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

Mnacho Echenim, Radu Iosif, Nicolas Peltier

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:

$$p(x) \Leftarrow x \mapsto (\mathsf{nil},\mathsf{nil}) \qquad p(x) \Leftarrow \exists y_1, y_2 \,.\, x \mapsto (y_1,y_2) * p(y_1) * p(y_2)$$

$p(y)$ 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-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-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 $f : A \rightharpoonup B$, let $\mathrm{dom}(f) \stackrel{\text{def}}{=} \{x \in A \mid f(x) \text{ is defined}\}$ and $\mathrm{rng}(f) \stackrel{\text{def}}{=} \{f(x) \mid x \in \mathrm{dom}(f)\}$ be its domain and range, respectively, and we write $f : A \rightharpoonup_{fin} B$ if $\|\mathrm{dom}(f)\| < \infty$. Given integers n, m, we denote by $[\![n \,..\, m]\!]$ the set $\{n, n+1, \ldots, m\}$ (with $[\![n \,..\, m]\!] = \emptyset$ if $n > m$). By a slight abuse of notation, we write $t \in \boldsymbol{t}$ if $\boldsymbol{t} = (t_1, \ldots, t_n)$ and $t = t_i$, for some $i \in [\![1 \,..\, n]\!]$.

Let $\mathsf{Var} = \{x, y, \ldots\}$ be an infinite countable set of *variables* and $\mathsf{Pred} = \{p, q, \ldots\}$ 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 nil, of arity zero. A *term* is an element of the set $\mathsf{Term} \stackrel{\text{def}}{=} \mathsf{Var} \cup \{\mathsf{nil}\}$. Let $\kappa \geq 1$ be an integer constant fixed throughout this paper, intended to denote the number of record fields. The logic SL^κ is the set of formulæ generated inductively as follows:

$$\phi \quad := \quad t_0 \mapsto (t_1, \ldots, t_\kappa) \mid p(t_1, \ldots, t_{\#p}) \mid t_1 \approx t_2 \mid t_1 \not\approx t_2 \mid \phi_1 * \phi_2 \mid \exists x \,.\, \phi_1$$

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

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

- $\mathfrak{s} : \mathsf{Term} \to \mathsf{Loc}$ is a total mapping of terms into locations, called a *store*, such that $\mathfrak{s}(\mathsf{nil}) = nil$,
- $\mathfrak{h} : \mathsf{Loc} \rightharpoonup_{fin} \mathsf{Loc}^\kappa$ is a finite partial mapping of locations into κ-tuples of locations, called a *heap*, such that $nil \notin \mathrm{dom}(\mathfrak{h})$.

A location is *allocated* in a heap $\mathfrak{h}$ if it occurs in $\mathrm{dom}(\mathfrak{h})$. Two heaps $\mathfrak{h}_1$ and $\mathfrak{h}_2$ are *disjoint* iff $\mathrm{dom}(\mathfrak{h}_1) \cap \mathrm{dom}(\mathfrak{h}_2) = \emptyset$, in which case their *disjoint union* is denoted by $\mathfrak{h}_1 \uplus \mathfrak{h}_2$, undefined if $\mathrm{dom}(\mathfrak{h}_1) \cap \mathrm{dom}(\mathfrak{h}_2) \neq \emptyset$.

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

$$\begin{array}{lcll}
(\mathfrak{s}, \mathfrak{h}) & \models & t_1 \approx t_2 & \Leftrightarrow \quad \mathfrak{h} = \emptyset \text{ and } \mathfrak{s}(t_1) = \mathfrak{s}(t_2) \\
(\mathfrak{s}, \mathfrak{h}) & \models & t_1 \not\approx t_2 & \Leftrightarrow \quad \mathfrak{h} = \emptyset \text{ and } \mathfrak{s}(t_1) \neq \mathfrak{s}(t_2) \\
(\mathfrak{s}, \mathfrak{h}) & \models & t_0 \mapsto (t_1, \ldots, t_\kappa) & \Leftrightarrow \quad \mathrm{dom}(\mathfrak{h}) = \{\mathfrak{s}(t_0)\} \text{ and } \mathfrak{h}(\mathfrak{s}(t_0)) = (\mathfrak{s}(t_1), \ldots, \mathfrak{s}(t_\kappa)) \\
(\mathfrak{s}, \mathfrak{h}) & \models & \phi_1 * \phi_2 & \Leftrightarrow \quad \text{there are disjoint heaps } \mathfrak{h}_1 \text{ and } \mathfrak{h}_2, \text{ such that } \mathfrak{h} = \mathfrak{h}_1 \uplus \mathfrak{h}_2 \\
 & & & \quad\quad \text{and } (\mathfrak{s}, \mathfrak{h}_i) \models \phi_i, \text{ for each } i = 1, 2 \\
(\mathfrak{s}, \mathfrak{h}) & \models & \exists x \,.\, \phi & \Leftrightarrow \quad (\mathfrak{s}[x \leftarrow \ell], \mathfrak{h}) \models \phi, \text{ for some } \ell \in \mathsf{Loc},
\end{array}$$

where $\mathfrak{s}[x \leftarrow \ell]$ is the store mapping x into ℓ and behaving like $\mathfrak{s}$ for all $t \in \mathsf{Term} \setminus \{x\}$. Note that the semantics of $t_1 \approx t_2$ and $t_1 \not\approx t_2$ 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 $\mathcal{S}$ of *rules* of the form $p(x_1, \ldots, x_{\#p}) \Leftarrow \rho$, where ρ is an SL^κ formula such that $\mathsf{fv}(\rho) \subseteq \{x_1, \ldots, x_{\#_p}\}$. We refer to $p(x_1, \ldots, x_{\#p})$ as the *head*, and to ρ as the *body* of the rule. A rule is a *base rule* if its body is a predicate-free formula. We write $p(x_1, \ldots, x_{\#p}) \Leftarrow_{\mathcal{S}} \rho$ if the rule $p(x_1, \ldots, x_{\#p}) \Leftarrow \rho$ belongs to $\mathcal{S}$. In this section, we consider a given set of rules $\mathcal{S}$.

The above semantics is extended to formulæ 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 $\mathtt{nodes}(t)$ and a function mapping each node $w \in \mathtt{nodes}(t)$ to its *label*, denoted by $t(w)$. The set $\mathtt{nodes}(t)$ 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 wi are elements of $\mathtt{nodes}(t)$ for some $i \in \mathbb{N} \setminus \{0\}$, then so is wj for all $j \in [\![0 \,..\, i-1]\!]$. We write $|w|$ for the length of the sequence w and λ for the empty sequence, so that $|\lambda| = 0$. The *root* of t is λ, the *children* of a node $w \in \mathtt{nodes}(t)$ are the nodes $wi \in \mathtt{nodes}(t)$, where $i \in \mathbb{N}$, and the *parent* of a node wi with $i \in \mathbb{N}$ is w (hence, λ has no parent). The subtree of t rooted at w is denoted by $t{\downarrow_w}$; it is formally defined by $\mathtt{nodes}(t{\downarrow_w}) \stackrel{\text{def}}{=} \{w' \mid ww' \in \mathtt{nodes}(t)\}$ and $t{\downarrow_w}(w') \stackrel{\text{def}}{=} t(ww')$, for all $w' \in \mathtt{nodes}(t{\downarrow_w})$. 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 u, such that, for all $w \in \mathtt{nodes}(u)$, we have $u(w) = (q(s_1, \ldots, s_{\#q}), \psi)$, for a predicate atom $q(s_1, \ldots, s_{\#q})$ and a formula ψ, where:*

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

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

Given an unfolding tree $u \in \mathcal{T}_{\mathcal{S}}(p(t_1, \ldots, t_{\#p}))$ 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 ψ by replacing each occurrence of an atom $q(s_1, \ldots, s_{\#q})$ by $\Upsilon(u{\downarrow_i})$, where i denotes the child of w that $q(s_1, \ldots, s_{\#q})$ is mapped to[6] by the bijection of point (3) in Definition 1. More precisely, if $\psi = \exists y_1 \ldots \exists y_n \,.\, \varphi * \mathop{\ast}_{i=1}^{m} q_i(s_1^i, \ldots, s_{\#q_i}^i)$, where φ is predicate-free, then $\Upsilon(u) = \exists y_1 \ldots \exists y_n \,.\, \varphi * \mathop{\ast}_{i=1}^{m} \Upsilon(u{\downarrow_i})$.

