Rémy Brochenin, Stéphane Demri
and Étienne Lozes

Reasoning about sequences of memory states

Research Report LSV-07-18

May 2007

Reasoning about sequences of memory states[⋆]

Rémi Brochenin, Stéphane Demri, and Etienne Lozes

LSV, ENS Cachan, CNRS, INRIA
{brocheni,demri,lozes}@lsv.ens-cachan.fr

Abstract. Motivated by the verification of programs with pointer variables, we introduce a temporal logic $\mathrm{LTL}^{\mathrm{mem}}$ whose underlying assertion language is the quantifier-free fragment of separation logic and the temporal logic on the top of it is the standard linear-time temporal logic LTL. We analyze the complexity of various model-checking and satisfiability problems for $\mathrm{LTL}^{\mathrm{mem}}$, considering various fragments of separation logic (including pointer arithmetic), various classes of models (with or without constant heap), and the influence of fixing the initial memory state. We provide a complete picture based on these criteria. Our main decidability result is PSPACE-completeness of the satisfiability problems on the record fragment and on a classical fragment allowing pointer arithmetic. Σ^0_1-completeness or Σ^1_1-completeness results are established for various problems by reducing standard problems for Minsky machines, and underline the tightness of our decidability results.

1 Introduction

Verification of programs with pointers. Model-checking of infinite-state systems is a very active area of formal verification [BCMS01] even though in full generality, simple reachability questions are undecidable. Nevertheless, many classes of infinite-state systems can be analyzed, such as Petri nets, timed automata, etc. Programs with pointer variables suffer the same drawback since reachability problems are also undecidable, see e.g. [BFN04,BBH⁺06]. It is worth noting that specific properties need to be verified for such programs, such as the existence of memory leaks, memory violation, or shape analysis. Prominent logics for analyzing such programs are Separation Logic [Rey02], pointer assertion logic PAL [JJKS97], TVLA [LAS00] and alias logic [BIL04], to quote a few examples.

Temporal Separation Logic: what for? Since [Pnu77], temporal logics are used as languages for formal specification of programs. General and powerful automata-based techniques for verification have been developed, see for example the works [VW94,KVW00]. On the other hand, Separation Logic is a static logic for program annotation [Rey02], and more recently for symbolic computation [BCO05b]. Extending the scope of application of Separation Logic to standard temporal logic-based verification techniques has many potential interests. First, it provides a rich underlying assertion language where properties more complex than accessibility can be stated. Second, this probably yields a significant feedback for the purely static Separation Logic extended with general recursion, which has not been much studied up to now. For instance, if we write $\mathsf{X}\mathtt{x}$ to denote the next value of $\mathtt{x}$ (also sometimes written $\mathtt{x}'$), the formula $(\mathtt{x} \hookrightarrow \mathsf{X}\mathtt{x})\mathsf{U}(\mathtt{x} \hookrightarrow \mathtt{null})$,

⋆ Work partially supported by the RNTL project "AVERILES". The first author is supported by a fellowship from CNRS/DGA. Extended version of [BDL07].

understood on a model with constant heap, characterises the existence of a simple flat list, which is usually written $\mu L(\mathtt{x}).\ \mathtt{x} \hookrightarrow \mathtt{null} \vee \exists \mathtt{x}'.\mathtt{x} \hookrightarrow \mathtt{x}' \wedge L(\mathtt{x}')$. Third, temporal logics allow to work in the very convenient framework of "programs-as-formulae" and decision procedures for logical problems can be directly used for program verification. For instance, the previous formula can be seen as a program walking on a list, and more generally programs without destructive updates can be expressed as formulae. Some programs with destructive updates that perform a simple pass on the heap, have an input-output relation that may be described by a formula. For instance, the formula $(\mathtt{x} \hookrightarrow_0 \mathsf{X}\mathtt{x} \wedge \mathsf{X}\mathtt{x} \hookrightarrow_1 \mathtt{x})\mathsf{U}\mathtt{x} \hookrightarrow_0 \mathtt{null}$ roughly expresses that the list in the initial heap h_0 is reversed in final heap h_1. Fourth, pointer arithmetic has been poorly studied until now, whereas arithmetical constraints in temporal logics are known to lead to undecidability, see e.g. [CC00]. Actually, there is a growing interest in understanding the interplay of pointer arithmetic, temporal reasoning, and non aliasing properties.

Our contribution. We introduce a linear-time temporal logic $\mathrm{LTL}^{\mathrm{mem}}$ to specify sequences of memory states with underlying assertion language based on quantifier-free Separation Logic [Rey02]. From a logical perspective, the logic $\mathrm{LTL}^{\mathrm{mem}}$ can be viewed as a many-dimensional logic [GKWZ03] since $\mathrm{LTL}^{\mathrm{mem}}$ contains a temporal dimension and the spatial dimension for memory states. Other many-dimensional logics can be found in [BWZ02,BC02,GKWZ03,DD07]. Our logic addresses a very general notion of models, including the aspects of pointer arithmetic and recursive structures with records. We distinguish the satisfiability problems from the model-checking problems, as well as distinct subclasses of interesting programs, like for instance the programs without destructive update. The most promising result for future implementation is the PSPACE-completeness of the satisfiability problems SAT(CL) and SAT(RF) where CL is the classical fragment without separation connectives and RF is the record fragment with no pointer arithmetic but with separation connectives. This result is very tight, as both propositional LTL and static Separation Logic are already PSPACE-complete [SC85,CYO01]. These results are obtained by reduction to the nonemptiness problem for Büchi automata on an alphabet made of symbolic memory states obtained by an abstraction that we show sound and complete, see e.g. [Loz04,CGH05]. Such abstractions are similar to resource graphs from [GM05]. This is a variant of the automata-based approach introduced in [VW94] for plain LTL and further developed with concrete domains of interpretation in [DD07]. Surprisingly, the abstraction method used to establish these results does not scale to the whole logic, due to a subtle interplay between separation connectives and pointer arithmetic. Moreover, we provide new undecidability results for several problems, for instance $\mathrm{SAT}^{ct}(\mathrm{LF})$ (satisfiability with constant heap on the list fragment).

Related work. Previous temporal logics designed for pointer verification include Evolution Temporal Logic [YRSW03], based on the three-valued logic abstraction method that made the success of TVLA [LAS00], and Navigation temporal logic [DKR04], based on a tableau method quite similar to our automaton-based reduction. In these works, the assertion language for states is quite rich, as it includes for instance list predicate, quantification over adresses, and a freshness predicate. Because of this high expressive power, only incomplete abstractions are proposed, whereas we stick to exact methods. More importantly, our work addresses models with constant heaps and pointer arithmetic, which has not been done so far, and leads to a quite different perspective.

Structure of the paper We define our logic $\mathrm{LTL}^{\mathrm{mem}}$ and several fragments and problems in Section 2. Section 3 introduces the symbolic memory states (also useful in Section 4) and presents the PSPACE-completeness of the satisfiability and model-checking problems for SL with pointer arithmetic. Section 4 is dedicated to the decidability proof of satisfiability for various fragments and its consequences for other problems. In Section 5, we mention several seemingly optimal undecidability results by encoding computations of Minsky machines. Section 6 contains concluding remarks.

2 Memory Model and Specification Language

In this section, we introduce a separation logic dealing with pointer arithmetic and record values, and a temporal logic $\mathrm{LTL}^{\mathrm{mem}}$. Unlike BI's pointer logic from [IO01], we allow pointer arithmetic.

2.1 A separation logic with pointer arithmetic

Memory states. Let us introduce our model of memory. It captures features of programs with pointer variables that use pointer arithmetic and records. We assume a countably infinite set $\mathtt{Var}$ of variables (as usual, for a fixed formula we need only a finite amount), and an infinite set $\mathtt{Val}$ of values containing the set $\mathbb{N}$ of naturals, thought as address indexes, and a special value nil. For simplicity, we assume that $\mathtt{Val} = \mathbb{N} \uplus \{nil\}$. In order to model field selectors, we consider some infinite set $\mathtt{Lab}$ of labels. We will usually range over values with u, v, over naturals with i, j, over labels with $l, r, next, prev$, and over variables with $\mathtt{x}, \mathtt{y}$. In the remainder, we will assume some fixed injection $(\mathtt{x}, i) \in \mathtt{Var} \times \mathbb{N} \mapsto \langle \mathtt{x}, i \rangle \in \mathtt{Var}$.

We use the notation $E \rightharpoonup_{fin} F$ for the set of partial functions from E to F of finite domain; and $E \rightharpoonup_{fin+} F$ for the set of partial functions from E to F of finite and nonempty domain. The sets $\mathcal{S}$ of stores and $\mathcal{H}$ of heaps are then defined as follows:

$$\mathcal{S} \overset{\text{def}}{\equiv} \mathtt{Var} \to \mathtt{Val} \qquad \mathcal{H} \overset{\text{def}}{\equiv} \mathbb{N} \rightharpoonup_{fin} (\mathtt{Lab} \rightharpoonup_{fin+} \mathtt{Val}).$$

We will range over a store with s, s' and over a heap with h, h', h_1, h_2. We call *memory state* a couple $(s, h) \in \mathcal{S} \times \mathcal{H}$.

We will refer to the domain of a heap h by $\mathtt{dom}(h) \subseteq \mathbb{N}$. Intuitively, in our memory model, each index is thought as an entry point on some record cell containing several fields. Cells are either not allocated, or allocated with some

Expressions

$e ::= \mathtt{x} \mid \mathtt{null}$

Atomic formulae

$\pi ::= e = e' \mid e + i \overset{l}{\hookrightarrow} e$

State Formulae

$$\begin{array}{lll} \mathcal{A} ::= & \pi & \\ & \mid \mathcal{A} * \mathcal{B} \mid \mathcal{A} \mathrel{-\!\!*} \mathcal{B} \mid \mathtt{emp} & \text{(spatial fragment)} \\ & \mid \mathcal{A} \wedge \mathcal{B} \mid \mathcal{A} \rightarrow \mathcal{B} \mid \bot & \text{(classical fragment)} \end{array}$$

Satisfaction

$(s,h) \models_{\mathrm{SL}} e = e'$	iff $[\![e]\!]_s = [\![e']\!]_s$, with $[\![\mathtt{x}]\!]_s = s(\mathtt{x})$ and $[\![\mathtt{null}]\!]_s = nil$
$(s,h) \models_{\mathrm{SL}} e + i \overset{l}{\hookrightarrow} e'$	iff $[\![e]\!]_s \in \mathbb{N}$ and $[\![e]\!] + i \in \mathtt{dom}(h)$ and $h(s(\mathtt{x}) + i)(l) = [\![e']\!]_s$
$(s,h) \models_{\mathrm{SL}} \mathtt{emp}$	iff $\mathtt{dom}(h) = \emptyset$
$(s,h) \models_{\mathrm{SL}} \mathcal{A}_1 * \mathcal{A}_2$	iff $\exists\ h_1, h_2$ s.t. $h = h_1 * h_2$, $(s,h_1) \models_{\mathrm{SL}} \mathcal{A}_1$ and $(s,h_2) \models_{\mathrm{SL}} \mathcal{A}_2$
$(s,h) \models_{\mathrm{SL}} \mathcal{A}' \mathrel{-\!\!*} \mathcal{A}$	iff for all h', if $h \perp h'$ and $(s,h') \models_{\mathrm{SL}} \mathcal{A}'$ then $(s, h * h') \models_{\mathrm{SL}} \mathcal{A}$
$(s,h) \models_{\mathrm{SL}} \mathcal{A}_1 \wedge \mathcal{A}_2$	iff $(s,h) \models_{\mathrm{SL}} \mathcal{A}_1$ and $(s,h) \models_{\mathrm{SL}} \mathcal{A}_2$
$(s,h) \models_{\mathrm{SL}} \mathcal{A}' \rightarrow \mathcal{A}$	iff $(s,h) \models_{\mathrm{SL}} \mathcal{A}'$ implies $(s,h) \models_{\mathrm{SL}} \mathcal{A}$
$(s,h) \models_{\mathrm{SL}} \bot$	never

Table 1. The syntax and semantics of SL with pointer arithmetic and records

record stored in. In a memory state (s,h), the memory cell at index i is *allocated* if $i \in \mathtt{dom}(h)$; in this case the stored record is $h(i) = \{l_1 \mapsto v_1, .., l_n \mapsto v_n\}$.

Note that the size of the information held in a memory cell is not fixed, nor bounded. Our models could be more concrete considering labels as offsets and relying on pointer arithmetic. But for our purpose, it will be convenient to consider pointer arithmetic independently.

Separation Logic. We now introduce the separation logic (SL) on top of which we will define our temporal logic. The syntax of the logic is given in Table 1.

In short, Separation logic is about reasoning on disjoint heaps, and we need to define what we mean by "disjoint heaps" in our model. Our level of granularity implies that a record cell cannot be decomposed in disjoint parts. Let h_1 and h_2 be two heaps; we say that h_1 and h_2 are disjoint, noted $h_1 \perp h_2$, if $\mathtt{dom}(h_1) \cap \mathtt{dom}(h_2) = \emptyset$. The operation $h_1 * h_2$ is defined for disjoint heaps as the *disjoint union* of the two partial functions. Semantics of formulae is defined by the satisfaction relation $\models_{\mathrm{SL}}$ (see Table 1).

Formulae π are *atomic formulae.* The formula $\mathtt{x} + i \overset{l}{\hookrightarrow} e$ states that the value of the l field of the record stored at the address pointed by $\mathtt{x}$ with offset i is equal to the value of the expression e. The formula $e = e'$ states the equality between the values of the two expressions, and $\mathtt{emp}$ means that the current heap has no memory cell allocated.

Formulae $\mathcal{A}$ of SL are called *state formulae.* The size of the state formula $\mathcal{A}$, written $|\mathcal{A}|$, is the length of the string $\mathcal{A}$ for some reasonably succinct encoding of variables and integers with a binary representation. We will use the map $|\cdot|$ for other syntactic objects such as $\mathrm{LTL}^{\mathrm{mem}}$ formulae. A formula $\mathcal{A} * \mathcal{B}$ with the *separation conjunction* states that $\mathcal{A}$ holds on some portion of the memory heap and $\mathcal{B}$ holds on a disjoint portion. A formula $\mathcal{A} \mathrel{-\!\!*} \mathcal{B}$ states that the current heap, when extended with any disjoint heap verifying $\mathcal{A}$, will verify $\mathcal{B}$. Boolean operators are understood as usual. Derivable connectives $\mathcal{A} \vee \mathcal{B}$, $\neg \mathcal{A}$ are defined as usual. In the remainder, we focus on several specific fragments of this separation logic. We say that a formula is in the *record fragment* (RF) if all subformulae

$\mathtt{x}+i \overset{l}{\hookrightarrow} e$ use $i = 0$. In that case, we write $\mathtt{x} \overset{l}{\hookrightarrow} e$. We say that a formula is in the *classical fragment* (CL) if it does not contain any of the connectives $*$ and $-\!\!*$. Finally, we say that a formula is in the *list fragment* (LF) if it is in the classical fragment and all subformulae $\mathtt{x} + i \overset{l}{\hookrightarrow} e$ use $i = 0$ and $l = next$, and we may simply write $\mathtt{x} \hookrightarrow e$. Clearly, the classical and record fragments are incomparable, while the list fragment is included in both of them.

Let us illustrate the expressive power of SL on examples. The formula $\neg\mathtt{emp} * \neg\mathtt{emp}$ means that at least two memory cells are allocated. The formula $\mathtt{x} \overset{l}{\mapsto} e$, defined as $\neg(\neg\mathtt{emp} * \neg\mathtt{emp}) \wedge \mathtt{x} \overset{l}{\hookrightarrow} e$, is the local version of $\mathtt{x} \overset{l}{\hookrightarrow} e$: $s, h \models_{\mathrm{SL}} \mathtt{x} \overset{l}{\mapsto} e$ iff $\mathsf{dom}(h) = \{s(\mathtt{x})\}$ and $h(s(\mathtt{x}))(l) = [\![\ e \ ]\!]_s$. The formula $(\mathtt{x} \overset{l}{\hookrightarrow} \mathtt{null}) -\!\!* \bot$ is satisfied at (s_0, h_0) whenever there is no heap h_1 with $h_1 \perp h_0$ that allocates the variable $\mathtt{x}$ to *nil* on l field. In other words, the variable $\mathtt{x}$ is allocated in the heap h_0.

As usual, $\mathcal{A}$ is valid iff for every memory state (s, h), we have $(s, h) \models_{\mathrm{SL}} \mathcal{A}$ (written $\models_{\mathrm{SL}} \mathcal{A}$). Satisfiability is defined dually.

2.2 Temporal extension

Memory states sequences. Models of the logic $\mathrm{LTL}^{\mathrm{mem}}$ are ω-sequences of memory states, that is elements in $(\mathcal{S} \times \mathcal{H})^\omega$ and they are understood as infinite computations of programs with pointer variables. We range over ρ for a given model, and its i^{th} state $\rho(i)$ will be noted (s_i, h_i). In order to analyze computations from programs without destructive update, we shall also consider models with constant heap, that is elements in $\mathcal{S}^\omega \times \mathcal{H}$.

