wedge \mathrm{reach}(\mathrm{u}, \mathrm{u}') \wedge \mathrm{reach}(\mathrm{u}', \mathrm{x}_2)) \Rightarrow \mathrm{u} \leq \mathrm{u}')$$

where $\leq$ is "less than or equal relation" 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.

3. Separation Logics and Friends

In this section, we present several logics that are closely related to some fragments of kSL. For instance, $1SL(*)$ can be also viewed as the spatial logic for graphs but restricted to the graphs obtained from heaps with one record field [DGG07, Loz04a] (see also context logic in [CGZ07, GZ07] or ambient logic [CG00]). More relationships shall be established below.

3.1 *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 2.2 that there is a formula $\mathrm{reach}(\mathrm{u}, \overline{\mathrm{u}})$ in $1SL2(*)$ such that $\mathfrak{h} \models_{\mathfrak{f}} \mathrm{reach}(\mathrm{u}, \overline{\mathrm{u}})$ iff $\mathfrak{f}(\mathrm{u})\mathfrak{R}^{\star}\mathfrak{f}(\overline{\mathrm{u}})$, 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 separation 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.

3.2 *Weak second-order logic*

Next we define weak second-order logic kWSOL, for $k \geq 1$. The sets PVAR and FVAR are defined as for kSL as well as the expressions e. We also consider a family $\mathrm{SVAR} = (\mathrm{SVAR}_i)_{i\geq 1}$ of second-order variables, denoted by $\mathrm{P}, \mathrm{Q}, \mathrm{R}, \ldots$ that are interpreted as finite relations over $\mathbb{N}$. Each variable in 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 $\mathrm{P} \in \mathrm{SVAR}_i$, $\mathfrak{f}(\mathrm{P})$ is a finite subset of $\mathbb{N}^i$.

Atomic formulae take the form

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

Formulae of kWSOL are defined by the grammar

$$\phi, \psi ::= \pi \mid \phi \wedge \psi \mid \neg\phi \mid \exists\, \mathrm{u}\, \phi \mid \exists\, \mathrm{P}\, \phi$$

where $\mathrm{P} \in \mathrm{SVAR}_n$ for some $n \geq 1$. We write kMSOL (monadic second-order logic) to denote the restriction of kWSOL to second-order variables in SVAR_1 and kDSOL (dyadic second-order logic) to denote its restriction to 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):

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

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 $(\mathfrak{s}, \mathfrak{h}) \models \phi$. 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 SVAR_2, and this can be used to relativize a formula from DSOL in order to check finite satisfiability.

Theorem 3.1. [BDL12] kWSOL and kDSOL have the same expressive power.

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 Section 5.2.

Translation in the other direction concerns us in this section. Separation logic 1SL can easily be translated into 1DSOL. The presentation given here is by a simple internalization.

First, some formula definitions useful for the translation.

- $\mathrm{init}(\mathrm{P}) \stackrel{\text{def}}{=} \forall\, \mathrm{u}\, \mathrm{v}\, (\mathrm{P}(\mathrm{u},\mathrm{v}) \Leftrightarrow \mathrm{u} \hookrightarrow \mathrm{v})$,
- $\mathrm{heap}(\mathrm{P}) \stackrel{\text{def}}{=} \forall\, \mathrm{u}\, \mathrm{v}\, \mathrm{w}\, ((\mathrm{P}(\mathrm{u},\mathrm{v}) \wedge \mathrm{P}(\mathrm{u},\mathrm{w})) \Rightarrow \mathrm{v} = \mathrm{w})$ (functionality),
- $\mathrm{P} = \mathrm{Q} \uplus \mathrm{R} \stackrel{\text{def}}{=} \forall\, \mathrm{u}\, \mathrm{v}\, ((\mathrm{P}(\mathrm{u},\mathrm{v}) \Leftrightarrow (\mathrm{Q}(\mathrm{u},\mathrm{v}) \vee \mathrm{R}(\mathrm{u},\mathrm{v})) \wedge \neg(\mathrm{Q}(\mathrm{u},\mathrm{v}) \wedge \mathrm{R}(\mathrm{u},\mathrm{v})))$.

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

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

$$\begin{aligned}
t_{\mathrm{P}}(\mathrm{u} \hookrightarrow \mathrm{v}) &\stackrel{\text{def}}{=} \mathrm{P}(\mathrm{u},\mathrm{v}) \\
t_{\mathrm{P}}(\phi * \psi) &\stackrel{\text{def}}{=} \exists\, \mathrm{Q}\, \mathrm{Q}' \big(\mathrm{P} = \mathrm{Q} \uplus \mathrm{Q}' \wedge t_{\mathrm{Q}}(\phi) \wedge t_{\mathrm{Q}'}(\psi)\big) \\
t_{\mathrm{P}}(\phi -\!\!* \psi) &\stackrel{\text{def}}{=} \forall\, \mathrm{Q}\ \big(((\exists\, \mathrm{Q}'\ \mathrm{heap}(\mathrm{Q}') \wedge \mathrm{Q}' = \mathrm{Q} \uplus \mathrm{P}) \wedge \mathrm{heap}(\mathrm{Q}) \wedge t_{\mathrm{Q}}(\phi)) \\
&\qquad \Rightarrow (\exists\, \mathrm{Q}'\ \mathrm{heap}(\mathrm{Q}') \wedge \mathrm{Q}' = \mathrm{Q} \uplus \mathrm{P} \wedge t_{\mathrm{Q}'}(\psi))\big)
\end{aligned}$$

Theorem 3.2. (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 $(\mathfrak{s}, \mathfrak{h}) \models \phi$ in 1SL iff $(\mathfrak{s}, \mathfrak{h}) \models t(\phi)$ in 1DSOL.

Note that of course there must then also exist a translation from the smaller fragment 1SL2($-\!\!*$) into 1DSOL. This result (along with Theorem 3.1 above) will be useful later in showing expressive power results of separation logic.

3.2.0.1 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$^+$14, BFGN14]. Whereas, it is shown in [BFGN14] that the satisfiability problem for many standard fragments of separation logic augmented with general inductive predicates is decidable and complexity is characterized (see also [IRS13] for bounded tree-width structures), other fragments have been shown to admit decidable entailment problem [IRS13, AGH$^+$14]. These are general results that are very promising for automatic verification of programs, despite the generality of the defined predicates.

3.3 *Interval temporal logics*

Temporal logics are dedicated to temporal reasoning, see e.g. [MP92] and it is common for such logics to reason about propositions that vary at different points in time, but it is useful as well to consider intervals; an early and classical study for reasoning about intervals can be found in [All83]. Interval-based temporal logics admit time intervals as first-class objects. One of the most prominent interval-based logics is Propositional Interval Temporal Logic (PITL), introduced in [Mos83] for the verification of hardware components. It contains the so-called 'chop' operation that consists of chopping an interval into two subintervals. Naturally, this is rather reminiscent of the separating conjunction, and indeed the correspondence has recently been made precise in [DD15]. Below, we consider PITL in which propositional variables are interpreted under the *locality condition* (the truth of a propositional variable on an interval depends only on the first letter of the interval) and for which decidability is guaranteed but computational complexity is very high. This will allow us to derive similar bounds for 1SL2($*$).

It is typical in interval temporal logic to not cleave the model into two disjoint pieces, but rather to permit overlap at one point. This gives nonempty models, and is one of the two main issues in showing correspondence between 1SL($*$) and PITL, though it is still possible. The other difficulty is in showing how to use 1SL($*$) to properly cut an interval so that the data encoding is not split in an improper way.

Given $\alpha \geq 1$, we consider the finite alphabet $\Sigma = [1, \alpha]$ and we write PITL$_\Sigma$ to denote propositional interval temporal logic in which the models are non-empty finite words $\mathfrak{w} \in \Sigma^+$. We write PITL instead of PITL$_\Sigma$ when the finite alphabet Σ is clear from the context. Formulae for PITL$_\Sigma$ are defined according to the following abstract grammar:

$$\phi, \psi \quad ::= \quad a \mid \mathrm{pt} \mid \neg\phi \quad \mid \quad \phi \wedge \psi \mid \phi \,\mathbf{C}\, \psi$$

with $a \in \Sigma$. Roughly speaking, a holds true at a word $\mathfrak{w}$ when a is the first letter of $\mathfrak{w}$. Similarly, the atomic formula pt holds true at a word $\mathfrak{w}$ when the word $\mathfrak{w}$ is only a single letter. The connective **C** is the *chop* operator, which chops a word.

Formally, we define a ternary relation chops on words:

$$\mathsf{chops} \overset{\text{def}}{=} \left\{(\mathfrak{w}_1, \mathfrak{w}_2, \mathfrak{w}_3) \in (\Sigma^+)^3 \mid \ \exists\, a, \mathfrak{w}', \mathfrak{w}'' \text{ s.t. } \mathfrak{w}_1 = \mathfrak{w}' a \mathfrak{w}'', \mathfrak{w}_2 = \mathfrak{w}' a, \mathfrak{w}_3 = a\mathfrak{w}''\right\}$$

Then, let us define the satisfaction relation $\models$ for PITL$_\Sigma$ between a word $\mathfrak{w} \in \Sigma^+$ and a formula ϕ:

- $\mathfrak{w} \models a \overset{\text{def}}{\Leftrightarrow}$ the first letter of $\mathfrak{w}$ is a.
- $\mathfrak{w} \models \mathrm{pt} \overset{\text{def}}{\Leftrightarrow}$ the length of $\mathfrak{w}$ is 1.
- $\mathfrak{w} \models \neg\phi \overset{\text{def}}{\Leftrightarrow} \mathfrak{w} \not\models \phi$.
- $\mathfrak{w} \models \phi \wedge \psi \overset{\text{def}}{\Leftrightarrow} \mathfrak{w} \models \phi$ and $\mathfrak{w} \models \psi$.
- $\mathfrak{w} \models \phi \,\mathbf{C}\, \psi \overset{\text{def}}{\Leftrightarrow}$ there exist $\mathfrak{w}_1, \mathfrak{w}_2$ such that $\mathsf{chops}\,(\mathfrak{w}, \mathfrak{w}_1, \mathfrak{w}_2)$, $\mathfrak{w}_1 \models \phi$ and $\mathfrak{w}_2 \models \psi$.

We then have the following results.

Lemma 3.3. Given $\alpha \geq 1$ and $\Sigma = [1, \alpha]$, there is a logarithmic-space translation $t(\cdot)$ of PITL formulae to 1SL2($*$) formulae such that a PITL$_\Sigma$ formula ϕ is satisfiable if and only if the 1SL2($*$) formula $t(\phi)$ is satisfiable.

There are two chief insights that permit this result. First, words may be represented by memory states which can be described by 1SL2($*$) formulae. Briefly, these memory states, called *fishbone heaps* in [DD15] and defined formally herein (see Section 4.1.4), encode a word of length n over a finite alphabet of size m by an acyclic path of length n

where each successive memory location has a number of predecessors that precisely encodes the element of the alphabet corresponding to its position within the word. Care must be taken during the translation so as not to divide these fishbones improperly, else the result may be interpreted as a different word. Second, there is a striking correspondence between PITL's chop and the separating conjunction. As PITL's chop conventionally separates an interval into two, duplicating the point of separation, the translation must take care to translate with respect to the duplicated point.

This result leads directly to Theorem 4.14, shown later.

3.4 *Modal logics*

Separation logic is related to modal logics in many ways, and actually the full paper is dedicated to put forward existing bridges. Nevertheless, below, we present several modal readings for separation logic.

At first, 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].

Secondly, several logical formalisms have included modalities from modal or temporal logics with separating connectives. Whereas a modal BI logic is introduced in [CG13] for expressing dynamic resource properties, temporal reasoning about memory states has been performed in several ways, for instance in Navigation Temporal Logic in [DKR04], in Evolution logic [YRSW03] or in some LTL extension with formulae from separation logic at the atomic level [BDL09, Bro13].

