Parameterized Tree Systems

Parosh Aziz Abdulla[1] parosh@it.uu.se,
Noomene Ben Henda[1] Noomene.BenHenda@it.uu.se,
Giorgio Delzanno[2] giorgio@disi.unige.it,
Frédéric Haziza[1] frederic.haziza@it.uu.se, and
Ahmed Rezine[1] Rezine.Ahmed@it.uu.se

[1] Uppsala University, Sweden
[2] Università di Genova, Italy.

Abstract. Several recent works have considered *parameterized verification*, i.e. automatic verification of systems consisting of an arbitrary number of finite-state processes organized in a *linear array*. The aim of this paper is to extend these works by giving a simple and efficient method to prove safety properties for systems with *tree-like* architectures. A process in the system is a finite-state automaton and a transition is performed jointly by a process and its parent and children processes. The method derives an over-approximation of the induced transition system, which allows the use of finite trees as symbolic representations of infinite sets of configurations. Compared to traditional methods for parameterized verification of systems with tree topologies, our method does not require the manipulation of tree transducers, hence its simplicity and efficiency. We have implemented a prototype which works well on several nontrivial tree-based protocols.

1 Introduction

In recent years, there has been an extensive amount of work on the verification of *parameterized systems*, e.g. [10, 17, 4, 8, 9]. Typically, a parameterized system consists of an arbitrary number of finite-state processes organized in a linear array. The task is to perform *parameterized verification*, i.e. to verify correctness of the system regardless of the number of processes inside the system. Examples of parameterized systems include mutual exclusion algorithms, bus protocols, telecommunication protocols, multi-threaded programs, and cache coherence protocols. This work aims at extending the paradigm of parameterized verification in order to verify systems which operate on tree-like architectures. More precisely, we consider analysis of safety properties for *parameterized tree systems*. Such a system consists of an arbitrary number of finite-state processes which operate on a tree-like architecture. Examples of parameterized tree systems include several interesting protocols such as the percolate protocol [17],the Tree-arbiter protocol [7], and the IEEE 1394 Tree identity protocol [16].

One of the most prominent techniques which have been used for verification of parameterized tree systems is that of *tree regular model checking* [13, 3, 17,

11, 6]. In tree regular model checking, configurations (states) of the system are represented by trees, sets of configurations by tree automata, and transitions by tree automata operating on pairs of trees, i.e. tree transducers. Safety properties can be checked through performing reachability analysis, which amounts to applying the tree transducer relation iteratively to the set of initial configurations. The main problem with transducer-based techniques, such as the ones mentioned above, is that they are very heavy and usually rely on several layers of computationally expensive automata-theoretic constructions; in many cases severely limiting their applicability.

In this paper, we propose a light-weight approach to parameterized tree verification which, in addition to its simplicity, also yields a much more efficient implementation than tree regular model checking. In our method, a configuration of the system is represented by a tree over a finite alphabet, where elements of the alphabet represent the local states of the individual processes. The behaviour of the system is induced by a set of rewriting rules which describe how the processes perform transitions. A transition performed by a process is conditioned by the current local state of the process and possibly the local states of neighboring processes, i.e. the parent and children processes. The transition may change the states of all involved processes. (see Figure 1).

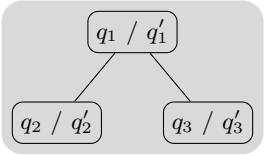


Fig. 1. A typical transition rule where a process and its two children change state from q_1, q_2, q_3 to q_1', q_2', q_3', respectively.

Observe that the set of configurations is infinite since we are dealing with trees of an arbitrary size. In fact, parameterized verification amounts to analyzing an infinite family of systems; namely one for each size of the system and one for each tree of that particular size.

The main idea of our method is to consider a transition relation which is an over-approximation of the one induced by the tree parameterized system. To do so, we modify the semantics of the transition rules, such that a rule is applied to a node and two nodes in its left and right subtrees (rather than its left and right children). The approximate transition system obtained in this manner is *monotonic* with respect to the tree embedding relation on configurations (larger configurations are able to simulate smaller ones). Since the approximate transition relation is monotonic, it can be analyzed using symbolic backward reachability algorithm based on a generic method introduced in [1]. An attractive feature of this algorithm is that it operates on sets of configurations which are upward closed with respect to the tree embedding relation. This allows an

efficient symbolic representation of upward sets of configurations, since such a set can be represented by (the finite set of) its minimal elements. Since the minimal elements are trees, reachability analysis can be performed by computing predecessors of trees, which is much simpler and more efficient than applying transducer relations on general tree regular languages. Also, as a side effect, the analysis of the approximate model is guaranteed to terminate. This follows from the fact that the embedding relation on configurations (trees) is a *well quasi-ordering* by Kruskal's theorem [18]. The whole verification process is fully automatic since both the approximation and the reachability analysis are carried out without user intervention. Observe that if the approximate transition system satisfies a safety property then we can safely conclude that the original system satisfies the property too.

Based on the method, we have implemented a prototype which works well on several tree-based protocols such as the percolate, leader election, Tree-arbiter, and the IEEE 1394 Tree identity protocols.

Outline In the next section, we give some preliminaries on trees. In Section 3, we define the basic model of parameterized tree systems. In Section 4, we describe the induced transition system and in Section 5, we define the over-approximated transition system on which we run our algorithm. We present a generic scheme for deciding reachability of upward closed sets in Section 6, and we show how to instantiate it on our model in Section 7. In Section 9, we report our experimental results on several tree protocols. Section 10 concludes the paper and gives direction for future works. Some proofs as well as the details of the case studies are omitted from the main text and can be found in the appendix.

2 Preliminaries

In this section, we give some basic definitions and notations needed in the rest of the paper. To simplify the presentation, we will only consider binary trees in this paper. However, all the concepts and algorithms can be extended in a straightforward manner in order to deal with trees of higher ranks.

For a set X, we use X^* to denote the set of words over X. We let ε denote the empty word and use $x \bullet x'$ to denote the concatenation of two words $x, x' \in X^*$. We extend the concatenation operation to sets of words $D \subseteq X^*$ by $x \bullet D := \{x \bullet x' |\ x' \in D\}$. Given two words $x, x' \in X^*$, we use $x \leq x'$ to denote that x is a prefix of x'; and use $x < x'$ to denote that $x \leq x'$ and $x \neq x'$. In case $x \leq x'$, we use $x' - x$ to denote the word x'' where $x \bullet x'' = x'$.

Binary Trees *A (binary) tree structure* N is a finite set of words over $\{0, 1\}$ which is closed under the prefix relation, i.e. $n \in N$ and $n' \leq n$ imply $n' \in N$. In the rest of the paper, we fix a finite set of symbols Σ and we use b as a variable ranging over $\{0, 1\}$.

A binary tree (*tree* for short) T over the alphabet Σ is a tuple (N, λ) where N is a tree structure and λ is a mapping from N to Σ. Each element of N is called

a node of T. We say that a node n' is *the parent* of the node n iff $n' \bullet b = n$ for some b. In such a case, n is said to be *a child* of n'. *A leaf* in T is a node which does not have any children; and *the root* of T is the node ε. Given a node n, we define *the descendants of* n by $\mathrm{Desc}(n) := \{n' \in N |\ n < n'\}$. We use $\mathit{Trees}(\Sigma)$ to denote the set of all trees over Σ.