Given an SL^κ formula ϕ and a structure $(\mathfrak{s}, \mathfrak{h})$, we write $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{S}} \phi$ if and only if $(\mathfrak{s}, \mathfrak{h}) \models \psi$ for some formula ψ is obtained from ϕ by syntactically replacing each occurrence of a predicate atom $p(t_1, \ldots, t_{\#p})$ in ϕ with a formula $\Upsilon(u)$, for some unfolding tree $u \in \mathcal{T}_{\mathcal{S}}(p(t_1, \ldots, t_{\#p}))$. A structure $(\mathfrak{s}, \mathfrak{h})$ such that $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{S}} \phi$ is called an $\mathcal{S}$-model of ϕ, or simply a model of ϕ, when $\mathcal{S}$ 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: $\phi \wedge x = y \Leftrightarrow \phi * x \approx y$, where $x = y$ iff x and y are assigned the same location.

[4] An unfolding tree for a generic SL^κ formula can be obtained by joining the unfolding trees of its predicate atoms under a common root.

[5] In particular, ψ 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.

Definition 2. *Given a set of rules $\mathcal{S}$ and two SL^κ formulæ ϕ and ψ, is it the case that every $\mathcal{S}$-model of ϕ is an $\mathcal{S}$-model of ψ? Instances of the entailment problem are denoted $\phi \models_{\mathcal{S}} \psi$.*

3 A Decidable Class of Entailments

In general, the entailment problem is undecidable [9, 1]. Thus we consider a subclass of entailments for which decidability (with elementary recursive complexity) was proved in [8] and provide a 2-EXPTIME lower bound for this problem. The decidable class is defined by three restrictions on the rules used for the interpretation of predicates, namely *progress*, *connectivity* and *establishment*, recalled next.

First, the *progress* condition requires that each rule adds to the heap exactly one location, namely the one associated with the first parameter of the head. Second, the *connectivity* condition requires that all locations added during an unfolding of a predicate atom $p(\boldsymbol{t})$ form a connected tree-like structure.

Definition 3. *A set of rules $\mathcal{S}$ is* progressing *if and only if the body ρ of each rule $p(x_1, \ldots, x_{\#p}) \Leftarrow_{\mathcal{S}} \rho$ is of the form $\exists z_1 \ldots \exists z_m \ . \ x_1 \mapsto (y_1, \ldots, y_\kappa) * \psi$ and ψ contains no occurrence of a points-to atom. If, moreover, each occurrence of a predicate atom in ψ is of the form $q(y_i, u_1, \ldots, u_{\#q-1})$, for some $i \in [\![1 \ldots \kappa]\!]$, then $\mathcal{S}$ 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 $\mathfrak{h}$ and a tree t, an* embedding of t into $\mathfrak{h}$ *is a bijection $\Lambda : \mathtt{nodes}(t) \rightarrow \mathrm{dom}(\mathfrak{h})$ such that $\Lambda(wi) \in \mathfrak{h}(\Lambda(w))$, for each node $wi \in \mathtt{nodes}(t)$, 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 $\mathcal{S}$ be a progressing and connected set of rules and $(\mathfrak{s}, \mathfrak{h})$ be a structure such that $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{S}} p(t_1, \ldots, t_{\#p})$. Then there exists an unfolding tree $u \in \mathcal{T}_{\mathcal{S}}(p(t_1, \ldots, t_{\#p}))$ such that $(\mathfrak{s}, \mathfrak{h}) \models \Upsilon(u)$, and an embedding of u into $\mathfrak{h}$.*

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 $(\mathfrak{s}, \mathfrak{h})$ such that $(\mathfrak{s}, \mathfrak{h}) \models_{\mathcal{S}} p(t_1, \ldots, t_{\#p})$ a *predicate decoration of $\mathfrak{h}$ w.r.t. $p(t_1, \ldots, t_{\#p})$* is a function $\Delta : \mathrm{dom}(\mathfrak{h}) \rightarrow \mathsf{Pred}$ defined as $\Delta(\ell) \stackrel{\text{def}}{=} q$ if and only if $u(\Lambda^{-1}(\ell)) = (q(s_1, \ldots, s_{\#q}), \psi)$, for some unfolding tree $u \in \mathcal{T}_{\mathcal{S}}(p(t_1, \ldots, t_{\#p}))$ such that $(\mathfrak{s}, \mathfrak{h}) \models \Upsilon(u)$ and some embedding Λ of u into $\mathfrak{h}$. Note that the unfolding tree u and function Λ 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 $\mathcal{S}$ is* established *if and only if, for each rule $p(x_1, \ldots, x_{\#p}) \Leftarrow_{\mathcal{S}} \exists z_1 \ldots \exists z_m \ . \ \psi$ and for each $\mathcal{S}$-model $(\mathfrak{s}, \mathfrak{h})$ of ψ, we have $\mathfrak{s}(z_1), \ldots, \mathfrak{s}(z_m) \in \mathrm{dom}(\mathfrak{h})$.*

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 $\mathcal{S}$ and two formulæ ϕ and ψ the problem $\phi \models_{\mathcal{S}} \psi$ belongs to ELEMENTARY.*

The rest of this paper is concerned with proving that the entailment problem $\phi \models_{\mathcal{S}} \psi$, for PCE sets of rules $\mathcal{S}$, 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*,*
- $\Gamma = \{\gamma_1, \ldots, \gamma_N, \text{B}\}$ *is a finite alphabet,* B *is the* blank *symbol,*
- $\delta \subseteq Q \times \Gamma \times Q \times (\Gamma \setminus \{\text{B}\}) \times \{\leftarrow, \rightarrow\}$ *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* $b \neq \text{B}$ *to the tape*[7] *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*
- $g : Q \rightarrow \{\vee, \wedge\}$ *partitions the set of states into* existential *(*$g(q) = \vee$*) and* universal *(*$g(q) = \wedge$*) 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, $w : \mathbb{N} \rightarrow \Gamma$ represents the contents of the *tape* and is such that $\|\{j \in \mathbb{N} \mid w(j) \neq \text{B}\}\| < \infty$, and $i \in [\![0 \,..\, \max\{j \in \mathbb{N} \mid w(j) \neq \text{B}\} + 1]\!]$ is the current position of the head on the tape. We denote by ϵ the empty word over Γ. 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 $i^{\leftarrow} \stackrel{\text{def}}{=} i - 1$ if $i > 0$ ($0^{\leftarrow}$ is undefined) and $i^{\rightarrow} \stackrel{\text{def}}{=} i + 1$. 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: $(q, w, i) \xrightarrow{(q,a,q',b,\mu)} (q', w', j)$ 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 $j = i^{\mu}$ is defined, i.e., either $i > 0$ or $\mu \neq \leftarrow$. We omit specifying the transition (q, a, q', b, μ) when it is not important. An *execution* is a sequence $(q_0, w_0, 0) \xrightarrow{(q_0,a_0,q_1,b_0,\mu_0)} (q_1, w_1, i_1) \xrightarrow{(q_1,a_1,q_2,b_1,\mu_1)} \ldots$ 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 $f : \mathbb{N} \rightarrow \mathbb{N}$, 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-space bounded, where $f(x) = c \cdot 2^{g(x)}$ 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* t*, 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* $(a, b, \mu) \in \Gamma \times \Gamma \setminus \{\text{B}\} \times \{\leftarrow, \rightarrow\}$*, where* a *is the symbol read,* b *is the symbol written and* μ *is the move of the head at that step,*

such that the root of t *is a branching node* $t(\lambda) = (q_0, w_0, 0)$ *and, moreover:*

a. *each branching node labeled by* (q, w, i) *such that* $g(q) = \vee$ *has exactly one child, which is an action node labeled by* (a, b, μ)*, 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* $(q, w, i) \xrightarrow{(q,a,q',b,\mu)} (q', w', j)$*;*

b. *each branching node labeled by* (q, w, i) *such that* $g(q) = \wedge$ *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, μ)*, the child of which is a branching node labeled by* (q', w', j)*, where* $(q, w, i) \xrightarrow{(q,a,q',b,\mu)} (q', w', j)$*.*

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 $g(p) = \wedge$ and there is no transition (q, a, q', b, μ) with $a = w(i)$.

