Entailment Checking in Separation Logic with Inductive Definitions is 2-EXPTIME hard
Mnacho Echenim, Radu Iosif, Nicolas Peltier

To cite this version:
Mnacho Echenim, Radu Iosif, Nicolas Peltier. Entailment Checking in Separation Logic with Inductive Definitions is 2-EXPTIME hard. LPAR 2020, 2020, Alicante, Spain. 10.1145/3380809 . hal-02990396

HAL Id: hal-02990396
https://hal.science/hal-02990396
Submitted on 5 Nov 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Entailment Checking in Separation Logic with Inductive Definitions is 2-EXPTIME-hard

Mnacho Echenim$^1$, Radu Iosif$^2$ and Nicolas Peltier$^1$

1 Univ. Grenoble Alpes, CNRS, LIG, F-38000 Grenoble France
2 Univ. Grenoble Alpes, CNRS, VERIMAG, F-38000 Grenoble France

Abstract

The entailment between separation logic formulæ with inductive predicates, also known as symbolic heaps, has been shown to be decidable for a large class of inductive definitions [8]. Recently, a 2-EXPTIME algorithm was proposed [11, 15] and an EXPTIME-hard bound was established in [9]; however no precise lower bound is known. In this paper, we show that deciding entailment between predicate atoms is 2-EXPTIME-hard. The proof is based on a reduction from the membership problem for exponential-space bounded alternating Turing machines [5].

1 Introduction

Separation logic is a particular case of the logic of bunched implications [12]. It was introduced in [14] as an extension of Hoare logic intended to facilitate reasoning on mutable data-structures, and it now forms the basis of highly successful static analyzers such as, e.g., Infer [4], SLAyer [2] or Predator [6]. The assertions in this logic describe heaps, that are finite partial functions mapping locations to tuples of locations (records), intended to model dynamically allocated objects. The usual connectives of propositional logic are enriched with a special connective, called the separating conjunction, that permits to assert that two formulæ hold on disjoint parts of the heap, allowing for more concise and more natural specifications. In this paper, we consider the fragment of separation logic formulæ known as symbolic heaps, consisting of separated conjunctions of atoms. Such atoms may be equational atoms, asserting equalities or disequalities between memory locations; points-to atoms asserting that some location refers to a given record; or may be built on additional predicates that assert that a part of the memory has some specific shape (such as a tree). For genericity, such predicates are associated with user-provided inductive definitions that allow one to describe custom data-structures. For example, the formula $x \mapsto (y,z) * p(y)$ states that the heap is composed of two disjoint parts: a first location $x$ pointing to a tuple of locations $(y,z)$ and a second part described by $p(y)$. Given the inductive definition:

$$p(x) \iff x \mapsto (\text{nil}, \text{nil}) \\
p(x) \iff \exists y_1, y_2 . x \mapsto (y_1, y_2) * p(y_1) * p(y_2)$$

$p(y)$ states that the considered part of the heap is a binary tree\(^1\) the rooted at $y$.

This logic provides a very convenient way to describe graph-like data-structures. Satisfiability is EXPTIME-complete for such formulæ [3], but entailment is not decidable in general\(^2\) [9, 1]. However, the entailment problem was proven to be decidable for a large class of inductive definitions, with syntactical restrictions that ensure the generated heap structures have a bounded-tree width [8], using a reduction to monadic second-order logic interpreted over graphs. An EXPTIME-hard bound was established in [9], and very recently, a 2-EXPTIME algorithm has been proposed. Although the algorithm in [11] (implemented in the system Harrsh) is practically successful, as evidenced by the experimental results reported in [11] and at https://github.com/katelaan/harrsh, it was discovered in [15]

\(^1\)For conciseness we omit the rules for the two cases where one of the children is nil but the other one is not.

\(^2\)Entailment does not reduce to satisfiability since the considered logic has no negation.
that it was incomplete, and some techniques are proposed to fix this issue (a complete description of the new algorithm is available in the technical report [13]). In this paper, we show that the problem is 2-\textsc{EXPTIME}-hard, even if only entailment between predicate atoms is considered. The proof relies on a reduction from the membership problem for alternating Turing machines [5] whose working tape is exponentially bounded in the size of the input. This result gives the tight complexity for the problem, whose upper bound is 2-\textsc{EXPTIME} [11, 13].

This paper is a thoroughly revised version of a paper that was presented at the workshop ADSL 2020 (with no formal proceedings). Due to space restrictions, the proofs of several technical lemmas are omitted. They can be found in [7].

2 Separation Logic with Inductive Definitions

For any set $S$, we denote by $|S| \in \mathbb{N} \cup \{\infty\}$ its cardinality. For a partial mapping $f : A \rightarrow B$, let $\text{dom}(f) = \{x \in A \mid f(x) \text{ is defined}\}$ and $\text{rng}(f) = \{f(x) \mid x \in \text{dom}(f)\}$ be its domain and range, respectively, and we write $f : A \rightarrow_{\text{fin}} B$ if $|\text{dom}(f)| < \infty$. Given integers $n, m$, we denote by $\llbracket n \ldots m \rrbracket$ the set $\{n, n+1, \ldots, m\}$ (with $\llbracket n \ldots m \rrbracket = \emptyset$ if $n > m$). By a slight abuse of notation, we write $t \in t$ if $t = (t_1, \ldots, t_n)$ and $t = t_i$ for some $i \in \llbracket 1 \ldots n \rrbracket$.

Let $\text{Var} = \{x, y, \ldots\}$ be an infinite countable set of variables and $\text{Pred} = \{p, q, \ldots\}$ be an infinite countable set of uninterpreted relation symbols, called predicates. Each predicate $p$ has an arity $\#p \geq 1$, denoting the number of its arguments. In addition, we consider a special function symbol $\text{nil}$, of arity zero. A term is an element of the set $\text{Term} = \text{Var} \cup \{\text{nil}\}$. Let $\kappa \geq 1$ be an integer constant fixed throughout this paper, intended to denote the number of record fields. The logic $\text{SL}^\kappa$ is the set of formulæ generated inductively as follows:

$$
\phi ::= t_0 \mapsto (t_1, \ldots, t_n) \mid p(t_1, \ldots, t_{\#p}) \mid t_1 \approx t_2 \mid t_1 \neq t_2 \mid \phi_1 \ast \phi_2 \mid \exists x . \phi_1
$$

where $p \in \text{Pred}$, $t_i \in \text{Term}$, for all $i \in \llbracket 0 \ldots \max(\kappa, \#p) \rrbracket$ and $x \in \text{Var}$. A predicate-free formula is a formula of $\text{SL}^\kappa$ in which no predicate occurs. A formula of the form $t_0 \mapsto (t_1, \ldots, t_n) \mid p(t_1, \ldots, t_{\#p})$ is called a points-to atom [resp. predicate atom]. We write $\text{fv}(\phi)$ for the set of free variables in $\phi$, i.e., the variables $x$ that occur in $\phi$ outside of the scope of any existential quantifier $\exists x$. If $\text{fv}(\phi) = \{x_1, \ldots, x_n\}$ then $\phi[y_1/x_1, \ldots, y_n/x_n]$ denotes the formula obtained from $\phi$ by simultaneously substituting each $x_i$ with $y_i$, for $i \in \llbracket 1 \ldots n \rrbracket$.

To interpret $\text{SL}^\kappa$ formulæ, we consider a fixed, countably infinite set $\text{Loc}$ of locations and a designated location $\text{nil} \in \text{Loc}$. The semantics of $\text{SL}^\kappa$ formulæ is defined in terms of structures $(s, b)$, where:

- $s : \text{Term} \rightarrow \text{Loc}$ is a total mapping of terms into locations, called a store, such that $s(\text{nil}) = \text{nil}$,
- $b : \text{Loc} \rightarrow_{\text{fin}} \text{Loc}^\kappa$ is a finite partial mapping of locations into $\kappa$-tuples of locations, called a heap,
- such that $\text{nil} \not\in \text{dom}(b)$.

A location is allocated in a heap $b$ if it occurs in $\text{dom}(b)$. Two heaps $b_1$ and $b_2$ are disjoint iff $\text{dom}(b_1) \cap \text{dom}(b_2) = \emptyset$, in which case their disjoint union is denoted by $b_1 \uplus b_2$, undefined if $\text{dom}(b_1) \cap \text{dom}(b_2) \neq \emptyset$.

