On an Invariance Problem for Parameterized Concurrent Systems

Marius Bozga
Univ. Grenoble Alpes, CNRS, Grenoble INP, VERIMAG, 38000, France

Lucas Bueri
Univ. Grenoble Alpes, CNRS, Grenoble INP, VERIMAG, 38000, France

Radu Iosif
Univ. Grenoble Alpes, CNRS, Grenoble INP, VERIMAG, 38000, France

Abstract

We consider concurrent systems consisting of replicated finite-state processes that synchronize via joint interactions in a network with user-defined topology. The system is specified using a resource logic with a multiplicative connective and inductively defined predicates, reminiscent of Separation Logic [19]. The problem we consider is if a given formula in this logic defines an invariant, namely whether any model of the formula, following an arbitrary firing sequence of interactions, is transformed into another model of the same formula. This property, called *havoc invariance*, is quintessential in proving the correctness of reconfiguration programs that change the structure of the network at runtime. We show that the havoc invariance problem is many-one reducible to the entailment problem $\phi \models \psi$, asking if any model of ϕ is also a model of ψ. Although, in general, havoc invariance is found to be undecidable, this reduction allows to prove that havoc invariance is in 2EXP, for a general fragment of the logic, with a 2EXP entailment problem.

2012 ACM Subject Classification Software and its engineering → Formal software verification

Keywords and phrases parameterized verification, invariant checking, resource logics, reconfigurable systems, tree automata

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2022.24

Related Version *Full Version*: https://arxiv.org/abs/2204.12117

1 Introduction

The parameterized verification problem asks to decide whether a system consisting of an arbitrary number of finite-state processes that communicate via synchronized (joint) actions satisfies a specification, such as deadlock freedom, mutual exclusion or a temporal logic property e.g., every request is eventually answered. The literature in this area has a wealth of decidability and complexity results (see [3] for a survey) classified according to the communication type (e.g., rendez-vous, broadcast) and the network topology e.g., rings where every process interacts with its left/right neighbours, cliques where each two process may interact, stars with a controller interacting with unboundedly many workers, etc.

As modern computing systems are dynamically adaptive, recent effort has been put into designing *reconfigurable systems*, whose network topologies change at runtime (see [12] for a survey) in order to address maintenance (e.g., replacement of faulty and obsolete components by new ones, firmware updates, etc.) and internal traffic issues (e.g., re-routing to avoid congestion in a datacenter [18]). Unfortunately the verification of dynamic reconfigurable systems (i.e., proving the absence of design errors) remains largely unexplored. Consequently, such systems are prone to bugs that may result in e.g., denial of services or data corruption[1].

[1] Google reports on a cascading cloud failure due to reconfiguration: https://status.cloud.google.com/incident/appengine/19007.

Proving correctness of parameterized reconfigurable networks is tackled in [1], where a Hoare-style program logic is proposed to write proofs of reconfiguration programs i.e., programs that dynamically add and remove processes and interactions from the network during runtime. The assertion language used by these proofs is a logic that describes sets of configurations defining the network topology and the local states of the processes. The logic views processes and interactions as resources that can be joined via a *separating conjunction*, in the spirit of Separation Logic [19]. The separating conjunction supports *local reasoning*, which is the ability of describing reconfigurations *only with respect to those components and interactions that are involved in the mutation*, while disregarding the rest of the system's configuration. Moreover, the separating conjunction allows to concisely describe networks of unbounded size, that share a similar architectural style (e.g., pipelines, rings, stars, trees) by means of inductively defined predicates.

Due to the interleaving of reconfigurations and interactions between components, the annotations of the reconfiguration program form a valid proof under so-called *havoc invariance* assumptions, stating global properties about the configurations, that remain, moreover, *unchanged under the ongoing interactions in the system.* These assumptions are needed to apply the sequential composition rule that infers a Hoare triple $\{\phi\}\ \mathsf{P};\mathsf{Q}\ \{\psi\}$ from two premisses $\{\phi\}\ \mathsf{P}\ \{\theta\}$ and $\{\theta\}\ \mathsf{Q}\ \{\psi\}$, where P and Q are reconfiguration actions that add and/or remove processes and communication channels. Essentially, because the states of the processes described by the intermediate assertion θ might change between the end of P and the beginning of Q, this rule is sound provided that θ is a havoc invariant formula.

This paper contributes to the automated generation of reconfiguration proofs, by a giving a procedure that discharges the havoc invariance side conditions. The challenge is that a formula of the configuration logic (that contains inductively defined predicates) describes an infinite set of configurations of arbitrary sizes. The main result is that the havoc invariance problem is effectively many-one reducible to the entailment problem $\phi \models \psi$, that asks if every model of a formula ϕ is a model of another formula ψ. Here ψ is the formula whose havoc invariance is being checked and ϕ defines the set of configurations γ' obtained from a model γ of ψ, by executing one interaction from γ. The reduction is polynomial if certain parameters are bounded by a constant (i.e., the arity of the predicates, the size of interactions and the number of predicate atoms is an inductive rule), providing a 2EXP upper bound for a fragment of the logic with a decidable (2EXP) entailment problem [4, §6]. Having a polynomial reduction motivates, moreover, future work on the definition of fragments of lower (e.g., polynomial) entailment complexity (see e.g., [9] for a fragment of Separation Logic with a polynomial entailment problem), that are likely to yield efficient decision procedures for the havoc invariance problem as well. In addition, we provide a 2EXP-hard lower bound for the havoc invariance problem in this fragment of the logic (i.e., assuming predicates of unbounded arity) and show that havoc invariance is undecidable, when unrestricted formulæ are considered as input.

Related Work. Specifying parameterized concurrent systems by inductive definitions is reminiscent of *network grammars* [20, 16, 13], that use inductive rules to describe systems with linear (pipeline, token-ring) architectures obtained by composition of an unbounded number of processes. In contrast, we use predicates of unrestricted arities to describe network topologies that can be, in general, more complex than trees. Moreover, we write inductive definitions using a resource logic, suitable also for writing Hoare logic proofs of reconfiguration programs, based on local reasoning [8].

Verification of network grammars against safety properties (unreachability of error configurations) requires the synthesis of *network invariants* [21], computed by rather costly fixpoint iterations [17] or by abstracting (forgetting the particular values of indices in) the composition of a small bounded number of instances [14]. In previous work, we have developped an invariant synthesis method based on *structural invariants*, that are synthesized with little computational effort and prove to be efficient in many practical examples [5, 6].

The havoc invariance problem considered in this paper is, however, different from safety checking and has not been addressed before, to the best of our knowledge. An explanation is that verification of reconfigurable systems has received fairly scant attention, relying mostly on runtime verification [7, 10, 15, 11], instead of deductive verification, reported in [1]. In [1] we addressed havoc invariance with a set of inference rules used to write proofs manually, whereas the goal of this paper is to discharge such conditions automatically.

1.1 A Motivating Example

Consider, for instance, a system consisting of a finite but unbounded number of processes, called *components* in the following. The components execute the same machine with states T and H, denoting whether the component has a token (T) or a hole (H). The components are placed in a ring, each component having exactly one left and one right neighbour, as in Fig. 1 (a). A component without a token may receive one, by executing a transition $\mathsf{H} \xrightarrow{in} \mathsf{T}$, simultaneously with its left neighbour, that executes the transition $\mathsf{T} \xrightarrow{out} \mathsf{H}$, as in Fig. 1 (a). Note that there can be more than one token, moving independently in the system, such that no token overtakes another token. The configurations of the token ring system are described by the following inductive rules:

$$\mathsf{ring}_{h,t}() \leftarrow \exists x \exists y \ . \ \langle x.out, y.in \rangle * \mathsf{chain}_{h,t}(y, x)$$

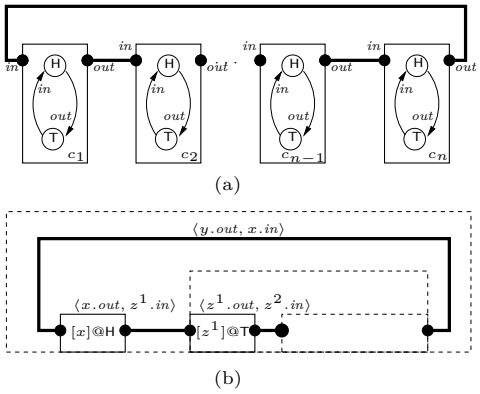
$$\mathsf{chain}_{h,t}(x, y) \leftarrow \exists z. \ [x]@q * \langle x.out, z.in \rangle * \mathsf{chain}_{h',t'}(z, y), \text{ for both } q \in \{\mathsf{H}, \mathsf{T}\}$$

$$\mathsf{chain}_{0,1}(x, x) \leftarrow [x]@\mathsf{T} \qquad \mathsf{chain}_{1,0}(x, x) \leftarrow [x]@\mathsf{H} \qquad \mathsf{chain}_{0,0}(x, x) \leftarrow [x]$$

$$\text{where } h' \stackrel{\text{def}}{=} \begin{cases} \max(h-1, 0) & \text{, if } q = \mathsf{H} \\ h & \text{, if } q = \mathsf{T} \end{cases} \text{ and } t' \stackrel{\text{def}}{=} \begin{cases} \max(t-1, 0) & \text{, if } q = \mathsf{T} \\ t & \text{, if } q = \mathsf{H} \end{cases}$$

The predicate $\mathsf{ring}_{h,t}()$ describes a ring with at least h (resp. t) components in state H (resp. T). The ring consists of an interaction between the ports *out* and *in* of two components x and y, respectively, described by $\langle x.out, y.in \rangle$ and a separate chain of components between x and y, described by $\mathsf{chain}_{h,t}(y, x)$. Inductively, a chain consists of a component $[x]@q$ in state $q \in \{\mathsf{H}, \mathsf{T}\}$, an interaction $\langle x.out, z.in \rangle$ and a separate $\mathsf{chain}_{h',t'}(z, y)$, where h' and t' are the least numbers of components in state H and T, respectively, after the removal of the component x. Fig. 1 (b) depicts the unfolding of the inductive definition of $\mathsf{ring}_{h,t}()$ with the existentially quantified variables z from the above rules α-renamed to z^1, z^2, etc.