[7] A machine never writes blank symbols, that are used only for the initially empty tape cells.

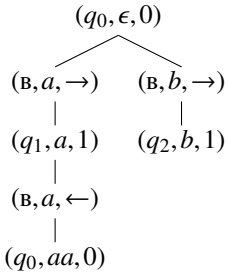

Figure 1 Derivation of ATM in Example 10

Example 10. *Consider an ATM* $M = (Q, \Gamma, \delta, q_0, g)$*, where:* $Q = \{q_0, q_1, q_2\}$, $\Gamma = \{a, b, c, \text{B}\}$, $\delta = \{(q_0, \text{B}, a, q_1, \rightarrow), (q_0, \text{B}, b, q_2, \rightarrow), (q_0, b, b, q_2, \rightarrow), (q_0, b, b, q_1, \rightarrow), (q_0, c, c, q_2, \rightarrow), (q_1, \text{B}, a, q_0, \leftarrow)\}$, $g(q_0) = q(q_2) = \wedge$ *and* $g(q_1) = \vee$*. A derivation for M, starting from an empty tape* ϵ*, 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* $w \in (\Gamma \setminus \{\text{B}\})^*$ *does M accept w ?*

The complexity class AEXPSPACE is the class of membership problems where M is exponential-space bounded. It is known that AEXPSPACE = co-AEXPSPACE = 2-EXPTIME [5], where co-AEXPSPACE is the complement class of AEXPSPACE[8].

In the following, we shall consider only the membership problem (M, ϵ). 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 ϵ. 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 $c \cdot 2^{g(|w|)}$ and accepts ϵ if and only if M accepts w. If $\mathfrak{N} \geq \log_2(c) + g(w)$, then $M_w; M$ runs in space $2^{\mathfrak{N}}$ and moreover, (M, w) and (M_w, ϵ) 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 $2^{\mathfrak{N}}$ on the empty input word, where $\mathfrak{N}$ 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 $p_M(x) \models_{\mathcal{S}_M} c_M(x)$, where $p_M(x)$, $c_M(x)$ are predicate atoms and $\mathcal{S}_M$ 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

[8] Every ATM can be complemented in linear time, by interchanging the existential with the universal states, thus all alternating classes are closed under complement.

violating the correctness of some tape contents or that of some position of the head. Consequently, $p_M(x) \models_{\mathcal{S}_M} c_M(x)$ holds if and only if M has no derivation on w. Since M is space bounded by $2^{\mathfrak{N}}$, where $\mathfrak{N}$ is bounded by a polynomial in the length of the input word w, we reduce from an arbitrary co-AEXPSPACE problem to the entailment problem for PCE sets of rules. Because co-AEXPSPACE = AEXPSPACE = 2-EXPTIME, we obtain the lower bound on the entailment problem for PCE sets of rules.

5.1 Syntactic Shorthands

Before giving the definitions of p_M, c_M and $\mathcal{S}_M$, 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 $\mathfrak{h}(\ell) = (\ell_1, \ell_2, \ell_3)$ to state that ℓ refers to a pair (ℓ_1, ℓ'_1) where ℓ'_1 itself refers to (ℓ_2, ℓ_3), and this additional location ℓ'_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 $\boldsymbol{t} = (t_1, \ldots, t_n)$ 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 ψ_i is t_i. By writing:

$$p(\boldsymbol{x}) \Leftarrow \exists y_1 \ldots \exists y_r \,.\, x_1 \mapsto (t_1, \ldots, t_n) * \psi$$

we denote the rules:

$$\begin{array}{rcl} p(\boldsymbol{x}) & \Leftarrow & \exists z_1 \exists y_1 \ldots \exists y_r \,.\, x_1 \mapsto (t_1, z_1) * \psi_1 * \widetilde{p}_1(z_1, \boldsymbol{x}, y_1, \ldots, y_r) \\ \widetilde{p}_i(z_i, \boldsymbol{x}, y_1, \ldots, y_r) & \Leftarrow & \exists z_{i+1} \,.\, z_i \mapsto (t_{i+1}, z_{i+1}) * \psi_{i+1} * \widetilde{p}_{i+1}(z_{i+1}, \boldsymbol{x}, y_1, \ldots, y_r), \text{ for } i \in [\![1 \,..\, n-2]\!] \\ \widetilde{p}_{n-2}(z_{n-1}, \boldsymbol{x}, y_1, \ldots, y_r) & \Leftarrow & z_{n-1} \mapsto (t_{n-1}, t_n) * \psi_n \end{array}$$

where $\widetilde{p}_1, \ldots, \widetilde{p}_{n-1}$ 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 ψ and every variable z_i is allocated by $\widetilde{p}_i$. In the following the term $(s^n, \boldsymbol{t})$ will be a shorthand for $(\underbrace{s, \ldots, s}_{n \text{ times}}, \boldsymbol{t})$ and $[\boldsymbol{t}]^n$ will stand for $(\mathsf{nil}^n, \boldsymbol{t})$. The interest of such special tuples will be explained later (essentially we will need to introduce "dummy" cells $(\mathsf{nil}, \ldots, \mathsf{nil})$ to ensure that all rules are progressing).

Global Variables We assume the existence of the following *global* variables that occur free in each formula: $\mathbf{0}, \mathbf{1}, \gamma_1, \ldots, \gamma_N$. The variables $\mathbf{0}$ and $\mathbf{1}$ denote binary digits, and the variable γ_i $(1 \leq i \leq N)$ denote non-blank symbols from the alphabet Γ[9]. These variables will always be assigned pairwise distinct allocated locations, as required by the following rules:

$$\mathsf{Const}(x) \Leftarrow x \mapsto (\mathbf{0}, \mathbf{1}, \gamma_1, \ldots, \gamma_N) * a(\mathbf{0}) * a(\mathbf{1}) * \circledast_{i=1}^{N} a(\gamma_i) \quad (1)$$

$$a(x) \Leftarrow x \mapsto (\mathsf{nil}, \mathsf{nil}) \quad (2)$$

[9] Since any membership problem is equivalent to a membership problem on a binary alphabet, via a binary encoding of Γ, having just $\mathbf{0}$ and $\mathbf{1}$ suffices. We consider distinct alphabet symbols $\gamma_1, \ldots, \gamma_N$ 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 $\mathbf{0}$ and $\mathbf{1}$. Thus any rule of the form:

$$p(x_1,\ldots,x_{\#p}) \Leftarrow \exists z_1 \ldots \exists z_n \,.\, x_1 \mapsto (\bullet, y) * \psi$$

stands for the following two rules:

$$\begin{array}{rcl} p(x_1,\ldots,x_{\#p}) & \Leftarrow & \exists z_1 \ldots \exists z_n \,.\, x_1 \mapsto (\mathbf{0}, y) * \psi \\ p(x_1,\ldots,x_{\#p}) & \Leftarrow & \exists z_1 \ldots \exists z_n \,.\, x_1 \mapsto (\mathbf{1}, y) * \psi \end{array}$$

