Abstraction Refinement for Trace Inclusion of Data Automata

Radu Iosif, Adam Rogalewicz, and Tomáš Vojnar

CNRS/Verimag, France and FIT BUT, Czech Republic
radu.iosif@imag.fr, {rogalew,vojnar}@fit.vutbr.cz

Abstract. A *data automaton* is a finite automaton equipped with variables (counters) ranging over a multi-sorted data domain. The transitions of the automaton are controlled by first-order formulae, encoding guards and updates. We observe, in addition to the finite alphabet of actions, the values taken by the counters along a run of the automaton, and consider the data languages recognized by these automata.
The problem addressed in this paper is the inclusion between the data languages recognized by such automata. Since the problem is undecidable, we give an abstraction-refinement semi-algorithm, proved to be sound and complete, but whose termination is not guaranteed.
The novel feature of our technique is checking for inclusion, without attempting to complement one of the automata, i.e. working in the spirit of antichain-based non-deterministic inclusion checking for finite automata [1]. The method described here has various applications, ranging from logics of unbounded data structures, such as arrays or heaps, to the verification of real-time systems.

1 Introduction

Many verification problems can be formulated as inclusion between two languages recognized by finite automata A and B, i.e. $\mathcal{L}(A) \subseteq \mathcal{L}(B)$. Traditional automata-based model checking of finite-state systems [17] use A as a specification of the system and B to describe the correctness property. Then $\mathcal{L}(A) \subseteq \mathcal{L}(B)$ means that every execution trace of the system satisfies the required property, and a counterexample trace $\tau \in \mathcal{L}(A) \setminus \mathcal{L}(B)$ indicates a violation of this property. Systems using variables that range over infinite data domains would thus benefit from a trace inclusion procedure between data automata.

On the other hand, deductive verification techniques also boil down to deciding a number of entailments between formulae $\varphi \rightarrow \psi$, known as verification conditions. When φ and ψ can be translated to finite automata A_φ and A_ψ, using well-known logic-automata connections [14], deciding a verification condition reduces to checking the inclusion $\mathcal{L}(A_\varphi) \subseteq \mathcal{L}(A_\psi)$. However, when the entailment occurs in a logic using infinite data types, translation to data automata and inclusion of languages of such automata becomes tantamount to an effective entailment procedure.

The bottleneck of such inclusion checks is the complementation of the right-hand side automaton, which incurs an unavoidable worst-case exponential blowup. For data automata, this becomes even more difficult, as most classes (e.g. timed automata) are not closed under complementation.

In this paper we present a novel method for checking inclusion between the data languages of finite automata equipped with variables (counters) ranging over infinite data domains. Our technique combines non-deterministic inclusions checking [1] with interpolation-based abstraction refinement [15] in a semi-algorithm that is proved to be sound (modulo termination) and complete. Our inclusion checking semi-algorithm essentially enumerates (product-)states of the form $\langle q, P\rangle$, where q is a control state of A and P is a set of control states of B and builds paths, e.g. finite sequences of edges $\langle q_0, P_0\rangle \xrightarrow{\sigma_1} \langle q_1, P_1\rangle \xrightarrow{\sigma_2} \ldots \xrightarrow{\sigma_n} \langle q_n, P_n\rangle$, where $q_0 \xrightarrow{\sigma_1} q_1 \ldots \xrightarrow{\sigma_n} q_n$ is a control path of A and $P_0 \xrightarrow{\sigma_1} P_1 \ldots \xrightarrow{\sigma_n} P_n$ corresponds to the subset construction of B for the word $\sigma_1 \ldots \sigma_n$. The path is labeled with boolean assignments to *predicates* associated to the edges of A and B. A path is *accepting* (in the abstract sense) when the predicate-based over-approximation of the (data) traces of A along this path is not included in the under-approximation of the traces of B for the same path.

An accepting path can witness a real counterexample $\tau \in \mathcal{L}(A)\backslash\mathcal{L}(B)$, or can be the effect of a too coarse abstraction. The abstraction refinement is based on a new interpolation scheme, called *generalized interpolation*, that provides local (edge) interpolants for valid entailments between path formulae. These interpolants are used to eliminate the spurious counterexample from future searches.

In general, for predicate abstraction verification of safety properties, termination depends on finding inductive interpolants that prove the given assertion. In our case, divergence might occur even when the set of predicates does not change, and the procedure explores longer and longer paths, repeating path segments labeled with the same interpolants, over and over. The solution we adopted consists in defining a *subsumption* relation between paths, based on folding the repeating edges into a finite automaton, and checking inclusion between the regular languages of these finite automata.

We have evaluated our method on a number of small examples of inclusion between integer-valued data automata with linear arithmetic transition relations. Finally, we have considered two examples steming from real verification problems: (i) a verification condition of a program with arrays which is translated into an inclusion between integer-valued counter automata generated from formulae of array logics [3], and (ii) a verification problem of a real-time system, translated into an inclusion between real-valued counter automata obtained from timed automata modeling of a train crossing system [13] and a timed property.

To improve readability, all proofs of lemmas and theorems are given in Appendix 6.

Related Work The study of data languages and data automata [2, 7] traditionally focuses on decidability of such logics for simple data theories (typically infinite data domains with equality). Our approach considers an undecidable

problem, the inclusion of data languages of automata whose transition rules are controlled by generic first-order theories, such as $\langle \mathbb{Z}, +, \leqslant \rangle$ or $\langle \mathbb{R}, +, \leqslant \rangle$, and provides a solution based on a (potentially infinite) abstraction refinement loop.

Other approaches to proving program refinement (equivalence) use abstraction refinement for solving recursive systems of Horn clauses [8]. Such techniques however can be used to test inclusion (equivalence) between the input-output summary relations of programs, instead of the data languages thereof. This occurs because Horn clauses use query variables of finite arity, whereas the data words of unbounded length would require unbounded arity.

Concerning entailments between formulae defining data structures with unbounded data elements, our previous works [11, 10] also use integer-valued counter automata. Inclusion of data languages is avoided by negation of the right-hand side formulae followed by complex normalization of formulae that lead to blowup. Finally, the fragment of Separation Logic with data described in [16] uses proof-theoretic techniques based on partial unfoldings of recursively defined predicates and needs acceleration lemmas to ensure termination.

2 Preliminary Definitions

We denote by $\mathbb{Z}$ the set of integers, and by $\mathbb{N}$ the set of positive integers, including zero. For any $k, \ell \in \mathbb{N}$, $k \leqslant \ell$ we write $[k, \ell]$ for the set $\{k, \ldots, \ell\}$. By $\mathbb{R}$ we denote the set of real numbers. For a finite set S, we denote by $\|S\| \in \mathbb{N}$ its cardinality.

Given a set $\mathcal{D} = \{D_1, \ldots, D_m\}$ of data domains, the *multi-sorted first order theory* $\mathrm{Th}(\mathcal{D}) = \langle D_1, \ldots, D_m, f_1, \ldots, f_\ell \rangle$ is a set of syntactically correct first-order formulae with functions having multi-sorted signatures, i.e. $f_i : D_{j_{i,1}} \times \ldots \times D_{j_{i,k_i}} \rightarrow D_{j_{i,k_i+1}}$, for some $j_{i,1}, \ldots, j_{i,k_i+1} \in \{1, \ldots, m\}$. Examples of multi-sorted theories include sets, multisets, arrays, stacks, queues, etc.

Let $\mathbf{x} = \{x_1, \ldots, x_n\}$ be a set of variables. For any variable $x \in \mathbf{x}$, let $\mathit{type}(x) \in [1, m]$ denote its *type*, i.e. $\mathit{type}(x) = j$ indicates that x ranges over the data domain D_j. A variable x is said to be *free* in a formula ϕ, denoted as $\phi(x)$, if it does not occur under the scope of a quantifier.

A *valuation* $\nu : \mathbf{x} \rightarrow \bigcup_{i=1}^{m} D_i$ is an assignment of the variables in $\mathbf{x}$ with values, compatible with their type, i.e. $\nu(x_i) \in D_{\mathit{type}(x_i)}$, for all $i \in [1, n]$. For a formula $\phi(\mathbf{x})$, we denote by $\nu \models_{\mathrm{Th}(\mathcal{D})} \phi$ the fact that substituting each $x \in \mathbf{x}$ by $\nu(x)$ yields a formula equivalent to **true** in the theory $\mathrm{Th}(\mathcal{D})$. In this case ν is said to be a *model* of ϕ. We denote by $[\![\phi]\!] = \{\nu \mid \nu \models_{\mathrm{Th}(\mathcal{D})} \phi\}$ the set of models of ϕ. A formula ϕ is said to be *satisfiable* whenever $[\![\phi]\!] \neq \varnothing$.

For a formula $\phi(\mathbf{x}, \mathbf{x}')$, where $\mathbf{x}' = \{x' \mid x \in \mathbf{x}\}$, and two valuations $\nu, \nu' : \mathbf{x} \rightarrow \bigcup_{i=1}^{m} D_i$, we denote by $(\nu, \nu') \models_{\mathrm{Th}(\mathcal{D})} \phi$ the fact that the formula obtained from ϕ, by substituting each $x \in \mathbf{x}$ with $\nu(x)$, and each $x' \in \mathbf{x}'$ with $\nu'(x')$ is equivalent to **true** in the theory $\mathrm{Th}(\mathcal{D})$.

Data Automata (DA) *Data Automata* (DA) are extensions of standard nondeterministic finite automata with typed variables ranging over a set of data

domains $\mathcal{D}$, equipped with a first order theory $\mathrm{Th}(\mathcal{D})$. Formally, a DA is a tuple $A = \langle \Sigma, \mathcal{D}, \mathbf{x}, Q, I, F, \Delta \rangle$:

- Σ is a finite alphabet, and $\diamond \in \Sigma$ is a special padding symbol,
- $\mathcal{D} = \{D_1, \ldots, D_m\}$ is a set of *data domains*,
- $\mathbf{x} = \{x_1, \ldots, x_n\}$ is a set of typed variables,
- Q is a finite set of *states* and $I, F \subseteq Q$ are the *initial* and *final* states,
- Δ is a set of *rules* of the form $q \xrightarrow{\sigma, \phi(\mathbf{x}, \mathbf{x}')} q'$, where $\sigma \in \Sigma$ is an alphabet symbol and $\phi(\mathbf{x}, \mathbf{x}') \neq \mathbf{false}$ is a formula in the multi-sorted first-order theory $\mathrm{Th}(\mathcal{D})$. We assume w.l.o.g. that, for each $q, q' \in Q$ and $\sigma \in \Sigma$ there exists at most one rule $q \xrightarrow{\sigma, \phi} q'$ in A[1].

A *control path* of A is a finite sequence $\pi : q_0 \xrightarrow{\sigma_0, \phi_0} q_1 \xrightarrow{\sigma_1, \phi_1} \ldots q_{n-1} \xrightarrow{\sigma_{n-1}, \phi_{n-1}} q_n$, where $q_{i-1} \xrightarrow{\sigma_{i-1}, \phi_{i-1}} q_i$ is a rule in Δ, for each $i \in [1, n]$. A pair (q, ν), where $q \in Q$ is a state, and $\nu : \mathbf{x} \to \bigcup_{i=1}^{m} D_i$ is a valuation of the typed variables in $\mathbf{x}$, is called a *configuration* of A. A *trace* τ corresponding to the control path π above, is a sequence τ of the form $(\sigma_0, \nu_0), (\sigma_1, \nu_1), \ldots (\sigma_{n-1}, \nu_{n-1}), (\diamond, \nu_n)$, where $\nu_i : \mathbf{x} \to \bigcup_{i=1}^{m} D_i$ are multi-sorted valuations of the typed variables in $\mathbf{x}$, for all $i \in [0, n]$, and $(\nu_{i-1}, \nu_i) \models_{\mathrm{Th}(\mathcal{D})} \phi_i$, for all $i \in [1, n]$. One can also define a data automaton as a finite automaton reading words over the infinite alphabet $\Sigma \times \left(\bigcup_{i=1}^{m} D_i\right)^n$ [2].

The language of A from q, denoted $\mathcal{L}_q(A)$ is the set of traces corresponding to paths from q to a final state, and the language of A is $\mathcal{L}(A) = \bigcup_{q \in I_A} \mathcal{L}_q(A)$. The *trace inclusion problem* asks, given two DA A and B, whether $\mathcal{L}(A) \subseteq \mathcal{L}(B)$.

Trees Let $\mathbb{N}^*$ be the set of sequences of natural numbers. We denote by $\epsilon \in \mathbb{N}^*$ the empty sequence, by $|p|$ the length of a sequence $p \in \mathbb{N}^*$, and by $p.q$ the concatenation of two sequences $p, q \in \mathbb{N}^*$. In the following, we use $p, q, r, \ldots \in \mathbb{N}^*$ for sequences, and $i, j, k, \ldots \in \mathbb{N}$ for integers. We say that q is a *prefix* of p, denoted $q \preceq p$ if $p = q.r$, for some sequence $r \in \mathbb{N}^*$. A *prefix-closed* set $S \subseteq \mathbb{N}^*$ has the property that, for all $p \in S$, $q \preceq p$ implies $q \in S$.

Given a finite set of symbols Θ, a *tree* is a finite partial function $t : \mathbb{N}^* \rightharpoonup_{fin} \Theta$, whose domain, denoted $dom(t)$, is a finite prefix-closed subset of $\mathbb{N}^*$. For each position $p \in dom(t)$, an integer $i \in \mathbb{N}$ such that $p.i \in dom(t)$ is called a *child* of p. A tree is *deterministic* if every position $p \in dom(t)$ has at most one child.

The set of edges of a tree is defined as $edges(t) = \{(p, p.i) \mid p.i \in dom(t)\}$. We define the *height* of a tree t as $ht(t) = \max\{|p| \mid p \in dom(t)\}$, and denote by $\mathrm{Fr}(t) = \{p \in dom(t) \mid \forall i \in \mathbb{N} \,.\, p.i \notin dom(t)\}$ the set of *leaves* (frontier) of the tree t. The *yield* of a tree $\mathrm{Yd}(t) = \{t(p) \mid p \in \mathrm{Fr}(t)\}$ denotes the set of symbols occurring on leaves. If $\Omega \subseteq \Theta$, we define $t{\downarrow_\Omega}$ to be the restriction of t to the set $\{p \in dom(t) \mid \exists q \in \mathrm{Fr}(t) \,.\, t(q) \in \Omega \wedge p \preceq q\}$, i.e. the maximal subtree u of t, such that $\mathrm{Yd}(u) \subseteq \Omega$.

For two trees t and u, we denote $t \bullet u$ the tree obtained from t by appending a copy of u to each leaf $p \in \mathrm{Fr}(t)$, such that $t(p) = u(\epsilon)$. If $\mathcal{U} = \{u_1, \ldots, u_k\}$ is a

[1] Two or more rules $\left\{q \xrightarrow{\sigma, \phi_i} q'\right\}_{i \in I}$ can be replaced by a single rule $q \xrightarrow{\sigma, \bigvee_{i \in I} \phi_i} q'$.

set of trees such that $u_i(\epsilon) \neq u_j(\epsilon)$, for all $i, j \in [1, k]$, $i \neq j$, we write $t \bullet \mathcal{U}$ for $(\ldots (t \bullet u_1) \ldots \bullet u_k)$.

Given a tree t and a position $p \in dom(t)$, we consider the *deterministic restriction* $t_{|p}$, defined as the restriction of t to the set $dom(t_{|p}) = \{q \mid q \preceq p\}$. Let $\langle\!\langle t \rangle\!\rangle = \{t_{|p} \mid p \in \mathrm{Fr}(t)\}$ be the set of maximal deterministic restrictions of t.

3 Generalized Interpolants

If $\varphi(\mathbf{x})$ and $\psi(\mathbf{y})$ are formulae, we write $\varphi \rightarrow \psi$ for the universal formula $\forall \mathbf{x} \cup \mathbf{y} \,.\, \neg\varphi \vee \psi$. Such formulae are called *entailments* in the following. Given formulae $\varphi(\mathbf{x})$ and $\psi(\mathbf{y})$ such that $\mathbf{x} \cap \mathbf{y} \neq \varnothing$ and $\varphi \rightarrow \psi$, the *Craig Interpolation Lemma* [6] guarantees the existence of a formula $I(\mathbf{x} \cap \mathbf{y})$ such that $\varphi \rightarrow I$ and $I \rightarrow \psi$. This formula is called a Craig interpolant, in the following. We assume, from now on, that Craig Interpolation Lemma applies to any multi-sorted theory $\mathrm{Th}(\mathcal{D})$ under consideration in this paper, such that, e.g. the additive theories of integers $\langle \mathbb{Z}, +, \leqslant \rangle$ and reals $\langle \mathbb{R}, +, \leqslant \rangle$.

For a set of variables $\mathbf{x}$ and an integer $i \geqslant 0$, let $\mathbf{x}^i = \{x^i \mid x \in \mathbf{x}\}$ be a set of *stamped* variables. We write $\mathbf{x}^{0 \cdots i}$ for $\mathbf{x}^0 \cup \ldots \cup \mathbf{x}^i$, in the following. For simplicity, we denote by $\varphi(\mathbf{x}^i, \mathbf{x}^j)$ the formula $\varphi[\mathbf{x}^i/\mathbf{x}, \mathbf{x}^j/\mathbf{x}']$. We may chose to omit the set of free variables of a formula, when it is clear from the context.

A *labeled tree* is a pair $T = (t, \lambda)$, where t is a tree and $\lambda : edges(t) \rightharpoonup_{fin} \mathrm{Th}(\mathcal{D})$ is a function labeling each edge $(p, p.i)$ of t by a formula $\lambda^i_p(\mathbf{x}, \mathbf{x}')$. From now on, for a labeled tree $T = (t, \lambda)$, we will systematically write $\lambda^i_p(\mathbf{x}, \mathbf{x}')$ instead of $\lambda((p, p.i))(\mathbf{x}, \mathbf{x}')$, and $\lambda_p(\mathbf{x}^{0 \cdots |p|})$ for the conjunction of all edge labels along the sequence p, i.e. $\lambda_p = \bigwedge_{q.j \preceq p} \lambda^j_q(\mathbf{x}^{|q|}, \mathbf{x}^{|q|+1})$.