3.4.1 *A taste of multi-dimensional modal logics*

Versions of separation logics with the connectives $*$ and $-\!\!*$ can be also understood as multi-dimensional modal logics, see e.g. [MV97, GKWZ03, HCGT14]. Multi-dimensional modal logics are known as modal logics with n-ary modalities, $n \geq 1$ and models are of the form $\mathfrak{M} = (W, \mathfrak{R}, V)$ with non-empty set W, $\mathfrak{R} \subseteq W^{n+1}$, for some $n \geq 1$ and $V : \mathrm{PROP} \to W$. For each $i \in [1, n+1]$, we can define the n-ary modality $\Diamond_i$ such that $\mathfrak{M}, w \models \Diamond_i(\phi_1, \ldots, \phi_{i-1}, \phi_{i+1}, \ldots, \phi_{n+1})$ iff there is $(w_1, \ldots, w_{n+1}) \in \mathfrak{R}$ with $w_i = w$ such that for all $j \in [1, n+1] \setminus \{i\}$, we have $\mathfrak{M}, w_j \models \phi_j$. Note that when $n = 1$, $\Diamond_1$ corresponds to the standard modality $\Diamond$ and $\Diamond_2$ corresponds to the standard backward modality $\Diamond^{-1}$. So, it is not difficult to observe that 1SL0 can be understood as a multi-dimensional modal logic in the above sense with a unique Kripke model and $n = 2$. Such a unique model $\mathfrak{M} = (W, \mathfrak{R}, V)$ is defined as follows:

- $W \stackrel{\text{def}}{=} \{(\mathfrak{s}, \mathfrak{h}) : \mathfrak{s} : \mathrm{PVAR} \to \mathbb{N},\ \mathfrak{h} : \mathbb{N} \rightharpoonup_{\text{fin}} \mathbb{N}\}$ (i.e., $W \stackrel{\text{def}}{=} \mathfrak{H}\mathfrak{S}_1$).
- $((\mathfrak{s}_1, \mathfrak{h}_1), (\mathfrak{s}_2, \mathfrak{h}_2), (\mathfrak{s}_3, \mathfrak{h}_3)) \in \mathfrak{R} \stackrel{\text{def}}{\Leftrightarrow} \mathfrak{s}_1 = \mathfrak{s}_2 = \mathfrak{s}_3$ and $\mathfrak{h}_1 = \mathfrak{h}_2 \uplus \mathfrak{h}_3$.
- $V(\mathrm{x}_i \hookrightarrow \mathrm{x}_j) \stackrel{\text{def}}{=} \{(\mathfrak{s}, \mathfrak{h}) : \mathfrak{h}(\mathfrak{s}(\mathrm{x}_i)) = \mathfrak{s}(\mathrm{x}_j)\}$.
- $V(\mathrm{x}_i = \mathrm{x}_j) \stackrel{\text{def}}{=} \{(\mathfrak{s}, \mathfrak{h}) : \mathfrak{s}(\mathrm{x}_i) = \mathfrak{s}(\mathrm{x}_j)\}$.
- $V(\mathrm{emp}) \stackrel{\text{def}}{=} \{(\mathfrak{s}, \mathfrak{h}) : \mathrm{dom}(\mathfrak{h}) = \emptyset\}$.

With such a model, $\phi * \psi$ corresponds to $\Diamond_1(\phi, \psi)$, $\phi \overset{-}{-\!\!*} \psi$ corresponds to $\Diamond_2(\psi, \phi)$ or to $\Diamond_3(\psi, \phi)$ ($\uplus$ is commutative). Finally, $\phi -\!\!* \psi$ corresponds to $\neg\Diamond_2(\neg\psi, \phi)$. We invite the reader to consult [HCGT14] for an explicit presentation of Kripke relational frames for abstract separation logics.

3.4.2 A modal logic for heaps

In order to conclude this section mainly dedicated to modal logics, we present a new modal logic with such frames [DD15]. 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.

$$\phi, \psi \quad ::= \quad \bot \mid \neg\, \phi \mid \phi \wedge \psi \mid \Diamond\, \phi \mid \Diamond^{-1}\, \phi \mid \langle \neq \rangle\, \phi \mid \langle \star \rangle\, \phi \mid \phi * \psi \mid \phi \mathrel{-\!\!*} \psi$$

We write MLH($*$) to denote the fragment of MLH without the magic wand operator $\mathrel{-\!\!*}$.

A *model for MLH* $\mathfrak{M}$ is a pair $(\mathbb{N}, \mathfrak{R})$ such that $\mathfrak{R}$ is a binary relation on $\mathbb{N}$ that is finite and functional. 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):

- $\mathfrak{M}, \mathfrak{l} \models \Diamond\phi \overset{\text{def}}{\Leftrightarrow}$ there is $\mathfrak{l}'$ such that $(\mathfrak{l}, \mathfrak{l}') \in \mathfrak{R}$ and $\mathfrak{M}, \mathfrak{l}' \models \phi$,
- $\mathfrak{M}, \mathfrak{l} \models \Diamond^{-1}\phi \overset{\text{def}}{\Leftrightarrow}$ there is $\mathfrak{l}'$ such that $(\mathfrak{l}', \mathfrak{l}) \in \mathfrak{R}$ and $\mathfrak{M}, \mathfrak{l}' \models \phi$,
- $\mathfrak{M}, \mathfrak{l} \models \langle\star\rangle\phi \overset{\text{def}}{\Leftrightarrow}$ there is $\mathfrak{l}'$ such that $(\mathfrak{l}, \mathfrak{l}') \in \mathfrak{R}^*$ and $\mathfrak{M}, \mathfrak{l}' \models \phi$,
- $\mathfrak{M}, \mathfrak{l} \models \langle\neq\rangle\phi \overset{\text{def}}{\Leftrightarrow}$ there is $\mathfrak{l}' \neq \mathfrak{l}$ such that $\mathfrak{M}, \mathfrak{l}' \models \phi$,
- $\mathfrak{M}, \mathfrak{l} \models \phi_1 * \phi_2 \overset{\text{def}}{\Leftrightarrow} (\mathbb{N}, \mathfrak{R}_1), \mathfrak{l} \models \phi_1$ and $(\mathbb{N}, \mathfrak{R}_2), \mathfrak{l} \models \phi_2$ for some partition $\{\mathfrak{R}_1, \mathfrak{R}_2\}$ of $\mathfrak{R}$,
- $\mathfrak{M}, \mathfrak{l} \models \phi_1 \mathrel{-\!\!*} \phi_2 \overset{\text{def}}{\Leftrightarrow}$ for all models $\mathfrak{M}' = (\mathbb{N}, \mathfrak{R}')$ such that $\mathfrak{R} \cap \mathfrak{R}' = \emptyset$ and $\mathfrak{R} \cup \mathfrak{R}'$ is functional, $\mathfrak{M}', \mathfrak{l} \models \phi_1$ implies $(\mathbb{N}, \mathfrak{R} \cup \mathfrak{R}'), \mathfrak{l} \models \phi_2$.

A formula ϕ is satisfiable whenever there is a model $\mathfrak{M}$ and a location $\mathfrak{l}$ such that $\mathfrak{M}, \mathfrak{l} \models \phi$. 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. However, it is possible to design a relational translation t from MLH formulae into 1SL2 formulae by recycling variables. 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, models for MLH are heap graphs. Modal logic MLH can be viewed as a fragment of 1SL2. Any formula $\psi_1 * \psi_2$ [resp. $\psi_1 \mathrel{-\!\!*} \psi_2$] in $t(\phi, \mathrm{u}_1)$ has at most one free variable. Note that because MLH has models with a deterministic relation, its restriction without separating connectives is strongly related to Deterministic PDL [BAHP82] and to Description Logic with functional roles, see e.g. [CG05].

4. Decidability and Computational Complexity

In this section, we present several results about the decidability status of separation logics and their fragments. Most of the undecidability results are obtained by quite simple reductions (only main ideas are presented below) whereas decidability is obtained by translation into a richer logical formalism, for example among decidable monadic second-order logics. We state a few results about complexity also; these shall be completed by a thorough discussion of expressive power in Section 5.

4.1 *Undecidability results*

We start by addressing the understanding, common to computer scientists, that "separation logic is undecidable"—although upon inspection, the proof does not in fact involve any appeal to separation logic. After, we consider various undecidable restrictions which get more to the heart of what makes separation logic undecidable.

4.1.1 Undecidability of 2SL

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]).

Theorem 4.1. [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 R, checking whether there is a finite structure $(\mathfrak{D}, \mathfrak{R})$ (a finite directed graph) such that $(\mathfrak{D}, \mathfrak{R}) \models \phi$ (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:

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

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 $T(\phi)$ is satisfiable in 2SL with

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

where $t(\cdot)$ is defined below:

- t is homomorphic for Boolean connectives.
- $t(\mathrm{u}_i = \mathrm{u}_j) \stackrel{\text{def}}{=} (\mathrm{u}_i = \mathrm{u}_j) \wedge (\mathrm{u}_i \hookrightarrow \mathrm{nil}, \mathrm{nil}) \wedge (\mathrm{u}_j \hookrightarrow \mathrm{nil}, \mathrm{nil})$,
- $t(\mathrm{R}(\mathrm{u}_i, \mathrm{u}_j)) \stackrel{\text{def}}{=} (\mathrm{u}_i \hookrightarrow \mathrm{nil}, \mathrm{nil}) \wedge (\mathrm{u}_j \hookrightarrow \mathrm{nil}, \mathrm{nil}) \wedge (\exists \mathrm{u} \ (\mathrm{u} \hookrightarrow \mathrm{u}_i, \mathrm{u}_j))$,
- $t(\exists \ \mathrm{u} \ \psi) \stackrel{\text{def}}{=} \exists \mathrm{u} \ (\mathrm{u} \hookrightarrow \mathrm{nil}, \mathrm{nil}) \wedge t(\psi)$,
- $t(\forall \ \mathrm{u} \ \psi) \stackrel{\text{def}}{=} \forall \ \mathrm{u} \ (\mathrm{u} \ \hookrightarrow \mathrm{nil}, \mathrm{nil}) \Rightarrow t(\psi)$.

Observe that nil is understood as a distinguished variable whose interpretation is not in the heap domain. It is also worth noting that $T(\phi)$ 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 4.2. [COY01] The set of valid formulae for 2SL is not recursively enumerable.

As a consequence, 2SL is not finitely axiomatizable. Indeed, ϕ is finitely valid iff $\forall \mathrm{u}, \mathrm{nil}\ ((\mathrm{u} \hookrightarrow \mathrm{nil}, \mathrm{nil}) \wedge (\neg \exists \mathrm{u}', \mathrm{u}''\ \mathrm{nil} \hookrightarrow \mathrm{u}', \mathrm{u}'')) \Rightarrow t(\phi)$ 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 4.2. 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]). Note also that the result applies to any kSL since 2SL can be viewed then as a syntactic fragment of kSL as soon as $k \geq 2$. Moreover, it is worth mentioning that the authors of the proof system in [LP14] have acknowledged in their POPL'14 presentation that their system is actually unsound for Reynolds' semantics –we believe it might be worth mentioning such a fact to avoid a similar situation in the future.

In Section 5.2, we are able to show a similar result with 1SL by using directly first-order theory of natural numbers with addition and multiplication.

4.1.2 *Undecidability of 1SL*

We have seen that the satisfiability problem for 2SL is undecidable [COY01] with a proof that does not require separating connectives and uses [Tra63] in an essential way; a stronger statement is made in [BDL12] by showing that the satisfiability problem for 1SL is undecidable as well. This is a consequence of the expressive equivalence between 1SL and weak second-order logic 1WSOL on memory states with a single record field.

Theorem 4.3. [BDL12] The satisfiability problem for 1SL is undecidable.

More recently, 1SL restricted to two variables (1SL2) has also been shown undecidable [DD15] (via a reduction from the halting problem for Minsky machines, briefly recalled below) but without touching the central question of expressive completeness, which is the purpose of [DD14]. By way of comparison with [GOR99, IRR$^+$04] and as discussed in Section 3.1, undecidability of 1SL2 cannot be derived from [GOR99, IRR$^+$04] since in 1SL models, we deal with a single functional binary relation, namely the finite heap. By contrast, FO2 is known to admit a decidable and NEXPTIME-complete satisfiability problem, see e.g. [GKV97].

4.1.3 *Propositional separation logics*

Whereas the satisfiability problem for any propositional fragment kSL0 is decidable and indeed PSPACE-complete (see Section 4.3.2), propositional versions of abstract separation logic with propositional variables are easily shown undecidable.

Theorem 4.4. [BK10, LWG10] The satisfiability problems for SL($\mathcal{P}_{\mathrm{fin}}(\mathbb{N}), \uplus, \{\emptyset\}$) 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 SL($\mathcal{P}_{\mathrm{fin}}(\mathbb{N}), \uplus, \{\emptyset\}$) by simple semantical arguments (and without using any proof-theoretical arguments, unlike what is done in [BK10, LWG10]). As far as we know, such a slightly alternative proof has not been published elsewhere.

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

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

(3) $\neg\phi$ is not satisfiable in SL($\mathfrak{H}\mathfrak{S}_k, \uplus, \mathfrak{U}_k$) (for any $k \geq 1$).
(4) ϕ is valid in 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, 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 SL($\mathcal{P}_{\rm fin}(\mathbb{N}), \uplus, \{\emptyset\}$) can be understood as the logic SL($\mathfrak{H}\mathfrak{S}_k, \uplus, \mathfrak{U}_k$) with k equal to zero.