The logic $\mathrm{LTL}^{\mathrm{mem}}$. Formulae of $\mathrm{LTL}^{\mathrm{mem}}$ are defined in Table 2. Atomic formulae of $\mathrm{LTL}^{\mathrm{mem}}$ are state formulae from SL except that variables can be prefixed by the symbol "X". For instance, $\mathsf{X}\mathtt{x}$ is interpreted by the value of $\mathtt{x}$ at the next memory state. We use the notation $\mathsf{X}^i\mathtt{x}$ for $\overbrace{\mathsf{X}\ldots\mathsf{X}}^{i \text{ times}}\mathtt{x}$ (but keep in mind that encoding $\mathsf{X}^i\mathtt{x}$ requires memory space in $\mathcal{O}(i)$). The temporal operators are the standard next-time operator X and until operator U present in LTL, see e.g. [SC85]. The satisfaction relation $\rho, t \models \phi$ where ρ is a model of $\mathrm{LTL}^{\mathrm{mem}}$, $t \in \mathbb{N}$ and ϕ is a formula is also defined in Table 2. We use standard abbreviations such as $\mathsf{F}\phi$, $\mathsf{G}\phi$ ... We freely use propositional variables p, q, having in mind that the propositional variable p should be understood as $\mathtt{x}_p = \mathtt{x}_\top$ for some fixed extra variables $\mathtt{x}_p, \mathtt{x}_q, \ldots, \mathtt{x}_\top$.

Given a fragment Frag of SL, $\mathrm{LTL}^{\mathrm{mem}}(\mathrm{Frag})$ is the restriction of $\mathrm{LTL}^{\mathrm{mem}}$ to formulae in which occur only state formulae built over Frag (with extended variables $\mathsf{X}^i\mathtt{x}$), and we write $\mathrm{SAT}(\mathrm{Frag})$ to denote the satisfiability problem for $\mathrm{LTL}^{\mathrm{mem}}(\mathrm{Frag})$: given a temporal formula ϕ in $\mathrm{LTL}^{\mathrm{mem}}(\mathrm{Frag})$, is there a model ρ such that $\rho, 0 \models \phi$? The variant problem in which we require that the model has a constant heap [resp. that the initial memory state is fixed] is denoted by $\mathrm{SAT}^{ct}(\mathrm{Frag})$ [resp. $\mathrm{SAT}_{init}(\mathrm{Frag})$]. The problem $\mathrm{SAT}^{ct}_{init}(\mathrm{Frag})$ is defined analogously.

Enriched expressions	$\eta ::= \ \mathtt{x} \mid \mathsf{X}\eta \mid \ \mathtt{null}$
Atomic formulae	$\pi ::= \ \eta = \eta' \mid \ \eta + i \overset{l}{\hookrightarrow} \eta'$
State formulae	$\mathcal{A} ::= \ \pi \mid \ \mathtt{emp} \mid \ \mathcal{A} * \mathcal{B} \mid \ \mathcal{A} \mathrel{-\!\!*} \mathcal{B} \mid \ \mathcal{A} \wedge \mathcal{B} \mid \ \mathcal{A} \rightarrow \mathcal{B} \mid \ \perp$
Temporal formulae	$\phi ::= \ \mathcal{A} \mid \ \mathsf{X}\phi \mid \ \phi \mathsf{U} \phi' \mid \ \phi \wedge \phi' \mid \ \neg\phi$

Semantics

$\rho, t \models \mathsf{X}\phi$	iff $\rho, t+1 \models \phi$.
$\rho, t \models \phi \mathsf{U} \phi'$	iff there is $t_1 \geq t$ s.t. $\rho, t_1 \models \phi'$ and $\rho, t' \models \phi$ for all $t' \in \{t, .., t_1 - 1\}$.
$\rho, t \models \phi \wedge \psi$	iff $\rho, t \models \phi$ and $\rho, t \models \psi$.
$\rho, t \models \neg\phi$	iff $\rho, t \not\models \phi$.
$\rho, t \models \mathcal{A}$	iff $s'_t, h_t \models_{\mathrm{SL}} \mathcal{A}[\mathsf{X}^k\mathtt{x} \leftarrow \langle \mathtt{x}, k \rangle]$ where $\rho = (s_t, h_t)_{t \geq 0}$ and s'_t is defined by $s'_t(\langle \mathtt{x}, k \rangle) = s_{t+k}(\mathtt{x})$.

Table 2. The syntax and semantics of $\mathrm{LTL}^{\mathrm{mem}}$

2.3 Programs with pointer variables

In this section, we define the model-checking problems for programs with pointer variables over $\mathrm{LTL}^{\mathrm{mem}}$ specifications. The set $\mathtt{I}$ of *instructions* used in the programs is defined by the grammar below:

$$\begin{aligned} \mathtt{instr} ::= {} & \mathtt{x} := \mathtt{y} \mid \mathtt{skip} \\ & \mid \mathtt{x} := \mathtt{y} \rightarrow l \mid \mathtt{x} \rightarrow l := \mathtt{y} \mid \mathtt{x} := \mathtt{cons}(l_1 : \mathtt{x}_1, .., l_k : \mathtt{x}_k) \mid \mathtt{free}\ \mathtt{x}, l \\ & \mid \mathtt{x} := \mathtt{y}[i] \mid \mathtt{x}[i] := \mathtt{y} \mid \mathtt{x} := \mathtt{malloc}(i) \mid \mathtt{free}\ \mathtt{x}, i \end{aligned}$$

The denotational semantics of an instruction $\mathtt{instr}$ is defined as a partial function $[\![\ \mathtt{instr}\ ]\!] : \mathcal{S} \times \mathcal{H} \rightarrow \mathcal{S} \times \mathcal{H}$, undefined when the instruction would cause a memory violation. We list in Table 3 the formal denotational semantics of our instruction set. Boolean combinations of equalities between expressions are called guards and its set is denoted by G. A program is defined as a triple (Q, δ, q_I) such that Q is a finite set of control states, q_I is the initial state and δ is the transition relation, a subset of $Q \times G \times \mathtt{I} \times Q$. We use $q \xrightarrow{g,\mathtt{instr}} q'$ to denote a transition. We say that a program is *without destructive update* if transitions are labeled only with instructions of the form $\mathtt{x} := \mathtt{y}$, $\mathtt{x} := \mathtt{y} \rightarrow l$, and $\mathtt{x} := \mathtt{y}[i]$. We write $\mathtt{P}$ to denote the set of programs and $\mathtt{P}^{ct}$ to denote the set of programs without destructive update.

A program is a finite object whose interpretation can be viewed as an infinite-state system. More precisely, given a program $\mathtt{p} = (Q, \delta, q_I)$, the transition system $\mathcal{S}_{\mathtt{p}} = (S, \rightarrow)$ is defined as follows: $S = Q \times (\mathcal{S} \times \mathcal{H})$ (set of configurations) and $(q, (s, h)) \rightarrow (q', (s', h'))$ iff there is a transition $q \xrightarrow{g,\mathtt{instr}} q' \in \delta$ such that $(s, h) \models g$ and $(s', h') = [\![\ \mathtt{instr}\ ]\!](s, h)$. Note that $\mathcal{S}_{\mathtt{p}}$ is not necessarily linear. A computation (or execution) of $\mathtt{p}$ is defined as an infinite path in $\mathcal{S}_{\mathtt{p}}$ starting with control state q_I.

Computations of $\mathtt{p}$ can be viewed as $\mathrm{LTL}^{\mathrm{mem}}$ models, using propositional variables to encode the extra information about the control states (details are omitted herein).

Model-checking aims at checking properties expressible in $\mathrm{LTL}^{\mathrm{mem}}$ along computations of programs. To a logical fragment (SL, CL, RF, or LF), we associate a

$[\![\ \mathtt{x} := \mathtt{y}\ ]\!]\ (s,h)$	$\overset{\text{def}}{\equiv}\ (s[\mathtt{x} \mapsto s(\mathtt{y})], h).$
$[\![\ \mathtt{x} := \mathtt{y} \rightarrow l\ ]\!]\ (s, h * \{i \mapsto \{l \mapsto v, \dots\}\})$	$\overset{\text{def}}{\equiv}\ (s[\mathtt{x} \mapsto v], h * \{i \mapsto \{l \mapsto v, \dots\}\})$ with $s(\mathtt{y}) = i$
$[\![\ \mathtt{x} \rightarrow l := \mathtt{y}\ ]\!]\ (s, h * \{i \mapsto \{l \mapsto v, \dots\}\})$	$\overset{\text{def}}{\equiv}\ (s, h * \{i \mapsto \{l \mapsto s(\mathtt{y}), \dots\}\})$ with $s(\mathtt{x}) = i$
$[\![\ \mathtt{x} := \mathtt{cons}(l_1 : \mathtt{x}_1, \dots, l_k : \mathtt{x}_k)\ ]\!]\ \ (s,h)$	$\overset{\text{def}}{\equiv}\ \big(s[\mathtt{x} \mapsto i], h * \{i \mapsto \{l_1 \mapsto s(\mathtt{x}_1), \dots, l_k \mapsto s(\mathtt{x}_k)\}\}\big)$ with $i \notin \mathtt{dom}(h)$
$[\![\ \mathtt{free}\ \mathtt{x}, l\ ]\!]\ (s, h * \{i \mapsto \{l \mapsto v, \dots\}\})$	$\overset{\text{def}}{\equiv}\ (s, h * \{i \mapsto \{\dots\}\})$ with $s(\mathtt{x}) = i$
$[\![\ \mathtt{skip}\ ]\!]\ (s,h)$	$\overset{\text{def}}{\equiv}\ (s,h)$
$[\![\ \mathtt{x} := \mathtt{y}[i]\ ]\!]\ (s, h * \{i + i' \mapsto \{\mathit{next} \mapsto v\}\})$	$\overset{\text{def}}{\equiv}\ (s[\mathtt{x} \mapsto v], h * \{i \mapsto \{\mathit{next} \mapsto v\}\}))$ with $s(\mathtt{y}) = i'$
$[\![\ \mathtt{x}[i] := \mathtt{y}\ ]\!]\ (s, h * \{i' + i \mapsto \{\mathit{next} \mapsto v\}\})$	$\overset{\text{def}}{\equiv}\ (s, h * \{i + i' \mapsto \{\mathit{next} \mapsto s(\mathtt{y})\}\})$ with $s(\mathtt{x}) = i'$
$[\![\ \mathtt{x} := \mathtt{malloc}(i)\ ]\!](s,h)$	$\overset{\text{def}}{\equiv}\ \big(s[\mathtt{x} \mapsto i'], h * \{i' \mapsto \{\mathit{next} \mapsto \mathit{nil}\}, \dots, i' + (i-1) \mapsto \{\mathit{next} \mapsto \mathit{nil}\}\}\big)$ with $i', \dots, i' + (i-1) \notin \mathtt{dom}(h)$
$[\![\ \mathtt{free}\ \mathtt{x}, i\ ]\!]\ (s, h * \{i' + i \mapsto f\})$	$\overset{\text{def}}{\equiv}\ (s,h)$ with $s(\mathtt{x}) = i'$

Table 3. Semantics for instructions

set of programs : all programs for SL and CL, programs with instructions having $i = 0$ for RF, and moreover with only the label next for LF. Given one of these fragments Frag of SL, we write MC(Frag) to denote the model-checking problem for Frag: given a temporal formula ϕ in $\mathrm{LTL}^{\mathrm{mem}}$ with state formulae built over Frag and a program $\mathtt{p}$ of the associated fragment, is there an infinite computation ρ of $\mathtt{p}$ such that $\rho, 0 \models \phi$ (which we write $\mathtt{p} \models \phi$)? The variant problem in which we require that the program is without destructive update [resp. that the initial memory state is fixed, say (s,h)] is denoted by MC^{ct}(Frag) [resp. MC_{init}(Frag)]. The problem MC^{ct}_{init}(Frag) is defined analogously. We may write $\mathtt{p}, (s,h) \models \phi$ to emphasize what is the initial memory state.

All the model-checking and satisfiability problems defined above can be placed in Σ^1_1 in the analytical hierarchy. Indeed, the models and computations of programs can be viewed as functions $f : \mathbb{N} \rightarrow \mathbb{N}$ by encoding memory states and configurations by natural numbers (details are tedious). Then, the satisfaction relation between models and $\mathrm{LTL}^{\mathrm{mem}}$ formulae and the transition relations obtained from programs can be encoded by a first-order formula. This guarantees that these problems are in Σ^1_1. Additionnally, all the above problems can easily be shown PSPACE-hard since they all generalize LTL satisfiability and model-checking [SC85].

Using extended variables $\mathsf{X}\mathtt{x}$, we may express some programs as formulae. This actually holds only for programs without update, for the semantics with constant heap. Intuitively, we express the control of the program with propositional variables, and define a formula that encode the transitions. As a consequence, the following result can be derived.

Lemma 1. *Let* Frag *be a fragment among* SL*,* CL*,* RF*, or* LF*. There is a logspace reduction from* $\mathrm{MC}^{ct}(\mathrm{Frag})$ *to* $\mathrm{SAT}^{ct}(\mathrm{Frag})$ *(resp. from* $\mathrm{MC}^{ct}_{init}(\mathrm{Frag})$ *to* $\mathrm{SAT}^{ct}_{init}(\mathrm{Frag})$*).*

Proof. We adapt the proof in [SC85] for reducing LTL model-checking to LTL satisfiability. To a program $\mathtt{p} = (Q, \delta, q_I)$, we associate the formula $\phi_{\mathtt{p}}$ below built over the propositional variables in Q:

$$q_I \wedge \mathsf{G} \bigwedge_{q \in Q} (q \Rightarrow (\bigwedge_{q' \in Q \setminus \{q\}} \neg q' \wedge \bigvee_{\tau \in \delta_q^+} \phi_\tau))$$

where ϕ_τ expresses that transition τ is fired between current state and next state and δ_q^+ is the set of transitions starting at state q. In order to define ϕ_τ, we need to translate instructions and guards into the logic. We translate instructions of the form $\mathtt{x} := \mathtt{y}$ into $\mathsf{X}\mathtt{x} = \mathtt{y}$, $\mathtt{x} := \mathtt{y} \rightarrow l$ into $\mathtt{y} \overset{l}{\hookrightarrow} \mathsf{X}\mathtt{x}$, and $\mathtt{x} := \mathtt{y}[i]$ into $\mathtt{y} + i \hookrightarrow \mathsf{X}\mathtt{x}$. Guards are translated accordingly. It is then standard to show that $\mathtt{p} \models \phi$ iff $\phi \wedge \phi_{\mathtt{p}}$ is satisfiable.

2.4 Discussion

We are currently investigating issues about the expressive power of this logical formalism. Let us discuss few issues below.

First, the interest of model-checking programs with heap updates stems from early works on automata-based verification. Decision procedures are obtained at the cost of limitations: to define approximations as done in [YRSW03,DKR04] or to restrict the programming language, see e.g. [BFLS06]. However, with this approach, compositionality principles are lost which is a pity since they made the success of separation logic, as frame rule and composition rule.

Second, assuming that the heap is constant is subject to promising development. Indeed, it is then possible to define spatial operators at the same syntactic level as temporal operators, and write formulae as e.g. $[((\mathtt{x} \hookrightarrow \mathsf{X}\mathtt{x})\mathsf{U}(\mathtt{x} \hookrightarrow \mathtt{null}))] \ast (\mathtt{y} \mapsto \mathtt{null})$. This might be a way to model modularity in model-checking programs without destructive updates, but there are other points of interest we will try to advocate now.

Recursion with local parameters The constant heap semantics provides an original viewpoint for recursion with local parameters and local quantification. The design of decision procedures in presence of recursive predicates has not yet completely satisfactory answers. Specific axiomatizations have been proposed for some standard recursive structures [BCO05a] as well as incomplete methods of inference even though they are apparently good in practice.

In order to be a bit more precise, let us consider the fragment of recursive separation logic where all recursive formulae are of the form:

$$(1) \quad \mu X(x_1, .., x_k).\ A(x_1, .., x_k) \vee \exists x'_1..x'_k.\ B(x_1, .., x_k, x'_1, .., x'_k) \wedge X(x'_1, .., x'_k)$$

This fragment is rich enough to express single lists, cyclic lists, and doubly-linked lists. However, we conjecture that it is not expressive enough for trees and DAGs.

We conjecture that deciding satisfiability in the fragment of recursive separation logic mentioned above reduces to $\mathrm{SAT}^{ct}(\mathrm{SL})$, and the model-checking problem reduces to SAT^{ct}_{init}, considering that (1) can be rewritten as:

$$\big(B(x_1, .., x_k, \mathsf{X}x_1, .., \mathsf{X}x_k)\big) \mathsf{U}\ A(x_1, .., x_k).$$

In this perspective, our results could rise interesting decidability results for model-checking some of the recursive separation logic with local quantifiers. For satisfiability, we expect to define decidable fragments for $\mathrm{SAT}^{ct}(\mathrm{SL})$, for instance considering the techniques for checking temporal properties of flat programs without destructive updates introduced in [FLS07]. Another interesting fragment of recursive separation logic is probably the one where recursion is guarded by the separation operator $*$, but we do not currently see how to treat it in the temporal logic perspective.