For a labeled tree $T = (t, \lambda)$, we write $dom(T)$ and $edges(T)$ for $dom(t)$ and $edges(t)$, respectively. A labeled tree $T = (t, \lambda)$ is said to be deterministic if t is deterministic, and we denote $\langle\!\langle T \rangle\!\rangle = \{(u, \lambda) \mid u \in \langle\!\langle t \rangle\!\rangle\}$. The *characteristic formula* of a labeled tree $T = (t, \lambda)$ is defined as $\Upsilon(T) = \bigvee_{p \in \mathrm{Fr}(T)} \lambda_p$, i.e. $\Upsilon(T) = \bigvee_{p \in \mathrm{Fr}(T)} \bigwedge_{q.j \preceq p} \lambda^j_q$. Observe that the set of free variables of $\Upsilon(T)$ is $\mathbf{x}^{0 \ldots ht(t)}$, for $T = (t, \lambda)$.

In the rest of this section we address entailment problems of the form $\Upsilon(A) \rightarrow \Upsilon(B)$, where $A = (t, \lambda)$ and $B = (u, \mu)$ are labeled trees, and, in particular, A is deterministic[2]. We show that the entailment $\Upsilon(A) \rightarrow \Upsilon(B)$ holds if and only if there exist two labeled trees $\mathcal{I} = (t, I)$ and $\mathcal{J} = (u, J)$, called *interpolants*, such that $\Upsilon(A) \rightarrow \Upsilon(\mathcal{I}) \rightarrow \Upsilon(\mathcal{J}) \rightarrow \Upsilon(B)$ hold (Thm. 1). This result is the main ingredient of the abstraction refinement procedure presented in the next section. We start by defining the building bricks of this interpolation calculus.

Definition 1. *Given a labeled tree $T = (t, \lambda)$, an* over-approximating interpolant (OI) *for T is a labeled tree $\mathcal{I} = (t, I)$ where, for all $(p, p.i) \in edges(t)$, we have $\lambda^i_p(\mathbf{x}, \mathbf{x}') \rightarrow I^i_p(\mathbf{x}, \mathbf{x}')$. Symmetrically, an* under-approximating interpolant

[2] A generalization of the main result by removing this condition is possible in principle, but has no application in the rest of this paper.

(UI) *for T is a labeled tree $\mathcal{I} = (t, I)$, where, for all $(p, p.i) \in edges(t)$, we have* $I_p^i(\mathbf{x}, \mathbf{x}') \rightarrow \lambda_p^i(\mathbf{x}, \mathbf{x}')$.

The next technical lemma is needed in the proof of Thm. 1.

Lemma 1. *Given any deterministic labeled tree T, and a formula φ, we have:*

1. *$\Upsilon(T) \rightarrow \varphi$ iff there exists a OI $\mathcal{I}$ for T such that $\Upsilon(\mathcal{I}) \rightarrow \varphi$.*
2. *$\varphi \rightarrow \Upsilon(T)$ iff there exists a UI $\mathcal{J}$ for T such that $\varphi \rightarrow \Upsilon(\mathcal{J})$.*

We formalize the entailment problem $\Upsilon(A) \rightarrow \Upsilon(B)$, for two labeled trees A and B, by asking for the existence of generalized interpolants, defined below.

Definition 2. *Let $A = (t_A, \lambda_A)$ and $B = (t_B, \lambda_B)$ be two labeled trees. A* generalized interpolant (GI) *for (A, B) is a pair of labeled trees $(\mathcal{I}, \mathcal{J})$, where $\mathcal{I} = (t_A, I)$ is a OI for A, $\mathcal{J} = (t_B, J)$ is a UI for B, and $\Upsilon(\mathcal{I}) \rightarrow \Upsilon(\mathcal{J})$.*

The following theorem gives the main result of this section.

Theorem 1. *Let $A = (t_A, \lambda_A)$ be a deterministic labeled tree, and $B = (t_B, \lambda_B)$ be a labeled tree. Then $\Upsilon(A) \rightarrow \Upsilon(B)$ iff there exists a GI for (A, B).*

We illustrate the construction of generalized interpolants by an example:

Example 1. Let $A = (t_A, \lambda)$ and $B = (t_B, \mu)$ be the two trees shown in Fig. 1, for which $\Upsilon(A) \rightarrow \Upsilon(B)$ holds. The generalized interpolants $\mathcal{I} = (t_A, I)$ and $\mathcal{J} = (t_B, J)$ are shown in thick boxes in Fig. 1.

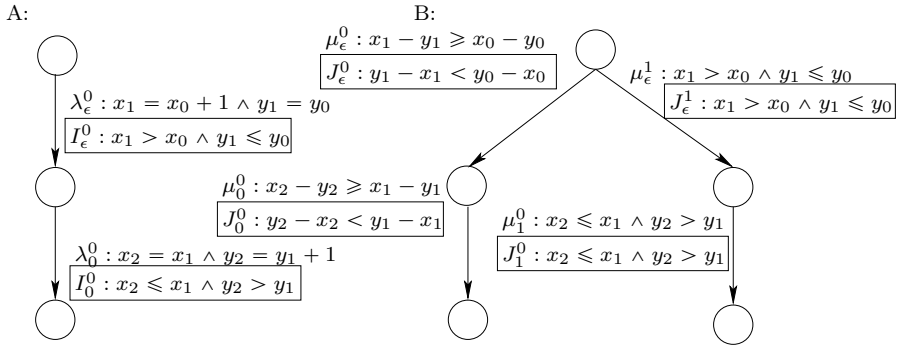

Fig. 1. The generalized interpolant for the trees A and B.

4 Trace Inclusion Algorithm

Let $A = \langle \Sigma, \mathcal{D}, \mathbf{x}, Q_A, I_A, F_A, \Delta_A \rangle$ and $B = \langle \Sigma, \mathcal{D}, \mathbf{x}, Q_B, I_B, F_B, \Delta_B \rangle$ be two DA working on the same alphabet Σ, same data domain $\mathcal{D}$ and same set of variables $\mathbf{x}$. We denote in the following $Q = Q_A \cup Q_B \cup \{\overline{q}, \overline{p}\}$, where $\overline{q}, \overline{p} \notin Q_A \cup Q_B$ are special designated states, and $\Delta = \Delta_A \cup \Delta_B$. We assume w.l.o.g. that $Q_A \cap Q_B = \emptyset$. A *product state* is a pair $s = \langle q, P \rangle$, where $q \in Q_A$ and $P \subseteq Q_B$. A *product edge* is defined as $\langle q, P \rangle \xrightarrow{\sigma} \langle q', P' \rangle$ if and only if $(q \xrightarrow{\sigma,\phi} q') \in \Delta_A$ and $P' = \left\{ p' \mid \exists p \in P \,.\, (p \xrightarrow{\sigma,\psi} p') \in \Delta_B \right\}$. A *product path* (called a *path* in the following, for simplicity) is an alternating sequence of product states and edges, starting and ending with product states. For a path $\rho : s_1 \xrightarrow{\sigma_1} \ldots \xrightarrow{\sigma_{n-1}} s_n$, we denote by $|\rho| = n - 1$ the number of edges on ρ, by $\rho_i = \left(s_i \xrightarrow{\sigma_i} s_{i+1} \right)$ the i-th edge on ρ, for all $i \in [1, |\rho|]$, and by $src(\rho) = s_1$ and $dest(\rho) = s_n$ its source and destination state, respectively.

Let us start by giving first an informal overview of the trace inclusion semi-algorithm 1. Since the trace inclusion problem is undecidable, termination of the semi-algorithm 1 is not guaranteed; in the following, we shall however call it an algorithm, for the sake of brevity. Algorithm 1 is a classical worklist iteration, that builds an *abstract product reachability tree* (APRT), whose positions are labeled with nodes of the form $\langle q, P, p \rangle$, where $q \in Q_A$, $P \subseteq Q_B$ and $p \in \mathbb{N}^*$. We add the tree position $p \in \mathbb{N}^*$ explicitly to the node in the APRT to distinguish nodes labeled with the same product states, and to handle sets of APRT nodes and edges, in the following.

Each path in the APRT starts with a node of the form $\langle i_\ell, I_B, \ell \rangle$, where $I_A = \{i_1, \ldots, i_K\}$ and $\ell \in [0, K-1]$, for some $K > 0$. For each node s of the APRT, the set `Edges` keeps the edges of the APRT, allowing to retrieve the unique path from a root node $\langle i, I_B, \ell \rangle$ to s (line 6). The processed nodes are kept in a global set `Visited`. The set `Next` keeps track of the frontier of the APRT, i.e. the newly generated nodes that have not been processed so far. Clearly, we always have that $\texttt{Visited} \cap \texttt{Next} = \emptyset$.

The *successor edges* of the APRT are generated using a *predicate map* Π that associates pairs of states $(q, q') \in Q \times Q$ with sets of formulae from the theory $\mathrm{Th}(\mathcal{D})$, called predicates. These formulae define the abstract state space in which the successor is computed; instead of computing the successors by applying concrete operations, we define a boolean assignment to the available predicates, in the spirit of predicate abstraction [12, 15]. The successor edges are kept in a set $\texttt{Edges} \subseteq \texttt{Visited} \times (\texttt{Visited} \cup \texttt{Next})$.

The convergence of the algorithm, and its overall efficiency depend on a *subsumption relation* $\sqsubseteq$ between paths in the APRT. Intuitively, $\rho \sqsubseteq \rho'$ if any continuation of ρ leading to a potential counterexample (i.e. a trace $\tau \in \mathcal{L}(A) \backslash \mathcal{L}(B)$), can be also be fired as a continuation of ρ'. The algorithm uses a set of *subsumption edges* $\texttt{Subsume} \subseteq \texttt{Visited} \times (\texttt{Visited} \cup \texttt{Next})$, such that $\texttt{Edges} \cap \texttt{Subsume} = \emptyset$, that keeps the destinations s and t of two paths ρ and τ, respectively, such that ρ can be extended in one step to a path subsumed by τ.

Algorithm 1 Trace Inclusion Algorithm

input: $A = \langle \Sigma, \mathcal{D}, \mathbf{x}, Q_A, \{i_1, \ldots, i_K\}, F_A, \Delta_A \rangle$ and $B = \langle \Sigma, \mathcal{D}, \mathbf{x}, Q_B, I_B, F_B, \Delta_B \rangle$
output: **true** if $\mathcal{L}(A) \subseteq \mathcal{L}(B)$, otherwise a counterexample trace $\tau \in \mathcal{L}(A) \backslash \mathcal{L}(B)$

1: **global** $\Pi \leftarrow \lambda\ (q, q')\ .\ \{\textbf{false}\}$
2: **global** `Subsume` $\leftarrow \emptyset$, `Next` $\leftarrow \{\langle i_\ell, I_B, \ell - 1 \rangle \mid \ell \in [1, K]\}$
3: **global** `Visited` $\leftarrow \{root\}$, `Edges` $\leftarrow \{(root, s) \mid s \in$ `Next`$\}$
4: **while** `Next` $\neq \emptyset$ **do**
5: chose $s \in$ `Next` and move s from `Next` to `Visited`
6: $\rho \leftarrow$ pathFromRoot(s)
7: **if** $Accept_\Pi(\rho)$ **then**
8: $(A_\rho, B_\rho) \leftarrow$ pathTrees(ρ)
9: $k \leftarrow$ findPivot(A_ρ, B_ρ)
10: **if** $k \geqslant 1$ **then**
11: $(\mathcal{I}, \mathcal{J}) \leftarrow$ generalizedInterpolant$(\Pi_{|k}(A_\rho), \Pi_{|k}(B_\rho))$
12: $\Pi \leftarrow$ updatePredicateMap$(\Pi, \mathcal{I}, \mathcal{J})$
13: **for** $(s, t) \in$ `Subsume` $\ .\ t \in$ subTree$(src(\rho_k))$ **do**
14: move s from `Visited` to `Next`
15: remove subTree$(src(\rho_k))$ from $\langle \Pi,$ `Visited`, `Next`, `Edges`, `Subsume`$\rangle$
16: add $src(\rho_k)$ to `Next`
17: **if** $k \neq 1$ **then**
18: add ρ_{k-1} to `Edges`
19: **else**
20: return extractCounterexample(ρ)
21: **else**
22: **match** $Post_\Pi(\rho)$ **with** $\{\tau_0, \ldots, \tau_m\}$
23: **for** $i \in [0, m]$ **do**
24: **local** `continue` $\leftarrow$ **true**
25: **if** $\exists u \in$ `Visited` $\cup$ `Next` $\ .\ \tau_i \sqsubseteq$ pathFromRoot(u) **then**
26: add (s, u) to `Subsume`
27: `continue` $\leftarrow$ **false**
28: **if** `continue` **then**
29: **local** `rem` $\leftarrow \{u \in$ `Next` $\mid$ pathFromRoot$(u) \sqsubset \tau_i\}$
30: **match** s **with** $\langle q, S, p \rangle$ and $dest(\tau_i)$ **with** $\langle q', S' \rangle$
31: **local** $t \leftarrow \langle q', S', p.i \rangle$
32: **for** $(s', u) \in$ `Edges` $\cup$ `Subsume` $\ .\ u \in$ `rem` **do**
33: add (s', t) to `Subsume`
34: remove `rem` from $\langle \Pi,$ `Visited`, `Next`, `Edges`, `Subsume`$\rangle$
35: add t to `Next`
36: add (s, t) to `Edges`
37: **return true**

An *accepting path* may witness a possible counterexample trace $\tau \in \mathcal{L}(A) \backslash \mathcal{L}(B)$. Since, however, we work in the abstract state space defined by the predicate map Π, acceptance of a path ρ is defined with respect to Π (line 7). If the test is negative, then no concrete counterexample can be found by analyzing ρ. If, one the other hand, the test is positive, two situations are possible: (i) the trace contains a concrete counterexample, or (ii) the acceptance was induced by a too coarse abstraction; in this case we say that ρ is *spurious*. In the first case, we report the counterexample (line 20), while in the second case we refine the abstraction. To this end, we compute the *pivot*, which is the rightmost position on the path that can prove its spuriousness (line 9). Once a pivot $k \geqslant 1$ is found, we compute a generalized interpolant for the labeled trees witnessing the actions of A and B alongside the product path ρ (line 8). This interpolant provides a new set of predicates that is added to the edges of A and B, in order to avoid analyzing the same trace in the future (line 12).

To reflect the changes in the APRT due to the refinement of Π, we need to recompute only the subtree of the APRT which is rooted at the node designated by the pivot. We remove this subtree from the APRT (line 15) and add the pivot node into `Next` for future processing (line 16).

In the case when the considered path ρ is not accepting (line 21), we add all successors (i.e. the destination states of the one-step continuations $\tau_0, \ldots, \tau_m$) of ρ into `Next` (lines 35 and 36), only if they are not subsumed by (the path to the root of) an existing node $u \in \texttt{Visited} \cup \texttt{Next}$. In this latter case, a subsumption edge (s, u) is added to `Subsumed` (line 26).

As an optimization, all nodes $u \in \texttt{Next}$, that are strictly subsumed by the destination of the currently considered successor τ_i (line 29), call this node t, are removed from the APRT (line 34), and each edge $(s', u) \in \texttt{Edges} \cup \texttt{Subsume}$ is replaced with a subsumption edge (s', t) (line 33).

4.1 Basic Definitions

This section gives the formal definitions of the main primitives of Algorithm 1. We define namely, the notions of path trees, predicate abstraction of paths, (concrete and abstract) feasible paths and successor relations.

Intuitively, a (product) path ρ is associated a pair of labeled trees (A_ρ, B_ρ), where A_ρ is deterministic, such that A_ρ and B_ρ capture all traces produced by A and B following the sequence of rules along ρ, respectively. Since we are looking for counterexamples $\tau \in \mathcal{L}(A) \backslash \mathcal{L}(B)$, we must consider all possibilities in which B responds to the actions taken by A. For this reason, B_ρ is usually a non-deterministic tree that accounts for all possible ways in which B can consume the traces produced by A.

Formally, a path $\rho : \langle q_1, P_1 \rangle \xrightarrow{\sigma_1} \cdots \xrightarrow{\sigma_{n-1}} \langle q_n, P_n \rangle$ corresponds to an inductively defined sequence of pairs of labeled trees $(A_0, B_0), (A_1, B_1), \ldots, (A_{n-1}, B_{n-1})$:

- $A_0 = (t, \lambda)$, where $dom(t) = \{\epsilon, 0\}$, $t(\epsilon) = \overline{q}$, $t(0) = q_1$, and $\lambda^0_\epsilon = \mathbf{true}$,
- $B_0 = (u, \mu)$, where $dom(u) = \{\epsilon\} \cup [0, k]$, $u(\epsilon) = \overline{p}$, $P_1 = \{u(0), \ldots, u(k)\}$, and $\mu^0_\epsilon = \ldots = \mu^k_\epsilon = \mathbf{true}$;

and, for all $i \in [1, n-1]$:

- $A_i = A_{i-1} \bullet (t, \lambda)$, where $dom(t) = \{\epsilon, 0\}$ and $\left(t(\epsilon) \xrightarrow{\sigma_i, \lambda^0_\epsilon} t(0) \right) \in \Delta_A$,
- $B_i = B_{i-1} \bullet \{(u_1, \mu_1), \ldots, (u_k, \mu_k)\}$, $P_i = \{u_1(\epsilon), \ldots, u_k(\epsilon)\}$, $P_{i+1} = \bigcup_{j=1}^{k} \mathrm{Yd}(u_j)$,

$$\forall j \in [1, k]\ \exists k_j \geqslant 0\ .\ dom(u_j) = \{\epsilon\} \cup [0, k_j] \wedge \\ \forall \ell\ .\ \ell \in [0, k_j] \Leftrightarrow \left(u_j(\epsilon) \xrightarrow{\sigma_i, (\mu_j)^\ell_\epsilon} u_j(\ell) \right) \in \Delta_B\ .$$

Observe that A_i is deterministic, for all $i \in [0, n-1]$. Since the final pair (A_{n-1}, B_{n-1}) is entirely determined by the path ρ, we shall write (A_ρ, B_ρ) for (A_{n-1}, B_{n-1}), and call A_ρ and B_ρ *path trees*, in the following.