and similarly for rules of the form $p(x_1,\ldots,x_{\#p}) \Leftarrow \exists z_1 \ldots \exists z_n \,.\, x_1 \mapsto (y, \bullet) * \psi$. 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 κ 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:*

$$\begin{array}{lll} p(x) & \Leftarrow \exists x_1 \,.\, x \mapsto (\mathbf{0}, x_1) * p_1(x_1) & p(x) \quad \Leftarrow \exists x_1 \,.\, x \mapsto (\mathbf{1}, x_1) * p_1(x_1) \\ p_1(x_1) & \Leftarrow x_1 \mapsto (\mathbf{0}, x_2) * p_2(x_2) & p_1(x_1) \Leftarrow x_1 \mapsto (\mathbf{1}, x_2) * p_2(x_2) \\ p_2(x_2) & \Leftarrow x_2 \mapsto (\mathbf{0}, \mathbf{0}) & p_2(x_2) \Leftarrow x_2 \mapsto (\mathbf{0}, \mathbf{1}) \\ p_2(x_2) & \Leftarrow x_2 \mapsto (\mathbf{1}, \mathbf{0}) & p_2(x_2) \Leftarrow x_2 \mapsto (\mathbf{1}, \mathbf{1}) \end{array}$$

We obtain $4 * 2 = 8$ *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 $\mathbf{0}$ and $\mathbf{1}$, namely the locations assigned to $\mathbf{0}$ and $\mathbf{1}$ by the formula 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 $\psi[\mathbf{0}/b, \mathbf{1}/\overline{b}] \vee \psi[\mathbf{1}/b, \mathbf{0}/\overline{b}]$. However, this direct substitution of the (existentially quantified) binary variables by $\mathbf{0}$ and $\mathbf{1}$ within the rules of an established system would break the establishment condition (Definition 6), because $\mathbf{0}$ and $\mathbf{1}$ are not necessarily allocated within the body of the rule[10]. This problem can be overcome by passing $\mathbf{0}$ and $\mathbf{1}$ as parameters to a fresh predicate. More precisely, a rule of the form (with $1 \leq i \leq m$):

$$p(x_1,\ldots,x_{\#p}) \Leftarrow \exists b_1 \ldots \exists b_i \exists y_1 \ldots \exists y_n \,.\, x_1 \mapsto [\boldsymbol{t}]^m * \psi \tag{3}$$

is a shorthand for the following set of rules:

$$\begin{array}{rcl} p(x_1,\ldots,x_{\#p}) & \Leftarrow & \exists y \,.\, x_1 \mapsto (\mathsf{nil}, y) * p'(y, x_1, \ldots, x_{\#p}, \mathbf{0}, \mathbf{1}) \\ p(x_1,\ldots,x_{\#p}) & \Leftarrow & \exists y \,.\, x_1 \mapsto (\mathsf{nil}, y) * p'(y, x_1, \ldots, x_{\#p}, \mathbf{1}, \mathbf{0}) \\ p'(y, x_1,\ldots,x_{\#p}, b_1, \overline{b}_1) & \Leftarrow & \exists b_2 \ldots \exists b_i \exists y_1 \ldots \exists y_n \,.\, y \mapsto [\boldsymbol{t}]^{m-1} * \psi \end{array}$$

Clearly, the elimination of the binary existential quantifiers from the rule (3) adds $2 \cdot i$ rules to the set. Note that the hat $[\boldsymbol{t}]^m$, 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 Const.

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 ψ. The rule (3) is equivalent to 2^i rules of the form $p(x_1,\ldots,x_{\#p}) \Leftarrow \exists y_1 \ldots \exists y_n \,.\, x_1 \mapsto [\boldsymbol{t}]^m * \psi$ where every b_j is replaced by $\mathbf{0}$ or $\mathbf{1}$ 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 $\boldsymbol{t}$ must be embedded into a tuple $[\boldsymbol{t}]^m$ of length at least i.

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

$$p(x_1,\ldots,x_{\#p}) \Leftarrow \exists c_1 \ldots \exists c_n \exists y_1 \ldots \exists y_m \,.\, x_1 \mapsto \boldsymbol{t} * \psi \mid (c_1,\ldots,c_n) \not\approx \overline{(b_1,\ldots,b_n)} \tag{4}$$

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

$$p(x_1,\ldots,x_{\#p}) \Leftarrow \exists y_1 \ldots \exists y_m \,.\, x_1 \mapsto (\boldsymbol{t}[b_i/c_i])[\bullet/c_j]_{j\in[\![1..n]\!]\setminus\{i\}} * \psi,\ i \in [\![1\,..\,n]\!] \tag{5}$$

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 $i \in [\![1\,..\,n]\!]$, and the other c_j for $j \neq i$ are arbitrary (hence they can be replaced by $\bullet$ since they occur only once in $\boldsymbol{t}$). 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 $\mathfrak{N}$, 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 ϵ, 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 $(q,i) \in Q \times [\![0\,..\,2^{\mathfrak{N}}-1]\!]$. 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* $(q,i) \in Q \times \mathbb{N}$*, or*
2. action nodes *labeled with tuples* $(a,b,\mu) \in \Gamma \times \Gamma \setminus \{\text{B}\} \times \{\leftarrow,\rightarrow\}$*, where a is the symbol read, b is the symbol written and* μ *is the move of the head at that step,*

such that the root of t is a branching node, $t(\lambda) = (q_0,0)$ *and, moreover:*

a. *each branching node labeled by* (q,i)*, such that* $g(q) = \vee$*, has exactly one child that is an action node labeled by* (a,b,μ)*, where* $(q,a,q',b,\mu) \in \delta$*, the child of which is a branching node labeled by* (q',j) *such that* $(q,a,b,q',\mu) \in \delta$ *and* $j \in \mathbb{N}$*;*
b. *each branching node labeled by* (q,i) *where* $g(q) = \wedge$ *has exactly one child for each tuple* $(q,a,q',b,\mu) \in \delta$*; this child is an action node labeled by* (a,b,μ)*, the child of which is a branching node labeled by* (q',j)*, 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 $j = i^{\mu}$). 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 $(q_0,\epsilon,0)$. 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 $(q',w[i \leftarrow b],i')$. 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 $a \neq w[i]$, and the latter node may be followed by a branching node (q',w',i') with $i' \neq i^{\mu}$. 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 $\underline{b}$ and $\underline{c}$ do not match the symbols read on the tape, and $\underline{0}$ 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.

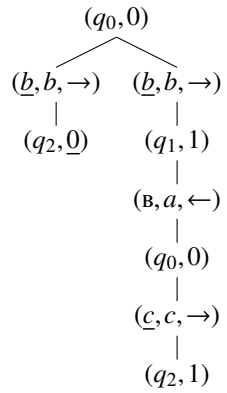

Figure 2 Pseudo-derivation of ATM in Example 10

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 $\overline{q}(x,a,b,\mu)$ 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 $i \in [\![0 \,..\, 2^{\mathfrak{N}} - 1]\!]$ on the tape succintly, by an $\mathfrak{N}$-tuple of binary digits $\text{bin}(i) \in \{\mathbf{0},\mathbf{1}\}^{\mathfrak{N}}$ and encode the left and right moves as $\overleftarrow{\ } \stackrel{\text{def}}{=} \mathbf{0}$ and $\overrightarrow{\ } \stackrel{\text{def}}{=} \mathbf{1}$. Let $\tau(q,a) \stackrel{\text{def}}{=} \delta \cap (\{q\} \times \{a\} \times Q \times \Gamma \setminus \{\text{B}\} \times \{\leftarrow,\rightarrow\})$ 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 $q \in Q$ and symbol $a \in \Gamma$:

$$q(x) \Leftarrow \exists x' \,.\, x \mapsto (\bullet^{\mathfrak{N}}, x') * \overline{q}'(x',a,b,\widetilde{\mu}) \quad \text{if } g(q) = \vee \text{ and } (q,a,q',b,\mu) \in \tau(q,a) \tag{6}$$

$$q(x) \Leftarrow \exists y_1 \ldots \exists y_n \,.\, x \mapsto (\bullet^{\mathfrak{N}}, y_1, \ldots, y_m) * \mathop{\circledast}_{j=1}^{m} \overline{q}_j(y_j,a,b_j,\widetilde{\mu}_j) \quad \text{if } g(q) = \wedge \text{ and } \tau(q,a) = \{(q,a,q_1,b_1,\mu_1),\ldots,(q,a,q_m,b_m,\mu_m)\} \tag{7}$$

$$\overline{q}(x,y,z,u) \Leftarrow \exists x' \,.\, x \mapsto (y,z,u,x') * q(x') \tag{8}$$