Programs as formulae Let us speculate some more. We may take advantage of expressing programs as formulae in order to reduce model-checking to satisfiability, a known approach since [SC85]. For programs without destructive update, we have Lemma 1. Moreover, we believe we can extend this result to programs with updates, but with a slightly different perspective. The constant heap semantics can be helpful to define the input-output relation of programs, even with destructive updates, provided some conditions on the way the program read and write over the memory are satisfied. To do so, we consider the extension of $\mathrm{LTL}^{\mathrm{mem}}$ with two predicates $\hookrightarrow_0$ and $\hookrightarrow_1$ instead of the single $\hookrightarrow$, and models are couples of state sequences with constant heap, that is tuples $\langle (s_i)_{i \geq 0}, h_0, h_1 \rangle$. Let us define the input-output relation R_P of a program P as : for all $(s_0, h_0), (s_1, h_1)$, $(s_0, h_0) R_P (s_1, h_1)$ if there is a run of P that starts with (s_0, h_0) and ends with (s_1, h_1). Then we conjecture that for an interesting class of programs, this relation is definable in $\mathrm{LTL}^{\mathrm{mem}}$ extended with $\hookrightarrow_0$ and $\hookrightarrow_1$. Basically, the encoding of the control of the program will be the same as for programs without destructive updates, but the encoding of the instructions will be different. For instance, $\mathtt{x} \rightarrow l := \mathtt{y}$ would be encoded by $(\mathsf{X}\mathtt{x}) \overset{l}{\hookrightarrow}_1 \mathtt{y}$ whereas $\mathtt{x} := \mathtt{y} \rightarrow l$ would be encoded as $\mathtt{y} \overset{l}{\hookrightarrow}_0 \mathsf{X}\mathtt{x}$.

3 Separation Logic: Complexity and Abstraction

After defining an abstraction for the fragment RF of SL, which will be proved to be sound, we will be able to decide the complexity of model checking and satisfiability for the whole SL (Section 4).

3.1 Syntactic measures

The main approach to get decision procedures to verify infinite-state systems consists in introducing a symbolic representation for infinite sets of configurations. The symbolic representation defined below plays a similar role and has similarities with symbolic heaps for Separation Logic in Smallfoot [BCO05b]. Let us start by some useful definitions. Following [Loz04], we introduce the set of *test formulae* that are formulae from SL of the forms below:

- $\texttt{alloc x} \overset{\text{def}}{\equiv} (\texttt{x} \overset{next}{\mapsto} \texttt{null}) {-\!\!*} \bot$ (x is allocated).
- $\texttt{size} \geq k \overset{\text{def}}{\equiv} \overbrace{\neg\texttt{emp} * \ldots * \neg\texttt{emp}}^{k \text{ times}}$ (at least k indices are allocated).
- $e + i \overset{l}{\hookrightarrow} e$, $e = e'$.

Given a formula ϕ of LTL$^{\text{mem}}$, we define its measure μ_ϕ understood as some pieces of information about the syntactic resources involved in ϕ. Indeed, forthcoming symbolic states are finite objects parameterized by such syntactic measures.

For a state formula $\mathcal{A}$ of LTL$^{\text{mem}}$, the size of memory examined by $\mathcal{A}$, written $w_\mathcal{A}$, is inductively defined as follows: $w_\mathcal{A}$ is 1 for atomic formulae, $max\{w_{\mathcal{A}_1}, w_{\mathcal{A}_2}\}$ for $\mathcal{A}_1 \wedge \mathcal{A}_2$ or $\mathcal{A}_1 \rightarrow \mathcal{A}_2$ or $\mathcal{A}_1 {-\!\!*} \mathcal{A}_2$, and $w_{\mathcal{A}_1} + w_{\mathcal{A}_2}$ for $\mathcal{A}_1 * \mathcal{A}_2$. Observe that $w_\mathcal{A} \leq |\mathcal{A}|$. Other simple sets about the syntactic resources of $\mathcal{A}$ need to be defined: $\texttt{Lab}_\mathcal{A}$ is the set of labels from $\texttt{Lab}$ occurring in $\mathcal{A}$, $\texttt{Var}_\mathcal{A}$ is the set of variables from $\texttt{Var}$ occurring in $\mathcal{A}$, $\epsilon_\mathcal{A}$ is the set of natural numbers i such that $e + i \overset{l}{\hookrightarrow} e'$ occurs in $\mathcal{A}$ and $m_\mathcal{A}$ is the maximal k such that $\mathsf{X}^k\texttt{x}$ occurs in $\mathcal{A}$ for some variable $\texttt{x}$. A measure is defined as an element of

$$\mathbb{N} \times \mathcal{P}_f(\mathbb{N}) \times \mathbb{N} \times \mathcal{P}_f(\texttt{Lab}) \times \mathcal{P}_f(\texttt{Var})$$

where $\mathcal{P}_f(X)$ denotes the set of finite subsets of some set X. The set of measures has a natural lattice structure for the pointwise order, noted below $\mu \leq \mu'$. We also write $\mu[w \leftarrow 0]$ to denote the measure μ except that $w = 0$.

The measure for $\mathcal{A}$, written $\mu_\mathcal{A}$, is the tuple $(m_\mathcal{A}, \epsilon_\mathcal{A}, w_\mathcal{A}, \texttt{Lab}_\mathcal{A}, \texttt{Var}_\mathcal{A})$. The measure of some formula ϕ of LTL$^{\text{mem}}$, written μ_ϕ, is $\sup\{\mu_\mathcal{A} : \mathcal{A} \text{ occurs in } \phi\}$.

Definition 1. *Given a measure $\mu = (m, \epsilon, w, X, Y)$, we write $\mathcal{T}_\mu$ to denote the finite set of test formulae ψ of the grammar:*

$$e ::= \langle \texttt{x}, u \rangle \mid \texttt{null} \qquad\qquad f ::= e + i$$
$$\psi ::= f \overset{l}{\hookrightarrow} e \mid \texttt{alloc}\ f \mid e = e' \mid \texttt{size} \geq k$$

with $u \leq m$, $i \in \epsilon$, $l \in X$, $k < w$ and $\texttt{x} \in Y$.

Observe that the cardinal of $\mathcal{T}_{\mu_\phi}$ is polynomial in $|\phi|$. The variable $\langle \texttt{x}, u \rangle$ will be used in the subsequent developments to deal with the interpretation of the term $\mathsf{X}^u\texttt{x}$ in the formulae of the temporal logic. Given a measure $\mu = (m, \epsilon, w, X, Y)$ and a memory state (s, h), we write $Abs_\mu(s, h) = \{\mathcal{A} \in \mathcal{T}_\mu : (s, h) \models_{\text{SL}} \mathcal{A}\}$ to denote the abstraction of (s, h) wrt μ. Given a measure μ and two memory states (s, h) and (s', h'), we write $(s, h) \simeq_\mu (s', h')$ iff $Abs_\mu(s, h) = Abs_\mu(s', h')$, that is, formulae in $\mathcal{T}_\mu$ cannot distinguish the two memory states.

The proof of Lemma 5 below is based on three technical lemmas. Before stating them and proving them, in Lemmas 2-4, we assume that the measure has $\epsilon = \{0\}$ since we are dealing with RF. Moreover, we introduce the following definition: $(m, \epsilon, w, \mathtt{Lab}_0, \mathtt{Var}_0) + (m, \epsilon, w', \mathtt{Lab}_0, \mathtt{Var}_0) = (m, \epsilon, w+w', \mathtt{Lab}_0, \mathtt{Var}_0)$, following the lattice structure of the set of measures.

Lemma 2 (Distributivity). *If $(s, h) \simeq_\mu (s', h')$, $h = h_1 * h_2$ and $\mu = \mu_1 + \mu_2$, then there are h'_1 and h'_2 such that $h' = h'_1 * h'_2$ and $(s, h_k) \simeq_{\mu_k} (s', h'_k)$ for $k \in \{1, 2\}$.*

Proof. Consider some stores, heaps and measures such that $(s, h) \simeq_\mu (s', h')$, $h = h_1 * h_2$ and $\mu = \mu_1 + \mu_2$ with $\mu = (m, \epsilon, w, \mathtt{Lab}_0, \mathtt{Var}_0)$, $\mu_1 = (m, \epsilon, w_1, \mathtt{Lab}_0, \mathtt{Var}_0)$ and $\mu_2 = (m, \epsilon, w_2, \mathtt{Lab}_0, \mathtt{Var}_0)$.

Each domain $\mathtt{dom}(h_k)$ ($k \in \{1, 2\}$) can be divided in the disjoint parts $\mathtt{dom}(h_k) \cap \mathtt{Im}(s)$ and $\mathtt{dom}(h_k) \backslash \mathtt{Im}(s)$. Each heap h'_k is defined using the following sets: $\mathtt{dom}(h'_1) \cap \mathtt{Im}(s) = S_1$, $\mathtt{dom}(h'_2) \cap \mathtt{Im}(s) = S_2$, $\mathtt{dom}(h_1) \backslash \mathtt{Im}(s) = C_1$ and $\mathtt{dom}(h_2) \backslash \mathtt{Im}(s) = C_2$.

Let $V_1, \ldots, V_a$ be the equivalence classes for the binary relation $\sim_{s'}$ defined over variables from $\{\langle \mathtt{y}, u \rangle : \mathtt{y} \in \mathtt{Var}_0, 0 \leq u \leq m\}$ as follows: $\mathtt{x} \sim_{s'} \mathtt{y}$ iff $s'(\mathtt{x}) = s'(\mathtt{y})$. Let $i_1, \ldots, i_a$ be the corresponding images through s and $i'_1, \ldots, i'_a$ be the corresponding images through s'. Let $u_1, \ldots, u_{a_1}$ be the indices such that $i_{u_n} \in \mathtt{dom}(h_1)$ and $v_1, \ldots, v_{a_2}$ be the indices such that $i_{v_n} \in \mathtt{dom}(h_2)$. Then, we define $S_1 = \{i'_{u_1}, \ldots, i'_{u_{a_1}}\}$ and $S_2 = \{i'_{v_1}, \ldots, i'_{v_{a_2}}\}$. Because $h_1 \perp h_2$, $\mathtt{dom}(h') \cap \mathtt{Im}(s') = S_1 \uplus S_2$ (disjoint union).

Now, we shall separate the set $\mathtt{dom}(h') \backslash \mathtt{Im}(s')$ into two parts A_1 and A_2. Let $B = \mathtt{dom}(h) \backslash \mathtt{Im}(s)$, $B_1 = \mathtt{dom}(h_1) \backslash \mathtt{Im}(s)$ and $B_2 = \mathtt{dom}(h_2) \backslash \mathtt{Im}(s)$. For $k \in \{1, 2\}$, $|B_k| = |\mathtt{dom}(h_k)| - |S_k|$. A_1 and A_2 contain respectively $|A_1|$ and $|A_2|$ random elements of $\mathtt{dom}(h') \backslash \mathtt{Im}(s')$ so that $|A_k| = |\mathtt{dom}(h_k)| - |S_k|$. In order to select the elements of A_1 and A_2, we distinguish four cases according to the respective sizes of $|\mathtt{dom}(h_1)|$ and $|\mathtt{dom}(h_2)|$.

Case 1: $|\mathtt{dom}(h)| < w$.
Consequently $|\mathtt{dom}(h')| = |\mathtt{dom}(h)|$ and $\mathtt{dom}(h') \backslash \mathtt{Im}(s')$ can be divided into two parts A_1, A_2 such that $A_1 \uplus A_2 = \mathtt{dom}(h') \backslash \mathtt{Im}(s')$, $|A_1| = |\mathtt{dom}(h_1)| - |S_1|$ and $|A_2| = |\mathtt{dom}(h_2)| - |S_2|$.

Case 2: $|\mathtt{dom}(h)| \geq w$.

Case 2.1: $|\mathtt{dom}(h_1)| \geq w_1$ and $|\mathtt{dom}(h_2)| \geq w_2$.
Consequently $|\mathtt{dom}(h')| \geq w$. There exist A_1, A_2 such that $A_1 \uplus A_2 = \mathtt{dom}(h') \backslash \mathtt{Im}(s')$, $|A_1| = |\mathtt{dom}(h_1)| - |S_1| \geq w_1$ and $|A_2| = |\mathtt{dom}(h_2)| - |S_2| \geq w_2$.

Case 2.2: $|\mathtt{dom}(h_1)| < w_1$ and $|\mathtt{dom}(h_2)| \geq w_2$.
There exist A_1, A_2 such that $A_1 \uplus A_2 = \mathtt{dom}(h') \backslash \mathtt{Im}(s')$, $|A_1| = |\mathtt{dom}(h_1)| - |S_1|$ and $|A_2| = |\mathtt{dom}(h_2)| - |S_2| \geq w_2$.

Case 2.3: $|\mathtt{dom}(h_1)| \geq w_1$ and $|\mathtt{dom}(h_2)| < w_2$.
Analogous to the Case 2.2.

The heap h'_1 is defined as $h'_{|A_1 \cup S_1}$ and the heap h'_2 is defined as $h'_{|A_2 \cup S_2}$. Since A_1, A_2, S_1 and S_2 are disjoint sets, we get that $A_1 \cup S_1$ and $A_2 \cup S_2$ are disjoint. Moreover, $A_1 \cup A_2 \cup S_1 \cup S_2 = \mathtt{dom}(h')$. So $h' = h'_1 * h'_2$.

It remains to show that for $k \in \{1,2\}$, $(s,h_k) \simeq_{\mu_k} (s',h'_k)$.

If $\texttt{size} \geq \kappa \in Abs_\mu(s,h_k)$, then $|\texttt{dom}(h_k)| \geq \kappa$. If $|\texttt{dom}(h_k)| \geq w_k$ then $|\texttt{dom}(h'_k)| = |A_k \cup S_k| = |A_k| + |S_k| \geq |\texttt{dom}(h_k)| - |S_k| + |S_k| \geq w_k$. If $|\texttt{dom}(h_k)| < w_k$, then $|\texttt{dom}(h'_k)| = |A_k \cup S_k| = |A_k| + |S_k| = |B_k| + |\texttt{dom}(h_k) \cap \texttt{Im}(s)| \geq |(\texttt{dom}(h_k) \backslash \texttt{Im}(s)) \cup (\texttt{dom}(h_k) \cap \texttt{Im}(s))| \geq |\texttt{dom}(h_k)| \geq \kappa$. In both cases, $\texttt{size} \geq \kappa \in Abs_\mu(s',h'_k)$.

If $\mathcal{A} = e_1 + i \overset{l}{\hookrightarrow} e_2 \in Abs_\mu(s,h_k)$, then $h_k(s(e_1+i))(l) = s(e_2)$. Consequently, $h(s(e_1+i))(l) = s(e_2)$. Hence $\mathcal{A} \in Abs_\mu(s,h)$, $\mathcal{A} \in Abs_\mu(s',h')$ and $h'(s'(e_1+i))(l) = s'(e_2)$. Since $s(e_1) \in \texttt{dom}(h_k)$, we have $s'(e_1+i) \in \texttt{dom}(h'_k)$. Hence, $h'_k(s'(e_1+i))(l) = h'(s'(e_1+i))(l) = s'(e_2)$. We conclude that $\mathcal{A} \in Abs_\mu(s',h'_k)$.

Preservation of test formulae about allocation and equality are preserved.

We have proved that $Abs_\mu(s,h_k) \subseteq Abs_\mu(s',h'_k)$. The proof for the other inclusion is similar. □

Lemma 3. *If $(s,h) \simeq_\mu (s',h')$ then for all $h_0 \perp h$, there is $h'_0 \perp h'$ such that $(s,h_0) \simeq_\mu (s',h'_0)$.*

Proof. Lemma 3 is actually a restriction of forthcoming Lemma 7 in which the constraint about size is removed.

Lemma 4 (Congruence). *If $(s,h_0) \simeq_\mu (s',h'_0)$, $h_0 \perp h_1$, $h'_0 \perp h'_1$ and $(s,h_1) \simeq_\mu (s',h'_1)$, then $(s,h_0 * h_1) \simeq_\mu (s',h'_0 * h'_1)$.*

Proof. Assume $(s,h_0) \simeq_\mu (s',h'_0)$ and $(s,h_1) \simeq_\mu (s',h'_1)$ with $h_0 \perp h_1$, $h'_0 \perp h'_1$ for some measure $\mu = (m,\epsilon,w,\texttt{Lab}_0,\texttt{Var}_0)$. We shall show that $(s,h_0*h_1) \simeq_\mu (s',h'_0*h'_1)$.

By symmetry of $\simeq_\mu$, it is sufficient to prove that $Abs_\mu(s,h_0*h_1) \subseteq Abs_\mu(s',h'_0*h'_1)$. Let $\mathcal{A} \in Abs_\mu(s,h_0*h_1)$. We make a case analysis according to the form of $\mathcal{A}$.

- If $\mathcal{A} = \texttt{size} \geq k$, then $k \leq |\texttt{dom}(h_0 * h_1)|$. We want to show that $k \leq |\texttt{dom}(h'_0 * h'_1)|$ which implies that $\mathcal{A} \in Abs_\mu(s,h'_0 * h'_1)$.
  - If $|\texttt{dom}(h_1)| \geq w$ or $|\texttt{dom}(h_0)| \geq w$, then $|\texttt{dom}(h'_1)| \geq w$ or $|\texttt{dom}(h'_0)| \geq w$, respectively. So $|\texttt{dom}(h'_0 * h'_1)| \geq w$ and $k \leq |\texttt{dom}(h'_0 * h'_1)|$.
  - If $|\texttt{dom}(h_1)| \leq w$ and $|\texttt{dom}(h_0)| \leq w$ then $|\texttt{dom}(h_0 * h_1)| = |\texttt{dom}(h_1)| + |\texttt{dom}(h_0)| = |\texttt{dom}(h'_1)| + |\texttt{dom}(h'_0)| = |\texttt{dom}(h'_0 * h'_1)|$. So $k \leq |\texttt{dom}(h'_0 * h'_1)|$.
- If $\mathcal{A}$ is $e = e'$ then $[\![e]\!]_s = [\![e']\!]_s$. So $\mathcal{A} \in Abs_\mu(s,h_1)$, which is equivalent to $\mathcal{A} \in Abs_\mu(s',h'_1)$. Therefore $[\![e]\!]_{s'} = [\![e']\!]_{s'}$ and $\mathcal{A} \in Abs_\mu(s',h'_0 * h'_1)$.
- If $\mathcal{A} = e \overset{l}{\hookrightarrow} e'$ then $(h_0 * h_1)([\![e]\!]_s)(l) = [\![e']\!]_s$. Hence there is $k \in \{0,1\}$ such that $h_k([\![e]\!]_s)(l) = [\![e']\!]_s$, $h'_k([\![e]\!]_{s'})(l) = [\![e']\!]_{s'}$ and $(h'_1 * h'_0)([\![e]\!]_{s'})(l) = s'[\![e']\!]_{s'}$. So $\mathcal{A} \in Abs_\mu(s,h'_1 * h'_0)$.
- If $\mathcal{A} = \texttt{alloc}\, e$ then $[\![e]\!]_s \in \texttt{dom}(h_0 * h_1)$. Hence there is $k \in \{0,1\}$ such that $[\![e]\!]_s \in \texttt{dom}(h_k)$. Consequently, $[\![e]\!]_{s'} \in \texttt{dom}(h'_k)$ and $[\![e]\!]_{s'} \in \texttt{dom}(h'_0 * h'_1)$ which entails $\mathcal{A} \in Abs_\mu(s,h'_0 * h'_1)$. □

Lemma 5 below states that our abstraction is correct for the fragments CL and RF.