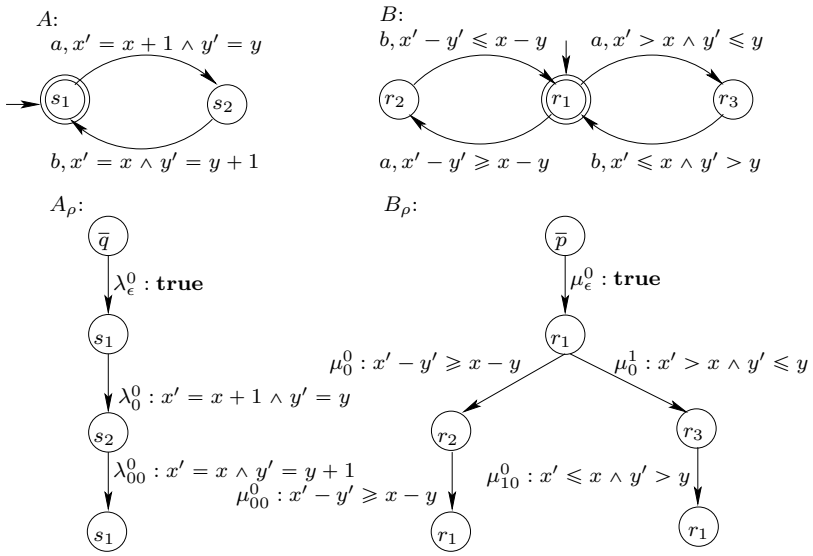

Fig. 2. Data word automata and path trees

Example 2. Fig. 2 depicts two data automata A and B, together with the path trees A_ρ and B_ρ for the path $\rho : \langle s_1, \{r_1\}\rangle \xrightarrow{a} \langle s_2, \{r_2, r_3\}\rangle \xrightarrow{b} \langle s_1, \{r_1\}\rangle$.

Let $\Pi : Q \times Q \to 2^{\mathrm{Th}(\mathcal{D})}$ be a mapping of pairs of control states into sets of formulae, called *predicates* in the following. We assume from now on that each predicate $\varphi \in \Pi(q, q')$ is a formula over the free variables $\mathbf{x} \cup \mathbf{x}'$, and moreover, that $\mathbf{false} \in \Pi(q, q')$, for all $q, q' \in Q$. If Π_1 and Π_2 are two predicate maps, we denote by $\Pi_1 \subseteq \Pi_2$ the fact that $\Pi_1(q, q') \subseteq \Pi_2(q, q')$, for all $q, q' \in Q$.

A predicate map Π induces a *coverage* of a pair of path trees (A_ρ, B_ρ), denoted by $(\Pi(A_\rho), \Pi(B_\rho))$, and defined next. Intuitively, $\Pi(A_\rho)$ is an over-approximation of A_ρ, i.e. it describes more traces of A than A_ρ, while $\Pi(B_\rho)$ is an under-approximation of B_ρ, i.e. it describes fewer traces of B than B_ρ. In other words, the entailments $\Upsilon(A_\rho) \to \Upsilon(\Pi(A_\rho))$ and $\Upsilon(\Pi(B_\rho)) \to \Upsilon(B_\rho)$ will hold. The reason behind this definition is that $\Upsilon(\Pi(A_\rho)) \to \Upsilon(\Pi(B_\rho))$ constitutes then a sufficient condition for the fact that no counterexample $\tau \in \mathcal{L}(A) \backslash \mathcal{L}(B)$ is captured by the particular path ρ.

Formally, considering the path trees $A_\rho = (t, \lambda)$ and $B_\rho = (u, \mu)$, defined above for the path ρ, we have $\Pi(A_\rho) = (t, \alpha)$ and $\Pi(B_\rho) = (u, \beta)$, where the edge labelings α and β are defined inductively as:

- for each $(p, p.i) \in \mathit{edges}(t_A)$, let $\alpha^0_\epsilon = \mathbf{true}$ and, assuming that α^j_q is already defined for each $(q, q.j) \in \mathit{edges}(t)$, such that $q.j \preceq p$, we have, further:

$$\alpha^i_p = \bigwedge \left\{ I \in \Pi\left(t(p), t(p.i)\right) \;\middle|\; \begin{array}{l} \alpha_p(\mathbf{x}^{0\cdots|p|}) \wedge \lambda^i_p(\mathbf{x}^{|p|}, \mathbf{x}^{|p|+1}) \to \\ \alpha_p(\mathbf{x}^{0\cdots|p|}) \wedge I(\mathbf{x}^{|p|}, \mathbf{x}^{|p|+1}) \end{array} \right\} ,$$

– for each $(p, p.i) \in edges(u)$, let $\beta^0_\epsilon = \mathbf{true}$ and, assuming that β^j_q is already defined for each $(q, q.j) \in edges(u)$, such that $q.j \preceq p$, we have, further:

$$\beta^i_p = \bigvee \left\{ J \in \Pi\left(u(p), u(p.i)\right) \;\middle|\; \begin{array}{l} \beta_p(\mathbf{x}^{0\ldots|p|}) \wedge J(\mathbf{x}^{|p|}, \mathbf{x}^{|p|+1}) \rightarrow \\ \beta_p(\mathbf{x}^{0\ldots|p|}) \wedge \mu^i_p(\mathbf{x}^{|p|}, \mathbf{x}^{|p|+1}) \end{array} \right\} .$$

Example 3. Let Π be the predicate map: $\Pi(s_1, s_2) = \Pi(r_1, r_3) = \{\mathbf{false}, x' > x \wedge y' \leqslant y'\}$, $\Pi(s_2, s_1) = \Pi(r_3, r_1) = \{\mathbf{false}, x' \leqslant x \wedge y' > y'\}$, $\Pi(r_1, r_2) = \{\mathbf{false}, x' - y' > x - y\}$, and $\Pi(r_2, r_1) = \{\mathbf{false}, x' - y' < x - y\}$. Fig. 3 shows the coverage of the path trees (A_ρ, B_ρ) from Fig. 2, induced by Π.

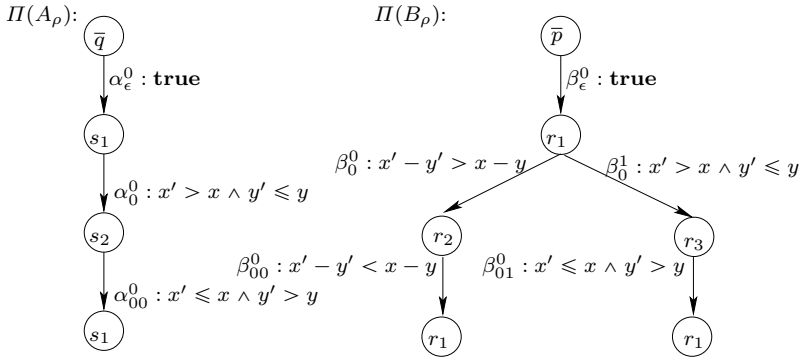

Fig. 3. Coverage of the path trees (A_ρ, B_ρ) from Fig. 2

A path ρ is said to be *feasible* iff the formula $\Upsilon(A_\rho)$ is satisfiable, and *Π-feasible* (for a given predicate map Π) iff the formula $\Upsilon(\Pi(A_\rho))$ is satisfiable, respectively. A path is said to be *infeasible* (*Π-infeasible*) if it is not feasible (Π-feasible). Observe that, for each predicate map Π, if ρ is Π-infeasible then any extension of it must be Π-infeasible as well. This is because, assuming that $\Pi(A_\rho) = (t, \alpha)$ as in the previous, we have $\mathbf{false} \in \Pi(t(p), t(p.i))$ for any $(p, p.i) \in edges(t_A)$, hence, according to the previous definition, we obtain that $\alpha^i_p = \mathbf{false}$ if $\alpha^j_q = \mathbf{false}$, for some $q.j \preceq p$.

For a path $\rho = \langle q_1, P_1\rangle \xrightarrow{\sigma_1} \cdots \xrightarrow{\sigma_{n-1}} \langle q_n, P_n\rangle$, and an alphabet symbol $\sigma \in \Sigma$, we define $Post(\rho, \sigma)$ to be the set of *feasible extensions* $\langle q_1, P_1\rangle \xrightarrow{\sigma_1} \cdots \xrightarrow{\sigma_{n-1}} \langle q_n, P_n\rangle \xrightarrow{\sigma} \langle q, P\rangle$ of ρ, and $Post_\Pi(\rho, \sigma)$ to be the set of *Π-feasible extensions* of ρ. We write $Post(\rho)$ for $\bigcup_{\sigma\in\Sigma} Post(\rho, \sigma)$ and $Post_\Pi(\rho)$ for $\bigcup_{\sigma\in\Sigma} Post_\Pi(\rho, \sigma)$.

The following lemma shows that restricting the predicate map by considering, for each $q, q' \in Q$, a subset of $\Pi(q, q')$, will enlarge the set of feasible extensions of a path. This is useful in proving the soundness of the trace inclusion algorithm.

Lemma 2. *For any two predicate maps $\Pi \supseteq \Pi'$, any path ρ, and any $\sigma \in \Sigma$, we have $Post_\Pi(\rho, \sigma) \subseteq Post_{\Pi'}(\rho, \sigma)$.*

4.2 Interpolant-based Refinement of Trace Abstractions

We now introduce the notions of acceptance and spuriousness, before describing how the generalized interpolants defined previously can be used to eliminate spurious counterexample paths. Intuitively, an accepting path is a symbolic encoding of counterexample traces $\tau \in \mathcal{L}(A) \backslash \mathcal{L}(B)$, for which: (i) A has a run over τ, starting in an initial state $i \in I_A$ and ending in a final state $q \in F_A$, and (ii) there is no run of B over τ, starting in an initial state $j \in I_B$ and ending in a final state $r \in F_B$.

Let us consider the path $\rho : s_1 \xrightarrow{\sigma_1} \cdots \xrightarrow{\sigma_n} s_{n+1}$. Formally, ρ is said to be *accepting*, denoted $Accept(\rho)$, if and only if:

1. $s_1 = \langle i, I_B \rangle$, where $i \in I_A$,
2. $\mathrm{Yd}(A_\rho) \subseteq F_A$, and
3. $cex(A_\rho, B_\rho)$ is satisfiable, where, for a pair of path trees (T, U), $cex(T, U)$ stands for $\Upsilon(T) \wedge \neg \Upsilon(U{\downarrow_{F_B}})$ and $U{\downarrow_{F_B}}$ is the maximal subtree of U whose leaves are labeled only by final states of B.

It is not very difficult to see that an accepting path constitutes a proof of the fact that $\mathcal{L}(A) \nsubseteq \mathcal{L}(B)$. In fact, any model of $cex(A_\rho, B_\rho)$ can be used to extract a counterexample $\tau \in \mathcal{L}(A) \backslash \mathcal{L}(B)$ for the trace inclusion.

Example 4. Fig. 4 shows the path $\rho : \langle s_1, \{r_1\} \rangle \xrightarrow{a} \langle s_2, \{r_2\} \rangle \xrightarrow{a} \langle s_2, \{r_2\} \rangle$ and the corresponding path trees. The formula:

$$cex(A_\rho, B_\rho) = (0 \leqslant x_1 \leqslant 1 \wedge x_2 = x_1) \wedge \neg(0 \leqslant x_1 \leqslant 1 \wedge x_2 = 1 - x_1)$$

is satisfiable, hence $Accept(\rho)$.

In the following, we shall work with an abstract version of acceptance, induced by the predicate map Π, and denoted $Accept_\Pi(\rho)$. The formal definition is the same as the one of $Accept(\rho)$, except for the third condition, asking that $cex(\Pi(A_\rho), \Pi(B_\rho))$ is satisfiable instead. A path ρ is said to be *spurious* (w.r.t. a given predicate map Π) if $Accept_\Pi(\rho) \wedge \neg Accept(\rho)$ holds. Basically, a path ρ which ends in a product state $\langle q, P \rangle$, with $q \in F_A$, is spurious if either: (i) $P \cap F_B = \varnothing$, i.e. no state in P is final, or (ii) no state from $P \cap F_B$ can be reached by B, following a trace produced by A along ρ, i.e. a model of $\Upsilon(A_\rho)$.

Example 5. Let us consider the path ρ, the path trees A_ρ and B_ρ from Fig. 2, and the covered trees $\Pi(A_\rho)$ and $\Pi(B_\rho)$ from Fig. 3. The path ρ is spurious, because $cex(\Pi(A_\rho), \Pi(B_\rho))$ is satisfiable, but $cex(A_\rho, B_\rho)$ is not.

If ρ is a spurious path, its spuriousness can be shown by replacing, for some $k \in [1, n]$, only the last $n - k$ abstract edges from (the path trees of) ρ with their concrete versions. Since we would like this number to be as small as possible, we need to consider the greatest such k. The *pivot* of ρ is the maximal position $k \in [1, n]$ that proves the spuriousness of ρ. Formally, if $A_\rho = (t, \lambda)$, $B_\rho = (u, \mu)$, $\Pi(A_\rho) = (t, \alpha)$ and $\Pi(B_\rho) = (u, \beta)$ are the labeled trees defined above, we define $\Pi_{|k}(A_\rho) = (t, \lambda_{|k})$ and $\Pi_{|k}(B_\rho) = (u, \mu_{|k})$, where:

$$\forall p.i \in dom(t) \,.\, (\lambda_{|k})^i_p = \begin{cases} \alpha^i_p \text{ if } |p| < k \\ \lambda^i_p \text{ if } |p| \geqslant k \end{cases} \quad \forall q.j \in dom(u) \,.\, (\mu_{|k})^j_q = \begin{cases} \beta^j_q \text{ if } |q| < k \\ \mu^j_q \text{ if } |q| \geqslant k \end{cases} .$$

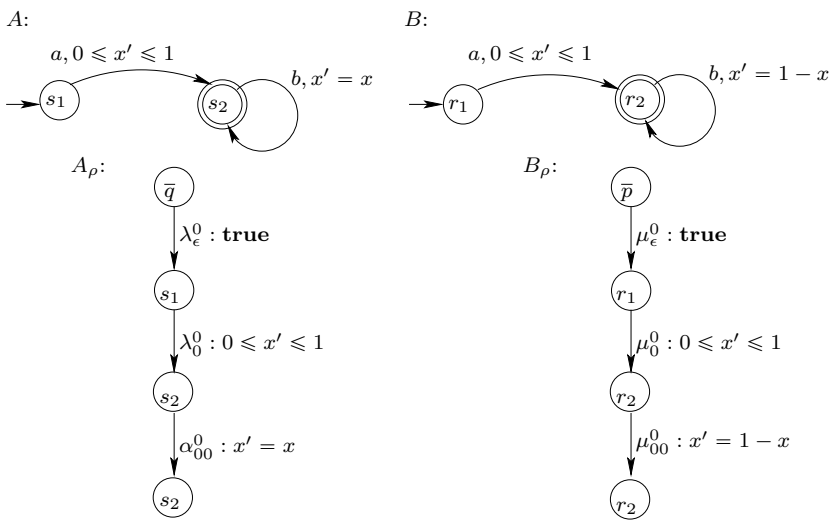

Fig. 4. Path trees for an accepting path, witnessing the counterexample traces $(a,0)(b,0)(b,0)$ and $(a,1)(b,1)(b,1)$.

We define $\mathit{pivot}(\rho) = \max\left\{k \in [1,n] \mid \neg \mathit{cex}(\Pi_{|k}(A_\rho), \Pi_{|k}(B_\rho))\right\}$, where, by convention, we let $\max \varnothing = -\infty$. Observe that $\mathit{pivot}(\rho) = -\infty$ if ρ is not spurious.

Notice that $\neg \mathit{cex}(\Pi_{|k}(A_\rho), \Pi_{|k}(B_\rho))$ means that the entailment $\Upsilon(\Pi_{|k}(A_\rho)) \rightarrow \Upsilon(\Pi_{|k}(B_\rho{\downarrow_{F_B}}))$ must hold. Then, by Thm. 1, there exists a generalized interpolant (GI) $(\mathcal{I}, \mathcal{J})$ for the pair $(\Pi_{|k}(A_\rho), \Pi_{|k}(B_\rho{\downarrow_{F_B}}))$, where $\mathcal{I} = (t, I)$ and $\mathcal{J} = (u{\downarrow_{F_B}}, J)$. We recall that $B_\rho{\downarrow_{F_B}}$ and $u{\downarrow_{F_B}}$ are the restrictions of B_ρ and u to the positions that are prefixes of leaves labeled with final states of B.

A predicate map Π is *compliant* with the GI $(\mathcal{I}, \mathcal{J})$ if and only if:

- for each $(p, p.i) \in \mathit{edges}(t)$, we have $I^i_p \in \Pi(t(p), t(p.i))$, and
- for each $(q, q.j) \in \mathit{edges}(u{\downarrow_{F_B}})$, we have $J^j_q \in \Pi(u(q), u(q,j))$.

where, as before, $\mathcal{I} = (t, I)$ and $\mathcal{J} = (u{\downarrow_{F_B}}, J)$. The next lemma shows that any spurious path has a GI that proves its spuriousness, and furthermore, updating to predicate map to become compliant with this GI guarantees that Algorithm 1 will never explore the same spurious path again in the future.

Lemma 3. *Given two predicate maps $\Pi, \Pi' : Q \times Q \rightarrow 2^{\mathit{Th}(\mathcal{D})}$ and a path $\rho = s1 \xrightarrow{\sigma_1} \cdots \xrightarrow{\sigma_n} s_{n+1}$ which is spurious w.r.t. Π, let $\mathit{pivot}(\rho) = k$. Then there exists a GI $(\mathcal{I}, \mathcal{J})$ for the pair $(\Pi_{|k}(A_\rho), \Pi_{|k}(B_\rho{\downarrow_{F_B}}))$. Moreover, for any predicate map Π', which is compliant with $(\mathcal{I}, \mathcal{J})$, we have $\neg \mathit{Accept}_{\Pi'}(\rho)$.*

4.3 Defining Subsumption for Faster Convergence

The following notion of *subsumption* between paths is crucial for the practical use of Algorithm 1. On one hand, a good definition of subsumption guarantees

convergence of the algorithm in a majority of non-trivial cases. On the other hand, even when the algorithm is bound to terminate (with subsumption trivially defined as identity of paths), more general definitions of subsumption will make the algorithm converge within significantly smaller numbers of steps.

Definition 3. *Given a predicate map Π, a partial order $\sqsubseteq_\Pi$ is said to be a* subsumption *if and only if the following hold, for all paths $\rho \sqsubseteq_\Pi \rho'$ and $\sigma \in \Sigma$:*

(a) $Accept_\Pi(\rho) \rightarrow Accept_\Pi(\rho')$,
(b) for each $\tau \in Post_\Pi(\rho, \sigma)$ there exists $\tau' \in Post_\Pi(\rho', \sigma)$ such that $\tau \sqsubseteq_\Pi \tau'$.

Observe that the subsumption relation $\sqsubseteq_\Pi$ depends, in general, on the predicate map under consideration. However, we will omit specifying the predicate map in the following, in order to avoid cluttering the notation.