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 $\mathbf{0},\mathbf{1},\gamma_1,\ldots,\gamma_N$ and ensures that the initial state q_0 of M is the first control state that occurs on an path of a pseudo-derivation:

$$p_M(x) \Leftarrow \exists y \exists z \,.\, x \mapsto (y,z) * p'_M(y) * \mathsf{Const}(z) \tag{9}$$

$$p'_M(y) \Leftarrow \exists z' \,.\, y \mapsto [z']^{\mathfrak{N}} * q_0(z') \tag{10}$$

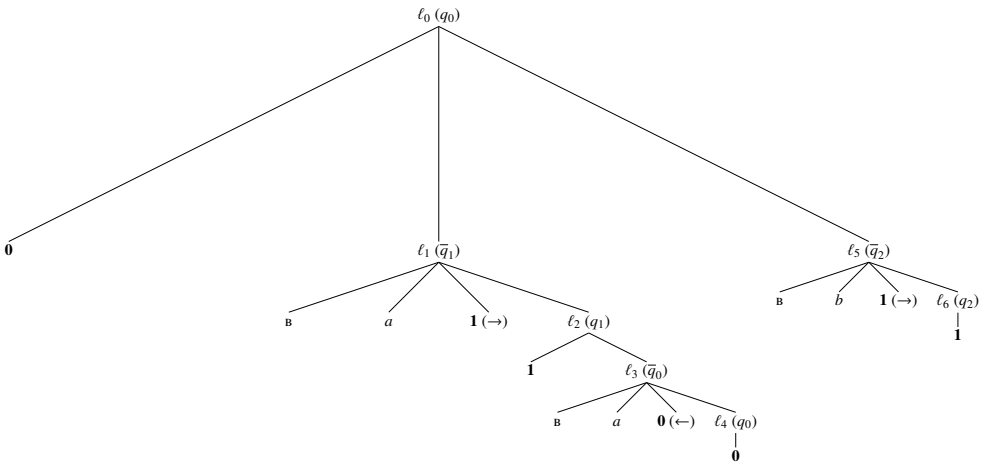

Figure 3 A heap encoding the derivation of Figure 1

The hat $[z']^{\mathfrak{N}}$ above ensures that every heap generated by p'_M begins with a tuple $[z']^{\mathfrak{N}} \stackrel{\text{def}}{=} (\overbrace{\mathsf{nil},\ldots,\mathsf{nil}}^{\mathfrak{N}}, z')$. The use of this tuple will be made clear in Section 5.3. For now, let $\mathcal{S}_M$ 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 $\overline{a}$ represents an action node. The definition below formalizes the encoding of a pseudo-derivation by a structure:

Definition 14. *A structure* $(\mathfrak{s},\mathfrak{h})$ *such that* $(\mathfrak{s},\mathfrak{h}) \models_{\mathcal{S}_M} p_M(x)$ encodes *a pseudo-derivation t of M, written as* $(\mathfrak{s},\mathfrak{h}) \rhd t$*, if and only if there exists a predicate decoration* Δ *of* $\mathfrak{h}$ *w.r.t.* $p_M(x)$*, two heaps* $\mathfrak{h}_1$ *and* $\mathfrak{h}_2$ *and a bijection* $f : \mathtt{nodes}(t) \rightarrow \mathrm{dom}(\mathfrak{h}_2)$ *such that, for all* $w \in \mathtt{nodes}(t)$*, the following hold:*

1. $\mathfrak{h} = \mathfrak{h}_1 \uplus \mathfrak{h}_2$,
2. $(\mathfrak{s},\mathfrak{h}_1) \models \exists y \exists z \exists z' \,.\, x \mapsto (y,z) * \mathsf{Const}(z) * y \mapsto [z']^{\mathfrak{N}}$,
3. *If w is a branching node with label $t(w) = (q,i)$ and children $w0,\ldots,wn$, then* $\Delta(f(w)) = q$ *and* $\mathfrak{h}_2(f(w)) = (\ell_1,\ldots,\ell_{\mathfrak{N}}, f(w0),\ldots,f(wn))$*, where* $\ell_j = \mathfrak{s}(\mathrm{bin}(i)_j)$*, for all* $j \in [\![1 \,..\, \mathfrak{N}]\!]$,
4. *If w is an action node with label $t(w) = (a,b,\mu)$ and only child $w0$, then we have* $\mathfrak{h}_2(f(w)) = (\mathfrak{s}(a), \mathfrak{s}(b), \mathfrak{s}(\widetilde{\mu}), f(w0))$.

A heap encoding the derivation of Figure 1 is depicted in Figure 3 (for readability, the part corresponding to the formula $\exists y \exists z \exists u \,.\, x \mapsto (y,z) * \mathsf{Const}(z) * y \mapsto [z']^{\mathfrak{N}}$ is not depicted, i.e., only the heap $\mathfrak{h}_2$ of Definition 14 is shown). We also give, for each location ℓ, the corresponding predicate $\Delta(\ell)$.

Lemma 15. *(A) For each pseudo-derivation t of M, there exists a structure* $(\mathfrak{s},\mathfrak{h}) \models_{\mathcal{S}_M} p_M(x)$ *such that* $(\mathfrak{s},\mathfrak{h}) \rhd t$*. (B) Dually, for each structure* $(\mathfrak{s},\mathfrak{h}) \models_{\mathcal{S}_M} p_M(x)$*, there exists a pseudo-derivation t of M such that* $(\mathfrak{s},\mathfrak{h}) \rhd t$*.*

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 $\underline{0}$ does not fulfill this condition).
II. For every $i \in [\![0 \,..\, 2^{\mathfrak{R}}-1]\!]$, if along a path from a branching node labeled (q,i) followed by an action node labeled (a,b,μ), to another branching node labeled (q',i) followed by an action node labeled (a',b',μ'), 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 $\underline{c}$ does not fulfill this condition).
III. For every $i \in [\![0 \,..\, 2^{\mathfrak{R}}-1]\!]$, if along a path from the root to a branching node labeled (q,i), followed by an action node labeled (a,b,μ), there is no branching node labeled (q',i), then necessarily $a = \text{B}$, i.e. the tape is initially empty (e.g., this condition is violated by the symbol $\underline{b}$ 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,ϵ) to an entailment problem, defined next. This does not change the final 2-EXPTIME-hardness result, because, as previously mentioned, 2-EXPTIME = AEXPSPACE = co-AEXPSPACE.

To this end, we consider a predicate c_M and a set of rules $\mathcal{S}_M$ containing rules for $p_M(x)$ and $c_M(x)$ such that the entailment $p_M(x) \models_{\mathcal{S}_M} c_M(x)$ 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 ϵ, admits no derivation.