The satisfaction relation $\models$ between structures and predicate-free $\text{SL}^\kappa$ formulæ is defined, as usual, recursively on the syntax of formulæ:

$$
\begin{align*}
(s, b) \models t_1 \approx t_2 & \iff b = \emptyset \text{ and } s(t_1) = s(t_2) \\
(s, b) \models t_1 \neq t_2 & \iff b = \emptyset \text{ and } s(t_1) \neq s(t_2) \\
(s, b) \models t_0 \mapsto (t_1, \ldots, t_n) & \iff \text{dom}(b) = \{s(t_0)\} \text{ and } b(s(t_0)) = (s(t_1), \ldots, s(t_n)) \\
(s, b) \models \phi_1 \ast \phi_2 & \iff \text{there are disjoint heaps } b_1 \text{ and } b_2, \text{ such that } b = b_1 \uplus b_2 \\
\text{and } (s, b_1) \models \phi_i, \text{ for each } i = 1, 2 \\
(s, b) \models \exists x . \phi & \iff (s[x \leftarrow \ell], b) \models \phi, \text{ for some } \ell \in \text{Loc},
\end{align*}
$$

2
where \( s[x \leftarrow \ell] \) is the store mapping \( x \) into \( \ell \) and behaving like \( s \) for all \( t \in \text{Term} \setminus \{x\} \). Note that the semantics of \( t_1 \models t_2 \) and \( t_1 \not\models t_2 \) is strict, meaning that these atoms are satisfied only if the heap is empty.\(^3\)

### 2.1 Unfolding Trees

We now extend the previous semantics to handle formulæ containing predicate atoms. We assume that such predicates are associated with a set \( S \) of rules of the form \( p(x_1, \ldots, x_{\#p}) \Leftarrow \rho \), where \( \rho \) is an SL\(^c\) formula such that \( v(\rho) \subseteq \{ x_1, \ldots, x_{\#p} \} \). We refer to \( p(x_1, \ldots, x_{\#p}) \) as the head, and to \( \rho \) as the body of the rule. A rule is a base rule if its body is a predicate-free formula. We write \( p(x_1, \ldots, x_{\#p}) \Leftarrow S \rho \) if the rule \( p(x_1, \ldots, x_{\#p}) \Leftarrow \rho \) belongs to \( S \). In this section, we consider a given set of rules \( S \).

The above semantics is extended to formule that are not predicate-free, by recursively replacing predicate symbols by the body of a defining rule until a predicate-free formula is obtained, in a finite number of steps. For technical convenience, we place the steps of an unfolding sequence in a tree, such that the descendants of a node represent the unfoldings of predicate atoms produced by the unfolding of that particular node. Formally, a tree \( t \) is defined by a set of nodes \( \text{nodes}(t) \) and a function mapping each node \( w \in \text{nodes}(t) \) to its label, denoted by \( t(w) \). The set \( \text{nodes}(t) \) is a finite prefix-closed subset of \( \mathbb{N}^* \), where \( \mathbb{N}^* \) is the set of finite sequences of non-negative integers, meaning that if \( w \) and \( w'i \) are elements of \( \text{nodes}(t) \) for some \( i \in \mathbb{N} \setminus \{0\} \), then so is \( w'j \) for all \( j \in [0 \ldots i-1] \). We write \( |w| \) for the length of the sequence \( w \) and \( \lambda \) for the empty sequence, so that \( |\lambda| = 0 \). The root of \( t \) is \( \lambda \), the children of a node \( w \in \text{nodes}(t) \) are the nodes \( w'i \in \text{nodes}(t) \), where \( i \in \mathbb{N} \), and the parent of a node \( w \) with \( i \in \mathbb{N} \) is \( w \) (hence, \( \lambda \) has no parent). The subtree of \( t \) rooted at \( w \) is denoted by \( t_w \); it is formally defined by \( \text{nodes}(t_w) \equiv \{ w' \mid w' \in \text{nodes}(t) \} \) and \( t_w(w') \equiv t(w'w') \), for all \( w' \in \text{nodes}(t_w) \). For simplicity, we define unfolding trees below only for predicate atoms.\(^4\)

**Definition 1.** An unfolding tree of a predicate atom \( p(t_1, \ldots, t_{\#p}) \) is a tree \( t \), such that, for all \( w \in \text{nodes}(t) \), we have \( u(w) = (q(s_1, \ldots, s_{\#q}), \psi) \), for a predicate atom \( q(s_1, \ldots, s_{\#q}) \) and a formula \( \psi \), where:

1. if \( w = \lambda \) then \( q(s_1, \ldots, s_{\#q}) = p(t_1, \ldots, t_{\#p}) \).
2. \( \psi = p(x_1/s_1, \ldots, x_{\#q}/s_{\#q}) \), for a rule \( q(x_1, \ldots, x_{\#q}) \Leftarrow \rho \), and
3. there exists a bijective mapping from the set of occurrences of predicate atoms in \( \psi \) and the children\(^5\) of \( w \), such that if an atom \( r(v_1, \ldots, v_{\#r}) \) is mapped to \( w'i \), for some \( i \in \mathbb{N} \), then \( u(wi) \) is of the form \( \psi = \exists v_1 \ldots \exists v_{\#r} . \varphi \Leftarrow \psi \psi \), for some formula \( \varphi \).

We denote by \( \mathcal{T}_S(p(t_1, \ldots, t_{\#p})) \) the set of unfolding trees for \( p(t_1, \ldots, t_{\#p}) \).

Given an unfolding tree \( t \in \mathcal{T}_S(p(t_1, \ldots, t_{\#p})) \) such that \( u(\lambda) = (p(t_1, \ldots, t_{\#p}), \psi) \), we define its characteristic formula inductively, as the predicate-free formula \( \Upsilon(u) \) obtained from \( \psi \) by replacing each occurrence of an atom \( q(s_1, \ldots, s_{\#q}) \) by \( \Upsilon(u_i) \), where \( i \) denotes the child of \( w \) that \( q(s_1, \ldots, s_{\#q}) \) is mapped to, by the bijection of point (3) in Definition 1. More precisely, if \( \psi = \exists y_1 \ldots \exists y_{\#r} . \varphi \Leftarrow \psi \psi \), where \( \varphi \) is predicate-free, then \( \Upsilon(u) = \exists y_1 \ldots \exists y_{\#r} . \varphi \Leftarrow \psi \Upsilon(u_i) \).

Given an SL\(^c\) formula \( \phi \) and a structure \( (s, h) \), we write \( (s, h) \models S \phi \) if and only if \( (s, h) \models \psi \) for some formula \( \psi \) is obtained from \( \phi \) by syntactically replacing each occurrence of a predicate atom \( p(t_1, \ldots, t_{\#p}) \) in \( \phi \) with a formula \( \Upsilon(u) \), for some unfolding tree \( t \in \mathcal{T}_S(p(t_1, \ldots, t_{\#p})) \). A structure \( (s, h) \) such that \( (s, h) \models S \phi \) is called an \( S \)-model of \( \phi \), or simply a model of \( \phi \), when \( S \) is clear from the context.

We may now define the class of entailment problems, which are the concern of this paper:

\(^3\)This semantics avoids using boolean conjunction: \( \phi \land x = y \equiv \phi \land x = y \), where \( x = y \) iff \( x \) and \( y \) are assigned the same location.

\(^4\)An unfolding tree for a generic SL\(^c\) formula can be obtained by joining the unfolding trees of its predicate atoms under a common root.

\(^5\)In particular, \( \psi \) is a predicate-free formula iff \( w \) is a leaf.

\(^6\)Note that the bijection between atoms and children is not necessarily unique. However, it is easy to check that all these mappings will eventually yield the same formula, up to a permutation of atoms.
Lemma 5. Let $h$ be a heap of a model of its characteristic formula. Moreover, each occurrence of a predicate atom in $\psi$ is of the form $q(y_1, u_1, \ldots, u_{\#_p-1})$, for some $i \in [1..k]$. Then $S$ is connected.

The progress and connectivity conditions induce a tight relationship between the models of predicate atoms and their corresponding unfolding trees, formalized below:

Definition 4. Given a heap $h$ and a tree $t$, an embedding of $t$ into $h$ is a bijection $\Lambda : \text{nodes}(t) \rightarrow \text{dom}(h)$ such that $\Lambda(wi) \in h(\Lambda(w))$, for each node $wi \in \text{nodes}(t)$, where $i \in \mathbb{N}$.

The following lemma states that every unfolding tree of a predicate atom can be embedded into the heap of a model of its characteristic formula.

Lemma 5. Let $S$ be a progressing and connected set of rules and $(s, h)$ be a structure such that $(s, h) \models_S p(t_1, \ldots, t_{\#_p})$. Then there exists an unfolding tree $u \in T_S(p(t_1, \ldots, t_{\#_p}))$ such that $(s, h) \models \Upsilon(u)$, and an embedding of $u$ into $h$.