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

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

Machine 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]. We build a formula ϕ_{M} such that M halts iff ϕ_{M} is valid in SL($\mathcal{P}_{\rm fin}(\mathbb{N}), \uplus, \{\emptyset\}$), which entails the undecidability of the satisfiability problem for SL($\mathcal{P}_{\rm fin}(\mathbb{N}), \uplus, \{\emptyset\}$). The formula ϕ_{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 M is encoded by some set $X \in \mathcal{P}_{\rm fin}(\mathbb{N})$ 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 instruction counter, first counter and second counter),
- $\mathrm{card}(X_0) = I$, $\mathrm{card}(X_1) = c_1$ and $\mathrm{card}(X_2) = c_2$,
- 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 formula ϕ_{M} has the following form:

$$((\mathrm{emp} \wedge p_1 \wedge p_2 \wedge \neg q \wedge \neg q') \wedge \mathrm{closure}) \Rightarrow (\top \overset{\neg}{\ast} (q' \wedge (p_1 \ast p_2 \ast (\mathrm{size} = \alpha \wedge q))))$$

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

The formula $\top \overset{\neg}{\ast} (q' \wedge (p_1 \ast p_2 \ast (\mathrm{size} = \alpha \wedge q)))$ states that there is $X \in \mathcal{P}_{\rm fin}(\mathbb{N})$ such that $X = X_0 \uplus X_1 \uplus X_2$, $\mathrm{card}(X_0) = \alpha$, X_1 encodes the first counter and X_2 encodes the second counter.

Let $\langle \mathrm{U} \rangle\ \psi$ be an abbreviation for $\top \overset{\neg}{\ast}\ \psi$ and $[\mathrm{U}]\ \psi$ be an abbreviation for $\top \mathrel{-\!\!\ast} \psi$, following an obvious analogy with the universal modality in Kripke models, see e.g. [GP92, Hem96]. The formula closure is defined as the conjunction of the following formulae:

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

In the sequel, the modalities $\langle \mathrm{U} \rangle$ and $[\mathrm{U}]$ are used at the outermost level only and therefore they are evaluated only on the empty set. More generally, the universal modality

[U] can be defined as follows:

$$[\mathrm{U}]\phi \stackrel{\text{def}}{=} (\mathrm{emp} \wedge (\top \mathrel{-\!\!*} \phi)) * \top$$

Consequently, whenever $X \models_{\mathfrak{V}} (q \wedge \mathrm{size} = I) * p_1 * p_2$ for some $I \in [1, \alpha]$, there is no $I' \neq I$ such that $X \models_{\mathfrak{V}} (q \wedge \mathrm{size} = I') * p_1 * p_2$. Moreover, there are unique X_0, X_1 and X_2 such that $X = X_0 \uplus X_1 \uplus X_2$, $X_0 \models_{\mathfrak{V}} (q \wedge \mathrm{size} = I)$, $X_1 \models_{\mathfrak{V}} p_1$ and $X_2 \models_{\mathfrak{V}} p_2$. We add to closure the following formulae:

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

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

$$(q \ \wedge \ \mathrm{size} = I) * (((q \ \wedge \ \mathrm{size} = J) * (\mathrm{size} = 1 \wedge p_1)) \overset{\neg}{\mathrel{-\!\!*}}$$

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

- Formulae for instructions of the form I: $\mathrm{c}_2 := \mathrm{c}_2 + 1$; goto J are defined similarly.
- For all instructions of the form I: if $\mathrm{c}_1 = 0$ then goto J_1 else ($\mathrm{c}_1 := \mathrm{c}_1 - 1$; goto J_2), we consider

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

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

$$(((q \ \wedge \ \mathrm{size} = I) * (p_1 \wedge \mathrm{size} = 1)) * ((q \ \wedge \ \mathrm{size} = J_2) \overset{\neg}{\mathrel{-\!\!*}} q')))$$

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

The correctness proof works as follows. Suppose that the machine M halts. This means that for any valuation $\mathfrak{V}$, if $\emptyset \models_{\mathfrak{V}} (\mathrm{emp} \ \wedge \ p_1 \ \wedge \ p_2 \ \wedge \ \neg q \wedge \ \neg q') \wedge \mathrm{closure}$, then there is some $X \in \mathcal{P}_{\mathrm{fin}}(\mathbb{N})$ 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 $X \in \mathcal{P}_{\mathrm{fin}}(\mathbb{N})$ such that $X \models_{\mathfrak{V}} (p_1 * p_2 * (\mathrm{size} = \alpha \wedge q)) \wedge q'$, which is equivalent to $\emptyset \models_{\mathfrak{V}} (\top \overset{\neg}{\mathrel{-\!\!*}} ((p_1 * p_2 * (\mathrm{size} = \alpha \wedge q))) \wedge q')$. Now suppose that the machine 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 $\mathfrak{V}_0$:

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

$$\{X \in \mathcal{P}_{\mathrm{fin}}(\mathbb{N}) : (I, c_1, c_2) \text{ reachable from } (1, 0, 0), \ X = X_0 \uplus X_1 \uplus X_2,$$

$$\mathrm{card}(X_0) = I, X_0 \in \mathcal{P}_{\mathrm{fin}}([1, \alpha - 1]), \mathrm{card}(X_1) = c_1, X_1 \in \mathcal{P}_{\mathrm{fin}}(\{\alpha + 2k + 1 : k \in \mathbb{N}\}),$$

$$\mathrm{card}(X_2) = c_2, X_2 \in \mathcal{P}_{\mathrm{fin}}(\{\alpha + 2k : k \in \mathbb{N}\})\}$$

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 $X \approx_{\mathfrak{V}_0} (I, c_1, c_2)$,
(2) $\emptyset \models_{\mathfrak{V}_0} (\mathrm{emp} \wedge p_1 \wedge p_2 \wedge \neg q \wedge \neg q') \wedge \mathrm{closure}$,
(3) there is no $X \in \mathcal{P}_{\mathrm{fin}}(\mathbb{N})$ such that $X \models_{\mathfrak{V}_0} (p_1 * p_2 * (\mathrm{size} = \alpha \wedge q)) \wedge q'$.

Consequently, ϕ_{M} is not valid in $\mathrm{SL}(\mathcal{P}_{\mathrm{fin}}(\mathbb{N}), \uplus, \{\emptyset\})$. This concludes the proof of Theorem 4.4. It is worth noting that this undecidability proof uses only semantical arguments.

4.1.4 Undecidability for separation logic with data

The logic 1SL($*$) happens to be decidable by reduction into 1MSOL (see forthcoming Section 4.2). Below, we show that 1SL3($*$)$[\mathbb{Z}, =]$, i.e. 1SL3($*$) augmented with data values in which only equality tests are possible is undecidable. Definition of 1SL($*$)$[\mathbb{Z}, =]$ can be found in Section 2.5. 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. This shift does not provide any substantial new insights but this is more coherent with our intention to compare separation logics with modal or temporal logics, in most occasions. Nevertheless, this alternative proof remains an original feature of this document.

Finite data words [Bou02] are ubiquitous structures that include timed words, 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. Formally, 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]. A wealth of specification formalisms for data words (and slight variants) has been introduced stemming from automata (see e.g. [KF94, NSV04, BL10]) to adequate logical languages such as first-order logic [BDM^{+}11, Dav09, SZ12] and temporal logics [Fig10]. In [DD15], a reduction is designed 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 and in particular this shows how data words can be encoded by heaps within 1SL2 (see below).

First, we need to explain how to encode data words by heaps and data maps, which is quite easy. Let $\mathfrak{dw} = (a^1, \mathfrak{v}^1) \cdots (a^L, \mathfrak{v}^L)$ be a data word in $([1, \alpha] \times \mathbb{N})^{+}$. The data word $\mathfrak{dw}$ is encoded by the heap $\mathfrak{h}_{\mathfrak{dw}}$ containing a path $\mathfrak{l}^0 \rightarrow \mathfrak{l}^1 \rightarrow \cdots \rightarrow \mathfrak{l}^L \rightarrow \mathfrak{l}^{L+1}$ where

- $\mathfrak{l}^0$ has no predecessor, $\mathfrak{l}^{L+1} \notin \mathrm{dom}(\mathfrak{h}_{\mathfrak{dw}})$ and $\mathfrak{l}^{L+1}$ has a unique predecessor,
- for every $i \in [1, L]$, $\mathfrak{l}^i$ has a^i predecessors and $a^i - 1$ of them have themselves no predecessors (the remaining predecessor is $\mathfrak{l}^{i-1}$),
- every location in $\mathrm{dom}(\mathfrak{h}_{\mathfrak{dw}})$ is either on that path or points to a location on that path.

In order to encode the data values, we introduce some partial function $\mathfrak{d}_{\mathfrak{dw}} : \mathbb{N} \rightharpoonup \mathbb{Z}$ such that for every $i \in [1, L]$, $\mathfrak{d}_{\mathfrak{dw}}(\mathfrak{l}^i) = \mathfrak{v}^i$.

Such a path from $\mathfrak{l}^1$ to $\mathfrak{l}^L$ is called the *main path*. Other simple encodings are possible but the current one is well-suited for the forthcoming developments. Note also that $\mathfrak{h}_{\mathfrak{dw}}$ and $\mathfrak{d}_{\mathfrak{dw}}$ are not uniquely specified, and in particular we understand them modulo isomorphism.

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) $\text{card}(\text{dom}(\mathfrak{h})) \geq 2$,
(fb2) there is a location reachable from all the locations of $\text{dom}(\mathfrak{h})$ that is not in $\text{dom}(\mathfrak{h})$, it has exactly one predecessor and
(fb3) there are no distinct locations $\mathfrak{l}_1, \mathfrak{l}_2, \mathfrak{l}_3, \mathfrak{l}_4, \mathfrak{l}_5$ such that $\mathfrak{l}_1 \rightarrow \mathfrak{l}_2 \rightarrow \mathfrak{l}_3 \leftarrow \mathfrak{l}_4 \leftarrow \mathfrak{l}_5$ 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 recognize the locations on the main path as those locations with at least one predecessor and in the heap domain. The existence of such a main path is guaranteed by (fb3). As shown in [DD15], there is a formula ϕ_{fb} in 1SL2($*$) such that $\mathfrak{h} \models \phi_{\text{fb}}$ iff $\mathfrak{h}$ is a fishbone.

Let us refine the notion of fishbone heap so that it takes into account constraints on numbers of predecessors. An *α-fishbone* is a fishbone heap such that every location on the main path has a number of predecessors in $[1, \alpha]$ Again, there is a formula $\phi_{\text{fb}}^{\alpha}$ such that $(\mathfrak{s}, \mathfrak{h}) \models \phi_{\text{fb}}^{\alpha}$ iff $\mathfrak{h}$ is an α-fishbone heap. There is also a formula $\text{mp}(\text{u})$ in 1SL2($*$) such that for any α-fishbone heap $\mathfrak{h}$, $(\mathfrak{s}, \mathfrak{h}) \models_{\mathfrak{f}} \text{mp}(\text{u})$ iff $\mathfrak{f}(\text{u})$ is on the main path (i.e., $\mathfrak{f}(\text{u}) \in \{\mathfrak{l}^1, \ldots, \mathfrak{l}^L\}$).

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 $\text{LTL}_{\downarrow}^{\alpha}(\text{F},\text{F}^{-1})$ be the set of formulae defined as follows:

$$\phi ::= a \ \mid \uparrow \mid \downarrow \phi \ \mid \ \neg\phi \ \mid \ \phi \wedge \phi \ \mid \ \text{F}\phi \ \mid \ \text{F}^{-1}\phi$$

with $a \in [1, \alpha]$. The operators F and 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 $\text{LTL}_{\downarrow}^{\alpha}(\text{F},\text{F}^{-1})$ 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 $i \in [1, L]$):

- $\mathfrak{dw}, i \models_{\mathfrak{v}} a \overset{\text{def}}{\Leftrightarrow} a^i = a$,
- $\mathfrak{dw}, i \models_{\mathfrak{v}} \uparrow \overset{\text{def}}{\Leftrightarrow} \mathfrak{v}^i = \mathfrak{v}$,
- $\mathfrak{dw}, i \models_{\mathfrak{v}} \downarrow \phi \overset{\text{def}}{\Leftrightarrow} \mathfrak{dw}, i \models_{\mathfrak{v}^i} \phi$,
- $\mathfrak{dw}, i \models_{\mathfrak{v}} \text{F}\phi \overset{\text{def}}{\Leftrightarrow}$ there is $i' \in [i, L]$ such that $\mathfrak{dw}, i' \models_{\mathfrak{v}^i} \phi$,
- $\mathfrak{dw}, i \models_{\mathfrak{v}} \text{F}^{-1}\phi \overset{\text{def}}{\Leftrightarrow}$ there is $i' \in [1, i]$ such that $\mathfrak{dw}, i' \models_{\mathfrak{v}^i} \phi$.

A sentence is satisfiable $\overset{\text{def}}{\Leftrightarrow}$ there is a data word $\mathfrak{dw}$ in $([1, \alpha] \times \mathbb{N})^+$ such that $\mathfrak{dw}, 1 \models \phi$ (no need to specify a data value since ϕ is closed). The satisfiability problem for $\text{LTL}_{\downarrow}^{\alpha}(\text{F},\text{F}^{-1})$ is known to be undecidable [FS09, Theorem 4].

