Reasoning on Dynamic Transformations of Symbolic Heaps

Nicolas Peltier
Univ. Grenoble Alpes, CNRS, LIG, Grenoble, France

Abstract
Building on previous results concerning the decidability of the satisfiability and entailment problems for separation logic formulas with inductively defined predicates, we devise a proof procedure to reason on dynamic transformations of memory heaps. The initial state of the system is described by a separation logic formula of some particular form, its evolution is modeled by a finite transition system and the expected property is given as a linear temporal logic formula built over assertions in separation logic.

2012 ACM Subject Classification Theory of computation → Separation logic; Theory of computation → Automated reasoning; Theory of computation → Modal and temporal logics

Keywords and phrases Separation Logic, Symbolic Heaps, Linear Temporal Logic

Digital Object Identifier 10.4230/LIPIcs.TIME.2022.9

Funding This work has been partially funded by the the French National Research Agency (ANR-21-CE48-0011).

1 Introduction

Separation logic (SL) [14] is a dialect of bunched logic [10], that was introduced in verification to reason on programs manipulating dynamically allocated memory. The logic uses a particular connective $*$ to assert that two formulas hold on disjoint parts of the memory, which allows for more concise specifications. It supports local reasoning, in the sense that the properties of a program can be asserted and proven by referring only to the part of the memory that is affected by the program, and not to the global state of the system. The expressive power of the logic may be enhanced by using inductively defined predicates, which can be used to define recursive data structures of unbounded sizes, such as lists or trees. For instance, the following rules define a predicate $\text{lseg}(x, y)$ denoting a non empty list segment from $x$ to $y$: $\{\text{lseg}(x, y) \iff x \mapsto (y), \text{lseg}(x, y) \iff \exists z. (x \mapsto (z) * \text{lseg}(z, y))\}$. Informally, $x, y, z$ denotes locations (i.e., memory addresses), $x \mapsto (y)$ states that location $x$ is allocated and points to location $y$ and the separating conjunction $x \mapsto (z) * \text{lseg}(z, y)$ states that the heap contains a list segment $\text{lseg}(z, y)$ together with an additional memory cell $x$ that points to $z$ (it implicitly entails that $x$ is distinct from all the memory locations allocated in the list segment from $z$ to $y$). These predicates may be hard coded, but they may also be defined by the user, to tackle custom data structures. For the fragment of separation logic called symbolic heaps (formally defined later), satisfiability is decidable [3], but entailment is undecidable in general (entailment cannot be reduced to satisfiability since the fragment does not include negations). However, a general class of decidable entailment problems is described in [7], based on restrictions on the form of the inductive rules that define the semantics of the inductive predicates. More recently, it was shown that the entailment problem is 2-EXPTIME complete [11, 4] for such inductive rules. Building on these results, we devise in the present work a proof procedure to reason on dynamic transformations of data structures specified by SL formulas with inductively defined predicates. More precisely, we consider entailments of the form $\phi \models^S_R \Phi$, where $\phi$ is an SL formula (more precisely a
symbolic heap), $R$ is a set of inductive rules, $S$ is a transition system and $\Phi$ is a formula combining symbolic heaps with temporal connectives of linear temporal logic (LTL) [13]. Informally, such an entailment is valid if the formula $\Phi$ holds w.r.t. all the runs obtained by starting from a structure satisfying the formula $\phi$ and following the transition system $S$. The symbolic heap $\phi$ describes the initial state of the system, $R$ defines the semantics of the inductively defined predicate symbols, $S$ describes how the system evolves along time and $\Phi$ gives the expected behavior of the system. The system $S$ may affect the considered structure by changing the value of variables, by allocating or freeing memory locations, or by redirecting already allocated locations. For instance, we may check whether an entailment $lseg(x, \text{nil}) \models^R S F lseg(x, x)$ holds, meaning that an initial list segment from $x$ to $\text{nil}$ is eventually transformed into a circular list, or that $lseg(x, \text{nil}) \models^R S G (q \Rightarrow lseg(x, \text{nil}))$ holds, meaning that each time the system reaches state $q$ the heap contains a list from $x$ to $\text{nil}$. We show that the entailment problem is undecidable in general, but decidable if the considered transition system satisfies some conditions, which, intuitively, prevent actions affecting the value of the variables to occur inside loops (the other actions are not restricted). The proposed decision procedure is modular, and relies on a combination of the algorithm described in [12, 11] for checking the satisfiability of separation logic formulas with usual model checking and model construction procedures for LTL.

Related work

Dynamic transformations are usually tackled in SL using Hoare logic, with pre and post-conditions defined with the help of separating implications (see, e.g., [1]). Separating implication is not used in our approach due to the difficulty of reasoning automatically with this connective, especially in connection with inductive definitions (however, the so-called context predicates introduced in Section 7 can be viewed as a restricted form of separating implication). The combination of SL with temporal connectives is rather natural and has been considered in [2]. In [8, 6], temporal extensions of the related bunched logic are considered. Our approach departs from this work because the fragment of separation logic that we consider is very different: while the logic in [2] is based on quantifier-free separation logic formulas (with arbitrary combinations of boolean and separating connectives), we focus on symbolic heaps, i.e., on separating conjunctions of inductively defined atoms (with existential quantification). Thus on one hand our basic assertion language is more restricted because we strongly restricts the nesting of separation connectives, but on the other hand the addition of inductively defined predicates greatly increases the expressive power of the language and allows one to tackle richer data structures. In particular we emphasize that – without temporal connectives – entailment is $2$-EXPTIME complete for the fragment that we consider, whereas satisfiability is $\text{PSPACE}$-complete for that considered in [2].

2 Separation Logic

We define the syntax and semantics of a fragment of separating logic called symbolic heaps and we recall the conditions on the inductive rules that ensure that the entailment problem is decidable. Most definitions are standard, see [14, 7] for additional explanations and examples.

▶ Definition 1 (Symbolic Heaps). Let $V$ be a countably infinite set of variables. Let $P$ be a finite set of predicate symbols. Each symbol $p$ in $P$ is associated with a unique natural number called the arity of $p$. Let $\kappa$ be a fixed natural number, denoting the number of record fields. An equational atom is an expression of the form $x \equiv y$ or $x \not\equiv y$, where $x, y \in V$. A points-to
atom is an expression of the form \( x \mapsto (y_1, \ldots, y_n) \) with \( x, y_1, \ldots, y_n \in V \). A predicate atom is an expression of the form \( p(x_1, \ldots, x_n) \) with \( p \in P \), \( n = \text{arity}(p) \) and \( x_1, \ldots, x_n \in V \). A spatial atom is either a points-to atom or a predicate atom. An atom is either an equational atom or a spatial atom. The set of symbolic heaps is the set of expressions of the form: 
\[ \exists x_1 \ldots \exists x_n.(\alpha_1 \cdots \cdots \alpha_m) \] where \( x_1, \ldots, x_n \) are variables and \( \alpha_1, \ldots, \alpha_m \) are atoms (with possibly \( n = 0 \) and/or \( m = 0 \)). The connective \( * \) is called separating conjunction. An empty separating conjunction is denoted by \( \text{emp} \). For every symbolic heap \( \phi \), we denote by \( \text{fv}(\phi) \) the set of variables freely occurring in \( \phi \).

For all vectors \( x = (x_1, \ldots, x_n) \) and \( y = (y_1, \ldots, y_n) \) of the same length, we denote by \( x \preceq y \) the separating conjunction \( x_1 \simeq y_1 \cdots \cdots x_n \simeq y_n \). If \( \phi \) is a symbolic heap, then \( \exists x.\phi \) denotes the symbolic heap \( \exists x_1 \ldots \exists x_n.\phi \). For every symbolic heap \( \phi \) we denote by \( \alpha_{\omega}(\phi) \) the set of free variables \( x \) such that \( \phi \) contains a points-to atom of the form \( x \mapsto (y) \).

**Definition 2 (Substitutions).** A substitution is a function mapping every variable \( x \) to a variable. For every substitution \( \sigma \) and for every symbolic heap \( \phi \), we denote by \( \sigma \phi \) the symbolic heap obtained from \( \phi \) by replacing every free occurrence of a variable \( x \) by \( \sigma(x) \). If \( x_1, \ldots, x_n \) are pairwise distinct variables, we denote by \( \{ x_1 \leftarrow y_1, \ldots, x_n \leftarrow y_n \} \) the substitution \( \sigma \) such that \( \sigma(x_i) = y_i \) for all \( i = 1, \ldots, n \) and \( \sigma(x) = x \) if \( x \not\in \{ x_1, \ldots, x_n \} \).

Symbolic heaps are interpreted in structures defined as follows.

**Definition 3 (SL Structures).** Let \( \mathcal{L} \) be a countably infinite set of so-called locations. An (SL) structure is a pair \( (s, h) \) where:
- \( s \) is a store, i.e., a function mapping every variable to a location.
- \( h \) is a heap, i.e., a finite partial function mapping locations to \( \kappa \)-tuples of locations. We denote by \( \text{dom}(h) \) the finite domain of \( h \), by \( |h| \) the cardinality of \( \text{dom}(h) \) and by \( \text{locs}(h) \) the set: \( \{ \ell_1 \mid \ell_0 \in \text{dom}(h), h(\ell_0) = (\ell_1, \ldots, \ell_n), 0 \leq i \leq \kappa \} \).

A location \( \ell \in \text{dom}(h) \) is allocated in \( h \). A variable \( x \) such that \( s(x) \in \text{dom}(h) \) is allocated in \( (s, h) \).

Intuitively, \( s \) gives the values of the variables and \( h \) denotes the dynamically allocated memory.

A heap will often be denoted as a set of tuples \( h = \{ (\ell_0, \ldots, \ell_n) \mid \ell_0 \in \text{dom}(h), h(\ell_0) = (\ell_1, \ldots, \ell_n) \} \). In particular, \( \emptyset \) denotes the heap that allocates no location. Two heaps \( h_1 \) and \( h_2 \) are disjoint if \( \text{dom}(h_1) \cap \text{dom}(h_2) = \emptyset \). In this case \( h_1 \sqcup h_2 \) denotes the union of \( h_1 \) and \( h_2 \) defined as follows: \( \text{dom}(h_1 \sqcup h_2) \equiv \text{dom}(h_1) \cup \text{dom}(h_2) \), and \( h(\ell) = h_1(\ell) \) for all \( i = 1, 2 \) and \( \ell \in \text{dom}(h_1) \).

The semantics of the predicate symbols is defined by user-provided inductive rules:

**Definition 4 (Inductive Rules).** A set of inductive definitions (SID) is a set of rules of the form \( p(x_1, \ldots, x_n) \Leftarrow \phi \) such that \( p \in P \), \( n = \text{arity}(p) \), \( x_1, \ldots, x_n \) are pairwise distinct variables, and \( \phi \) is a symbolic heap with \( \text{fv}(\phi) \subseteq \{ x_1, \ldots, x_n \} \).

For every symbolic heap \( \phi \), we write \( \phi \Leftarrow_R \phi' \) if \( \phi \) is of the form \( \exists u.(p(y_1, \ldots, y_n) * \phi') \), \( R \) contains a rule \( p(x_1, \ldots, x_n) \Leftarrow \exists v.\psi \) (where \( \psi \) contains no quantifier) and \( \phi' = \exists u.\exists v.(\psi(x_i \leftarrow y_i \mid i = 1, \ldots, n) * \phi') \). We assume by \( \alpha \)-renaming that the vector \( u \) contains no variable in \( u \), \( \text{fv}(\phi) \) or \( (x_1, \ldots, x_n) \). As usual \( \Leftarrow_R \) is the reflexive and transitive closure of \( \Leftarrow_R \).