A *reconfiguration action* is an atomic creation or deletion of a component or interaction. A *reconfiguration sequence* is a finite sequence of reconfiguration actions that takes as input a mapping of program variables to components and executes the actions from the sequence, in interleaving with the interactions in the system. For instance, the reconfiguration sequence from Fig. 1 (c) takes as input the mapping of `x` and `y` to two adjacent components in the token ring, removes the interaction $\langle x.out, y.in \rangle$ by executing $\texttt{disconnect}(\texttt{x}.out, \texttt{y}.in)$ and creates a new component in state H (by executing $\texttt{new}(\texttt{x}, \mathsf{H})$) that is connected in between `x` and `y` via two new interactions created by executing $\texttt{connect}(\texttt{z}.out, \texttt{y}.in)$ and $\texttt{connect}(\texttt{x}.out, \texttt{z}.in)$, respectively. Fig. 1 (c) shows a proof (with annotations in curly braces) of the fact that the outcome of the reconfiguration of a ring of components is a ring whose least number of components in state H is increased from one to two. This proof is split into several subgoals:



$\{\mathsf{ring}_{1,1}()\}$
$\{\langle x.out, y.in \rangle * \mathsf{chain}_{1,1}(y,x)\}$
`disconnect(x.`out`, y.`in`);`
$\{\mathsf{chain}_{1,1}(y,x)\}$ (‡)
`new(H,z);`
$\{z@\mathsf{H} * \mathsf{chain}_{1,1}(y,x)\}$ (‡)
`connect(z.`out`,y.`in`);`
$\{z@\mathsf{H} * \langle z.out, y.in \rangle * \mathsf{chain}_{1,1}(y,x)\}$
$\{\mathsf{chain}_{2,1}(z,x)\}$ (‡)
`connect(x.`out`,z.`in`)`
$\{\mathsf{chain}_{2,1}(z,x) * \langle x.out, z.in \rangle\}$
$\{\mathsf{ring}_{2,1}()\}$

(c)

Figure 1 Inductive Specification and Reconfiguration of a Token Ring.

1. Entailments required to apply the consequence rule of Hoare logic e.g., $\mathsf{ring}_{1,1}() \models \exists x \exists y \ . \ \langle x.out, y.in \rangle * \mathsf{chain}_{1,1}(y,x)$. The entailment problem has been addressed in [4, §6], with the definition of a general fragment of the configuration logic, for which the entailment problem is decidable in double exponential time.
2. Hoare triples that describe the effect of the atomic reconfiguration actions e.g., $\{\langle x.out, y.in \rangle * \mathsf{chain}_{1,1}(y,x)\}\texttt{disconnect}(\texttt{x}.out, \texttt{y}.in)\{\mathsf{chain}_{1,1}(y,x)\}$. These are obtained by applying the frame rule to the local[2] specifications of the atomic actions. The local specification of reconfiguration actions and the frame rule for local actions are described in [1, §4.2].
3. *Havoc invariance* proofs for the annotations marked with (‡) in Fig. 1 (c). For instance, the formula $\mathsf{chain}_{1,1}(y,x)$ is havoc invariant because the interactions in a chain of components will only move tokens to the right without creating more or losing any, hence there will be the same number of components in state H (T) no matter which interactions are fired.

2 Definitions

We denote by $\mathbb{N}$ the set of positive integers, including zero. For a set A, we denote $A^1 \stackrel{\text{def}}{=} A$, $A^{i+1} \stackrel{\text{def}}{=} A^i \times A$, for all $i \geq 0$, where $\times$ denotes the Cartesian product, and $A^+ \stackrel{\text{def}}{=} \bigcup_{i \geq 1} A^i$. The cardinality of a finite set A is denoted by $\|A\|$. By writing $A \subseteq_{fin} B$ we mean that A is a finite subset of B. Given integers i and j, we write $[i,j]$ for the set $\{i, i+1, \ldots, j\}$, assumed to be empty if $i > j$. For a function $f : A \to B$, we denote by $f[a_i \leftarrow b_i]_{i \in [1,n]}$ the function that maps a_i into b_i for each $i \in [1,n]$ and agrees with f everywhere else.

2.1 Configurations

We model a parallel system as a hypergraph, whose vertices are *components* (i.e., the nodes of the network) and hyperedges are *interactions* (i.e., describing the way the components communicate with each other). The components are taken from a countably infinite set $\mathbb{C}$, called the *universe*. We consider that each component executes its own copy of the same *behavior*, represented as a finite-state machine $\mathbb{B} = (\mathcal{P}, \mathcal{Q}, \rightarrow)$, where $\mathcal{P}$ is a finite set of

[2] A Hoare triple $\{\phi\}$ P $\{\psi\}$ is *local* if it mentions only those components and interactions added or deleted by P. Local specifications are plugged into a global context by the *frame rule* that infers $\{\phi * F\}$ P $\{\psi * F\}$ from $\{\phi\}$ P $\{\psi\}$ if the variables modified by P are not free in F.

ports, $\mathcal{Q}$ is a finite set of *states* and $\rightarrow \subseteq \mathcal{Q} \times \mathcal{P} \times \mathcal{Q}$ is a transition relation. Intuitively, each transition $q \xrightarrow{p} q'$ of the behavior $\mathbb{B}$ is triggerred by a visible event, represented by the port p. The universe $\mathbb{C}$ and the behavior $\mathbb{B} = (\mathcal{P}, \mathcal{Q}, \rightarrow)$ are considered to be fixed in the following.

A *configuration* is a snapshot of the system, describing the topology of the network (i.e., the set of present components and interactions) together with the local state of each component, formally defined below (see also [4]):

▶ **Definition 1.** *A* configuration *is a tuple* $\gamma = (\mathcal{C}, \mathcal{I}, \varrho)$*, where:*

- $\mathcal{C} \subseteq_{fin} \mathbb{C}$ *is a finite set of* components*, that are present in the configuration,*
- $\mathcal{I} \subseteq_{fin} (\mathbb{C} \times \mathcal{P})^+$ *is a finite set of* interactions*, where each interaction is a sequence* $(c_i, p_i)_{i \in [1,n]} \in (\mathbb{C} \times \mathcal{P})^n$ *that binds together the ports* $p_1, \ldots, p_n$ *of the pairwise distinct components* $c_1, \ldots, c_n$*, respectively. The ordered sequence of ports* $(p_1, \ldots, p_n)$ *is called an* interaction type *and we denote by* $\mathcal{P}^+$ *the set of interaction types.*
- $\varrho : \mathbb{C} \rightarrow \mathcal{Q}$ *is a* state map *associating each (possibly absent) component, a state of the behavior* $\mathbb{B}$*, such that the set* $\{c \in \mathbb{C} \mid \varrho(c) = q\}$ *is infinite, for each* $q \in \mathcal{Q}$*.*

We denote by Γ *the set of configurations.*

The last condition requires that there is an infinite pool of components in each state $q \in \mathcal{Q}$; since $\mathbb{C}$ is infinite and $\mathcal{Q}$ is finite, this condition is feasible.

▶ **Example 2.** The configurations of the system from Fig. 1 (a) are $(\{c_1, \ldots, c_n\}, \{(c_i, \mathit{out}, c_{(i \bmod n)+1}, \mathit{in}) \mid i \in [1,n]\}, \varrho)$, where $\varrho : \mathbb{C} \rightarrow \{\mathsf{H}, \mathsf{T}\}$ is a state map. The ring topology is given by components $\{c_1, \ldots, c_n\}$ and interactions $\{(c_i, \mathit{out}, c_{(i \bmod n)+1}, \mathit{in}) \mid i \in [1,n]\}$. ⌟

Note that Def. 1 allows configurations with interactions that involve absent components i.e., not from the set $\mathcal{C}$ of present components in the given configuration. The following definition distinguishes such configurations:

▶ **Definition 3.** *A configuration* $\gamma = (\mathcal{C}, \mathcal{I}, \varrho)$ *is said to be* tight *if and only if for any interaction* $(c_i, p_i)_{i \in [1,n]} \in \mathcal{I}$ *we have* $\{c_i \mid i \in [1,n]\} \subseteq \mathcal{C}$ *and* loose *otherwise.*

For instance, every configuration of the system from Fig. 1 (a) is tight and becomes loose if a component is deleted.

2.2 Configuration Logic

Let $\mathbb{V}$ and $\mathbb{A}$ be countably infinite sets of *variables* and *predicates*, respectively. For each predicate $\mathsf{A} \in \mathbb{A}$, we denote its arity by $\#\mathsf{A}$. The formulæ of the *Configuration Logic* (CL) are described inductively by the following syntax:

$$\phi := \mathsf{emp} \mid [x] \mid \langle x_1.p_1, \ldots, x_n.p_n \rangle \mid x@q \mid x = y \mid x \neq y \mid \mathsf{A}(x_1, \ldots, x_{\#\mathsf{A}}) \mid \phi * \phi \mid \exists x \ . \ \phi$$

where $x, y, x_1, \ldots \in \mathbb{V}$, $q \in \mathcal{Q}$ and $\mathsf{A} \in \mathbb{A}$. A formula $[x]$, $\langle x_1.p_1, \ldots, x_n.p_n \rangle$, $x@q$ and $\mathsf{A}(x_1, \ldots, x_{\#\mathsf{A}})$ is called a *component*, *interaction*, *state* and *predicate* atom, respectively. We use the shorthand $[x]@q \stackrel{\text{def}}{=} [x] * x@q$. Intuitively, a formula $[x]@q * [y]@q' * \langle x.\mathit{out}, y.\mathit{in} \rangle * \langle x.\mathit{in}, y.\mathit{out} \rangle$ describes a configuration consisting of two distinct components, denoted by the values of x and y, in states q and q', respectively, and two interactions binding the *out* port of one to the *in* port of the other component.

A formula with no occurrences of predicate atoms (resp. existential quantifiers) is called *predicate-free* (resp. *quantifier-free*). A qpf formula is both predicate- and quantifier-free. A variable is *free* if it does not occur in the scope of a quantifier and $\mathrm{fv}(\phi)$ is the set of free variables of ϕ. A *substitution* $\phi[x_i/y_i]_{i \in [1,n]}$ replaces simultaneously every free occurrence of x_i by y_i in ϕ, for all $i \in [1,n]$. The size of a formula ϕ is the total number of occurrences of symbols needed to write it down, denoted by $\mathrm{size}(\phi)$.

The only connective of the logic is the *separating conjunction* $*$. Intuitively, $\phi_1 * \phi_2$ means that ϕ_1 and ϕ_2 hold separately, on disjoint parts of the same configuration. Its formal meaning is coined by the following definition of *composition* of configurations:

▶ **Definition 4.** *The composition of two configurations $\gamma_i = (\mathcal{C}_i, \mathcal{I}_i, \varrho)$, for $i = 1, 2$, such that $\mathcal{C}_1 \cap \mathcal{C}_2 = \emptyset$ and $\mathcal{I}_1 \cap \mathcal{I}_2 = \emptyset$, is defined as $\gamma_1 \bullet \gamma_2 \stackrel{\text{def}}{=} (\mathcal{C}_1 \cup \mathcal{C}_2, \mathcal{I}_1 \cup \mathcal{I}_2, \varrho)$. The composition $\gamma_1 \bullet \gamma_2$ is undefined if $\mathcal{C}_1 \cap \mathcal{C}_2 \neq \emptyset$ or $\mathcal{I}_1 \cap \mathcal{I}_2 \neq \emptyset$.*

▶ **Example 5.** Let $\gamma_i = (\{c_i\}, \{(c_i, out, c_{3-i}, in)\}, \varrho)$ be configurations, for $i = 1, 2$. Then $\gamma_1 \bullet \gamma_2 = (\{c_1, c_2\}, \{(c_1, out, c_2, in), (c_2, out, c_1, in)\}, \varrho)$. ⌟

The meaning of the predicates is given by a set of inductive definitions:

▶ **Definition 6.** *A* set of inductive definitions (SID) Δ *consists of* rules *of the form* $\mathsf{A}(x_1, \ldots, x_{\#\mathsf{A}}) \leftarrow \phi$*, where* $x_1, \ldots, x_{\#\mathsf{A}}$ *are pairwise distinct variables, called* parameters*, such that* $\mathrm{fv}(\phi) \subseteq \{x_1, \ldots, x_{\#\mathsf{A}}\}$*. We say that the rule* $\mathsf{A}(x_1, \ldots, x_{\#\mathsf{A}}) \leftarrow \phi$ defines A *and denote by* $\mathrm{def}_\Delta(\mathsf{A})$ *the set of rules from* Δ *that define* A *and by* $\mathrm{Def}(\Delta) \stackrel{\text{def}}{=} \{\mathsf{A} \mid \mathrm{def}_\Delta(\mathsf{A}) \neq \emptyset\}$ *the set of predicates defined by* Δ*.*

Note that having distinct parameters in a rule is without loss of generality, as e.g., a rule $\mathsf{A}(x_1, x_1) \leftarrow \phi$ can be equivalently written as $\mathsf{A}(x_1, x_2) \leftarrow x_1 = x_2 * \phi$. As a convention, we shall always use the names $x_1, \ldots, x_{\#\mathsf{A}}$ for the parameters of a rule that defines A. An example of a SID is given in §1.1.