Let $\mathfrak{B} \stackrel{\text{def}}{=} \max_{q\in Q, a\in\Gamma} \|\tau(q,a)\|$ 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:

$$\begin{array}{lcl} r(x) & \Leftarrow & \exists y_1 \ldots \exists y_n \,.\, x \mapsto (\bullet^{\mathfrak{R}}, y_1, \ldots, y_n) * \circledast_{j=1}^{n} \bar{r}(y_j), \text{ for each } n \in [\![0 \,..\, \mathfrak{B}]\!] \\ \bar{r}(x) & \Leftarrow & \exists y \,.\, x \mapsto (a,b,\bullet,y) * r(y), \text{ for each } a \in \Gamma \text{ and } b \in \Gamma \setminus \{\text{B}\} \end{array}$$

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

(a) If μ is $\rightarrow$ then we guess $\text{bin}(i) = \boldsymbol{b} \stackrel{\text{def}}{=} (b_1,\ldots,b_n,\mathbf{0},\mathbf{1}^{\mathfrak{R}-1-n})$ for some $n \in [\![0 \,..\, \mathfrak{R}-1]\!]$ and let $\boldsymbol{c} \stackrel{\text{def}}{=} (\overline{b}_1,\ldots,\overline{b}_n,\mathbf{0},\mathbf{1}^{\mathfrak{R}-1-n})$ be the complement of $\text{bin}(i+1) = (b_1,\ldots,b_n,\mathbf{1},\mathbf{0}^{\mathfrak{R}-1-n})$.
(b) Otherwise, $\text{bin}(i) = \boldsymbol{b} \stackrel{\text{def}}{=} (b_1,\ldots,b_n,\mathbf{1},\mathbf{0}^{\mathfrak{R}-1-n})$ and let $\boldsymbol{c} \stackrel{\text{def}}{=} (\overline{b}_1,\ldots,\overline{b}_n,\mathbf{1},\mathbf{0}^{\mathfrak{R}-1-n})$ be the complement of $\text{bin}(i-1) = (b_1,\ldots,b_n,\mathbf{0},\mathbf{1}^{\mathfrak{R}-1-n})$.

For every $n \in [\![0 \,..\, \mathfrak{R}-1]\!]$, $m \in [\![0 \,..\, \mathfrak{B}]\!]$ and $i \in [\![1 \,..\, m]\!]$, we consider the following rules:

$$c_1(x) \Leftarrow \exists b_1 \ldots \exists b_n \exists y \,.\, x \mapsto \left([y]^{\mathfrak{R}}\right) * d_1(y, \mathbf{1}, \underbrace{b_1 \ldots b_n, \mathbf{0}, \mathbf{1}^{\mathfrak{R}-n-1}}_{\boldsymbol{b}}, \underbrace{\overline{b}_1 \ldots \overline{b}_n, \mathbf{0}, \mathbf{1}^{\mathfrak{R}-n-1}}_{\boldsymbol{c}}) \quad (11)$$

$$c_1(x) \Leftarrow \exists b_1 \ldots \exists b_n \exists y \,.\, x \mapsto \left([y]^{\mathfrak{R}}\right) * d_1(y, \mathbf{0}, \underbrace{b_1 \ldots b_n, \mathbf{1}, \mathbf{0}^{\mathfrak{R}-n-1}}_{\boldsymbol{b}}, \underbrace{\overline{b}_1 \ldots \overline{b}_n, \mathbf{1}, \mathbf{0}^{\mathfrak{R}-n-1}}_{\boldsymbol{c}}) \quad (12)$$

$$d_1(x,u,\boldsymbol{b},\boldsymbol{c}) \Leftarrow \exists y_1 \ldots \exists y_m \,.\, x \mapsto (\bullet^{\mathfrak{R}}, y_1, \ldots, y_m) * \mathop{\circledast}_{j \in [\![1..m]\!] \setminus \{i\}} \overline{r}(y_j) * \overline{d}_1(y_i,u,\boldsymbol{b},\boldsymbol{c}) \quad (13)$$

$$d_1(x,u,\boldsymbol{b},\boldsymbol{c}) \Leftarrow \exists y_1 \ldots \exists y_m \,.\, x \mapsto (\boldsymbol{b}, y_1, \ldots, y_m) * \mathop{\circledast}_{j \in [\![1..m]\!] \setminus \{i\}} \overline{r}(y_j) * \overline{e}_1(y_i,u,\boldsymbol{b},\boldsymbol{c}) \quad (14)$$

$$\overline{d}_1(x,u,\boldsymbol{b},\boldsymbol{c}) \Leftarrow \exists y \,.\, x \mapsto (a,b,\bullet,y) * d_1(y,u,\boldsymbol{b},\boldsymbol{c}), \text{ for each } a \in \Gamma,\, b \in \Gamma \setminus \{\text{B}\} \quad (15)$$

$$\overline{e}_1(x,u,\boldsymbol{b},\boldsymbol{c}) \Leftarrow \exists y \,.\, x \mapsto (a,b,u,y) * f_1(y,\boldsymbol{b},\boldsymbol{c}), \text{ for each } a \in \Gamma,\, b \in \Gamma \setminus \{\text{B}\} \quad (16)$$

$$f_1(x,\boldsymbol{b},\boldsymbol{c}) \Leftarrow \exists y_1 \ldots \exists y_m \exists \boldsymbol{e} \,.\, x \mapsto (\boldsymbol{e}, y_1, \ldots, y_m) * \mathop{\circledast}_{j=1}^{m} \overline{r}(y_j) \mid \boldsymbol{e} \not\approx \overline{\boldsymbol{c}} \quad (17)$$

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 $\mathbf{0}$ or $\mathbf{1}$) and the binary vectors $\boldsymbol{b}, \boldsymbol{c} \in \{\mathbf{0},\mathbf{1}\}^{\mathfrak{R}}$, according to the cases (a) and (b) above, respectively. Note that we use the hat $[y]^{\mathfrak{R}}$ to eliminate the binary variables $b_1, \ldots, b_n$, as $n < \mathfrak{R}$, according to the elimination procedure described in §5.1. Then a path to the branching node, labeled (q', i'), 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,i) and (a,b,μ), such that $i' \neq i^{\mu}$. 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 $\boldsymbol{b} \in \{\mathbf{0},\mathbf{1}\}^{\mathfrak{R}}$ denoting the position of a write action that has an inconsistent read descendant and let $\boldsymbol{c}$ be its binary complement. Then, for every $m \in [\![0 \,..\, \mathfrak{B}]\!]$ and $i \in [\![1 \,..\, m]\!]$, we consider the rules below (explanations will be provided afterward):

$$c_2(x) \Leftarrow \exists b_1 \ldots \exists b_{\mathfrak{R}} \exists y \,.\, x \mapsto \left([y]^{\mathfrak{R}}\right) * d_2(y, \underbrace{b_1, \ldots, b_{\mathfrak{R}}}_{\boldsymbol{b}}, \underbrace{\overline{b}_1, \ldots, \overline{b}_{\mathfrak{R}}}_{\boldsymbol{c}}) \quad (18)$$

$$d_2(x,\boldsymbol{b},\boldsymbol{c}) \Leftarrow \exists y_1 \ldots \exists y_m \,.\, x \mapsto (\bullet^{\mathfrak{R}}, y_1, \ldots, y_m) * \mathop{\circledast}_{j \in [\![1..m]\!] \setminus \{i\}} \overline{r}(y_j) * \overline{d}_2(y_i,\boldsymbol{b},\boldsymbol{c}) \quad (19)$$

$$\overline{d}_2(x,\boldsymbol{b},\boldsymbol{c}) \Leftarrow \exists y \,.\, x \mapsto (a,b,\bullet,y) * d_2(y,\boldsymbol{b},\boldsymbol{c}), \text{ for each } a \in \Gamma,\, b \in \Gamma \setminus \{\text{B}\} \quad (20)$$

$$\overline{d}_2(x,\boldsymbol{b},\boldsymbol{c}) \Leftarrow \exists y \,.\, x \mapsto (a,b,\bullet,y) * e_2(y,\gamma,\boldsymbol{b},\boldsymbol{c}), \text{ for each } a \in \Gamma,\, \gamma \in \Gamma \setminus \{b\} \quad (21)$$