The satisfiability relation is defined inductively as follows. We emphasize that equational atoms are valid only if the heap is empty; this convention allows us to simplify notations (it avoids having to use both the separating conjunction and the standard one).
Definition 5 (Satisfiability). We write \((\sigma, h) \models_{\mathcal{R}} \phi\) if one of the following conditions holds:
- \(h = \emptyset\) and either \((\phi = (x \simeq y)\) and \(\sigma(x) = \sigma(y)\), or \((\phi = (x \not\simeq y)\) and \(\sigma(x) \neq \sigma(y)\)).
- \(\phi = x \mapsto (y_1, \ldots, y_n)\) and \(h = \{(\sigma(x), \sigma(y_1), \ldots, \sigma(y_n))\}\).
- \(\phi = \phi_1 \ast \phi_2\) and there exist disjoint heaps \(h_1\) and \(h_2\) such that \(h = h_1 \uplus h_2\) and \((\sigma, h_1) \models_{\mathcal{R}} \phi_1\), for all \(i = 1, 2\).
- \(\phi = p(x_1, \ldots, x_n)\) with \(p \in \mathcal{P}\), \(p(x_1, \ldots, x_n) \iff_{\mathcal{R}} \psi\), \(\psi\) contains no predicate symbols and \((\sigma, h) \models_{\mathcal{R}} \psi\).
- \(\phi = \exists x. \psi\), and there exists a store \(s'\) coinciding with \(s\) on all variables distinct from \(x\) such that \((s', h) \models_{\mathcal{R}} \psi\).

An \(\mathcal{R}\)-model of \(\phi\) is a structure \((\sigma, h)\) such that \((\sigma, h) \models_{\mathcal{R}} \phi\). If \(\phi, \phi'\) are symbolic heaps, we write \(\phi \models_{\mathcal{R}} \phi'\) if the entailment \((\sigma, h) \models_{\mathcal{R}} \phi \implies (\sigma, h) \models_{\mathcal{R}} \phi'\) for all SL structures \((\sigma, h)\), and \(\phi \equiv_{\mathcal{R}} \psi\) if \(\phi \models_{\mathcal{R}} \psi\) and \(\psi \models_{\mathcal{R}} \phi\).

Restricting Inductive Definitions

While the entailment problem is undecidable in general for symbolic heaps with inductively defined predicates, a very general decidable class is identified in [7]. This fragment is defined by restricting the form of the inductive rules, which must satisfy three conditions, recalled below (we use the slightly more general version of establishment given in [11]).

Definition 6 (Progress, Connectedness and Establishment (PCE)). A rule \(p(x_1, \ldots, x_n) \leftarrow \exists y. \phi\) (where \(\phi\) contains no quantifier) is:
- progressing if \(\phi\) is of the form \(x_1 \mapsto (z_1, \ldots, z_n) \ast \phi'\), where \(\phi'\) contains no points-to atom (i.e., the rule allocates exactly one location \(x_1\));
- connected if, moreover, every predicate atom in \(\phi'\) is of the form \(q(z, \nu)\) with \(z \in \{z_1, \ldots, z_n\}\) (i.e., the locations allocated by the called predicates are successors of \(x_1\)).

A SID \(\mathcal{R}\) is progressing (resp. connected) if all the rules in \(\mathcal{R}\) are progressing (resp. connected).

It is is established if for every atom \(p(x_1, \ldots, x_n)\) and for every formula \(\phi\) containing no predicate symbol, if \(p(x_1, \ldots, x_n) \iff_{\mathcal{R}} \phi\) and \(x\) is existentially quantified in \(\phi\) then \(\phi\) contains atoms \(y_i \simeq y_{i+1}\) (for \(i = 0, \ldots, n\), with \(n \geq 0\)) such that \(x = y_{n+1}\) and either \(\phi\) contains a points-to atom of the form \(y_0 \mapsto (z)\) or \(y_0 \in \{x_1, \ldots, x_n\}\) (i.e., every existentially quantified variable either is equal to a free variable or is eventually allocated).

Example 7. The following set, defining a list segment ending at an arbitrary location, is progressing and connected, but not established:

\[
\{\text{lseg}'(x) \iff \exists y. x \mapsto (y), \quad \text{lseg}'(x) \iff \exists z. (x \mapsto (z) \ast \text{lseg}'(z))\}
\]

In the remainder of the paper we assume that a set of inductive rules \(\mathcal{R}\) is given, satisfying the PCE conditions. This is the case for the rules given in the Introduction for the predicate \text{lseg}. We now introduce a notion of heap constraints, which combine positive and negative assertions denoted by symbolic heaps, with constraints specifying that some variables are unallocated:

Definition 8 (Heap Constraint). A heap constraint is a triple \((S^+, S^-, X)\), where \(S^+\) and \(S^-\) are sets of symbolic heaps, \(S^+ \neq \emptyset\) and \(X \subseteq \mathcal{V}\). We write \((\sigma, h) \models_{\mathcal{R}} (S^+, S^-, X)\) if for all \(\phi \in S^+\): \((\sigma, h) \models_{\mathcal{R}} \phi\); for all \(\phi \in S^-\): \((\sigma, h) \not\models_{\mathcal{R}} \phi\); and for all \(x \in X\): \(\sigma(x) \notin \dom(h)\).

The decidability of the satisfiability problem for such constraints follows from [12, 11]:

Lemma 9. There exists an algorithm that, given a heap constraint \((S^+, S^-, X)\) and a progressing, connected and established set of rules \(\mathcal{R}\), checks whether there exists an SL structure \((\sigma, h)\) such that \((\sigma, h) \models_{\mathcal{R}} (S^+, S^-, X)\).
3 Actions Operating on SL Structures

We define the basic actions that can occur in transition systems. The set of actions includes tests, affectations, redirections of allocated locations, as well as allocations and desallocations.

Definition 10 (Actions). Let $V^*$ be a finite set of variables. A term is either an element of $V^*$ or an expression of the form $x.i$ where $x \in V^*$ and $i \in \{1, \ldots, \kappa\}$. A condition is a boolean combination of atomic conditions, that are expressions of the form $t \approx s$ where $t, s$ are terms. An action is an expression of one of the following forms: pass (null action); $t := s$, where $t$ and $s$ are terms (affectation or redirection); alloc$(x)$ or free$(x)$, where $x \in V^*$ (allocation and desallocation); or test$(\gamma)$, where $\gamma$ is a condition (test).

The semantics of conditions is defined below.

Definition 11 (Semantics of Conditions). For every structure $(s, h)$ and for every term $t$ we write $t \triangleright (s, h) \ell$ ($t$ evaluates to $\ell$) if either $t \in V^*$ and $s(\ell) = \ell$, or $t = x.i$ with $x \in V^*$, $s(x) \in \text{dom}(h)$, $h(s(x)) = (\ell_1, \ldots, \ell_n)$ and $\ell = \ell_i$. We write $(s, h) \models t \approx s$ if there exists $\ell \in L$ such that $t \triangleright (s, h) \ell$ and $s \triangleright (s, h) \ell$. The relation $\models$ is extended to every boolean combination of atomic conditions inductively as usual.

Observe that the semantics of $x \approx y$ is different from that of $x \simeq y$, which requires that the heap be empty. Furthermore, $(s, h) \models x.i \approx x.i$ (for $i = 1, \ldots, \kappa$) holds iff $x$ is allocated. We thus denote by $h(x)$ (for “$x$ is allocated”) the formula $x.1 \approx x.1$. The semantics of actions is rather natural, and formally defined below (to make allocations deterministic we assume that the variable allocated by alloc$(x)$ points to itself).

Definition 12 (Semantics of Actions). For every SL structure $(s, h)$ and action $a$, we denote by $(s, h)[a]$ the result of the application of the action $a$ on $(s, h)$ defined as follows:

- If $a = \text{pass}$ then $(s, h)[a] \overset{\triangleleft}{=} (s, h)$.
- If $a = (x := s)$ with $x \in V^*$ and $s \triangleright (s, h) \ell$ then $(s, h)[a] \overset{\triangleleft}{=} (s', h)$, where $s'(x) = \ell$ and $s'$ coincides with $s$ on all variables distinct from $x$.
- If $a = (x.i := s)$ with $x \in V^*$, $s(x) \in \text{dom}(h)$, $h(s(x)) = (\ell_1, \ldots, \ell_n)$ and $s \triangleright (s, h) \ell$ then $(s, h)[a] \overset{\triangleleft}{=} (s, h')$, where $\text{dom}(h') = \text{dom}(h)$, $h'(s(x)) = (\ell_1, \ldots, \ell_{i-1}, \ell, \ell_{i+1}, \ldots, \ell_n)$, and $h'$ coincides with $h$ on all locations distinct from $s(x)$.
- If $a = \text{free}(x)$, $s(x) \in \text{dom}(h)$ then $(s, h)[a] \overset{\triangleleft}{=} (s, h')$ where $\text{dom}(h') = \text{dom}(h) \setminus \{s(x)\}$ and $h'$ coincides with $h$ on all locations distinct from $s(x)$.
- If $a = \text{alloc}(x)$, $s(x) \notin \text{dom}(h)$ then $(s, h)[a] \overset{\triangleleft}{=} (s, h')$ where $\text{dom}(h') = \text{dom}(h) \cup \{s(x)\}$, $h(s(x)) = (s(x), \ldots, s(x))$ and $h'$ coincides with $h$ on all locations distinct from $s(x)$.
- If $a = \text{test}(\gamma)$ and $(s, h) \models \gamma$ then $(s, h)[a] \overset{\triangleleft}{=} (s, h)$.

Otherwise $(s, h)[a]$ is undefined.

Proposition 13. For all structures $(s, h)$ and actions $a$, if $(s', h') = (s, h)[a]$ then $s'(V^*) \cup \text{locs}(h') \subseteq s(V^*) \cup \text{locs}(h)$.

Proof. By an immediate case analysis on the set of actions.
4 Transition Systems

Transition systems are finite state automata where the transitions are labeled by actions:

Definition 14 (Transition Systems). Let \( S \) be a countably infinite set of states. A transition system is a triple \( S = (Q, R, q_I) \) where \( Q \) is a finite subset of \( S \), \( R \) is a finite set of transition rules of the form \( (q, a, q') \) where \( q, q' \in Q \) and \( a \) is an action, and \( q_I \in Q \) is the initial state.

A run in \( S \) from a structure \( (s, h) \) is an infinite sequence of tuples \( (q_i, s_i, h_i, a_i) \) such that \( q_0 = q_I, (s_0, h_0) = (s, h) \), and for every \( i \in \mathbb{N} \): \( (q_i, a_i, q_{i+1}) \in R \) and \( (s_i, h_i) \) must be defined, for all \( i \in \mathbb{N} \). For simplicity we assume that all runs are infinite (finite runs may be encoded if needed by adding a final state \( q_F \) with a transition \( (q_F, \text{pass}, q_F) \)).

Example 15. The following transition system adds an element \( x \) to a list starting at \( y \):

Example 16. The following transition system desallocates a list segment from \( x \) to \( y \).

5 Temporal Formulas

We now define temporal formulas built over a set of assertions containing symbolic heaps, states, actions and conditions, using the usual set of LTL connectives:

Definition 17 (Syntax of LTL Formulas). The set \( \mathcal{A}_S \) of LTL atoms contains all symbolic heaps \( \phi \) with \( \text{fv}(\phi) \subseteq V^* \), all atomic conditions, all actions and all states in \( S \). The set of LTL formulas is the least set containing \( \mathcal{A}_S \) and such that for all LTL formulas \( \Phi, \Psi \): \( \neg \Phi \), \( \Phi \lor \Psi \), \( \Phi \land \Psi \), \( X \Phi \), \( U \Phi \Psi \) are LTL formulas.

The additional connectives \( \land, F, R \) etc. are defined as usual. The semantics of LTL formulas is recalled below. Note that LTL atoms are interpreted arbitrarily at this point.