The size of a SID is $\mathrm{size}(\Delta) \stackrel{\text{def}}{=} \sum_{\mathsf{A}(x_1,\ldots,x_{\#\mathsf{A}}) \leftarrow \phi \in \Delta} \mathrm{size}(\phi) + \#\mathsf{A} + 1$. Other parameters, relevant for complexity evaluation, are the maximal

(1) arity $\#(\Delta) \stackrel{\text{def}}{=} \max\{\#\mathsf{A} \mid \mathsf{A}(x_1, \ldots, x_{\#\mathsf{A}}) \leftarrow \phi \in \Delta\}$ of a defined predicate,

(2) size of an interaction type $\mathsf{N}(\Delta) \stackrel{\text{def}}{=} \max\{n \mid \langle y_1.p_1, \ldots, y_n.p_n \rangle \text{ occurs in } \Delta\}$, and

(3) number of predicate atoms $\mathsf{H}(\Delta) \stackrel{\text{def}}{=} \max\{h \mid \mathsf{A}(x_1, \ldots, x_{\#\mathsf{A}}) \leftarrow \exists y_1 \ldots \exists y_m \,.\, \phi * \mathbin{\ast}_{\ell=1}^{h} \mathsf{B}_\ell(\mathbf{z}_\ell), \phi \text{ is a qpf formula}\}$.

The semantics of CL formulæ is defined by a satisfaction relation $\gamma \models^\nu_\Delta \phi$ between configurations and formulæ. This relation is parameterized by a *store* $\nu : \mathbb{V} \to \mathbb{C}$ mapping the free variables of a formula into components from the universe (possibly absent from γ) and an SID Δ. The definition of the satisfaction relation is by induction on the structure of formulæ, where $\gamma = (\mathcal{C}, \mathcal{I}, \varrho)$ is a configuration (Def. 1):

$$
\begin{array}{lll}
\gamma \models^\nu_\Delta \mathsf{emp} & \iff & \mathcal{C} = \emptyset \text{ and } \mathcal{I} = \emptyset \\
\gamma \models^\nu_\Delta [x] & \iff & \mathcal{C} = \{\nu(x)\} \text{ and } \mathcal{I} = \emptyset \\
\gamma \models^\nu_\Delta \langle x_1.p_1, \ldots, x_n.p_n \rangle & \iff & \mathcal{C} = \emptyset \text{ and } \mathcal{I} = \{(\nu(x_1), p_1, \ldots, \nu(x_n), p_n)\} \\
\gamma \models^\nu_\Delta x@q & \iff & \gamma \models^\nu_\Delta \mathsf{emp} \text{ and } \varrho(\nu(x)) = q \\
\gamma \models^\nu_\Delta x \sim y & \iff & \gamma \models^\nu_\Delta \mathsf{emp} \text{ and } \nu(x) \sim \nu(y), \text{ for all } \sim \in \{=, \neq\} \\
\gamma \models^\nu_\Delta \mathsf{A}(y_1, \ldots, y_{\#\mathsf{A}}) & \iff & \gamma \models^\nu_\Delta \phi[x_1/y_1, \ldots, x_{\#\mathsf{A}}/y_{\#\mathsf{A}}], \text{ for some rule} \\
 & & \mathsf{A}(x_1, \ldots, x_{\#\mathsf{A}}) \leftarrow \phi \text{ from } \Delta \\
\gamma \models^\nu_\Delta \phi_1 * \phi_2 & \iff & \text{there exist } \gamma_1 \text{ and } \gamma_2, \text{ such that } \gamma = \gamma_1 \bullet \gamma_2 \text{ and} \\
 & & \gamma_i \models^\nu_\Delta \phi_i, \text{ for all } i = 1, 2 \\
\gamma \models^\nu_\Delta \exists x \,.\, \phi & \iff & \gamma \models^{\nu[x \leftarrow c]}_\Delta \phi, \text{ for some } c \in \mathbb{C}
\end{array}
$$

If $\gamma \models^\nu_\Delta \phi$, we say that the pair (γ, ν) is a Δ-model of ϕ. If ϕ is a predicate-free formula, the satisfaction relation does not depend on the SID, written $\gamma \models^\nu \phi$. A formula ϕ is *satisfiable* if and only if it has a model. A formula ϕ Δ-*entails* a formula ψ, written $\phi \models_\Delta \psi$, if and only if any Δ-model of ϕ is a Δ-model of ψ. Two formulæ are Δ-*equivalent*, written $\phi \equiv_\Delta \psi$ if and only if $\phi \models_\Delta \psi$ and $\psi \models_\Delta \phi$. A formula ϕ is Δ-*tight* if γ is tight (Def. 3), for any Δ-model (γ, ν) of ϕ. We omit mentioning Δ whenever it is clear from the context or not needed.

2.3 The Havoc Invariance Problem

This paper is concerned with the *havoc invariance* problem i.e., the problem of deciding whether the set of models of a given CL formula is closed under the execution of a sequence of interactions. The execution of an interaction $(c_i, p_i)_{i\in[1,n]}$ synchronizes transitions labeled by the ports $p_1, \ldots, p_n$ from the behaviors (i.e., replicas of the state machine $\mathbb{B}$) of $c_1, \ldots, c_n$, respectively. This joint execution of several transitions in different components of the system is formally described by the *step* relation below:

▶ **Definition 7.** *The* step *relation* $\Rightarrow \subseteq \Gamma \times (\mathbb{C} \times \mathcal{P})^+ \times \Gamma$ *is defined as:*

$$(\mathcal{C}, \mathcal{I}, \varrho) \xrightarrow{(c_i,p_i)_{i\in[1,n]}} (\mathcal{C}, \mathcal{I}, \varrho') \text{ if and only if } (c_i, p_i)_{i\in[1,n]} \in \mathcal{I} \text{ and } \varrho' = \varrho[c_i \leftarrow q_i']_{i\in[1,n]}$$
$$\text{where } \varrho(c_i) = q_i \text{ and } q_i \xrightarrow{p_i} q_i' \text{ is a transition of } \mathbb{B}, \text{ for all } i \in [1, n]$$

The havoc *relation* $\rightsquigarrow^*$ *is the reflexive and transitive closure of the relation* $\rightsquigarrow \subseteq \Gamma^2$*:* $(\mathcal{C}, \mathcal{I}, \varrho) \rightsquigarrow (\mathcal{C}, \mathcal{I}, \varrho')$ *if and only if* $(\mathcal{C}, \mathcal{I}, \varrho) \xrightarrow{(c_i,p_i)_{i\in[1,n]}} (\mathcal{C}, \mathcal{I}, \varrho')$*, for some interaction* $(c_i, p_i)_{i\in[1,n]} \in \mathcal{I}$*.*

▶ **Example 8.** Let $\gamma_i = (\{c_1, c_2, c_3\}, \{(c_i, out, c_{i \bmod 3+1}, in) \mid i \in [1, 3]\}, \varrho_i)$, for $i \in [1, 3]$ be configurations, where $\varrho_1(c_1) = \varrho_1(c_2) = \mathsf{H}$, $\varrho_1(c_3) = \mathsf{T}$, $\varrho_2(c_1) = \mathsf{T}$, $\varrho_2(c_2) = \varrho_2(c_3) = \mathsf{H}$, $\varrho_3(c_1) = \varrho_3(c_3) = \mathsf{H}$, $\varrho_3(c_2) = \mathsf{T}$. Then we have $\gamma_i \rightsquigarrow^* \gamma_j$, for all $i, j \in [1, 3]$. ⌟

Two interactions $(c_1, p_1, \ldots, c_n, p_n)$ and $(c_{i_1}, p_{i_1}, \ldots, c_{i_n}, p_{i_n})$ such that $\{i_1, \ldots, i_n\} = [1, n]$, are equivalent from the point of view of the step relation, since the set of executed transitions is the same; nevertheless, we chose to distinguish them in the following, for reasons of simplicity. Note, moreover, that the havoc relation does not change the component or the interaction set of a configuration, only its state map.

▶ **Definition 9.** *Given an SID* Δ *and a predicate* A*, the problem* $\mathsf{HavocInv}[\Delta, \mathsf{A}]$ *asks whether for all* $\gamma, \gamma' \in \Gamma$ *and each store* ν*, such that* $\gamma \models^{\nu}_{\Delta} \mathsf{A}(x_1, \ldots, x_{\#\mathsf{A}})$ *and* $\gamma \rightsquigarrow^* \gamma'$*, it is the case that* $\gamma' \models^{\nu}_{\Delta} \mathsf{A}(x_1, \ldots, x_{\#\mathsf{A}})$*?*

▶ **Example 10.** Consider a model $\gamma = (\{c_1, \ldots, c_n\}, \{(c_i, out, c_{(i \bmod n)+1}, in) \mid i \in [1, n]\}, \varrho)$ of the formula $\mathsf{ring}_{1,1}()$ i.e., having the property that $\varrho(c_i) = \mathsf{H}$ and $\varrho(c_j) = \mathsf{T}$ for at least two indices $i \neq j \in [1, n]$, where the SID that defines $\mathsf{ring}_{1,1}()$ is given in §1.1. Similar to Example 8, in any configuration $\gamma' = (\{c_1, \ldots, c_n\}, \{(c_i, out, c_{(i \bmod n)+1}, in) \mid i \in [1, n]\}, \varrho')$ such that $\gamma \rightsquigarrow^* \gamma'$, we have $\varrho'(c_k) = \mathsf{H}$ and $\varrho'(c_\ell) = \mathsf{T}$, for some $k \neq \ell \in [1, n]$, hence γ' is a model of $\mathsf{ring}_{1,1}()$, meaning that $\mathsf{ring}_{1,1}()$ is havoc invariant. Examples of formulæ that are not havoc invariant include e.g., $[x]@\mathsf{T} * \langle x.out, y.in\rangle * [y]@\mathsf{H}$. ⌟

Without loss of generality, we consider the havoc invariance problem only for single predicate atoms. This is because, for any formula ϕ, such that $\mathrm{fv}(\phi) = \{x_1, \ldots, x_n\}$, one may consider a fresh predicate symbol (i.e., not in the SID) A_ϕ and add the rule $\mathsf{A}_\phi(x_1, \ldots, x_n) \leftarrow \phi$ to the SID. Then ϕ is havoc invariant if and only if $\mathsf{A}_\phi(x_1, \ldots, x_n)$ is havoc invariant.