Inclusions and Embeddings Consider two trees $T = (N, \lambda)$ and $T' = (N', \lambda')$ in $\mathit{Trees}(\Sigma)$.

An *inclusion* of T in T' is an injection $f : N \to N'$ such that for any $n \in N$:

- $n \bullet b \in N \implies f(n) \bullet b = f(n \bullet b)$, and
- $\lambda(n) = \lambda'(f(n))$.

We write $T \subseteq_f T'$ to denote that f is an inclusion of T in T', and write $T \subseteq T'$ if $T \subseteq_f T'$ for some inclusion f. Informally, if $T \subseteq T'$ then T' contains a copy of T.

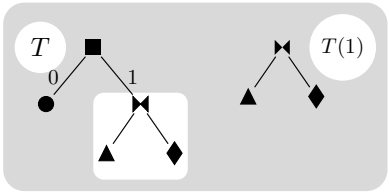
An *embedding* of T in T' is an injection $f : N \to N'$ such that for any $n \in N$:

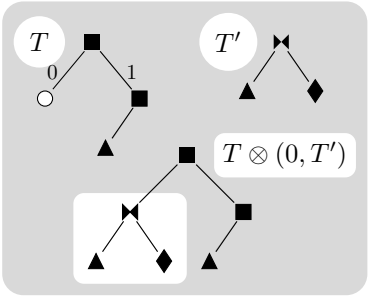
- $n \bullet b \in N \implies f(n) \bullet b \leq f(n \bullet b)$, and
- $\lambda(n) = \lambda'(f(n))$.

We use $T \preceq_f T'$ to denote that f is an embedding of T in T', and write $T \preceq T'$ if $T \preceq_f T'$ for some embedding f. Observe that $\preceq$ is a weaker relation than $\subseteq$. The difference between the two relations is that an inclusion preserves the parent/child relation between nodes, while an embedding preserves a weaker relation, namely that of ascendant/descendant.

Operations on Trees In this paragraph, we fix a tree $T = (N, \lambda) \in \mathit{Trees}(\Sigma)$.

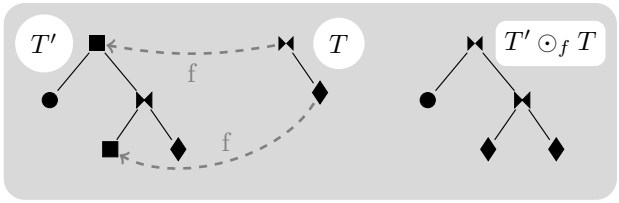
For a node $n \in N$, we use $T(n)$ to denote the subtree of T rooted at n. Formally, we let $T(n) = (N', \lambda')$ where $N' := \{n'' - n |\ n'' \in N \wedge n \leq n''\}$; and for any $n' \in N'$, $\lambda'(n') := \lambda(n \bullet n')$.





Now we fix a tree $T' = (N', \lambda') \in \mathit{Trees}(\Sigma)$ and define the the following operation: Given a node $n \in N$, we denote by $T \otimes (n, T')$ the tree $T'' = (N'', \lambda'')$ where $N'' := (N - \mathrm{Desc}(n)) \bigcup (n \bullet N')$ and for any $n'' \in N''$, $\lambda''(n'') := \lambda(n'')$ if $n \not\leq n''$, and $\lambda''(n'') := \lambda'(n'' - n)$ otherwise. Intuitively, we obtain T'' by replacing in T the subtree rooted at n by T'.

Consider a (partial) function $f : N \rightharpoonup N'$. We define the *renaming of* T' *with respect to* f *and* T, denoted by $T' \odot_f T$, to be the tree $T'' = (N', \lambda'')$ where for any $n' \in N'$, $\lambda''(n') = \lambda'(n')$ if $n' \notin \mathrm{Img}(f)$, and $\lambda''(n') = \lambda(f^{-1}(n'))$ otherwise.



3 Parameterized Tree Systems

A parameterized tree system consists of an arbitrary (but finite) number of identical processes, arranged in a (binary) tree topology. Each process is a finite-state automaton. The transitions of the automaton are conditioned by the current local state and possibly the local states of other processes (parent, children, etc). A transition may change the states of all processes involved in the condition. A parameterized tree system induces an infinite family of finite-state systems, namely one for each size and each structure of the tree. The aim is to verify correctness of the systems for the whole family regardless of the number of processes in the system or the particular form of the tree.

Formally, a parameterized tree system $\mathcal{P}$ is a tuple (Q, R) where Q is a finite set of *local states*, and $R \subseteq \mathit{Trees}(Q \times Q)$ is a finite set of trees called *rewrite rules*. For each rule $r = (N, \lambda) \in R$, we associate two special trees in $\mathit{Trees}(Q)$ called *left* and *right* trees of r, and denoted respectively by $\mathit{lhs}(r)$ and $\mathit{rhs}(r)$. We define $\mathit{lhs}(r) := (N, \mathit{lhs}(\lambda))$ and $\mathit{rhs}(r) := (N, \mathit{rhs}(\lambda))$, where $\mathit{lhs}(\lambda)$ and $\mathit{rhs}(\lambda)$ are obtained from λ by projecting on the first and the second component of $Q \times Q$. More precisely, for any node $n \in N$, if $\lambda(n) = (q, q')$ then $\mathit{lhs}(\lambda)(n) := q$ and $\mathit{rhs}(\lambda)(n) := q'$.

Example 1. We consider the percolate protocol where the set of states Q is defined by $\{q_0, q_1, q_u\}$ and the transition rules $R = \{r_1, r_2, r_3, r_4\}$ are as depicted in Figure 2. The protocol evaluates the disjunction of the values in the leaves up to the root.

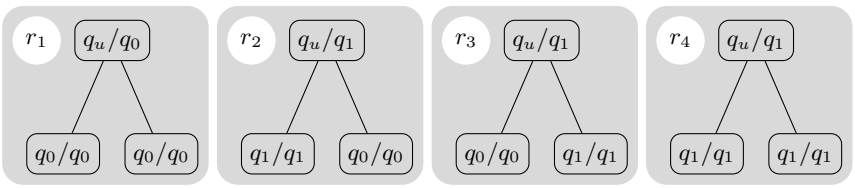


Fig. 2. The transition rules of the percolate protocol.

4 Operational Semantics

The operational semantics of a parameterized tree system can be captured by a transition system. In this section, we first describe the induced transition system. Then we introduce the *coverability problem*.

Transition System A *transition system* $\mathcal{T}$ is a pair $(C, \Longrightarrow)$, where C is an (infinite) set of *configurations* and $\Longrightarrow$ is a binary relation on C. We use $\stackrel{*}{\Longrightarrow}$ to denote the reflexive transitive closure of $\Longrightarrow$. Given an ordering $\trianglelefteq$ on C, we say that $\mathcal{T}$ is *monotonic with respect to* $\trianglelefteq$ if the following holds: For any configurations $c_1, c_2, c_3 \in C$ with $c_1 \Longrightarrow c_3$ and $c_1 \trianglelefteq c_2$, there is a configuration $c_4 \in C$ such that $c_2 \Longrightarrow c_4$ and $c_3 \trianglelefteq c_4$. We will consider several transition systems in this paper.