Definition 18 (Semantics of LTL Formulas). An LTL interpretation \( I \) is a mapping from \( \mathcal{A}_S \times \mathbb{N} \) to \{true, false\}. For any LTL formula \( \Phi \), we write \( I \models \Phi \) if \( (I, 0) \models \Phi \), and \( (I, i) \models \Phi \) iff one of the following conditions holds:
An LTL interpretation \( I \) is compatible with a run \((q_i,s_i,h_i,a_i)_{i\in\mathbb{N}}\) in \( S \) w.r.t. a formula \( \Phi \) if the following conditions hold, for all \( i \in \mathbb{N} \):

- For every symbolic heap or condition \( \phi \) occurring in \( \Phi \), \( I(\phi,i) = true \) if and only if \( (s_i,h_i) \models_R \phi \).
- For all actions \( a \), \( I(a,i) = true \) if and only if \( a = a_i \).
- For all states \( q \in Q \), \( I(q,i) = true \) if and only if \( q = q_i \).

An LTL interpretation \( I \) is compatible with an SL structure \( (s,h) \) and a transition system \( S = (Q,R,q_0) \), w.r.t. a formula \( \Phi \) if it is compatible with some run \((q_i,s_i,h_i,a_i)_{i\in\mathbb{N}}\) in \( S \).

We are now in the position to define the satisfiability relation that relates SL structures to LTL formulas, w.r.t. a given transition system.

**Definition 20 (Entailment).** Let \( S \) be a transition system. For every structure \( (s,h) \) and LTL formula \( \Phi \), we write \( (s,h) \models_S^R \Phi \) if \( I \models \Phi \) for every LTL interpretation compatible with \( (s,h) \) and \( S \) w.r.t. \( \Phi \). We write \( (s,h) \models_R^{S/(q_i,a_i)_{i\in\mathbb{N}}} \Phi \) if there exists a run \((q_i,s_i,h_i,a_i)_{i\in\mathbb{N}}\) in \( S \) with \( s_0 = s \) and \( h_0 = h \), and an LTL interpretation \( I \) that is compatible with \((q_i,s_i,h_i,a_i)_{i\in\mathbb{N}}\) such that \( I \models \Phi \).

For every symbolic heap \( \phi \), we write \( \phi \models_R^S \Phi \) if the entailment \( (s,h) \models_R \phi \) implies \( (s,h) \models_S^R \Phi \) holds for all structures \((s,h)\).

**Example 21.** If \( S \) is the transition system of Example 15, then the entailments \( 1seg(y,z) \models_R^S F 1seg(x,z) \) and \( 1seg(y,z) \models_R^S X X G 1seg(x,z) \) are valid. Note that the structures in which \( x \) is initially allocated are not considered for testing the entailment.

If \( S \) now denotes the transition system of Example 16, then the entailment \( 1seg(x,y) \models_R^S F emp \) is not valid (because the initial list segment may be cyclic).

It is easy to see that model checking is decidable:

**Lemma 22.** The problem of checking whether \((s,h) \models^{S/(q_i,a_i)_{i\in\mathbb{N}}}^R \Phi \) is decidable (if the sequence \((q_i,a_i)_{i\in\mathbb{N}} \) is ultimately periodic).

**Proof.** Since \( s, h, q_i \) and \( a_i \) are given, the run (if it exists) \((q_i,s_i,h_i,a_i)_{i\in\mathbb{N}}\) such that \( s_0 = s \) and \( h_0 = h \), and the compatible LTL interpretation \( I \) are easy to compute, using Definition 11. Using Proposition 13, we get \( s_i(V^*) \cup locs(h_i) \subseteq s(V^*) \cup locs(h) \) for all \( i \in \mathbb{N} \), thus the set of structures \( \{(s_i,h_i) \mid i \in \mathbb{N}\} \) is necessarily finite. Thus, the interpretation \( I \) is ultimately periodic and the test \( I \models \Phi \) can be performed using well-known algorithms for LTL.

However, the entailment problem is undecidable in general:

**Theorem 23.** The problem of checking whether \( \phi \models_R^S \Phi \) is undecidable (even if \( R \) is progressing, connected and established).
9.8 Reasoning on Dynamic Transformations of Symbolic Heaps

Proof (Sketch). Turing machines (TM) may be simulated by transition systems: the elements of the alphabet are denoted by pairwise distinct free variables, a tape \((x_1, \ldots, x_n)\) is encoded as a heap (denoting a doubly linked list): \(\{(\ell_i, \ell_i', \ell_{i-1}, \ell_{i+1}) \mid i = 1, \ldots, n\}\) with \(\ell_i' = \mathsf{s}(x_i)\), and the position of the head is denoted by a variable \(x\). Moves are encoded by actions \(x := \star 2\) (left move) or \(x := \star 3\) (right move). Tests are performed by actions of the form \(\mathsf{test}(x.1 \approx y)\), where \(y\) is the variable associated with the considered symbol. The action \(x.1 := y\) writes \(y\) at the current position in the tape. Note that if the initial heap does not contain enough allocated locations then the transition system may be “stuck” (because a right move cannot be applied, hence no run will exist).

However, the following rules define a predicate \(p\) that allocates a tape of arbitrary size filled with a symbol \(u\) (which may be instantiated by a blank denoted by some free variable \(b\)). The variables \(y\) and \(z\) denote the start and the end of the tape, respectively:

\[
\{p(x, y, z, u) : x \mapsto (u, y, z), \ p(x, y, z, u) \equiv \exists x'. (x \mapsto (u, y, x')) \star p(x', x, z, u)\}.
\]

It is easy to check that the non-termination of the considered TM (on an empty tape) can be checked by testing whether the entailment \(p(x, y, z, b) \models S\mathcal{G}(\neg \bigvee_{q \in Q_F} q)\) holds, where \(Q_F\) is the set of final states (note however that the entailment \(p(x, y, z, b) \models S\mathcal{F}(\neg \bigvee_{q \in Q_F} q)\) does not encode termination, as it may have counter models in which \(p(x, y, z, b)\) does not allocate enough memory cells to execute the TM).

To overcome this issue we require that no action of the form \(x := t\) occurs inside a loop:

\begin{itemize}
  \item \textbf{Definition 24 (Oriented Transition System).} A transition system \(\mathcal{S} = (Q, R, q_I)\) is oriented if for every transition \((q, a, q')\) in \(R\), if \(a\) is of the form \(x := t\) then \(q >_\mathcal{S} q'\).
\end{itemize}

The transition system of Example 15 is oriented, but not that of Example 16.

\section{Symbolic Execution of Actions}

We now show how to execute actions symbolically on SL formulas. We first define an LTL formula encoding the conditions ensuring that an action can be performed:

\begin{itemize}
  \item \textbf{Definition 25 (Precondition).} For all actions \(x\), \(\mathsf{pre}(a)\) is defined as follows (with \(x, y \in V^*\)):
    \begin{align*}
    \mathsf{pre}(\mathsf{alloc}(x)) & \triangleq \neg A(x), \ \mathsf{pre}(\mathsf{free}(x)) \triangleq A(x), \ \mathsf{pre}(x.1 := y) \triangleq A(x), \ \mathsf{pre}(x := y.1) \triangleq A(y), \\
    \mathsf{pre}(x.i := y.j) & \triangleq A(x \wedge A(y)), \ \mathsf{pre}(x) & \triangleq \gamma \text{ and } \mathsf{pre}(a) \triangleq 1 \text{ otherwise.}
    \end{align*}
  \item \textbf{Proposition 26.} For every action \(a\) and for every structure \((s, h)\), \((s, h) \models_R \mathsf{pre}(a)\) iff \((s, h)[a]\) is defined.
\end{itemize}

Proof. Immediate.

Given a symbolic heap \(\phi\) and action \(a\), it is sometimes possible to compute the strongest postcondition of \(\phi\) w.r.t. to \(a\), which describes the state of the memory after action \(a\) is performed on a structure satisfying \(\phi\):

\begin{itemize}
  \item \textbf{Definition 27 (Strongest Postcondition).} For every symbolic heap \(\phi\) and for every action \(a\), we define a formula \(\mathsf{spec}(\phi, a)\) (strongest postcondition of \(\phi\) w.r.t. \(a\)) as follows (where \(x'\) denotes a fresh variable).
    \begin{itemize}
      \item \(\mathsf{spec}(\exists y. \phi, \mathsf{alloc}(x)) \triangleq \exists y.(x \mapsto (x, \ldots, x) \star \phi).
      \item \(\mathsf{spec}(\exists y.(x \mapsto (y_1, \ldots, y_n) \star \phi), \mathsf{free}(x)) \triangleq \exists y.\phi.
      \item \(\mathsf{spec}(\phi, x.1 := y) \triangleq \exists x'.(\phi(x \mapsto x') \star x \approx y) \quad (\text{if } x, y \in V^*).
      \item \(\mathsf{spec}(\exists u.(\phi \star y \mapsto (y_1, \ldots, y_n)), x := y.i) \triangleq \exists u \exists x'.((\phi \star y \mapsto (y_1, \ldots, y_n))(x \mapsto x') \star x \approx y_i).
    \end{itemize}
\end{itemize}
Example 3.1. For instance, we have:

\[ \text{spc}(\exists u. (x \mapsto (x_1, \ldots, x_n) \ast \phi), x.i := z) \overset{\text{def}}{=} \exists u. (x \mapsto (x_1, \ldots, x_i, z, x_{i+1}, \ldots, x_n) \ast \phi) \] (if \( z \in \mathcal{V}^* \)).

\[ \text{spc}(\exists u. (x \mapsto (x_1, \ldots, x_n) \ast \phi), x.i := z) \overset{\text{def}}{=} \exists u. (x \mapsto (x_1, \ldots, x_i, z, x_{i+1}, \ldots, x_n) \ast \phi). \]

\[ \text{spc}(\exists u. (x \mapsto (x_1, \ldots, x_n) \ast \phi), x.i := z) \overset{\text{def}}{=} \exists y. (x \mapsto (x_1, \ldots, x_i, y, x_{i+1}, \ldots, x_n) \ast \phi) \] (if \( x \neq y \)).

Otherwise \( \text{spc}(\phi, a) \) is undefined.

In all cases, \( \phi \) may be emp.

Example 28. For instance, we have:

\[ \text{spc}(x \mapsto (y, z), x, 1 := x) = x \mapsto (x, y) \]

\[ \text{spc}(x \mapsto (y, z), x, 1 := x) = 3 x' \mapsto (y, z) \ast x \simeq y) \]

\[ \text{spc}(x \mapsto (y, z), \text{free}(x)) = \text{emp} \]

But both \( \text{spc}(x \mapsto (y, z), y, 1 := x) \) and \( \text{spc}(\text{lseg}(x, y), x, 1 := y) \) are undefined.

Lemma 29. Let \( \phi \) be a symbolic heap and let \( a \) be an action. If \( \text{spc}(\phi, a) \) is defined then for every structure \((s, h)\) such that \((s, h)[a]\) is defined, we have \((s, h) \models R \phi \implies (s, h)[a] \models R \text{spc}(\phi, a)\).

Proof. By inspection of the different actions, using Definition 11.