$$e_2(x,\gamma,\boldsymbol{b},\boldsymbol{c}) \Leftarrow \exists y_1 \ldots \exists y_m \,.\, x \mapsto (\boldsymbol{b}, y_1, \ldots, y_m) * \mathop{\circledast}_{j \in [\![1..m]\!] \setminus \{i\}} \overline{r}(y_j) * \overline{f}_2(y_i,\gamma,\boldsymbol{b},\boldsymbol{c}) \quad (22)$$

$$\overline{f}_2(x,\gamma,\boldsymbol{b},\boldsymbol{c}) \Leftarrow \exists y \,.\, x \mapsto (a,b,\bullet,y) * f_2(y,\gamma,\boldsymbol{b},\boldsymbol{c}), \text{ for each } a \in \Gamma,\, b \in \Gamma \setminus \{\text{B}\} \quad (23)$$

$$\overline{f}_2(x,\gamma,\boldsymbol{b},\boldsymbol{c}) \Leftarrow \exists y \,.\, x \mapsto (a,b,\bullet,y) * g_2(y,\gamma,\boldsymbol{b},\boldsymbol{c}), \text{ for each } a \in \Gamma,\, b \in \Gamma \setminus \{\text{B}\} \quad (24)$$

$$f_2(x,\gamma,\boldsymbol{b},\boldsymbol{c}) \Leftarrow \exists y_1 \ldots \exists y_m \exists \boldsymbol{e} \,.\, x \mapsto (\boldsymbol{e}, y_1, \ldots, y_m) * \mathop{\circledast}_{j \in [\![1..m]\!] \setminus \{i\}} \overline{r}(y_j) * \overline{f}_2(y_i,\gamma,\boldsymbol{b},\boldsymbol{c}) \mid \boldsymbol{e} \not\approx \overline{\boldsymbol{c}} \quad (25)$$

$$g_2(x,\gamma,\boldsymbol{b},\boldsymbol{c}) \Leftarrow \exists y_1 \ldots \exists y_m \,.\, x \mapsto (\boldsymbol{b}, y_1, \ldots, y_m) * \mathop{\circledast}_{j \in [\![1..m]\!] \setminus \{i\}} \overline{r}(y_j) * \overline{g}_2(y_i,\gamma) \quad (26)$$

$$\overline{g}_2(x,\gamma) \Leftarrow \exists y \,.\, x \mapsto (\gamma,b,\bullet,y) * r(y), \text{ for each } b \in \Gamma \setminus \{\text{B}\} \quad (27)$$

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 $[y]^{\mathfrak{R}}$ to choose the tuple of binary variables $\boldsymbol{b} = (b_1, \ldots, b_{\mathfrak{R}})$ and their complements $\boldsymbol{c} = (\overline{b}_1, \ldots, \overline{b}_{\mathfrak{R}})$. First, the path to a branching node labeled by the binary position $\boldsymbol{b}$ is non-deterministically chosen by an alternation of branching and action nodes allocated by the 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 γ, distinct from the symbol written on the tape at position $\boldsymbol{b}$, and store it in the second parameter of $e_2(x,\gamma,\boldsymbol{b},\boldsymbol{c})$. Next, a path to a second branching node labeled by the binary position $\boldsymbol{b}$ is non-deterministically chosen by an alternation of branching and action nodes

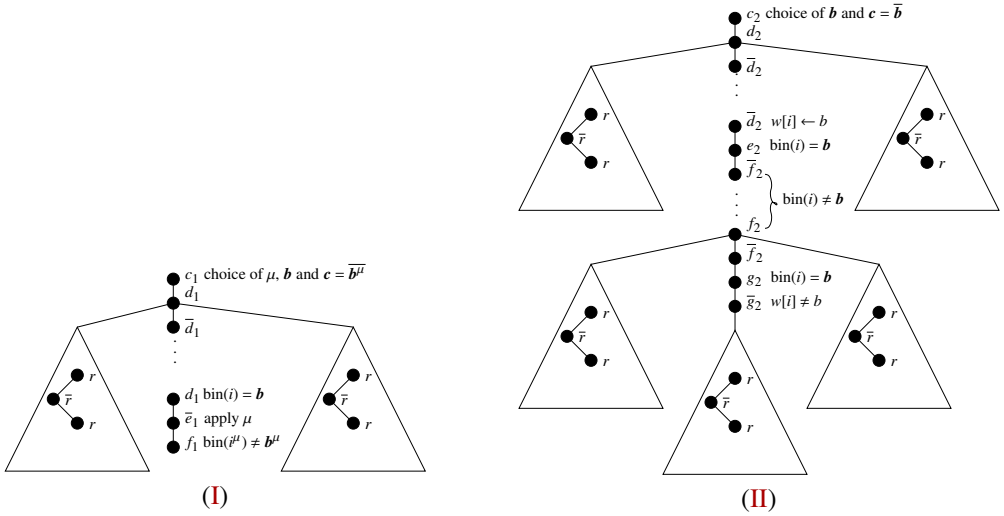

Figure 4 Pseudo-derivations violating conditions (I) and (II)

allocated by the the rules (25) and (23) respectively, while checking that no branching node with the same position $\boldsymbol{b}$ occurs on this second path (due to the side condition $\boldsymbol{e} \not\approx \overline{\boldsymbol{c}}$ of Rule (25) and the fact that $\boldsymbol{b} = \overline{\boldsymbol{c}}$) . 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 γ (i.e. is different than the symbol previously written at position $\boldsymbol{b}$, 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 $\boldsymbol{b} \in \{\boldsymbol{0}, \boldsymbol{1}\}^{\mathfrak{N}}$ denoting the position where a symbol different from B has been read, with no previous write action at that position and let $\boldsymbol{c}$ be its complement. We consider the rules below, for every $m \in [\![0 \,..\, \mathfrak{B}]\!]$ and $i \in [\![1 \,..\, m]\!]$:

$$c_3(x) \Leftarrow \exists b_1 \ldots \exists b_{\mathfrak{N}} \exists y \,.\, x \mapsto ([y]^{\mathfrak{N}}) * d_3(y, \underbrace{b_1, \ldots, b_{\mathfrak{N}}}_{\boldsymbol{b}}, \underbrace{\overline{b}_1, \ldots, \overline{b}_{\mathfrak{N}}}_{\boldsymbol{c}}) \tag{28}$$

$$d_3(x, \boldsymbol{b}, \boldsymbol{c}) \Leftarrow \exists y_1 \ldots \exists y_m \exists \boldsymbol{e} \,.\, x \mapsto (\boldsymbol{e}, y_1, \ldots, y_m) * \mathop{\Large *}_{j \in [\![1..m]\!] \setminus \{i\}} \overline{r}(y_j) * \overline{d}_3(y_i, \boldsymbol{b}, \boldsymbol{c}) \mid \boldsymbol{e} \not\approx \overline{\boldsymbol{c}} \tag{29}$$

$$\overline{d}_3(x, \boldsymbol{b}, \boldsymbol{c}) \Leftarrow \exists y \,.\, x \mapsto (a, b, \bullet, y) * d_3(y, \boldsymbol{b}, \boldsymbol{c}), \text{ for all } a \in \Gamma,\, b \in \Gamma \setminus \{\text{B}\} \tag{30}$$

$$\overline{d}_3(x, \boldsymbol{b}, \boldsymbol{c}) \Leftarrow x \mapsto (a, b, \bullet, y) * e_3(y, \boldsymbol{b}, \boldsymbol{c}), \text{ for all } a \in \Gamma,\, b \in \Gamma \setminus \{\text{B}\} \tag{31}$$

$$e_3(x, \boldsymbol{b}, \boldsymbol{c}) \Leftarrow \exists y_1 \ldots \exists y_m \,.\, x \mapsto (\boldsymbol{b}, y_1, \ldots, y_m) * \mathop{\Large *}_{j \in [\![1..m]\!] \setminus \{i\}} \overline{r}(y_j) * \overline{f}_3(y_i) \tag{32}$$

$$\overline{f}_3(x) \Leftarrow \exists y \,.\, x \mapsto (a, b, \bullet, y) * r(y), \text{ for all } a, b \in \Gamma \setminus \{\text{B}\} \tag{33}$$