Let us define $T(\phi)$ as follows:

$$T(\phi) \overset{\text{def}}{=} \phi_{\text{fb}}^{\alpha} \wedge t(\text{u}_0, \phi) \wedge \text{mp}(\text{u}_0) \wedge \neg\exists \ \text{u}_1 \ (\text{u}_1 \hookrightarrow \text{u}_0 \wedge \text{mp}(\text{u}_1))$$

We aim at satisfying that ϕ is satisfiable iff $T(\phi)$ is satisfiable in 1SL3($*$)$[\mathbb{Z}, =]$. The map t takes two arguments: a quantified variable among $\{\text{u}_0, \text{u}_1\}$ (variables are indeed recycled, see e.g. [Gab81]) and a formula. A third variable 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 t as follows ($i \in \{0, 1\}$).

- t is homomorphic for Boolean connectives,
- $t(\mathrm{u}_i, \uparrow) \overset{\text{def}}{=} (\mathrm{u}_2 \sim \mathrm{u}_i)$ and $t(\mathrm{u}_i, a) \overset{\text{def}}{=} (\sharp \mathrm{u}_i = a)$,
- $t(\mathrm{u}_i, \downarrow \psi) \overset{\text{def}}{=} \exists\, \mathrm{u}_2\ ((\mathrm{u}_2 \sim \mathrm{u}_i) \wedge t(\mathrm{u}_i, \psi))$,
- $t(\mathrm{u}_i, \mathrm{F}\psi) \overset{\text{def}}{=} \exists\, \mathrm{u}_{1-i}\ (t(\mathrm{u}_{1-i}, \psi) \wedge \mathrm{mp}(\mathrm{u}_{1-i}) \wedge \mathrm{reach}(\mathrm{u}_i, \mathrm{u}_{1-i}))$,
- $t(\mathrm{u}_i, \mathrm{F}^{-1}\psi) \overset{\text{def}}{=} \exists\, \mathrm{u}_{1-i}\ (t(\mathrm{u}_{1-i}, \psi) \wedge \mathrm{mp}(\mathrm{u}_{1-i}) \wedge \mathrm{reach}(\mathrm{u}_{1-i}, \mathrm{u}_i))$.

We recall that $\mathrm{u}_2 \sim \mathrm{u}_i$ holds true when u_2 and u_i are allocated and have the same data value.

We have already seen how to define the formulae $\sharp \mathrm{u}_i = a$, $\mathrm{reach}(\mathrm{u}_i, \mathrm{u}_{1-i})$, $\mathrm{mp}(\mathrm{u}_{1-i})$ and $\phi^{\alpha}_{\mathrm{fb}}$. It is easy to check that ϕ is satisfiable iff $T(\phi)$ is satisfiable in 1SL3($*$)[$\mathbb{Z}, =$] since the map t only internalizes the semantics of $\mathrm{LTL}^{\alpha}_{\downarrow}(\mathrm{F},\mathrm{F}^{-1})$ in 1SL3($*$)[$\mathbb{Z}, =$].

Theorem 4.5. [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].

4.2 *Decidability results*

Having seen various undecidable fragments of separation logic, we now turn our attention to fragments that admit a decidable satisfiability problem. As one may conjecture from the results above, the magic wand is quite powerful; its removal results in a decidable logic. Alternatively, the magic wand can be retained at the cost of one additional variable. These and other decidability results are discussed below.

4.2.1 Removing the wand

First we show that 1SL($*$) is decidable with non-elementary recursive complexity. Actually, the complexity lower bound holds already when only two quantified variables are used whereas the problem for the full fragment 1SL2 happens to be undecidable too [DD15]. 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 $(\mathfrak{D}, \mathfrak{f}, =)$ where $\mathfrak{D}$ is a countable domain, $\mathfrak{f}$ is a unary function, and $=$ is equality. 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 $t_{\mathrm{P}}(.)$, computable in logarithmic space, such that a 1MSOL sentence ϕ is satisfiable iff

$$\overbrace{(\neg \exists\ \mathrm{P}\ \forall \mathrm{u}\ \mathrm{P}(\mathrm{u}))}^{\text{infinity}} \wedge \exists\ \mathrm{P}\ t_{\mathrm{P}}(\phi)$$

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

$$\exists\ \mathrm{P}\ (\forall \mathrm{u}\ \mathrm{P}(\mathrm{u}) \Leftrightarrow\ (\exists \mathrm{u}'\ \mathrm{u} \hookrightarrow \mathrm{u}')) \wedge t_{\mathrm{P}}(\phi)$$

is satisfiable where $t_{\mathrm{P}}(\cdot)$ is defined with the following clauses:

- $t_{\mathrm{P}}(\mathrm{u} \hookrightarrow \mathrm{u}') \overset{\text{def}}{=} \mathrm{P}(\mathrm{u}) \wedge \mathrm{u} \hookrightarrow \mathrm{u}'$,

- $t_{\mathrm{P}}(\mathrm{u} = \mathrm{u}') \overset{\text{def}}{=} \mathrm{u} = \mathrm{u}'$,
- $t_{\mathrm{P}}(\phi * \psi) \overset{\text{def}}{=} \exists\ \mathrm{Q}, \mathrm{Q}'\ (\mathrm{P} = \mathrm{Q} \uplus \mathrm{Q}') \wedge t_{\mathrm{Q}}(\phi) \wedge t_{\mathrm{Q}'}(\psi)$ where $\mathrm{P} = \mathrm{Q} \uplus \mathrm{Q}'$ is an abbreviation for $\forall \mathrm{u}\ (\mathrm{P}(\mathrm{u}) \Leftrightarrow (\mathrm{Q}(\mathrm{u}) \vee \mathrm{Q}'(\mathrm{u}))) \wedge \neg(\mathrm{Q}(\mathrm{u}) \wedge \mathrm{Q}'(\mathrm{u}))$.

We leave out the Boolean connectives and first-order quantification, for which t_{P} is homomorphic.

Theorem 4.6. [BDL12, Corollary 3.3] The satisfiability problem for 1SL($*$) is decidable.

As conjectured in [BDL08], recently it has been shown that 1MSOL is strictly more expressive than 1SL($*$) [AD09, Corollary 5.3]. 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.

4.2.2 *Removing a variable*

On the other hand, the wand can be retained if one is willing to give up a variable. A recent result [DGLWM14] establishes this and, coupled with the results of [DD15, DD14], we now have a complete understanding of the decidability boundary in fragments of separation logic that restrict the number of variables (see also Figure 4 in Section 7).

Theorem 4.7. [DGLWM14] Satisfiability for 1SL1 is decidable.

4.2.3 *Other fragments*

By taking advantage of the decidability of 1MSOL over memory states with one record field, we can deduce the following results.

Corollary 4.8. [DD15] The satisfiability problem for MLH($*$) is decidable.

Non-elementary complexity of MLH($*$) is also established in [DD15]. So, solving the satisfiability problem for MLH($*$) requires time bounded below by towers of exponentials of height that depend on the formula size, see e.g. [Sch13].

In Section 2.5, 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 $1\mathrm{SL}^{\mathrm{sdc}}_{<}$, which provides a quite strong new decidability result.

Theorem 4.9. [BBL09, Corollary 1] The satisfiability problem for $1\mathrm{SL}^{\mathrm{sdc}}_{<}$ is decidable.

It remains partly 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].

4.3 *Computational complexity*

In this section, we consider the computational complexity of decision problems for separation logics (mainly satisfiability, validity and entailment) and we provide a few characterizations ranging from problems that can be solved in polynomial time to problems that are decidable with a non-elementary complexity. In order to guarantee a reasonably fair variety of fragments we consider propositional fragments as well as logics in which quantification is permitted, although such quantification is often quite restricted.

4.3.1 *A well-known fragment in polynomial time*

Even though performing reasoning in propositional logic kSL0 (with $k \geq 1$) can be computationally expensive, see below 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* and ϕ_s defines *spatial formulae*:

$$\phi_p ::= \perp \mid \top \mid (\mathrm{x}_i = \mathrm{x}_j) \mid \neg(\mathrm{x}_i = \mathrm{x}_j) \mid \phi_p \wedge \phi_p$$

$$\phi_s ::= \mathrm{emp} \mid \top \mid \mathrm{x}_i \mapsto \mathrm{x}_j \mid \mathrm{sreach}(\mathrm{x}_i, \mathrm{x}_j) \mid \phi_s * \phi_s \qquad \phi ::= \phi_p \wedge \phi_s$$

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

The *entailment problem* for SF is defined as follows:

input: two formulae ϕ, ψ in SF.
question: is it the case that for all memory states $(\mathfrak{s}, \mathfrak{h})$, we have $(\mathfrak{s}, \mathfrak{h}) \models \phi$ implies $(\mathfrak{s}, \mathfrak{h}) \models \psi$?

Obviously the validity problem is more general than the entailment problem and such a problem makes sense because formulae in SF are not closed under negation. Nevertheless, it is clear that the entailment can be defined for many other versions of separation logic. Note also that the rule for strengthening precedent (SP)

$$\frac{\phi \Rightarrow \psi' \quad \{\psi'\}\ \mathrm{C}\ \{\psi\}}{\{\phi\}\ \mathrm{C}\ \{\psi\}}$$

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 4.10. [CHO$^+$11, Theorems 16 & 24] 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 untractability. For instance considering the variant clause $\phi ::= \phi_p \wedge \phi_s \wedge \phi'_s$ (i.e., allowing a bit of conjunction) already leads to coNP-hardness [CHO$^+$11]. The algorithm presented in [CHO$^+$11] has been implemented and used for automatic verification, see [HIOP13].

In [PWZ13], an extension of SF is considered, called 1SLL$\mathbb{B}$, in which formulae are Boolean combinations of pure formulae and spatial formulae.

Corollary 4.11. The satisfiability problem for 1SLL$\mathbb{B}$ is NP-complete.

NP-hardness is an obvious consequence that 1SLL$\mathbb{B}$ contains equality constraints and it is closed under Boolean connectives. The NP upper bound follows from the linear-time reduction from 1SLL$\mathbb{B}$ to the logic of graph reachability and stratified sets (GRASS) [PWZ13].

4.3.2 Fragments in polynomial space

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

The proof for the PSPACE upper bound is sketched in Section 5.1 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]. Let $\mathcal{Q}_1\ p_1\ \cdots\ \mathcal{Q}_n\ p_n\ \phi$ be a QBF formula with $\{\mathcal{Q}_1, \ldots, \mathcal{Q}_n\} \subseteq \{\exists, \forall\}$ and ϕ is a propositional formula built over the proposition variables in $\{p_1, \ldots, p_n\}$. In the translation process, we consider n program variables, say $\mathrm{x}_1, \ldots, \mathrm{x}_n$ so that the truth of p_i is encoded by the satisfaction of $\mathrm{alloc}(\mathrm{x}_i)$. In order to encode independence between the different variables, we enforce that all the program variables have distinct values in the original heap (see also below). Moreover, existential quantification over p_i amounts to restrict the current heap either by the empty heap (in that case $\mathrm{alloc}(\mathrm{x}_i)$ holds in the other heap) or by a unique memory cell so that $\mathrm{alloc}(\mathrm{x}_i)$ holds, which allows to simulate quantification. However, it is necessary to enforce in the initial heap that $\mathrm{alloc}(\mathrm{x}_i)$ holds for any program variable x_i. Let us define the map t as follows:

- $t(p_i) \stackrel{\text{def}}{=} \mathrm{alloc}(\mathrm{x}_i)$.
- t is homomorphic for Boolean connectives.
- $t(\exists\ p_i\ \psi) \stackrel{\text{def}}{=} (\mathrm{emp} \vee (\mathrm{alloc}(\mathrm{x}_i) \wedge \neg(\neg\mathrm{emp} * \neg\mathrm{emp})))) * t(\psi)$.
- $t(\forall\ p_i\ \psi) \stackrel{\text{def}}{=} \neg((\mathrm{emp} \vee (\mathrm{alloc}(\mathrm{x}_i) \wedge \neg(\neg\mathrm{emp} * \neg\mathrm{emp})))) * \neg t(\psi))$.

One can check that $\mathcal{Q}_1\ p_1\ \cdots\ \mathcal{Q}_n\ p_n\ \phi$ is QBF satisfiable iff

$$(\bigwedge_{i \neq j} \mathrm{x}_i \neq \mathrm{x}_j) \wedge (\bigwedge_{i} \mathrm{alloc}(\mathrm{x}_i)) \wedge t(\mathcal{Q}_1\ p_1\ \cdots\ \mathcal{Q}_n\ p_n\ \phi)$$

is 1SL0 satisfiable.

The above reduction implies that the satisfiability problem for kSL0($*$) is PSPACE-hard too and other fragments have been considered in [COY01] by restricting further the use of Boolean or separating connectives. For instance, the satisfiability problem for kSL0($-\!*$) is PSPACE-hard too. PSPACE upper bound can be pushed a bit further by allowing a unique quantified variable.