Similarly, it is possible in some cases to define the weakest precondition of a symbolic heap w.r.t. an action, asserting conditions that guarantee that the given formula is satisfied after the action is performed:

**Definition 30 (Weakest Precondition).** For every symbolic heap \( \phi \) and for every action \( a \), the formula \( \text{wpc}(\phi, a) \) is defined as follows (where \( x' \) denotes a fresh variable).

\[ \text{wpc}(\phi, \text{pass}) \overset{\text{def}}{=} \phi. \]

\[ \text{wpc}(\exists x. (\phi \ast x \mapsto (y_1, \ldots, y_n)), \text{alloc}(x)) \overset{\text{def}}{=} \exists x. (\phi \ast y_1 \simeq x \ast \cdots \ast y_n \simeq x). \]

\[ \text{wpc}(\exists x. \phi, \text{free}(x)) \overset{\text{def}}{=} \exists y_1 \exists y_2 \ldots \exists y_n. (\phi \ast x \mapsto (y_1, \ldots, y_n)). \]

\[ \text{wpc}(\phi, x := y) \overset{\text{def}}{=} \phi (x := y) \] if \( x, y \in \mathcal{V}^* \).

\[ \text{wpc}(\exists x. (\phi \ast x \mapsto (x_1, \ldots, x_n), x.i := y) \overset{\text{def}}{=} \exists x' (\phi \ast x \mapsto (x_1, \ldots, x_i, y', x_{i+1}, \ldots, x_n) \ast x_i \simeq y). \]

\[ \text{wpc}(\exists x. (\phi \ast x \mapsto (x_1, \ldots, x_n) \ast y \mapsto (y_1, \ldots, y_n)), x.i := y.j) \overset{\text{def}}{=} \exists x' (\phi \ast x \mapsto (x_1, \ldots, x_i, x', x_{i+1}, \ldots, x_n) \ast y \mapsto (y_1, \ldots, y_n) \ast x_i \simeq y_j). \]

\[ \text{wpc}(\exists x. (\phi \ast x \mapsto (x_1, \ldots, x_n), x.i := x.j) \overset{\text{def}}{=} \exists x' (\phi \ast x \mapsto (x_1, \ldots, x_i, x', x_{i+1}, \ldots, x_n) \ast x_i \simeq x_j). \]

\[ \text{wpc}(\exists x. (\phi \ast x \mapsto (x_1, \ldots, x_n)), y := y.i) \overset{\text{def}}{=} \exists x'. ((\phi \ast x \mapsto (x_1, \ldots, x_n)) (y \leftarrow x_i)) \] if \( x \neq y \). The case where \( x \ast y \) is handled by encoding the action \( x := x.i \) as the sequence \( z := x ; x := z.i \), where \( z \) is a special variable in \( \mathcal{V}^* \) not occurring in the considered transition system.

Otherwise, \( \text{wpc}(a, a) \) is undefined.

Example 31. For instance, we have:

\[ \text{wpc}(x \mapsto (y, z), x, 1 := x) = 3 x' \mapsto (x', z) \ast y \simeq x) \]

\[ \text{wpc}(x \mapsto (y, z), x := y) = y \mapsto (y, z) \]

\[ \text{wpc}(x \mapsto (y, z), \text{alloc}(x)) = y \ast x \ast z \simeq z \]

Both \( \text{wpc}(x \mapsto (y, z), \text{free}(y)) \) and \( \text{wpc}(\text{lseg}(x, y), x, 1 := y) \) are undefined.
Lemma 32. Let φ be a symbolic heap and let a be an action. If \( \text{wpc}(\phi, a) \) is defined then for every structure \((s, h)\) such that \((s, h)[a]\) is defined, we have \((s, h)[a] \models wpc(\phi, a) \iff (s, h)[a] \models wpc(\phi)\).

Proof. By inspection of the different cases.

Intuitively, the weakest pre-conditions will be used to propagate towards the initial time all the constraints occurring along the run, while strongest post-conditions will be used to ensure that, at any time, the shape of the heap can be described as a symbolic heap, so that all the conditions that hold along the run can be embedded in a heap constraint.

7 Context Predicates

As shown in the previous section, post and preconditions cannot be defined for all symbolic heaps. Indeed, in some cases, the conditions can be computed only if the consider formula contains some specific points-to atom(s) \(x \mapsto (\ldots)\), where \(x\) is some variable involved in the action (for instance for actions \(x.i := y\)). In this section we devise an algorithm that, given a symbolic heap \(\phi\) and a variable \(x\), returns a disjunction of symbolic heaps equivalent to \(\phi\) (on structures that allocate \(x\)), and such that all symbolic heaps contain a points-to atom of the form \(x \mapsto (\ldots)\). The latter condition will enable the computation of post and preconditions. To this aim, we consider so-called context predicates (adapted from [5]). For every pair of predicates \(p, q\) with \(\text{arity}(p) = n\) and \(\text{arity}(q) = m\), we define a predicate \((q \circ p)\) of arity \(n + m\) in such a way that \((q \circ p)(x_1, \ldots, x_n, y_1, \ldots, y_m)\) is satisfied by all (non empty) structures that will satisfy \(p(x_1, \ldots, x_n)\) after a disjoint heap satisfying \(q(y_1, \ldots, y_m)\) is added to the current heap. Intuitively, the rules of \((q \circ p)\) are defined exactly as those of \(p\), except that exactly one call to \(q(y_1, \ldots, y_m)\) is removed. More formally, for each rule \(p(u_1, \ldots, u_n) \Leftarrow \exists w. (u_1 \mapsto (y) \circ p'(z) \circ \psi)\) in \(R\) we introduce two rules:

\[
(q \circ p)(u_1, \ldots, u_n, v_1, \ldots, v_m) \Leftarrow \exists w. (u_1 \mapsto (y) \circ (q \circ p')(z, v_1, \ldots, v_m) \circ \psi)
\]

\[
(q \circ p)(u_1, \ldots, u_n, v_1, \ldots, v_m) \Leftarrow \exists w. (u_1 \mapsto (y) \circ \psi) \circ (v_1, \ldots, v_m) \circ \zeta \quad \text{if } q = p'
\]

It is easy to check that these rules fulfill the conditions of Definition 6. Note that the \(\circ\) operation may be nested, e.g., one may consider predicates such as \((lseg \circ lseg) \circ lseg\).

Thus \(R\) is actually infinite, and the rules must be computed on demand.

Example 33. For instance \((lseg \circ lseg)\) is defined by the rules:

\[
(lseg \circ lseg)(x, y, u, v) \Leftarrow \exists z. (x \mapsto (z) \circ z \simeq u \circ v \simeq y)
\]

and

\[
(lseg \circ lseg)(x, y, u, v) \Leftarrow \exists z. x \mapsto (z) \circ (lseg \circ lseg)(x, z, u, v)
\]

The proposed transformation algorithm relies on the use of these context predicates. The idea is that, by Definition 6, a variable \(x\) is allocated in a structure validating a predicate atom \(\phi\) iff the corresponding unfolding of \(\phi\) contains a predicate atom of the form \(q(z_1, \ldots, z_m)\), for some \(q \in \mathcal{P}\), where \(z_1\) has the same value as \(x\). Using context predicates it is possible to transform the formula in a way that this atom occurs explicitly in it, since a predicate atom \(p(y)\) calling \(q(z)\) is equivalent to \(q(z) \circ (q \circ p)(z, y)\). Then, it suffices to unfold this atom once to get a points-to atom of the form \(x \mapsto (\ldots)\). More formally:
**Definition 34** (Computation of $\langle \phi \rangle_x$). Let $\phi$ be a symbolic heap and let $x \in V^*$. The set $\langle \phi \rangle_x$ is defined as follows:

1. If $x \in v_+(\phi)$ then $\langle \phi \rangle_x \equiv \{ \phi \}$.
2. Otherwise, $\langle \phi \rangle_x$ is the set of formulas that are of one of the following forms:
   a. $\exists u(x \simeq x' \star x \mapsto (y \star \psi)$ where $\phi$ is of the form $\exists u(x' \mapsto (y \star \psi)$.
   b. $\exists \exists y(x \simeq x' \star y \star \psi)$ where $\phi$ is of the form $\exists \exists y(x' \mapsto (y \star \psi)$ and $p(x, y) \in \mathcal{R} \exists \psi'$.
   c. $\exists \psi \exists x_1 \ldots \exists x_m (((q \mapsto p(y, z_1, \ldots, z_m) \simeq z_1 \simeq x \star \psi \star \psi')$ where $\phi$ is of the form $\exists \psi \exists x_1 \ldots \exists x_m (((q \mapsto p(y, z_1, \ldots, z_m) \simeq z_1 \simeq x \star \psi \star \psi')$ and $q(x, z_1, \ldots, z_m) \in \mathcal{R} \exists \psi'$.

Item 1 corresponds to the trivial case where $\phi$ already contains an atom $x \mapsto (\ldots)$. Item 2a corresponds to the case where $\phi$ contains an atom $x \mapsto (\ldots)$ where $x \simeq x'$ holds. Item 2b handles the case where $\phi$ contains an atom $p(x', y)$ that (immediately) allocates $x$ (by the progress condition this happens if $x \simeq x'$ holds). Finally, Item 2c tackles the general case, where $\phi$ contains an atom $p(y)$ which (eventually) calls an atom $q(z_1, z_2, \ldots, z_m)$ that allocates $x$. For instance $\langle \mathsf{1seg}(x, y) \rangle_z$ contains the symbolic heaps $\exists u(z \mapsto (u \star \mathsf{1seg}(u, y) \simeq z)$ and $\exists u, v, w(lseg \mapsto \mathsf{1seg}(x, y, u, v) \simeq z \mapsto (w \star \mathsf{1seg}(w, v) \simeq u \simeq z)$. Note that both formulas contain a points-to atom of the form $z \mapsto (\ldots)$. The following lemma state that $\langle \phi \rangle_x$ fulfills all the expected properties.

**Lemma 35.** Let $\phi$ be a symbolic heap and let $x \in V^*$. For every formula $\psi \in \langle \phi \rangle_x$, $x \in v_+(\phi')$. Thus if $(s, h) \models \phi'$ then $s(x) \in \text{dom}(h)$.

**Proof.** Let $\phi' \in \langle \phi \rangle_x$. If $x \in v_+(\phi)$ then $\langle \phi \rangle_x = \{ \phi \}$ thus $\phi' = \phi$ and $x \in v_+(\phi')$. In all other cases in Definition 34, either $\phi'$ contains a points-to atom $x \mapsto (y)$, or $\phi'$ contains a formula $\psi'$ such that there exists an atom $\alpha$ of root $x$ ($\alpha$ is either $p(x, y)$ or $q(x, z_2, \ldots, z_m)$) such that $\alpha \in \mathcal{R} \exists \psi'$. By the progress condition necessarily $x \in v_+(\phi')$, so that $x \in v_+(\phi')$. The second part of the lemma follows immediately from the definition of the semantics.

**Lemma 36.** Let $\phi$ be a symbolic heap and let $x \in V^*$. For every formula $\psi \in \langle \phi \rangle_x$ and for all SL structures $(s, h)$: $(s, h) \models \psi \Rightarrow (s, h) \models \phi$.

**Lemma 37.** Let $\phi$ be a symbolic heap and let $x \in V^*$. For every SL structure $(s, h)$ such that $s(x) \in \text{dom}(h)$ and $(s, h) \models \phi$, we have $(s, h) \models \psi$, for some $\psi \in \langle \phi \rangle_x$.

### 8 Axioms

Building on the previous results, we define LTL axioms ensuring that an LTL interpretation is compatible with some SL structure, for a given transition system $S = (Q, R, q_1)$. The axioms are obtained by embedding all the previous definitions and properties in LTL ($a, \gamma$ and $x$ range over the set of actions, symbolic heaps, conditions and variables in $V^*$, respectively and $t, s$ are terms).

1. $G(x.i = s \Rightarrow \mathsf{a}(x))$ for all $i \in \{1, \ldots, \kappa\}$.
2. $G(a \Rightarrow (\psi \Rightarrow X \text{spec}(\psi, a)))$ (if $\text{spec}(\psi, a)$ is defined).
3. $G(a \Rightarrow (\text{wpc}(\psi, a) \Rightarrow X \psi))$ (if $\text{wpc}(\psi, a)$ is defined).
4. $G(\langle \mathsf{a}(x) \Rightarrow \psi \Rightarrow \bigvee_{y \in \text{dom}(\psi)} \mathsf{a}(y)\rangle) \land G(\psi \Rightarrow \bigwedge_{y \in \text{dom}(\psi)} \mathsf{a}(y))$.
5. $G(\exists u(x \mapsto ((x_1, \ldots, x_n) \star \psi) \Rightarrow (x.i \simeq y \Rightarrow \exists u(x \mapsto ((x_1, \ldots, x_n) \star \psi \star x_i \simeq y)))$, if $y \in V^*$.
6. $G(\exists u(x \mapsto ((x_1, \ldots, x_n) \star y \mapsto (y_1, \ldots, y_k) \star \psi) \Rightarrow (x.i \simeq y \Rightarrow \exists u(x \mapsto ((x_1, \ldots, x_n) \star y \mapsto (y_1, \ldots, y_k) \star \psi \star x_i \simeq y)))$.
7. $G ((\exists u. \psi) \Rightarrow (x \simeq y \Rightarrow \exists u.(\psi \star x \simeq y)))$. 