A simple-minded implementation of subsumption may use the following definition: $\rho \leqq \rho'$ if and only if $\Upsilon(\Pi(A_\rho)) \rightarrow \Upsilon(\Pi(A_{\rho'}))$ and $\Upsilon(\Pi(B_\rho)) \leftarrow \Upsilon(\Pi(B_{\rho'}))$. It is not difficult to check that $\leqq$ is indeed a subsumption relation, according to Def. 3. However, Algorithm 1, running with this implementation of subsumption, will not terminate even in some simple case, as shown by the example below.

Example 6. Let Π be the predicate map: $\Pi(s_1, s_2) = \Pi(r_1, r_1) = \{\mathbf{false}, x' = y'\}$ and $\Pi(s_2, s_2) = \{\mathbf{false}, x - x' = y - y'\}$. Fig. 5 shows the Π-covered path trees corresponding to the diverging path $\rho : \langle s_1, \{r_1\}\rangle \xrightarrow{a} \langle s_2, \{r_1\}\rangle \xrightarrow{a} \langle s_2, \{r_1\}\rangle \xrightarrow{a} \langle s_2, \{r_1\}\rangle \ldots$

A more sophisticated definition of subsumption has to address the problem of divergence caused by multiple repetitions of the same subpath within a path. To this end, we view a path $\rho : s_1 \xrightarrow{\sigma_1} \cdots \xrightarrow{\sigma_n} s_{n+1}$ as the word $w(\rho) = (\sigma_1, s_1) \ldots (\sigma_n, s_n)(\diamond, s_{n+1}) \in \left(\Sigma \times (Q_A \times 2^{Q_B})\right)^*$, and we define the *folding* of ρ, for the predicate map Π, as the *finite automaton* $Fold_\Pi(\rho)$ recognizing all paths obtained from ρ by repeating certain subpaths of ρ arbitrarily many times.

A basic requirement on the folding operation is that, for any path ρ, we must have $w(\rho) \in \mathcal{L}(Fold_\Pi(\rho))$, i.e. by folding a path we do not lose it. Then subsumption is defined as: $\rho \Subset \rho'$ if and only if $\mathcal{L}(Fold_\Pi(\rho)) \subseteq \mathcal{L}(Fold_\Pi(\rho'))$, where $\mathcal{L}(Fold_\Pi(\rho))$ denotes the language recognized by the automaton $Fold_\Pi(\rho)$.

In order to ensure that the relation defined above is indeed a subsumption relation, one needs, moreover, to provide sufficient conditions that guarantee the two points of Def. 3 above. A sufficient condition for point (a) of Def. 3 is that no accepting paths (w.r.t. the given predicate map Π) are introduced by the folding operation. Formally, for every path ρ and every predicate map Π:

$$\neg Accept_\Pi(\rho) \Rightarrow [\forall \tau \,.\, w(\tau) \in \mathcal{L}(Fold_\Pi(\rho)) \Rightarrow \neg Accept_\Pi(\tau)] \tag{1}$$

It is easy to check that, whenever condition (1) is satisfied, we deduce:

$$\begin{aligned} Accept_\Pi(\rho) &\Rightarrow \exists \tau \,.\, w(\tau) \in \mathcal{L}(Fold_\Pi(\rho)) \wedge Accept_\Pi(\tau) \\ \text{since } \mathcal{L}(Fold_\Pi(\rho)) \subseteq \mathcal{L}(Fold_{\Pi'}(\rho')) &\Rightarrow \exists \tau \,.\, w(\tau) \in \mathcal{L}(Fold_\Pi(\rho')) \wedge Accept_\Pi(\tau) \\ \text{by (1)} &\Rightarrow Accept_\Pi(\rho') \ . \end{aligned}$$

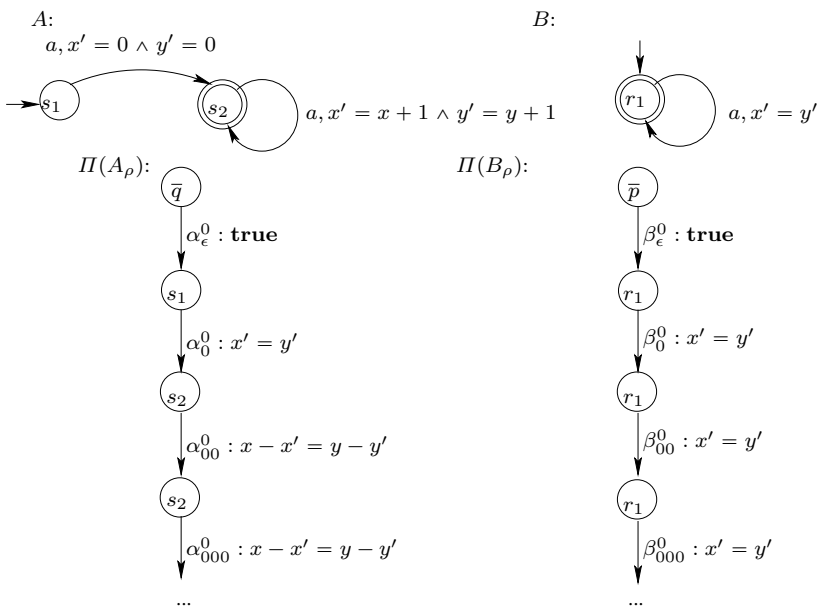

Fig. 5. Diverging path trees

Algorithm 2 implements the folding operation, needed for the subsumption test described above. First, the algorithm sweeps the input path ρ in order to find matching pairs of states, that become candidates for the cycles of the automaton. The sweeping is implemented by the SIMPLECYCLES function which will always detect only the innermost cycles, among all possibilities. The result is a list of pairs, each of which marks the beginning and the end of a cyclic subpath of ρ, i.e. that starts and ends with the same product state (we always consider only simple cycles that do not repeat any state, except for the endpoints). In the following, for any $i, j \in [1, n]$, $i \leqslant j$, we write ρ_i for the edge $s_i \xrightarrow{\sigma_i} s_{i+1}$, and $\rho_{i,j}$ for the subpath $s_i \xrightarrow{\sigma_i} \ldots s_j \xrightarrow{\sigma_j} s_{j+1}$ of ρ.

Example 7. Let $\rho : s_1 \xrightarrow{\sigma_1} s_2 \xrightarrow{\sigma_2} s_2 \xrightarrow{\sigma_3} s_3 \xrightarrow{\sigma_4} s_3 \xrightarrow{\sigma_5} s_2 \xrightarrow{\sigma_6} s_1$ be a path. The result of SIMPLECYCLES(ρ) is the list of pairs $(2, 3)(4, 5)$ indicating the positions of the sources and destinations of the innermost loops on ρ, respectively.

The core of the folding procedure described by Algorithm 2 is the loop between lines (5-22), which traverses the list of simple cycle candidates generated by SIMPLECYCLES, in order to identify the paths that can be folded into loops, without breaking condition (1) above. This test takes into account the loop candidate, the predicate map Π, as well as the previously folded loops and linear path segments, kept in two lists, `pastCycles` and `pastPaths`, respectively. The need for these lists will be made clear next, when we explain the implementation of the ISFOLDABLE test (line 14).

Example 8. (contd. from Ex. 7) Assume that both loop candidates from Ex. 7, namely $s_2 \xrightarrow{\sigma_2} s_2$ and $s_3 \xrightarrow{\sigma_4} s_3$ are foldable in the context of the path ρ. Then the result of Algorithm 2 is the finite automaton $\mathit{Fold}_{\Pi}(\rho)$ depicted below:

$$\langle s_1, 1\rangle \xrightarrow{\sigma_1} \overset{\sigma_2}{\curvearrowright}\langle s_2, 2\rangle \xrightarrow{\sigma_3} \overset{\sigma_4}{\curvearrowright}\langle s_3, 3\rangle \xrightarrow{\sigma_5} \langle s_2, 4\rangle \xrightarrow{\sigma_6} \langle s_1, 5\rangle \ .$$

Observe that the set states of the automaton $\mathit{Fold}_{\Pi}(\rho)$ are pairs $\langle s_i, j\rangle$, where s_i is a state that occurs on ρ and $j \in [1, n+1]$ is a unique identifier, that distinguishes different occurrences of the same state. These identifiers are handled by the newPath and newCycle primitives that add linear path segments and cycles to the automaton, respectively. The result of the folding operation is thus always a finite automaton recognizing languages of the form $\pi_1 \cdot \lambda_1^* \cdot \pi_2 \cdot \ldots \cdot \pi_k \cdot \lambda_k^* \cdot \pi_{k+1}$, where $\pi_1, \ldots, \pi_k$ are possibly empty words, and $\pi_{k+1}, \lambda_1, \ldots, \lambda_k$ are non-empty words[3].

Let us first prove that the folding operation implemented by Algorithm 2 satisfies point (b) of Def. 3. Notice that the isFoldable condition plays no role in this proof, which uses a combinatorial argument based on the "shape" of the folded language.

Lemma 4. *Given a predicate map Π and two paths $\rho \Subset \rho'$, for any path $\tau \in \mathit{Post}_{\Pi}(\rho, \sigma)$ there exists a path $\tau' \in \mathit{Post}_{\Pi}(\rho', \sigma)$ such that $\tau \Subset \tau'$.*

Finally, we discuss the implementation of the isFoldable test from Algorithm 2 (line 14). We provide two sets of sufficient conditions that will ensure soundness of this test with respect to condition (1), which in turn, is sufficient to ensure that condition (a) of Def. 3 is satisfied by the folding operation. Observe first that the isFoldable test takes as arguments a cyclic subpath $\rho_{i,j}$ of ρ, a predicate map Π and two lists of pairs, `pastPaths` and `pastCycles` containing the source and destination positions of the previous linear paths and (folded) cycles. More precisely, if ℓ is the current position of the path we wish to fold, then $\texttt{pastPaths} = (1, i_1) \cdot (i_2, i_3) \cdot \ldots \cdot (i_p, \ell)$ and $\texttt{pastCycles} = (i_1, i_2) \cdot \ldots \cdot (i_{p-1}, i_p)$, for some strictly increasing sequence $i_1 < i_2 < \ldots < i_p \in [1, \ell]$. The intuition is that the already folded part of ρ, up to ℓ, is of the form: $\rho_{1,i_1} \cdot \rho_{i_1,i_2}^* \cdot \rho_{i_2,i_3} \cdot \ldots \cdot \rho_{i_{p-1},i_p}^* \cdot \rho_{i_p,\ell}$.

With the above notations, the first set of conditions is:

(S1) for all (p, q) in $\texttt{pastPaths} \cdot (\ell, i)$, we have $\Upsilon(\Pi(A_{\rho_{p,q}})) \rightarrow \Upsilon(\Pi(B_{\rho_{p,q}}))$,

(S2) for all (p, q) in $\texttt{pastCycles} \cdot (i, j)$, we have $\Upsilon(\Pi(A_{\rho_{p,q}})) \rightarrow \Upsilon(\Pi(B_{\rho_{p,q}}))$.

It is not very difficult to see that, if the conditions above hold, then for any instance $\pi = \rho_{1,i_1} \cdot \rho_{i_1,i_2}^{k_1} \cdot \ldots \cdot \rho_{\ell,i} \cdot \rho_{i,j}^{k_p}$ of the folded path, for some $k_1, \ldots, k_p \geqslant 0$, we have $\neg \mathit{Accept}_{\Pi}(\pi)$. Consequently, at the end of the main loop of Algorithm 2, the condition (1) will be satisfied.

[3] A symbol of the form $(\diamond, s)$ always occurs in π_{k+1}, and this is the only occurrence of a symbol of this form.

Algorithm 2 Path Folding Algorithm

input a path $\rho : s_1 \xrightarrow{\sigma_1} \cdots \xrightarrow{\sigma_n} s_{n+1}$, and a predicate map $\Pi : Q \times Q \to 2^{\mathrm{Th}(\mathcal{D})}$
output a finite automaton $Fold_\Pi(\rho) = \langle \mathrm{Th}(\mathcal{D}) \times \mathrm{Th}(\mathcal{D}), \mathcal{Q}, \delta \rangle$ such that:
(a) $w(\rho) \in \mathcal{L}(Fold_\Pi(\rho))$, and
(b) $\neg Accept_\Pi(\rho) \Rightarrow [\forall \tau \,.\, w(\tau) \in \mathcal{L}(Fold_\Pi(\rho)) \Rightarrow \neg Accept_\Pi(\tau)]$

$\mathcal{Q} \leftarrow \{s_i\}_{i=1}^{n+1} \times [1, n+1]$; $\delta \leftarrow \varnothing$
pastPaths $\leftarrow \epsilon$; pastCycles $\leftarrow \epsilon$
nextCycles $\leftarrow$ simpleCycles(ρ)
$\ell \leftarrow 1$
while nextCycles $\neq \epsilon$ **do**
  $(i, j) \leftarrow$ head(nextCycles)
  nextCycles $\leftarrow$ tail(nextCycles)
  $\delta \leftarrow \delta \cup$ newPath($\rho_{\ell,i}$)
  if pastPaths $= \epsilon$ **then**
    pastPaths $\leftarrow (\ell, i)$
  else
    match pastPaths **with** $\pi \cdot (k, \ell)$
    pastPaths $\leftarrow \pi \cdot (k, i)$
  if isFoldable($\rho_{i,j}$, Π, pastPaths, pastCycles) **then**
    $\delta \leftarrow \delta \cup$ newCycle($\rho_{i,j}$)
    pastCycles $\leftarrow$ pastCycles $\cdot (i, j)$
    pastPaths $\leftarrow$ pastPaths $\cdot (\ell, i)$
  else
    $\delta \leftarrow \delta \cup$ newPath($\rho_{i,j}$)
    match pastPaths **with** $\pi \cdot (\ell, i)$
    pastPaths $\leftarrow \pi \cdot (\ell, j)$
  $\ell \leftarrow j$
if $\ell < n$ **then**
  $\delta \leftarrow \delta \cup$ newPath($\rho_{\ell,n}$)

function simpleCycles(ρ)
  $j \leftarrow 1$
  $S \leftarrow \varnothing$; $C \leftarrow \epsilon$
  while $j \leqslant |\rho|$ **do**
    if $\exists i \in S \,.\, src(\rho_i) = src(\rho_j)$ **then**
      $S \leftarrow \{j\}$; $C \leftarrow C \cdot (i, j)$
    else
      $S \leftarrow S \cup \{j\}$
    $j = j + 1$
  return C

Example 9. Fig. 3 shows (covered) paths trees corresponding to the cyclic path $\rho : \langle s_1, \{r_1\} \rangle \to \langle s_2, \{r_2, r_3\} \rangle \to \langle s_1, \{r_1\} \rangle$. We check the condition (S2):

$$(x_1 > x_0 \wedge y_1 \leqslant y_0) \wedge (x_2 \leqslant x_1 \wedge y_2 > y_1) \to$$
$$((x_1 - y_1 > x_0 - y_0) \wedge (x_2 - y_2 < x_1 - y_1)) \vee ((x_1 > x_0 \wedge y_1 \leqslant y_0) \wedge (x_2 \leqslant x_1 \wedge y_2 > y_1)) \quad .$$

The entailment is valid, so the path ρ can be safely folded into a loop.

There are however cases in which the simple conditions (S1-S2) are not enough, as shown in the example below. In this example, the knowledge propagated from the context of the cycle is essential for deciding that the cycle can be folded soundly, with respect to the condition (1).

Example 10. Let Π be the predicate map: $\Pi(s_1, s_2) = \{\mathbf{false}, x = 0 \wedge y = 0 \wedge x' = x \wedge y' = y\}$, $\Pi(s_2, s_2) = \{\mathbf{false}, x' - x = y' - y\}$, $\Pi(r_1, r_2) = \{\mathbf{false}, x \in [0, 1] \wedge y \in [0, 1] \wedge x' = x \wedge y' = y\}$, $\Pi(r_2, r_3) = \Pi(r_3, r_3) = \{\mathbf{false}, x \leqslant y \wedge x' \geqslant y'\}$. Fig. 6 shows diverging Π-covered path trees for the path $\rho : \langle s_1, \{r_1\} \rangle \to \langle s_2, \{r_2\} \rangle \to \langle s_2, \{r_3\} \rangle \to \langle s_2, \{r_3\} \rangle \to \ldots$.

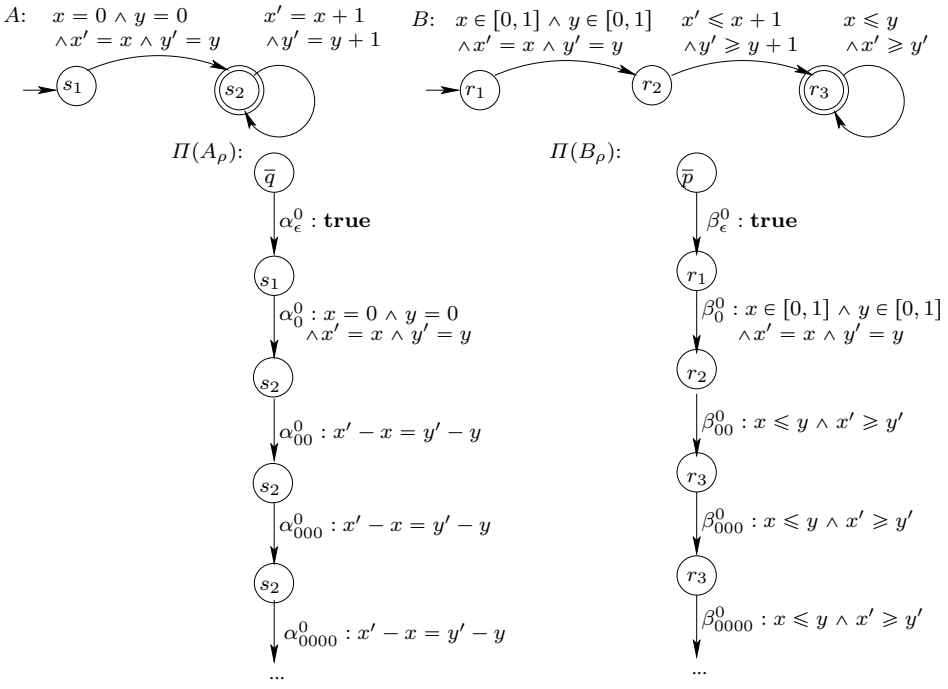

Fig. 6. Diverging (covered) path trees

Further notation is needed in order to formalize the folding conditions for this case. For a labeled tree $T = (t, \lambda)$, we define the *characteristic relation* of T by the formula:

$$\Xi(T) = \exists \mathbf{x}^{0\ldots ht(t)} \Upsilon(T) \wedge \mathbf{x} = \mathbf{x}^1 \wedge \mathbf{x}' = \mathbf{x}^{ht(t)} \ .$$

Intuitively, $\Xi(T)$ defines the relation between the variables at the first and last level on any maximal path in the tree[4]. Observe that the free variables of $\Xi(T)$ are $\mathbf{x} \cup \mathbf{x}'$. Then, for a path ρ, $\Xi(A_\rho)$ and $\Xi(B_\rho)$ define the transformations performed by A and B on the values of the variables along the path ρ, respectively.