Lemma 5 (Soundness of abstraction for SL**).** *Let (s,h) and (s',h') be two memory states such that $(s,h) \simeq_{\mu} (s',h')$ [resp. $(s,h) \simeq_{\mu[w \leftarrow 0]} (s',h')$]. For any state formula $\mathcal{A}$ such that $\mu_{\mathcal{A}} \leqslant \mu$ and $\mathcal{A}$ belongs to* RF *[resp.* CL*], we have $(s,h) \models_{SL} \mathcal{A}$ iff $(s',h') \models_{SL} \mathcal{A}$.*

Proof. The proof of Lemma 5 for the classical fragment is rather straightforward. Indeed any state formula is a Boolean combination of test formulae. Concerning the record fragment, because of the presence of separation operators the proof deserves more effort. Suppose that $(s,h) \simeq_{\mu} (s',h')$ and $\mathcal{A} \in$ RF such that $\mu_{\mathcal{A}} \leq \mu$. By structural induction, we show that $(s,h) \models_{\mathrm{SL}} \mathcal{A}$ iff $(s',h') \models_{\mathrm{SL}} \mathcal{A}$.

- The base case when $\mathcal{A}$ has one of the forms $e = e'$, $e \overset{l}{\hookrightarrow} e'$ and `emp` is by an easy verification.
- Similarly, in the induction step, the cases when the outermost connective is Boolean are straightforward.
- Assume that $(s,h) \models_{\mathrm{SL}} \mathcal{A}$ with $\mathcal{A} = \mathcal{B}_1 * \mathcal{B}_2$. Then there are h_1 and h_2 such that $h = h_1 * h_2$ and $(s,h_k) \models_{\mathrm{SL}} \mathcal{B}_k$ for $k = 1,2$. As $\mu \geqslant \mu_{\mathcal{A}}$ and $\mu_{\mathcal{A}} \geqslant \mu_{\mathcal{B}_1} + \mu_{\mathcal{B}_2}$, there are μ_1 and μ_2 such that $\mu_k \geqslant \mu_{\mathcal{B}_k}$ and $\mu_1 + \mu_2 = \mu$. By the distributivity lemma, there are h'_1 and h'_2 such that $h' = h'_1 * h'_2$ and $(s,h_k) \simeq_{\mu_k} (s',h'_k)$ for $k = 1,2$. By the induction hypothesis, for $k \in \{1,2\}$, $(s',h'_k) \models_{\mathrm{SL}} \mathcal{B}_k$. Semantics of the separation operator $*$ guarantees that $(s',h') \models_{\mathrm{SL}} \mathcal{A}$.
- Finally, assume $\mathcal{A} = \mathcal{B}_1 \mathrel{-\!\!*} \mathcal{B}_2$. Let $h'_1 \perp h'$ be such that $(s',h'_1) \models_{\mathrm{SL}} \mathcal{B}_1$. By Lemma 3, there is a heap h_1 such that $(s,h_1) \simeq_{\mu} (s',h'_1)$ and $h_1 \perp h$, and so $(s,h_1) \models_{\mathrm{SL}} \mathcal{B}_1$ by the induction hypothesis. Then $(s, h * h_1) \models_{\mathrm{SL}} \mathcal{B}_2$, and by the congruence lemma, $(s', h' * h'_1) \models_{\mathrm{SL}} \mathcal{B}_2$. Hence $(s',h') \models_{\mathrm{SL}} \mathcal{B}_1 \mathrel{-\!\!*} \mathcal{B}_2$.

Note that we can extend this result to the whole SL by considering test formulae of the form $e + i = e' + j$.

3.2 Complexity of reasoning tasks for SL

In this section, we show that model-checking, satisfiability and validity, for SL, are PSPACE-complete. We use the abbreviations mc(SL), sat(SL) and val(SL) for the respective problems. These abbreviations are extended to any fragment of separation logic, for instance sat(RF) is the satisfiability problem for the record fragment.

PSPACE-hardness of mc(LF) and sat(LF) is a consequence of [CYO01, Sect. 5.2]. As SL strictly contains LF, this entails the PSPACE-hardness for mc(SL) and sat(SL). Since SL is closed under negation, PSPACE-completeness of val(SL) will follow from PSPACE-completeness of sat(SL).

In order to show that mc(SL) and sat(SL) are in PSPACE, we establish the lemmas below. The following lemma 6 establishes a reduction from mc(SL) to mc(RF), so that we only need to consider RF in order to find the complexity of model-checking. Then, in lemma 7, we will provide a small model property for RF, which will allow us to prove that mc(RF) is in PSPACE (see Lemma 8). Finally, we solve the problem of satisfiablility thanks to lemma 10, which entails a reduction from sat(SL) to mc(SL).

Lemma 6. *There is a logspace reduction from mc(*SL*) to mc(*RF*).*

Proof. Let $t(\mathcal{A})$ be the formula obtained from $\mathcal{A}$ in SL by replacing each occurrence of $\mathtt{x} + i \overset{l}{\hookrightarrow} e$ by $\langle \mathtt{x}, i \rangle \overset{l}{\hookrightarrow} e$. The formula $t(\mathcal{A})$ belong to RF. Given a store s, we write $t(s)$ to denote the store such that $t(s)(\langle \mathtt{x}, i \rangle) = s(\mathtt{x}) + i$. One can show that for every heap h, $(s, h) \models_{\mathrm{SL}} \mathcal{A}$ iff $(t(s), h) \models_{\mathrm{SL}} t(\mathcal{A})$. The proof is by structural induction on $\mathcal{A}$. □

We need to establish a quite technical lemma. Below, the size of a measure $\mu = (m, \{0\}, w, X, Y)$ is in

$$\mathcal{O}(m + w + |X| \times log(|X|) + |Y| \times log(|Y|)).$$

Lemma 7. *Let $\mu = (m, \{0\}, w, X, Y)$ be a measure, and l_0 be a label that does not belong to the finite set of labels X. If $(s, h) \simeq_\mu (s', h')$ and $h_0 \bot h$ is a heap then there is a heap h'_0 such that*

- $h'_0 \perp h'$,
- $(s, h_0) \simeq_\mu (s', h'_0)$,
- $|\mathtt{dom}(h'_0)| \leq max(w, (m+1) \times |Y|)$,
- $max\ \mathtt{dom}(h'_0) \cup \mathrm{Im}^2(h'_0) \leq max\ (\{s'(\langle \mathtt{x}, i \rangle) : \mathtt{x} \in Y, 0 \leq i \leq m\} \cup \mathtt{dom}(h')) + w$, *where $\mathrm{Im}^2(h)$, for a given heap h, is the set of natural numbers i such that there are i' and l for which $h(i')(l) = i$.*
- *for all $n \in \mathtt{dom}(h'_0)$, $\{l : h(n)(l)$ is defined$\} \subseteq X \uplus \{l_0\}$.*

The heap h'_0 is said to be a small disjoint heap wrt μ and (s', h') and it can be encoded in polynomial space in $|\mu| + |h_0| + |(s', h')|$.

Proof. Assume that $(s, h) \simeq_\mu (s', h')$ and $h_0 \bot h$. We introduce two disjoint heaps h_{01} and h_{02} such that $\mathtt{dom}(h_{01}) = \mathtt{Im}(s) \cap \mathtt{dom}(h_0)$ and $\mathtt{dom}(h_{02}) = \mathtt{dom}(h_0) \backslash \mathtt{Im}(s)$. Observe that $h_0 = h_{01} * h_{02}$. We define the heap h'_0 as the disjoint union $h'_{01} * h'_{02}$ where h'_{01} and h'_{02} are defined so as to satisfy $\mathtt{dom}(h'_{01}) = \mathtt{Im}(s') \cap \mathtt{dom}(h'_0)$ and $\mathtt{dom}(h'_{02}) = \mathtt{dom}(h'_0) \backslash \mathtt{Im}(s')$. Let $\mathbf{V}$ be the set of variables $\{\langle \mathtt{y}, i \rangle : \mathtt{y} \in Y, 0 \leq i \leq m\}$.

- In order to define h'_{01}, let $V_1, \ldots, V_a$ be the equivalence classes for the relation $s(\mathtt{x}) = s(\mathtt{x}')$ among variables $\mathtt{x}, \mathtt{x}'$ in $\mathbf{V}$. The classes are the same for the equivalence relation defined by s' since $(s, h) \simeq_\mu (s', h')$. For each class V_k, let i_k be the image of the variables of V_k through s, and i'_k through s'. Then, for all $k \leq a$ and $l \in \mathtt{dom}(h_{01}(i_k))$, the heap h'_{01} is defined as follows:
  - if $l \notin X$, then $h'_{01}(i'_k)(l_0) = nil$ and $h'_{01}(i'_k)(l)$ is undefined,
  - if $l \in X$ and $h_{01}(i_k)(l) = i_n$ for some n, then $h'_{01}(i'_k)(l) = i'_n$,
  - if $l \in X$ and $h_{01}(i_k)(l) \neq i_n$ for all n, then $h'_{01}(i'_k)(l) = nil$.

  The domain of h'_{01} is included in $\mathtt{Im}(s')$, since $\mathtt{Im}(s'_{|\mathbf{V}}) = \{i'_1, \ldots, i'_a\}$.
- In order to define h'_{02}, let $b = max(0, min\{|\mathtt{dom}(h_{02})|, w - |\mathtt{dom}(h_{01})|\})$ and $j'_1, \ldots, j'_b$ be the b smallest indices distinct from $i'_1, \ldots, i'_a$ and not in $\mathtt{dom}(h')$. Hence, when $|\mathtt{dom}(h_{01})| \geq w$, there are no such indices. Otherwise, for all $k \leq b$, we define $h'_{02}(j'_k)(l_0) = nil$. We set $\mathtt{dom}(h'_{02}) = \{j'_1, \ldots, j'_b\}$.

As announced, we define h'_0 as the heap $h'_{01} * h'_{02}$. Let us show that the heap h'_0 has all the desired properties.

- Let us check that $h' \perp h'_0$. First, $h' \perp h'_{01}$ since $h \perp h_{01}$. Second, $h' \perp h'_{02}$ by construction.
- Let us check that $(s, h_0) \simeq_\mu (s', h'_0)$. We proceed by a case analysis on the form of the test formulae.
  $(\mathtt{x} = \mathtt{y})$ Since $(s, h) \simeq_\mu (s', h')$, $s(\mathtt{x}) = s(\mathtt{y})$ iff $s'(\mathtt{x}) = s'(\mathtt{y})$.
  $(\mathtt{size} \ \geq \ k)$ We first note: $|\mathtt{dom}(h_{01})| = |\mathtt{dom}(h'_{01})|$. If $|\mathtt{dom}(h_{02})| \geq w - |\mathtt{dom}(h_{01})|$, then $|\mathtt{dom}(h'_{02})| \geq w - |\mathtt{dom}(h'_{01})|$, so both $|\mathtt{dom}(h_0)| \geq w$ and $|\mathtt{dom}(h'_0)| \geq w$, and all of the $\mathtt{size} \ \geq \ k$ formulae are satisfied by both models. If $|\mathtt{dom}(h_{02})| < w$, then $|\mathtt{dom}(h_0)| = |\mathtt{dom}(h'_0)|$, and for all k, $\mathtt{size} \ \geq \ k \in Abs_\mu(s, h_0)$ iff $\mathtt{size} \ \geq \ k \in Abs_\mu(s', h'_0)$.
  $(e_1 \overset{l}{\hookrightarrow} e_2)$ If $e_1 \overset{l}{\hookrightarrow} e_2 \in Abs_\mu(s, h_0)$, then $e_1 \in V_k$ for some k, $e_2 \in V_n$ for some n, $l \in X$, and $h_0(i_k)(l) = i_n$. Consequently, $h'_0(i'_k)(l) = i'_n$ and $s', h'_0 \models_{\mathrm{SL}} e_1 \overset{l}{\hookrightarrow} e_2$.
  $(\mathtt{alloc}\ e_1)$ If $\mathtt{alloc}\ e_1 \in Abs_\mu(s, h_0)$, then $e_1 \in V_k$ for some k, and $i_k \in \mathtt{dom}(h_0)$. Consequently, $i'_k \in \mathtt{dom}(h'_0)$ and $(s', h'_0) \models_{\mathrm{SL}} \mathtt{alloc}\ e_1$.
  Therefore (s, h_0) and (s', h'_0) have the same abstraction.
- Let us check that $|\mathtt{dom}(h'_0)| \leq max(w, (m+1) \times |Y|)$. Since $a \leq (m+1) \times |Y|$, if $|\mathtt{dom}(h'_{01})| \geq w$, then h'_{02} is the empty heap and $|\mathtt{dom}(h'_0)| \leq a$. Otherwise, $|\mathtt{dom}(h'_{01})| + |\mathtt{dom}(h'_{02})| \leq w$. Consequently, $|\mathtt{dom}(h'_0)| \leq max(w, (m+1) \times |Y|)$.
- Let us check that $max\ \mathtt{dom}(h'_0) \cup \mathrm{Im}^2(h'_0) \leq max\ \{s'(\langle \mathtt{x}, i \rangle) : \langle \mathtt{x}, i \rangle \in \mathbf{V}\} \cup \mathtt{dom}(h') + w$. We have chosen the domain and image of h'_{01} to be included in the image of s' plus nil, and therefore $\mathtt{dom}(h'_{01})$ satisfies the above condition. The image of h'_{02} is $\{nil\}$. The domain of h_{02} is composed of the smallest integers which neither belong to $\{s'(\langle \mathtt{x}, i \rangle) : \langle \mathtt{x}, i \rangle \in \mathbf{V}\}$, nor to $\mathtt{dom}(h')$. As this set has less than w elements, its bigger element is less than the w^{th} element of these smallest integers, which is bounded by $max\ \{s'(\langle \mathtt{x}, i \rangle) : \mathtt{x} \in Y, 0 \leq i \leq w\} \cup \mathtt{dom}(h') + w$.
- Let us check that for every $n \in \mathtt{dom}(h'_0)$, $\{l : h(n)(l) \text{ is defined}\} \subseteq X \uplus \{l_0\}$. This condition is satisfied by construction of h'_{01} and h'_{02}.

Lemma 8. *mc(*RF*) is in* PSPACE.

Proof. The algorithm is described in Figure 1.

First of all, the algorithm can be implemented in polynomial space since the quantifications are over sets of exponential size in $|\mathcal{A}| + |(s, h)|$, and the recursion depth is linear in $\mathcal{A}$. Hence there is only a linear amount of polynomial increases of the size of h in the $-\!\!*$ recursive calls. It remains to show that the algorithm is correct, that given $\mathcal{A}$ with $\mu_{\mathcal{A}} \leq \mu$, $(s, h) \models_{\mathrm{SL}} \mathcal{A}$ iff $\mathrm{MC}((s, h), \mathcal{A}, \mu)$ returns $\top$. The only point to check in the proof by structural induction is the case when the outermost connective is the operator $-\!\!*$. Whenever $(s, h) \not\models_{\mathrm{SL}} \mathcal{A}_1 -\!\!* \mathcal{A}_2$, there is a heap $h_0 \perp h$ such that $(s, h_0) \models_{\mathrm{SL}} \mathcal{A}_1$ and $(s, h * h_0) \not\models_{\mathrm{SL}} \mathcal{A}_2$. By Lemma 7 with $(s', h') = (s, h)$, there is a small disjoint heap h'_0 wrt μ and (s, h) such that $(s, h'_0) \simeq_\mu (s, h_0)$. Since the measure of $\mathcal{A}_1$ is lower than μ, Lemma

```
function MC((s,h),A,μ)

(base-cases) If $\mathcal{A}$ is atomic, then return $(s,h) \models_{\mathrm{SL}} \mathcal{A}$;
(Boolean-cases) If $\mathcal{A} = \mathcal{A}_1 \wedge \mathcal{A}_2$, then return $(\mathrm{MC}((s,h),\mathcal{A}_1,\mu)$ and $\mathrm{MC}((s,h),\mathcal{A}_2,\mu))$;
    Other Boolean operators are treated analogously.
(∗ case) If $\mathcal{A} = \mathcal{A}_1 * \mathcal{A}_2$, then return $\perp$ if there are no $h_1, h_2$ such that $h = h_1 * h_2$ $(\mathrm{MC}((s,h_1),\mathcal{A}_1,\mu)$
    and $\mathrm{MC}((s,h_2),\mathcal{A}_2,\mu))$;
(−∗ case) If $\mathcal{A} = \mathcal{A}_1 {-\!\!*} \mathcal{A}_2$, then return $\perp$ if for some small disjoint heap $h'$ wrt $\mu$ and $(s,h)$ verifying
    $\mathrm{MC}((s,h'),\mathcal{A}_1,\mu)$, we have not $\mathrm{MC}((s,h*h'),\mathcal{A}_2,\mu)$;

Return $\top$;
```

Fig. 1. Model-checking algorithm

5 entails $(s,h'_0) \models_{\mathrm{SL}} \mathcal{A}_1$. Also, by Lemma 4, $(s,h*h'_0) \not\models_{\mathrm{SL}} \mathcal{A}_2$. Consequently, $(s,h) \not\models_{\mathrm{SL}} \mathcal{A}_1 {-\!\!*} \mathcal{A}_2$ iff there is a *small* heap h'_0 such that $(s,h'_0) \models_{\mathrm{SL}} \mathcal{A}_1$ and $(s,h*h'_0) \not\models_{\mathrm{SL}} \mathcal{A}_2$. □

We need another technical lemma. Before stating this property, let us define, given a permutation σ and a heap h, the heap $\sigma \bullet h = \sigma \cdot (h \circ \sigma^{-1})$, where $\sigma \cdot h$ is the partial function which maps i to the partial function $\sigma \circ (h(i))$. For instance, given a label l and an address i, we have $(\sigma \bullet h)(i)(l) = \sigma(h(\sigma^{-1}(i))(l))$. This operation allows us to rename all the addresses according to the permutation: the memory graph keeps the same shape, but vertices are placed on different addresses.