**TIME 2022**
9:12 Reasoning on Dynamic Transformations of Symbolic Heaps

8. \( G(\text{pass} \lor t := s \lor \text{test}(y)) \Rightarrow (A(x) \Leftrightarrow X A(x)), \) where \( t \neq x. \)

9. \( G(\text{true}(x) \Rightarrow \bigwedge_{x \in V}(x \Rightarrow X \neg A(y))) \land (x \neq y \Rightarrow (A(y) \Leftrightarrow X A(y))). \)

10. \( G(\text{alloc}(x) \Rightarrow \bigwedge_{y \in V}.((x \Rightarrow y) \Rightarrow X A(y))) \land (x \neq y \Rightarrow (A(y) \Leftrightarrow X A(y))). \)

11. \( G(\neg x \lor \neg y), \) if \( x \neq y \) and (either \( x, y \) are both actions, or \( \{x, y\} \subseteq Q). \)

12. \( G(q \Rightarrow \bigvee_{(q, a, q') \in R}(a \land X q)). \)

13. \( G(a \Rightarrow \text{prev}(a)). \)

14. \( \bigwedge_{x \in S^+} \phi \land \bigwedge_{x \in S^-} \neg A(x) \Rightarrow \bigvee_{x \in S^-} \phi, \) if \( (S^+, S^-, X) \) is an unsatisfiable heap constraint. This formula is denoted by \( \Gamma(S^+, S^-, X) \) in the following.

This set of axioms is infinite, as the set of symbolic heaps is infinite. To ensure termination, we need to further restrict the axioms. To this aim, we define (given a symbolic heap \( \phi \)) two sets \( Fw(S, \phi) \) and \( Bw(S, \phi, \Phi) \), which, informally, contain triples \( (\psi, q, X) \), where \( \psi \) denotes a symbolic heap obtained by (forward or backward) propagation along the runs in \( S \) (starting from formulas occurring in the initial entailment), and \( q \) is the corresponding state. The set \( X \) contains variables that either occur in \( \nu_\phi(\phi) \) or are known to be non allocated at state \( q \) (this information is essential for finiteness because it allows one to “block” some generation rules). The sets are defined inductively as follows:

- \( (\phi, q_1, \emptyset) \in Fw(S, \phi) \), if and only if \( q \in Q \) and \( \psi \) occurs in \( \Phi \) then \((\psi, q, \emptyset) \in Bw(S, \phi, \Phi) \).

- If \((\psi, q, X) \in Fw(S, \phi), (q, a, q') \in R \) and \( \phi' = \text{spc}(\psi, a) \) then \((\psi', q', X') \in Fw(S, \phi) \), where \( X' = X \) if \( a \) is not of the form \( x := t \) with \( x \in \nu_\phi \), and otherwise \( X' = \emptyset \).

- If \((\psi, q', X) \in Bw(S, \phi, \Phi), (q, a, q') \in R \) and \( \phi' = \text{wpc}(\psi, a) \) then \((\phi', q', X') \in Bw(S, \phi, \Phi) \), where \( X' = \emptyset \) if \( a \) is of the form \( x := t \) with \( x \in \nu_\phi \), and otherwise \( X' = \emptyset \).

- If \((\psi, q, X) \in Fw(S, \phi) \) (resp. \((\psi, q, X) \in Bw(S, \phi, \Phi) \)) and \( \xi \in \psi \) then \((\xi, X \cup \{x\}) \in Fw(S, \phi) \) (resp. \((\xi, X \cup \{x\}) \in Bw(S, \phi, \Phi) \)).

- If \( (\exists u. \psi, q, X) \in Fw(S, \phi) \) then \((\exists u. (\psi \land x \approx y), q, X) \in Bw(S, \phi, \Phi) \), for all \( x, y \in \nu_\phi \cup \nu_\phi^* \).

The sets \( Fw(S, \phi) \) and \( Bw(S, \phi, \Phi) \) are finite (up to some simplifications) if \( S \) is oriented (see Lemma 41 in Appendix D). We denote by \( A(R, S, \phi) \) the set of axioms satisfying the following conditions. For Axiom 3 we require that the considered symbolic heap \( \psi \) occurs in some triple in \( Bw(S, \phi, \Phi) \). For Axiom 14 all the symbolic heaps in \( S^+ \) and \( S^- \) must occur in \( Bw(S, \phi, \Phi) \). For Axiom 4 \( \psi \) must occur in either \( Fw(S, \phi) \) or \( Bw(S, \phi, \Phi) \). For 5, 6 and 7, the symbolic heap at the left-hand side of \( \Rightarrow \) must occur in \( Fw(S, \phi) \) (which entails that the one occurring at the right-hand side occurs in \( Bw(S, \phi, \Phi) \)). The following theorems relate the considered entailment problem with standard LTL satisfiability.

1. **Theorem 38.** Every LTL model \( \mathcal{I} \) that is compatible with \((s, h) \) and \( S \) w.r.t. all symbolic heaps occurring in \( A(R, S, \phi) \cup \{\phi, q_1, \Phi\} \) satisfies \( A(R, S, \phi) \cup \{\phi, q_1, \Phi\} \).

**Proof (Sketch).** The soundness of Axioms 2 and 3 stems from Lemmata 29 and 32, respectively. The soundness of Axiom 13 stems from Proposition 26. Axioms 12 and 11 encode the semantics of actions and states, according to the transition system \( S \). The soundness of Axiom 4 is a consequence of Lemmata 36 and 37. The soundness of Axioms 14 follows from the semantics of heap constraints. The soundness of Axioms 8, 9, 10 is a consequence of Definition 11. Finally, the soundness of Axioms 1, 5, 6 and 7 stems from the semantics of atomic conditions (Axioms 5, 6 and 7 embed conditions of the form \( t \approx s \) into symbolic heaps).

2. **Theorem 39.** If \( A(R, S, \phi) \cup \{\phi, q_1, \Phi\} \) admits an LTL model \( \mathcal{I} \) then there exists a structure \((s, h) \) such that \( \mathcal{I} \) is compatible with \((s, h) \) and \( S \), w.r.t. \( \phi \) and all symbolic heaps in \( \Phi \).
9 Proof Procedure

Algorithm 1 Entailment Checking Algorithm.

Require: A progressing, connected and established SID \( \mathcal{R} \), an oriented transition system \( \mathcal{S} \),
Require: a symbolic heap \( \phi \) and an LTL formula \( \Phi \)
\( \mathcal{A} \leftarrow \{ \phi, q_I, \neg \Phi \} \)
while \( \mathcal{A} \) admits an LTL interpretation \( \mathcal{I} \) do
    \( S^+ \leftarrow \{ \phi \in \mathcal{A}_S \mid \mathcal{I}(\phi, 0) = true, \phi \) is a symbolic heap \}\)
    \( S^- \leftarrow \{ \phi \in \mathcal{A}_S \mid \mathcal{I}(\phi, 0) = false, \phi \) is a symbolic heap \}\)
    \( X \leftarrow \{ x \in V^+ \mid \mathcal{I}(\phi, 0) \not\in \mathcal{A}(x) \} \)
if \( (S^+, S^-, X) \) is unsatisfiable \{This test is decidable by Lemma 9\} then
    \( \mathcal{A} \leftarrow \mathcal{A} \cup \Gamma(S^+, S^-, X) \)
else
    Let \( (s, h) \) be an \( \mathcal{R} \)-model of \( (S^+, S^-, X) \)
    if \( r_{r} \) is defined and \( (s, h) \models_{r} S \neg \Phi \) \{the test is decidable by Lemma 22\} then
        Return \( (s, h) \)
    else
        Let \( \Psi \) be a formula in \( \mathcal{A}(\mathcal{R}, \mathcal{S}, \phi) \) s.t. \( (s, h) \not\models_{r} \Phi \) \{\Psi exists by Theorem 39\}
        \( \mathcal{A} \leftarrow \mathcal{A} \cup \{ \Psi \} \)
    end if
end if
end while
Return \( \top \)

Even if \( \mathcal{S} \) is oriented, the set \( \mathcal{A}(\mathcal{R}, \mathcal{S}, \phi) \) is exponential w.r.t. the size of \( \mathcal{R} \), \( \phi \) and \( \mathcal{S} \), and only a small part of this set will be relevant, hence computing all axioms explicitly is not practical. Algorithm 1 computes these axioms on demand, in the spirit of the well-known DPLL(T) procedure (see, e.g., [9]) by calling external tools to solve LTL and SL satisfiability problems. The idea is to construct an LTL interpretation and to refine it incrementally by adding relevant axioms until we get either a model that is compatible with some SL structure, or a set of axioms that is unsatisfiable (in LTL). For all LTL interpretations \( \mathcal{I} \), \( r_{r} \) is the sequence \( (q_i, a_i)_{i \in \mathbb{N}} \) (if it exists) such that \( q_i \) is the unique state in \( Q \) (resp. the only action) with \( I(q_i, i) = true \) (resp. \( I(a_i, i) = true \).

Theorem 40. If Algorithm 1 returns \( \top \) then the entailment \( \phi \models_{\mathcal{R}} \Phi \) holds. If it returns an SL structure \( (s, h) \) then \( (s, h) \models_{\mathcal{R}} \phi \) and \( (s, h) \not\models_{\mathcal{R}} \Phi \). Moreover, if \( \mathcal{S} \) is oriented then the algorithm always terminates.

Proof. Termination is immediate (if \( \mathcal{S} \) is oriented) since at each new iteration a new formula from \( \mathcal{A}(\mathcal{R}, \mathcal{S}, \phi) \) is added in \( \mathcal{A} \) and the set \( \mathcal{A}(\mathcal{R}, \mathcal{S}, \phi) \) is finite (as \( \mathcal{B}(\mathcal{S}, \phi, \Phi) \) and \( \mathcal{B}(\mathcal{S}, \phi, \Phi) \) are both finite). If \( \top \) is returned then by definition of the algorithm \( \mathcal{A}(\mathcal{R}, \mathcal{S}, \phi) \cup \{ q_i, \phi, \neg \Phi \} \) is unsatisfiable thus the entailment \( \phi \models_{\mathcal{R}} \Phi \) is valid by Theorem 38. If the algorithm returns a structure \( (s, h) \) then by definition \( (s, h) \models_{\mathcal{R}} (q_i, a_i)_{i \in \mathbb{N}} \neg \Phi \) for some sequence \( (q_i, a_i)_{i \in \mathbb{N}} \), thus there is a run \( (q_i, s_i, h_i, a_i)_{i \in \mathbb{N}} \) and a compatible LTL interpretation \( \mathcal{I} \) such that \( \mathcal{I} \not\models \Phi \).  

A natural issue is to determine whether Algorithm 1 is complete for refutation (when $S$ is not oriented), i.e., whether it always returns a counter model if the entailment is not valid (by Theorem 23 it cannot be complete for validity). Another natural continuation is to extend the expressive power of the logic by considering more complex temporal connectives (to allow for quantification over paths). It would also be interesting to extend the language in order to handle more complex (possibly non deterministic) actions. For instance, it should be noticed that actions in our framework cannot create new locations (as evidenced by Proposition 13). This is important, because, otherwise, since universal quantification is not allowed, the corresponding pre/post-conditions could not be expressed in the language. This entails that C-like allocations for instance are not built-in: they must be performed by handling a stack of available locations, allocated in the symbolic heap describing the initial state of the system by an atom such as $\text{lseg}(x, y)$ (an instruction such as $\text{malloc}(z)$ can be simulated by two actions $z := x$ and $x := x$.1). The complexity of the entailment problem for oriented systems also deserves to be precisely identified (it is 2-EXPTIME hard by [4]).