Theorem 4.13. [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]).

4.3.3 *Non-elementary fragments*

Reduction from PITL satisfiability to 1SL2($*$) satisfiability leads to the following lower bound since PITL satisfiability problem is known to have non-elementary complexity (see definition of PITL in Section 3.3).

Theorem 4.14. [DD15] The satisfiability problem for 1SL2($*$) is decidable with non-elementary complexity.

Again, 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. [Sch13].

We have seen that 1SL($*$) is decidable whereas satisfiability for full 1SL is undecidable. However, 1SL($*$) is certainly not the largest decidable fragment of 1SL. Below, we investigate another decidable extension of 1SL($*$) introduced in [BDL12, Section 4] thanks to a restricted use of the magic wand; quantification over disjoint heaps is done only for heaps whose domain has cardinality smaller than some fixed n.

Let us define 1SL($* + \overset{n}{\neg\!\!*}$) as an extension of 1SL($*$) by adding the binary operators $\overset{n}{\neg\!\!*}$ for every $n \in \mathbb{N}$. Unlike $-\!\!*$, a formula with outermost connective $\overset{n}{\neg\!\!*}$ states the existence of a disjoint heap for which the cardinality of the domain is bounded by n. More formally, we require that $(\mathfrak{s}, \mathfrak{h}) \models_{\mathfrak{f}} \phi_1 \overset{n}{\neg\!\!*} \phi_2$ iff there is $\mathfrak{h}' \perp \mathfrak{h}$ such that $\mathrm{card}(\mathrm{dom}(\mathfrak{h}')) \leq n$, $(\mathfrak{s}, \mathfrak{h}') \models_{\mathfrak{f}} \phi_1$ and $(\mathfrak{s}, \mathfrak{h} \uplus \mathfrak{h}') \models_{\mathfrak{f}} \phi_2$.

1SL($* + \overset{n}{\neg\!\!*}$) allows to encode the restricted use of the magic wand in the Hoare-like proof systems as in the backward-reasoning form rule (MUBR) recalled below (see e.g. [Rey08]).

$$\frac{}{\{(\exists z\ x \mapsto z) * ((x \mapsto y) -\!\!* \ \phi)\}\ [x] := y\ \{\phi\}}$$

Note that $(x \mapsto y) -\!\!* \ \phi$ is logically equivalent to $\neg((x \mapsto y) \overset{1}{\neg\!\!*} \neg\phi)$.

Theorem 4.15. [BDL12, Theorem 4.9]

(I) There is a polynomial-time reduction from the 1SL($* + \overset{n}{\neg\!\!*}$) satisfiability problem to the 1SL($*$) satisfiability problem.

(II) 1SL($* + \overset{n}{\neg\!\!*}$) satisfiability problem is decidable.

5. Expressive Power

In this section, we take care of the expressive power for propositional separation logics but also for first-order separation logics (with different types of characterization). 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]). 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.

On non-linear structures, bisimulation invariant fragment of MSO and modal μ-calculus have been shown equivalent [JW96]. For sets of nodes, XPath has been established equally expressive as first-order logic with two quantified variables, see e.g. an overview in [MdR05]. 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].

5.1 *Boolean formulae for propositional separation logics*

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

Theorem 5.1. [Loz04a, Chapter 5] Any formula ϕ in 1SL0 built over the program variables in $\{\mathrm{x}_1, \ldots, \mathrm{x}_q\}$ is logically equivalent to a Boolean combination of atomic formulae among $\mathrm{size} \geq k$, $\mathrm{alloc}(\mathrm{x}_i)$, $\mathrm{x}_i \hookrightarrow \mathrm{x}_j$ and $\mathrm{x}_i = \mathrm{x}_j$ ($k \in \mathbb{N}$, $i, j \in \{1, \ldots, q\}$).

The formulae of the form $\mathrm{size} \geq k$ and $\mathrm{alloc}(\mathrm{x}_i)$ are introduced in Section 2.2 and we recall that $\mathrm{alloc}(\mathrm{x}_i)$ holds when $\mathfrak{s}(\mathrm{x}_i)$ belongs to the heap domain and $\mathrm{size} \geq k$ holds when the cardinal of the heap domain is at least k. By way of example $(\neg\mathrm{emp} * (\mathrm{x}_1 \hookrightarrow \mathrm{x}_2 \mathrel{-\!\!*} \perp))$ is equivalent to $\mathrm{size} \geq 2 \wedge \mathrm{alloc}(\mathrm{x}_1)$. Furthermore, the cardinal of the heap domain without the interpretation of x_1 and x_2 (in the case it belongs to the domain) is at least $k \geq 0$, can be expressed as follows:

$$(\mathrm{alloc}(\mathrm{x}_1) \wedge \mathrm{alloc}(\mathrm{x}_2) \wedge \mathrm{size} \geq k + 2) \vee$$

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

$$(\neg\mathrm{alloc}(\mathrm{x}_1) \wedge \neg\mathrm{alloc}(\mathrm{x}_2) \wedge \mathrm{size} \geq k)$$

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

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

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

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

$$(\neg\mathrm{alloc}(\mathrm{x}_1) \wedge \neg\mathrm{alloc}(\mathrm{x}_2) \wedge \mathrm{size}_{\overline{2}} \geq k)$$

It is clear that such a formula can be generalized to any $q \geq 1$. So using atomic formulae of the form $\mathrm{size} \geq k$ or $\mathrm{size}_{\overline{q}} \geq k$ does not make a difference in terms of expressive power.

Theorem 5.1 can be viewed as a means to eliminate separating connectives $*$ and $\mathrel{-\!\!*}$ and this is analogous to quantifier elimination in Presburger arithmetic for which periodicity constraints need to be introduced in order to eliminate the quantifiers (see e.g. [Coo72]). Similarly, the atomic formulae $\mathrm{size} \geq k$ and $\mathrm{alloc}(\mathrm{x}_i)$ 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.

Even though Theorem 5.1 provides a nice characterization of the expressive power for 1SL0, several features limit its application. First, Theorem 5.1 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 below). Moreover, neither Theorem 5.1 states how to compute the equivalent formula nor it provides a precise information about the maximal bound k in atomic formulae $\mathrm{size} \geq k$ that are used to build a Boolean combination equivalent to ϕ in 1SL0. 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]. 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 5.1 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]. More details can be found in [DGLWM14]. Let $q, \alpha \geq 1$. We write $\mathrm{Test}_{q,\alpha}$ to denote the following set of atomic formulae:

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

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 $\mathrm{Test}_{q,\alpha}$: $(\mathfrak{s}, \mathfrak{h}) \approx_{q,\alpha} (\mathfrak{s}', \mathfrak{h}')$ iff

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

One can show that for any formula ϕ in 1SL0 with $q \geq 1$ program variables and with size $|\phi|$ (for some reasonably succinct encoding), for any $\alpha \geq |\phi|$, if $(\mathfrak{s}, \mathfrak{h}) \approx_{q,\alpha} (\mathfrak{s}', \mathfrak{h}')$, then $(\mathfrak{s}, \mathfrak{h}) \models \phi$ iff $(\mathfrak{s}', \mathfrak{h}') \models \phi$. 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 $\mathrm{Test}_{q,|\phi|}$. Theorem 5.1 is then a direct consequence of this property.

Corollary 5.2. Let ϕ be a satisfiable formula in 1SL0 with q program variables. Then there is memory state $(\mathfrak{s}, \mathfrak{h})$ such that $(\mathfrak{s}, \mathfrak{h}) \models \phi$ and $\mathrm{ran}(\mathfrak{s}) \cup \mathrm{dom}(\mathfrak{h}) \cup \mathrm{ran}(\mathfrak{h}) \subseteq [0, q + |\phi| + 1]$.

Satisfaction of ϕ depends only on the satisfaction of formulae from $\mathrm{Test}_{q,|\phi|}$. So, to check satisfiability status of ϕ, only the truth value of formulae in $\mathrm{Test}_{q,|\phi|}$ 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 $\mathrm{Test}_{q,|\phi|}$), whence the introduction of symbolic memory states. A *symbolic memory state* $\mathfrak{S}$ over (q, α) is a structure $((V, E), \mathfrak{n})$ such that:

(1) There is a partition P of $\{\mathrm{x}_1, \ldots, \mathrm{x}_q\}$ such that $P \subseteq V$. This encodes the store.
(2) (V, E) is a functional directed graph and a node v in (V, E) is at distance at most one of some set of variables X in P.

```
1: if $\psi$ is atomic then return AMC($\mathfrak{S}, \psi$);
2: if $\psi = \neg\psi_1$ then return not MC($\mathfrak{S}, \psi_1$);
3: if $\psi = \psi_1 \wedge \psi_2$ then return (MC($\mathfrak{S}, \psi_1$) and MC($\mathfrak{S}, \psi_2$));
4: if $\psi = \psi_1 * \psi_2$ then return $\top$ iff there are $\mathfrak{S}_1$ and $\mathfrak{S}_2$ such that $*_a(\mathfrak{S}, \mathfrak{S}_1, \mathfrak{S}_2)$
   and MC($\mathfrak{S}_1, \psi_1$) = MC($\mathfrak{S}_2, \psi_2$) = $\top$;
5: if $\psi = \psi_1 -\!\!* \psi_2$ then return $\perp$ iff for some $\mathfrak{S}'$ and $\mathfrak{S}''$ such that
   $*_a(\mathfrak{S}'', \mathfrak{S}', \mathfrak{S})$, MC($\mathfrak{S}', \psi_1$) = $\top$ and MC($\mathfrak{S}'', \psi_2$) = $\perp$;
```

Figure 2. Function MC($\mathfrak{S}, \psi$)

```
1: if $\psi$ is emp then return $\top$ iff $E = \emptyset$ and $\mathfrak{n} = 0$;
2: if $\psi$ is $x_i = x_j$ then return $\top$ iff $x_i, x_j \in X$, for some $X \in P$;
3: if $\psi$ is $x_i \hookrightarrow x_j$ then return $\top$ iff $(X, X') \in E$ where $x_i \in X \in P$ and
   $x_j \in X' \in P$;
```

Figure 3. Function AMC($\mathfrak{S}, \psi$)

(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 $\{x_1, \ldots, x_q\}$.

Given $q, \alpha \geq 1$, the number of symbolic memory states over (q, α) is exponential in $q + \alpha$. Given a memory state $(\mathfrak{s}, \mathfrak{h})$, we define its *abstraction* $\mathrm{Symb}[\mathfrak{s}, \mathfrak{h}]$ over (q, α) as the symbolic memory state $((V, E), \mathfrak{n})$ such that

- $\mathfrak{n} = min(\alpha, \mathrm{card}(\mathrm{dom}(\mathfrak{h}) \setminus \{\mathfrak{s}(x_i) : i \in [1, q]\}))$.
- P is a partition of $\{x_1, \ldots, x_q\}$ so that for all x, x', we have $\mathfrak{s}(x) = \mathfrak{s}(x')$ iff x and x′ belong to the same set in P.
- V is made of elements from P as well as of locations from the set below:

$$\{\mathfrak{h}(\mathfrak{s}(x_i)) : \mathfrak{s}(x_i) \in \mathrm{dom}(\mathfrak{h}), i \in [1, q]\} \setminus \{\mathfrak{s}(x_i) : i \in [1, q]\}$$

- The graph (V, E) is defined as follows:
  (1) $(X, X') \in E$ if $X, X' \in P$ and $\mathfrak{h}(\mathfrak{s}(x)) = \mathfrak{s}(x')$ for some $x \in X, x' \in X'$.
  (2) $(X, \mathfrak{l}) \in E$ if $X \in P$ and $\mathfrak{h}(\mathfrak{s}(x)) = \mathfrak{l}$ for some variable x in X and $\mathfrak{l} \notin \{\mathfrak{s}(x_i) : i \in [1, q]\}$.

We define symbolic memory states to be *isomorphic* if (1) the partition P is identical, (2) the finite digraphs agrees on the atomic formulae of the form either $x_i \hookrightarrow x_j$ or $\mathrm{alloc}(x_i)$, and (3) the numerical values are identical. Note that given a symbolic memory state $\mathfrak{S}$ over (q, α), there exists a memory state $(\mathfrak{s}, \mathfrak{h})$ such that $\mathrm{Symb}[\mathfrak{s}, \mathfrak{h}]$ and $\mathfrak{S}$ are isomorphic. Not only every symbolic memory state has always a concretization 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 $\mathrm{Symb}[\mathfrak{s}, \mathfrak{h}]$ and $\mathrm{Symb}[\mathfrak{s}', \mathfrak{h}']$ are isomorphic symbolic memory states.