For two formulae $\varphi(\mathbf{x}, \mathbf{x}')$ and $\psi(\mathbf{x}, \mathbf{x}')$ defining relations, let $(\varphi \circ \psi)(\mathbf{x}, \mathbf{x}') = \exists \mathbf{y} \ . \ \varphi(\mathbf{x}, \mathbf{y}) \wedge \psi(\mathbf{y}, \mathbf{x}')$ denote their composition. The *reflexive and transitive closure* $\varphi^*(\mathbf{x}, \mathbf{x}') = \bigvee_{i=0}^{\infty} \varphi^i(\mathbf{x}, \mathbf{x}')$, where φ^0 is the identity relation $\mathbf{x} = \mathbf{x}'$, and $\varphi^i = \varphi \circ \varphi^{i-1}$, for all $i > 0$. Incidentally, for a formula $\psi(\mathbf{x}^{0\ldots n})$, we define $\varphi^{\wedge m}(\mathbf{x}^{0\ldots m\cdot n}) = \psi(\mathbf{x}^{0\ldots n}) \wedge \ldots \wedge \psi(\mathbf{x}^{m\cdot(n-1)\ldots m\cdot n})$.

A relation $\varphi(\mathbf{x}, \mathbf{x}')$ is said to be *extensive* if $\varphi \rightarrow \varphi \circ \varphi$ and *reductive* if $\varphi \circ \varphi \rightarrow \varphi$. A cyclic path ρ is said to be *quasi-foldable with respect to* Π if

[4] We recall that the topmost edges of a path tree $T = (t, \lambda)$ are always $\lambda^i_\epsilon = \mathbf{true}$, so they do not need to be considered. Hence the first variables are $\mathbf{x}^1$ instead of $\mathbf{x}^0$.

$\Xi(\Pi(A_\rho))$ is reductive and $\Xi(\Pi(B_\rho))$ is extensive. The intuition is that, by increasing the number of iterations of a quasi-foldable path one also decreases the chance that this path becomes accepting, since $\Xi(\Pi(A_{\rho^n})) \leftarrow \Xi(\Pi(A_{\rho^m}))$ and $\Xi(\Pi(B_{\rho^n})) \rightarrow \Xi(\Pi(B_{\rho^m}))$, whenever $n \geqslant m$.

However this condition alone does not guarantee (1), for which reason we introduce the following sufficient set of conditions:
(C1) $\rho_{i,j}$ is quasi-foldable w.r.t. Π, and
(C2) $\mathcal{P}_\rho(i_1, \ldots, i_p) \wedge \Upsilon(\Pi(A_{\rho_{i,j}}))^{\wedge 2} \rightarrow \Upsilon(\Pi(B_{\rho_{i,j}}))^{\wedge 2}$,
where, as in the previous, $\rho_{i,j}$ is a cycle, and:

$$\mathcal{P}_\rho(i_1, \ldots, i_p) : \Xi(\Pi(A_{\rho_{1,i_1}})) \circ \Xi(\Pi(A_{\rho_{i_1,i_2}}))^* \circ \ldots \circ \Xi(\Pi(A_{\rho_{i_{p-1},i_p}}))^* \circ \Xi(\Pi(A_{\rho_{i_p,i}})) \ .$$

Observe that condition (C2) requires the computation of a relation $\mathcal{P}_\rho(i_1, \ldots, i_p)$ which consists of zero or more transitive closures. In general, this is not always possible, for e.g. arbitrary linear arithmetic relations. However, several works have shown that it is possible to compute transitive closures (accelerate) for restricted classes of linear arithmetic [4]. An implementation of the folding method using conditions (C1-C2) could use such results, and apply over- and under-approximation of the more general classes of relations into relations that can be effectively accelerated.

The following lemma proves that implementing isFoldable using the conditions (C1-C2) is sufficient to ensure the soundness condition (1).

Lemma 5. *Let Π be a predicate map, ρ be a path, $|\rho| = n$ its length, and $i \in [1, n]$ be a index, such that $\rho_{i,n}$ is a quasi-foldable cyclic path, and the following hold:*
1. $\forall \tau \ . \ w(\tau) \in \mathcal{L}(Fold_\Pi(\rho_{1,i})) \cdot w(\rho_{i,n}) \Rightarrow \neg Accept_\Pi(\tau)$,
2. $\mathcal{P}_\rho(i_1, \ldots, i_p) \wedge \Upsilon(\Pi(A_{\rho_{i,n}}))^{\wedge 2} \rightarrow \Upsilon(\Pi(B_{\rho_{i,n}}))^{\wedge 2}$,
for some sequence $1 \leqslant i_1 < i_2 < \ldots < i_p \leqslant i$, then the following holds:

$$\forall \tau \ . \ w(\tau) \in \mathcal{L}(Fold_\Pi(\rho)) \Rightarrow \neg Accept_\Pi(\tau) \ .$$

The following example shows that the diverging path from Example 10 can now be folded soundly using conditions (C1-C2).

Example 11. (contd. from Ex. 10) Consider again the path from Ex. 10 $\rho : \langle s_1, \{r_1\}\rangle \rightarrow \langle s_2, \{r_2\}\rangle \rightarrow \langle s_2, \{r_3\}\rangle \rightarrow \langle s_2, \{r_3\}\rangle$. We would like to fold ρ into $\langle s_1, \{r_1\}\rangle \rightarrow \langle s_2, \{r_2\}\rangle \rightarrow \langle s_2, \overrightarrow{\{r_3\}}\rangle$. For this, we have to show:
– (C1) $\rho_{3,3}$ is quasi-foldable:
  • $\Xi(\Pi(A_{\rho_{3,3}})) : x' - x = y' - y$ is reducible:

$$(x' - x = y' - y) \circ (x' - x = y' - y) \rightarrow (x' - x = y' - y) \ .$$

  • $\Xi(\Pi(B_{\rho_{3,3}})) : x \leqslant y \wedge x' \geqslant y'$ is extensible:

$$(x \leqslant y \wedge x' \geqslant y') \rightarrow (x \leqslant y \wedge x' \geqslant y') \circ (x \leqslant y \wedge x' \geqslant y')$$

– (C2) The entailment $\mathcal{P}_\rho \wedge \Upsilon(\Pi(A_{\rho_{3,3}}))^{\wedge 2} \rightarrow \Upsilon(\Pi(B_{\rho_{3,3}}))^{\wedge 2}$ is valid, where:

$$\begin{aligned}\mathcal{P}_\rho &: x_0 = 0 \wedge y_0 = 0 \wedge x_1 = x_0 \wedge y_1 = y_0 \wedge x_2 - x_1 = y_2 - y_1\\ \Upsilon(\Pi(A_{\rho_{3,3}}))^{\wedge 2} &: x_3 - x_2 = y_3 - y_2 \wedge x_4 - x_3 = y_4 - y_3\\ \Upsilon(\Pi(B_{\rho_{3,3}}))^{\wedge 2} &: x_2 \leqslant y_2 \wedge x_3 \geqslant y_3 \wedge x_3 \leqslant y_3 \wedge x_4 \geqslant y_4 \ .\end{aligned}$$

4.4 Soundness of the Trace Inclusion Algorithm

In this section we address the soundness of our trace inclusion semi-algorithm 1. Since termination is not guaranteed, in general, we prove that, whenever the algorithm terminates without reporting a counterexample, we have indeed that $\mathcal{L}(A) \subseteq \mathcal{L}(B)$ holds. The dual question "if there exists a counterexample trace $\tau \in \mathcal{L}(A) \backslash \mathcal{L}(B)$, will Algorithm 1 discover it ?" can also be answered positively, using an implementation that enumerates the abstract paths in a systematic way, e.g. by using a breadth-first path exploration. This can be done using a queue to implement the `Next` set in Algorithm 1.

If $P \subseteq \mathbb{N}$ is a set of positive integers, we denote by $\min P$ the (unique) least integer in P, and $\min P = \infty$ if and only if $P = \varnothing$. For a path ρ and a predicate map $\Pi : Q \times Q \rightarrow 2^{\mathrm{Th}(\mathcal{D})}$, we denote by $Dist_\Pi(\rho)$ the minimal $k \geqslant 0$ for which there exists a sequence of paths $\tau_0, \ldots, \tau_k$, $\tau_0 = \rho$ and $\tau_{i+1} \in Post_\Pi(\tau_i)$, for all $i \in [0, k-1]$, such that $Accept_\Pi(\tau_k)$. Obviously, $Dist_\Pi(\rho) = \infty$ if there is no such sequence. Intuitively, $Dist_\Pi(\rho)$ is the length of the minimal accepting continuation of ρ, under the predicate map Π. For a finite set S of paths, we define $Dist_\Pi(S) = \min\{Dist_\Pi(\rho) \mid \rho \in S\}$.

For simplicity, from now on we identify any node $s = \langle q, P, p\rangle$ in the APRT built by Algorithm 1 with the unique path $\langle i, I_B\rangle \xrightarrow{\sigma_1} \langle q_1, P_1\rangle \xrightarrow{\sigma_2} \ldots \xrightarrow{\sigma_k} \langle q, P\rangle$ that can be recovered from the path leading to s in the APRT. An APRT $\mathcal{A} = \langle \Pi, \mathtt{Visited}, \mathtt{Next}, \mathtt{Edges}, \mathtt{Subsume}\rangle$ is said to be *closed*, if and only if for every $s \in \mathtt{Visited}$ and every successor $t \in Post_\Pi(s)$ there exists an edge $(s, u) \in \mathtt{Edges} \cup \mathtt{Subsumed}$ such that $t \sqsubseteq u$. Proving soundness of Algorithm 1 relies on the following invariants:

Lemma 6. *Let* $Roots = \{\langle i, I_B, \ell - 1\rangle \mid \ell \in [1, K]\}$, *where* $I_A = \{i_1, \ldots, i_K\}$. *Then the following hold, each time line 4 is reached by Algorithm 1:*

– $\mathcal{A} = \langle \Pi, \mathtt{Visited}, \mathtt{Next}, \mathtt{Edges}, \mathtt{Subsume}\rangle$ *is closed* (Inv_1)
– $Dist_\Pi(Roots) < \infty \Rightarrow Dist_\Pi(\mathtt{Visited}) > Dist_\Pi(\mathtt{Next})$ (Inv_2)

The following technical lemma is needed in the following arguments, concerning soundness (under the termination assumption) and completeness of (a slightly refined version of) Algorithm 1.

Lemma 7. *Let* A *and* B *be two DA such that* $\mathcal{L}(A) \nsubseteq \mathcal{L}(B)$, *Then, there exists a path* ρ *such that, for every predicate map* Π, *we have* $Accept_\Pi(\rho)$.

The following soundness theorem is the main result of this section. Essentially, it shows that whenever Algorithm 1 terminates and reports that the inclusion holds, this is indeed the case.

Theorem 2. *Let A and B be two DA. If Algorithm 1 terminates and returns* **true***, when given in input A and B, then $\mathcal{L}(A) \subseteq \mathcal{L}(B)$.*

At this point, a dual question arises: if $\mathcal{L}(A) \nsubseteq \mathcal{L}(B)$, can Algorithm 1 prove this fact ? In the current presentation, this is not guaranteed, because Algorithm 1 may explore paths in a non-exhaustive way and miss a concrete accepting path. However, an implementation of this algorithm that would explore the paths systematically (e.g. in breadth-first order) will meet the latter requirement (this can be ensured by using a FIFO queue for the implementation of the `Next` set).

Suppose that $\mathcal{L}(A) \nsubseteq \mathcal{L}(B)$. By Lemma 7, we have $Dist_{\Pi}(Roots) < \infty$, for every predicate map Π. Then, by Lemma 6 (Inv_2), the number of steps between each refinement (change of the predicate map Π) must be finite, since $Dist_{\Pi}(\texttt{Visited}) > 0$ decreases each time all nodes from `Next` have been processed. Since all accepting paths are explored in an exhaustive manner, the counterexample will be eventually reported.

5 Verification Examples

The implementation of the semi-algorithm presented is currently undergoing. To validate the method, we have simulated the abstraction refinement loop on two examples steming from existing verification problems. The first example is a verification condition in an array logic, proved in [3] using integer-valued counter automata. The second example is the verification of a train crossing real-time system [13] using real-valued counter automata translated from timed automata, using the technique from [9]. We have computed the interpolants using the MathSAT5 SMT solver [5].

5.1 Array Example

We experimented our method on a practical verification task from the area of verification of array manipulating programs. Namely, we consider the code fragment shown in Fig. 7, which rotates array `a` by one position to the right, storing the result into array `b`.

```
assume(n>0);
int a[n], b[n];
for (i=0; i<n-1; i++) b[i+1] = a[i];
b[0] = a[n-1];
```

Fig. 7. A program rotating an array.

The inductive loop invariant needed to prove that the code performs the rotation, is:

$$\varphi : 0 \leqslant \mathtt{i} < \mathtt{n} \ \wedge \ \forall 0 \leqslant \mathtt{j} < i \ . \ \mathtt{b}[\mathtt{j}+1] = \mathtt{a}[\mathtt{j}] \ .$$

Following the approach proposed in [3], the invariant can be encoded using the DA shown in Fig. 5.1(a). The main idea of the encoding is viewing the sequences

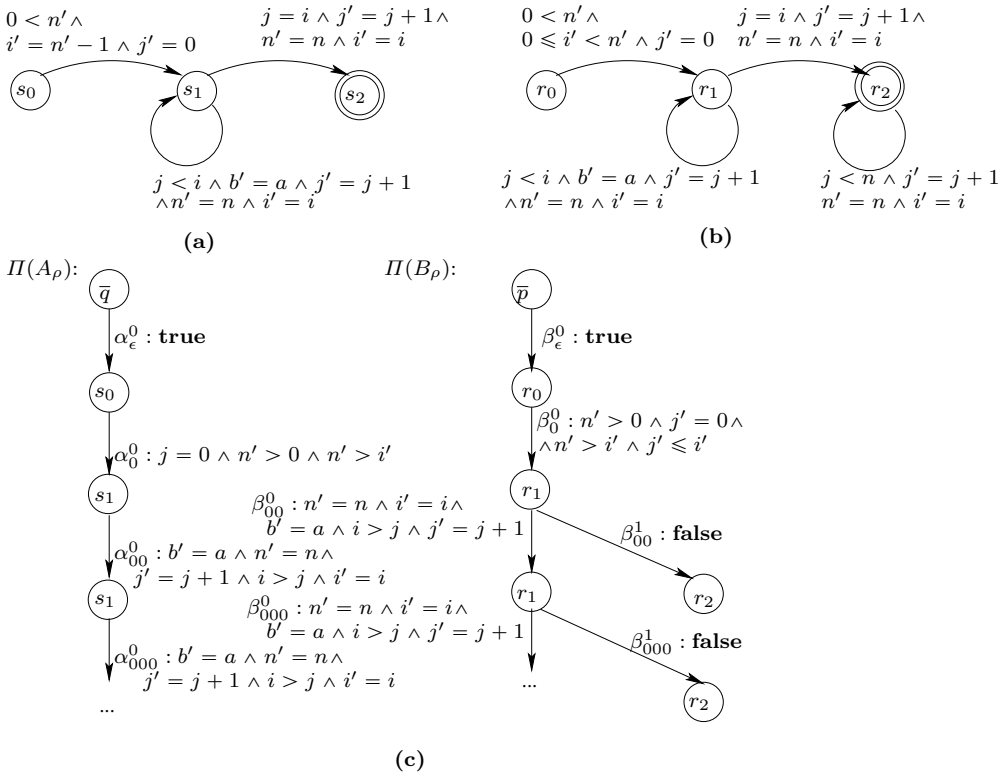

Fig. 8. DA for the array example, with the covered path trees for the unique diverging path, after 3 refinements

of values that variables a and b can have in accepting runs of the automaton as possible contents of arrays `a` and `b`.

Using our method, we show that the case when the entire contents of `a` is rotated in `b` (apart from cell `a[n-1]` that is handled outside of the loop), which can be expressed by the formula:

$$\psi : \mathtt{i} = \mathtt{n} - 1 \geqslant 0 \ \wedge \ \forall 0 \leqslant \mathtt{j} < i \ . \ \mathtt{b[j+1]} = \mathtt{a[j]}$$

encoded by the DA shown in Fig. 5.1(b), is entailed by the loop invariant, i.e. $\psi \rightarrow \varphi$ holds. Our technique was able to establish that this is indeed the case using 3 refinements. Fig. 5.1(c) shows the path trees corresponding to the diverging path $\rho : \langle s_0, \{r_0\}\rangle \rightarrow \langle s_1, \{r_1\}\rangle \rightarrow \langle s_1, \{r_1, r_2\}\rangle \rightarrow \langle s_1, \{r_1, r_2\}\rangle \rightarrow \ldots$ Using the conditions (S1) and (S2), we can fold the path ρ into the automaton:

$$\mathit{Fold}_\Pi(\rho) : \langle s_0, \{r_0\}\rangle \rightarrow \langle s_1, \{r_1\}\rangle \rightarrow \langle s_1, \{\overset{\frown}{r_1}, r_2\}\rangle$$

which is sufficient to terminate the abstraction-refinement loop.

5.2 Timed Automata Example

The second verification example we consider is a real-time controller of the train crossing example from [13] (Section 6.2, pg. 235). The example models a system composed of a train and a railway crossing gate, modeled by timed automata. The two systems communicate via signals *app* (approaching), *in* (the crossing), *out* (of the crossing), (gate) *down*, (gate) *up*. The synchronous composition of the two systems is a timed automaton, which is translated into a real-valued counter automaton A (Fig. 5.2 (a)) using the technique from [9] (Example 1, pg. 4). For the composed system we check the property that the gate never stays closed for more than 5 minutes[5]. The property is given by a timed automaton translated to the real-valued counter automaton B, in Fig. 5.2 (b).