Lemma 9. *Let $\mathcal{A}$ be a state formula of* SL *with measure $\mu = (0,\epsilon,w,X,Y)$ and (s,h) be a memory state. For all permutations σ of $\mathbb{N}$ such that for all $(\mathtt{x},i) \in Y \times \epsilon$, $\sigma(s(\mathtt{x})+i) = \sigma(s(\mathtt{x}))+i$, we have $(s,h) \models_{SL} \mathcal{A}$ iff $(\sigma \circ s, \sigma \bullet h) \models_{SL} \mathcal{A}$.*

Proof. We will have to use two properties about $\bullet$, which can easily be checked:

- for all permutations σ and disjoint heaps h_1 and h_2, $\sigma \bullet (h_1 * h_2) = (\sigma \bullet h_1) * (\sigma \bullet h_2)$.
- for all permutations σ and heaps h, $\sigma^{-1} \bullet (\sigma \bullet h) = h$.

It is sufficient to show one direction of the equivalence since the other direction is obtained by application of the first one with the store $\sigma \circ s$ and the well-defined inverse bijection σ^{-1}. Indeed, for all $(\mathtt{x},i) \in Y \times \epsilon$, $\sigma^{-1}((\sigma \circ s)(\mathtt{x}) + i) = \sigma^{-1}((\sigma \circ s)(\mathtt{x}))+i$. Let $\mathcal{A}$ be a formula, μ be a measure greater than $\mu_{\mathcal{A}}$, s be a store and h be a heap. Assume that $(s,h) \models_{\mathrm{SL}} \mathcal{A}$. We show that $(\sigma \circ s, \sigma \bullet h) \models_{\mathrm{SL}} \mathcal{A}$. We are going to prove this by induction on $\mathcal{A}$.

If $\mathcal{A}$ is an atomic formula, then we proceed by a case analysis.

$\mathcal{A}$ is $\mathtt{x} = \mathtt{y}$: $s(\mathtt{x}) = s(\mathtt{y})$ and $\sigma \circ s(\mathtt{x}) = \sigma(s(\mathtt{x})) = \sigma(s(\mathtt{y})) = \sigma \circ s(\mathtt{y})$,

$\mathcal{A}$ is $\mathtt{x} + i \overset{l}{\hookrightarrow} \mathtt{y}$: $h(s(\mathtt{x})+i)(l) = s(\mathtt{y})$, and $\sigma \bullet h(\sigma \circ s(\mathtt{x})+i)(l) = \sigma \cdot h(\sigma^{-1}(\sigma(s(\mathtt{x}))+i))(l) = \sigma \cdot h(\sigma^{-1}(\sigma(s(\mathtt{x})+i)))(l) = \sigma \cdot h(s(\mathtt{x})+i)(l) = \sigma(h(s(\mathtt{x})+i)(l)) = \sigma(s(\mathtt{y})) = \sigma \circ s(\mathtt{y})$,

$\mathcal{A}$ is $\mathtt{emp}$: $\mathtt{dom}(\sigma \bullet h)$ is empty iff $\mathtt{dom}(h)$ is empty.

If $\mathcal{A} = \mathcal{A}_1 * \mathcal{A}_2$, then there are h_1 and h_2 such that $h = h_1 * h_2$ and $(s,h_i) \models_{\mathrm{SL}} \mathcal{A}_i$ for $i \in \{1,2\}$. The measure of each $\mu_{\mathcal{A}_i} \leqslant \mu_{\mathcal{A}} \leqslant \mu$. Then, by induction,

$(\sigma \circ s, \sigma \bullet h_i) \models_{\mathrm{SL}} \mathcal{A}_i$. Since $\sigma \bullet h = \sigma \bullet (h_1 * h_2) = (\sigma \bullet h_1) * (\sigma \bullet h_2)$, we can conclude that $(\sigma \circ s, \sigma \bullet h) \models_{\mathrm{SL}} \mathcal{A}$.

If $\mathcal{A} = \mathcal{A}_1 \mathrel{-\!\!*} \mathcal{A}_2$, then, let h_0 be a heap which is orthogonal to $\sigma \bullet h$. Assume that $(\sigma \circ s, h_0) \models_{\mathrm{SL}} \mathcal{A}_1$. Then by induction, $(\sigma^{-1} \circ (\sigma \circ s), \sigma^{-1} \bullet h) \models_{\mathrm{SL}} \mathcal{A}_1$, that is $(s, \sigma^{-1} \bullet h_0) \models_{\mathrm{SL}} \mathcal{A}_1$. So $(s, h * (\sigma^{-1} \bullet h_0)) \models_{\mathrm{SL}} \mathcal{A}_2$, and by induction $(\sigma \circ s, \sigma \bullet (h * (\sigma^{-1} \bullet h_0))) \models_{\mathrm{SL}} \mathcal{A}_2$, that is $(\sigma \circ s, (\sigma \bullet h) * (\sigma \bullet (\sigma^{-1} \bullet h_0))) \models_{\mathrm{SL}} \mathcal{A}_2$, and finally $(\sigma \circ s, (\sigma \bullet h) * h_0) \models_{\mathrm{SL}} \mathcal{A}_2$. So, $(\sigma \circ s, \sigma \bullet h) \models_{\mathrm{SL}} \mathcal{A}$.

We state below a small memory state property that happens to be central to establish the future PSPACE upper bounds.

Lemma 10 (Small memory state property). *A state formula $\mathcal{A}$ in* SL *is satisfiable iff there is a store s such that $(s, \emptyset) \models_{SL} \neg(\mathcal{A} \mathrel{-\!\!*} \perp)$ and for each variable $\mathtt{x} \in Y$, $s(\mathtt{x}) \leq (|Y| + 1) \times max\ \epsilon$, where $\emptyset$ stands for the heap with empty domain, Y is the set of variables occuring in $\mathcal{A}$, and ϵ is the set of indices i such that $\mathtt{x} + i$ occurs in $\mathcal{A}$ for some variable $\mathtt{x}$.*

Proof. First, let us notice that a state formula $\mathcal{A}$ in SL is satisfiable iff there is a store s such that $(s, \emptyset) \models_{\mathrm{SL}} \neg(\mathcal{A} \mathrel{-\!\!*} \perp)$, where $\emptyset$ is the heap with empty domain. This is straightforward by definition of $\models_{\mathrm{SL}}$. So, we only have to prove that, given $\mathcal{A}$ a state formula of SL and a store s such that $(s, \emptyset) \models_{\mathrm{SL}} \mathcal{A}$, there is a store s' such that $(s', \emptyset) \models_{\mathrm{SL}} \mathcal{A}$ and for $\mathtt{x} \in Y$, $s'(\mathtt{x}) \leq (|Y| + 1) \times max\ \epsilon$. In order to obtain these smaller models, we are going to decrease the value of the variables in several steps. Each step consists in applying a permutation to the memory graph.

Assume some given store s, such that $(s, \emptyset) \models_{\mathrm{SL}} \mathcal{A}$. Let us define one of the permutations which decrease the value of the variables through s. Let i be $max\ \epsilon$.

Let $\mathtt{x}_0$ be a dummy variable such that $s(\mathtt{x}_0) = 0$, and $\{\mathtt{x}_1, \ldots \mathtt{x}_n\}$ an enumeration of the variables occuring in $\mathcal{A}$ such that for $j \in [\![0; n-1]\!]$, $s(\mathtt{x}_j) \leq s(\mathtt{x}_{j+1})$. If there is no k such that $s(\mathtt{x}_{k+1}) \geq s(\mathtt{x}_k) + i$ then for all $\mathtt{x} \in Y$, $s(\mathtt{x}) \leq (n+1) \times i$.

Otherwise, let k be the smallest index such that $s(\mathtt{x}_{k+1}) \geq s(\mathtt{x}_k) + i$. Let $\alpha = s(\mathtt{x}_{k+1}) - (s(\mathtt{x}_k) + i)$. Then, we define σ as follows:

- If $j \leq s(\mathtt{x}_k) + i$ then $\sigma(j) = j$;
- If $s(\mathtt{x}_{k+1}) \leq j \leq s(\mathtt{x}_n) + i$, then $\sigma(j) = j - \alpha$;
- If $j \geq s(\mathtt{x}_n) + i$ then $\sigma(j) = j$;
- If $s(\mathtt{x}_k) + i < j < s(\mathtt{x}_{k+1})$ then we have to complete this function so as to obtain a bijection, $\sigma(j) = j - (s(\mathtt{x}_k) + i) + (s(\mathtt{x}_n) + i - \alpha)$.

This permutation satisfies the prerequisites of Lemma 9, and thus may be applied to $(s, \emptyset)$, which then still satisfies $\mathcal{A}$. We apply this type of permutation until there is no k such that $s(\mathtt{x}_{k+1}) \geq s(\mathtt{x}_k) + i$. So, by a simple multiplication, for all $\mathtt{x} \in Y$, $s(\mathtt{x}) \leq (n+1) \times i$, or more explicitly $s(\mathtt{x}) \leq (|Y| + 1) \times max\ \epsilon$. $\square$

Proposition 1. *The model-checking, satisfiability and validity problems for* SL *are* PSPACE*-complete.*

Proof. PSPACE-hardness results are consequences of [CYO01, Sect. 5.2]. The PSPACE upper bound for mc(SL) is a consequence of Lemmas 6 and 8. The PSPACE upper bound for sat(SL) is obtained by enumerating the small memory states of $\neg(\mathcal{A} \mathrel{-\!\!*} \perp)$ with empty heap (see Lemma 10) and then using Lemma 8.

4 Decidable Satisfiability Problems by Abstracting Computations

In this section we establish the PSPACE-completeness of the problems SAT(CL) and SAT(RF). To do so, we abstract memory states whose size is a priori unbounded by finite symbolic memory states. As usual, temporal infinity in models is handled by Büchi automata recognizing ω-sequences. We propose below an abstraction that is correct for CL (allowing pointer arithmetic) and for RF (allowing all operators from Separation Logic) taken separately but that is not exact for the full language SL.

4.1 Symbolic models

Given a measure μ, we write Σ_μ to denote the powerset of $\mathcal{T}_\mu$; Σ_μ is thought as an alphabet, and elements $a \in \Sigma_\mu$ are called *letters*. A symbolic model wrt μ is defined as an infinite sequence $\sigma \in \Sigma_\mu^\omega$. Symbolic models are abstractions of models from $\mathrm{LTL}^{\mathrm{mem}}$: given a model $\rho : \mathbb{N} \to \mathcal{S} \times \mathcal{H}$ and a measure μ, we write $Abs_\mu(\rho) : \mathbb{N} \to \Sigma_\mu$ to denote the symbolic model wrt μ such that for every t, $Abs_\mu(\rho)(t) \overset{\text{def}}{=} \{\mathcal{A} \in \mathcal{T}_\mu : \rho, t \models \mathcal{A}[\langle \mathtt{x}, u\rangle \leftarrow \mathsf{X}^u\mathtt{x}]\}$.

To a letter a, we associate the formula $\mathcal{A}_a = \bigwedge_{\mathcal{A} \in a} \mathcal{A} \wedge \bigwedge_{\mathcal{A} \notin a} \neg\mathcal{A}$. For all symbolic models σ and formulae ϕ such that $\mu_\phi \leq \mu$, we define the symbolic satisfaction relation $\sigma, t \models_\mu \phi$ as the satisfaction relation for models except for the clause about atomic subformulae that is updated as follows: $\sigma, t \models_\mu \mathcal{A}$ iff $\models_{\mathrm{SL}} \mathcal{A}_{\sigma(t)} \Rightarrow \mathcal{A}[\mathsf{X}^u\mathtt{x} \leftarrow \langle \mathtt{x}, u\rangle]$. We write $\mathrm{L}^\mu(\phi)$ to denote the set of symbolic models σ wrt μ such that $\sigma, 0 \models_\mu \phi$. As a corollary of Lemma 5, we get a soundness result for our abstraction:

Proposition 2. *Let ϕ be a formula of $\mathrm{LTL}^{\mathrm{mem}}(\mathrm{RF})$ [resp. of $\mathrm{LTL}^{\mathrm{mem}}(\mathrm{CL})$] and $\mu_\phi \leq \mu$. For any model ρ, we have that $\rho \models \phi$ iff $Abs_\mu(\rho) \models_\mu \phi$ [resp. $Abs_{\mu[w\leftarrow 0]}(\rho) \models_\mu \phi$].*

Proof. We treat the case $\phi \in \mathrm{LTL}^{\mathrm{mem}}(\mathrm{RF})$ (for the case $\phi \in \mathrm{LTL}^{\mathrm{mem}}(\mathrm{CL})$, replace below μ by $\mu[w \leftarrow 0]$). Suppose that $\rho, t \models \mathcal{B}$ for an atomic formula $\mathcal{A}$ of $\mathrm{LTL}^{\mathrm{mem}}$. By definition, $Abs_\mu(\rho)(t) \overset{\text{def}}{=} \{\mathcal{B} \in \mathcal{T}_\mu : \rho, t \models \mathcal{B}[\langle \mathtt{x}, u\rangle \leftarrow \mathsf{X}^u\mathtt{x}]\}$. Let us show that $\models_{\mathrm{SL}} \mathcal{A}_{Abs_\mu(\rho)(t)} \Rightarrow \mathcal{B}[\mathsf{X}^u\mathtt{x} \leftarrow \langle \mathtt{x}, u\rangle]$. If for some memory state $(s, h) \models_{\mathrm{SL}} \mathcal{A}_{Abs_\mu(\rho)(t)}$, then by Lemma 5, $(s, h) \models_{\mathrm{SL}} \mathcal{B}[\mathsf{X}^u\mathtt{x} \leftarrow \langle \mathtt{x}, u\rangle]$. Suppose now that $Abs_\mu(\rho), t \models_\mu \mathcal{B}$. Hence, $\models_{\mathrm{SL}} \mathcal{A}_{Abs_\mu(\rho)(t)} \Rightarrow \mathcal{B}[\mathsf{X}^u\mathtt{x} \leftarrow \langle \mathtt{x}, u\rangle]$. Since $\rho, t \models_{\mathrm{SL}} \mathcal{A}_{Abs_\mu(\rho)(t)}[\langle \mathtt{x}, u\rangle \leftarrow \mathsf{X}^u\mathtt{x}]$, then $\rho, t \models (\mathcal{B}[\mathsf{X}^u\mathtt{x} \leftarrow \langle \mathtt{x}, u\rangle])[\langle \mathtt{x}, u\rangle \leftarrow \mathsf{X}^u\mathtt{x}]$, that is $\rho, t \models \mathcal{B}$. The induction step for the cases with Boolean and temporal operators is then by an easy verification.

Note that Abs_{μ} is not surjective; we note L^{μ}_{sat} the set of symbolic models wrt μ that are abstractions of some model of $\mathrm{LTL}^{\mathrm{mem}}$. Consequently, ϕ in $\mathrm{LTL}^{\mathrm{mem}}(\mathrm{RF})$ is satisfiable iff $\mathrm{L}^{\mu_{\phi}}(\phi) \cap \mathrm{L}^{\mu_{\phi}}_{sat}$ is nonempty.