References

6 Didier Galmiche and Daniel Méry. Labelled tableaux for linear time bunched implication logic. In ASL 2022 (Workshop on Advancing Separation Logic), 2022.
A Proof of Lemma 9

We use the algorithm developed in [5] to test the validity of entailments between SL formulas (if the considered SID is progressing, connected and established), combined with the technique devised in [12] to cope with conjunctions (see also [11]). Let \( X = \{x_1, \ldots, x_n\} \), where the order on the \( x_i \) is arbitrary. For every \( i = 1, \ldots, n \), we denote by \( \Psi_i \) the formula: 
\[
\bigvee_{j=1}^{i-1} x_i \simeq x_j \lor x_i \mapsto (x_1, \ldots, x_i).
\]
Let \( \Psi = \Psi_1 \cdots \Psi_n \). By definition, if \( (s, h') \models_R \Psi \) then 
\[
dom(h') = \{ s(x) \mid x \in X \},
\]
hence, for every structure \((s, h)\), there is at most one heap \( h' \subseteq h \) with \((s, h') \models_R \Psi \). Moreover, for all stores \( s \), we have \((s, h_2) \models_R \Psi \), where \( h_2 \) denotes the heap: \( \{ (\ell, \ldots, \ell) \mid \exists x \in X \ s.t. \ell = s(x) \} \). Let \( \Phi = \bigwedge_{\phi \in S^+}(\Psi \ast \phi) \land \neg (\bigvee_{\phi \in S^-}(\Psi \ast \phi)) \). Note that since \( S^+ \) is not empty, \( \Phi \) is a guarded formula (as defined in [12, Fig. 1]), except that it contains existential quantifiers (the fact that \( S^+ \) is non empty is essential, as otherwise the negation would not be guarded). The satisfiability of \( \Phi \) can be tested by combining the techniques devised in [12] and [5]. The idea is to compute an abstraction of the possible models of \( \Phi \) bottom-up. Points-to atoms, inductive predicates, separating conjunctions and existential quantifications can be handled as explained in [5], whereas conjunctions and guarded negations are handled as it is done in [12]. We prove that \((S^+, S^-, X)\) is satisfiable iff \( \Phi \) is satisfiable:

- Assume that \((s, h) \models_R (S^+, S^-, X)\). Then \((s, h) \models_R \phi\) for all \( \phi \in S^+ \), \((s, h) \not\models_R \phi\) for all \( \phi \in S^- \), and \( s(x) \not\in \dom(h) \) for all \( x \in X \). Then \( h_2 \) and \( h \) are disjoint, thus we get \((s, h) \cup h_2) \models_R \phi \ast \Psi\) for all \( \phi \in S^+ \). If \((s, h) \cup h_2) \models_R \phi \ast \Psi\) for some \( \phi \in S^- \) then since \( h_2 \) is the unique heap such that \((s, h_2) \models_R \Psi\), we deduce that we must have \((s, h) \models_R \phi\), which contradicts our assumption. Thus \((s, h) \cup h_2) \models_R \Phi\).

- Assume that \((s, h) \models_R \Phi\). Then, we get \((s, h) \models_R \phi \ast \Psi\), for all \( \phi \in S^+ \). Since \( h_2 \) is the unique heap such that \((s, h_2) \models_R \Psi\), this entails that \((s, h') \models_R \phi\), with \( h' = h \setminus h_2 \). Since \( \dom(h_2) = \{ s(x) \mid x \in X \} \) we get \( s(x) \not\in \dom(h') \), for all \( x \in X \). If \((s, h') \models_R \phi\), for some \( \phi \in S^- \) then we deduce \((s, h' \cup h_2) \models_R \phi \ast \Psi\), which contradicts our hypothesis. Thus \((s, h') \models_R (S^+, S^-, X)\).

B Proof of Lemma 36

Assume that \((s, h) \models_R \Psi\). We show, by induction on \(|h|\), that \((s, h) \models_R \phi\). If \( x \in \nu_a(\phi) \) then \( \langle \phi \rangle_x = \{ \phi \} \) thus \( \phi = \psi \) and the proof is immediate. Otherwise, we distinguish the following cases, following Definition 34:

- \( \phi' = \exists u. (x \simeq x' \ast x \mapsto (y) \ast \psi) \) and \( \phi = \exists u. (x' \mapsto (y) \ast \psi) \). It is clear that \( \phi' \models_R \phi \).
- \( \phi' = \exists u \exists v. (x \simeq x' \ast y' \ast \psi) \), and \( \phi = \exists u. (p(x', y) \ast \psi) \) with \( p(x, y) \iff \exists v. \psi' \). In this case, we get \( \phi' \models_R \exists u. (x \simeq x' \ast p(x, y) \ast \psi) \), thus \( \phi' \models_R \exists u. (p(x', y) \ast \psi) = \phi \).
- \( \phi' = \exists u \exists z_1 \ldots \exists z_m. (q \rightarrow p)(y, z_1, \ldots, z_m) \ast z_1 \simeq x \ast \psi' \ast \psi \) and \( \phi = \exists u. (p(y) \ast \psi) \), with \( q(x, z_1, \ldots, z_m) \iff \exists v. \psi \). Then we get \( \phi' \models_R \exists u \exists z_1 \ldots \exists z_m. ((q \rightarrow p)(y, z_1, \ldots, z_m) \ast z_1 \simeq x \ast \psi' \ast \psi) \) and by definition of the rules associated with the predicate \((q \rightarrow p)\), one of the following conditions holds (with \( y = (y_1, \ldots, y_n) \)):
  - \( \phi' \models_R \exists u \exists z_1 \ldots \exists z_m. \exists u. (y_1 \mapsto (y') \ast (q \rightarrow p)(x', z_1, \ldots, z_m) \ast \psi') \) with \( p(y) \iff y_1 \mapsto (y') \ast p'(x') \ast \psi'' \). This entails that there exists a store \( s' \) coinciding with \( s \) with all the variables not occurring in \( u, z_1, \ldots, z_m, w, v \) and disjoint
heaps $h', h''$ such that $h = h' \sqcup h''$, $(s, h') \models_R y_1 \mapsto (y') + \psi'$ and $(s, h'') \models_R (q \rightarrow p')(z') z_i \models z \equiv x * \psi' * \psi$. This entails that $h' \neq \emptyset$ thus $|h'| > |h''|$. By definition, $(q \rightarrow p')(z', z_i \ldots , z_m) \models z \equiv x * \psi' * \psi \in (y'(z') + \psi')_x$, hence by the induction hypothesis we get $(s, h') \models_R y_1 \mapsto (y') + \psi' + p'(z') + \psi'$, hence $(s, h) \models_R \exists u. (p(y) + \psi) = \phi$.

$$\phi' \models_R \exists x_1 \ldots \exists x_m \exists u \exists w. (y_1 \mapsto (y') + \psi' \simeq (z_1 \ldots , z_m) * \psi' + z_1 \equiv x * \psi' * \psi) \text{ with } (q(x, z_2 \ldots , z_m) \models_R \exists u. \psi') \text{, we deduce that }$$

$$p'(y') \models_R \exists x_1 \ldots \exists x_m \exists u \exists w. (y_1 \mapsto (y') + \psi' \simeq (z_1 \ldots , z_m) * \psi' + z_1 \equiv x * q(x, z_2 \ldots , z_m) * \psi)$$

Thus $\phi' \models_R \exists u \exists w. (y_1 \mapsto (y') + \psi' + q(z') + \psi)$, hence $\phi' \models_R \exists u. (p(y) + \psi) = \phi$.

### Proof of Lemma 37

Assume that $(s, h) \models_R \phi$ and that $s(x) \in dom(h)$. We show, by induction on $|h|$, that $(s, h) \models_R \psi$, for some $\psi \in \langle \phi \rangle x$. The symbolic heap $h$ is necessarily of the form $\exists u. (\phi_1 \ast \ldots \ast \phi_k)$, where the $\phi_1, \ldots, \phi_k$ are atoms. We assume by $\alpha$-renaming that $x$ does not occur in $u$. By definition of the semantics of SL, there exists a store $s'$ (coinciding with $s$ on all variables not occurring in $u$) and disjoint heaps $h_i$ such that $(s', h_i) \models_R \phi_i$ (for all $i = 1, \ldots, n$) and $h = h_1 \uplus \ldots \uplus h_k$. Since $s(x) \in dom(h_i)$, necessarily $s(x) \in dom(h_i)$ for some $i = 1, \ldots, k$, say $i = 1$. Let $\phi' = \phi_2 \ast \ldots \ast \phi_k$. We distinguish several cases.

- Assume that $\phi_1$ is a points-to atom $x' \rightarrow (y)$. If $x = x'$, then $x \in \nu_x(\phi)$, so that $\langle \phi \rangle x = \{ \phi \}$, hence the proof is immediate. Otherwise, since (by definition of the semantics of SL) $\text{dom}(h_1) = \{ s'(x') \}$ and $s(x) \in h_1$ we must have $s(x') = s(x) = s'(x')$. By Definition 34 (2a), $\langle \phi \rangle x$ contains a formula of the form $\exists u. \psi' \equiv x \equiv x' * \psi'$. We have $(s', h) \models_R x \mapsto \langle \psi \equiv x \equiv x' \ast \phi' \rangle$, so that $(s, h) \models_R \exists u. \psi' \equiv x \equiv x' \ast \phi'$. Thus $(s, h) \models_R \exists u \exists v. \psi' \equiv x \equiv x' \ast \phi'$, and by Def. 34 (2b), this formula is in $\langle \phi \rangle x$.

- Finally, assume that $\phi_1$ is a predicate atom $p(x', y)$ and that $s'(y_1) \neq s(x)$. Necessarily, $p'(x', y) \models_R \exists u. \psi'$ with $(s', h_1) \models_R x' \mapsto (y') + \psi'$, for some store $s'$ coinciding with $s'$ on all variables not occurring in $u$. Since $s'(y_1) \neq s(x)$ and $s(x) \in \text{dom}(h_1)$, $\psi$ must be of the form $p'(z') + \psi'$ (with possibly $\psi' = \text{emp}$) and there exist disjoint heaps $h', h''$ such that $h_1 = h' \sqcup h''$, $(s', h') \models_R x' \mapsto (y') + \psi'$ and $(s', h'') \models_R p'(z)$. This entails that $h' \neq \emptyset$, thus $|h'| > |h''|$, and by the induction hypothesis, we deduce that $(p'(z))_x$ contains a formula $\psi''$ such that $(s', h'') \models_R \psi'$. We get $(s, h) \models_R \exists u \exists v. \psi' \equiv x \equiv x' \ast \phi'$. By Definition 34, $\psi''$ is of one of the following forms:

**2b** $\psi'' = \exists u. (\xi \equiv z_1 \equiv x)$, with $z = (z_1, \ldots , z_m)$ and $p'(x, z_2 \ldots , z_m) \models_R \exists u. \xi$. Then we get $(s, h) \models_R \exists u. \exists v. (x' \mapsto (y') + \xi + z_1 \equiv x * \psi' + \psi')$. By definition of the rules defining $\langle p \rightarrow p \rangle$, we have: $\langle p \rightarrow p \rangle \equiv \langle p \rightarrow p \rangle(x, y, z)$, so that $(s, h) \models_R \exists u. \exists v. (x' \mapsto (y') + \psi' + \xi \equiv z_1 \equiv x \ast \phi')$, hence $(s, h) \models_R \exists u. \exists v. (x' \mapsto (y') + \psi' + \xi \equiv z_1 \equiv x \ast \phi') \exists u. \psi' \equiv z_1 \equiv x \ast \phi'$, where $z' = \{ z_1, \ldots , z_m \}$ is a vector of fresh pairwise distinct variables. By Definition 34 (2c), the latter formula occurs in $\langle \phi \rangle x$.