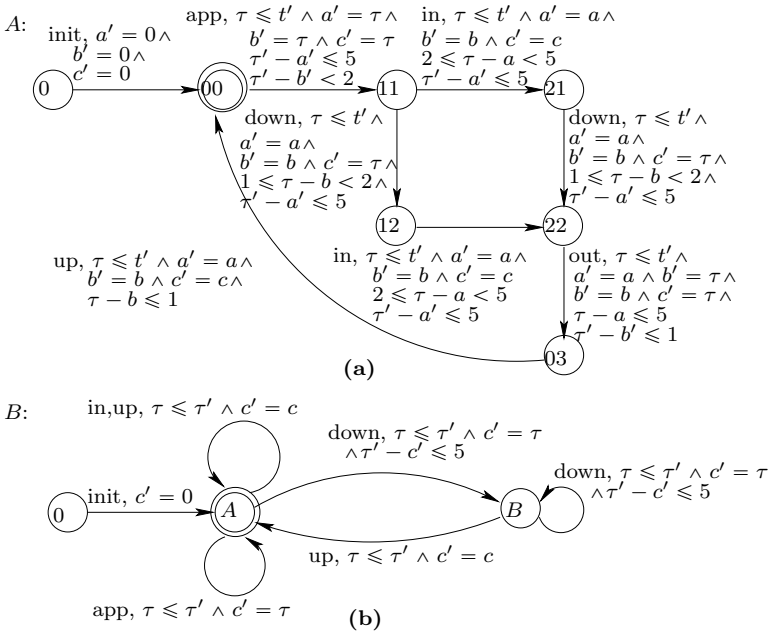

Fig. 9.

First, our method detects that the path:

$$\rho_1 : \langle 0, \{0\}\rangle \xrightarrow{init} \langle A, \{00\}\rangle \xrightarrow{app} \langle A, \{11\}\rangle \xrightarrow{in} \langle A, \{21\}\rangle \xrightarrow{down} \langle B, \{22\}\rangle$$

is infeasible, since the edge $11 \rightarrow 21$ in the corresponding covered path tree $\Pi(A_{\rho_1})$ is labeled by **false**. Then, after 3 refinements of the predicate map, we

[5] In Timed Linear Temporal Logic $\Box(down \rightarrow \Diamond_{\leqslant 5} up)$.

obtain the diverging path:

$$\begin{aligned}\rho_2 : \langle 0, \{0\}\rangle &\xrightarrow{init} \langle A, \{00\}\rangle \\ &\xrightarrow{app} \langle A, \{11\}\rangle \xrightarrow{down} \langle B, \{12\}\rangle \xrightarrow{in} \langle B, \{22\}\rangle \xrightarrow{out} \langle B, \{03\}\rangle \xrightarrow{up} \langle A, \{00\}\rangle \\ &\xrightarrow{app} \langle A, \{11\}\rangle \xrightarrow{down} \langle B, \{12\}\rangle \xrightarrow{in} \langle B, \{22\}\rangle \xrightarrow{out} \langle B, \{03\}\rangle \xrightarrow{up} \langle A, \{00\}\rangle \ \ldots\end{aligned}$$

The corresponding covered path trees are depicted in Fig. 5.2. The subsumption relation uses the conditions (S1) and (S2), and folds the path ρ into the automaton $Fold_{\Pi}(\rho_2)$:

$$\begin{aligned}&\langle 0, \{0\}\rangle \xrightarrow{init} \langle A, \{00\}\rangle \\ &\Big(\xrightarrow{app} \langle A, \{11\}\rangle \xrightarrow{down} \langle B, \{12\}\rangle \xrightarrow{in} \langle B, \{22\}\rangle \xrightarrow{out} \langle B, \{03\}\rangle \xrightarrow{up} \langle A, \{00\}\rangle \Big)^*\end{aligned}$$

sufficient to end the refinement loop.

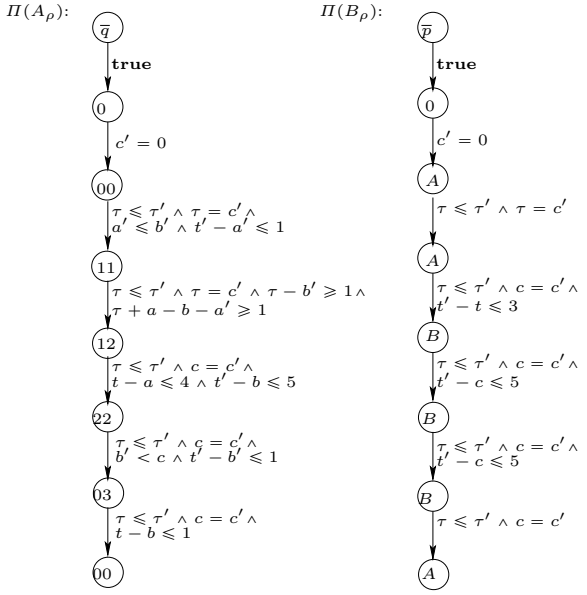

Fig. 10.

6 Conclusions

We have presented an interpolation-based abstraction refinement method for trace inclusion between general data automata, with the only restriction that the Craig Interpolation Lemma holds for the base theory, which describes the transition rules. The technique uses predicates that label the edges of the automata and does refinement using a novel interpolation scheme. The procedure

has been applied to several examples, including a verification problem for array programs, and a real-time system. Future work includes the extension of the method to data tree automata, and application to logics for heaps with data.

References

1. P.A. Abdulla, Y.-F. Chen, L. Holik, R. Mayr, and T. Vojnar. When simulation meets antichains. In *Proc. of TACAS'10*, volume 6015 of *LNCS*, pages 158–174. Springer, 2010.
2. M. Bojańczyk, C. David, A. Muscholl, T. Schwentick, and L. Segoufin. Two-variable logic on data words. *ACM Trans. Comput. Logic*, 12(4):27:1–27:26, 2011.
3. M. Bozga, P. Habermehl, R. Iosif, F. Konecný, and T. Vojnar. Automatic verification of integer array programs. In *Proc. of CAV'09*, volume 5643 of *LNCS*, pages 157–172. Springer, 2009.
4. M. Bozga, R. Iosif, and F. Konecný. Fast acceleration of ultimately periodic relations. In *Proc. of CAV'10*, volume 6174 of *LNCS*, pages 227–242. Springer, 2010.
5. A. Cimatti, A. Griggio, B. Schaafsma, and R. Sebastiani. The MathSAT5 SMT Solver. In *Proc. of TACAS*, volume 7795 of *LNCS*. Springer, 2013.
6. W. Craig. Three uses of the herbrand-gentzen theorem in relating model theory and proof theory. *J. Symb. Log.*, 22(3):269–285, 1957.
7. N. Decker, P. Habermehl, M. Leucker, and D. Thoma. Ordered navigation on multi-attributed data words. In *Proc. of CONCUR'14*, volume 8704 of *LNCS*, pages 497–511. Springer, 2014.
8. D. Felsing, S. Grebing, V. Klebanov, P. Rümmer, and M. Ulbrich. Automating regression verification. In *Proc. of ASE'14*. ACM, 2014.
9. L. Fribourg. A closed-form evaluation for extended timed automata. Technical report, CNRS et Ecole Normale Sup'erieure de Cachan, 1998.
10. P. Habermehl, R. Iosif, and T. Vojnar. A logic of singly indexed arrays. In *Proc. of LPAR'08*, volume 5330 of *LNCS*. Springer, 2008.
11. P. Habermehl, R. Iosif, and T. Vojnar. What else is decidable about integer arrays? In *Proc. of FOSSACS'08*, volume 4962 of *LNCS*, pages 474–489. Springer, 2008.
12. T. A. Henzinger, R. Jhala, R. Majumdar, and G. Sutre. Software Verification with Blast. In *Proc. of 10th SPIN Workshop*, volume 2648 of *LNCS*. Springer, 2003.
13. T. A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111:394–406, 1992.
14. B. Khoussainov and A. Nerode. *Automata Theory and Its Applications*. Birkhauser Boston, 2001.
15. K. L. McMillan. Lazy abstraction with interpolants. In *Proc. of CAV'06*, volume 4144 of *LNCS*. Springer, 2006.
16. H. H. Nguyen and W.-N. Chin. Enhancing program verification with lemmas. In *Proc. of CAV'08*, volume 5123 of *LNCS*, pages 355–369. Springer, 2008.
17. M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification (preliminary report). In *Proc. of LICS'86*, pages 332–344. IEEE Computer Society, 1986.

Proofs

Proof of Lemma 1

Proof. Since $T = (t, \lambda)$ is deterministic, we have $dom(t) = \{i_1 \dots i_k \mid k \in [1, N]\} \cup \{\epsilon\}$, for some integers $i_1, \dots, i_N \in \mathbb{N}$, where $N = ht(t)$. We denote $p_0 = \epsilon$ and $p_j = i_1 \dots i_j$, for all $j \in [1, N]$, in the rest of this proof. With this notation, we have $\Upsilon(T) = \bigwedge_{j=0}^{N-1} \lambda_{p_j}^{i_{j+1}}(\mathbf{x}^j, \mathbf{x}^{j+1})$.

(1) "$\Rightarrow$" By induction on k, we prove the existence of a sequence of formulae $I_{p_0}^{i_1}(\mathbf{x}, \mathbf{x}'), \dots, I_{p_{N-1}}^{i_N}(\mathbf{x}, \mathbf{x}')$, such that, for all $k \in [0, N-1]$:

(i) $\lambda_{p_k}^{i_{k+1}}(\mathbf{x}, \mathbf{x}') \rightarrow I_{p_k}^{i_{k+1}}(\mathbf{x}, \mathbf{x}')$, and
(ii) $\bigwedge_{j=0}^{k} I_{p_j}^{i_{j+1}}(\mathbf{x}^j, \mathbf{x}^{j+1}) \wedge \bigwedge_{j=k+1}^{N-1} \lambda_{p_j}^{i_{j+1}}(\mathbf{x}^j, \mathbf{x}^{j+1}) \rightarrow \varphi$.

The first point above ensures that $\mathcal{I} = (t_A, I)$ is a OI for A. The second point ensures that $\Upsilon(\mathcal{I}) \rightarrow \varphi$, by taking $k = N-1$. To prove the invariants, consider first the base case $k = 0$:

$$\bigwedge_{j=0}^{N-1} \lambda_{p_j}^{i_{j+1}}(\mathbf{x}^j, \mathbf{x}^{j+1}) \rightarrow \varphi$$
$$\lambda_{p_0}^{i_1}(\mathbf{x}^0, \mathbf{x}^1) \rightarrow \varphi \vee \bigvee_{j=1}^{N-1} \neg \lambda {p_j}^{i_{j+1}}(\mathbf{x}^j, \mathbf{x}^{j+1}) \ .$$

By the Craig Interpolation Lemma, there exists $I_{p_0}^{i_1}(\mathbf{x}, \mathbf{x}')$ such that:

$$\lambda_{p_0}^{i_1}(\mathbf{x}^0, \mathbf{x}^1) \rightarrow I_{p_0}^{i_1}(\mathbf{x}^0, \mathbf{x}^1) \text{ (i) and}$$
$$I_{p_0}^{i_1}(\mathbf{x}^0, \mathbf{x}^1) \rightarrow \varphi \vee \bigvee_{j=1}^{N-1} \neg \lambda {p_j}^{i_{j+1}}(\mathbf{x}^j, \mathbf{x}^{j+1})$$
$$I_{p_0}^{i_1}(\mathbf{x}^0, \mathbf{x}^1) \wedge \bigwedge_{j=1}^{N-1} \lambda {p_j}^{i_{j+1}}(\mathbf{x}^j, \mathbf{x}^{j+1}) \rightarrow \varphi \text{ (ii)} \ .$$

For the induction step $k \rightarrow k+1$, $k \in [0, N-1]$, we have:

$$\bigwedge_{j=0}^{k} I_{p_j}^{i_{j+1}}(\mathbf{x}^j, \mathbf{x}^{j+1}) \wedge \bigwedge_{j=k+1}^{N-1} \lambda_{p_j}^{i_{j+1}}(\mathbf{x}^j, \mathbf{x}^{j+1}) \rightarrow \varphi$$
$$\lambda_{p_{k+1}}^{i_{k+2}}(\mathbf{x}^{k+1}, \mathbf{x}^{k+2}) \rightarrow \varphi \vee \bigvee_{j=0}^{k} \neg I_{p_j}^{i_{j+1}}(\mathbf{x}^j, \mathbf{x}^{j+1})$$
$$\vee \bigvee_{j=k+2}^{N-1} \neg \lambda_{p_j}^{i_{j+1}}(\mathbf{x}^j, \mathbf{x}^{j+1})$$

By the Craig Interpolation Lemma, there exists a formula $I_{p_{k+1}}^{i_{k+2}}(\mathbf{x}, \mathbf{x}')$ such that:

$$\lambda_{p_{k+1}}^{i_{k+2}}(\mathbf{x}^{k+1}, \mathbf{x}^{k+2}) \rightarrow I_{p_{k+1}}^{i_{k+2}}(\mathbf{x}^{k+1}, \mathbf{x}^{k+2}) \text{ (i) and}$$
$$I_{p_{k+1}}^{i_{k+2}}(\mathbf{x}^{k+1}, \mathbf{x}^{k+2}) \rightarrow \varphi \vee \bigvee_{j=0}^{k} \neg I_{p_j}^{i_{j+1}}(\mathbf{x}^j, \mathbf{x}^{j+1})$$
$$\vee \bigvee_{j=k+2}^{N-1} \neg \lambda_{p_j}^{i_{j+1}}(\mathbf{x}^j, \mathbf{x}^{j+1})$$
$$\bigwedge_{j=0}^{k+1} I_{p_j}^{i_{j+1}}(\mathbf{x}^j, \mathbf{x}^{j+1}) \wedge \bigwedge_{j=k+2}^{N-1} \lambda_{p_j}^{i_{j+1}}(\mathbf{x}^j, \mathbf{x}^{j+1}) \rightarrow \varphi \text{ (ii)} \ .$$

"$\Leftarrow$" We compute, successively:

$$\Upsilon(\mathcal{I}) = \bigwedge_{j=0}^{N-1} I_{p_j}^{i_{j+1}} \rightarrow \varphi$$
$$\bigwedge_{j=0}^{N-2} I_{p_j}^{i_{j+1}} \wedge \lambda_{p_{N-1}}^{i_N} \xrightarrow{\left(\lambda_{p_{N-1}}^{i_N} \rightarrow I_{p_{N-1}}^{i_N}\right)} \bigwedge_{j=0}^{N-1} I_{p_j}^{i_{j+1}} \rightarrow \varphi$$
$$\dots$$
$$\Upsilon(T) = \bigwedge_{j=0}^{N-1} \lambda_{p_j}^{i_{j+1}} \rightarrow \varphi \ .$$

(2) "$\Rightarrow$" If $\varphi \rightarrow \Upsilon(T) = \bigwedge_{j=0}^{N-1} \lambda_{p_j}^{i_{j+1}}(\mathbf{x}^j, \mathbf{x}^{j+1})$, we have $\varphi \rightarrow \lambda_{p_j}^{i_{j+1}}(\mathbf{x}^j, \mathbf{x}^{j+1})$, for all $j \in [0, N-1]$. By the Craig Interpolation Lemma, we obtain formulae $I_{p_0}^{i_1}(\mathbf{x}, \mathbf{x}'), \ldots, I_{p_{N-1}}^{i_N}(\mathbf{x}, \mathbf{x}')$ such that $\varphi \rightarrow I_{p_j}^{i_{j+1}}(\mathbf{x}^j, \mathbf{x}^{j+1})$ and $I_{p_j}^{i_{j+1}} \rightarrow \lambda_{p_j}^{i_{j+1}}$, for all $j \in [0, N-1]$. Hence $\mathcal{I} = (t_B, I)$ is a UI for B, and, moreover $\varphi \rightarrow \bigwedge_{j=0}^{N-1} I_{p_j}^{i_{j+1}}(\mathbf{x}^j, \mathbf{x}^{j+1}) = \Upsilon(\mathcal{J})$. "$\Leftarrow$" We compute, successively:

$$
\begin{array}{l}
\varphi \rightarrow \bigwedge_{j=0}^{N-1} I_{p_j}^{i_{j+1}}(\mathbf{x}^j, \mathbf{x}^{j+1}) = \Upsilon(\mathcal{J}) \\
\varphi \rightarrow \bigwedge_{j=0}^{N-1} I_{p_j}^{i_{j+1}}(\mathbf{x}^j, \mathbf{x}^{j+1}) \xrightarrow{\left(I_{p_{N-1}}^{i_N} \rightarrow \lambda_{p_{N-1}}^{i_N}\right)} \bigwedge_{j=0}^{N-2} I_{p_j}^{i_{j+1}}(\mathbf{x}^j, \mathbf{x}^{j+1}) \wedge \lambda_{p_{N-1}}^{i_N}(\mathbf{x}^{N-1}, \mathbf{x}^N) \\
\quad \ldots \\
\varphi \rightarrow \bigwedge_{j=0}^{N-1} \lambda_{p_j}^{i_{j+1}}(\mathbf{x}^j, \mathbf{x}^{j+1}) = \Upsilon(T) \ .
\end{array}
$$

□

Proof of Theorem 1

Proof. "$\Rightarrow$" Let $N = \min(ht(A), ht(B))$. If $\Upsilon(A) \rightarrow \Upsilon(B)$, by the Craig Interpolation Lemma, there exists a formula $\varphi(\mathbf{x}^{0 \cdots N})$, such that $\Upsilon(A) \rightarrow \varphi$ and $\varphi \rightarrow \Upsilon(B)$. By Lemma 1 (1), there exist an OI $\mathcal{I} = (t_A, I)$ for A such that $\Upsilon(\mathcal{I}) \rightarrow \varphi$. It is sufficient, at this point, to build a UI $\mathcal{J} = (t_B, J)$ for B, such that $\varphi \rightarrow \Upsilon(\mathcal{J})$. We have $\varphi \rightarrow \Upsilon(B) = \bigvee_{T \in \langle\langle B \rangle\rangle} \Upsilon(T)$, hence

$$\varphi = \Upsilon(B) \wedge \varphi = \left(\bigvee_{T \in \langle\langle B \rangle\rangle} \Upsilon(T) \right) \wedge \varphi = \bigvee_{T \in \langle\langle B \rangle\rangle} (\varphi \wedge \Upsilon(T)) \ .$$

Since $\varphi \wedge \Upsilon(T) \rightarrow \Upsilon(T)$, there exist a UI $\mathcal{J}_T = (t, J_T)$ for T, such that $\varphi \wedge \Upsilon(T) \rightarrow \Upsilon(\mathcal{J}_T)$, by Lemma 1 (2). We define $\mathcal{J}$ as follows. For each edge $(p, p.i) \in \mathit{edges}(t_B)$, we define:

$$J_p^i(\mathbf{x}, \mathbf{x}') = \bigvee_{\substack{T \in \langle\langle B \rangle\rangle \\ (p, p.i) \in \mathit{edges}(T)}} {J_T}_p^i(\mathbf{x}, \mathbf{x}') \ .$$

To verify that indeed $\mathcal{J} = (t_B, J)$ is a UI for B, let $(p, p.i) \in \mathit{edges}(t_B)$ be an arbitrary edge. For all $T \in \langle\langle B \rangle\rangle$ such that $(p, p.i) \in \mathit{edges}(T)$, we have ${J_T}_p^i(\mathbf{x}, \mathbf{x}') \rightarrow \lambda_p^i(\mathbf{x}, \mathbf{x}')$, by Def. 1, hence $J_p^i = \bigvee \left\{ {J_T}_p^i \mid T \in \langle\langle B \rangle\rangle, (p, p.i) \in \mathit{edges}(T) \right\} \rightarrow \lambda_p^i$. Finally, to check that indeed $\varphi \rightarrow \Upsilon(\mathcal{J})$, observe the following:

$$\varphi = \Upsilon(B) \wedge \varphi = \bigvee_{T \in \langle\langle B \rangle\rangle} (\Upsilon(T) \wedge \varphi) \rightarrow \bigvee_{T \in \langle\langle B \rangle\rangle} \Upsilon(\mathcal{J}_T) \rightarrow \Upsilon(\mathcal{J}) \ .$$

□

Proof of Lemma 2

Proof. Let $\tau \in \mathit{Post}_{\Pi}(\rho, \sigma)$ be a Π-feasible extension of ρ, where $s = \langle q, P \rangle$ is the last product state on ρ, and $s \xrightarrow{\sigma} s' = \langle q', P' \rangle$ is the last product edge on τ, for some rule $\left(q \xrightarrow{\sigma,\varphi} q'\right) \in \Delta_A$, and several other rules of B. Since τ is Π-feasible, i.e. $\Upsilon(\Pi(A_\tau))$ is satisfiable, it is sufficient to show that $\Upsilon(\Pi(A_\tau)) \rightarrow \Upsilon(\Pi'(A_\tau))$. This would imply that $\Upsilon(\Pi'(A_\tau))$ is satisfiable, and τ is Π'-feasible, i.e. $\tau \in \mathit{Post}_{\Pi'}(\rho, \sigma)$, which is what needs to be proved.