First, a parameterized system $\mathcal{P} = (Q, R)$ induces a transition system $\mathcal{T}(\mathcal{P}) = (C, \longrightarrow)$ where $C = \mathit{Trees}(Q)$. Intuitively, a configuration $c = (N, \lambda) \in C$ represents an instance of the system with $|N|$ processes. These processes are arranged according to the tree structure N and their current local states are given by λ. More precisely, each node $n \in N$ represents a process in the state $\lambda(n)$.

Next, we define the transition relation $\longrightarrow$ on the set of configurations as follows. Let $r \in R$ be a rewrite rule. Consider two configurations c_1 and c_2. We write $c_1 \stackrel{r}{\longrightarrow} c_1$ to denote that there is an f such that the following conditions hold: (i) $\mathit{lhs}(r) \subseteq_f c_1$, and (ii) $c_2 = c_1 \odot_f \mathit{rhs}(r)$. Intuitively, c_2 can be derived from c_1 by changing the labels of all the nodes in $\mathrm{Img}(f)$ according to the labeling function of $\mathit{rhs}(r)$. Below, we give informal explanations of the conditions. First, in condition (i), we identify the "active processes" (those which participate in the transition) by the inclusion f ($\mathrm{Img}(f)$). Implicitly, we interpret $\mathit{lhs}(r)$ as a guard and therefore require, through condition (i), that the configuration c_1 contains a tree which is a copy of the left hand side of the rule. Then, in condition (ii), we interpret $\mathit{rhs}(r)$ as an operation and require that, in c_2, the processes in $\mathrm{Img}(f)$ (the active ones) should all change state according to $\mathit{rhs}(r)$. Observe that the local states of the "passive processes", i.e. those not participating in the transition, should remain unchanged through the transition, and also that the transition does not change the structure of the tree [3] (see Figure 3).

We use $c \longrightarrow c'$ to denote that $c \stackrel{r}{\longrightarrow} c'$ for some rule $r \in R$.

Safety Properties In order to analyze safety properties, we study the *coverability problem* defined below. For a parameterized tree system $\mathcal{P} = (Q, R)$, we assume that we are given a set of initial configurations *Init*, each of which characterizes a possible state of the system prior to starting the execution.

We recall the definition of the relation $\preceq$ defined in Section 2. A set of configurations $D \subseteq C$ is said to be *upward closed* (with respect to $\preceq$) if $c \in D$ and $c \preceq c'$ implies $c' \in D$. For sets of configurations $D, D' \subseteq C$ we use $D \longrightarrow D'$

[3] In fact, our method can also cope with non-structure preserving rules, such dynamic creation and deletion of processes. However, for simplicity of presentation, we choose not to do so.

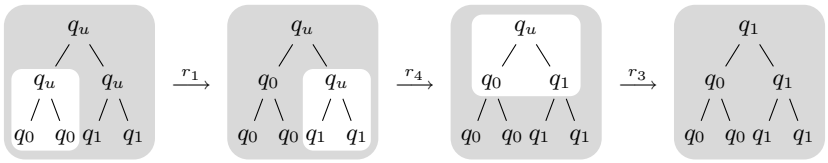


Fig. 3. A possible run of the percolate protocol. We highlight in white the zone where the rule applies (see Example 1).

to denote that there are $c \in D$ and $c' \in D'$ with $c \longrightarrow c'$. The *coverability problem* for parameterized tree systems is defined as follows:

PAR-TREE-COV
Instance
- A parameterized tree system $\mathcal{P} = (Q, R)$.
- An upward closed set F of configurations.

Question $\mathit{Init} \xrightarrow{*} F$?

It can be shown, using standard techniques (see [19, 14]), that checking safety properties (expressed as regular languages) can be translated into instances of the coverability problem. Therefore, checking safety properties amounts to solving PAR-TREE-COV (i.e. to the reachability of upward closed sets).

5 Approximation

In this section, we introduce an over-approximation of the transition relation of a parameterized tree system.

In Section 4, we mentioned that each parameterized tree system $\mathcal{P} = (Q, R)$ induces a transition system $\mathcal{T}(\mathcal{P}) = (C, \longrightarrow)$. A parameterized tree system $\mathcal{P}$ also induces an *approximate* transition system $\mathcal{A}(\mathcal{P}) = (C, \rightsquigarrow)$, where the set C of configurations is identical to the one in $\mathcal{T}(\mathcal{P})$ and the transition relation $\rightsquigarrow$ is defined below.

First, we define a special operation on trees needed in order to describe the semantics of $\rightsquigarrow$.

Tree Subtraction In this paragraph, we fix two trees $T = (N, \lambda), T' = (N', \lambda') \in \mathit{Trees}(\Sigma)$ such that $T' \preceq_f T$ for some embedding f. We define $T \ominus_f T'$ to be the tree T'' obtained from T' by performing a sequence of operations described below. First, we enumerate the nodes of T' in a bottom-up fashion. Formally, let $\{n_i\}_{1 \leq i \leq |N|}$ be an enumeration of the set N' of nodes in T' such that for any $i, j : 1 \leq i \neq j \leq |N|$, $n_i < n_j$ implies that $j < i$. In other words, if n_j is a descendant of n_i in T', then n_j occurs earlier than n_i in the enumeration. Based on the enumeration, we define a sequence of trees $\{T_i\}_{1 \leq i \leq |N|-1}$ as follows. We let $T_1 := T$. For any $i : 1 \leq i \leq |N| - 2$, we denote by n_i^p the parent of n_i, i.e.

$n_i^p \bullet b = n_i$ for some b; and we define

$$T_{i+1} := T_i \otimes (f(n_i^p) \bullet b, T(f(n_i))) .$$

Finally, we let $T'' := T_{|N|-1}$. In other words, we go through the nodes of T' one by one in a bottom-up manner. For each node n_i and its parent n_i^p in T' (say $n_i^p \bullet b = n_i$ for some b), we consider their images $f(n_i^p)$ and $f(n_i)$ in T. We replace the subtree rooted in the child of the image $f(n_i^p) \bullet b$ by the one rooted in the image $f(n_i)$ (see Figure 4). Notice that the resulting tree T'' and the trees T', T are related by $T' \subseteq T'' \preceq T$. In the sequel, we denote by $\widehat{f}$ the inclusion of T' in T'' such that $\widehat{f}(\varepsilon) = f(\varepsilon)$ (such a function exists and is unique by the definition above).

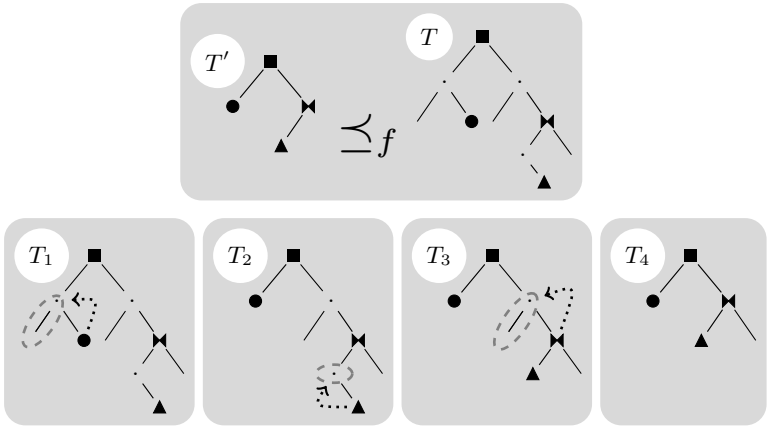


Fig. 4. In the first row, we give an example of two trees T, T' satisfying $T' \preceq_f T$ for some embedding f. In the second row, we give the sequence of trees used in the definition of $T \ominus_f T'$. In each of the trees, the arrow shows where subtrees are re-rooted, while the nodes surrounded by a discontinuous line are those which are removed.