**2c** $\psi'' = \exists z_1 \ldots , z_m. (q \rightarrow p')(z, z_2 \ldots , z_m) + z_1 \equiv x * \xi$, with $q(x, z_2 \ldots , z_m) \models_R \xi$. We get $(s, h) \models_R \exists u. \exists v. (z_1 \ldots , z_m \mapsto (y') + (q \rightarrow p')(z, z_2 \ldots , z_m) + z_1 \equiv x \ast \xi' + \phi')$. By definition of the rules defining $\langle q \rightarrow p \rangle$, we have $(q \rightarrow p \rangle(x, y, z_2 \ldots , z_m) \models_R \exists u. \exists v. (x' \mapsto (y') + (q \rightarrow p')(z, z_2 \ldots , z_m) + \psi')$, hence $(s, h) \models_R \exists u. \exists v. (z_1 \ldots , z_m \mapsto (q \rightarrow p')(z, z_2 \ldots , z_m) + \psi')$. By Definition 34 (2c), this formula is in $\langle \phi \rangle x$. 

---

This is a part of the proof of a theorem or lemma in a paper on dynamic transformations of symbolic heaps. The notation and symbols are specific to the field of symbolic execution and heap analysis in computer science. The proof involves reasoning about symbolic heaps, using existential quantifiers and implication operators, and applies induction on the size of the heap to establish the desired property.
\section*{D Finiteness of $\mathcal{B}_\omega(\mathcal{S}, \phi, \Phi)$ and $\mathcal{F}_\omega(\mathcal{S}, \phi)$}

We write $\phi \rightarrow_s \psi$ if $\psi$ is obtained from $\phi$ by using one of the above simplification rules.

\begin{align*}
C_x : \exists u.(x \neq x \xi) & \rightarrow \perp \\
E_x : \exists u.\forall x.(x \neq x_1 \ldots \neq x_n \neq x) & \rightarrow \exists u.\xi \text{ if } x \neq \nu(\xi) \cup \{x_1, \ldots, x_n\}
\end{align*}

\begin{align*}
C_s : \exists u.(x \mapsto (y) \neq (z) \mapsto (z) \xi) & \rightarrow \perp \\
E_s : \exists u.\exists x.(x \equiv y \neq \xi) & \rightarrow \exists u.\xi \{x \leftarrow y\}
\end{align*}

It is easy to verify that $\rightarrow_s$ is well-founded, and that $\phi \rightarrow_s \psi \implies \phi \equiv_R \psi$.

\begin{lemma}
If $\mathcal{S}$ is oriented then the sets $\mathcal{B}_\omega(\mathcal{S}, \phi, \Phi)$ and $\mathcal{F}_\omega(\mathcal{S}, \phi)$ are finite (up to associativity and commutativity of $*$, $\alpha$-renaming and equivalence w.r.t. $\rightarrow_s$).
\end{lemma}

\begin{proof}
We assume that all symbolic heaps are in normal form w.r.t. $\rightarrow_s$. Let $\mathcal{S} = (Q, R, q_0)$. We give the proof for $\mathcal{B}_\omega(\mathcal{S}, \phi, \Phi)$, the set $\mathcal{F}_\omega(\mathcal{S}, \phi)$ can be handled in a similar way (the only difference is that one must consider the order $\preceq_\mathcal{S}$ instead of $\succeq_\mathcal{S}$, and that $\mathcal{F}_\omega(\mathcal{S}, \phi)$ does not depend on $\mathcal{B}_\omega(\mathcal{S}, \phi, \Phi)$, while $\mathcal{B}_\omega(\mathcal{S}, \phi, \Phi)$ depends on $\mathcal{F}_\omega(\mathcal{S}, \phi)$). If $\mathcal{B}_\omega(\mathcal{S}, \phi, \Phi)$ is infinite then by definition (assuming that $\mathcal{F}_\omega(\mathcal{S}, \phi)$ is finite) by König’s lemma there must exist an infinite sequence of pairwise distinct triples $(\phi_i, q_i, X_i)$ $(i \in \mathbb{N})$ such that $X_0 = \emptyset$ and for every $i \in \mathbb{N}$, one of the following conditions holds:

- There exists an action $a_i$ such that $(q_{i+1}, a_i, q_i) \in R$ and $\phi_{i+1} = wpc(\phi_i, a_i)$, where $X_{i+1} = X_i \setminus \{x_i\}$ if $a_i$ is of the form $x_i := t_i$ with $x_i \in \mathcal{V}^*$, and $X_{i+1} = \emptyset$ otherwise, or;
- $\phi_{i+1} = \psi_i$ with $\psi_i \in \psi(\phi_i)_{x_i}$ for some variable $x \in \mathcal{V}^* \setminus X_i$, $q_{i+1} = q_i$, and $X_{i+1} = X_i \cup \{x_i\}$.

In both cases we have $q_{i+1} \succeq_\mathcal{S} q_i$, by definition of $\preceq_\mathcal{S}$ (see Definition 14). Since the set of states $Q$ is finite, necessarily there exists a natural number $k$ such that, $q_{i+1} \neq q_i$ holds for all $i \geq k$. Since by hypothesis $\mathcal{S}$ is oriented, this entails that $R$ contains no transition of the form $(q_{i+1}, x_i := t_i, q_i)$ with $x_i \in \mathcal{V}^*$ and $i \geq k$. Consequently, we must have $X_{i+1} \succeq X_i$, for all $i \geq k$. By definition of $\mathcal{B}_\omega(\mathcal{S}, \phi, \Phi)$, $X_i \subseteq \mathcal{V}^*$ for all $i \in \mathbb{N}$, and since $\mathcal{V}^*$ is finite we deduce that there exists $l \in \mathbb{N}$ such that $l \geq k$ and $X_i = X_{i+l}$ for all $i \geq l$. By definition of $\mathcal{B}_\omega(\mathcal{S}, \phi, \Phi)$, this entails that $\phi_{i+l}$ must be of form $wpc(\phi_i, a_i)$, for all $i \geq l$, and $a_i$ is not of the form $x_i := t_i$. Note that this implies that all the predicates symbols occurring in $\phi_i$ occur in $\phi_l$ (since all the predicates in $wpc(\phi_i, a_i)$ must occur in $\phi_l$). For all $i \geq l$, we denote by $n_i$ the number of atoms in $\phi_l$ that are not equal to $\phi_l$ and not of the form $x \mapsto (y)$ with $x \in \mathcal{V}^*$. By inspection of the different cases in Definition 30 (taking into account the fact that $a_i$ is not of the form $x_i := t_i$), it is easy to check that $n_{i+1} = n_i$ holds for all $i \geq l$. Indeed, the only case in which $wpc(\phi_i, a_i)$ contains an atom that does not occur in $\phi_i$ is when this atom is either an equation or a points-to atom with a left-hand side in $\mathcal{V}^*$ (furthermore, the simplification rules in $\rightarrow_s$ cannot add new atoms in the formula). By irreducibility w.r.t. the rule $C_\mathcal{S}$, this entails that the number of spatial atoms in $\phi_i$ (for $i \geq l$) is at most $\text{card}(\mathcal{V}^*) + n_i$. Assume that $\phi_l$ contains a variable $x$ that does not occur in a spatial atom. By irreducibility w.r.t. the rule $E_\mathcal{S}$, $x$ cannot occur in an equation. By irreducibility w.r.t. $C_\mathcal{S}$, it cannot occur in a disequation $x \neq x$. Thus the only atoms in which $x$ occurs are of the form $x \neq x$, with $x_i \neq x$, and the rule $E_\mathcal{S}$ applies, which contradicts the fact that $\phi_l$ is in normal form w.r.t. $\rightarrow_s$. Consequently, all the existential variables in $\phi_l$ occur in a spatial atom. Since the number of such atoms is bounded, necessarily the number of existential variables is bounded. As both the set of free variables $\mathcal{V}^*$ and the set of predicate symbols in $\phi_l$ is finite, this entails that there exist finitely many symbolic heaps $\phi_i$ (with $i \geq l$), which contradicts our assumption.
\end{proof}
We construct a run \((q_1, s_1, h_1, a_1)_{i \in \mathbb{N}}\), and a corresponding sequence of triples \((\phi_i', q_i, X_i)_{i \in \mathbb{N}}\) by induction on \(i\), with \((s_0, h_0) \models \mathcal{R} \phi\). We simultaneously establish the following inductive invariant:

a) The equivalence \(\mathcal{I}(\xi, i) = true \iff (s_i, h_i) \models \mathcal{R} \xi\) holds for all atomic conditions \(\xi\), and also for all symbolic heaps \(\xi\) such that there exists \(q \in Q\) and \(X \subseteq \mathcal{V}\) with \((\xi, q, X) \in \mathcal{Bw}(\mathcal{S}, \phi, \Phi)\), for all \(x \in X \setminus u_\mathcal{R}(\xi)\).

b) For all \(q \in Q\) and for all actions \(a\), \(\mathcal{I}(q, i) = true\) iff \(q = q_i\) and \(\mathcal{I}(a, i) = true\) iff \(a = a_i\).

c) \((\phi_i', q_i, X_i) \in \mathcal{Fw}(\mathcal{S}, \phi)\) with \(\mathcal{I}(\phi_i', i) = true\) and \(\forall x \in X_i \setminus u_\mathcal{R}(\phi_i') : s_i(x) \notin \text{dom}(h) \land (s_i, h_i) \not\models \mathcal{A}(x)\).

Note that the invariant entails in particular that \(\mathcal{I}\) is compatible with \((q_i, s_i, h_i, a_i)_{i \in \mathbb{N}}\), w.r.t. all symbolic heaps occurring in \(\Phi\). Indeed, by definition of \(\mathcal{Bw}(\mathcal{S}, \phi, \Phi)\), \((\psi, q, \emptyset) \in \mathcal{Bw}(\mathcal{S}, \phi, \Phi)\) for all symbolic heaps occurring in \(\Phi\) and for all states \(q\).

**Base case** \((i = 0)\). Let \(q_0 \overset{def}{=} q_1\), \(\phi_0 \overset{def}{=} \phi\) and \(X_0 \overset{def}{=} \emptyset\). Let \(S^+\) (resp. \(S^-\)) be the set of symbolic heaps \(\psi\) occurring in \(\mathcal{A}(\mathcal{R}, \mathcal{S}, \phi)\), \(\{\phi\}\) or \(\Phi\) such that \(\mathcal{I}(\psi, 0) = true\) (resp. \(\mathcal{I}(\psi, 0) = false\)). By hypothesis, \(\Phi \in S^+\), thus \(S^+ \neq \emptyset\). Let \(X\) be the set of variables \(\psi\) such that \(\mathcal{I}(\psi, 0) = false\). By definition, \(\mathcal{I} \not\models \Gamma(S^+, S^-, X)\), thus, by Axiom 14, \((S^+, S^-, X)\) cannot be unsatisifiable, and there exists a structure \((s_0, h_0)\) such that \((s_0, h_0) \models \mathcal{R}(S^+, S^-, X)\). By construction, \(\mathcal{I}(\xi, 0) = true \iff (s_0, h_0) \models \mathcal{R} \xi\) holds for all symbolic heaps \(\xi\) occurring in \(\mathcal{A}(\mathcal{R}, \mathcal{S}, \phi)\), \(\{\phi\}\) or \(\Phi\), and in particular, \((s_0, h_0) \models \mathcal{R} \phi\). Still by construction, \(\mathcal{I}(\psi, 0) = false \iff (s_0, h_0) \not\models \mathcal{A}(x)\). Conversely, if \(\mathcal{I}(\psi, 0) = true\), then by Axiom 4, necessarily \((\emptyset, 0) \models \xi\), for some \(\xi \in \langle \phi \rangle\), thus \((s_0, h_0) \models \mathcal{R} \xi\) and by Lemma 35, we get \((s_0, h_0) \models \mathcal{R} \mathcal{A}(x)\). Thus Property a holds for all symbolic heaps and for all conditions of the form \(\mathcal{A}(x)\).