Let $A_\tau = (t, \lambda)$ – we recall that A_τ is deterministic. Let us denote $\Pi(A_\tau) = (t, \alpha)$ and $\Pi'(A_\tau) = (t, \beta)$, in the following. We shall prove, by induction on the length of $p \in \mathit{dom}(t)$, that $\alpha_p^i \rightarrow \beta_p^i$, for all $p.i \in \mathit{dom}(t)$. For the base case $|p| = 0$, i.e. $p = \epsilon$, we have, because $\Pi'(t(\epsilon), t(i)) \subseteq \Pi(t(\epsilon), t(i))$:

$$\alpha_\epsilon^i = \bigwedge \left\{ I \in \Pi(t(\epsilon), t(i)) \mid \lambda_\epsilon^i \rightarrow I \right\} \rightarrow \bigwedge \left\{ I \in \Pi'(t(\epsilon), t(i)) \mid \lambda_\epsilon^i \rightarrow I \right\} = \beta_\epsilon^i \ .$$

For the induction step, we assume that $\alpha_p \rightarrow \beta_p$, and we prove that, for any assertion I, $(\alpha_p \wedge \lambda_p^i \rightarrow \alpha_p \wedge I)$ if $(\beta_p \wedge \lambda_p^i \rightarrow \beta_p \wedge I)$. Suppose that $(\alpha_p \wedge \lambda_p^i \rightarrow \alpha_p \wedge I)$ does not hold, i.e. there exists a valuation $\nu : \mathbf{x}^{0 \cdots |p|} \rightarrow \mathcal{D}$ such that $\nu \models_{\mathrm{Th}(\mathcal{D})} \alpha_p$, $\nu \models_{\mathrm{Th}(\mathcal{D})} \lambda_p^i$ and $\nu \not\models_{\mathrm{Th}(\mathcal{D})} I$. Since $\alpha_p \rightarrow \beta_p$, by the induction hypothesis, we have that $\nu \models_{\mathrm{Th}(\mathcal{D})} \beta_p$, $\nu \models_{\mathrm{Th}(\mathcal{D})} \lambda_p^i$ and $\nu \not\models_{\mathrm{Th}(\mathcal{D})} I$, i.e. $\nu \not\models_{\mathrm{Th}(\mathcal{D})} \beta_p \wedge I$, which contradicts the fact that $(\beta_p \wedge \lambda_p^i \rightarrow \beta_p \wedge I)$. We compute, next:

$$\begin{aligned} \left\{ I \in \Pi(t(p), t(p.i)) \mid \alpha_p \wedge \lambda_p^i \rightarrow \alpha_p \wedge I \right\} &\supseteq \left\{ I \in \Pi(t(p), t(p.i)) \mid \beta_p \wedge \lambda_p^i \rightarrow \beta_p \wedge I \right\} \\ &\supseteq \left\{ I \in \Pi'(t(p), t(p.i)) \mid \beta_p \wedge \lambda_p^i \rightarrow \beta_p \wedge I \right\} \ . \end{aligned}$$

and, consequently:

$$\begin{aligned} \alpha_p^i &= \bigwedge \left\{ I \in \Pi(t(p), t(p.i)) \mid \alpha_p \wedge \lambda_p^i \rightarrow \alpha_p \wedge I \right\} \\ &\rightarrow \bigwedge \left\{ I \in \Pi'(t(p), t(p.i)) \mid \beta_p \wedge \lambda_p^i \rightarrow \beta_p \wedge I \right\} = \beta_p^i \ . \end{aligned}$$

Since $\alpha_p^i \rightarrow \beta_p^i$, for all $p.i \in \mathit{dom}(t)$, we conclude $\Upsilon(\Pi(A_\tau)) \rightarrow \Upsilon(\Pi'(A_\tau))$. □

Proof of Lemma 3

Proof. Let $A_\rho = (t, \lambda)$, $B_\rho = (u, \mu)$, $\Pi_{|k}(A_\rho) = (t, \lambda_{|k})$ and $\Pi_{|k}(B_\rho) = (u, \mu_{|k})$ in the following. By the definition of the pivot, we have $\neg \mathit{cex}(\Pi_{|k}(A_\rho), \Pi_{|k}(B_\rho))$, and therefore, the entailment $\Upsilon(\Pi_{|k}(A_\rho)) \rightarrow \Upsilon(\Pi_{|k}(B_\rho{\downarrow_{F_B}}))$ holds. By Thm. 1, there exists a GI $(\mathcal{I}, \mathcal{J})$, with $\mathcal{I} = (t, I)$ and $\mathcal{J} = (u{\downarrow_{F_B}}, J)$, for the pair $(\Pi_{|k}(A_\rho), \Pi_{|k}(B_\rho{\downarrow_{F_B}}))$.

For the second point, assume that Π' is compliant with $(\mathcal{I}, \mathcal{J})$, and let $\Pi'(A_\rho) = (t, \alpha)$, $\Pi'(B_\rho) = (u, \beta)$. Then the following hold:

- for each $(p, p.i) \in \mathit{edges}(t)$ we have $(\lambda_{|k})_p^i \rightarrow I_p^i$ and $I_p^i \in \Pi'(t(p), t(p.i))$, therefore $\alpha_p^i \rightarrow I_p^i$. Consequently, we obtain $\Upsilon(\Pi'(A_\rho)) \rightarrow \Upsilon(\mathcal{I})$.
- for each $(q, q.j) \in \mathit{edges}(u{\downarrow_{F_B}})$ we have $J_q^j \rightarrow (\mu_{|k})_q^j$ and $J_q^j \in \Pi'(t(p), t(p.j))$, therefore $J_q^j \rightarrow \beta_q^j$. Consequently, we obtain $\Upsilon(\mathcal{J}) \rightarrow \Upsilon(\Pi'(B_\rho))$.

Since (I, J) is a GI, we have $\Upsilon(I) \rightarrow \Upsilon(J)$, hence $\Upsilon(\Pi'(A_\rho)) \rightarrow \Upsilon(\Pi'(B_\rho))$, and, consequently $\neg \mathit{cex}(\Pi'(A_\rho), \Pi'(B_\rho))$, hence $\neg \mathit{Accept}_{\Pi'}(\rho)$. □

Proof of Lemma 4

Proof. Let $s = dest(\rho)$ be the last product state on ρ, and $s \xrightarrow{\sigma} t$ be the last product edge on τ, i.e. $w(\rho) = u \cdot (\diamond, s)$ and $w(\tau) = u \cdot (\sigma, s)(\diamond, t)$. Let $\mathcal{L}(Fold_{\Pi}(\rho)) = \pi_1 \cdot \lambda_1^* \ldots \lambda_k^* \cdot \pi_{k+1}$ and $\mathcal{L}(Fold_{\Pi}(\rho')) = \nu_1 \cdot \mu_1^* \ldots \mu_\ell^* \cdot \nu_{\ell+1}$. Since $\mathcal{L}(Fold_{\Pi}(\rho)) \subseteq \mathcal{L}(Fold_{\Pi}(\rho'))$, then both π_{k+1} and $\nu_{\ell+1}$ must end with the same symbol $(\diamond, s)$, and since $w(\rho') \in \mathcal{L}(Fold_{\Pi}(\rho'))$, we have $dest(\rho') = s$. Then we define τ' to be the extension of ρ' with the edge $s \xrightarrow{\sigma} t$. We must show next that $\mathcal{L}(Fold_{\Pi}(\tau)) \subseteq \mathcal{L}(Fold_{\Pi}(\tau'))$.

For simplicity, we assume that $k = \ell = 1$, i.e. $Fold_{\Pi}(\rho) = \pi_1 \cdot \lambda^* \cdot \pi_2$ and $Fold_{\Pi}(\rho') = \nu_1 \cdot \mu^* \cdot \nu_2$. The general proof for $k, \ell \geqslant 1$ uses essentially the same argument. We have that, for any $x \geqslant 0$ there exists $y \geqslant 0$ such that $\pi_1 \cdot \lambda^x \cdot \pi_2 = \nu_1 \cdot \mu^y \cdot \nu_2$, and consequently, $|\pi_1| + x|\lambda| + |\pi_2| = |\nu_1| + y|\mu| + |\nu_2|$. Taking $x' = x + 1$, there exists $y' \geqslant 0$ such that $|\lambda| = (y' - y)|\mu|$. Since λ is a simple cycle, in which no two same symbols occur, the only possibility is thus $y' = y + 1$, i.e. $|\lambda| = |\mu|$.

We shall prove that $|\pi_1| = |\nu_1|$ as well. Suppose, by contradiction, that $|\pi_1| > |\nu_1|$ (the case $|\pi_1| < |\nu_1|$ is symmetric). By taking a sufficiently large $x \geqslant 0$, we have $\pi_1 \cdot \lambda^x \cdot \pi_2 = \nu_1 \cdot \mu^y \cdot \nu_2$ for some $y \geqslant 0$, and $\pi_1 = \nu_1 \cdot \alpha$, for some prefix $\alpha \preceq \mu^\ell$ and some $\ell \geqslant 0$. But π_1 may not have repeating symbols either, then the only possibility is $\ell = 1$, i.e. $\mu = \alpha \cdot \beta$, for some word β. Since $|\lambda| = |\mu|$, we obtain that $\lambda = \beta \cdot \alpha$, again, by considering a sufficiently large $x \geqslant 0$. But then we have $w(\rho) = \pi_1 \cdot \lambda^p \cdot \pi_2 = \nu_1 \cdot (\alpha \cdot \beta)^{p-1} \alpha \cdot \pi_2$, for some $p > 0$, in which case $\mathcal{L}(Fold_{\Pi}(\rho)) = \nu_1 \cdot \mu^* \cdot \alpha \cdot \pi_2$, which leads to a contradiction (the first repeating state on ρ must be the last state occurring on π_1, instead of the last one on ν_1, which clearly occurs earlier). Hence, we have that $|\pi_1| = |\nu_1|$, and consequently $|\pi_2| = |\nu_2|$, which leads to $\pi_1 = \nu_1$, $\lambda = \mu$ and $\pi_2 = \nu_2$.

Finally, since $w(\rho) = u \cdot (\diamond, s)$, we have $\pi_2 = v \cdot (\diamond, s)$, for some word v. We recall also that $w(\tau) = u \cdot (\sigma, s)(\diamond, t)$. There are two possibilities:

- π_2 does not have a symbol of the form (σ', t), and either does ν_2, hence:

$$\begin{aligned}\mathcal{L}(Fold_{\Pi}(\tau)) &= \{u \cdot (\sigma, s)(\diamond, t) \mid u \cdot (\diamond, s) \in \mathcal{L}(Fold_{\Pi}(\rho))\} \\ &\subseteq \{u' \cdot (\sigma, s)(\diamond, t) \mid u' \cdot (\diamond, s) \in \mathcal{L}(Fold_{\Pi}(\rho'))\} \\ &= \mathcal{L}(Fold_{\Pi}(\tau'))\end{aligned}$$

- $\pi_2 = \nu_2 = u \cdot (\sigma', t) \cdot v \cdot (\diamond, s)$, hence:

$$\begin{aligned}&\mathcal{L}(Fold_{\Pi}(\tau)) \\ &= \left\{w \cdot ((\sigma', t) \cdot v \cdot (\sigma, s))^k \cdot (\diamond, t) \mid w \cdot (\sigma', t) \cdot v \cdot (\diamond, s) \in \mathcal{L}(Fold_{\Pi}(\rho)), k \geqslant 0\right\} \\ &\subseteq \left\{w' \cdot ((\sigma', t) \cdot v \cdot (\sigma, s))^k \cdot (\diamond, t) \mid w' \cdot (\sigma', t) \cdot v \cdot (\diamond, s) \in \mathcal{L}(Fold_{\Pi}(\rho')), k \geqslant 0\right\} \\ &= \mathcal{L}(Fold_{\Pi}(\tau'))\end{aligned}$$

□

Proof of Lemma 5

Proof. Let τ be a path such that $w(\tau) \in \mathcal{L}(Fold_{\Pi}(\rho))$, and $\rho_{i,n}$ occurs at least once as the suffix of τ. Hence τ can be factorized as $\tau = \mu \cdot \rho_{i,n}^m$, where $w(\mu) \in$

$\mathcal{L}(\mathit{Fold}_{\Pi}(\rho_{1,i}))$, for some $m > 0$. Let $\overline{\nu}$ be a sequence of valuations of $\mathbf{x}$ such that $\overline{\nu} \models_{\mathrm{Th}(\mathcal{D})} \Upsilon(\Pi(A_\tau))$, i.e. when substituting each tuple $\mathbf{x}^i$ by $\overline{\nu}_i(\mathbf{x})$ in $\Upsilon(\Pi(A_\tau))$, for all $i \in [0, |\tau|]$. We must show that $\overline{\nu} \models_{\mathrm{Th}(\mathcal{D})} \Upsilon(\Pi(B_\tau))$. Since $\overline{\nu}$ was chosen arbitrarily, we have $\Upsilon(\Pi(A_\tau)) \rightarrow \Upsilon(\Pi(B_\tau))$, i.e. $\neg \mathit{Accept}_{\Pi}(\tau)$.

Without losing generality, we assume that $|\rho_{i,n}| = 1$ – the general case has a similar argument. In this case, we have $\Upsilon(\Pi(A_{\rho_{i,n}})) \Leftrightarrow \Xi(\Pi(A_{\rho_{i,n}}))$ and $\Upsilon(\Pi(B_{\rho_{i,n}})) \Leftrightarrow \Xi(\Pi(B_{\rho_{i,n}}))$, hence we can use these formulae interchangeably.

We denote $\overline{\upsilon} = \overline{\nu}_{1,i}$, $\upsilon_0 = \overline{\nu}_i$, $\upsilon_1 = \overline{\nu}_{i+1}, \ldots, \upsilon_m = \overline{\nu}_{|\overline{\nu}|}$. Then we have:

$$\upsilon_0, \ldots, \upsilon_m \models_{\mathrm{Th}(\mathcal{D})} \left(\Upsilon(\Pi(A_{\rho_{i,n}}))\right)^{\wedge m}$$

and hence $\upsilon_\ell, \upsilon_{\ell+1} \models_{\mathrm{Th}(\mathcal{D})} \Xi(\Pi(A_{\rho_{i,n}}))$, for each $\ell \in [0, m-1]$. By the fact that $\Xi(\Pi(A_{\rho_{i,n}}))$ is reductive, we obtain that $\upsilon_0, \upsilon_m \models_{\mathrm{Th}(\mathcal{D})} \Xi(\Pi(A_{\rho_{i,n}}))$, and thus $\upsilon_0, \upsilon_m \models_{\mathrm{Th}(\mathcal{D})} \Upsilon(\Pi(A_{\rho_{i,n}}))$. We have thus:

$$\overline{\upsilon}, \upsilon_0, \upsilon_m \models_{\mathrm{Th}(\mathcal{D})} \Upsilon(\Pi(A_{\rho_{1,i} \cdot \rho_{i,n}}))$$

and, by the hypothesis (1), we obtain:

$$\overline{\upsilon}, \upsilon_0, \upsilon_m \models_{\mathrm{Th}(\mathcal{D})} \Upsilon(\Pi(B_{\rho_{1,i} \cdot \rho_{i,n}}))$$

since $w(\rho_{1,i} \cdot \rho_{i,n}) = w(\rho_{1,i}) \cdot w(\rho_{i,n}) \in \mathcal{L}(\mathit{Fold}_{\Pi}(\rho_{1,i})) \cdot w(\rho_{i,n})$, which implies $\neg \mathit{Accept}_{\Pi}(\rho_{1,i} \cdot \rho_{i,n})$. By the extensivity of $\Xi(\Pi(B_{\rho_{i,n}}))$, we obtain a valuation υ' such that $\upsilon_0, \upsilon' \models_{\mathrm{Th}(\mathcal{D})} \Upsilon(\Pi(B_{\rho_{i,n}}))$ and $\upsilon', \upsilon_m \models_{\mathrm{Th}(\mathcal{D})} \Upsilon(\Pi(B_{\rho_{i,n}}))$.