The embedding whose existence is stated by Lemma 5 provides a way of decorating the allocated locations in a heap by the predicate symbols that caused their allocation. Given a structure $(s, h)$ such that $(s, h) \models_S p(t_1, \ldots, t_{\#_p})$ a predicate decoration of $h$ w.r.t. $p(t_1, \ldots, t_{\#_p})$ is a function $\Delta : \text{dom}(h) \rightarrow \text{Pred}$ defined as $\Delta(t) \equiv q$ if and only if $u(\Delta^{-1}(t)) = (q(s_1, \ldots, s_{\#_p}), \psi)$, for some unfolding tree $u \in T_S(p(t_1, \ldots, t_{\#_p}))$ such that $(s, h) \models \Upsilon(u)$ and some embedding $\Lambda$ of $u$ into $h$. Note that the unfolding tree $u$ and function $\Delta$ are not unique, hence predicate decorations are not unique in general.

The third condition ensuring decidability requires that all the existentially quantified variables introduced during an unfolding can only be associated with locations that are allocated in the heap of any model of the formula (the condition given below is equivalent to the one given in [8]).

Definition 6. A set of rules $S$ is established if and only if, for each rule $p(x_1, \ldots, x_{\#_p}) \Leftarrow S \exists x_1 \ldots \exists x_{\#} \cdot \psi$ and for each $S$-model $(s, h)$ of $\psi$, we have $s(z_1), \ldots, s(z_m) \in \text{dom}(h)$.

Checking establishment is co-NP-hard [10]. In the following, we consider only sets of rules that are progressing, connected and established (PCE). The interest for PCE sets of rules is motivated by the following decidability result, proved in [8]:

Theorem 7. Given a PCE set of rules $S$ and two formulæ $\phi$ and $\psi$ the problem $\phi \models_S \psi$ belongs to ELEMENTARY.

The rest of this paper is concerned with proving that the entailment problem $\phi \models_S \psi$, for PCE sets of rules $S$, is 2-EXPTIME-hard. Previously, an EXPTIME-hard lower bound for this problem was established in [9].
4 Alternating Turing Machines

The proof of 2-EXPTIME-hardness relies on a reduction from the membership problem for alternating Turing machines. We recall some basic definitions below.

Definition 8. An Alternating Turing Machine (ATM) is a tuple $M = (Q, \Gamma, \delta, q_0, g)$ where:

- $Q$ is a finite set of control states,
- $\Gamma = \{\gamma_1, \ldots, \gamma_N, b\}$ is a finite alphabet, $b$ is the blank symbol,
- $\delta \subseteq Q \times \Gamma \times (\Gamma \setminus \{b\}) \times \{\leftarrow, \rightarrow\}$ is the transition relation, $(q, a, q', b, \mu) \in \delta$ meaning that, in state $q$, upon reading symbol $a$, the machine moves to state $q'$, writes $b \neq b$ to the tape \footnote{A machine never writes blank symbols, that are used only for the initially empty tape cells.} and moves the head by one to the left [resp. right] if $\mu = \leftarrow$ [resp. $\mu = \rightarrow$],
- $q_0 \in Q$ is the initial state, and
- $g : Q \to \{\lor, \land\}$ partitions the set of states into existential ($g(q) = \lor$) and universal ($g(q) = \land$) states.

A configuration of an ATM $M = (Q, \Gamma, \delta, q_0, g)$ is a tuple $(q, w, i)$ where $q \in Q$ is the current state, $w : \mathbb{N} \to \Gamma$ represents the contents of the tape and is such that $\|\{j \in \mathbb{N} \mid w(j) \neq b\}\| < \infty$, and $i \in \{0, \ldots, \max\{j \in \mathbb{N} \mid w(j) \neq b\} + 1\}$ is the current position of the head on the tape. We denote by $\epsilon$ the empty word over $\Gamma$. For any tape $w$ and integer $i$, we denote by $w[i \leftarrow a]$ the tape $w'$ such that $w'(i) = a$ and $w'(j) = w(j)$ for all $j \neq i$. In the following, we write $i^\leftarrow \overset{\delta}{\Rightarrow} i - 1$ if $i > 0$ ($0^\leftarrow \overset{\delta}{\Rightarrow}$ is undefined) and $i^\leftarrow \overset{\delta}{\Rightarrow} i + 1$. Note that, since $0$ denotes the leftmost position on the tape, no transition moves the head left of 0.

The step relation of $M$ is the following relation between configurations: $(q, w, i) \overset{(q, a, q', b, \mu)}{\rightarrow} (q', w', j)$ if and only if there exists a transition $(q, a, q', b, \mu) \in \delta$ such that $w(i) = a$, $w' = w[i \leftarrow b]$ and $j = \delta^3$ is defined, i.e., either $i > 0$ or $\mu \neq \leftarrow$. We omit specifying the transition $(q, a, q', b, \mu)$ when it is not important. An execution is a sequence $(q_0, w_0, 0) \overset{(q_0, a_0, q_1, b_0, \mu_0)}{\rightarrow} (q_1, w_1, i_1) \overset{(q_1, a_1, q_2, b_1, \mu_1)}{\rightarrow} \cdots$ Note that an execution is entirely determined by the initial configuration $(q_0, w_0, 0)$ and the sequence $(q_0, a_0, q_1, b_0, \mu_0), (q_1, a_1, q_2, b_1, \mu_1), \ldots$ of transition rules applied to it.

Given a function $f : \mathbb{N} \to \mathbb{N}$, an execution is $f$-space bounded if and only if $|w_i| \leq f(|w_0|)$, for all $i > 0$. The ATM $M$ is exponential-space bounded if there exists a constant $c$ such that every execution is $f^c$-space bounded, where $f(x) = c \cdot 2^{25^x}$ for some constant $c$ and some univariate polynomial function $g$.

Definition 9. A derivation of an ATM $M = (Q, \Gamma, \delta, q_0, g)$, starting from a configuration $(q_0, w_0, 0)$, is a finite tree whose nodes are either:

1. branching nodes labeled with configurations $(q, w, i) \in Q \times \Gamma^* \times \mathbb{N}$, or
2. action nodes labeled with tuples $(a, b, \mu) \in \Gamma \times \{\{\leftarrow, \rightarrow\}, a$ is the symbol read, $b$ is the symbol written and $\mu$ is the move of the head at that step, such that the root of $t$ is a branching node $t(l) = (q_0, w_0, 0)$ and, moreover:
   a. each branching node labeled by $(q, w, i)$ such that $g(q) = \lor$ has exactly one child, which is an action node labeled by $(a, b, \mu)$, where $a = w(i)$ and $(q, a, q', b, \mu) \in \delta$; the child of which is a branching node labeled by $(q', w', j)$, such that $(q, w, i) \overset{(q, a, q', b, \mu)}{\rightarrow} (q', w', j)$;
   b. each branching node labeled by $(q, w, i)$ such that $g(q) = \land$ has exactly one child for each tuple $(q, a, q', b, \mu) \in \delta$ such that $a = w(i)$; this child is an action node labeled by $(a, b, \mu)$, the child of which is a branching node labeled by $(q', w', j)$, where $(q, w, i) \overset{(q, a, q', b, \mu)}{\rightarrow} (q', w', j)$.

We say that $M$ accepts $w$ if and only if $M$ admits a derivation starting from $(q_0, w, 0)$.

Note that the leaves of such a tree are necessarily branching nodes labeled by a triple $(q, w, i)$ such that $g(q) = \land$ and there is no transition $(q, a, q', b, \mu)$ with $a = w(i)$. 

Echenim, Iosif and Peltier
classes are closed under complement.

Entailment Checking in Separation Logic with Inductive Definitions is 2-EXPTIME

Example 10. Consider an ATM $M = (Q, \Gamma, \delta, q_0, g)$, where: $Q = \{q_0, q_1, q_2\}$, $\Gamma = \{a, b, c, \mathbf{b}\}$, $\delta = \{(q_0, b, a, q_1, \rightarrow), (q_0, b, b, q_2, \rightarrow), (q_0, b, b, q_2, \rightarrow), (q_0, b, b, q_2, \rightarrow), (q_0, b, c, c, q_2, \rightarrow), (q_1, b, a, q_0, \leftarrow), \}$. $g(q_0) = q(q_2) = \land$ and $g(q_1) = \lor$. A derivation for $M$, starting from an empty tape $\epsilon$, is depicted in Figure 1 (the ATM contains additional transitions not used here, they will be useful in upcoming examples). The run is on a tape of length 2, hence the position is encoded by a single digit.

Definition 11. The membership problem $(M, w)$ asks the following: given an ATM $M = (Q, \Gamma, \delta, q_0, g)$ and a word $w \in (\Gamma \setminus \{\mathbf{b}\})^*$ does $M$ accept $w$?