The Approximate Transition Relation Consider two configurations c_1, c_2 and a rule $r \in R$. We write $c_1 \overset{r}{\rightsquigarrow} c_2$ to denote that there is an f such that (i) $rhs(r) \preceq_f c_1$, and (ii) $c_2 = (c_1 \ominus_f lhs(r)) \odot_{\widehat{f}} rhs(r)$. Intuitively, starting from c_1 and an embedding f of $lhs(r)$ in c_1, we first remove all nodes in c_1 such that $lhs(r)$ is included in the resulting configuration. This is done by taking $lhs(r) \ominus_f c_1$ and the inclusion $\widehat{f}$. Then we apply the rule r and obtain c_2 from $lhs(r) \ominus_f c_1$ in a similar manner to how it is described in the previous section, i.e. by renaming the labels of the nodes in $\mathrm{Img}(\widehat{f})$ according to $rhs(r)$ (see Figure 5). We use $c_1 \rightsquigarrow_1 c_2$ if $c_1 \overset{r}{\rightsquigarrow} c_2$ for some $r \in R$.

Observe that the relation $\rightsquigarrow$ is an over-approximation of the transition relation defined in the previous section (i.e. $\rightsquigarrow \supseteq \longrightarrow$) by the following argument.

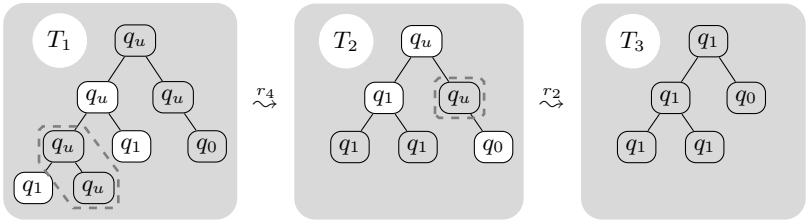


Fig. 5. A possible run of the approximate transition system induced by the percolate protocol (see Example 1). The nodes with a white background represent those where the rule will apply while the discontinuous lines surround the nodes which are removed.

Consider two configurations $c_1, c_2 \in C$ with $c_1 \longrightarrow c_2$. By definition, this implies the existence of a rule $r \in R$ and an inclusion f of $lhs(r)$ in c_1 such that $c_2 = c_1 \odot_f rhs(r)$. Observe that, by definition of the $\ominus$ operation, since f is an inclusion it follows that $c_1 \ominus_f lhs(r) = c_1$ and $\widehat{f} = f$. Therefore, we obtain $c_2 = c_1 \odot_f rhs(r) = (c_1 \ominus_f lhs(r)) \odot_{\widehat{f}} rhs(r)$, and as a consequence, $c_1 \overset{r}{\rightsquigarrow} c_2$.

We are now ready to state a key property of the approximated transition system.

Lemma 1. *The approximate transition system $(C, \rightsquigarrow)$ is monotonic with respect to $\preceq$.*

We define the coverability problem for the approximate system as follows.

APRX-PAR-TREE-COV
Instance
- A parameterized tree system $\mathcal{P} = (Q, R)$
- An upward closed set F of configurations.

Question $Init \overset{*}{\rightsquigarrow} F$?

Since $\longrightarrow \subseteq \rightsquigarrow$, a negative answer to APRX-PAR-TREE-COV implies a negative answer to PAR-TREE-COV.

6 Scheme

In this section, we recall a generic scheme from [1] for performing symbolic backward reachability analysis. The scheme in question is based on symbolic representations of infinite sets of configurations called *constraints*. Throughout this section, we fix a transition system $\mathcal{T} = (C, \Longrightarrow)$ and a set $Init \subseteq C$ of initial configurations.

Constraint Systems A *constraint system* Ψ relative to the transition system $\mathcal{T}$ is a set whose elements are called constraints and can be finitely encoded, such that there is a function $[\![\cdot]\!] : \Psi \to 2^C$. For a finite set Φ of constraints, we let $[\![\Phi]\!] = \bigcup_{\phi \in \Phi} [\![\phi]\!]$. We say that a set $D \subseteq C$ is *computable* or *representable* (in the

constraint system Ψ) if it is possible to compute a finite set of constraints $\Phi \subseteq \Psi$ such that $D = [\![\Phi]\!]$.

We define an *entailment relation* $\sqsubseteq$ on constraints, where $\phi_1 \sqsubseteq \phi_2$ iff $[\![\phi_2]\!] \subseteq [\![\phi_1]\!]$. For sets Φ_1, Φ_2 of constraints, abusing notation, we let $\Phi_1 \sqsubseteq \Phi_2$ denote that for each $\phi_2 \in \Phi_2$ there is a $\phi_1 \in \Phi_1$ with $\phi_1 \sqsubseteq \phi_2$. Notice that $\Phi_1 \sqsubseteq \Phi_2$ implies that $[\![\Phi_2]\!] \subseteq [\![\Phi_1]\!]$.

For a constraint ϕ, we let $\mathrm{Pre}(\phi)$ be the set of constraints, such that $[\![\mathrm{Pre}(\phi)]\!] = \{c |\ \exists c' \in [\![\phi]\!].\ c \Longrightarrow c'\}$. In other words, $\mathrm{Pre}(\phi)$ characterizes the set of configurations from which we can reach a configuration in ϕ through the application of a single rewrite rule. Such a set does not necessarily exist, nevertheless, for our class of systems, we will show that such a set always exists and is in fact computable. For a set Φ of constraints, we let $\mathrm{Pre}(\Phi) = \bigcup_{\phi \in \Phi} \mathrm{Pre}(\phi)$.

Symbolic Backward Reachability We present a scheme for a symbolic algorithm which, given a finite set Φ_F of constraints, checks whether $\mathit{Init} \overset{*}{\Longrightarrow} [\![\Phi_F]\!]$.

In the scheme, we perform a backward reachability analysis, generating a sequence $\{\Phi_i\}_{i \in \mathbb{N}} : \Phi_0 \sqsupseteq \Phi_1 \sqsupseteq \Phi_2 \sqsupseteq \cdots$ of finite sets of constraints such that $\Phi_0 = \Phi_F$, and $\Phi_{i+1} = \Phi_i \cup \mathrm{Pre}(\Phi_i)$. Since $[\![\Phi_0]\!] \subseteq [\![\Phi_1]\!] \subseteq [\![\Phi_2]\!] \subseteq \cdots$, the procedure terminates when we reach a point j where $\Phi_j \sqsubseteq \Phi_{j+1}$. Consequently, Φ_j characterizes the set of all predecessors of $[\![\Phi_F]\!]$. This means that $\mathit{Init} \overset{*}{\Longrightarrow} [\![\Phi_F]\!]$ iff $\mathit{Init} \cap [\![\Phi_j]\!] \neq \emptyset$.