3 From Havoc Invariance to Entailment

We describe a many-one reduction of the havoc invariance (Def. 9) to the entailment problem, following three steps. Given an instace $\mathsf{HavocInv}[\Delta, \mathsf{A}]$ of the havoc invariance problem, the SID Δ is first translated into a tree automaton recognizing trees labeled with predicate-free formulæ, that symbolically encode the set of Δ-models of the predicate atom $\mathsf{A}(x_1, \ldots, x_{\#\mathsf{A}})$.

Second, we define a structure-preserving tree transducer that simulates the effect of executing exactly one interaction from such a model. Third, we compute the image of the language recognized by the first tree automaton via the transducer, as a second tree automaton, which is translated back into another SID $\overline{\Delta}$ defining one or more predicates $\overline{\mathsf{A}}_1, \ldots, \overline{\mathsf{A}}_p$, among other. Finally, we prove that $\mathsf{HavocInv}[\Delta, \mathsf{A}]$ has a positive answer if and only if each of the entailments $\{\overline{\mathsf{A}}_i(x_1, \ldots, x_{\#\mathsf{A}}) \models_{\Delta \cup \overline{\Delta}} \mathsf{A}(x_1, \ldots, x_{\#\mathsf{A}})\}_{i=1}^{p}$ produced by the reduction, holds.

For the sake of self-containment, we recall below the definitions of trees, tree automata and (structure-preserving) tree transducers. Let $(\Sigma, \#)$ be a ranked alphabet, where each symbol $\alpha \in \Sigma$ has an associated arity $\#\alpha \geq 0$. A *tree* over Σ is a finite partial function $\mathfrak{t} : \mathbb{N}^* \rightharpoonup_{fin} \Sigma$, whose domain $\mathrm{dom}(\mathfrak{t}) \subseteq_{fin} \mathbb{N}^*$ is both *prefix-closed* i.e., $u \in \mathrm{dom}(t)$, for all $u, v \in \mathbb{N}^*$, such that $u \cdot v \in \mathrm{dom}(\mathfrak{t})$, and *complete* i.e., $\{n \in \mathbb{N} \mid u \cdot n \in \mathrm{dom}(t)\} = [1, \#\mathfrak{t}(u)]$, for all $u \in \mathrm{dom}(\mathfrak{t})$. Given $u \in \mathrm{dom}(\mathfrak{t})$, the *subtree* of $\mathfrak{t}$ rooted at u is the tree $\mathfrak{t}_{|u}$, such that $\mathrm{dom}(t_{|u}) \stackrel{\text{def}}{=} \{w \mid u \cdot w \in \mathrm{dom}(\mathfrak{t})\}$ and $\mathfrak{t}_{|u}(w) \stackrel{\text{def}}{=} \mathfrak{t}(u \cdot w)$. We denote by $\mathbb{T}(\Sigma)$ the set of trees over a ranked alphabet Σ.

A *tree automaton* (TA) is a tuple $\mathcal{A} = (\Sigma, \mathcal{S}, \mathcal{F}, \delta)$, where Σ is a ranked alphabet, $\mathcal{S}$ is a finite set of states, $\mathcal{F} \subseteq \mathcal{S}$ is a set of final states and δ is a set of transitions $\alpha(s_1, \ldots, s_{\#a}) \to s$; when $\#\alpha = 0$, we write $\alpha \to s$ instead of $\alpha() \to s$. A *run* of $\mathcal{A}$ over a tree $\mathfrak{t}$ is a function $\pi : \mathrm{dom}(\mathfrak{t}) \to \mathcal{S}$, such that, for all $u \in \mathrm{dom}(\mathfrak{t})$, we have $\pi(u) = s$ if $(\mathfrak{t}(u))(\pi(u \cdot 1), \ldots, \pi(u \cdot \#\mathfrak{t}(u))) \to s \in \delta$. Given a state $q \in \mathcal{S}$, a run π of $\mathcal{A}$ is *q-accepting* if and only if $\pi(\epsilon) = q$, in which case $\mathcal{A}$ is said to q-accept $\mathfrak{t}$. We denote by $\mathcal{L}_q(\mathcal{A})$ the set of trees q-accepted by $\mathcal{A}$ and let $\mathcal{L}(\mathcal{A}) \stackrel{\text{def}}{=} \bigcup_{q \in \mathcal{F}} \mathcal{L}_q(\mathcal{A})$. A language L is *recognizable* if and only if there exists a TA $\mathcal{A}$, such that $L = \mathcal{L}(\mathcal{A})$.

A *tree transducer* (TT) is a tree automaton over an alphabet of pairs $\mathcal{T} = (\Sigma^2, \mathcal{S}, \mathcal{F}, \delta)$, such that $\#\alpha = \#\beta = n$, for each transition $(\alpha, \beta)(s_1, \ldots, s_n) \to s \in \delta$. Intuitively, a transition of the transducer reads a symbol α from the input tree and writes another symbol β to the output tree, at the same position. Cleary, any tree $\mathfrak{t} : \mathbb{N}^* \rightharpoonup_{fin} \Sigma^2$ with labels from the set of pairs $\{(\alpha, \beta) \in \Sigma^2 \mid \#\alpha = \#\beta\}$ can be viewed as a pair of trees $(\mathfrak{t}_1, \mathfrak{t}_2)$ over Σ, such that $\mathrm{dom}(\mathfrak{t}_1) = \mathrm{dom}(\mathfrak{t}_2) = \mathrm{dom}(\mathfrak{t})$. In order to define the image of a tree language via a transducer, we define

(i) *projection* $L{\downarrow_i} \stackrel{\text{def}}{=} \{\mathfrak{t}_i \mid (\mathfrak{t}_1, \mathfrak{t}_2) \in L\}$, for all $i = 1, 2$, where $L \subseteq \mathbb{T}(\Sigma^2)$, and

(ii) *cylindrification* $L{\uparrow^i} \stackrel{\text{def}}{=} \{(\mathfrak{t}_1, \mathfrak{t}_2) \mid \mathfrak{t}_i \in L\}$, for all $i = 1, 2$, where $L \subseteq \mathbb{T}(\Sigma)$.

The *image* of a language $L \subseteq \mathbb{T}(\Sigma)$ via a transducer $\mathcal{T}$ is the language $\mathcal{T}(L) \stackrel{\text{def}}{=} \big(L{\uparrow^1} \cap \mathcal{L}(\mathcal{T})\big){\downarrow_2}$. It is manifest that $\mathcal{T}(L)$ is recognizable whenever L is recognizable.

3.1 From SID to Tree Automata and Back

We define a two-way connection between SIDs and TAs, as follows:

1. Given a finite SID Δ we define a TA $\mathcal{A}_\Delta$, whose states q_{A} are named after the predicates A that occur in Δ and whose alphabet consists of the predicate-free formulæ from the rules of Δ, with variables mapped to canonical names, together with a tuple of arities, needed for later bookkeeping. Each tree $\mathfrak{t} \in \mathcal{L}_{q_{\mathsf{A}}}(\mathcal{A}_\Delta)$ defines a unique predicate-free formula $\Phi(\mathfrak{t})$, such that the Δ-models of a predicate atom $\mathsf{A}(x_1, \ldots, x_{\#\mathsf{A}})$ are exactly the models of some $\Phi(\mathfrak{t})$, for $\mathfrak{t} \in \mathcal{L}_{q_{\mathsf{A}}}(\mathcal{A}_\Delta)$.
2. Conversely, given a TA $\mathcal{A}$ over an alphabet of formulæ annotated with arities, the tuple of arities associated with each alphabet symbol allows to define a SID $\Delta_{\mathcal{A}}$, whose predicates A_q are named after the states q of the TA, such that the models of the formulæ $\Phi(\mathfrak{t})$, such that $\mathfrak{t} \in \mathcal{L}_q(\mathcal{A})$ are exactly the $\Delta_{\mathcal{A}}$-models of the predicate atom $\mathsf{A}_q(x_1, \ldots, x_{\#\mathsf{A}_q})$.

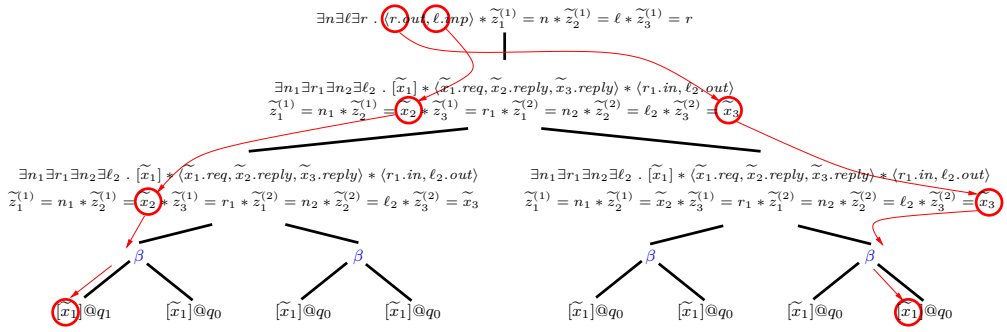


Figure 2 Tree Labeled with Formulæ Encoding a System from Example 12.

Let us fix a countably infinite set of variables $\widetilde{\mathsf{Var}} \stackrel{\text{def}}{=} \{\widetilde{x}_i \mid i \geq 1\} \cup \{\widetilde{z}_i^{(\ell)} \mid i, \ell \geq 1\}$, with the understanding that $\widetilde{x}_i$ are canonical names for the variables from the left-hand side and $\widetilde{z}_i^{(\ell)}$ are canonical names for the variables occurring in the ℓ-th predicate atom from the right-hand side of a rule. An *alphabet symbol* $\alpha = \langle \psi, a_0, \ldots, a_h \rangle$ consists of a predicate-free formula ψ and a tuple of positive integers $a_0, \ldots, a_h \in \mathbb{N}$, such that $\mathrm{fv}(\psi) = \{\widetilde{x}_i \mid i \in [1, a_0]\} \cup \{\widetilde{z}_i^{(\ell)} \mid \ell \in [1, h],\ i \in [1, a_\ell]\}$. We take the arity of such a symbol to be $\#\alpha \stackrel{\text{def}}{=} h$ and denote by $\widetilde{\Sigma}$ the (infinite) set of alphabet symbols. Trees labeled with symbols from $\widetilde{\Sigma}$ define predicate-free *characteristic formulæ*, as follows:

▶ **Definition 11.** *Given a tree* $\mathfrak{t} \in \mathbb{T}(\widetilde{\Sigma})$*, where* $\mathfrak{t}(\epsilon) = \langle \exists y_1 \ldots \exists y_m \;.\; \phi, a_0, \ldots, a_h \rangle$ *with* ϕ *a* qpf *formula, and a node* $u \in \mathbb{N}^*$*, we define the* qpf characteristic formula*:*

$$\Psi^u(\mathfrak{t}) \stackrel{\text{def}}{=} \phi[\widetilde{x}_j / x_j^u]_{j \in [1, a_0]} [\widetilde{z}_j^{(\ell)} / x_j^{u \cdot \ell}]_{\ell \in [1,h], j \in [1, a_\ell]} [y_j / y_j^u]_{j \in [1,m]} * \mathbin{\ast}_{\ell \in [1,h]} \Psi^{u \cdot \ell}(\mathfrak{t}_{|\ell})$$

Assuming that $\mathfrak{t}(v) = \langle \exists y_1 \ldots \exists y_{m^v} \;.\; \phi^v, a_0^v, \ldots, a_h^v \rangle$*, for all* $v \in \mathrm{dom}(t)$*, we consider also the predicate-free formula* $\Phi^u(\mathfrak{t}) = (\exists x_j^{u \cdot v})_{v \in \mathrm{dom}(t) \setminus \{\epsilon\},\ j \in [1, a_0^v]} (\exists y_j^{u \cdot v})_{v \in \mathrm{dom}(t),\ j \in [1, m^v]} \;.\; \Psi^u(\mathfrak{t})$.