After the initial guess of the binary position $\boldsymbol{b}$, by rule (28), a path to a branching node labeled by $\boldsymbol{b}$ 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 $\boldsymbol{b}$ 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:

$$c_M(x) \Leftarrow \exists y \exists z \,.\, x \mapsto (y,z) * c_i(y) * \mathsf{Const}(z), \text{ for all } i \in \{1,2,3\} \tag{34}$$

Let $\mathcal{S}_M$ 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 $(\mathfrak{s},\mathfrak{h})$, such that $(\mathfrak{s},\mathfrak{h}) \rhd t$, we have $(\mathfrak{s},\mathfrak{h}) \models_{\mathcal{S}_M} c_M(x)$ if and only if t is not a derivation of M.*

Lemma 17. *The entailment $p_M(x) \models_{\mathcal{S}_M} c_M(x)$ holds if and only if the membership problem (M,ϵ) has a negative answer.*

Proof: "⇒" Suppose that M accepts ϵ. By Definition 9 there exists a derivation t starting from ϵ. Since t is a derivation, it is also a pseudo-derivation of M and, by Lemma 15 (A), there exists a structure $(\mathfrak{s},\mathfrak{h})$ such that $(\mathfrak{s},\mathfrak{h}) \models_{\mathcal{S}_M} p_M(x)$ and $(\mathfrak{s},\mathfrak{h}) \rhd t$. By Lemma 16, we obtain $(\mathfrak{s},\mathfrak{h}) \not\models_{\mathcal{S}_M} c_M(x)$, thus $p_M(x) \not\models_{\mathcal{S}_M} c_M(x)$. "⇐" Suppose that $p_M(x) \not\models_{\mathcal{S}_M} c_M(x)$, hence there exists a structure $(\mathfrak{s},\mathfrak{h})$ such that $(\mathfrak{s},\mathfrak{h}) \models_{\mathcal{S}_M} p_M(x)$ and $(\mathfrak{s},\mathfrak{h}) \not\models_{\mathcal{S}_M} c_M(x)$. By Lemma 15 (B), there exists a pseudo-derivation t of M such that $(\mathfrak{s},\mathfrak{h}) \rhd t$. By Lemma 16, t is a derivation of M, hence (M,ϵ) has a positive answer. □

We state the main result of this paper below:

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

Proof: Given an exponential-space bounded ATM M we define a set of rules $\mathcal{S}_M$, based on the description of M, such that $p_M(x) \models_{\mathcal{S}_M} c_M(x)$ if and only if (M,ϵ) 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 $\mathcal{S}$ is $O(\|Q\| \cdot \mathfrak{R} \cdot \mathfrak{B})$ and the succint representation of each rule, using binary choices and binary variables can be generated in time $O(\|\Gamma\| \cdot \mathfrak{B} \cdot \mathfrak{R})$. Finally, the complete elimination of binary variables is possible in polynomial time. Since we reduce from the complement of a AEXPSPACE-complete problem and co-AEXPSPACE =AEXPSPACE =2-EXPTIME, we obtain the 2-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-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-EXPTIME-hardness proof relies only on entailments between atoms (more precisely they are of the form $p(\boldsymbol{x}) \models_{\mathcal{S}} q(\boldsymbol{x})$) 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 $(\mathsf{nil},\mathsf{nil})$ are shared between nodes. Thus, 2-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 EXPTIME-complete for local structures). This draws a very precise boundary for the complexity of the entailment problem in the considered fragment of SL^{κ}, since it is known that the problem is 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.

References

[1] Timos Antonopoulos, Nikos Gorogiannis, Christoph Haase, Max I. Kanovich, and Joël Ouaknine. Foundations for decision problems in separation logic with general inductive predicates. In Anca Muscholl, editor, *Foundations of Software Science and Computation Structures - 17th International Conference, FOSSACS 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014, Proceedings*, volume 8412 of *LNCS*, pages 411–425, 2014.

[2] Josh Berdine, Byron Cook, and Samin Ishtiaq. Slayer: Memory safety for systems-level code. In Ganesh Gopalakrishnan andShaz Qadeer, editor, *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*, volume 6806 of *LNCS*, pages 178–183. Springer, 2011.

[3] James Brotherston, Carsten Fuhs, Juan Antonio Navarro Pérez, and Nikos Gorogiannis. A decision procedure for satisfiability in separation logic with inductive predicates. In Thomas A. Henzinger and Dale Miller, editors, *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14 - 18, 2014*, pages 25:1–25:10. ACM, 2014.

[4] Cristiano Calcagno, Dino Distefano, Jérémy Dubreil, Dominik Gabi, Pieter Hooimeijer, Martino Luca, Peter W. O'Hearn, Irene Papakonstantinou, Jim Purbrick, and Dulma Rodriguez. Moving fast with software verification. In Klaus Havelund, Gerard J. Holzmann, and Rajeev Joshi, editors, *NASA Formal Methods - 7th International Symposium, NFM 2015, Pasadena, CA, USA, April 27-29, 2015, Proceedings*, volume 9058 of *LNCS*, pages 3–11. Springer, 2015.

[5] Ashok K. Chandra, Dexter Kozen, and Larry J. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, 1981. doi:10.1145/322234.322243.

[6] Kamil Dudka, Petr Peringer, and Tomás Vojnar. Predator: A practical tool for checking manipulation of dynamic data structures using separation logic. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*, volume 6806 of *LNCS*, pages 372–378. Springer, 2011.

[7] Mnacho Echenim, Radu Iosif, and Nicolas Peltier. Entailment checking in separation logic with inductive definitions is 2-exptime hard, 2020. arXiv:2004.07578.

[8] Radu Iosif, Adam Rogalewicz, and Jiri Simacek. The tree width of separation logic with recursive definitions. In *Proc. of CADE-24*, volume 7898 of *LNCS*, 2013.

[9] Radu Iosif, Adam Rogalewicz, and Tomás Vojnar. Deciding entailments in inductive separation logic with tree automata. In Franck Cassez and Jean-François Raskin, editors, *Automated Technology for Verification and Analysis - 12th International Symposium, ATVA 2014, Sydney, NSW, Australia, November 3-7, 2014, Proceedings*, volume 8837 of *LNCS*, pages 201–218. Springer, 2014.

[10] Christina Jansen, Jens Katelaan, Christoph Matheja, Thomas Noll, and Florian Zuleger. Unified reasoning about robustness properties of symbolic-heap separation logic. In Hongseok Yang, editor, *Programming Languages and Systems (ESOP'17)*, pages 611–638. Springer Berlin Heidelberg, 2017.

[11] Jens Katelaan, Christoph Matheja, and Florian Zuleger. Effective entailment checking for separation logic with inductive definitions. In Tomás Vojnar and Lijun Zhang, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 25th International Conference, TACAS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings, Part II*, volume 11428 of *LNCS*, pages 319–336. Springer, 2019.

[12] Peter W. O'Hearn and David J. Pym. The logic of bunched implications. *Bulletin of Symbolic Logic*, 5(2):215–244, 1999. doi:10.2307/421090.

[13] Jens Pagel, Christoph Matheja, and Florian Zuleger. Complete entailment checking for separation logic with inductive definitions. Technical report, 2020.

[14] J.C. Reynolds. Separation Logic: A Logic for Shared Mutable Data Structures. In *Proc. of LICS'02*, 2002.

[15] Florian Zuleger and Jens Katelaan. Extending the profile abstraction for complete entailment checking of symbolic heaps of bounded treewidth. In *Second workshop of Automated Deduction in Separation Logic*,

2020.