4.2 ω-regularity and PSPACE upper bound

In order to show that SAT(RF) and SAT(CL) are in PSPACE we shall explain why testing the nonemptiness of $\mathrm{L}^{\mu_{\phi}}(\phi) \cap \mathrm{L}^{\mu_{\phi}}_{sat}$ can be done in PSPACE. Below we treat explicitly the case for RF. For CL, replace every occurrence of μ_{ϕ} by $\mu_{\phi}[w \leftarrow 0]$ and every occurrence of μ by $\mu[w \leftarrow 0]$. To do so, we show that each language can be recognized by an exponential-size Büchi automaton satisfying the good properties to establish the PSPACE upper bound. If $\mathbb{A}$ is a Büchi automaton, we note $\mathrm{L}(\mathbb{A})$ the language recognized by $\mathbb{A}$. Following [VW94,DD07], let $\mathbb{A}$ be the generalized Büchi automaton defined by the structure $(\Sigma, Q, \delta, I, \mathcal{F})$ such that $(\mu \geq \mu_{\phi})$:

- Q is the set of so-called atoms of ϕ, that are sets of temporal formulae included in the so-called closure set $cl(\phi)$ (see [VW94]),
- $I = \{X \in Q : \phi \in X\}$.
- $\Sigma = \Sigma_{\mu}$.
- $X \xrightarrow{a} Y$ iff 1. for every atomic formula $\mathcal{A}$ of X, $\models_{\mathrm{SL}} \mathcal{A}_a \Rightarrow \mathcal{A}[\mathsf{X}^u\mathtt{x} \leftarrow \langle \mathtt{x}, u\rangle]$.
  2. for every $\mathsf{X}\phi' \in cl(\phi)$, $\mathsf{X}\phi' \in X$ iff $\phi' \in Y$.
- Let $\{\phi_1 \mathsf{U} \phi'_1, \ldots, \phi_n \mathsf{U} \phi'_n\}$ be the set of until formulae in $cl(\phi)$. Let $\mathcal{F}$ be equal to $\{F_1, \ldots, F_n\}$ where $F_i = \{X \in Q : \phi_i \mathsf{U} \phi'_i \notin X \text{ or } \phi'_i \in X\}$ for $i \in \{1, \ldots, n\}$.

Let $\mathbb{A}^{\mu}_{\phi}$ be the Büchi automaton equivalent to the generalized Büchi automaton $\mathbb{A}$. It is easy to observe that $\mathbb{A}^{\mu_{\phi}}_{\phi}$ has an exponential amount of states in the size of ϕ and its transition relation can be checked in polynomial space in the size of ϕ. Moreover,

Lemma 11. *Let ϕ in $\mathrm{LTL}^{\mathrm{mem}}(\mathrm{RF})$ [resp. $\mathrm{LTL}^{\mathrm{mem}}(\mathrm{CL})$] and $\mu \geq \mu_{\phi}$ [resp. $\mu[w \leftarrow 0] \geq \mu_{\phi}[w \leftarrow 0]$]. Then, $\mathrm{L}(\mathbb{A}^{\mu}_{\phi}) = \mathrm{L}^{\mu}(\phi)$ [resp. $\mathrm{L}(\mathbb{A}^{\mu[w \leftarrow 0]}_{\phi}) = \mathrm{L}^{\mu[w \leftarrow 0]}(\phi)$].*

We can also build a Büchi automaton $\mathbb{A}^{\mu}_{sat}$ such that $\mathrm{L}(\mathbb{A}^{\mu}_{sat}) = \mathrm{L}^{\mu}_{sat}$. $\mathbb{A}^{\mu}_{sat}$ is defined as $(\Sigma, Q, \delta, I, F)$, where $\Sigma = \Sigma_{\mu}$, $Q = \Sigma_{\mu}$, $F = I = Q$ and $a \xrightarrow{a'} a''$ iff:

1. $\mathcal{A}_a, \mathcal{A}_{a''}$ are satisfiable, and $a = a'$,
2. for every formula $\langle \mathtt{x}, u\rangle = \langle \mathtt{x}', u'\rangle \in \mathcal{T}_{\mu}$ with $u, u' \geq 1$, $\langle \mathtt{x}, u\rangle = \langle \mathtt{x}', u'\rangle \in a$ iff $\langle \mathtt{x}, u-1\rangle = \langle \mathtt{x}', u'-1\rangle \in a''$.

If $\mu = \mu_{\phi}$, then $\mathbb{A}^{\mu}_{sat}$ is of exponential-size in the size of ϕ and the transition relation can be checked in polynomial space in the size of ϕ. More importantly, this automaton recognizes satisfiable symbolic models.

Lemma 12. *Let ϕ in $\mathrm{LTL}^{\mathrm{mem}}(\mathrm{RF})$ [resp. $\mathrm{LTL}^{\mathrm{mem}}(\mathrm{CL})$] and $\mu = \mu_{\phi}$ [resp. $\mu = \mu_{\phi}[w \leftarrow 0]$]. Then, $\mathrm{L}(\mathbb{A}^{\mu}_{sat}) = \mathrm{L}^{\mu}_{sat}$.*

Proof. It is immediate that the abstraction of any model belongs to $\mathrm{L}(\mathbb{A}^{\mu}_{sat})$. Therefore, the set of abstractions of memory states is included in $\mathrm{L}(\mathbb{A}^{\mu}_{sat})$.

The other inclusion is shown by induction. Let $\mu = (m, \epsilon, w, \mathtt{Lab}_0, \mathtt{Var}_0)$ be the measure μ_ϕ and α be $\max \epsilon + 1$. Let $(a_i)_{i \in \mathbb{N}}$ be a sequence of symbolic memory states in $\mathrm{L}(\mathbb{A}^{\mu}_{sat})$. We are going to prove by induction on k that, for all k, there are $s_0, \ldots, s_{k+m}$ and $h_0, \ldots, h_k$ such that, for all $u \leq k$:

- for all $\mathcal{A} \in \mathcal{T}_\mu$, $(s^\star_u, h_u) \models \mathcal{A}$ iff $\mathcal{A} \in a_u$, where $s^\star_u : \langle \mathtt{x}, v\rangle \mapsto s_{u+v}(\mathtt{x})$;
- $\mathtt{Im}(s^\star_u) \subseteq \alpha\mathbb{N} \cup \{nil\}$.

First, the inductive step. We are given $s_0, \ldots, s_{k+m}$ and $h_0, \ldots, h_k$ satisfying the above condition. We want to find s_{k+m+1} and h_{k+1}. Since $(a_i)_{i \in \mathbb{N}} \in \mathrm{L}(\mathbb{A}^{\mu}_{sat})$, and by definition of the transitions in this automaton, $\mathcal{A}_{a_{k+1}}$ is satisfiable. Let $s'_0, \ldots, s'_m$ and h' be the stores and heap obtained thanks to satisfaction of $\mathcal{A}_{a_{k+1}}$. By definition of $\mathbb{A}^{\mu}_{sat}$, $\langle \mathtt{x}, u+1\rangle = \langle \mathtt{x}', u'+1\rangle \in a_k$ iff $\langle \mathtt{x}, u\rangle = \langle \mathtt{x}', u'\rangle \in a_{k+1}$ for all $0 \leq u, u' \leq m-1$, and therefore the equality between variables is constant. Consequently, we have $s_{k+1+u}(\mathtt{x}) = s_{k+1+u'}(\mathtt{x}')$ iff $s'_u(\mathtt{x}) = s'_{u'}(\mathtt{x}')$ for all $0 \leq u, u' \leq m-1$. So, there is a permutation σ such that $\sigma \circ s'_v = s_{k+1+v}$ for all $0 \leq v \leq m-1$. Moreover, this permutation can be chosen such that $\mathtt{Im}(\sigma \circ s_v) \subseteq \alpha\mathbb{N} \cup \{nil\}$ for all $0 \leq v \leq m$ since, for all $0 \leq v \leq m-1$, $\mathtt{Im}(s_{k+1+v}) \subseteq \alpha\mathbb{N} \cup \{nil\}$. We define $s_{k+m+1} = \sigma \circ s'_m$.

If we consider RF, this permutation moreover satisfies the prerequisites of lemma 9, since $\epsilon = \{0\}$. We can define $h_{k+1} = \sigma \bullet h'$. Thanks to the lemma 9, we know that both of these models satisfy the same test formulae, which are exactly a_{k+1}.

If we are dealing with CL, then the definition of s_{k+m+1} ensures that the equalities satisfied are exactly those of a_{k+1}. This time the prerequisites of lemma 9 are not satisfied, unless $\epsilon = \{0\}$. We know that $w = 0$, which means that the only test formula about size in a_{k+1} is $\mathtt{size} \geq 0$; therefore there is no constraint on the size of the heap. The heap is defined by enumerating the test formulae of the form $\langle \mathtt{x}, u\rangle + j \overset{l}{\hookrightarrow} \langle \mathtt{x}', u'\rangle$ of a_{k+1}, and defining for each of them $h_{k+1}(s_{k+1+u}(\mathtt{x}) + j)(l) = s_{k+1+u'}(\mathtt{x}')$; and then for each of the test formulae of the form $\mathtt{alloc}\ \langle \mathtt{x}, u\rangle + j$ of a_{k+1}, we define $h_{k+1}(s_{k+1+u}(\mathtt{x}) + j)(l_0) = nil$, for some $l_0 \notin \mathtt{Lab}_0$. Thanks to the distance α between variables, the test formulae about the heap (i.e. $\hookrightarrow$ and $\mathtt{alloc}$, and $\mathtt{size} \geq 0$) which are not in a_{k+1} are not satisfied. Equalities, and inequalities, between variables are preserved since the stores have only been modified by a permutation.

Now, let us study the base case of the induction. Since $(a_i)_{i \in \mathbb{N}} \in \mathrm{L}(\mathbb{A}^{\mu}_{sat})$, and by definition of the transitions in this automaton, $\mathcal{A}_{a_0}$ is satisfiable and there are $s'_0, \ldots, s'_m$ and h'_0 satisfying $\mathcal{A}_0$. These are appropriate for the initialization if $\epsilon = \{0\}$. If $\epsilon \neq \{0\}$, then we consider the fragment CL, so $w = 0$, and there is no constraint on the size of the heap. We apply a permutation σ which maps all the images of variables to multiples of α. The obtained stores are $s_i = \sigma \circ s'_i$. As in the inductive step, the heap is defined by enumerating the test formulae $\langle \mathtt{x}, u\rangle + j \overset{l}{\hookrightarrow} \langle \mathtt{x}', u'\rangle$ and $\mathtt{alloc}\ \langle \mathtt{x}, u\rangle + j$ of a_{k+1}, and by defining the heap accordingly. Thanks to the distance α imposed between the values of variables, test formulae about the heap which are not in a_0 are not satisfied. Equalities between variables are preserved since the store has only been modified by a permutation. □

This lemma is essential and it is not possible to extend it to the whole logic LTL$^{\text{mem}}$ even by allowing test formulae of the form $\mathtt{x} + i = \mathtt{y} + j$ since we would need automata with counters. Now, we can state our main complexity result.

Theorem 1. SAT(RF) *and* SAT(CL) *are* PSPACE-*complete.*

Proof. (sketch) The lower bound is from LTL [SC85]. Let ϕ be an instance formula of SAT(RF) (for SAT(CL) replace below μ_ϕ by $\mu_\phi[w \leftarrow 0]$). As seen earlier, ϕ is satisfiable iff $\mathrm{L}^{\mu_\phi}(\phi) \cap \mathrm{L}^{\mu_\phi}_{sat}$ is nonempty. Hence, ϕ is satisfiable iff $\mathrm{L}(\mathbb{A}^{\mu_\phi}_\phi) \cap \mathrm{L}(\mathbb{A}^{\mu_\phi}_{sat}) \neq \emptyset$. The intersection automaton is of exponential size in the size of ϕ and can be checked nonempty by a nondeterministic on-the-fly algorithm. Since nonemptiness problem for Büchi automata is NLOGSPACE-complete and the transition relation in the intersection automaton can be checked in polynomial space in the size of ϕ, we obtain a nondeterministic polynomial space algorithm for testing satisfiability of ϕ. As usual, by Savitch's theorem, we get the PSPACE upper bound. □

4.3 Other problems in PSPACE

Let Frag be either the classical fragment or the record fragment. Lemma 1 provides a reduction from MC^{ct}_{init}(Frag) to SAT^{ct}_{init}(Frag) based on a program-as-formula encoding. As we will see now, we may also reduce SAT^{ct}_{init}(Frag) to SAT(Frag) internalizing an approximation of the initial memory state whose logical language cannot distinguish from the initial memory state. As a consequence, the PSPACE upper bound for SAT(Frag) entails the PSPACE upper bound for both SAT^{ct}_{init}(Frag) and MC^{ct}_{init}(Frag).

Proposition 3. *The problems* SAT^{ct}_{init}(RF), MC^{ct}_{init}(RF), SAT^{ct}_{init}(CL) *and* MC^{ct}_{init}(CL) *are* PSPACE-*complete.*

Proof. By Lemma 1 and since SAT^{ct}_{init}(RF) is known to be PSPACE-hard, it remains to establish the PSPACE upper bound for SAT^{ct}_{init}(RF).

Given a formula ϕ and an initial memory state (s, h), we shall build in polynomial-time a formula $\phi^{ct}_{s,h}$ in SAT(RF) such that ϕ is satisfiable in a model with initial memory state (s, h) and constant heap iff $\phi^{ct}_{s,h}$ is satisfiable by a general model. Since we have shown that SAT(RF) is in PSPACE, this guarantees that SAT^{ct}_{init}(RF) is in PSPACE. The idea of the proof is to internalize the initial memory state and the fact that the heap is constant in the logic SAT(RF). Actually, one cannot exactly express that the heap is constant (see details below) but the approximation we use will be sufficient for our purpose.

Apart from the variables of ϕ, the formula $\phi^{ct}_{s,h}$ is built over additional variables in $V = \{\mathtt{x}_i : i \in \mathtt{dom}(h) \cup \mathtt{Im}(s)\} \cup \{\mathtt{x}_{i,l} : i \in \mathtt{dom}(h), l \in \mathtt{dom}(h(i))\}$. The formula $\phi^{ct}_{s,h}$ is of the form $\mathsf{G}(\psi_1 \wedge \psi_2 \wedge \psi_3) \wedge \psi_s \wedge \psi'$, where the subformulae are defined as follows.

- ψ_1 states that the heap is almost equal to h since we cannot forbid additional labels in the logical language ($\mathtt{dom}(h) = \{i_1, \ldots, i_k\}$): $\psi_1 \stackrel{\text{def}}{=} (\bigwedge_{l \in \mathtt{dom}(h(i_1))} \mathtt{x}_{i_1} \stackrel{l}{\mapsto} \mathtt{x}_{i_1,l}) * \ldots * (\bigwedge_{l \in \mathtt{dom}(h(i_k))} \mathtt{x}_{i_k} \stackrel{l}{\mapsto} \mathtt{x}_{i_k,l})$.
- ψ_2 states which variables are equal and which ones are not, depending on the initial memory state. By way of example, for $i \neq j \in \mathtt{dom}(h)$, a conjunct of ψ_2 is $\mathtt{x}_i \neq \mathtt{x}_j$. Similarly, if $h(i)(l) = j$ and $j \in \mathtt{dom}(h)$ then $\mathtt{x}_{i,l} = \mathtt{x}_j$ is a conjunct of ψ_2. Details are omitted.
- ψ_3 states that the auxiliary variables remain constant: $\bigwedge_{\mathtt{x} \in V} \mathtt{x} = \mathsf{X}\mathtt{x}$.
- The formula ψ' is obtained from ϕ by replacing each occurrence of $\mathtt{x} \stackrel{l}{\hookrightarrow} e$ by

$$\mathtt{x} \stackrel{l}{\hookrightarrow} e \wedge \bigwedge_{i \in \mathtt{dom}(h), l \notin \mathtt{dom}(h(i))} \mathtt{x} \neq \mathtt{x}_i.$$

  The additional conjunct is useful because our logical language cannot state that a label is not in the domain of some allocated address.
- ψ_s states constraints about the initial store s: $\psi_s \stackrel{\text{def}}{=} \bigwedge_{\mathtt{x} \in \phi} \mathtt{x} = \mathtt{x}_{s(\mathtt{x})}$.

It is then easy to check that ϕ is satisfiable in a model with initial memory state (s, h) with constant heap iff $\phi^{ct}_{s,h}$ is satisfiable by a general model.

As far as the results for the classical fragment are concerned, by Lemma 1, there is a logspace reduction from $\mathrm{MC}^{ct}_{init}(\mathrm{CL})$ to $\mathrm{SAT}^{ct}_{init}(\mathrm{CL})$ and as done above one can reduce $\mathrm{SAT}^{ct}_{init}(\mathrm{CL})$ to $\mathrm{SAT}(\mathrm{CL})$. □

Proposition 4. $\mathrm{MC}^{ct}_{init}(\mathrm{SL})$ *is* PSPACE-*complete.*

Proof. Since $\mathrm{MC}^{ct}_{init}(\mathrm{SL})$ has been already proved to be PSPACE-hard, it remains to prove it is in PSPACE. The proof goes by designing a polynomial time reduction to the model-checking problem for propositional LTL. Let $(\mathtt{p}, s_0, h_0, \phi)$ be an instance of $\mathrm{MC}^{ct}_{init}(\mathrm{RF})$, with $\mathtt{p} = (Q, \delta, q_{init})$ a program without destructive update, (s_0, h_0) an initial memory state, and ϕ a temporal formula in $\mathrm{LTL}^{\mathrm{mem}}(\mathrm{SL})$. Let Σ be the finite set of stores $\{s : \mathtt{Im}(s) \subseteq \mathtt{Im}(s_0) \cup \mathtt{Im}(h_0)\}$ restricted to variables occurring in $\mathtt{p}$ and ϕ. All the memory states in the transition system $\mathcal{S}_{\mathtt{p}}$ restricted to the configurations reachable from the initial memory state (s_0, h_0) are in $\Sigma \times \{h_0\}$, since $\mathtt{p}$ is without destructive updates.