▶ **Example 12.** We consider a system whose components form a tree, in which each parent sends a request (*req*) to and receives replies (*reply*) from both its children. In addition, the leaves of the tree form a ring, with the *out* port of each leaf connected to the *in* port of its right neighbour. The system is described by the following inductive definitions:

$$Root() \leftarrow \exists n \exists \ell \exists r \;.\; \langle r.out, \ell.in \rangle * Node(n, \ell, r) \tag{1}$$

$$Node(n, \ell, r) \leftarrow \exists n_1 \exists r_1 \exists n_2 \exists \ell_2 \;.\; [n] * \langle n.req, n_1.reply, n_2.reply \rangle * \langle r_1.in, \ell_2.out \rangle * Node(n_1, \ell, r_1) * Node(n_2, \ell_2, r) \tag{2}$$

$$Node(n, \ell, r) \leftarrow [n]@q_0 \qquad Node(n, \ell, r) \leftarrow [n]@q_1 \tag{3}$$

Fig. 2 shows a tree $\mathfrak{t} \in \mathbb{T}(\widetilde{\Sigma})$ describing an instance of the system, where $\widetilde{\Sigma} = \{\alpha, \beta, \gamma_0, \gamma_1\}$:

$$\alpha \stackrel{\text{def}}{=} \langle \exists n \exists \ell \exists r \;.\; \langle r.out, \ell.in \rangle * \widetilde{z}_1^{(1)} = n * \widetilde{z}_2^{(1)} = \ell * \widetilde{z}_3^{(1)} = r, 0, 3 \rangle$$

$$\beta \stackrel{\text{def}}{=} \langle \exists n_1 \exists r_1 \exists n_2 \exists \ell_2 \;.\; [n] * \langle n.req, \ell.reply, r.reply \rangle * \langle r_1.in, \ell_2.out \rangle * \widetilde{z}_1^{(1)} = n_1 * \widetilde{z}_2^{(1)} = \ell * \widetilde{z}_3^{(1)} = r_1 * \widetilde{z}_1^{(2)} = n_2 * \widetilde{z}_2^{(2)} = \ell * \widetilde{z}_3^{(2)} = r, 3, 3, 3 \rangle$$

$$\gamma_0 \stackrel{\text{def}}{=} \langle [\widetilde{x}_1]@q_0, 3 \rangle \qquad \gamma_1 \stackrel{\text{def}}{=} \langle [\widetilde{x}_1]@q_1, 3 \rangle$$

For simplicity, Fig. 2 shows only the formulæ, not the arity lists of the alphabet symbols. ⌟

The models of the characteristic formula $\Psi^\epsilon(\mathfrak{t})$ of a tree $\mathfrak{t} \in \mathbb{T}(\widetilde{\Sigma})$ define walks in the tree that correspond to chains of equalities between variables. Formally, a *walk* in $\mathfrak{t}$ is a sequence of nodes $u_1, \ldots, u_n \in \mathrm{dom}(\mathfrak{t})$, such that u_i is either the parent or a child of u_{i+1}, for all $i \in [1, n-1]$. Note that a walk can visit the same node of the tree several times. In particular, if the characteristic formula $\Psi^\epsilon(\mathfrak{t})$ is tight (i.e., has only tight models in the sense of Def. 3) there exist equality walks between the node containing an interaction atom and the nodes where these variables are instantiated by component atoms. For instance, walks between the root containing $\langle r.out, \ell.in \rangle$ and the left- and right-most leafs, labeled with component atoms that associate elements of $\mathbb{C}$ to the variables ℓ and r are shown in Fig. 2.

▶ **Lemma 13.** *Let $\mathfrak{t} \in \mathbb{T}(\widetilde{\Sigma})$ be a tree, such that $\Psi^\epsilon(\mathfrak{t})$ is tight, (γ, ν) be a model of $\Psi^\epsilon(\mathfrak{t})$ and y^v, z^w be two variables that occur in a component and interaction atom of $\Psi^\epsilon(\mathfrak{t})$, respectively. Then $\nu(y^v) = \nu(z^w)$ if and only if there exists a walk $u_1, \ldots, u_n$ in $\mathfrak{t}$ and variables $y^v = x_{i_1}^{u_1}, \ldots, x_{i_n}^{u_n} = z^w$, such that either $x_{i_j}^{u_j}$ and $x_{i_{j+1}}^{u_{j+1}}$ are the same variable, or the equality atom $x_{i_j}^{u_j} = x_{i_{j+1}}^{u_{j+1}}$ occurs in $\Psi^\epsilon(\mathfrak{t})$, for all $j \in [1, n-1]$.*

Let Δ be a fixed and finite SID in the following. We build a TA $\mathcal{A}_\Delta$ that recognizes the Δ-models of each predicate atom defined by Δ, in the sense of Lemma 16 below.

▶ **Definition 14.** *We associate each rule $r : \mathsf{A}(x_1, \ldots, x_{\#\mathsf{A}}) \leftarrow \exists y_1 \ldots \exists y_m \ . \ \phi * \ast_{\ell \in [1,h]} \mathsf{B}_\ell(z_1^\ell, \ldots, z_{\#\mathsf{B}_\ell}^\ell) \in \Delta$, where ϕ is a* qpf *formula, with the alphabet symbol:*

$$\alpha_r \stackrel{\text{def}}{=} \left\langle \exists y_1 \ldots \exists y_m \ . \ \left(\phi * \ast_{\ell \in [1,h],\ i \in [1,\#\mathsf{B}_\ell]} \widetilde{z}_i^{(\ell)} = z_i^\ell \right) [x_j / \widetilde{x}_j]_{j \in [1,\#\mathsf{A}]}, \#\mathsf{A}, \#\mathsf{B}_1, \ldots, \#\mathsf{B}_h \right\rangle$$

Let $\mathcal{A}_\Delta \stackrel{\text{def}}{=} (\Sigma_\Delta, \mathcal{S}_\Delta, \delta_\Delta)$ be a TA, where $\Sigma_\Delta \stackrel{\text{def}}{=} \{\alpha_r \mid r \in \Delta\}$, $\mathcal{S}_\Delta \stackrel{\text{def}}{=} \{q_\mathsf{A} \mid \mathsf{A} \in \mathrm{Def}(\Delta)\}$ and $\delta_\Delta \stackrel{\text{def}}{=} \{\alpha_r(q_{\mathsf{B}_1}, \ldots, q_{\mathsf{B}_h}) \to q_\mathsf{A} \mid r \in \Delta\}$.

▶ **Example 15** (contd. from Example 12)**.** The TA corresponding to the SID in Example 12 is $\mathcal{A}_\Delta = (\widetilde{\Sigma}, \mathcal{S}_\Delta, \delta_\Delta)$, where $\widetilde{\Sigma} = \{\alpha, \beta, \gamma_0, \gamma_1\}$, $\mathcal{S}_\Delta = \{q_{Root}, q_{Node}\}$ and $\delta_\Delta = \{\alpha(q_{Node}) \to q_{Root}, \beta(q_{Node}, q_{Node}) \to q_{Node}, \gamma_0 \to q_{Node}, \gamma_1 \to q_{Node}\}$. ⌟

The following lemma proves that the predicate-free formulæ corresponding (in the sense of Def. 11) to the trees recognized by $\mathcal{A}_\Delta$ in a state q_A define the Δ-models of the predicate atom $\mathsf{A}(x_1, \ldots, x_{\#\mathsf{A}})$:

▶ **Lemma 16.** *For any predicate $\mathsf{A} \in \mathrm{Def}(\Delta)$, configuration γ, store ν and node $u \in \mathbb{N}^*$, we have $\gamma \models_\Delta^\nu \mathsf{A}(x_1^u, \ldots, x_{\#\mathsf{A}}^u)$ if and only if $\gamma \models^\nu \Phi^u(\mathfrak{t})$, for some tree $\mathfrak{t} \in \mathcal{L}_{q_\mathsf{A}}(\mathcal{A}_\Delta)$.*

Conversely, given a tree automaton $\mathcal{A} = (\Sigma, \mathcal{S}, \delta)$, we construct a SID $\Delta_\mathcal{A}$ that defines the models of the predicate-free formulæ corresponding (Def. 11) to the trees recognized by $\mathcal{A}$ (Lemma 19). We assume that the alphabet Σ consists of symbols $\langle \psi, a_0, \ldots, a_h \rangle$ of arity h, where ψ is a predicate-free formula with free variables $\mathrm{fv}(\psi) = \{\widetilde{x}_i \mid i \in [1, a_0]\} \cup \{\widetilde{z}_i^{(\ell)} \mid \ell \in [1, h],\ i \in [1, a_\ell]\}$ and that the transitions of the TA meet the requirement:

▶ **Definition 17.** *A TA $\mathcal{A}$ is* SID-compatible *iff for any transitions $\langle \psi, a_0, \ldots, a_h \rangle(q_1, \ldots, q_h) \to q_0$ and $\langle \psi', a_0', \ldots, a_h' \rangle(q_1', \ldots, q_h') \to q_0'$ of $\mathcal{A}$, we have $q_i = q_i'$ only if $a_i = a_i'$, for all $i \in [0, h]$.*

Let us fix a SID-compatible TA $\mathcal{A} = (\Sigma, \mathcal{S}, \delta)$ for the rest of this section.

▶ **Definition 18.** *The SID $\Delta_\mathcal{A}$ has a rule:*

$$\mathsf{A}_{q_0}(x_1, \ldots, x_{a_0}) \leftarrow \exists y_1^1 \ldots \exists y_{a_h}^h \ . \ \phi[\widetilde{x}_i / x_i]_{i \in [1,a_0]} [\widetilde{z}_i^{(\ell)} / y_i^\ell]_{\ell \in [1,h],\ i \in [1,a_\ell]} \ * \ast_{\ell \in [1,h]} \mathsf{A}_{q_\ell}(y_1^\ell, \ldots, y_{a_\ell}^\ell)$$

for each transition $\langle \phi, a_0, \ldots, a_h \rangle(q_1, \ldots, q_h) \to q_0$ of $\mathcal{A}$ and those rules only.