Observe that, in order to implement the scheme (i.e. transform it into an algorithm), we need to be able to (i) compute Pre; (ii) check for entailment between constraints; and (iii) check for emptiness of $\mathit{Init} \cap [\![\phi]\!]$ for any constraint ϕ. A constraint system satisfying these three conditions is said to be *effective*. Moreover, in [1], it is shown that termination is guaranteed in case the constraint system is *well quasi-ordered (WQO)* with respect to $\sqsubseteq$, i.e. for each infinite sequence $\phi_0, \phi_1, \phi_2, \ldots$ of constraints, there are $i < j$ with $\phi_i \sqsubseteq \phi_j$.

7 Algorithm

In this section, we instantiate the scheme of Section 6 to derive an algorithm for solving APRX-PAR-TREE-COV. We do that by introducing an effective and well quasi-ordered constraint system.

Throughout this section, we assume a parameterized tree system $\mathcal{P} = (Q, R)$ and the induced approximate transition system $\mathcal{A}(\mathcal{P}) = (C, \rightsquigarrow)$. We define a constraint to be a tree in $\mathit{Trees}(Q)$. Although we use the same syntax as for configurations, constraints are interpreted differently. More precisely, given a constraint ϕ, we let $[\![\phi]\!] = \{c \in C |\ \phi \preceq c\}$.

An aspect of our constraint system is that each constraint characterizes a set of configurations which is upward closed with respect to $\preceq$. Conversely (by Higman's Lemma [15]), any upward closed set F of configurations can be characterized as $[\![\Phi_F]\!]$ where Φ_F is a finite set of constraints. In this manner, APRX-PAR-TREE-COV is reduced to checking the reachability of a finite set of constraints.

Below we show effectiveness and well quasi-ordering of our constraint system, meaning that we obtain an algorithm for solving APRX-PAR-TREE-COV. First, observe that the entailment relation can be computed in a straightforward manner since for any constraints ϕ, ϕ', we have $\phi \sqsubseteq \phi'$ iff $\phi \preceq \phi'$.

In order to check the initial condition, we rely on previous works on *regular tree languages* [12] and provide a sufficient condition on *Init* which guarantees effectiveness of $Init \cap [\![\phi]\!] = \emptyset$ for any constraint ϕ. More precisely, we require that the set *Init* can be characterized by a regular tree language.

For the computation of Pre we rely on the following result.

Lemma 2. *For any constraint ϕ, the set of constraints Pre(ϕ) is computable and finite.*

It was shown in [18] that the embedding relation on trees $\preceq$ is a well quasi-order (Kruskal's theorem). This combined with results in [1] guarantee termination of our scheme when instantiated on the constraints we have defined above.

8 Case Studies

In this section, we provide descriptions of two tree protocols we have analyzed using our method. For each protocol, we define the corresponding parameterized tree system model and we give the sets of unsafe (F) and initial ($Init$) configurations.

8.1 The Tree-arbiter Protocol

The protocol supervises the access to a shared resource of a set of processes arranged in a tree topology. The processes competing for the resource reside in the leaves.

A process in the protocol can be in state *idle* (i), *requesting* (r), *token* (t) or *below* (b). All the processes are initially in state i. A node is in state b whenever is has a descendant in state t. When a leaf is in state r, the request is propagated upwards until it encounters a node which is aware of the presence of the token (i.e. a node in state t or b). A node that has the token (in state t) can choose to pass it upwards or pass it downwards to a requesting child (node in state r).

We model the tree-arbiter protocol with a parameterized tree system $\mathcal{P} = (Q, R)$ where $Q = \{q_s^n |\ s \in \{i, r, t, b\} \wedge n \in \{leaf, inner, root\}\}$ and R is as depicted in the figure below (figure 6). Observe that in the definition of Q, we use the scripts s and n to model respectively the state and the nature (leaf, inner or root) of the nodes. In the definition of the rules, we will drop the script(s) whenever we mean that it is arbitrary (it can take any value).

The rules to model this protocol are as follows: 2 rules to propagate the request upwards, 2 rules to propagate the token downwards, 2 rules to propagate the token upwards and one rule to initiate a request from a leaf.

The set of bad constraints F is represented by trees where at least two processes (i.e. two leaves) obtain the token (i.e. in state q_t^{leaf}).

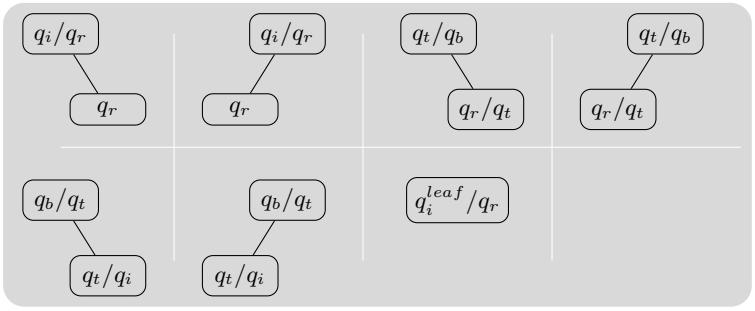
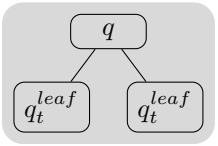


Fig. 6. The rewrite rules for the tree-arbiter protocol. We mention here that there are more rules in the model we have verified. For example, the rule in the top-left corner is represented in the concrete model by 2 rules, each of which corresponds to a particular combination of the natures of the parent and child nodes: For the parent there are 2 possibilities (q_i^{inner}/q_r^{inner} and q_i^{root}/q_r^{root}) while for the child, there are 2 (q_r^{inner} and q_r^{leaf}).

The set of initial configurations *Init* contains all trees where the leaf nodes are either idle or requesting, inner nodes are idle, and the root has the token.



8.2 The IEEE 1394 Tree Identification Protocol

The 1394 High Performance serial bus [16] is used to transport digitized video and audio signals within a network of multimedia systems and devices.

The tree identification protocol is used in one of the phases implementing the IEEE 1394 protocol. More precisely, it is run after a bus reset in the network and leads to the election of a unique leader node.

In this section, we consider a version working on tree topologies. Furthermore, we assume that (i) each inner node is connected to 3 neighbors, (ii) the root is connected to 2 neighbors, and (ii) communication is atomic.

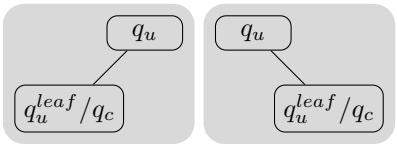
Initially, all nodes are in state *undefined* (u). We identify two steps in the protocol depending on the number n of neighbors which are still in state u. If $n > 1$, the node waits for ("be my parent") requests from its neighbors. If $n = 1$, the node sends a request to the remaining neighbor in state u. Observe that we implicitly assume that the leaf nodes are the first to communicate with their neighbors.