The complexity class $\text{AEXPSPACE}$ is the class of membership problems where $M$ is exponential-space bounded. It is known that $\text{AEXPSPACE} = \text{co-AEXPSPACE} = 2\text{-EXPTIME}$ [5], where $\text{co-AEXPSPACE}$ is the complement class of $\text{AEXPSPACE}$.

In the following, we shall consider only the membership problem $(M, \epsilon)$. This is without loss of generality: indeed, let $(M, w)$ be any instance of the membership problem, and let $c$ and $g$ be the constant and polynomial function witnessing the fact that $M$ is exponential-space bounded. Let $M_w$ be an ATM that produces $w$ starting from input $\epsilon$. Clearly, $M_w$ uses at most $|w|$ working space, thus the machine $M_w; M$, which runs $M_w$ on the empty word and then continues with $M$, runs in space $c \cdot 2^{|w|}$ and accepts $\epsilon$ if and only if $M$ accepts $w$. If $9l \geq \log_2(c) + g(|w|)$, then $M_w; M$ runs in space $2^{|q_0|}$ and moreover, $(M, w)$ and $(M_w, \epsilon)$ have the same answer. Therefore, we assume from now on that $M = (Q, \Gamma, \delta, q_0, g)$ is an ATM started in the configuration $(q_0, \epsilon, 0)$ and that $M$ runs in space at most $2^{|q_0|}$ on the empty input word, where $9l$ is bounded by a polynomial in the length of $w$.

5 The Reduction

This section describes the reduction of the membership problem (Definition 11) for exponential-space bounded ATMs to the entailment problem (see Definition 2) for PCE sets of rules (Definitions 3 and 6). The main idea of the reduction is the following. Since the membership problem is existential (asking for the existence of a derivation) and the entailment problem is universal (every model of the left-hand side is a model of the right-hand side), a direct reduction is not possible. Instead, we reduce from the complement of the membership problem $(M, w)$ (there is no derivation of $M$ on $w$) to an entailment problem instance $p_M(x) \models_{S_M} c_M(x)$, where $p_M(x)$, $c_M(x)$ are predicate atoms and $S_M$ is a PCE set of rules derived from the description of $M$. Intuitively, $p_M(x)$ defines all heaps with a predicate decoration that simulates the control structure of $M$ (i.e., the branching and action nodes alternate and the control states given by the predicate decoration are consistent with the transitions of $M$), with no regard to the tape contents or the position of the head. Then $c_M(x)$ defines only those heaps that encode derivations

---

8Every ATM can be complemented in linear time, by interchanging the existential with the universal states, thus all alternating classes are closed under complement.
having just 0 shall write tuples, with the convention that these tuples are always encoded as binary heaps. More precisely, we

5.1 Syntactic Shorthands

Before giving the definitions of \( p_M, c_M \) and \( S_M \), we introduce several syntactic shorthands that simplify the presentation. To simplify notations, we shall assume in the remainder of the paper that all heaps and unfolding trees are defined on the extended syntax. For instance, although the final encoding uses only binary heaps, i.e. for \( \kappa = 2 \), we shall actually write formulæ in which points-to atoms refer to arbitrary tuples, with the convention that these tuples are always encoded as binary heaps. More precisely, we shall write \( h(\ell) = (\ell_1, \ell_2, \ell_3) \) to state that \( \ell \) refers to a pair \( (\ell_1, \ell'_2) \) where \( \ell'_2 \) itself refers to \( (\ell_2, \ell_3) \), and this additional location \( \ell'_1 \) will never be explicitly referred to. Similarly, unfolding trees will also be defined by taking into account this syntactic extension, i.e., points-to atoms with arbitrary tuples will be allowed to occur in the labels of the unfolding trees, bearing in mind that such atoms will actually yield additional unfolding steps, which will not be explicitly considered in the tree.

Encoding Tuples Let \( t = (t_1, \ldots, t_n) \) be a tuple of terms, with \( n > 2 \). Let \( \psi = \psi_1 * \cdots * \psi_n \) be a (possibly empty) separated conjunction of predicate atoms, where the first argument of every predicate atom in \( \psi_i \) is \( t_i \). By writing:

\[
p(x) \iff \exists y_1 \ldots \exists y_r . x_1 \mapsto (t_1, \ldots, t_n) * \psi
\]

we denote the rules:

\[
p(x) \iff \exists z_1 \exists y_1 \ldots \exists y_r . x_1 \mapsto (t_1, z_1) * \psi_1 * \overline{p}_1(z_1, x, y_1, \ldots, y_r)
\]

\[
\overline{p}_i(z_i, x, y_1, \ldots, y_r) \iff \exists z_{i+1} \exists z_i \mapsto (t_{i+1}, z_{i+1}) * \psi_{i+1} * \overline{p}_{i+1}(z_{i+1}, x, y_1, \ldots, y_r), \text{ for } i \in [1 \ldots n-2]
\]

\[
\overline{p}_{n-2}(z_{n-1}, x, y_1, \ldots, y_r) \iff z_{n-1} \mapsto (t_{n-1}, t_n) * \psi_n
\]

where \( \overline{p}_1, \ldots, \overline{p}_{n-1} \) are fresh pairwise distinct predicate symbols.

The intuition is that the tuple \( (t_1, \ldots, t_n) \) is represented by a binary tree of the form \( (t_1, (\ldots, (t_{n-1}, t_n) \ldots)) \) of depth \( n-1 \). This allows one to encode records of various, non-constant lengths \( n \) by using only a constant number of record fields (here, \( \kappa = 2 \)). Note that the obtained rules are progressing, and, by definition of \( \psi_1, \ldots, \psi_n \), they are connected. Moreover they are established when the initial rule is established, since the variables \( y_1, \ldots, y_r \) are allocated in \( \psi \) and every variable \( z_i \) is allocated by \( \overline{p}_i \). In the following the term \( (s^a, t) \) will be a shorthand for \( (s, \ldots, s, t) \) and \( [t]^n \) will stand for \( (\text{nil}^n, t) \). The interest of such special tuples will be explained later (essentially we will need to introduce “dummy” cells (\( \text{nil}, \ldots, \text{nil} \)) to ensure that all rules are progressing).

Global Variables We assume the existence of the following global variables that occur free in each formula: \( 0, 1, y_1, \ldots, y_N \). The variables 0 and 1 denote binary digits, and the variable \( y_i \) (\( 1 \leq i \leq N \)) denote non-blank symbols from the alphabet \( \Gamma^0 \). These variables will always be assigned pairwise distinct allocated locations, as required by the following rules:

\[
\text{Const}(x) \iff x \mapsto (0, 1, y_1, \ldots, y_N) * a(0) * a(1) * \psi_i^{N} a(y_i) \quad (1)
\]

\[
a(x) \iff x \mapsto (\text{nil}, \text{nil}) \quad (2)
\]

Since any membership problem is equivalent to a membership problem on a binary alphabet, via a binary encoding of \( \Gamma \), having just 0 and 1 suffices. We consider distinct alphabet symbols \( y_1, \ldots, y_N \) only to avoid clutter.
Considering global variables is without loss of generality in the following, because these variables can be added to the parameter list of each head in the system (at the expense of cluttering the presentation).

**Binary Choices** We introduce a special symbol $\bullet$ which, when occurring in the body of a rule, ranges over the global variables $0$ and $1$. Thus any rule of the form:

$$p(x_1, \ldots, x_p) \leftarrow \exists x_1 \ldots \exists x_n . x_1 \mapsto (\bullet, y) * \psi$$

stands for the following two rules:

$$p(x_1, \ldots, x_p) \leftarrow \exists x_1 \ldots \exists x_n . x_1 \mapsto (0, y) * \psi$$

$$p(x_1, \ldots, x_p) \leftarrow \exists x_1 \ldots \exists x_n . x_1 \mapsto (1, y) * \psi$$

and similarly for rules of the form $p(x_1, \ldots, x_p) \leftarrow \exists x_1 \ldots \exists x_n . x_1 \mapsto (y, \bullet) * \psi$. The elimination of the occurrences of $\bullet$ must be done after the encoding of tuples by binary trees, so that the number of rules is increased by a constant $\kappa^2 = 2^2$. Note also that the fact that each rule allocates only one cell and that $\kappa = 2$ (more generally that $\kappa$ is a constant) is essential here, since otherwise the elimination of $\bullet$ would yield an exponential blow-up.