Let i be one plus the maximal natural number j such that $\mathsf{X}^j\mathtt{x}$ appears in ϕ. We define the transition graph $G = (Q_G, \rightarrow, Q_{init})$ such that $Q_G = Q \times \Sigma^i$, Q_{init} is the set of tuples $(q_{init}, s_1, s_2, .., s_i)$ such that $(s_1, h_0), .., (s_i, h_0)$ is a prefix of a run of $\mathtt{p}$ with initial memory state (s_0, h_0), and the transition relation $\rightarrow$ is defined as follows:

$$(q, s_1, .., s_i) \;\rightarrow\; (q', s'_1, .., s'_i) \text{ iff } \begin{cases} s_{k+1} = s'_k,\; k = 1, .., i-1, \text{ and } \exists q \xrightarrow{g, \mathtt{instr}} q' \in \delta \\ \text{s.t. } (s_1, h_0) \models g \text{ and } (s_2, h_0) = [\![\; \mathtt{instr} \;]\!](s_1, h_0) \end{cases}$$

We now define the propositional LTL model by associating to each vertex of the transition graph a set of propositional variables that are true. We define $Prop$ to be the set of atomic formulae occurring in ϕ, so that ϕ can be seen as a propositional LTL formula over $Prop$. Then the LTL model is the vertex-labeled transition graph $M = (G, \lambda)$, with

$$\lambda : \ Q_G \to \mathcal{P}(Prop), \ (q, s_1, .., s_i) \mapsto \ \{\mathcal{A} \in Prop \ : \ s_1, .., s_i, h_0 \models_{\mathrm{SL}} \mathcal{A}\}$$

By construction, $M, (q_{init}, s_1, s_2, .., s_i) \models \phi$ in LTL for some $(q_{init}, s_1, s_2, .., s_i) \in Q_{init}$ if and only if $\mathtt{p}, (s_0, h_0) \models \phi$. The model M can be computed in polynomial space in the size of $(\mathtt{p}, s_0, h_0, \phi)$, in the sense that the (nondeterministic) transition function and the labelling function are computable in polynomial space. M is of exponential space in the size of $(\mathtt{p}, s_0, h_0, \phi)$, but let us explain now why the existence of $(q_{init}, s_1, s_2, .., s_i) \in Q_{init}$ such that $M, (q_{init}, s_1, s_2, .., s_i) \models \phi$ can be checked in polynomial space. Let $\mathbb{A}_\phi$ be the automaton recognizing the models of ϕ over the set $Prop$ of propositions: it is of exponential space in the size of $(\mathtt{p}, s_0, h_0, \phi)$, and so is the product with M. Now the existence of $(q_{init}, s_1, s_2, .., s_i) \in Q_{init}$ such that $M, (q_{init}, s_1, s_2, .., s_i) \models \phi$ reduces to check the non-emptiness of $\mathbb{A}_\phi \cap M$, which is decidable in space $\mathcal{O}(\log(|\mathbb{A}_\phi \cap M|)$ by a nondeterministic on the fly algorithm. This can be done in polynomial space in the size of $(\mathtt{p}, s_0, h_0, \phi)$ by a non-deterministic algorithm, and by Savitch's theorem this can be turned into a deterministic, polynomial space algorithm. $\square$

Proposition 5. $\mathrm{SAT}^{ct}_{init}(\mathrm{SL} \setminus \{\mathrel{-\!\!*}\})$ *is* PSPACE*-complete.*

Proof. PSPACE-hardness is a consequence of the PSPACE-hardness of $\mathrm{SAT}^{ct}_{init}(\mathrm{CL})$ since CL is a fragment of $\mathrm{SL} \setminus \{\mathrel{-\!\!*}\}$.

In order to get the PSPACE upper bound, we are going to reduce the problem $\mathrm{SAT}^{ct}_{init}(\mathrm{SL}\setminus\{\mathrel{-\!\!*}\})$ to $\mathrm{SAT}^{ct}_{init}(\mathrm{RF})$. Let $(s_0, h), \phi$ be an instance of $\mathrm{SAT}^{ct}_{init}(\mathrm{SL}\setminus\{\mathrel{-\!\!*}\})$. We shall build an instance $(s'_0, h), \phi'$ of $\mathrm{SAT}^{ct}_{init}(\mathrm{RF})$.

Let $E = \mathtt{dom}(h) \cup \{k - i \in \mathbb{N}, k \in \mathtt{dom}(h)$ and $\mathsf{X}^u\mathtt{x} + i$ occurs in $\phi\}$. We define a new variable $\langle k \rangle$ for each $k \in E$. We also define a variable $\langle \mathtt{x}, i \rangle$ for $\mathtt{x}$ and i occuring in ϕ in an expression of the form $\mathsf{X}^u\mathtt{x} + i$ (possibly u or i is equal to zero). The initial store s'_0 is the extension of s_0 which maps $\langle k \rangle$ to k, and $\langle \mathtt{x}, i \rangle$ to $s_0(\mathtt{x}) + i$. Finally:

$$\begin{aligned} \phi' = {} & \phi[\mathsf{X}^u\mathtt{x} + i \leftarrow \mathsf{X}^u\langle \mathtt{x}, i \rangle] \\ & \wedge \mathsf{G} \bigwedge_{k \in E} (\langle k \rangle = \mathsf{X}\langle k \rangle) \\ & \wedge \mathsf{G} \bigwedge_{\mathtt{x}+i \in \phi} \ \bigwedge_{(k+i) \in \mathtt{dom}(h)} (\mathtt{x} = \langle k \rangle \Leftrightarrow \langle \mathtt{x}, i \rangle = \langle k + i \rangle) \end{aligned}$$

s'_0 and ϕ' have a polynomial size in the size of the instance $(s_0, h), \phi$.

Assume that $(s_0, h), \phi$ is accepted by $\mathrm{SAT}^{ct}_{init}(\mathrm{SL}\setminus\{\mathrel{-\!\!*}\})$. Then there is $(s_i)_{i \geq 1}$ such that $(s_i, h)_{i \in \mathbb{N}} \models \phi$. Let s'_i be s_i extended so as to map $\langle k \rangle$ to k and $\langle \mathtt{x}, j \rangle$ to $s_i(\mathtt{x}) + j$. Clearly $(s'_i, h)_{i \in \mathbb{N}} \models \phi[\mathsf{X}^u\mathtt{x} + i \leftarrow \mathsf{X}^u\langle \mathtt{x}, i \rangle]$. Our definition of the s'_i also ensures that $(s'_i, h)_{i \in \mathbb{N}} \models \mathsf{G} \bigwedge_{k \in E}(\langle k \rangle = \mathsf{X}\langle k \rangle)$ since the value of a variable $\langle k \rangle$ is constantly equal to k, and that $(s'_i, h)_{i \in \mathbb{N}} \models \mathsf{G} \bigwedge_{\langle \mathtt{x}, i \rangle \in \phi} \bigwedge_{(k+i) \in \mathtt{dom}(h)} (\mathtt{x} = \langle k \rangle \Leftrightarrow \langle \mathtt{x}, i \rangle = \langle k + i \rangle)$ since at all positions the value of $\langle k + i \rangle$ is that of $\langle k \rangle$ plus i and the value of $\langle \mathtt{x}, i \rangle$ is that of $\mathtt{x}$ plus i. So $(s'_i, h)_{i \in \mathbb{N}} \models \phi'$, and therefore $(s'_0, h), \phi'$ is accepted by $\mathrm{SAT}^{ct}_{init}(\mathrm{RF})$.

Now, assume that $(s'_0, h), \phi'$ is accepted by $\mathrm{SAT}^{ct}_{init}(\mathrm{RF})$. Then there is a sequence $(s'_i)_{i\geq 1}$ such that $(s'_i, h)_{i\in\mathbb{N}} \models \phi'$. Then $(s'_i, h)_{i\in\mathbb{N}} \models \mathsf{G} \bigwedge_{k\in E}(\langle k\rangle = \mathsf{X}\langle k\rangle)$, and so, at each time state t, we have $s'_t(\langle k\rangle) = s'_0(\langle k\rangle) = k$. Also, $(s'_i, h)_{i\in\mathbb{N}} \models \mathsf{G} \bigwedge_{\mathtt{x}+i\in\phi} \bigwedge_{(k+i)\in\mathtt{dom}(h)}(\mathtt{x} = \langle k\rangle \Leftrightarrow \langle \mathtt{x}, i\rangle = \langle k+i\rangle)$, and so, if $k \in \mathtt{dom}(h)$ and $\mathsf{X}^u\mathtt{x}+i$ occurs in ϕ, we have $s'_{t+u}(\mathtt{x}) = k - i$ iff $s'_{t+u}(\langle \mathtt{x}, i\rangle) = k$ (I).

We write $h' \leq h$ when there is an other heap h'' for which $h = h' * h''$. Let us prove by induction on subformulae ϕ_0 of ϕ the following property. "For all $t \in \mathbb{N}$ and $h' \leq h$, we have $(s'_i, h')_{i\in\mathbb{N}}, t \models \phi_0$ iff $(s'_i, h')_{i\in\mathbb{N}}, t \models \phi_0[\mathsf{X}^u\mathtt{x}+i \leftarrow \mathsf{X}^u\langle \mathtt{x}, i\rangle]$." This will ensure that $(s'_i, h)_{i\in\mathbb{N}}, 0 \models \phi$, so that $(s'_0, h), \phi$ is accepted by $\mathrm{SAT}^{ct}_{init}(\mathrm{SL}\backslash\{-\!\!*\})$, from which we will conclude that $(s_0, h), \phi$ is also accepted. Indeed the restriction of s'_0 to the variables occuring in ϕ is s_0. Here is the proof by induction:

- If ϕ_0 is $\mathsf{X}^u\mathtt{x}+i \overset{l}{\hookrightarrow} \mathsf{X}^{u'}\mathtt{y}$, let $k = s'_{t+u}(\langle \mathtt{x}, i\rangle)$.
  - Suppose that $k \notin \mathtt{dom}(h)$. We are going to prove that neither $(s'_i, h')_{i\in\mathbb{N}}, t \models \phi_0[\mathsf{X}^u\mathtt{x}+i \leftarrow \mathsf{X}^u\langle \mathtt{x}, i\rangle]$, nor $(s'_i, h')_{i\in\mathbb{N}}, t \models \phi_0$. First, clearly not $(s'_i, h')_{i\in\mathbb{N}}, t \models \phi_0[\mathsf{X}^u\mathtt{x}+i \leftarrow \mathsf{X}^u\langle \mathtt{x}, i\rangle]$. Second, assume there is $k' \in \mathtt{dom}(h)$ such that $k' = s'_{t+u}(\mathtt{x}) + i$. Thanks to the property (I), from $s'_{t+u}(\mathtt{x}) = k' - i$, we get $s'_{t+u}(\langle \mathtt{x}, i\rangle) = k'$, and so $k = k' \in \mathtt{dom}(h)$, which leads to a contradiction. So there is no such k', and not $(s'_i, h')_{i\in\mathbb{N}}, t \models \phi_0$.
  - Now suppose that $k \in \mathtt{dom}(h)$. We have $s'_{t+u}(\mathtt{x}) = k = s'_{t+u}(\langle \mathtt{x}, i\rangle) - i$ thanks to property (I). Consequently, $h'(s'_{t+u}(\mathtt{x})+i) = s'_{t+u'}(\mathtt{y})$ iff $h'(s'_{t+u}(\langle \mathtt{x}, i\rangle)) = s'_{t+u}(\langle \mathtt{y}, 0\rangle)$, and $(s'_i, h')_{i\in\mathbb{N}}, t \models \phi_0$ iff $(s'_i, h')_{i\in\mathbb{N}}, t \models \phi_0[\mathsf{X}^u\mathtt{x}+i \leftarrow \mathsf{X}^u\langle \mathtt{x}, i\rangle]$.
- If $\phi_0 = \mathcal{A}_1 * \mathcal{A}_2$, then there are h'_1 and h'_2 such that $(s'_i, h'_1)_{i\in\mathbb{N}}, t \models \mathcal{A}_1$ and $(s'_i, h'_2)_{i\in\mathbb{N}}, t \models \mathcal{A}_2$. By the induction hypothesis, since $h = (h'_1 * h'_2) * h'' = h'_1 * (h'_2 * h'')$, we have $(s'_i, h'_1)_{i\in\mathbb{N}}, t \models \mathcal{A}_1[\mathsf{X}^u\mathtt{x}+i \leftarrow \mathsf{X}^u\langle \mathtt{x}, i\rangle]$ iff $(s'_i, h'_1)_{i\in\mathbb{N}}, t \models \mathcal{A}_1$; and the same equivalence is true for h'_2. From the two equivalences for h'_1 and h'_2, we can conclude the same equivalence for $h' = h'_1 * h'_2$.
- Other cases are straightforward.

□

If we allow the operator $-\!\!*$ in the above proposition, the current proof may not be adapted, since we would have to deal with heaps which are not sub-heaps of h in the induction step.

5 Undecidability Results

In this section, we show several undecidability results by using reduction from problems for Minsky machines. So, first, we recall that a Minsky machine M consists of two counters C_1 and C_2, and a sequence of $n \geq 1$ instructions of one of the forms below:

l: $\mathsf{C}_i := \mathsf{C}_i + 1$; goto l' $\quad$ l: if $\mathsf{C}_i = 0$ then goto l' else $\mathsf{C}_i := \mathsf{C}_i - 1$; goto l''

In a nondeterministic machine, after an incrementation or a decrementation a nondeterministic choice of the form "goto l_1 or goto l_2" is performed.

The configurations of M are triples (l, c_1, c_2), where $1 \leq l \leq n$, $c_1 \geq 0$, and $c_2 \geq 0$ are the current values of the location counter and the two counters C_1 and C_2, respectively. The consecution relation on configurations is defined in the obvious way. A computation of M is a sequence of related configurations, starting with the initial configuration $(1, 0, 0)$.

Different encodings of counters are used here. For instance, in [BFLS06], a counter C with value n is represented by a list of length n pointed to by a `x` dedicated to C. The same idea is used in the proof of Proposition 6 below. In order to show undecidability of SAT(SL) we alternatively encode counters by relying on pointer arithmetic and properties of heaps. Programs without destructive updates can simulate finite computations of Minsky machines by guessing at the start of the computation the maximal value of counters (encoded by a list of the length of the maximal value). As a consequence,

Proposition 6. $\mathrm{SAT}^{ct}(\mathrm{LF})$ *and* $\mathrm{MC}^{ct}(\mathrm{LF})$ *are* Σ^0_1*-complete.*

Proof. By Proposition 3, $\mathrm{SAT}^{ct}_{init}(\mathrm{LF})$ is decidable in polynomial space using a finite abstraction argument. Hence, $\mathrm{SAT}^{ct}(\mathrm{LF})$ is in Σ^0_1 by adding an existential quantification over the initial memory state. Similarly, by Proposition 3, $\mathrm{MC}^{ct}_{init}(\mathrm{LF})$ is decidable in polynomial space. Hence, $\mathrm{MC}^{ct}(\mathrm{LF})$ is also in Σ^0_1.

By Lemma 1, we only need to show that $\mathrm{MC}^{ct}(\mathrm{LF})$ is Σ^0_1-hard. We reduce the Σ^0_1-complete halting problem for Minsky machines to it. The halting problem consists in determining whether M can reach a configuration with location counter n.

Let us build a formula ϕ and a program `p` in $\mathtt{P}^{ct}$ such that the existence of some memory state (s_0, h_0) such that $\mathtt{p}, (s_0, h_0) \models \phi$ is equivalent to the fact that the machine M reaches a configuration with location counter n. In order to encode the values of counters, we consider a variable `z` pointing to a list (as shown below) in the initial memory state (s_0, h_0):

$$_{\mathtt{z}}\square \xrightarrow{next} \square \xrightarrow{next} \cdots \square \xrightarrow{next} \square \xrightarrow{next} nil$$

The variable `z` remains constant along any execution of `p` and the length of the list encodes the maximal value of the counters in some finite computation (hopefully ending at the instruction corresponding to location n). We consider also the variable $\mathtt{x}_i$ for $i = 1, 2$ and along any execution of `p`, $\mathtt{x}_i$ points to a cell of the above sequence: the length of the list starting at $\mathtt{x}_i$ encodes the value of the counter C_i. Hence, in `p`, each $\mathtt{x}_i$ is initialized to `null`.

The program `p` is made of the following stages:

1. Check that `z` points to a list;
2. Initialize the variables;
3. Simulate M.

Figure 2 shows how to perform stage 1 with a simple "while" loop. Observe that checking whether a counter is equal to zero corresponds in `p` to an equality test with `null`. In order to simulate M, its structure can be embedded in the

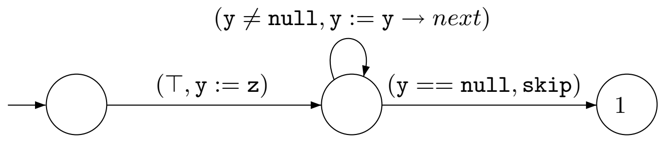

Fig. 2. Checking that z points to a list

control graph of p. For instance, a decrementation instruction is encoded in p by the transitions shown in Figure 3. An incrementation instruction requires a bit more care and its encoding in p is presented in Figure 4. Indeed, an auxiliary variable y′ initialized to y visits the list until it meets x_i.