By hypothesis we have \((\emptyset, 0) \models q_i\), and, by Axiom 11, \((\emptyset, 0) \models \neg q_i\) for all states \(q_i \neq q_i\). By Axioms 12 and 11, there exists a unique action \(a_0\) such that \((\emptyset, 0) \models a_0\). Thus Property b holds.

By definition of \(\mathcal{Fw}(\mathcal{S}, \phi)\) we have \((\psi, q, \emptyset) \in \mathcal{Fw}(\mathcal{S}, \phi)\) thus Property c holds.

It only remains to prove that Property a holds for all atomic conditions (other than those of the form \(\mathcal{A}(x)\)). Consider any atomic condition \(\alpha\), and assume that \(\mathcal{I}(\alpha, 0) = true\) (the case whether \(\mathcal{I}(\alpha, 0) = false\) is handled in a similar way). We show that \((s_0, h_0) \models \alpha\). Assume that \(\alpha\) is of the form \(x \approx y\), with \(x, y \in \mathcal{V}\). By definition \(\phi_0\) is of the form \(\exists u. \phi'\) for some symbolic heap \(\phi'\) containing no quantifier. By definition of \(\mathcal{Bw}(\mathcal{S}, \phi, \Phi)\) we have \((\exists u. \phi' \star x \approx y), q_0, X_0) \in \mathcal{Bw}(\mathcal{S}, \phi, \Phi)\). By Axiom 7, since \(\mathcal{I}(\phi_0', 0) = true\), we get \(\mathcal{I}(\exists u. \phi' \star x \approx y), 0) = true\) thus \((s_0, h_0) \models \mathcal{R} \exists u. (\phi' \star x \approx y)\) (by Property a, which has already been established for symbolic heaps). Thus \(s_0(x) = s_0(y)\) and therefore \((s_0, h_0) \models \mathcal{R} x \approx y\). Assume that \(\alpha\) is of the form \(x.i \approx y\), with \(x, y \in \mathcal{V}\). By Axiom 1 we must have \(\mathcal{I}(\mathcal{A}(x), 0) = true\), hence by Axiom 4 we deduce that \(\mathcal{I}(\psi, 0) = true\), for some \(\psi \in \langle \phi_0' \rangle x\) (note that, by definition of \(\mathcal{Fw}(\mathcal{S}, \phi)\), we have \((\psi, q_0, X_0 \cup \{x\}) \in \mathcal{Fw}(\mathcal{S}, \phi)\)). By Lemma 35, \(\psi\) is of the form \(\exists u. (x \mapsto (x_1, \ldots, x_k) \star \psi')\). We have \((\exists u. (x \mapsto (x_1, \ldots, x_k) \star \psi') \star x \approx y), q_0, X_0 \cup \{x\}) \in \mathcal{Bw}(\mathcal{S}, \phi, \Phi)\), thus by Axiom 5 we deduce that \(\mathcal{I}(\exists u. (x \mapsto (x_1, \ldots, x_k) \star \psi') \star x \approx y), 0) = true\), so that \((s_0, h_0) \models \mathcal{R} x.i \approx y\). The proof is similar if \(\alpha\) is of the form \(x.i \approx y.j\) (using Axiom 6).

**Inductive case.** Assume that \((q_i, s_i, h_i, a_i)\) has been constructed and that the invariant above holds for all \(i \leq k\). As \(\mathcal{I}(a_k, k) = true\), by Axiom 13, we have \((\mathcal{I}, k) \models \mathcal{P}(a_k)\), hence \((s_k, h_k) \models \mathcal{R} \mathcal{P}(a_k)\) (by Property a). By Proposition 26, we deduce that \((s_k, h_k)[a_k] = \mathcal{R} \mathcal{P}(a_k)\).
is defined. Let \((s_{k+1}, h_{k+1}) = (s_k, h_k)[a_k]\). By Axioms 12 and 11, there exist a unique action \(a_{k+1}\) and state \(q_{k+1}\) such that \(T(a_{k+1}, k+1) = T(q_{k+1}, k+1) = \text{true}\), with \((q_k, q_{k+1}, X_{k+1}) \in R\).

We show that Property \(a\) is satisfied for \(k+1\). We first observe that, if \(a_k\) is not of the form free\((x)\) resp. alloc\((x)\) then we have by Axiom 9 (resp. Axiom 10), \(T(\lambda(y), k+1) = false\) (resp. \(T(\lambda(y), k+1) = true\)) if \(\lambda(x) \equiv y, k\) true and \(T(\lambda(y), k+1) = \lambda(y), k\) otherwise.

Furthermore, by Axiom 8, \(T(\lambda(y), k+1) = T(\lambda(y), k)\) holds for all \(y \in V^+\) if \(a_k\) is not of the above forms.

Consider a triple \((\psi, q, X) \in Bw(S, \phi, \Phi)\) such that for all \(x \in X \setminus v_+ (\psi)\), \(s_{k+1}(x) \not\in \text{dom}(h_{k+1})\) \(\{\}\). If \(a_k\) contains a term \(x_i\) where \(x \not\in v_+(\psi)\), then (by definition of \((s_k, h_k)[a_k]\)) \(s_{k+1}(x) \in \text{dom}(h_{k+1})\), so that \(x \not\in X\). Thus, by definition of \(Bw(S, \phi, \Phi)\), \((\xi, q, X \cup \{x\}) \in Bw(S, \phi, \Phi)\), for all \(\xi \in \psi_x\). Note that (since actions of the form \(x := x_i\) are forbidden) we must have \(T(\lambda(x), k+1) \iff s_{k+1}(x) \in \text{dom}(h_{k+1})\), hence \(T(\lambda(x), k+1) = true\) and by Axiom 4, necessarily \(T(k+1) \iff \psi = \bigvee_{\xi \in \psi_x} \xi\). By Lemma 35, \(x \in v_+(\xi)_x\) for all \(\xi \in \psi_x\). By repeating this process (if needed) on any other variable \(y\) such that the condition above holds (in case \(a_k\) contains another occurrence of a term \(y,j\)), we eventually obtain a set of symbolic heaps \(S\) such that \(T(k+1) \iff \psi = \bigvee_{\xi \in S} \xi\), for all \(\xi \in S, wpc(\xi, a_k)\) is defined, and there exists \(X'\) such that \((\xi, q, X') \in Bw(S, \phi, \Phi)\), with \(X' = X \cup Y\), for some set of variables \(Y \subseteq v_+(\xi)\). This entails (by definition of \(Bw(S, \phi, \Phi)\)) that, for all \(\xi \in S\), \((wpc(\xi, a_k), q_{k+1}, X''') \in Bw(S, \phi, \Phi)\), for some \(X''\) that is either empty (if \(a_k\) is of the form \(x := t\) with \(t \in V^+\)) or identical to \(X'\) (otherwise). Furthermore, we have \((s_{k+1}, h_{k+1}) \models_{\text{c}} \psi \iff \exists \xi \in S \; \text{s.t.} \; (s_{k+1}, h_{k+1}) \models_{\text{c}} \xi\), by Lemmata 36 and 37. By Property \(a\) in the inductive invariant (at rank \(k\)) the equivalence \(T(\lambda(\xi), a_k), k) = true \iff (s_{k+1}, h_{k}) \models_{\text{c}} \psi \iff \psi = \bigvee_{\xi \in S} \xi\), for all \(\xi \in S\), \(wpc(\xi, a_k)\) is defined, and there exists \(X'\) such that \((\xi, q, X') \in Bw(S, \phi, \Phi)\), with \(X' = X \cup Y\), for some set of variables \(Y \subseteq v_+(\xi)\). This entails (by definition of \(Bw(S, \phi, \Phi)\)) that, for all \(\xi \in S\), \((wpc(\xi, a_k), q_{k+1}, X''') \in Bw(S, \phi, \Phi)\), for some \(X''\) that is either empty (if \(a_k\) is of the form \(x := t\) with \(t \in V^+\)) or identical to \(X'\) (otherwise).

We now show that \(Fw(S, \phi)\) contains a tuple \((\phi'_{k+1}, q_{k+1}, X_{k+1})\) such that \(T(\phi'_{k+1}, k+1) = true\). Let \(Y\) be the set of variables \(y\) such that \(a_k\) contains a term of the form \(\psi_i\) (for some \(i \in \mathbb{N}\)) and \(y \not\in v_+(\phi_i)\). By applying the function \(\langle \_ \rangle_x\) on all variables in \(Y\), we get a set of symbolic heaps such that \((T, k) \models \phi' \iff \bigvee_{\xi \in S} \xi\). Furthermore, for all variables \(y \in Y\), we have \(s(y) \in \text{dom}(h_k)\) (since \((s_k, h_k)[a_k]\) is defined), thus \(y \not\in X_k\). By definition of \(Fw(S, \phi)\), we deduce that for all \(\xi \in S\), \((\xi, q_k, X_k \cup Y) \in Fw(S, \phi)\). Moreover, by Lemma 35, we have \(Y \subseteq v_+(\xi)\), and \(spc(\xi, a_k)\) is defined for all \(\xi \in S\). Then we get by Axiom 2, \((T, k+1) = \text{spec}(\xi, a_k)\), for some \(\xi \in S\). We define: \(\phi'_{k+1} \overset{\text{def}}{=} \text{spec}(\xi, a_k)\). By definition of \(Fw(S, \phi)\), \((spc(\xi, a_k), q_{k+1}, X_{k+1}) \in Fw(S, \phi)\) for some set \(X_{k+1}\). Let \(x \in x \in X_{k+1} \setminus v_+(\phi'_{k+1})\). Assume that \(s_{k+1}(x) \in \text{dom}(h_k)\). By definition of \(Fw(S, \phi)\), \(a_k\) cannot be of the form \(z := t\), where \(z \in V^+\) (otherwise \(X_{k+1}\) would be empty). Thus \(X_{k+1} = X_k \cup Y\). Since \(x \not\in \text{dom}(a_{k+1})\), we have \(a_k \neq \text{free}(x)\). Since \(x \not\in v_+(\phi'_{k+1})\), we have \(a_k \neq \text{alloc}(x)\), by definition of \(spc(\xi, a_k)\). Thus \(a_k\) is of the form \(t := s\) where \(t\) is not a variable, and we must have \(x \in \text{dom}(h_k)\), and \(v_+(\phi'_{k+1}) = v_+(\xi) \supseteq Y\). This entails that \(x \in X_k\), which contradicts Property \(c\) at rank \(k\).
Finally, using the symbolic heap $\phi'_{k+1}$, the equivalence $\mathcal{I}(\alpha, k + 1) = true \iff (s_{k+1}, h_{k+1}) \models_{\mathcal{R}} \alpha$ can be established for all atomic conditions $\alpha$ exactly as for the base case. The case where $\alpha = \text{\texttt{A}}(x)$ and $\alpha_k$ is of the form $x := t$ is handled by noting that we have both $(s_{k+1}, h_{k+1}) \models \text{\texttt{A}}(x) \iff \bigvee \xi \in \langle \phi'_{k+1} \rangle \xi$ (since $(s_{k+1}, h_{k+1}) \models \phi'_{k+1}$, using Lemmata 35, 36 and 37) and $(\mathcal{I}, k + 1) \models \text{\texttt{A}}(x) \iff \bigvee \xi \in \langle \phi'_{k+1} \rangle \xi$ (by Axiom 4).