Definition 5.1. Given symbolic memory states $\mathfrak{S}$, $\mathfrak{S}_1$ and $\mathfrak{S}_2$, we write $*_a(\mathfrak{S}, \mathfrak{S}_1, \mathfrak{S}_2)$ if there exist a store $\mathfrak{s}$ and disjoint heaps $\mathfrak{h}_1$ and $\mathfrak{h}_2$ such that $\mathrm{Symb}[\mathfrak{s}, \mathfrak{h}_1 \uplus \mathfrak{h}_2] = \mathfrak{S}$, $\mathrm{Symb}[\mathfrak{s}, \mathfrak{h}_1] = \mathfrak{S}_1$ and $\mathrm{Symb}[\mathfrak{s}, \mathfrak{h}_2] = \mathfrak{S}_2$. ∇

Given $q, \alpha \geq 1$, the ternary relation $*_a$ can be decided in polynomial time in $q + log(\alpha)$ for all the symbolic memory states built over (q, α).

Figure 2 presents a procedure MC($\mathfrak{S}, \psi$) returning a Boolean value in $\{\perp, \top\}$ and taking as arguments, a symbolic memory state over (q, α) and a formula ψ with $|\psi| \leq \alpha$. 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($\mathfrak{S}, \psi$) also returns a Boolean value in $\{\perp, \top\}$, makes no recursive calls and is dedicated to atomic formulae (see Figure 3). The design of MC is similar to nondeterministic polynomial-space procedures, see e.g. [Lad77, Spa93, COY01].

Lemma 5.3. Let $q, \alpha \geq 1$, $\mathfrak{S}$ be a symbolic memory state over (q, α) and ϕ be in 1SL0 built over $\mathrm{x}_1, \ldots, \mathrm{x}_q$ such that $|\phi| \leq \alpha$. We have MC($\mathfrak{S}, \phi$) returns $\top$ iff there exists $(\mathfrak{s}, \mathfrak{h})$ such that $\mathrm{Symb}[\mathfrak{s}, \mathfrak{h}] = \mathfrak{S}$ and $(\mathfrak{s}, \mathfrak{h}) \models \phi$.

Consequently, we get the following complexity characterization

Theorem 5.4. Model-checking and satisfiability problems for 1SL0 are PSPACE-complete.

PSPACE-hardness is due of [COY01] and it is briefly explained in Section 4.3.2. PSPACE upper bound is a consequence of Lemma 5.3 by recalling that ϕ is satisfiable iff there is a memory state $(\mathfrak{s}, \mathfrak{h})$ such that $(\mathfrak{s}, \mathfrak{h}) \models \phi$ iff there is an symbolic memory state $\mathfrak{S}$ over $(q, |\phi|)$ such that MC($\mathfrak{S}, \phi$) = $\top$. It is sufficient to guess $\mathfrak{S}$ and then check whether MC($\mathfrak{S}, \phi$) returns $\top$. All of this can be done in nondeterministic polynomial-space and by Savitch's Theorem [Sav70], we get the PSPACE upper bound.

Corollary 5.5. Given a formula ϕ in 1SL0, computing a Boolean combination of atomic formulae from $\mathrm{Test}_{q,|\phi|}$ logically equivalent to ϕ can be done in polynomial space.

5.2 *Results for first-order separation logics*

Recent results have provided some insight into the power of separation logic's second-order features. [BDL12] demonstrated expressive equivalence of 1DSOL and 1WSOL, and showed a translation of 1WSOL into 1SL($-\!\!*$). Recently, [DD14] sharpens this result by showing that a translation from 1DSOL into the two-variable restriction 1SL2($-\!\!*$) is possible. The proofs in both of these works are constructive.

Theorem 5.6. [DD14, BDL12] There is a translation T from 1DSOL to 1SL2($-\!\!*$) such that for every sentence ϕ in 1DSOL and for every memory state $(\mathfrak{s}, \mathfrak{h})$, we have $(\mathfrak{s}, \mathfrak{h}) \models \phi$ in 1DSOL iff $(\mathfrak{s}, \mathfrak{h}) \models T(\phi)$ in 1SL2($-\!\!*$).

These translations can accommodate both program variables and additional record fields (see discussion in [DD14, Section 5.5]). Note that the reverse is also true, as shown earlier by Theorem 3.2. Since there exist translations in both directions between 1DSOL and 1SL, we have the following result.

Theorem 5.7. [DD14, BDL12] 1SL, 1SL2($-\!\!*$), 1DSOL, and 1WSOL have the same expressive power.

Such a comparison of equivalence of expressive power is commonly sought with two-variable logics. There are a number of results comparing the expressive power of FO2 to other logics. Some examples include equivalence between unary LTL and FO2, see e.g. [EVW97, Wei11]. Boolean modal logic with converse and identity is as expressive as FO2 [LSW01] (see also [HK14]).

Note also that, as shown in [Loz12], Theorem 5.7 is a key argument to show that separation logic is complete for sequential mono-procedural programs, when separation logic is understood as a programming language, an assertion language and a set of rules involving Hoare triples. Completeness means that every valid Hoare triple (see Section 2.1.1) is derivable in a given proof system dealing with Hoare triples. Obviously, completeness depends on the programming language, on the assertion logic and on the set of inference rules. More details can be found in [Loz12], see also related results in [COY07, TCA09] or in [TC14].

Corollary 5.8. 1SL3($-\!\!*$) without the equality predicate is as expressive as 1SL2($-\!\!*$).

This is a consequence of the material presented at the end of Section 2.2 where $\mathrm{u} = \overline{\mathrm{u}}$ can be shown equivalent to the formula below (u' is a new quantified variable):

$$\forall\, \mathrm{u}' \;((\mathrm{u}' \hookrightarrow \mathrm{u}) -\!\!*(\mathrm{u}' \hookrightarrow \overline{\mathrm{u}}))$$

Note also that the satisfiability problem for 1SL2 without program variables is shown undecidable in [DD15]. This can be refined even further by banishing the equality predicate in that fragment. Indeed, $\mathrm{u} = \overline{\mathrm{u}}$ can be shown equivalent to the formula below:

$$\top -\!\!* \neg(((\exists\, \overline{\mathrm{u}}\; (\mathrm{u} \hookrightarrow \overline{\mathrm{u}})) \wedge \neg(\exists\, \mathrm{u}\; (\overline{\mathrm{u}} \hookrightarrow \mathrm{u}))) * \top)$$

This follows a similar idea developed in [Loz04a, Section 5.1.1]. By contrast, the decidability status of 1SL2($-\!\!*$) without program variables and without equality is open. Similarly, the decidability status of 1SL($-\!\!*$) where $\hookrightarrow$ is replaced by $\mapsto$ (exact points-to relation) is also open.

Using similar ideas, one can show that any formula in 1SL1 can be transformed into an equivalent formula in 1SL1 but without the equality predicate. Indeed, $\mathrm{x}_i = \mathrm{x}_j$ is logically equivalent to $\forall\, \mathrm{u}\; ((\mathrm{u} \hookrightarrow \mathrm{x}_i) -\!\!*(\mathrm{u} \hookrightarrow \mathrm{x}_j))$ and $\mathrm{x}_i = \mathrm{u}$ (in 1SL1, one quantified variable is allowed) can be eliminated by making a case analysis when first-order quantification is performed on u.

Since 1SL2($-\!\!*$) and 1WSOL have the same expressive power, this implies the following impossibility result.

Theorem 5.9. The set of valid formulae for 1SL is not recursively enumerable.

As a consequence, 1SL is not finitely axiomatizable and this refines Theorem 5.9. A quick argument for proving Theorem 5.9 consists in noting that second-order logic is not finitely axiomatizable and 1SL 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 5.9, it is nevertheless sufficient to combine 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 t_1 such that for any formula ϕ from first-order arithmetic, ϕ is valid iff $t_1(\phi)$ 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. By way of example, the atomic formula $\mathrm{u}_1 \times \mathrm{u}_2 = \mathrm{u}_3$ amounts to check whether the product set made of the interpretation of the monadic second-order variables P_1 and P_2 has the same cardinality as the interpretation of the monadic second-order variable P_3. Obviously, this assumes that each variable u_i has a unique corresponding monadic second-order variable P_i. So the formula $\mathrm{u}_1 \times \mathrm{u}_2 = \mathrm{u}_3$ can be encoded

by:

$$\exists \mathrm{P}\ \mathrm{PRODUCT}(\mathrm{P}, \mathrm{P}_1 \times \mathrm{P}_2) \wedge \mathrm{EQCARD}(\mathrm{P}, \mathrm{P}_3)$$

where $\mathrm{PRODUCT}(\mathrm{P}, \mathrm{P}_1 \times \mathrm{P}_2) \stackrel{\text{def}}{=} \forall\ \mathrm{u}, \mathrm{u}'\ \mathrm{P}(\mathrm{u}, \mathrm{u}') \Leftrightarrow \mathrm{P}_1(\mathrm{u}) \wedge \mathrm{P}_2(\mathrm{u}')$. The formula $\mathrm{EQCARD}(\mathrm{P}, \mathrm{P}_3)$ 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.

- As shown in [DD14], there is a logarithmic-space reduction t_2 such that for any formula ϕ from 1WSOL, ϕ is valid iff $t_2(\phi)$ is valid in 1SL2($-\!\!*$). Actually, it is sufficient to show the result for 1DSOL and then this can be extended to 1WSOL by encoding n-ary finite relations with n finite binary relations. For example, $\mathrm{P}(\mathrm{e}_1, \ldots, \mathrm{e}_n)$ can be encoded by the conjunction $\exists\ \mathrm{v}\ \bigwedge_{i=1}^{n} \mathrm{P}_i(\mathrm{e}_i, \mathrm{v})$ where $\mathrm{P}_1, \ldots,$ P_n are new binary second-order variables attached uniquely to the n-ary second-order variable P, see e.g. [BDL12, Proposition 2.6].

5.2.0.1 Allowing infinite domains. Let $k\mathrm{SL}^{\infty}$ be the variant of kSL in which the heap domain can be either finite or infinite (in kSL, the heap domain is necessarily finite). It is easy to show that the set of valid formulae for $k\mathrm{SL}^{\infty}$ without separating connectives is recursively enumerable, which contrasts with Theorem 4.2. Indeed, the heap can be encoded by a $(k+1)$-ary relation that is functional on its first argument (no more cardinality constraint). By contrast, $1\mathrm{SL}^{\infty}$ (with separating connectives) does not admit a recursively enumerable set of valid formulae (and this holds for any $k\mathrm{SL}^{\infty}$ with $k \geq 1$) since finiteness of the heap domain can be expressed in $1\mathrm{SL}^{\infty}$ itself. 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. There is a simple formula seg in 1SL2 that characterizes segmented heaps, see e.g. [DD14]. Now, it is easy to show that infinite heaps can be characterized by the formula $\mathrm{seg} \stackrel{\neg}{*} \forall\ \mathrm{u}\ \mathrm{alloc}(\mathrm{u})$.

5.3 *1SL(∗) versus 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].

Theorem 5.10. [AD09, Ant10] 1SL($*$) is strictly less expressive than 1MSOL.

In [AD09, Ant10], using EF-games, it is shown that there is no formula in 1SL($*$) that characterizes 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, see e.g. Section 4.2. It is sufficient to state that every location has at most two predecessors and every location $\mathfrak{l}$ can reach a non-allocated location (the root of the tree to which $\mathfrak{l}$ belongs too if $\mathfrak{l}$ is in the domain of the heap):

$$\forall \, \mathrm{u} \, (\sharp \mathrm{u} \leq 2 \wedge \exists \, \overline{\mathrm{u}} \, (\mathrm{reach}(\mathrm{u}, \overline{\mathrm{u}}) \wedge \neg \mathrm{alloc}(\overline{\mathrm{u}})))$$

Note that the quantification above is fine even when u is not in the heap domain (take the value for u to witness the satisfaction of $\exists \, \overline{\mathrm{u}} \, (\mathrm{reach}(\mathrm{u}, \overline{\mathrm{u}}) \wedge \neg \mathrm{alloc}(\overline{\mathrm{u}}))$). 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 P), we label each location of the tree by either P_0, P_1 and 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 P_0. The formula defined below assumes that the heap is already known as a forest of binary trees.

$$\exists \;\; \mathrm{u} \, (\neg \mathrm{alloc}(\mathrm{u}) \wedge \sharp \mathrm{u} \geq 1) \, \wedge$$

$$\exists \;\; \mathrm{P}, \mathrm{P}_0, \mathrm{P}_1, \mathrm{P}_2 \; ((\forall \, \overline{\mathrm{u}} \, \mathrm{P}(\overline{\mathrm{u}}) \Leftrightarrow \mathrm{reach}(\overline{\mathrm{u}}, \mathrm{u})) \wedge (\mathrm{P} = \mathrm{P}_0 \uplus \mathrm{P}_1 \uplus \mathrm{P}_2) \;\wedge\; \mathrm{P}_0(\mathrm{u}) \, \wedge$$

$$(\forall \, \overline{\mathrm{u}} \, (\mathrm{P}(\overline{\mathrm{u}}) \wedge \sharp \overline{\mathrm{u}} = 0) \Rightarrow \mathrm{P}_1(\overline{\mathrm{u}})) \wedge$$

$$(\forall \, \overline{\mathrm{u}}, \overline{\mathrm{u}}' \; ((\overline{\mathrm{u}} \hookrightarrow \overline{\mathrm{u}}') \wedge \mathrm{P}(\overline{\mathrm{u}}) \wedge \sharp \overline{\mathrm{u}}' = 1) \Rightarrow \bigwedge_{i=0}^{2} (\mathrm{P}_i(\overline{\mathrm{u}}) \Leftrightarrow \mathrm{P}_i(\overline{\mathrm{u}}'))) \wedge$$

$$\bigwedge_{i,j,k \in \{0,1,2\}, i \equiv_3 j+k} (\forall \, \overline{\mathrm{u}}, \overline{\mathrm{u}}', \overline{\mathrm{u}}'' (\mathrm{P}_j(\overline{\mathrm{u}}') \wedge \mathrm{P}_k(\overline{\mathrm{u}}'') \wedge (\overline{\mathrm{u}}' \neq \overline{\mathrm{u}}'') \wedge (\overline{\mathrm{u}}' \hookrightarrow \overline{\mathrm{u}}) \wedge (\overline{\mathrm{u}}'' \hookrightarrow \overline{\mathrm{u}})) \Rightarrow \mathrm{P}_i(\overline{\mathrm{u}})))$$

Note that we used a shortcut formula $(\mathrm{P} = \mathrm{P}_0 \uplus \mathrm{P}_1 \uplus \mathrm{P}_2)$ to state that the interpretation of P is the disjoint union of the interpretation of P_0, P_1 and P_2, details are omitted.