**Example 12.** A rule $p(x) \leftarrow x \mapsto (\bullet^4)$ is first transformed into:

$$p(x) \leftarrow \exists x_1 . x \mapsto (\bullet, x_1) * p_1(x_1) \quad p_1(x_1) \leftarrow \exists x_2 . x_1 \mapsto (\bullet, x_2) * p_2(x_2) \quad p_2(x_2) \leftarrow x_2 \mapsto (\bullet, \bullet)$$

Afterwards, the symbol $\bullet$ is eliminated, yielding:

$$p(x) \leftarrow \exists x_1 . x \mapsto (0, x_1) * p_1(x_1) \quad p(x) \leftarrow \exists x_1 . x \mapsto (1, x_1) * p_1(x_1)$$

$$p_1(x_1) \leftarrow x_1 \mapsto (0, x_2) * p_2(x_2) \quad p_1(x_1) \leftarrow x_1 \mapsto (1, x_2) * p_2(x_2)$$

$$p_2(x_2) \leftarrow x_2 \mapsto (0, 0) \quad p_2(x_2) \leftarrow x_2 \mapsto (0, 1)$$

$$p_2(x_2) \leftarrow x_2 \mapsto (1, 0) \quad p_2(x_2) \leftarrow x_2 \mapsto (1, 1)$$

We obtain $4 \cdot 2 = 8$ rules. If the first transformation is omitted then we get $2^4 = 16$ rules.

**Binary Variables** A binary variable $b$ is understood as ranging over the domain of the interpretation of $0$ and $1$, namely the locations assigned to $0$ and $1$ by the formula $\text{Const}(1)$. Additionally, for each binary variable $b$, we consider the associated variable $\overline{b}$, intended to denote the complement of $b$. More precisely, the formula $\exists b . \psi$ is to be understood as $\psi[0/b, 1/\overline{b}] \lor \psi[1/b, 0/\overline{b}]$. However, this direct substitution of the (existentially quantified) binary variables by $0$ and $1$ within the rules of an established system would break the establishment condition (Definition 6), because $0$ and $1$ are not necessarily allocated within the body of the rule\(^\text{10}\). This problem can be overcome by passing $0$ and $1$ as parameters to a fresh predicate. More precisely, a rule of the form (with $1 \leq i \leq m$):

$$p(x_1, \ldots, x_p) \leftarrow \exists b_1 \ldots \exists b_i \exists y_1 \ldots \exists y_n . x_1 \mapsto [t]^m * \psi$$

is a shorthand for the following set of rules:

$$p(x_1, \ldots, x_p) \leftarrow \exists y . x_1 \mapsto (\text{nil}, y) * p'(y, x_1, \ldots, x_p, 0, 1)$$

$$p(x_1, \ldots, x_p) \leftarrow \exists y . x_1 \mapsto (\text{nil}, y) * p'(y, x_1, \ldots, x_p, 1, 0)$$

$$p'(y, x_1, \ldots, x_p, b_1, b_1) \leftarrow \exists b_2 \ldots \exists b_i \exists y_1 \ldots \exists y_n . y \mapsto [t]^m * \psi$$

Clearly, the elimination of the binary existential quantifiers from the rule (3) adds $2 \cdot i$ rules to the set. Note that the hat $[t]^m$, of height $m \geq i$ decreases at each step of the elimination which ensures that the

\(^{10}\)In fact they are allocated by the side condition $\text{Const}$. 

8
Entailment Checking in Separation Logic with Inductive Definitions is 2-EXPTIME-hard
Echenim, Iosif and Peltier

definition is well-founded. It is easy to check that the resulting rules are progressing and connected. Furthermore, they are also established, if the variables \( y_1, \ldots, y_n \) are allocated in \( \psi \). The rule (3) is equivalent to \( 2^n \) rules of the form \( p(x_1, \ldots, x_{\#p}) \iff \exists y_1 \ldots \exists y_m \cdot x_1 \mapsto [t]^m \star \psi \) where every \( b_j \) is replaced by \( 0 \) or \( 1 \) and \( \overline{b_j} \) is replaced by the complement of \( b_j \). However, adding the variables \( b_j \) and \( \overline{b_j} \) one by one as parameters to the predicate allows one to represent these rules concisely, using only \( 2 \cdot i \) additional rules. This comes with a cost: since the progress condition requires each rule to allocate exactly one location, the vector \( t \) must be embedded into a tuple \([t]^m\) of length at least \( i \).

Next, we introduce a syntactic shorthand to denote disequality constraints on vectors of binary variables. For a vector \( b = (b_1, \ldots, b_n) \) of binary variables, we denote by \( \tilde{b} \) the vector \((\overline{b}_1, \ldots, \overline{b}_n)\). The following rule:

\[
p(x_1, \ldots, x_{\#p}) \iff \exists c_1 \ldots \exists c_n \exists y_1 \ldots \exists y_m \cdot x_1 \mapsto t \star \psi \mid (c_1, \ldots, c_n) \neq (b_1, \ldots, b_n)
\]

where each \( c_j \) \((1 \leq i \leq n)\) occurs at most once in \( t \) and does not occur in \( \psi \) and \( b_1, \ldots, b_n \in \{x_1, \ldots, x_{\#p}\} \), is a shorthand for the following set of rules:

\[
p(x_1, \ldots, x_{\#p}) \iff \exists y_1 \ldots \exists y_m \cdot x_1 \mapsto (t[b_i/c_i]) \cdot (c_j) \cdot \mathbf{p}[\cdot] \cdot \mathbf{i} \star \psi, \ i \in [1 \ldots n]
\]

Intuitively, rule (4) introduces new binary variables \( c_1, \ldots, c_n \), such that not all of them are equal to the complements of \( b_1, \ldots, b_n \), respectively. In other words, one \( c_i \) must be equal to \( b_i \), for some \( i \in [1 \ldots n] \), and the other \( c_j \) for \( j \neq i \) are arbitrary (hence they can be replaced by \( \cdot \) since they occur only once in \( t \)). Note that expanding rule (4) as described above (see rule (5)) results in at most \( n \) rules of the form (3), hence the full elimination of binary variables from the system is possible in polynomial time. This is mainly because in our reduction, described next, both \( i \) (in (3)) and \( n \) (in (4)) are bounded by \( \#l \), which in turn, is polynomially bounded by the length of the input to the membership problem.

5.2 Pseudo-derivations as Heaps

In this section, we show how to encode the general structure of a derivation as a heap and define a set of rules that generates exactly the structures corresponding to these derivations. Importantly, since \( M \) starts on the empty word \( \epsilon \), the tape contents in a branching node can be derived from the sequence of actions along the path from the root to that node. For this reason, we shall not explicitly represent tape contents within the configurations and simply label branching nodes with pairs \((q, i) \in Q \times [0 \ldots 2^n - 1]\).

We first define pseudo-derivations, in which the conditions on derivations are relaxed by removing all the constraints related to the content of the tape and the position of the head (such conditions will be considered in Section 5.3). In other words, in a pseudo-derivation, the ATM is treated as a mere alternating automaton, enriched with arbitrary (and possibly inconsistent) read/write/move actions on the tape. More formally:

**Definition 13.** A **pseudo-derivation** of \( M = (Q, \Gamma, \delta, q_0, g) \) is a tree \( t \), whose nodes are either:

1. branching nodes labeled with pairs \((q, i) \in Q \times \mathbb{N}, \text{ or} \)
2. action nodes labeled with tuples \((a, b, \mu) \in \Gamma \times \Gamma \setminus \{a\} \times \{\leftarrow, \rightarrow\}, \text{ where } a \text{ is the symbol read, } b \text{ is the symbol written and } \mu \text{ is the move of the head at that step, such that the root of } t \text{ is a branching node, } t(\lambda) = (q_0, 0) \text{ and, moreover:} \)
   a. each branching node labeled by \((q, i)\), such that \( g(q) = \lor\), has exactly one child that is an action node labeled by \((a, b, \mu) \in \delta, \text{ the child of which is a branching node labeled by } (q', j) \text{ such that } (q, a, b, q', \mu) \in \delta \text{ and } j \in \mathbb{N}\);
   b. each branching node labeled by \((q, i)\) where \( g(q) = \land\) has exactly one child for each tuple \((q, a, q', b, \mu) \in \delta; \text{ this child is an action node labeled by } (a, b, \mu), \text{ the child of which is a branching node labeled by } (q', j), \text{ where } j \in \mathbb{N}\).
Definition 13 is similar to Definition 9 except that all the conditions related to the content of the tape and to the position of the head have been removed (i.e., one does not check that the symbol $a$ occurs at position $i$ in the tape or that $j = i'$). Any derivation starting from an empty tape $ε$ can be associated with a pseudo-derivation, simply by replacing the label $(q, w, i)$ of the branching nodes by $(q, i)$. Conversely, for some pseudo-derivations, we may obtain an isomorphic derivation by inductively replacing the labels of the branching nodes from the root to the leaves as follows. Initially, the label $(q_0, 0)$ of the root of the tree is replaced by $(q_0, ε, 0)$. Afterwards, if a branching node is relabeled by $(q, w, i)$ and is followed by an action node $ω$ labeled by $(a, b, µ)$, then the label $(q', i')$ of the branching node following $ω$ is replaced by $(q', w[i ← b], i')$. If the obtained tree is a derivation, then we say that the pseudo-derivation yields a derivation. Note that this is not always the case, because the conditions on the read actions and on the moves in the tape are not necessarily satisfied: a branching node $(q, w, i)$ may be followed by an action $(a, b, µ)$ such that $a ≠ w[i]$, and the latter node may be followed by a branching node $(q', w', i')$ with $i' ≠ i"$. Figure 2 gives an example of a pseudo-derivation yielding no derivation, for the ATM of Example 10. The parts of the labels that do not fulfill the desired properties are underlined (the symbols $b$ and $c$ do not match the symbols read on the tape, and $0$ does not match the position of the head). The conditions ensuring that a pseudo-derivation yields a derivation will be given in Section 5.3.