The following lemma states that $\Delta_{\mathcal{A}}$ defines the set of models of the characteristic formulæ (Def. 11) of the trees recognized by $\mathcal{A}$.

▶ **Lemma 19.** *For any state $q \in \mathcal{S}$, configuration γ, store ν and node $u \in \mathbb{N}^*$, we have $\gamma \models^{\nu}_{\Delta_{\mathcal{A}}} \mathsf{A}_q(x^u_1, \ldots, x^u_{\#\mathsf{A}_q})$ if and only if $\gamma \models^{\nu} \Phi^u(\mathfrak{t})$, for some tree $\mathfrak{t} \in \mathcal{L}_q(\mathcal{A})$.*

3.2 Encoding Havoc Steps by Tree Transducers

The purpose of this section is the definition of a transducer that simulates one havoc step. Before giving its definition, we note that the havoc invariance problem can be equivalently defined by considering the transformation induced by a single havoc step, instead of an arbitrary sequence of steps. The following lemma can be taken as an equivalent definition:

▶ **Lemma 20.** $\mathsf{HavocInv}[\Delta, \mathsf{A}]$ *has a positive answer if and only if, for all $\gamma, \gamma' \in \Gamma$ and each store ν, such that $\gamma \models^{\nu}_{\Delta} \mathsf{A}(x_1, \ldots, x_{\#\mathsf{A}})$ and $\gamma \rightsquigarrow \gamma'$, it is the case that $\gamma' \models^{\nu}_{\Delta} \mathsf{A}(x_1, \ldots, x_{\#\mathsf{A}})$.*

We fix a SID Δ for the rest of this section and recall the existence of a fixed finite-state behavior $\mathbb{B} = (\mathcal{P}, \mathcal{Q}, \rightarrow)$ with ports $\mathcal{P}$, states $\mathcal{Q}$ and transitions $q \xrightarrow{p} q' \in \mathcal{Q} \times \mathcal{P} \times \mathcal{Q}$. We define a transducer $\mathcal{T}_\tau$ parameterized by a given interaction type $\tau = (p_1, \ldots, p_n) \in \mathcal{P}^+$. The havoc step transducer is the automata-theoretic union of the typed transducers over the set of interaction types that occurs in Δ.

Given a tree $\mathfrak{t} \in \mathbb{T}(\widetilde{\Sigma})$, an interaction-typed transducer $\mathcal{T}_\tau$

(1) guesses an interaction atom $\langle z_1.p_1, \ldots, z_n.p_n \rangle$ that occurs in some label of $\mathfrak{t}$,

(2) tracks the equality walks (Lemma 13) between each variable z_i and the component atom $[x_i]@q_i$ that defines the store value of z_i and its current state, and

(3) replaces each state component atom $[x_i]@q_i$ by $[x_i]@q'_i$, where $q_i \xrightarrow{p_i} q'_i$ is a transition from $\mathbb{B}$, for each $i \in [1, n]$.

The output of the transducer is a tree $\mathfrak{t}' \in \mathbb{T}(\widetilde{\Sigma})$, that symbolically encodes the effect of executing some interaction of type τ over $\mathfrak{t}$. The main challenge in defining $\mathcal{T}_\tau$ is that the equality walks between an interaction atom $\langle z_1.p_1, \ldots, z_n.p_n \rangle$ and the component atoms instantiating the variables $z_1, \ldots, z_n$ may visit a tree node more than once. To capture this, the transducer will guess at once the equalities summarizing the different fragments of the walk that lie in the currently processed subtree of $\mathfrak{t}$. Accordingly, the states of $\mathcal{T}_\tau$ are conjunctions of equalities, with special variables $\widetilde{b_i}$ (resp. $\widetilde{e_i}$) indicating whether a component (resp. interaction) atom has already been encountered in the current subtree, intuitively marking the *beginning* (resp. *end*) of the walk.

For an interaction type $\tau = (p_1, \ldots, p_n)$, let $\widetilde{\mathsf{TVar}_\tau} \stackrel{\text{def}}{=} \{\widetilde{x}_i \mid i \in [1, \#(\Delta)]\} \cup \{\widetilde{b_i}, \widetilde{e_i} \mid i \in [1, n]\}$ and let $Eq(\widetilde{\mathsf{TVar}_\tau})$ be the set of separating conjunctions of equality atoms i.e., $\varphi \stackrel{\text{def}}{=} \ast_{i \in I}\ x_i = y_i$, such that $\mathrm{fv}(\varphi) \subseteq \widetilde{\mathsf{TVar}_\tau}$. Note that $\exists x\ .\ \varphi$, for $\varphi \in Eq(\widetilde{\mathsf{TVar}_\tau})$, is equivalent to a formula from $Eq(\widetilde{\mathsf{TVar}_\tau})$ obtained by eliminating the quantifier: either x occurs in an atom $x = y$ for a variable y distinct from x then $(\exists x\ .\ \varphi) \equiv \varphi[x/y]$, or $x \notin \mathrm{fv}(\varphi)$, in which case $(\exists x\ .\ \varphi) \equiv \varphi$.

▶ **Definition 21.** *The transducer $\mathcal{T}_\tau \stackrel{\text{def}}{=} (\Sigma^2_\Delta, \mathcal{S}_\tau, \mathcal{F}_\tau, \delta_\tau)$, where $\tau = (p_1, \ldots, p_n)$, is as follows:*

- $\mathcal{S}_\tau = \{\varphi \in Eq(\widetilde{\mathsf{TVar}_\tau}) \mid \varphi \not\models (\widetilde{b_i} = \widetilde{b_j}),\ \varphi \not\models (\widetilde{e_i} = \widetilde{e_j}), \varphi \not\models (\widetilde{b_i} = \widetilde{e_j})$, *for any $i \neq j$*$\}$,
- $\mathcal{F}_\tau = \{\varphi \in \mathcal{S}_\tau \mid \varphi \models \ast_{i \in [1,n]} (\widetilde{b_i} = \widetilde{e_i})\}$, *and*
- δ_τ *contains transitions of the form $(\alpha, \alpha')(\varphi_1, \ldots, \varphi_h) \rightarrow_{\mathcal{T}} \varphi$ where:*
  - $\alpha = (\exists y_1 \ldots \exists y_m\ .\ \psi, a_0, \ldots, a_h)$ *and* $\alpha' = (\exists y_1 \ldots \exists y_m\ .\ \psi', a_0, \ldots, a_h)$, *where ψ and ψ' are* qpf *formulæ such that* $\mathrm{fv}(\psi) = \mathrm{fv}(\psi') \subseteq \widetilde{\mathsf{TVar}_\tau} \cup \{y_1, \ldots, y_m\}$,

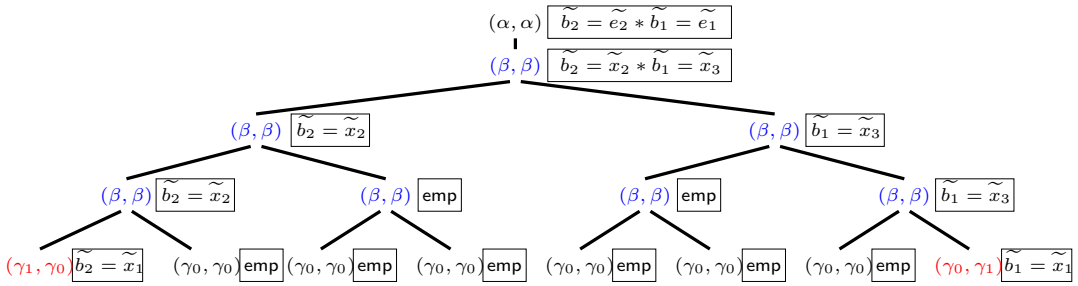


Figure 3 Tree Transducer for the Interactions of Type (out, in) in the System from Fig. 2.

- *there exists a set* $I = \{i_1, \ldots, i_r\} \subseteq [1, n]$, *variables* $\xi_1, \ldots, \xi_r \in \mathrm{fv}(\psi)$ *and transitions* $q_1 \xrightarrow{p_{i_1}} q'_1$, $\ldots$, $q_r \xrightarrow{p_{i_r}} q'_r$ *in* $\mathbb{B}$, *such that* $\psi = (\circledast_{k\in[1,r]}[\xi_k]@q_k) * \eta$ *and* $\psi' = (\circledast_{k\in[1,r]}[\xi_k]@q'_k) * \eta$, *for some* qpf *formula* η,
- *there exists a set* $J \in \{\emptyset, [1, n]\}$, *such that* ψ *contains an interaction atom* $\langle \zeta_1.p_1, \ldots, \zeta_n.p_n\rangle$ *if* $J = [1, n]$,
- *the sets* I *and* $\{i \in [1, n] \mid \widetilde{b_i} \in \mathrm{fv}(\varphi_\ell)\}_{\ell\in[1,h]}$ *are pairwise disjoint,*
- *at most one of the sets* J, $\{i \in [1, n] \mid \widetilde{e_i} \in \mathrm{fv}(\varphi_\ell)\}_{\ell\in[1,h]}$ *is not empty,*
- φ *is the result of eliminating the quantifiers from the separating conjunction of equalities:*

$$\exists \widetilde{z}_1^{(1)} \ldots \exists \widetilde{z}_{a_h}^{(h)} \exists y_1 \ldots \exists y_m \ . \ \circledast_{\ell\in[1,h]} \varphi_\ell[\widetilde{x}_j/\widetilde{z}_j^{(\ell)}]_{j\in[1,a_\ell]} * \circledast_{k\in[1,r]} \widetilde{b_{i_k}} = \xi_k * \circledast_{\ell\in J} \widetilde{e_\ell} = \zeta_\ell * \psi_{eq}$$

where ψ_{eq} *is the separating conjunction of the equality atoms from* ψ.

▶ **Example 22.** (contd. from Examples 12 and 15) Fig. 3 shows a run of the transducer $\mathcal{T}_{(out,in)}$, that describes the symbolic execution of the interaction corresponding to the $\langle r.out, \ell.in\rangle$ interaction atom from the root of the tree in Fig. 2. The states of the transducer are separating conjunctions of equality atoms, enclosed within square boxes. The transducer replaces the component atoms $\gamma_1 = \langle[\widetilde{x}_1]@q_1, 3\rangle$ with $\gamma_0 = \langle[\widetilde{x}_1]@q_0, 3\rangle$ (resp. γ_0 with γ_1) in the left-most (resp. right-most) leaf of the tree. ⌟

Let $L \subseteq \mathbb{T}(\widetilde{\Sigma})$ be an arbitrary language. The following lemmas prove that the transducer $\mathcal{T}_\tau$ from Def. 21 correctly simulates a havoc step produced by an interaction of type τ.

▶ **Lemma 23.** *For each tree* $\mathfrak{t} \in L$, *such that* $\Phi^\epsilon(\mathfrak{t})$ *is tight, configurations* $\gamma = (\mathcal{C}, \mathcal{I}, \varrho), \gamma' \in \Gamma$ *and store* ν, *such that* $\gamma \models^\nu \Phi^\epsilon(\mathfrak{t})$ *and* $\gamma \xrightarrow{(c_i,p_i)_{i\in[1,n]}} \gamma'$, *for some* $c_1, \ldots, c_n \in \mathcal{C}$ *and* $(c_i, p_i)_{i\in[1,n]} \in \mathcal{I}$, *there exists a tree* $\mathfrak{t}' \in \mathcal{T}_{(p_1,\ldots,p_n)}(L)$, *such that* $\gamma' \models^\nu \Phi^\epsilon(\mathfrak{t}')$.