Formally, we derive a parameterized tree system model $\mathcal{P} = (Q, R)$ as follows. We define the set of states by $Q = \{q_s^n |\ s \in \{u, c, l\} \wedge n \in \{leaf, inner, root\}\}$ where the scripts s and n describe respectively the state and the nature of the node. In the definition of the state (s), the letters u, c and l stand respectively for *undefined*, *child* and *leader*. In a similar manner to the previous section, we

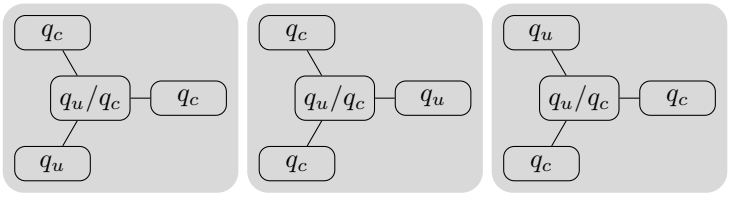
drop the script(s) whenever we mean that it can take any value (see caption of Figure 6).

The rewrite rules R are described below.

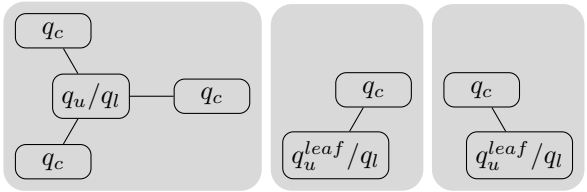
– The leaves initiate the communications:



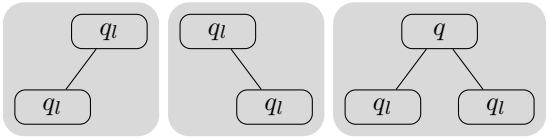
– The inner nodes become children or wait for requests:



– The leader is chosen:



The set of initial configurations *Init* is represented by trees where all nodes are in state undefined, and the set of bad constraints F is represented by trees where at least 2 leaders are elected.



9 Experiments

We have implemented a prototype tool in C++ and run it on several models of protocol with tree-like topologies. The experiments have been performed on a dual Opteron 2.8 GHz, with 8 GB of RAM memory and the results are reported in Table 1.

For each example, we give the number of iterations performed by the reachability algorithm, the largest number of constraints maintained at the end of the execution, the time and total memory consumption. Full details of the examples can be found in the appendix.

Table 1. Experimental Results

Protocol	Time	# iterations	# constraints	Memory
Token	1s	1	3	<1 MB
Two way token	1s	1	3	<1 MB
Percolate	1s	1	2	<1 MB
Leader	1s	4	41	63 MB
Tree Arbiter	37s	12	1173	70 MB
IEEE 1394	1h15m25s	17	4145	137 MB

10 Conclusions and Future Work

We have presented a method for verification of tree parameterized systems where the components are organized in a tree. We derive an over-approximation of the transition relation which allows the use of symbolic reachability analysis defined on upward closed sets of trees (configurations). This technique has been implemented and successfully tested on a number of tree-based protocols.

It would be interesting to see if one can extend our method to other classes of architectures such as unordered trees, DAGs, and more general classes of graphs. In a similar manner to the case of words [2] we intend to consider tree systems where the individual processes may contain unbounded variables. This would allow to analyze algorithms for manipulation of heaps, (balanced) binary trees, etc. Finally, we intend to extend our framework to check for liveness properties on tree-like architecture systems (as done for words in [5]).

References

1. P. A. Abdulla, K. Čerāns, B. Jonsson, and Y.-K. Tsay. General decidability theorems for infinite-state systems. In *Proc. LICS '96,* 11^{th} *IEEE Int. Symp. on Logic in Computer Science*, pages 313–321, 1996.
2. P. A. Abdulla, G. Delzanno, and A. Rezine. Parameterized verification of infinite-state processes with global conditions. In *Proc.* 19^{th} *Int. Conf. on Computer Aided Verification*, volume 4590 of *Lecture Notes in Computer Science*, pages 145–157, 2007.
3. P. A. Abdulla, B. Jonsson, P. Mahata, and J. d'Orso. Regular tree model checking. In *Proc.* 14^{th} *Int. Conf. on Computer Aided Verification*, volume 2404 of *Lecture Notes in Computer Science*, 2002.
4. P. A. Abdulla, B. Jonsson, M. Nilsson, and J. d'Orso. Regular model checking made simple and efficient. In *Proc. CONCUR 2002,* 13^{th} *Int. Conf. on Concurrency Theory*, volume 2421 of *Lecture Notes in Computer Science*, pages 116–130, 2002.
5. P. A. Abdulla, B. Jonsson, M. Nilsson, J. d'Orso, and M. Saksena. Regular model checking for s1s + ltl. In *CAV04*, Lecture Notes in Computer Science, pages 348–360, Boston, July 2004. Springer Verlag.
6. P. A. Abdulla, A. Legay, J. d'Orso, and A. Rezine. Tree regular model checking: A simulation-based approach. *The Journal of Logic and Algebraic Programming*, 69(1-2):93–121, 2006.

7. R. Alur, R. K. Brayton, T. A. Henzinger, S. Qadeer, and S. K. Rajamani. Partial-order reduction in symbolic state space exploration. In *Proc.* 9^{th} *Int. Conf. on Computer Aided Verification*, volume 1254, pages 340–351, 1997.
8. B. Boigelot, A. Legay, and P. Wolper. Iterating transducers in the large. In *Proc.* 15^{th} *Int. Conf. on Computer Aided Verification*, volume 2725 of *Lecture Notes in Computer Science*, pages 223–235, 2003.
9. A. Bouajjani, P. Habermehl, and T. Vojnar. Abstract regular model checking. In *CAV04*, Lecture Notes in Computer Science, pages 372–386, Boston, July 2004. Springer Verlag.
10. A. Bouajjani, B. Jonsson, M. Nilsson, and T. Touili. Regular model checking. In Emerson and Sistla, editors, *Proc.* 12^{th} *Int. Conf. on Computer Aided Verification*, volume 1855 of *Lecture Notes in Computer Science*, pages 403–418. Springer Verlag, 2000.
11. A. Bouajjani and T. Touili. Extrapolating Tree Transformations. In *Proc.* 14^{th} *Int. Conf. on Computer Aided Verification*, volume 2404 of *Lecture Notes in Computer Science*, 2002.
12. H. Common, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. *Tree Automata Techniques and Applications.* October 1999.
13. D. Dams, Y. Lakhnech, and M. Steffen. Iterating transducers. In G. Berry, H. Comon, and A. Finkel, editors, *Computer Aided Verification*, volume 2102 of *Lecture Notes in Computer Science*, 2001.
14. P. Godefroid and P. Wolper. Using partial orders for the efficient verification of deadlock freedom and safety properties. *Formal Methods in System Design*, 2(2):149–164, 1993.
15. G. Higman. Ordering by divisibility in abstract algebras. *Proc. London Math. Soc. (3)*, 2(7):326–336, 1952.
16. IEEE Computer Society. IEEE standard for a high performance serial bus. Std 1394-1995, Aug. 1996.
17. Y. Kesten, O. Maler, M. Marcus, A. Pnueli, and E. Shahar. Symbolic model checking with rich assertional languages. *Theoretical Computer Science*, 256:93–112, 2001.
18. J. Kruskal. Well-quasi-ordering, the tree theorem, and Vazsonyi's conjecture. *Transactions of the American Mathematical Society*, 95:210–225, 1960.
19. M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. LICS '86,* 1^{st} *IEEE Int. Symp. on Logic in Computer Science*, pages 332–344, June 1986.