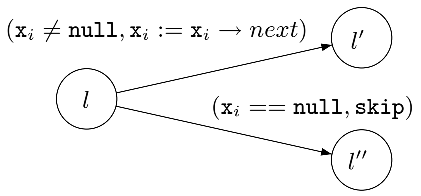

Fig. 3. Simulating a decrementation

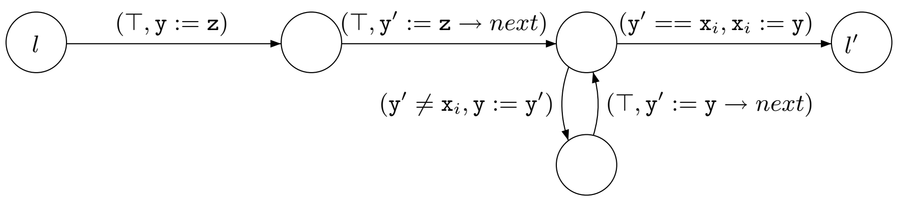

Fig. 4. Simulating an incrementation

In the above encoding, every instruction l in M corresponds to a control state of p. Hence, the formula ϕ is simply $\mathsf{F}n$ (assuming that we encode the propositional variable n by additional variables dedicated only for this purpose).

It is then easy to show that there is an initial memory state (s_0, h_0) such that p reaches the control n starting with (s_0, h_0) iff the machine M reaches the location counter n. Observe that both p and M are deterministic. □

By constrast, programs with destructive update can work with unbounded heaps, and by using the representation of counters as above, they can faithfully simulate a Minsky machine even if an empty heap is the initial heap. Because LTL can express repeated accessibility, Σ_1^1-hardness can be obtained.

Proposition 7. *The problems* MC(LF) *and* MC_{init}(LF) *are* Σ_1^1*-complete.*

Proof. (sketch) It is possible to reduce the recurring problem for nondeterministic Minsky machines to MC(LF) and to MC_{init}(LF). This problem is Σ_1^1-hard [AH94]. The question is whether the machine has a computation with the location counter n repeated infinitely often; and this can be expressed by $\mathsf{GF}n$ in $\mathrm{LTL}^{\mathrm{mem}}$.

The proof is quite similar to the proof of Proposition 6 except that there is no maximal value of the counters, the initial heap is empty (which can be expressed in $\mathrm{LTL}^{\mathrm{mem}}$) and the behavior of counters is encoded by updating the memory states. For instance, incrementing C_i amounts to execute $\mathtt{x}_i := \mathtt{cons}(\mathit{next} : \mathtt{x}_i)$ (the length of the list pointed by $\mathtt{x}_i$ is incremented), decrementing C_i amounts to execute $\mathtt{x}_i := \mathtt{x}_i \rightarrow \mathit{next}$. Zero tests are encoded by `null` tests and the initial values of the variables is `null`. Details are omitted since there are no technical difficulties. $\square$

Let us briefly explain how to encode incrementation and decrementation with separating connectives and pointer arithmetic. Observe that expressions of the form $\mathtt{x} = \mathtt{y} + 1$ are not allowed in the logical language. We repair this "defect" in two different ways: using non-aliasing expressed by the separating conjunction, and using the precise pointing assertion $\mathtt{x} \overset{\mathit{next}}{\mapsto} \eta$ stating that the heap contains only one cell, in conjunction with the $-\!\!*$ operator.

$$\begin{aligned}
\phi^{*}_{\mathtt{x}++} &= (\mathsf{X}\mathtt{x} \overset{\mathit{next}}{\hookrightarrow} \mathtt{null} \wedge \mathtt{x}+1 \overset{\mathit{next}}{\hookrightarrow} \mathtt{null}) \wedge \neg(\mathsf{X}\mathtt{x} \overset{\mathit{next}}{\hookrightarrow} \mathtt{null} * \mathtt{x}+1 \overset{\mathit{next}}{\hookrightarrow} \mathtt{null})\\
\phi^{*}_{\mathtt{x}--} &= (\mathsf{X}\mathtt{x}+1 \overset{\mathit{next}}{\hookrightarrow} \mathtt{null} \wedge \mathtt{x} \overset{\mathit{next}}{\hookrightarrow} \mathtt{null}) \wedge \neg(\mathsf{X}\mathtt{x}+1 \overset{\mathit{next}}{\hookrightarrow} \mathtt{null} * \mathtt{x} \overset{\mathit{next}}{\hookrightarrow} \mathtt{null})\\
\phi^{-\!*}_{\mathtt{x}++} &= \mathtt{emp} \quad\wedge\quad \big((\mathsf{X}\mathtt{x} \overset{\mathit{next}}{\mapsto} \mathtt{null}) -\!\!* \mathtt{x}+1 \overset{\mathit{next}}{\mapsto} \mathtt{null}\big)\\
\phi^{-\!*}_{\mathtt{x}--} &= \mathtt{emp} \quad\wedge\quad \big((\mathtt{x} \overset{\mathit{next}}{\mapsto} \mathtt{null}) -\!\!* \mathsf{X}\mathtt{x}+1 \overset{\mathit{next}}{\mapsto} \mathtt{null}\big)
\end{aligned}$$

The formulae based on the separating conjunction correctly express incrementation and decrementation when the cells at index $\mathtt{x}, \mathtt{x}+1, \mathtt{x}-1$ are allocated, whereas formulae based on the operator $-\!\!*$ do not need the same assumption.

Let $\mathrm{SAT}^{?}_{?}(\mathrm{SL})$ be any satisfiability problem among the four variants.

Proposition 8. *$\mathrm{SAT}^{?}_{?}(\mathrm{SL})$ is Σ^1_1-complete.*

Proof. We reduce the recurrence problem for nondeterministic Minsky machines [AH94] to $\mathrm{SAT}^{?}_{?}(\mathrm{SL})$. Let ϕ_0 be the formula $\mathsf{G}(\mathtt{emp} \wedge \bigwedge_{i=1}^{2}(\mathtt{x}_i \neq \mathtt{null}))$. Incrementation and decrementation are performed thanks to the formulae $\phi^{-\!*}_{\mathtt{x}++}$ and $\phi^{-\!*}_{\mathtt{x}--}$, respectively. For any model ρ such that $\rho, 0 \models \phi_0$, and for any t, we have $\rho, t \models \phi^{-\!*}_{\mathtt{x}_i++}$ iff $s_t(\mathtt{x}_i) + 1 = s_{t+1}(\mathtt{x}_i)$. Hence, we have a means to encode incrementation. Similarly, $\rho, t \models \phi^{-\!*}_{\mathtt{x}_i--}$ and $s_t(\mathtt{x}_i) > 0$ iff $s_t(\mathtt{x}_i) - 1 = s_{t+1}(\mathtt{x}_i)$. The fact that a counter does not change is encoded by $\mathtt{x}_i = \mathsf{X}\mathtt{x}_i$. Given that $\phi_1 = \mathsf{G}(\mathtt{x}_{zero} = \mathsf{X}\mathtt{x}_{zero} \wedge \mathtt{x}_{zero} \neq \mathtt{null})$ holds, zero tests are encoded by $\mathtt{x}_i = \mathtt{x}_{zero}$.

Given a nondeterministic Minsky machine M, we write ψ_l to denote the formula encoding instruction l. For intance for the instruction "l: if $\mathsf{C}_1 = 0$ then goto l' else $\mathsf{C}_1 := \mathsf{C}_1 - 1$; goto l'_1 or goto l'_2" ψ_l is equal to the formula below:

$$\mathsf{G}((l \wedge \mathtt{x}_1 \neq \mathtt{x}_{zero}) \Rightarrow (\mathtt{x}_2 = \mathsf{X}\mathtt{x}_2 \wedge (\mathsf{X}l'_1 \vee \mathsf{X}l'_2) \wedge \phi^{-\!*}_{\mathtt{x}_1--})) \wedge$$

$$\mathsf{G}((l \wedge \mathtt{x}_1 = \mathtt{x}_{zero}) \Rightarrow (\mathtt{x}_1 = \mathsf{X}\mathtt{x}_1 \wedge \mathtt{x}_2 = \mathsf{X}\mathtt{x}_2 \wedge \mathsf{X}l')).$$

Hence, $(\mathtt{x}_1 = \mathtt{x}_2 = \mathtt{x}_{zero}) \wedge \phi_0 \wedge \phi_1 \wedge \bigwedge_l \psi_l \wedge \mathsf{GF}n$ is satisfiable iff M has a computation with location counter n repeated infinitely often. $\square$

Proposition 9. *The problem* $\mathrm{SAT}(\mathrm{SL} \setminus \{-\!\!*\})$ *is* Σ^1_1*-complete.*

The proof of Proposition 9 is similar to the proof of Theorem 8 except that incrementation and decrementation are performed with the formulae $\phi^*_{\mathtt{x}++}$ and $\phi^*_{\mathtt{x}--}$, respectively.

6 Conclusion

In the paper, we have introduced a temporal logic $\mathrm{LTL}^{\mathrm{mem}}$ for which assertion language is quantifier-free separation logic.

Figure 5 shows the reductions between problems. Curved lines represent reductions for proving hardness in a class. Straight lines represent reductions for showing that a problem belongs to its class.

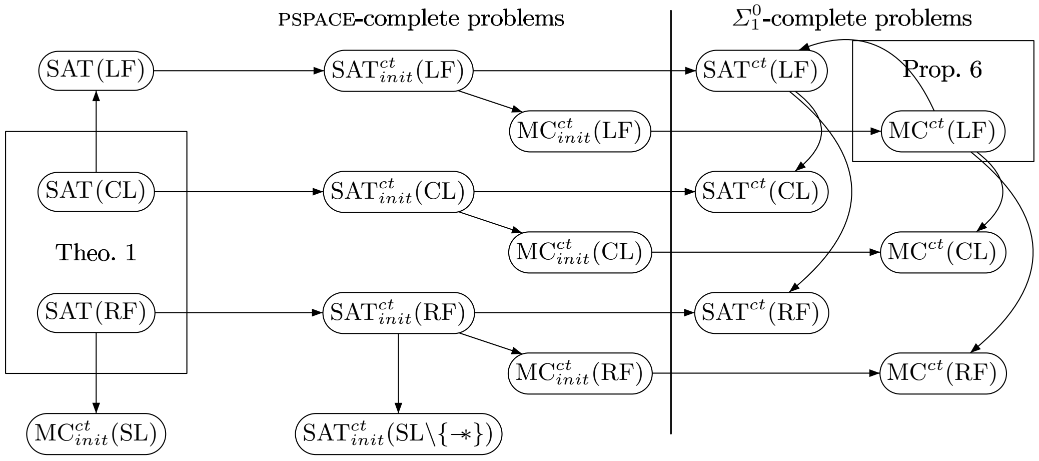
PSPACE-complete problems
Σ^0_1-complete problems
SAT(LF)
$\mathrm{SAT}^{ct}_{init}(\mathrm{LF})$
$\mathrm{SAT}^{ct}(\mathrm{LF})$
Prop. 6
$\mathrm{MC}^{ct}_{init}(\mathrm{LF})$
$\mathrm{MC}^{ct}(\mathrm{LF})$
SAT(CL)
$\mathrm{SAT}^{ct}_{init}(\mathrm{CL})$
$\mathrm{SAT}^{ct}(\mathrm{CL})$
Theo. 1
$\mathrm{MC}^{ct}_{init}(\mathrm{CL})$
$\mathrm{MC}^{ct}(\mathrm{CL})$
SAT(RF)
$\mathrm{SAT}^{ct}_{init}(\mathrm{RF})$
$\mathrm{SAT}^{ct}(\mathrm{RF})$
$\mathrm{MC}^{ct}_{init}(\mathrm{RF})$
$\mathrm{MC}^{ct}(\mathrm{RF})$
$\mathrm{MC}^{ct}_{init}(\mathrm{SL})$
$\mathrm{SAT}^{ct}_{init}(\mathrm{SL}\setminus\{-\!\!*\})$

Fig. 5. Reductions

Figure 6 contains a summary of the complexity results about fragments of $\mathrm{LTL}^{\mathrm{mem}}$.

	MC	MC^{ct}	MC^{ct}_{init}	MC_{init}	SAT	SAT^{ct}	SAT^{ct}_{init}
LF	Σ^1_1-c.	Σ^0_1-c.	PSPACE-c.	Σ^1_1-c.	PSPACE-c.	Σ^0_1-c.	PSPACE-c.
CL and RF	Σ^1_1-c.	Σ^0_1-c.	PSPACE-c.	Σ^1_1-c.	PSPACE-c.	Σ^0_1-c.	PSPACE-c.
$\mathrm{SL}\setminus\{-\!\!*\}$	Σ^1_1-c.	Σ^0_1-c.	PSPACE-c	Σ^1_1-c.	Σ^1_1-c.	Σ^0_1-c.	PSPACE-c
SL	Σ^1_1-c.	Σ^0_1-c.	PSPACE-c	Σ^1_1-c.	Σ^1_1-c.	Σ^1_1-c.	Σ^1_1-c.

Fig. 6. Complexity of reasoning about program with pointer variables

Finally, extending $\mathrm{LTL}^{\mathrm{mem}}$ with a special propositional variable $\mathtt{heap}^{=}$ stating that the current heap is equal to the next one, can lead to undecidability (look at the problems of the form $\mathrm{SAT}^{ct}_{?}(\mathrm{Frag})$). However, it is open whether satisfiability becomes decidable if we restrict the interplay between the "until" operator U

and heap$^=$, for instance to forbid subformulae of the form G heap$^=$ with positive polarity.

References

[AH94] R. Alur and Th. Henzinger. A really temporal logic. *Journal of the Association for Computing Machinery*, 41:181–204, 1994.

[BBH$^+$06] A. Bouajjani, M. Bozga, P. Habermehl, R. Iosif, P. Moro, and T. Vojnar. Programs with lists are counter automata. In *CAV'06*, volume 4144 of *Lecture Notes in Computer Science*, pages 517–531. Springer, 2006.

[BC02] Ph. Balbiani and J.F. Condotta. Computational complexity of propositional linear temporal logics based on qualitative spatial or temporal reasoning. In *FROCOS'02*, volume 2309 of *Lecture Notes in Artificial Intelligence*, pages 162–173. Springer, 2002.

[BCMS01] O. Burkart, D. Caucal, F. Moller, and B. Steffen. Verification of infinite structures. In *Handbook of Process Algebra*, pages 545–623. Elsevier, 2001.

[BCO05a] 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.

[BCO05b] J. Berdine, C. Calcagno, and P. W. O'Hearn. Symbolic execution with separation logic. In *APLAS'05*, volume 3780 of *Lecture Notes in Computer Science*, pages 52–68. Springer, 2005.

[BDL07] R. Brochenin, S. Demri, and E. Lozes. Reasoning about sequences of memory states. In *LFCS'07*, volume 4514 of *Lecture Notes in Computer Science*, pages 100–114. Springer, 2007.

[BFLS06] S. Bardin, A. Finkel, E. Lozes, and A. Sangnier. From pointer systems to counter systems using shape analysis. *5th International Workshop on Automated Verification of Infinite-State Systems (AVIS'06)*, 2006.

[BFN04] S. Bardin, A. Finkel, and D. Nowak. Toward symbolic verification of programs handling pointers. In *3rd International Workshop on Automated Verification of Infinite-State Systems (AVIS'04)*, 2004.

[BIL04] M. Bozga, R. Iosif, and Y. Lakhnech. On logics of aliasing. In *SAS'04*, volume 3148 of *Lecture Notes in Computer Science*, pages 344–360. Springer, 2004.

[BWZ02] B. Bennett, F. Wolter, and M. Zakharyaschev. Multi-dimensional modal logic as a framework for spatio-temporal reasoning. *Applied Intelligence*, 17(3):239–251, 2002.

[CC00] H. Comon and V. Cortier. Flatness is not a weakness. In *CSL'00*, volume 1862 of *Lecture Notes in Computer Science*, pages 262–276. Springer, 2000.

[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.

[CYO01] C. Calcagno, H. Yang, and P. O'Hearn. Computability and complexity results for a spatial assertion language for data structures. In *FST&TCS'01*, volume 2245 of *Lecture Notes in Computer Science*, pages 108–119. Springer, 2001.

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

[DKR04] D. Distefano, J.-P. Katoen, and A. Rensink. Who is pointing when to whom? on the automated verification of linked list structures. In *FST&TCS'04*, volume 3328 of *Lecture Notes in Computer Science*, pages 250–262. Springer, 2004.

[FLS07] A. Finkel, E. Lozes, and A. Sangnier. Towards model-checking pointer systems without destructive update. 2007. Under submission.

[GKWZ03] D. Gabbay, A. Kurucz, F. Wolter, and M. Zakharyaschev. *Many-dimensional modal logics: theory and applications*. CUP, 2003.

[GM05] D. Galmiche and D. Mery. Characterizing provability in BI's pointer logic through resource graphs. In *LPAR'05*, volume 3835 of *Lecture Notes in Computer Science*, pages 459–473. Springer, 2005.

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

[JJKS97] J. Jensen, M. Jorgensen, N. Klarlund, and M. Schwartzbach. Automatic verification of pointer programs using monadic second-order logic. In *PLDI'97*, pages 226–236. ACM, 1997.

[KVW00] O. Kupferman, M. Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. *Journal of the Association for Computing Machinery*, 47(2):312–360, 2000.

[LAS00] T. Lev-Ami and M. Sagiv. TVLA: A system for implementing static analyses. In *SAS'00*, pages 280–301, 2000.

[Loz04] E. Lozes. Separation logic preserves the expressive power of classical logic. In *2nd Workshop on Semantics, Program Analysis, and Computing Environments for Memory Management (SPACE'04)*, 2004.

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

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

[SC85] A. Sistla and E. Clarke. The complexity of propositional linear temporal logic. *Journal of the Association for Computing Machinery*, 32(3):733–749, 1985.

[VW94] M. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115:1–37, 1994.

[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 204–22. Springer, 2003.