Note that the condition of $\Phi^\epsilon(\mathfrak{t})$ having only tight models is necessary to avoid interactions $(c_i, p_i)_{i\in[1,n]}$ that fire by "accident" i.e., when the interaction is created by an atom $\langle\zeta_1.p_1, \ldots, \zeta_n.p_n\rangle$, with the components $c_1, \ldots, c_n$ created by component atoms $[\xi_1]@q_1, \ldots, [\xi_n]@q_n$, such that the equality $\xi_i = \zeta_i$ is not the consequence of $\Phi^\epsilon(\mathfrak{t})$, for some $i \in [1, n]$. The effect of such interactions is not captured by the transducer introduced by Def. 21. Tightness is, moreover, a necessary condition of Lemma 13, that ensures the existence of equality walks between the variables occurring in an interaction atom and those of the atoms creating the components to which these variables are mapped, in a model of $\Phi^\epsilon(\mathfrak{t})$.

▶ **Lemma 24.** *For each tree* $\mathfrak{t}' \in \mathcal{T}_{(p_1,\ldots,p_n)}(L)$, *configuration* $\gamma' \in \Gamma$ *and store* ν, *such that* $\gamma' \models^\nu \Phi^\epsilon(\mathfrak{t}')$, *there exists a configuration* $\gamma = (\mathcal{C}, \mathcal{I}, \varrho)$ *and a tree* $\mathfrak{t} \in L$, *such that* $\gamma \models^\nu \Phi^\epsilon(\mathfrak{t})$ *and* $\gamma \xrightarrow{(c_i,p_i)_{i\in[1,n]}} \gamma'$, *for some* $c_1, \ldots, c_n \in \mathcal{C}$ *and* $(c_i, p_i)_{i\in[1,n]} \in \mathcal{I}$.

3.3 The Main Result

We establish the main result of this section, which is a many-one reduction of the havoc invariance to the entailment problem. The result is sharpened by proving that the reduction

- **(i)** preserves the class of the SID (see Def. 25 below), and
- **(ii)** is polynomial when several parameters of the SID are bounded by constants and simply exponential otherwise.

In particular, a class-preserving polynomial reduction ensures that the decidability and complexity upper bounds of the entailment problem carry over to the havoc invariance problem.

▶ **Definition 25.** *For two predicate-free formulæ ϕ and ψ, we write $\phi \simeq \psi$ if and only if they become equivalent when dropping the state atoms from both. For an arity-preserving equivalence relation $\sim \subseteq \mathbb{A} \times \mathbb{A}$ (i.e., $\#\mathsf{A} = \#\mathsf{B}$, for all $\mathsf{A} \sim \mathsf{B}$), for any two rules r_1 and r_2, we write $\mathsf{r}_1 \approx \mathsf{r}_2$ if and only if $\mathsf{r}_1 = \mathsf{A}(x_1, \ldots, x_{\#\mathsf{A}}) \leftarrow \exists y_1 \ldots \exists y_m \ . \ \phi * \mathbin{\ast}_{\ell \in [1,h]} \mathsf{B}_\ell(\mathbf{z}_\ell)$, $\mathsf{r}_2 = \mathsf{A}'(x_1, \ldots, x_{\#\mathsf{A}'}) \leftarrow \exists y'_1 \ldots \exists y'_p \ . \ \psi * \mathbin{\ast}_{\ell \in [1,h]} \mathsf{B}'_\ell(\mathbf{u}_\ell)$, $\exists y_1 \ldots \exists y_m \ . \ \phi \simeq \exists y'_1 \ldots \exists y'_p \ . \ \psi$, $\mathsf{A} \sim \mathsf{A}'$ and $\mathsf{B}_\ell \sim \mathsf{B}'_\ell$, for all $\ell \in [1, h]$. For two SIDs Δ_1 and Δ_2, we write $\Delta_1 \preceq \Delta_2$ if and only if for each rule $\mathsf{r}_1 \in \Delta_1$ there exists a rule $\mathsf{r}_2 \in \Delta_2$, such that $\mathsf{r}_1 \approx \mathsf{r}_2$. We denote by $\Delta_1 \approx \Delta_2$ the conjunction of $\Delta_1 \preceq \Delta_2$ and $\Delta_2 \preceq \Delta_1$.*

If $\mathsf{A}_1 \sim \mathsf{A}_2$ and $\Delta_1 \approx \Delta_2$ then Δ_1-models of $\mathsf{A}_1(x_1, \ldots, x_{\#\mathsf{A}_1})$ differ from the Δ_2-models of $\mathsf{A}_2(x_1, \ldots, x_{\#\mathsf{A}_1})$ only by a renaming of the states occurring within state atoms. This is because any derivation of the satisfaction relation $\gamma \models^{\nu}_{\Delta_1} \mathsf{A}_1(x_1, \ldots, x_{\#\mathsf{A}_1})$ can be mimicked (modulo the state atoms that may change) by a derivation of $\gamma \models^{\nu}_{\Delta_2} \mathsf{A}_2(x_1, \ldots, x_{\#\mathsf{A}_2})$, and viceversa. We are now in the position of stating the main result of this section:

▶ **Theorem 26.** *Assuming that $\mathsf{A}(x_1, \ldots, x_{\#\mathsf{A}})$ is a Δ-tight formula, each instance $\mathsf{HavocInv}[\Delta, \mathsf{A}]$ of the havoc invariance problem can be reduced to a set $\{\overline{\mathsf{A}}_i(x_1, \ldots, x_{\#\mathsf{A}}) \models_{\Delta \cup \overline{\Delta}} \mathsf{A}(x_1, \ldots, x_{\#\mathsf{A}})\}_{i=1}^{p}$ of entailments, where $\Delta \approx \overline{\Delta}$, for an arity-preserving equivalence relation $\sim \subseteq \mathbb{A} \times \mathbb{A}$, such that $\overline{\mathsf{A}}_i \sim \mathsf{A}$, for all $i \in [1, p]$. The reduction is polynomial, if $\#(\Delta)$, $\mathsf{N}(\Delta)$ and $\mathsf{H}(\Delta)$ are bounded by constants and simply exponential, otherwise.*

4 Decidability and Complexity

We prove the undecidability of the havoc invariance problem (Def. 9) using a reduction from the universality of context-free languages, a textbook undecidable problem [2].

▶ **Theorem 27.** *The $\mathsf{HavocInv}[\Delta, \mathsf{A}]$ problem is undecidable.*

The undecidability proof for the havoc invariance problem uses an argument similar to the one used to prove undecidability of the entailment problem [4, Theorem 4]. We leverage further from this similarity and carve a fragment of CL with a decidable havoc invariance problem, based on the reduction from Theorem 26. For self-containment reasons, we recall the definition of a CL fragment for which the entailment problem is decidable (see [4, §6] for more details and proofs). This definition relies on three, easily checkable, syntactic restrictions on the rules of the SID and a decidable semantic restriction on the models of a predicate atom defined by the SID. The syntactic restrictions use the notion of profile:

▶ **Definition 28.** *The* profile *of a SID Δ is the pointwise greatest function $\lambda_\Delta : \mathbb{A} \to \mathrm{pow}(\mathbb{N})$, mapping each predicate A into a subset of $[1, \#\mathsf{A}]$, such that, for each rule $\mathsf{A}(x_1, \ldots, x_{\#\mathsf{A}}) \leftarrow \phi$ from Δ, each atom $\mathsf{B}(y_1, \ldots, y_{\#\mathsf{B}})$ from ϕ and each $i \in \lambda_\Delta(\mathsf{B})$, there exists $j \in \lambda_\Delta(\mathsf{A})$, such that x_j and y_i are the same variable.*

The profile identifies the parameters of a predicate that are always replaced by a variable $x_1, \ldots, x_{\#\mathsf{A}}$ in each unfolding of $\mathsf{A}(x_1, \ldots, x_{\#\mathsf{A}})$, according to the rules in Δ; it is computed by a greatest fixpoint iteration, in polynomial time.

▶ **Definition 29.** *A rule* $\mathsf{A}(x_1, \ldots, x_{\#\mathsf{A}}) \leftarrow \exists y_1 \ldots \exists y_m \,.\, \phi * \mathbin{⨵}_{\ell=1}^{h} \mathsf{B}_\ell(z_1^\ell, \ldots, z_{\#\mathsf{B}_\ell}^\ell)$*, where* ϕ *is a* qpf *formula, is said to be:*

1. progressing *(P) if and only if* $\phi = [x_1] * \psi$*, where* ψ *consists of interaction atoms involving* x_1 *and (dis-)equalities, such that* $\bigcup_{\ell=1}^{h} \{z_1^\ell, \ldots, z_{\#\mathsf{B}_\ell}^\ell\} = \{x_2, \ldots, x_{\#\mathsf{A}}\} \cup \{y_1, \ldots, y_m\}$*,*
2. connected *(C) if and only if, for each* $\ell \in [1, h]$ *there exists an interaction atom in* ψ *that contains both* z_1^ℓ *and a variable from* $\{x_1\} \cup \{x_i \mid i \in \lambda_\Delta(\mathsf{A})\}$*,*
3. equationally-restricted (e-restricted or R) *if and only if, for every disequality* $x \neq y$ *from* ϕ*, we have* $\{x, y\} \cap \{x_i \mid i \in \lambda_\Delta(\mathsf{A})\} \neq \emptyset$*.*

A SID Δ *is* progressing *(P),* connected *(C) and* e-restricted *(R) if and only if each rule in* Δ *is* progressing*,* connected *and* e-restricted*, respectively.*

▶ **Example 30.** For example, the rules for the $\mathsf{chain}_{h,t}(x_1, x_2)$ predicates from the SID in §1.1 are PCR, but not the rules for $\mathsf{ring}_{h,t}()$ predicates, that are neither progressing nor connected. The latter can be replaced with the following PCR rules:

$$\overline{\mathsf{ring}}_{h,t}(x) \leftarrow \exists y \exists z \,.\, [x]@q * \langle x.out, z.in \rangle * \mathsf{chain}_{h',t'}(z, y) * \langle y.out, x.in \rangle, \text{ for all } h, t \in \mathbb{N}$$

Similarly, rule (2) for the *Node* predicate is PCR, but not rules (1) and (3), from Example 12. In order to obtain a SID that is PCR, these rules can be replaced with, respectively:

$$\begin{aligned} \mathit{Root}(n) \leftarrow & \exists n_1 \exists \ell_1 \exists r_1 \exists n_2 \exists \ell_2 \exists r_2 \,.\, [n] * \langle n.req, n_1.reply, n_2.reply \rangle * \langle r_1.in, \ell_2.out \rangle * \\ & \mathit{Node}(n_1, \ell_1, r_1) * \mathit{Node}(n_2, \ell_2, r_2) \\ \mathit{Node}(n, \ell, r) \leftarrow & [n] * \langle n.req, \ell.reply, r.reply \rangle * \langle \ell.in, r.out \rangle * \mathit{Leaf}(\ell) * \mathit{Leaf}(r) \quad \mathit{Leaf}(n) \leftarrow [n] \quad \lrcorner \end{aligned}$$