We represent the pseudo-derivations of $M$ as tree-shaped heaps generated by a set of rules where, intuitively, each predicate $q(x)$ allocates a branching node labeled by a pair $(q, i)$ and each predicate $\overline{q}(x, a, b, µ)$ allocates an action node labeled $(a, b, µ)$. In our representation, the state $q$ will actually be omitted (see, e.g., Rule (6)), because it is implicitly defined by the unfolding tree. Further, we represent each position $i ∈ \{0..2^{2^N}-1\}$ on the tape succinctly, by an $N$-tuple of binary digits $\text{bin}(i) ∈ \{0, 1\}^N$ and encode the left and right moves as $← = \text{def} \ 0$ and $→ = \text{def} \ 1$. Let $τ(q, a) = \text{def} \ (\delta(q) × a) × Q × Γ \setminus \{b\} × \{←, →\}$ be the set of transitions of $M$ with source state $q$, reading symbol $a$ from the tape. We consider the following rules, for each state $q ∈ Q$ and symbol $a ∈ Γ$:

\[
q(x) \iff \exists x'. x → (\bullet^R, x') * \overline{q}(x', a, b, \overline{µ}) \quad \text{if} \ g(q) = 1 \text{ and } (q, a, q', b, µ) ∈ τ(q, a) \\
q(x) \iff \exists y_1...y_m. x → (\bullet^R, y_1,...,y_m) * \bigotimes_{j=1}^m \overline{q}_j(y_j, a, b_j, \overline{µ}_j) \quad \text{if} \ g(q) = 1 \text{ and } τ(q, a) = \{(q, a, q_1, b_1, µ_1),..., (q, a, q_m, b_m, µ_m)\} \\
\overline{q}(x, y, z, u) \iff \exists x'. x → (y, z, u, x') * q(x')
\]

(6) \hspace{1cm} (7) \hspace{1cm} (8)

The heaps defined by the above rules ensure only that the control structure of a derivation of $M$ is respected, namely that the branching and action nodes alternate correctly, and that the sequence of control states labeling the branching nodes on any path is consistent with the transition relation of $M$. In other words, these trees encode pseudo-derivations of $M$. Further, we introduce a top-level predicate $p_M(x)$ that allocates the special variables $0, 1, y_1,...,y_N$ and ensures that the initial state $q_0$ of $M$ is the first control state that occurs on an path of a pseudo-derivation:

\[
p_M(x) \iff \exists y \exists x'. x → (y, z) * p'_M(y) * \text{Const}(z) \quad \text{if} \ g(q) = 1 \\
p'_M(y) \iff \exists x'. y → [z']^N * q_0(z')
\]

(9) \hspace{1cm} (10)
The hat $[z']^R$ above ensures that every heap generated by $p'_M$ begins with a tuple $[z']^R \triangleq (\text{nil}, \ldots, \text{nil}, z')$. The use of this tuple will be made clear in Section 5.3. For now, let $S_M$ be the set consisting of the rules above. In the following, we stick to the convention that predicate symbol $q$ represents a branching node, whereas $\mathcal{R}$ represents an action node. The definition below formalizes the encoding of a pseudo-derivation by a structure:

**Definition 14.** A structure $(s, b)$ such that $(s, b) \models_{S_M} p_M(x)$ encodes a pseudo-derivation $t$ of $M$, written as $(s, b) \triangleright t$, if and only if there exists a predicate decoration $\Delta$ of $b$ w.r.t. $p_M(x)$, two heaps $b_1$ and $b_2$ and a bijection $f : \text{nodes}(t) \to \text{dom}(b_2)$ such that, for all $w \in \text{nodes}(t)$, the following hold:

1. $b = b_1 \uplus b_2$.
2. $(s, b_1) \models \exists y \exists z. x \mapsto (y, z) \cdot \text{Const}(z) \cdot y \mapsto [z']^R$.
3. If $w$ is a branching node with label $l(w) = (q, i)$ and children $w0, \ldots, wn$, then $\Delta(f(w)) = q$ and $b_2(f(w)) = (\ell_1, \ldots, \ell_n, f(w0), \ldots, f(wn))$, where $\ell_j = s(\text{bin}(i)_j)$, for all $j \in \{1 \ldots n\}$.
4. If $w$ is an action node with label $l(w) = (a, b, \mu)$ and only child $w0$, then we have $b_2(f(w)) = (s(a), s(b), s(\mu), f(w0))$.

A heap encoding the derivation of Figure 1 is depicted in Figure 3 (for readability, the part corresponding to the formula $\exists y \exists z. x \mapsto (y, z) \cdot \text{Const}(z) \cdot y \mapsto [z']^R$ is not depicted, i.e., only the heap $b_2$ of Definition 14 is shown). We also give, for each location $\ell$, the corresponding predicate $\Delta(\ell)$.

**Lemma 15.** (A) For each pseudo-derivation $t$ of $M$, there exists a structure $(s, b) \models_{S_M} p_M(x)$ such that $(s, b) \triangleright t$. (B) Dually, for each structure $(s, b) \models_{S_M} p_M(x)$, there exists a pseudo-derivation $t$ of $M$ such that $(s, b) \triangleright t$.

### 5.3 Encoding Complement Membership as Entailment Problems

In this section, we show how to encode the conditions that ensure that a pseudo-derivation is a derivation, namely that the considered pseudo-derivation also fulfills all the conditions related to the tape contents and the position of the head. More precisely, we recall that a pseudo-derivation of $M$ yields a derivation of $M$ if the contents of the tape and the head’s position are consistent with the sequence of actions leading to that particular configuration. This is the case if the following conditions hold:
I. If a branching node labeled \((q, i)\) is followed by an action node labeled \((a, b, \rightarrow)\) [resp. \((a, b, \leftarrow)\)], itself followed by a branching node labeled \((q', i')\) then necessarily \(i' = i + 1\) [resp. \(i = i' + 1\)], i.e. the position of the head changes according to the action executed between the adjacent configurations (for instance, in Figure 2, the position 0 does not fulfill this condition).

II. For every \(i \in [0..2^{|\mathbb{B}|} - 1]\), if along a path from a branching node labeled \((q, i)\) followed by an action node labeled \((a, b, \mu)\), to another branching node labeled \((q', i)\) followed by an action node labeled \((a', b', \mu')\), there is no branching node labeled \((q'', i)\), then necessarily \(a' = b\). Indeed, the symbol read on position \(i\) must be the one previously written, since it was not changed in the meantime (e.g., in Figure 2, the symbol \(b\) does not fulfill this condition).

III. For every \(i \in [0..2^{|\mathbb{B}|} - 1]\), if along a path from the root to a branching node labeled \((q, i)\), followed by an action node labeled \((a, b, \mu)\), there is no branching node labeled \((q', i)\), then necessarily \(a = b\), i.e. the tape is initially empty (e.g., this condition is violated by the symbol \(b\) in Figure 2).

In the following, we shall not check that the above conditions hold for some derivation of \(M\), but rather the opposite: that for each pseudo-derivation of \(M\), at least one of the above conditions is broken. In other words, we reduce from the complement of the membership problem \((M, \epsilon)\) to an entailment problem, defined next. This does not change the final 2-EXPSPACE-hardness result, because, as previously mentioned, 2-EXPSPACE = \(\text{AEXPSPACE} = \text{co-AEXPSPACE}\).

To this end, we consider a predicate \(c_M\) and a set of rules \(S_M\) containing rules for \(p_M(x)\) and \(c_M(x)\) such that the entailment \(p_M(x) \models_S_M c_M(x)\) holds if and only if every pseudo-derivation of \(M\) violates at least one of the conditions (I), (II) or (III); in other words, if and only if \(M\), started on input \(\epsilon\), admits no derivation.