A Appendix – Proofs

A.1 Approximation

We devote this section to proving Lemma 1. In order to do that, we rely on several auxiliary results related to the tree operations from Section 2.

Auxiliary Results First, we consider the $\ominus$ operation and show the following property.

Lemma A1. *For any trees $T, T' \in Trees(\Sigma)$ such that $T \preceq_f T'$ for some embedding f, the following holds: There are two functions $\widehat{f}, \widetilde{f}$, such that:*

1. $T \subseteq_{\widehat{f}} T' \ominus_f T \preceq_{\widetilde{f}} T'$.
2. $\widetilde{f} \circ \widehat{f} = f$.

Proof. By definition of the $\ominus$ operation, we know that $T \subseteq T' \ominus_f T \preceq T'$. Assume that $T, T', T' \ominus_f T$ are respectively of the forms $(N, \lambda), (N', \lambda'), (N'', \lambda'')$. Below, we define the functions $\widehat{f} : N \to N''$ and $\widetilde{f} : N'' \to N'$ such that the lemma statements hold.

First, for any $n \in N$, we define $\widehat{f}(n) := f(\varepsilon) \bullet n$. Second, for any $n'' \in N''$, we consider three cases reflecting the membership of $n'' \in N''$:

- If $f(\varepsilon) \not\leq n''$, then we let $\widetilde{f}(n'') := n''$.
- If $n'' \in \mathrm{Img}(\widehat{f})$, then we take $\widetilde{f} := f(\widehat{f}^{-1}(n''))$.
- Otherwise, since $f(\varepsilon) \leq n''$ and $n'' \notin \mathrm{Img}(\widehat{f})$, there is a unique node $n''_l \in \mathrm{Img}(\widehat{f})$ such that $n''_l \leq n''$ and for any other node $n''_o \in \mathrm{Img}(\widehat{f})$ the following holds: $n''_o \leq n'' \implies n''_o \leq n''_l$. In other words, n''_l is the longest prefix of n'' in $\mathrm{Img}(\widehat{f})$. We define $\widetilde{f}(n'') := f(\widehat{f}^{-1}(n''_l)) \bullet (n'' - n''_l)$.

Observe now that following the above definitions, properties (1) and (2) trivially hold. □

Then, we show how the $\ominus$ operation preserves the inclusion ($\subseteq$) and embedding ($\preceq$) relations in the following sense.

Lemma A2. *For any $T, T', T'' \in Trees(\Sigma)$ and any mappings f and g, the following hold:*

1. $T' \preceq_g T'' \implies T' \odot_f T \preceq T'' \odot_{g \circ f} T$, *and*
2. $T \preceq_f T' \preceq_g T'' \implies T' \ominus_f T \preceq T'' \ominus_{g \circ f} T$.

Proof. Property (1) holds trivially since by definition of the $\odot$ operation $T' \odot_f T \preceq_g T'' \odot_{g \circ f} T$.

In order to prove (2), we provide a mapping h which guarantees that $T' \ominus_f T \preceq_h T'' \ominus_{g \circ f} T$. First, we assume that $T' \ominus_f T$ is of the form (N', λ'). Then, we recall from Lemma A1 that there are some functions $\widehat{f}, \widetilde{f}, \widehat{g \circ f}, \widetilde{g \circ f}$ such that:

- $T \subseteq_{\widehat{f}} T' \ominus_f T \preceq_{\widetilde{f}} T'$, and

– $T \subseteq_{\widehat{g\circ f}} T'' \ominus_{g\circ f} T \preceq_{\widetilde{g\circ f}} T''$.

For a node $n' \in N'$, we define $h(n')$ by three cases reflecting the membership of n'.

– If $f(\varepsilon) \not\leq n'$, we define $h(n') := g(n')$.
– If $n' \in \mathrm{Img}(\widehat{f})$, then we take $h(n') := g(f(\varepsilon)) \bullet (n' - f(\varepsilon))(= \widehat{g \circ f}(\widehat{f}^{-1}(n')))$.
– Otherwise, we let $h(n') := \widetilde{g \circ f}^{-1}(g(\widetilde{f}(n')))$. Observe that this is well defined (definition of $\widetilde{g \circ f}$).

□

We are now ready to prove the lemma.

Proof of Lemma 1 We recall the lemma in question.

Lemma 1. *The approximate transition system $(C, \rightsquigarrow)$ is monotonic with respect to $\preceq$.*

Proof. Consider three configurations $c_1, c_2, c_3 \in C$ such that $c_1 \preceq_g c_3$ for some embedding g, and $c_1 \rightsquigarrow c_2$. The task is to prove that there is a configuration $c_4 \in C$ such that $c_2 \preceq c_4$ and $c_3 \rightsquigarrow c_4$.

By definition $c_1 \rightsquigarrow c_2$ iff there is a rule $r \in R$ and a mapping f such that $lhs(r) \preceq_f c_1$ and $(c_1 \ominus_f lhs(r)) \odot_{\widehat{f}} rhs(r) = c_2$. By definition of f and g, it follows that $lhs(r) \preceq_{g\circ f} c_3$. Let $c_4 = (c_3 \ominus_{g\circ f} lhs(r)) \odot_{\widehat{g\circ f}} rhs(r)$. As a consequence and by definition of the transition relation $\rightsquigarrow$, we obtain $c_3 \rightsquigarrow c_4$. It remains to show that $c_2 \preceq c_4$ which gives the result.

By Lemma A2(2) and the fact that $lhs(r) \preceq_f c_1 \preceq_g c_3$, we have $c_1 \ominus_f lhs(r) \preceq c_3 \ominus_{g\circ f} lhs(r)$. We use a similar reasoning (see proof of Lemma A2(2)) to define the embedding h satisfying $c_1 \ominus_f lhs(r) \preceq_h c_3 \ominus_{g\circ f} lhs(r)$. Observe that we have $h \circ \widehat{f} = \widehat{g \circ f}$ by the following argument: By definition, the mappings $h \circ \widehat{f}$ and $\widehat{g \circ f}$ are both inclusions of $lhs(r)$ in $c_3 \ominus_{g\circ f} lhs(r)$ rooted at the same node $g(f(\varepsilon))$. Now by Lemma A2(1) and the fact that $c_1 \ominus_f lhs(r) \preceq_h c_3 \ominus_{g\circ f} lhs(r)$, we obtain

$$\begin{aligned} c_2 = (c_1 \ominus_f lhs(r)) \odot_{\widehat{f}} rhs(r) &\preceq (c_3 \ominus_{g\circ f} lhs(r)) \odot_{h\circ\widehat{f}} rhs(r) \\ &= (c_3 \ominus_{g\circ f} lhs(r)) \odot_{\widehat{g\circ f}} rhs(r) = c_4. \end{aligned}$$

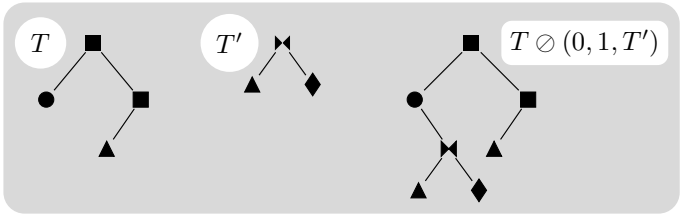
□

A.2 Algorithm

We devote this section to the proof of Lemma 2. In order to do that we first introduce some preliminaries on trees. Afterwards, we define the *tree addition* operation which will be needed to capture the effect of the approximation in the computation of Pre. Finally, we prove the lemma.