A first property is that PCR SIDs define only tight configurations (Def. 3), a prerequisite for the reduction from Theorem 26:

▶ **Lemma 31.** *Let* Δ *be a PCR SID and let* $\mathsf{A} \in \mathrm{Def}(\Delta)$ *be a predicate. Then, for any* Δ*-model* (γ, ν) *of* $\mathsf{A}(x_1, \ldots, x_{\#\mathsf{A}})$*, the configuration* γ *is tight.*

The last restriction for the decidability of entailments relates to the degree of the models of a predicate atom. The degree of a configuration is defined in analogy with the degree of a graph as the maximum number of interactions involving a component:

▶ **Definition 32.** *The* degree *of a configuration* $\gamma = (\mathcal{C}, \mathcal{I}, \varrho)$ *is defined as* $\delta(\gamma) \stackrel{\text{def}}{=} \max_{c \in \mathbb{C}} \delta_c(\gamma)$*, where* $\delta_c(\gamma) \stackrel{\text{def}}{=} \|\{(c_1, p_1, \ldots, c_n, p_n) \in \mathcal{I} \mid c = c_i,\ i \in [1, n]\}\|$*.*

For instance, the configuration of the system from Fig. 1 (a) has degree two. The *degree boundedness* problem $\mathsf{DegreeBound}[\Delta, \mathsf{A}]$ asks, given a predicate A and a SID Δ, if the set $\{\delta(\gamma) \mid \gamma \models_\Delta \exists x_1 \ldots \exists x_{\#\mathsf{A}} \,.\, \mathsf{A}(x_1, \ldots, x_{\#\mathsf{A}})\}$ is finite. This problem is decidable [4, Theorem 3]. The entailment problem $\mathsf{A}(x_1, \ldots, x_{\#\mathsf{A}}) \models_\Delta \exists x_{\#\mathsf{A}+1} \ldots \exists x_{\#\mathsf{B}} \,.\, \mathsf{B}(x_1, \ldots, x_{\#\mathsf{B}})$ is known to be decidable for PCR SIDs Δ, provided, moreover, that $\mathsf{DegreeBound}[\Delta, \mathsf{A}]$ holds:

▶ **Theorem 33** ([4])**.** *The entailment problem*

$$\mathsf{A}(x_1, \ldots, x_{\#\mathsf{A}}) \models_\Delta \exists x_{\#\mathsf{A}+1} \ldots \exists x_{\#\mathsf{B}} \,.\, \mathsf{B}(x_1, \ldots, x_{\#\mathsf{B}}),$$

where Δ *is PCR and* $\mathsf{DegreeBound}[\Delta, \mathsf{A}]$ *has a positive answer, is in* 2EXP*, if* $\#(\Delta)$ *and* $\mathsf{N}(\Delta)$ *are bounded by constants and in* 4EXP*, otherwise.*

Back to the havoc invariance problem, we give first a lower bound using a reduction from the entailment problem $\mathsf{A}(x_1, \ldots, x_{\#\mathsf{A}}) \models_\Delta \exists x_{\#\mathsf{A}+1} \ldots \exists x_{\#\mathsf{B}} \,.\, \mathsf{B}(x_1, \ldots, x_{\#\mathsf{B}})$, where Δ is a PCR SID and $\mathsf{HavocInv}[\Delta, \mathsf{A}]$ has a positive answer. To the best of our efforts, we could not prove that the entailment problem is 2EXP-hard under the further assumption that $\#(\Delta)$ is bounded by a constant, which leaves the question of a matching lower bound for the havoc invariance problem open, in this case.

▶ **Lemma 34.** *The* $\mathsf{HavocInv}[\Delta, \mathsf{A}]$ *problem for PCR SIDs* Δ*, such that* $\mathsf{DegreeBound}[\Delta, \mathsf{A}]$ *has a positive answer, is* 2EXP*-hard.*

The main result of this section is a consequence of Theorems 26 and 33. In the absence of a constant bound on the parameters $\#(\Delta)$, $\mathsf{N}(\Delta)$ and $\mathsf{H}(\Delta)$, the entailment resulting from the reduction (Theorem 26) is of simply exponential size in the input and the time complexity of solving the entailments is 4EXP(Theorem 33), yielding a 5EXP upper bound:

▶ **Theorem 35.** *The* $\mathsf{HavocInv}[\Delta, \mathsf{A}]$ *problem, for PCR SIDs such that* $\mathsf{DegreeBound}[\Delta, \mathsf{A}]$ *has a positive answer is in* 2EXP*, if* $\#(\Delta)$*,* $\mathsf{N}(\Delta)$ *and* $\mathsf{H}(\Delta)$ *are bounded by constants and in* 5EXP*, otherwise.*

5 Conclusions

We have considered a logic for describing sets of configurations of parameterized concurrent systems, with user-defined network topology. The havoc invariance problem asks whether a given formula in the logic is invariant under the execution of the system starting from each configuration that is a model of a formula. An algorithm for this problem uses a many-one reduction to the entailment problem, thus leveraging from earlier results on the latter problem. We study the decidability and complexity of the havoc invariance problem and show that a doubly-exponential algorithm exists for a fairly general fragment of the logic, that encompasses all our examples. This result is relevant for automating the generation of correctness proofs for reconfigurable systems, that change the network topology at runtime.

References

1. Emma Ahrens, Marius Bozga, Radu Iosif, and Joost-Pieter Katoen. Local reasoning about parameterized reconfigurable distributed systems. *CoRR*, abs/2107.05253, 2021. `arXiv:2107.05253`.
2. Yehoshua Bar-Hillel, Micha Perles, and Eli Shamir. On formal properties of simple phrase structure grammars. *Sprachtypologie und Universalienforschung*, 14:143–172, 1961.
3. Roderick Bloem, Swen Jacobs, Ayrat Khalimov, Igor Konnov, Sasha Rubin, Helmut Veith, and Josef Widder. *Decidability of Parameterized Verification*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2015.
4. Marius Bozga, Lucas Bueri, and Radu Iosif. Decision problems in a logic for reasoning about reconfigurable distributed systems. In *International Joint Conference on Automated Reasoning, to appear*, 2022. `arXiv:2202.09637`.
5. Marius Bozga, Javier Esparza, Radu Iosif, Joseph Sifakis, and Christoph Welzel. Structural invariants for the verification of systems with parameterized architectures. In *Tools and Algorithms for the Construction and Analysis of Systems - 26th International Conference, TACAS 2020*, volume 12078 of *LNCS*, pages 228–246. Springer, 2020.
6. Marius Bozga and Radu Iosif. Specification and safety verification of parametric hierarchical distributed systems. In *Formal Aspects of Component Software - 17th International Conference, FACS 2021, Virtual Event, October 28-29, 2021, Proceedings*, volume 13077 of *Lecture Notes in Computer Science*, pages 95–114. Springer, 2021.

7 Antonio Bucchiarone and Juan P. Galeotti. Dynamic software architectures verification using dynalloy. *Electron. Commun. Eur. Assoc. Softw. Sci. Technol.*, 10, 2008.

8 Cristiano Calcagno, Peter W. O'Hearn, and Hongseok Yang. Local action and abstract separation logic. In *22nd IEEE Symposium on Logic in Computer Science (LICS 2007), 10-12 July 2007, Wroclaw, Poland, Proceedings*, pages 366–378. IEEE Computer Society, 2007. doi:10.1109/LICS.2007.30.

9 Byron Cook, Christoph Haase, Joël Ouaknine, Matthew J. Parkinson, and James Worrell. Tractable reasoning in a fragment of separation logic. In *CONCUR*, volume 6901 of *Lecture Notes in Computer Science*, pages 235–249. Springer, 2011.

10 Julien Dormoy, Olga Kouchnarenko, and Arnaud Lanoix. Using temporal logic for dynamic reconfigurations of components. In Luís Soares Barbosa and Markus Lumpe, editors, *Formal Aspects of Component Software - 7th International Workshop, FACS 2010*, volume 6921 of *Lecture Notes in Computer Science*, pages 200–217. Springer, 2010.

11 Antoine El-Hokayem, Marius Bozga, and Joseph Sifakis. A temporal configuration logic for dynamic reconfigurable systems. In Chih-Cheng Hung, Jiman Hong, Alessio Bechini, and Eunjee Song, editors, *SAC '21: The 36th ACM/SIGAPP Symposium on Applied Computing, Virtual Event, Republic of Korea, March 22-26, 2021*, pages 1419–1428. ACM, 2021. doi:10.1145/3412841.3442017.

12 Klaus-Tycho Foerster and Stefan Schmid. Survey of reconfigurable data center networks: Enablers, algorithms, complexity. *SIGACT News*, 50(2):62–79, 2019.

13 Dan Hirsch, Paolo Inverardi, and Ugo Montanari. Graph grammars and constraint solving for software architecture styles. In *Proceedings of the Third International Workshop on Software Architecture*, ISAW '98, pages 69–72, New York, NY, USA, 1998. Association for Computing Machinery. doi:10.1145/288408.288426.

14 Yonit Kesten, Amir Pnueli, Elad Shahar, and Lenore D. Zuck. Network invariants in action. In *CONCUR 2002 - Concurrency Theory, 13th International Conference*, volume 2421 of *LNCS*, pages 101–115. Springer, 2002.

15 Arnaud Lanoix, Julien Dormoy, and Olga Kouchnarenko. Combining proof and model-checking to validate reconfigurable architectures. *Electron. Notes Theor. Comput. Sci.*, 279(2):43–57, 2011.

16 Daniel Le Metayer. Describing software architecture styles using graph grammars. *IEEE Transactions on Software Engineering*, 24(7):521–533, 1998. doi:10.1109/32.708567.

17 David Lesens, Nicolas Halbwachs, and Pascal Raymond. Automatic verification of parameterized linear networks of processes. In *The 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 346–357. ACM Press, 1997.

18 Mohammad Noormohammadpour and Cauligi S. Raghavendra. Datacenter traffic control: Understanding techniques and tradeoffs. *IEEE Commun. Surv. Tutorials*, 20(2):1492–1525, 2018.

19 John C. Reynolds. Separation logic: A logic for shared mutable data structures. In *17th IEEE Symposium on Logic in Computer Science (LICS 2002), 22-25 July 2002, Copenhagen, Denmark, Proceedings*, pages 55–74. IEEE Computer Society, 2002. doi:10.1109/LICS.2002.1029817.

20 Ze'ev Shtadler and Orna Grumberg. Network grammars, communication behaviors and automatic verification. In Joseph Sifakis, editor, *Automatic Verification Methods for Finite State Systems, International Workshop*, volume 407 of *LNCS*, pages 151–165. Springer, 1989.

21 Pierre Wolper and Vinciane Lovinfosse. Verifying properties of large sets of processes with network invariants. In *Automatic Verification Methods for Finite State Systems, International Workshop*, volume 407 of *LNCS*, pages 68–80. Springer, 1989.