Let \(\mathbb{B} = \max_{\mu \in Q, a \in \Gamma} ||\tau(q, a)||\) be the maximum branching degree (i.e. the maximum number of children of a node) of a derivation of \(M\). We define an auxiliary predicate \(r(x)\) that generates all tree-shaped heaps in which branching nodes correctly alternate with action nodes, with no regard for the labels of those nodes:

\[
\begin{align*}
  r(x) & \iff \exists y_1 \ldots \exists y_n \cdot x \rightarrow (\bullet^{|\mathbb{B}|}, y_1, \ldots, y_n) \ast \exists_{j=1}^n \tau(y_j), \text{ for each } n \in [0..|\mathbb{B}|] \\
  \tau(x) & \iff \exists y \cdot x \rightarrow (a, b, \bullet, y) \ast r(y), \text{ for each } a \in \Gamma \text{ and } b \in \Gamma \setminus \{b\}
\end{align*}
\]

First, we define the heap encodings of those pseudo-derivation trees that violate condition (I). To this end, we guess a vector \(\bar{b}\) in \([0,1]^{|\mathbb{B}|}\), encoding a position on the tape \(i \in [0..2^{|\mathbb{B}|} - 1]\), a shift \(\mu \in \{\leftarrow, \rightarrow\}\), encoded by \(\bar{\mu} \in \{0,1\}\) and get the binary complement of the (encoding of the) position reached from \(\bar{b}\) by applying \(\mu\). Here we distinguish two cases, depending on the choice of \(\mu\):

(a) If \(\mu = \rightarrow\) then we guess \(\text{bin}(i) = \bar{b}^\text{at} = (b_1, \ldots, b_n, 0, 1^{|\mathbb{B}| - n})\) for some \(n \in [0..|\mathbb{B}| - 1]\) and let \(c = \bar{c}^\text{at} = (\bar{b}_1, \ldots, \bar{b}_n, 0, 1^{|\mathbb{B}| - n})\) be the complement of \(\text{bin}(i + 1) = (b_1, \ldots, b_n, 1, 0^{|\mathbb{B}| - n})\).

(b) Otherwise, \(\text{bin}(i) = \bar{b}^\text{at} = (b_1, \ldots, b_n, 1, 0^{|\mathbb{B}| - n})\) and let \(c = \bar{c}^\text{at} = (\bar{b}_1, \ldots, \bar{b}_n, 1, 0^{|\mathbb{B}| - n})\) be the complement of \(\text{bin}(i - 1) = (b_1, \ldots, b_n, 0, 1^{|\mathbb{B}| - n})\).

For every \(n \in [0..|\mathbb{B}| - 1]\), \(m \in [0..|\mathbb{B}|]\) and \(i \in [1..m]\), we consider the following rules:

\[
\begin{align*}
  c_1(x) & \iff \exists b_1 \ldots \exists b_n \exists y \cdot x \rightarrow ([y]^{|\mathbb{B}|}) \ast d_1(y, b_1 \ldots b_n, 0, 1^{|\mathbb{B}| - n}, \bar{b}_1 \ldots \bar{b}_n, 0, 1^{|\mathbb{B}| - n - 1}) \\
  c_1(x) & \iff \exists b_1 \ldots \exists b_n \exists y \cdot x \rightarrow ([y]^{|\mathbb{B}|}) \ast d_1(y, 0, b_1 \ldots b_n, 1, 0^{|\mathbb{B}| - n - 1}, \bar{b}_1 \ldots \bar{b}_n, 1, 0^{|\mathbb{B}| - n - 1})
\end{align*}
\]
entailment checking in separation logic with inductive definitions is $\text{2-EXPTIME}$-hard

\[
d_1(x, u, b, c) \iff \exists y_1 \ldots \exists y_m \cdot x \mapsto (\bullet^0, y_1, \ldots, y_m) \ast \bigwedge_{j=1}^{\lceil \frac{m}{2} \rceil} \bar{d}_1(y_j, u, b, c) \tag{13}
\]

\[
d_1(x, u, b, c) \iff \exists y_1 \ldots \exists y_m \cdot x \mapsto (\bullet^0, y_1, \ldots, y_m) \ast \bigwedge_{j=1}^{\lceil \frac{m}{2} \rceil} \bar{d}_1(y_j, u, b, c) \tag{14}
\]

\[
\bar{d}_1(x, u, b, c) \iff \exists y \cdot x \mapsto (a, b, \bullet, y) \ast d_1(y, u, b, c), \text{ for each } a \in \Gamma, b \in \Gamma \setminus \{b\} \tag{15}
\]

\[
\bar{d}_1(x, u, b, c) \iff \exists y \cdot x \mapsto (a, b, \bullet, y) \ast d_1(y, u, b, c), \text{ for each } a \in \Gamma, b \in \Gamma \setminus \{b\} \tag{16}
\]

\[
f_1(x, b, c) \iff \exists y_1 \ldots \exists y_m \cdot x \mapsto (e, y_1, \ldots, y_m) \ast \bigwedge_{j=1}^{\lceil \frac{m}{2} \rceil} \bar{d}_1(y_j, y, \gamma) \tag{17}
\]

For a graphical depiction of the idea behind the encoding of violations of condition (I), we refer to Figure 4 (I). Intuitively, rules (11) and (12) choose the move $\mu \in \{\leftarrow, \rightarrow\}$ (encoded by 0 or 1) and the binary vectors $b, c \in \{0,1\}^n$, according to the cases (a) and (b) above, respectively. Note that we use the hat $[y]^0$ to eliminate the binary variables $b_1, \ldots, b_n$, as $n < 9\ell$, according to the elimination procedure described in §5.1. Then a path to the branching node, labeled $(q', \ell')$, that violates condition (I) is non-deterministically chosen, by alternating the branching and action nodes allocated by rules (13) and (15), respectively. The offending branching node is allocated by rule (17) and its predecessors are the branching and the action nodes, labeled with $(q, \ell)$ and $(a, b, \mu)$, such that $\ell' \neq \ell$. These latter nodes are allocated by rules (14) and (16), respectively.

The pseudo-derivations of $M$ that violate condition (II) are encoded by the tree-structured heaps defined by the rules below. To this end, we guess a binary vector $b \in \{0,1\}^n$ denoting the position of a write action that has an inconsistent read descendant and let $c$ be its binary complement. Then, for every $m \in \{0 \ldots \eta\}$ and $i \in \{1 \ldots m\}$, we consider the rules below (explanations will be provided afterward):

\[
c_2(x) \iff \exists b_1 \ldots \exists b_\eta \exists y \cdot x \mapsto \left(\left[ y \right]^0 \right) \ast d_2(y, b_1, \ldots, b_\eta, \bar{b}_1, \ldots, \bar{b}_\eta) \tag{18}
\]

\[
d_2(x, b, c) \iff \exists y_1 \ldots \exists y_m \cdot x \mapsto (\bullet^0, y_1, \ldots, y_m) \ast \bigwedge_{j=1}^{\lceil \frac{m}{2} \rceil} \bar{d}_2(y_j, y, \gamma, b, c) \tag{19}
\]

\[
\bar{d}_2(x, b, c) \iff \exists y \cdot x \mapsto (a, b, \bullet, y) \ast d_2(y, b, c), \text{ for each } a \in \Gamma, b \in \Gamma \setminus \{b\} \tag{20}
\]

\[
\bar{d}_2(x, b, c) \iff \exists y \cdot x \mapsto (a, b, \bullet, y) \ast d_2(y, b, c), \text{ for each } a \in \Gamma, b \in \Gamma \setminus \{b\} \tag{21}
\]

\[
e_2(x, y, b, c) \iff \exists y_1 \ldots \exists y_m \cdot x \mapsto (b, y_1, \ldots, y_m) \ast \bigwedge_{j=1}^{\lceil \frac{m}{2} \rceil} \bar{e}_2(y_j, y, \gamma, \gamma, b, c) \tag{22}
\]

\[
\bar{e}_2(x, y, b, c) \iff \exists y \cdot x \mapsto (a, b, \bullet, y) \ast \bigwedge_{j=1}^{\lceil \frac{m}{2} \rceil} \bar{e}_2(y_j, y, \gamma, \gamma, b, c) \tag{23}
\]

\[
\bar{e}_2(x, y, b, c) \iff \exists y \cdot x \mapsto (a, b, \bullet, y) \ast \bigwedge_{j=1}^{\lceil \frac{m}{2} \rceil} \bar{e}_2(y_j, y, \gamma, \gamma, b, c) \tag{24}
\]

\[
g_2(x, y, b, c) \iff \exists y_1 \ldots \exists y_m \cdot x \mapsto (e, y_1, \ldots, y_m) \ast \bigwedge_{j=1}^{\lceil \frac{m}{2} \rceil} \bar{g}_2(y_j, y, \gamma) \tag{25}
\]

\[
\bar{g}_2(x, y) \iff \exists y \cdot x \mapsto (\gamma, b, \bullet, y) \ast \bigwedge_{j=1}^{\lceil \frac{m}{2} \rceil} \bar{g}_2(y_j, y, \gamma) \tag{26}
\]