It easy to check that $\upsilon_0 \models_{\mathrm{Th}(\mathcal{D})} [\exists \mathbf{x} \,.\, \mathcal{P}_\rho(i_1, \ldots, i_p)][\mathbf{x}/\mathbf{x}']$, i.e. υ_0 is in the post-image of the relation $\mathcal{P}_\rho(i_1, \ldots, i_p)$. But then we have:

$$\upsilon_0, \upsilon_{m-1}, \upsilon_m \models_{\mathrm{Th}(\mathcal{D})} [\exists \mathbf{x} \,.\, \mathcal{P}_\rho(i_1, \ldots, i_p)][\mathbf{x}^{\mathbf{0}}/\mathbf{x}'] \wedge \left(\Upsilon(\Pi(A_{\rho_{i,n}}))\right)^{\wedge 2}$$

hence, by the hypothesis (2), we obtain:

$$\upsilon_0, \upsilon_{m-1}, \upsilon_m \models_{\mathrm{Th}(\mathcal{D})} \left(\Upsilon(\Pi(B_{\rho_{i,n}}))\right)^{\wedge 2}$$

We have just showed that:

$$\overline{\upsilon}, \upsilon_0, \upsilon_{m-1}, \upsilon_m \models_{\mathrm{Th}(\mathcal{D})} \left(\Upsilon(\Pi(B_{\rho_{1,i} \cdot \rho_{i,n} \cdot \rho_{i,n}}))\right)$$

Applying the same reasoning inductively, we can show that

$$\overline{\upsilon}, \upsilon_0, \ldots, \upsilon_m \models_{\mathrm{Th}(\mathcal{D})} \left(\Upsilon(\Pi(B_{\rho_{1,i} \cdot \rho_{i,n}^m}))\right) \quad .$$

that is, $\overline{\nu} \models_{\mathrm{Th}(\mathcal{D})} \Upsilon(\Pi(B_\tau))$, which concludes the proof. □

Proof of Lemma 6

Lemma 8. *If $\sqsubseteq$ is a subsumption relation, $\rho \sqsubseteq \rho'$ are paths, and $\Pi : Q \times Q \rightarrow 2^{\mathrm{Th}(\mathcal{D})}$ is a predicate map, we have $\mathit{Dist}_{\Pi}(\rho) \geqslant \mathit{Dist}_{\Pi}(\rho')$.*

Proof. Suppose $Dist_{\Pi}(\rho) = k < \infty$, otherwise there is nothing to prove. Let $\tau_0, \tau_1, \ldots, \tau_k$ be a sequence of paths, such that $\tau_0 = \rho$, $Accept_{\Pi}(\tau_k)$ and $\tau_{i+1} \in Post_{\Pi}(\tau_i)$, for all $i \in [0, k-1]$. Applying Def. 3 inductively, we prove the existence of a sequence $\tau'_0, \tau'_1, \ldots, \tau'_k$ such that $\tau'_0 = \rho'$, $Accept_{\Pi}(\tau'_k)$ and $\tau'_{i+1} \in Post_{\Pi}(\tau'_i)$, for all $i \in [0, k-1]$. Then $Dist_{\Pi}(\rho) \geqslant Dist_{\Pi}(\rho')$, because for any sequence originating in ρ and leading to an accepting path, there exists a sequence of the same length originating in ρ' and leading also to an accepting path. □

Lemma 9. *Let ρ be a path and $\Pi : Q \times Q \to 2^{Th(\mathcal{D})}$ be a predicate map. Then the following hold:*

1. *if $Dist_{\Pi}(\rho) < \infty$ then $Dist_{\Pi}(\tau) < Dist_{\Pi}(\rho)$, for some $\tau \in Post_{\Pi}(\rho)$.*
2. *if $Dist_{\Pi}(\rho) = \infty$ then $Dist_{\Pi}(\tau) = \infty$, for all $\tau \in Post_{\Pi}(\rho)$.*

Proof. It is easy to show that $Dist_{\Pi}(\rho) = 1 + \min\{Dist(\tau) \mid t \in Post_{\Pi}(\rho)\}$, where, by convention, we consider that $1 + \infty = \infty$. For point (1), if $Dist_{\Pi}(\rho) < \infty$, let $\tau \in Post_{\Pi}(\rho)$ be the successor of τ such that $Dist_{\Pi}(\rho) = 1 + Dist_{\Pi}(\tau)$. Then $Dist_{\Pi}(\tau) < Dist_{\Pi}(\rho)$. For point (2) if $Dist_{\Pi}(\rho) = \infty$ this implies the fact that $\min\{Dist(\tau) \mid \tau \in Post_{\Pi}(\rho)\} = \infty$, hence $Dist_{\Pi}(\tau) = \infty$, for all $\tau \in Post_{\Pi}(\rho)$. □

Lemma 10. *Let $\Pi_1 \subseteq \Pi_2$ be two predicate maps, and ρ be a path, such that $Dist_{\Pi_2}(\rho) < \infty$. Then $Dist_{\Pi_1}(\rho) \leqslant Dist_{\Pi_2}(\rho)$.*

Proof. If $Dist_{\Pi_2}(\rho) < \infty$, there exists a sequence of paths $\tau_0, \ldots, \tau_k$, such that $\rho = \tau_0$, $Accept_{\Pi_2}(\tau_k)$ and $\tau_{i+1} \in Post_{\Pi_2}(\tau_i)$, for all $i \in [0, k-1]$. Since $\tau_1 \in Post_{\Pi_2}(\rho)$, by Lemma 2, $\rho_1 \in Post_{\Pi_1}(\rho)$ as well. We apply this argument inductively, for all $\tau_{i+1} \in Post_{\Pi_2}(\tau_i)$ and show that $\tau_{i+1} \in Post_{\Pi_1}(\tau_i)$, for all $i \in [1, k-1]$. Since $Accept_{\Pi_2}(\tau_k)$, this implies $Accept_{\Pi_1}(\tau_k)$ (by induction on the height of the path trees A_{τ_k} and B_{τ_k}, we prove that $\Upsilon(\Pi_2(A_{\tau_k})) \to \Upsilon(\Pi_1(A_{\tau_k}))$ and $\Upsilon(\Pi_1(B_{\tau_k})) \to \Upsilon(\Pi_2(B_{\tau_k}))$, respectively). Then $Dist_{\Pi_2}(\rho) \geqslant Dist_{\Pi_1}(\rho)$, since for every accepting continuation of ρ under Π_2, there exists an accepting continuation of ρ under Π_1 of the same length. However, there might exist shorter accepting continuations under Π_1 but not under Π_2. □

Back to the proof of Lemma 6:

Proof. Initially $\texttt{Visited} = \varnothing$, hence $Dist_{\Pi}(\texttt{Visited}) = \infty$, and $\texttt{Next} = Roots$, hence $Dist_{\Pi}(\texttt{Next}) = Dist_{\Pi}(Roots)$. Then all three invariants hold trivially the first time the control reaches line 4. We denote:

$$\mathcal{A}_{old} = \langle \Pi_{old}, \texttt{Visited}_{old}, \texttt{Next}_{old}, \texttt{Edges}_{old}, \texttt{Subsume}_{old} \rangle$$
$$\mathcal{A}_{new} = \langle \Pi_{new}, \texttt{Visited}_{new}, \texttt{Next}_{new}, \texttt{Edges}_{new}, \texttt{Subsume}_{new} \rangle$$

the configuration of the APRT at two arbitrary consecutive visits of line 4, and assume that all invariants hold for $\mathcal{A}_{old}$.

(Inv_1) $\mathcal{A}_{old}$ can be modified in the following ways:

(i) if the control reaches line 6, the first change is the update of the predicate map to a predicate map $\Pi_{new} \supseteq \Pi_{old}$ (line 12). The next configuration of the APRT is $\mathcal{A}' = \langle \Pi_{new}, \texttt{Visited}_{old}, \texttt{Next}_{old}, \texttt{Edges}_{old}, \texttt{Subsume}_{old} \rangle$. We show first that $\mathcal{A}'$ is closed. Let $s \in \texttt{Visited}_{old}$ and $t \in Post_{\Pi_{new}}(s)$ be a successor of s. Then, by Lemma 2, we have $t \in Post_{\Pi_{old}}(s)$. Moreover, since $s \in \texttt{Visited}_{old}$ and $\mathcal{A}_{old}$ is closed, there exists an edge $(s, u) \in \texttt{Edges}_{old} \cup \texttt{Subsume}_{old}$ such that $t \sqsubseteq u$. Hence $\mathcal{A}'$ is closed.
Second, if $s_p \in \texttt{Visited}_{old}$ is the pivot state of the current path ($s_p = src(\rho_k)$, on line 9), $\mathcal{A}_{new}$ is obtained from $\mathcal{A}'$ by the following successive transformations:
(a) let $T = \text{SUBTREE}(s_p)$
(b) move the set of nodes $S = \{s \mid \exists t \in T \,.\, (s, t) \in \texttt{Subsume}_{old}\}$ from $\texttt{Visited}$ to $\texttt{Next}$ (lines 13-14),
(c) remove T from $\texttt{Visited} \cup \texttt{Next}$ (line 15),
(d) add s_p to $\texttt{Next}$ (line 16).
To prove that $\mathcal{A}_{new}$ is closed, let $s \in \texttt{Visited}_{new} = \texttt{Visited}_{old} \setminus (S \cup T)$, and $t \in Post_{\Pi_{new}}(s)$. Then, by Lemma 2, $t \in Post_{\Pi_{old}}(s)$ as well. Since $\mathcal{A}_{old}$ is closed, there exists an edge $(s, u) \in \texttt{Edges}_{old} \cup \texttt{Subsume}_{old}$ such that $t \sqsubseteq u$. It is thus sufficient to show that $(s, u) \in \texttt{Edges}_{new} \cup \texttt{Subsume}_{new}$. To this end, we distinguish two cases:
(a) if $(s, u) \in \texttt{Edges}_{old}$, again, we distinguish two cases. If $u \in T$, since $s \notin T$, the only possibility is $u = s_p$ and s is the predecessor of u in $\mathcal{A}_{old}$. But then we have $(s, u) \in \texttt{Edges}_{new}$, because this edge was inserted at line 18. Otherwise, if $u \notin T$, we have $(s, u) \in \texttt{Edges}_{new} \cup \texttt{Subsume}_{new}$, because no edge with destination u is removed from $\texttt{Edges}_{old} \cup \texttt{Subsume}_{old}$.
(b) otherwise, if $(s, u) \in \texttt{Subsume}_{old}$, the only possibility is that $u \notin T$, hence $u \in \texttt{Visited}_{new} \cup \texttt{Next}_{new}$ and $(s, u) \in \texttt{Edges}_{new} \cup \texttt{Subsume}_{new}$. The reason for this is that, supposing $u \in T$ implies that s is inserted in $\texttt{Next}_{new}$ (line 14), which contradicts the hypothesis $s \in \texttt{Visited}_{new}$, since $\texttt{Visited}_{new} \cap \texttt{Next}_{new} = \emptyset$.

(ii) if the control reaches line 23, we have $\Pi_{new} = \Pi_{old}$ and $\texttt{Visited}_{new} = \texttt{Visited}_{old} \cup \{s_0\}$, where s_0 is the APS picked at line 6. Let $s \in \texttt{Visited}_{new}$ be an APS and $t \in Post_{\Pi_{new}}(s, \sigma)$ be one of its successors. If $s \neq s_0$, then there exists an edge $(s, u) \in \texttt{Edges}_{old} \cup \texttt{Subsume}_{old}$ such that $t \sqsubseteq u$, since $\mathcal{A}_{old}$ is closed. Then $u \notin \texttt{rem}$, and $(s, u) \in \texttt{Edges}_{new} \cup \texttt{Subsume}_{new}$, since only edges ending in states from $\texttt{rem}$ are removed in this case (line 34). Otherwise, if $s = s_0$, then t is covered either by the edge $(s, u) \in \texttt{Subsume}_{new}$, where $t \sqsubseteq u$ (line 26), or by $(s, t) \in \texttt{Edge}_{new}$ (line 36).

(Inv_2) We consider only the case $Dist_{\Pi_{new}}(Roots) < \infty$, since there is nothing to prove if $Dist_{\Pi_{new}}(Roots) = \infty$. Since $\Pi_{new} \supseteq \Pi_{old}$, by Lemma 10, we have that $Dist_{\Pi_{old}}(Roots) < \infty$ as well. By the hypothesis, we have $Dist_{\Pi_{old}}(\texttt{Visited}_{old}) > Dist_{\Pi_{old}}(\texttt{Next}_{old})$. Then clearly, $Dist_{\Pi_{old}}(\texttt{Next}_{old}) < \infty$. We distinguish two cases:

(i) if $Dist_{\Pi_{new}}(\texttt{Visited}_{new}) = \infty$, then it must be that $Dist_{\Pi_{new}}(\texttt{Next}_{new}) < \infty$, in order to have $Dist_{\Pi_{new}}(\texttt{Visited}_{new}) > Dist_{\Pi_{new}}(\texttt{Next}_{new})$. To show this latter fact, suppose, by contradiction, that $Dist_{\Pi_{new}}(\texttt{Next}_{new}) = \infty$. Since $Roots \subseteq \texttt{Visited}_{new} \cup \texttt{Next}_{new}$, we have then $Dist_{\Pi_{new}}(Roots) \geqslant Dist_{\Pi_{new}}(\texttt{Visited}_{new} \cup \texttt{Next}_{new}) = \infty$, which leads to a contradiction.

(ii) otherwise, $Dist_{\Pi_{new}}(\texttt{Visited}_{new}) < \infty$ and, by (Inv_1), we have that $\mathcal{A}_{new}$ is closed. Let $s \in \texttt{Visited}_{new}$ be a node. Since $Dist_{\Pi_{new}}(s) < \infty$, there exists a sequence of nodes $s_0, \ldots, s_k$, such that $s = s_0$, $s_{i+1} \in Post_{\Pi_{new}}(s_i)$, for all $i \in [0, k-1]$, and $Accept_{\Pi_{new}}(s_k)$. Since $\mathcal{A}_{new}$ is closed, there exists $t_1 \in \texttt{Visited}_{new} \cup \texttt{Next}_{new}$ such that $(s_0, t_1) \in \texttt{Edges}_{new} \cup \texttt{Subsume}_{new}$ and $s_1 \sqsubseteq t_1$. By Def. 3 (b), there exists $s_2' \in Post_{\Pi_{new}}(t_1)$ such that $s_2 \sqsubseteq s_2'$, and since $\mathcal{A}_{new}$ is closed, there exists $t_2 \in \texttt{Visited}_{new} \cup \texttt{Next}_{new}$ such that $(t_1, t_2) \in \texttt{Edges}_{new} \cup \texttt{Subsume}_{new}$ and $s_2' \sqsubseteq t_2$, hence we obtain $s_2 \sqsubseteq t_2$, by transitivity of $\sqsubseteq$. Applying this argument inductively, we obtain a sequence $t_0, \ldots, t_k$ in $\mathcal{A}_{new}$ such that $s_0 = t_0$ and $(t_i, t_{i+1}) \in \texttt{Edges}_{new} \cup \texttt{Subsume}_{new}$, for all $i \in [1, k-1]$. Moreover, we also have that $s_i \sqsubseteq t_i$, and, by Lemma 9 (1) $\infty > Dist_{\Pi_{new}}(s_i) \geqslant Dist_{\Pi_{new}}(t_i)$, for all $i \in [1, k]$. In particular, we also have $Accept_{\Pi_{new}}(t_k)$, by Def. 3 (a) and the fact that $Accept_{\Pi_{new}}(s_k)$. But then $t_k \notin \texttt{Visited}_{new}$ (accepting nodes are never added to $\texttt{Visited}$), therefore there exists a node $t_\ell \in \texttt{Next}_{new}$ for some $\ell \in [1, k]$. We compute:

$$\begin{aligned} Dist_{\Pi_{new}}(\texttt{Visited}_{new}) &= Dist_{\Pi_{new}}(s) \\ &> Dist_{\Pi_{new}}(s_\ell) \\ &\geqslant Dist_{\Pi_{new}}(t_\ell) \\ &\geqslant Dist_{\Pi_{new}}(\texttt{Next}_{new}) \ . \end{aligned}$$

□

Proof of Lemma 7

Proof. If $\mathcal{L}(A) \nsubseteq \mathcal{L}(B)$, there exists a trace $(\sigma_1, \nu_1) \ldots (\sigma_n, \nu_n)(\diamond, \nu_{n+1}) \in \mathcal{L}(A) \backslash \mathcal{L}(B)$. Since $(\sigma_1, \nu_1) \ldots (\sigma_n, \nu_n)(\diamond, \nu_{n+1}) \in \mathcal{L}(A)$, then A has a run that produces it. We extend this run into a path ρ by considering all possible runs of B on the word $\sigma_1 \ldots \sigma_n$. Then clearly $\nu_1, \ldots, \nu_n, \nu_{n+1} \models_{\mathrm{Th}(\mathcal{D})} cex(A_\rho, B_\rho)$. Since for every predicate map Π, we have $\Upsilon(A_\rho) \rightarrow \Upsilon(\Pi(A_\rho))$ and $\Upsilon(\Pi(B_\rho)) \rightarrow \Upsilon(B_\rho)$, we have $\nu_1, \ldots, \nu_n, \nu_{n+1} \models_{\mathrm{Th}(\mathcal{D})} cex(\Pi(A_\rho), \Pi(B_\rho))$ as well, hence $Accept_\Pi(\rho)$. □

Proof of Thm. 2

Proof. If Algorithm 1 terminates returning **true**, then $\texttt{Next} = \varnothing$, i.e. $Dist_\Pi(\texttt{Next}) = \infty$, and $Dist_\Pi(\texttt{Visited}) > Dist_\Pi(\texttt{Next})$ does not hold. Due to Lemma 6 (Inv_2), we obtain $Dist_\Pi(Roots) = \infty$. Suppose now, by contradiction, that $\mathcal{L}(A) \nsubseteq \mathcal{L}(B)$, i.e. there exists a trace $\tau \in \mathcal{L}(A) \backslash \mathcal{L}(B)$. By Lemma 7, there exists a finite path ρ, such that $Accept_\Pi(\rho)$, for any predicate map Π. But then $Dist_\Pi(Roots) < \infty$, contradiction. □