6. A Discussion about Decision Procedures and Calculi

In this section, mainly, we provide pointers about proof systems for separation logics, from sequent-style calculi to decision procedures built on SMT solvers.

6.1 *Analytic 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]. 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 4.2). 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]). It is also possible to design Ladner-like algorithm for kSL0 as witnessed by the works in [COY01, DGLWM14], see also Section 5.1.

6.2 *Satisfiability Modulo Theories for separation logics*

6.2.0.2 Direct approach versus translation for modal logics. In order to mechanize modal logics, there exist at least two main approaches with well-identified motivations. The direct approach consists in building specialized 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. 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. 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 mechanize 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.

6.2.0.3 Translation versus specialized algorithms for separation logic. Despite its young age, one can observe that the mechanization 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$^+$11]. Many decision procedures have been designed for fragments of separation logics or abstract variants, from analytic methods [GM10, HCGT14] to translation to theories handled by SMT solvers [PWZ13], passing via graph-based algorithms [HIOP13]. The translation approach has been already advocated in [CGH05] in which propositional kSL0 is translated into a fragment of classical logic that can be decided in polynomial space. However, the framework of satisfiability modulo theories (SMT) remains probably the most promising one to develop decision procedures dedicated to reasoning tasks for separation logics, see e.g [BPS09, RBHC07, PWZ13, DGLWM14]. It is worth noting that verification of programs that manipulate linked-list structures can be also done with a SAT solver, when the assertions are written in some restricted logical formalism, see a remarkable example in [IBI$^+$13].

6.2.0.4 SMT solvers, program verification, and separation logic. 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). A nice feature of such solvers is their ability to combine distinct theories allowing to express richer statements. As advocated in [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 1SLL$\mathbb{B}$ (see also Section 4.3.1) 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. Some strongly related work can be also found in [PR13]. 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 separation logic that have been considered (roughly, symbolic heaps with recursive definitions, see also Sections 4.3.1 and 3.2).

7. Conclusion

In this paper, we have presented a wide range of results about separation logic as far as decidability, complexity (see e.g., Section 4), and expressive power are concerned while pinpointing several relationships with modal or temporal logics (see Section 3). Figure 4 summarizes some of these results.

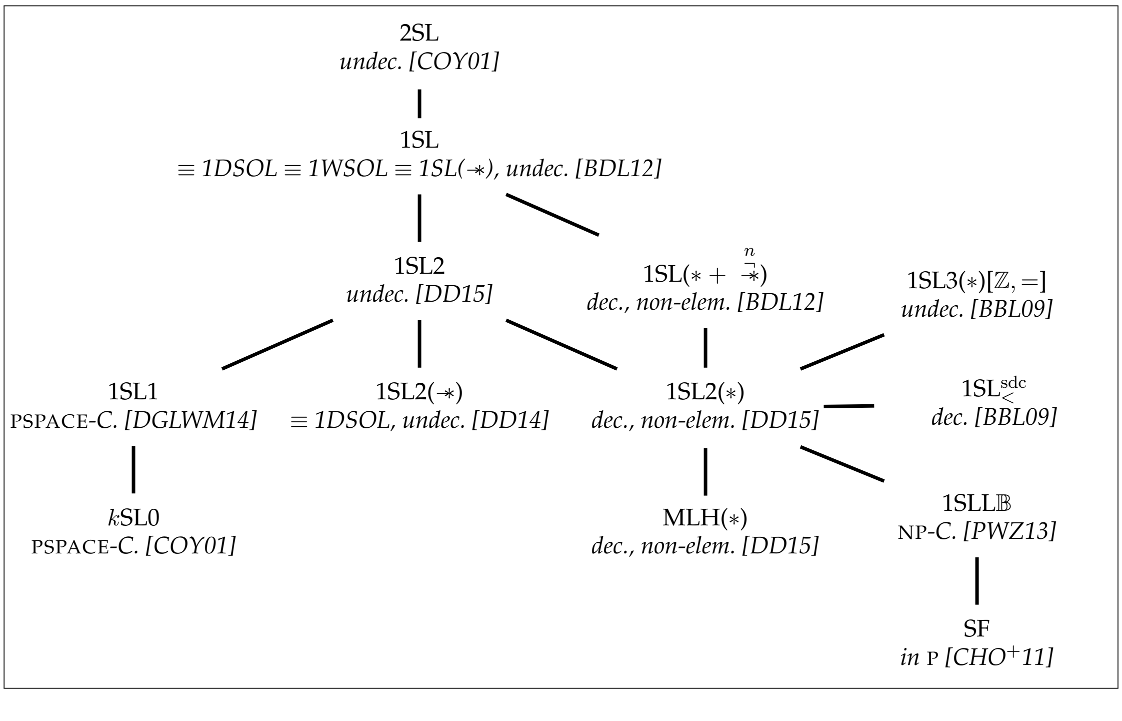

Figure 4. A few results about decidability and complexity

Like the most popular classes of non-classical logics in computer science such as description logic (see e.g. [BCM+03]) or temporal logic (see e.g. [GHR94]), separation logic encompasses a large set of different logical formalisms. More importantly, the early works on separation logic or on BI, see e.g. [IO01, Rey02, POY04], have been the source of many theoretical works and it led to the development of many tools, see e.g. [BCO05, CDOY09, CD11, SC14]. This is a remarkable example of logical formalism that has been developed so quickly in both directions, in a so short amount of time. Even though it is not reasonable to expect to present all the research directions about separation logic within a single paper (or even only its main theoretical results), the research has been essentially driven towards the improvement of automatic program analysis, extending significantly Hoare logic [Hoa69]. That is why promising research directions include, for instance, a more extensive use of fine-tuned SMT solvers (see e.g. [PWZ13]), the development of methods for general inductive predicates (see e.g. [AGH^{+}14]) or the design of generic proof systems (see e.g. [BV14]). In some other respect, building new bridges between separation logics and, modal and temporal logics can certainly be beneficial to transfer known results or proof techniques from modal logic (see e.g. [BV14]).

7.0.0.5 Acknowledgments. This survey has greatly benefited from discussions and exchanges with many colleagues that help us to clarify numerous questions while writing this paper. We would like to thank Timos Antonopoulos (Oxford University), Kshitij Bansal (New York University), Anuj Dawar (Cambridge University), Didier Galmiche (LORIA, Nancy), Rajeev Goré (Australian National University), Christoph Haase (LSV, Cachan), Lauri Hella (University of Tampere), Zhe Hou (Australian National University), Dominique Larchey-Wendling (LORIA, Nancy), Etienne Lozes (LSV, Cachan), Daniel Mery (LORIA, Nancy), Ben Moszkowski (De Montfort University), Alwen Tiu (Nanyang Technological University), Heribert Vollmer (Leibniz Universität, Hannover), Thomas Wies (New York University), Zhilin Wu (LIAFA, Paris).

We would like also to thank two anonymous reviewers for their suggestions and remarks that help us to improve the quality of the manuscript.

References

[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.

[ABM01] C. Areces, P. Blackburn, and M. Marx. Hybrid logics: characterization, interpolation and complexity. *The Journal of Symbolic Logic*, 66(3):977–1010, 2001.

[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.

[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.

[All83] J. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.

[Ant10] T. Antonopoulos. *Expressive Power of Query Languages*. PhD thesis, University of Cambridge, 2010.

[AV11] S. Abramsky and J. Väänänen. From IF to BI: a tale of dependence and separation. *CoRR*, abs/1102.1388, 2011.

[BAHP82] M. Ben-Ari, J. Halpern, and A. Pnueli. Deterministic propositional dynamic logic: finite models, complexity, and completeness. *Journal of Computer and System Sciences*, 25:402–417, 1982.

[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.

[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.

[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.

[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.

[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.

[BCOP05] R. Bornat, C. Calcagno, P. O'Hearn, and M. Parkinson. Permission accounting in separation logic. In *POPL'05*, pages 259–270. ACM, 2005.

[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.

[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.

[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.

[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.

[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.

[BDL12] R. Brochenin, S. Demri, and E. Lozes. On the almighty wand. *Information and Computation*, 211:106–137, 2012.

[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.

[BdRV01] P. Blackburn, M. de Rijke, and Y. Venema. *Modal Logic*. Cambridge University Press, 2001.

[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. Article No 25.

[BGG97] E. Börger, E. Grädel, and Y. Gurevich. *The Classical Decision Problem*. Perspectives in Mathematical Logic. Springer, 1997.

[BK10] J. Brotherston and M. Kanovich. Undecidability of propositional separation logic and its neighbours. In *LICS'10*, pages 130–139. IEEE, 2010.

[BK14] J. Brotherston and M. Kanovich. Undecidability of propositional separation logic and its neighbours. *Journal of the Association for Computing Machinery*, 61(2), 2014.

[BL10] M. Bojańczyk and S. Lasota. An extension of data automata that captures XPath. In *LICS'10*, pages 243–252. IEEE, 2010.

[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.

[Bou02] P. Bouyer. A logical characterization of data languages. *Information Processing Letters*, 84(2):75–85, 2002.

[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.

[Bro12] J. Brotherston. Bunched logics displayed. *Studia Logica*, 100(6):1223–1254, 2012.

[Bro13] R. Brochenin. *Separation Logic: Expressiveness, Complexity, Temporal Extension*. PhD thesis, LSV, ENS Cachan, September 2013.

[BT14a] Ph. Balbiani and T. Tinchev. Decidability and computability in PRSPDL. In *AIML'14*, pages 16–33. College Publications, 2014.

[BT14b] C. Barrett and C. Tinelli. Satisfiability Modulo Theories. In E. Clarke, T. Henzinger, and H. Veith, editors, *Handbook of Model Checking*. Springer, 2014. to appear.

[Bur72] R. Burstall. Some techniques for proving correctness of programs which alter data structures. *Machine Intelligence*, 7:23–50, 1972.

[BV14] J. Brotherston and J. Villard. Parametric completeness for separation theories. In *POPL'14*, pages 453–464. ACM, 2014.

[BvBW06] P. Blackburn, J.F. van Benthem, and F. Wolter, editors. *Handbook of modal logic (Vol. 3)*. Elsevier, 2006.

[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.

[CDOY09] C. Calcagno, D. Distefano, P. O'Hearn, and H. Yang. Compositional shape analysis by means of bi-abduction. In *POPL'09*, pages 289–300. ACM, 2009.

[CG00] L. Cardelli and A. Gordon. Anytime, Anywhere: Modal Logics for Mobile Ambients. In *POPL'00*, pages 365–377. ACM, 2000.

[CG05] D. Calvanese and G. De Giacomo. Expressive description logics. In *Description Logics Handbook*, pages 178–218. Cambridge University Press, 2005.

[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.

[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.

[CGP00] E. Clarke, O. Grumberg, and D. Peled. *Model checking*. The MIT Press Books, 2000.

[CGZ07] C. Calcagno, Ph. Gardner, and U. Zarfaty. Context logic as modal logic: Completeness and parametric inexpressivity. In *POPL'07*, pages 123–134. ACM, 2007.

[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.

[CKM14] W. Charatonik, E. Kieroński, and F. Mazowiecki. Decidability of Weak Logics with Deterministic Transitive Closure. In *CSL-LICS'14*. ACM,

2014. Article No 29.

[Coo72] D. Cooper. Theorem proving in arithmetic without multiplication. *Machine Learning*, 7:91–99, 1972.