For a depiction of the idea behind the encoding of violations of condition (II), we refer to Figure 4 (II). Rule (18) uses the hat $[y]^0$ to choose the tuple of binary variables $b = (b_1, \ldots, b_\eta)$ and their complements $c = (\bar{b}_1, \ldots, \bar{b}_\eta)$. First, the path to a branching node labeled by the binary position $b$ is non-deterministically chosen by an alternation of branching and action nodes allocated by the rules (19) and (20), respectively, until the node and its predecessor are allocated by rules (22) and (21), respectively. We also guess a symbol $\gamma$, distinct from the symbol written on the tape at position $b$, and store it in the second parameter of $e_2(x, y, b, c)$. Next, a path to a second branching node labeled by the binary position $b$ is non-deterministically chosen by an alternation of branching and action nodes

13
allocated by the the rules (25) and (23) respectively, while checking that no branching node with the same position $b$ occurs on this second path (due to the side condition $e \neq \overline{e}$ of Rule (25) and the fact that $b = \overline{e}$). At the end, we reach the offending branching node (26), whose predecessor is allocated by rule (24). At this point, we check that the symbol read by the last action node is $\gamma$ (i.e. is different than the symbol previously written at position $b$, by rule (21)). This check is done by rules (26) and (27), ensuring that condition (II) is violated.

Next, we define the tree-structured heap encoding of the derivation trees that violate condition (III). To this end, we guess a binary vector $b \in \{0, 1\}^\infty$ denoting the position where a symbol different from $\overline{b}$ has been read, with no previous write action at that position and let $c$ be its complement. We consider the rules below, for every $m \in \llbracket 0..\beta \rrbracket$ and $i \in \llbracket 1..m \rrbracket$:

\[
c_3(x) \quad \iff \exists b_1 \ldots \exists b_\beta \exists y. \ x \mapsto ((y)^\beta) \ast d_3(y, b_1, \ldots, b_\beta, \overline{b}_1, \ldots, \overline{b}_\beta)
\]

\[
d_3(x, b, c) \quad \iff \exists y_1 \ldots \exists y_\beta \exists e. \ x \mapsto (e, y_1, \ldots, y_\beta) \ast \overline{\gamma}(y_j) \ast \overline{d}_3(y_j, b, c) \mid e \neq \overline{c}
\]

\[
\overline{d}_3(x, b, c) \quad \iff \exists y. \ x \mapsto (a, b, \bullet, y) \ast d_3(y, b, c), \text{ for all } a \in \Gamma, b \in \Gamma \setminus \{\overline{n}\}
\]

\[
\overline{e}_3(x, b, c) \quad \iff x \mapsto (a, b, \bullet, y) \ast \overline{e}_3(y, b, c), \text{ for all } a \in \Gamma, b \in \Gamma \setminus \{\overline{n}\}
\]

\[
\overline{f}_3(x) \quad \iff \exists y. \ x \mapsto (a, b, \bullet, y) \ast r(y), \text{ for all } a, b \in \Gamma \setminus \{\overline{n}\}
\]

After the initial guess of the binary position $b$, by rule (28), a path to a branching node labeled by $b$ is non-deterministically guessed, by an alternation of branching and action nodes corresponding to the rules (29) and (30), respectively, while checking that no branching node labeled with position $b$ occurs on this path. Once this node is reached, by rule (31), we check that its action node child reads a symbol different than $b$, by rules (32) and (33), which is in violation of condition (III).

Finally, the predicate $c_M(x)$ that chooses the condition (I), (II) or (III) to be violated, is defined by
the following rules:

\[ c_M(x) \iff \exists y \exists z . x \mapsto (y, z) \ast c_i(y) \ast \text{Const}(z), \text{ for all } i \in \{1, 2, 3\} \]  

(34)

Let \( S_M \) denote the set of rules introduced so far. The following lemma states the property of the models of \( c_M(x) \):

Lemma 16. Given a pseudo-derivation \( t \) of \( M \) and a structure \((s, h)\), such that \((s, h) \triangleright t\), we have \((s, h) \models_{S_M} c_M(x)\) if and only if \( t \) is not a derivation of \( M \).

Lemma 17. The entailment \( p_M(x) \models_{S_M} c_M(x) \) holds if and only if the membership problem \((M, \epsilon)\) has a negative answer.

Proof: “⇒” Suppose that \( M \) accepts \( \epsilon \). By Definition 9 there exists a derivation \( t \) starting from \( \epsilon \). Since \( t \) is a derivation, it is also a pseudo-derivation of \( M \) and, by Lemma 15 (A), there exists a structure \((s, h)\) such that \((s, h) \models_{S_M} p_M(x)\) and \((s, h) \triangleright t\). By Lemma 16, we obtain \((s, h) \not\models_{S_M} c_M(x)\), thus \( p_M(x) \not\models_{S_M} c_M(x) \).

“⇐” Suppose that \( p_M(x) \not\models_{S_M} c_M(x) \), hence there exists a structure \((s, h)\) such that \((s, h) \models_{S_M} p_M(x)\) and \((s, h) \not\models_{S_M} c_M(x)\). By Lemma 15 (B), there exists a pseudo-derivation \( t \) of \( M \) such that \((s, h) \triangleright t\). By Lemma 16, \( t \) is a derivation of \( M \), hence \((M, \epsilon)\) has a positive answer. □

We state the main result of this paper below:

Theorem 18. The entailment problem \( p(x) \models_{S} q(x) \), where \( S \) is a progressing, connected and established set of rules and \( p, q \) are predicate symbols in \( \text{Pred} \) that occur as heads in \( S \), is 2-\( \text{EXPTIME} \)-hard.

Proof: Given an exponential-space bounded ATM \( M \) we define a set of rules \( S_M \), based on the description of \( M \), such that \( p_M(x) \models_{S_M} c_M(x) \) if and only if \((M, \epsilon)\) has a negative answer (Lemma 17). Moreover, the set of rules is easy shown to be progressing, connected and established. The reduction is possible in time polynomial in the size of the standard encoding of \( M \). Indeed, the number of rules in \( S \) is \( O(||Q|| \cdot \text{B} \cdot \text{R}) \) and the succinct representation of each rule, using binary choices and binary variables can be generated in time \( O(||\Gamma|| \cdot \text{B} \cdot \text{R}) \). Finally, the complete elimination of binary variables is possible in polynomial time. Since we reduce from the complement of a \( \text{AEXPSPACE} \)-complete problem and \( \text{co-AEXPSPACE} = \text{AEXPSPACE} = 2-\text{EXPTIME} \), we obtain the 2-\( \text{EXPTIME} \)-hardness result. □

6 Conclusion

The entailment problem, for symbolic heaps with inductively defined predicates satisfying some additional conditions, was shown to be decidable (with elementary recursive time complexity) in [8]. We showed that this problem has an actual 2-\( \text{EXPTIME} \)-hard lower bound. In the light of the recent results of [11, 15, 13], this settles an open problem concerning the tight complexity of what is currently the most general decidable class of entailments for Separation Logic with inductive definitions. Note that the 2-\( \text{EXPTIME} \)-hardness proof relies only on entailments between atoms (more precisely they are of the form \( p(x) \models_{S} q(x) \)) and that inductive rules defining \( p \) and \( q \) contain no equational atom. Further, the constructed structures are actually quite restricted: they are directed acyclic graphs, with “almost” a tree shape, where only a polynomial number of children pointing to (\text{nil}, \text{nil}) are shared between nodes. Thus, 2-\( \text{EXPTIME} \)-hardness also holds for systems that are restricted to generate structures of this form. Note that the existence of these shared children makes the structures non local, in the sense of [9] (entailment is \( \text{EXPTIME} \)-complete for local structures). This draws a very precise boundary for the complexity of the entailment problem in the considered fragment of SL*, since it is known that the problem is \( \text{EXPTIME} \)-complete if the structures are trees (possibly enriched with backward links from children to parents) [9].

Concerning future work, we are now trying to extend the decidability and complexity results to a larger class of inductive definitions, by relaxing some of the conditions in Section 3.
Entailment Checking in Separation Logic with Inductive Definitions is 2-EXPTIME-hard

References


Entailment Checking in Separation Logic with Inductive Definitions is 2-EXPTIME-hard

Echenim, Iosif and Peltier

2020.