[Cop02] J. Copeland. The genesis of possible worlds semantics. *Journal of Philosophical Logic*, 31(1):99–137, 2002.

[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.

[COY07] C. Calcagno, P. O'Hearn, and H. Yang. Local action and abstract separation logic. In *LICS'07*, pages 366–378. IEEE, 2007.

[Dav09] C. David. *Analyse de XML avec données non-bornées*. PhD thesis, LIAFA, Université Paris VII, 2009.

[DD07] S. Demri and D. D'Souza. An automata-theoretic approach to constraint LTL. *Information and Computation*, 205(3):380–415, 2007.

[DD14] S. Demri and M. Deters. Expressive completeness of separation logic with two variables and no separating conjunction. In *CSL-LICS'14*. ACM, 2014. Article No 37.

[DD15] S. Demri and M. Deters. Two-variable separation logic and its inner circle. *ACM Transactions on Computational Logic*, 2015. To appear.

[Dem05] S. Demri. A reduction from DLP to PDL. *Journal of Logic and Computation*, 15(5):767–785, 2005.

[DGG04] A. Dawar, Ph. Gardner, and G. Ghelli. Adjunct elimination through games in static ambient logic. In *FSTTCS'04*, volume 3328 of *Lecture Notes in Computer Science*, pages 211–223. Springer, 2004.

[DGG07] A. Dawar, Ph. Gardner, and G. Ghelli. Expressiveness and complexity of graph logic. *Information and Computation*, 205(3):263–310, 2007.

[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.

[DKR04] D. Distefano, J.-P. Katoen, and A. Rensink. Who is pointing when to whom? On the automated verification of linked list structures. In *FSTTCS'04*, volume 3328 of *LNCS*, pages 250–262. Springer, 2004.

[DL09] S. Demri and R. Lazić. LTL with the freeze quantifier and register automata. *ACM Transactions on Computational Logic*, 10(3), 2009.

[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.

[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.

[dR92] M. de Rijke. The modal logic of inequality. *The Journal of Symbolic Logic*, 57(2):566–584, 1992.

[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.

[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.

[Fig10] D. Figueira. *Reasoning on words and trees with data*. PhD thesis, ENS Cachan, December 2010.

[Fin75] K. Fine. Some connections between elementary and modal logic. In *3rd Scandinavian Logic Symposium*, pages 15–31. North Holland, 1975.

[Fit83] M. Fitting. *Proof methods for modal and intuitionistic logics*. D. Reidel Publishing Co., 1983.

[Flo67] R. Floyd. Assigning meanings to programs. *Proceedings of Symposia in Applied Mathematics*, 19:19–32, 1967.

[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.

[Gab81] D. Gabbay. Expressive functional completeness in tense logic. In *Aspects of Philosophical Logic*, pages 91–117. Reidel, 1981.

[GHR94] D. Gabbay, I. Hodkinson, and M. Reynolds. *Temporal Logic: Mathematical Foundations and Computational Aspects*, volume 1. Oxford University Press, 1994.

[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.

[GKWZ03] D. Gabbay, A. Kurucz, F. Wolter, and M. Zakharyaschev. *Many-dimensional modal logics: theory and practice*. Cambridge University Press, 2003.

[GM10] D. Galmiche and D. Méry. Tableaux and resource graphs for separation logic. *Journal of Logic and Computation*, 20(1):189–231, 2010.

[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.

[GOR99] E. Grädel, M. Otto, and E. Rosen. Undecidability results on two-variable logics. *Archive for Mathematical Logic*, 38(4–5):313–354, 1999.

[Gor14] M. Gordon. Hoare Logic. Lecture notes, February 2014. Available at `http://www.cl.cam.ac.uk/~mjcg/HoareLogic/`.

[GP92] V. Goranko and S. Passy. Using the universal modality: gains and questions. *Journal of Logic and Computation*, 2(1):5–30, 1992.

[GZ07] Ph. Gardner and U. Zarfaty. An Introduction to Context Logic. In *WoLLIC'07*, volume 4576 of *Lecture Notes in Computer Science*, pages 189–202. Springer, 2007.

[HC68] G. Hughes and M. Cresswell. *An introduction to modal logic*. Methuen and Co., 1968.

[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.

[Hem96] E. Hemaspaandra. The price of universality. *Notre Dame Journal of Formal Logic*, 37(2):173–203, 1996.

[Her13] A. Herzig. A simple separation logic. In *WoLLIC'13*, volume 8071 of *Lecture Notes in Computer Science*, pages 168–178. Springer, 2013.

[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.

[HK14] L. Hella and A. Kuustisto. One-dimensional fragment of first-order logic. In *AIML'14*, pages 274–293. College Publications, 2014.

[HKT00] D. Harel, D. Kozen, and J. Tiuryn. *Dynamic Logic*. MIT Press, 2000.

[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.

[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.

[Hoa69] C.A.R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, 1969.

[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.

[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.

[IO01] S. Ishtiaq and P. O'Hearn. BI as an assertion language for mutable data structures. In *POPL'01*, pages 14–26. ACM, 2001.

[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.

[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.

[Jen13a] J. Jensen. *Enabling Concise and Modular Specifications in Separation Logic*. PhD thesis, IT University of Copenhagen, 2013.

[Jen13b] J. Jensen. Techniques for model construction in separation logic, September 2013. `http://www.itu.dk/research/pls/wiki/index.php/Jonas_Buhrkal_Jensen`.

[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.

[Kam68] J. Kamp. *Tense Logic and the Theory of Linear Order*. PhD thesis, UCLA, USA, 1968.

[KF94] M. Kaminski and N. Francez. Finite-memory automata. *Theoretical Computer Science*, 134(2):329–363, 1994.

[KMSV14] J. Kontinen, J.-S. Müller, H. Schnoor, and H. Vollmer. Modal independence logic. In *AIML'14*, pages 353–372. College Publications, 2014.

[KR04] V. Kuncak and M. Rinard. On spatial conjunction as second-order logic. Technical Report MIT–CSAIL–TR–2004–067, MIT CSAIL, October 2004.

[Kri59] S. Kripke. A completeness theorem in modal logic. *The Journal of Symbolic Logic*, 24(1):1–14, 1959.

[Lad77] R. Ladner. The computational complexity of provability in systems of modal propositional logic. *SIAM Journal of Computing*, 6(3):467–480, 1977.

[Lew18] C. I. Lewis. *A survey of symbolic logic*. University of California Press, Berkeley, 1918.

[LG13] D. Larchey-Wendling and D. Galmiche. Nondeterministic phase semantics and the undecidability of boolean BI. *ACM Transactions on Computational Logic*, 14(1), 2013.

[Lib04] L. Libkin. *Elements of Finite Model Theory*. Springer, 2004.

[Loz04a] E. Lozes. *Expressivité des Logiques Spatiales*. Phd thesis, ENS Lyon, 2004.

[Loz04b] E. Lozes. Separation logic preserves the expressive power of classical logic. In *SPACE'04*, 2004.

[Loz12] E. Lozes. *Separation Logic: Expressiveness and Copyless Message-Passing*. Habilitation thesis, ENS Cachan, July 2012.

[LP14] W. Lee and S. Park. A proof system for separation logic with magic wand. In *POPL'14*, pages 477–490. ACM, 2014.

[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.

[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.

[Lut06] C. Lutz. Complexity and succinctness of public announcement logic. In *AAMAS'06*, pages 137–143. ACM, 2006.

[LWG10] D. Larchey-Wendling and D. Galmiche. The undecidability of Boolean BI through phase semantics. In *LICS'10*, pages 140–149. IEEE, 2010.

[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.

[MdR05] M. Marx and M. de Rijke. Semantic characterizations of navigational XPath. *SIGMOD Record*, 34(2):41–46, 2005.

[Min67] M. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, 1967.

[Moo77] R. C. Moore. Reasoning about knowledge and action. In *IJCAI-5*, pages 223–227, 1977.

[Mor76] Ch. Morgan. Methods for automated theorem proving in non classical logics. *IEEE Transactions on Computers*, 25(8):852–862, 1976.

[Mos83] B. Moszkowski. Reasoning about digital circuits. Technical Report STAN-CS-83-970, Dept. of Computer Science, Stanford University, Stanford, CA, 1983.

[MP92] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer-Verlag, New York, NY, USA, 1992.

[MPQ11] P. Madhusudan, G. Parlato, and X. Qiu. Decidable logics combining heap structures and data. In *POPL'11*, pages 611–622. ACM, 2011.

[MV97] M. Marx and Y. Venema. *Multi-Dimensional Modal Logic*. Applied Logic. Kluwer, 1997.

[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.

[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.

[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.

[OP99] P. O'Hearn and D. Pym. The logic of bunched implications. *Bulletin of Symbolic Logic*, 5(2):215–244, 1999.

[Pnu77] A. Pnueli. The temporal logic of programs. In *FOCS'77*, pages 46–57. IEEE, 1977.

[POY04] D. Pym, P. O'Hearn, and H. Yang. Possible worlds and resources: the semantics of BI. *Theoretical Computer Science*, 315(1):257–305, 2004.

[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.

[Pri67] A. Prior. *Past, Present and Future*. Oxford University Press, 1967.

[PW04] R. Pucella and V. Weissman. Reasoning about dynamic policies. In *FOS-*

SACS'04, volume 2987 of *Lecture Notes in Computer Science*, pages 453–467, 2004.

[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.

[Pym02] D. Pym. *The Semantics and Proof Theory of the Logic of Bunched Implications*, volume 26 of *Applied Logic*. Kluwer Academic Publishers, 2002.

[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.

[Qiu13] X. Qiu. *Automatic techniques for proving correctness of heap-manipulating programs*. PhD thesis, University of Illinois, 2013.

[Rab69] M. Rabin. Decidability of second-order theories and automata on infinite trees. *Transactions of the American Mathematical Society*, 41:1–35, 1969.

[Rab14] A. Rabinovich. A Proof of Kamp's theorem. *Logical Methods in Computer Science*, 10(1), 2014.

[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.

[Rey00] J.C. Reynolds. Intuitionistic reasoning about shared mutable data structure. In *Millennial Perspectives in Computer Science*, pages 303–321. Palgrave Macmillan Limited, 2000.

[Rey02] J.C. Reynolds. Separation logic: a logic for shared mutable data structures. In *LICS'02*, pages 55–74. IEEE, 2002.

[Rey08] J.C. Reynolds. An Overview of Separation Logic. In *VSTTE'05, Selected papers and discussions*, volume 4171 of *Lecture Notes in Computer Science*, pages 460–469. Springer, 2008.

[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.

[Sav70] W.J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177–192, 1970.

[SC14] M. Sighireanu and D. Cok. Report on SL-COMP 2014. *Journal of Satisfiability, Boolean Modeling and Computation*, 2014. To appear.

[Sch13] S. Schmitz. Complexity hierarchies beyond elementary. *CoRR*, abs/1312.5686, 2013.

[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.

[ST11] L. Segoufin and S. Toruńczyk. Automata based verification over linearly ordered data domains. In *STACS'11*, pages 81–92, 2011.

[Str94] H. Straubing. *Finite Automata, Formal Logic, and Circuit Complexity*. Progress in Theoretical Computer Science. Birkhäuser, 1994.

[SZ12] Th. Schwentick and Th. Zeume. Two-variable logic and two order relations. *Logical Methods in Computer Science*, 8:1–27, 2012.

[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.

[TC14] M. Tatsuta and W.-N. Chin. Completeness of Separation Logic with Inductive Definitions for Program Verification. In *SEFM'14*, volume 8702 of *Lecture Notes in Computer Science*, pages 20–34. Springer, 2014.

[TCA09] M. Tatsuta, W.-N. Chin, and M.F. Al Ameen. Completeness of pointer program verification by separation logic. In *SEFM'09*, pages 179–188. IEEE Computer Society, 2009.

[Tra63] B. Trakhtenbrot. Impossibility of an algorithm for the decision problem in finite classes. *AMS Translations, Series 2*, 23:1–5, 1963.

[Tue11] Th. Tuerk. *A separation logic fragment for HOL*. PhD thesis, University of Cambridge, 2011.

[Var82] M. Vardi. The complexity of relational query languages. In *14th Annual ACM Symposium on Theory of Computing*, pages 137–146. ACM, 1982.

[vB76] J. van Benthem. *Correspondence Theory*. PhD thesis, Mathematical Institute, University of Amsterdam, 1976.

[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.

[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.

[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.

[Wei11] Ph. Weis. *Expressiveness and Succinctness of First-Order Logic on Finite Words*. PhD thesis, University of Massachussetts, 2011.

[Wol83] P. Wolper. Temporal logic can be more expressive. *Information and Computation*, 56:72–99, 1983.

[Yan01] H. Yang. *Local Reasoning for Stateful Programs*. PhD thesis, University of Illinois, Urbana-Champaign, 2001.

[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.

[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.

[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.