Preliminaries In this paragraph, we fix two trees $T = (N, \lambda)$ and $T' = (N', \lambda')$.

Given a $n \in N$ which is a leaf in T and a $b \in \{0, 1\}$, we use $T \oslash (n, b, T')$ to denote the tree $T'' = (N'', \lambda'')$ obtaining from T by appending T' as a subtree of n "in the b direction". More formally, we let $N'' := N \bigcup n \bullet b \bullet N'$; and for any $n'' \in N''$ we define $\lambda''(n'') := \lambda(n'')$ if $n \bullet b \not\leq n''$, and $\lambda''(n'') := \lambda'(n \bullet b - n'')$ otherwise.



A non-empty set of nodes $N'' \subseteq N$ is said to be *almost prefix closed* if it is the image of an inclusion in T. More precisely, there exists a tree T'' such that $T'' \subseteq_g T$ for an inclusion g satisfying $\mathrm{Img}(g) = N''$ (see Figure 7). Observe that by definition, any almost prefix closed set N'' contains a unique node n such that $g(n) = \varepsilon$. In the sequel, we use $root(N'')$ to refer to such a node, and we denote by $leaves(N'') \subseteq N''$ the set of nodes without children or whose all children in T do not belong to N''.

We introduce now a special class of tree relations. Given a set $N'' \subseteq N$ and a function $f : N'' \to N'$. We say that f is a *partial embedding* of T in T' with respect to N'' if the following conditions hold: (i) N'' is an almost prefix closed set of nodes, and (ii) f satisfies the embedding conditions (see Section 2) (see Figure 7). More precisely, for any node $n'' \in N''$

- $n'' \bullet b \in N'' \implies f(n'') \bullet b \leq f(n'' \bullet b)$, and
- $\lambda(n'') = \lambda'(f(n''))$.

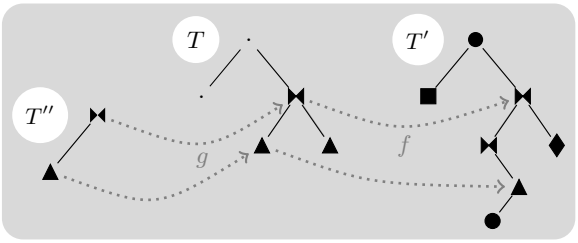


Fig. 7. In the tree T, we give an example of an almost prefix closed set defined by the incluion g of the tree T'' in T. The trees T and T' are related by the partial embedding f defined on the almost prefix closed set considered earlier. In the rest of the figures, we will use T and T' as the starting tree examples.

Intuitively, the existence of a partial embedding f of T in T' implies the following: Starting from T, we can construct a tree by adding to T the nodes

from N' which are not in $\mathrm{Img}(f)$, such that (i) we do not modify the "structure" of the remaining nodes in T, and (ii) T' is included in the resulting tree. We formalize this construction by the *addition* operation $T \oplus_f T'$ described below.

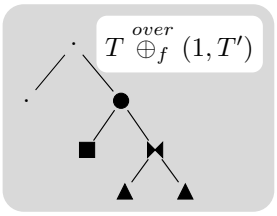
Tree Addition In the following, we fix two trees $T = (N, \lambda), T' = (N', \lambda') \in \mathit{Trees}(\Sigma)$ and a partial embedding f of T in T' defined over some almost prefix closed set $N'' \subseteq N$.

We define several auxiliary operations in order to describe the addition of subtrees from T' in different positions (above, below and in-between) with respect to some reference node(s).

Given a node $n \in N''$, we first consider the addition in T of certain nodes from T' above n.

Formally, we denote by $T \overset{over}{\oplus_f} (n, T')$ the tree T'' of the following form:

$$T'' = T \otimes (n, T_a) \text{ where } T_a = T' \otimes (f(n), T(n))$$



Intuitively, we first construct T_a by concatenating T' with the subtree $T(n)$ at $f(n)$. Then, we derive T'' by replacing in T the subtree $T(n)$ by T_a.

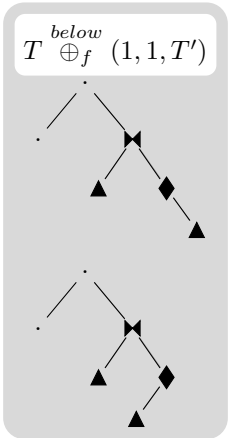
We consider now, the addition in T of some nodes from T' below a reference node of T. In such a case, we also need to take into consideration the position (left or right subtree) in which we add the nodes.

More precisely, given a node $n \in N''$ and some $b \in \{0, 1\}$, we denote by $T \overset{below}{\oplus_f} (n, b, T')$ the set of trees $S \subseteq \mathit{Trees}(\Sigma)$ constructed as described below: In case the node $f(n) \bullet b$ is not defined (i.e., $\notin N'$), then we let $S := \{T\}$. Otherwise, we look at the tree T and consider two subcases depending on whether $n \bullet b$ is defined in T or not.

- If $n \bullet b \in N$, we let $T_a := T \otimes (n \bullet b, T'(f(n) \bullet b))$, we denote by N_l the set of leaves in T_a which are descendants of $n \bullet b$, and we define

$$S := \{T_a \oslash (n_l, b_l, T(n \bullet b)) \mid n_l \in N_l \wedge b_l \in \{0, 1\}\}.$$

- Otherwise, we let $S := \{T \oslash (n, b, T'(f(n) \bullet b))\}$.



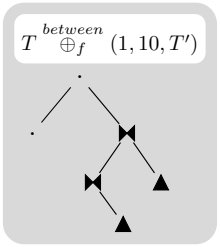
We generalize the above operation by

$$T \overset{below}{\oplus_f} (n, T') := \bigcup_{T'' \in T \overset{below}{\oplus_f} (n,0,T')} T'' \overset{below}{\oplus_f} (n, 1, T').$$

Finally, we consider the addition of nodes between two given reference nodes in T. In such a case, we require that the given nodes are directly related to each others (parent/child).

For two nodes $n, n' \in N''$ such that $n \bullet b = n'$ for some $b \in \{0,1\}$, we denote by $T \overset{between}{\oplus_f} (n, n', T')$ the tree T'' of the following form:

$$T'' = T \otimes (n, T_a(f(n) \bullet b)) \text{ where } T_a = T' \otimes (f(n'), T(n'))$$



Intuitively, we first construct T_a by replacing the subtree $T'(f(n'))$ by $T(n')$, then we derive T'' by replacing the subtree $T(n)$ by $T_a(f(n))$.

We are now ready to generalize the above operations by defining $T \oplus_f T'$. We first consider an enumeration of the nodes N'' in a bottom-up fashion. In other words, let $\{n_i\}_{1 \leq i \leq |N''|}$ be an arrangement of the nodes of N'' such that for any $i, j : 1 \leq i \neq j \leq |N''|$, $n_i < n_j$ implies that $j < i$. Then, we define a sequence of sets of trees $\{S_i\}_{0 \leq i \leq |N''|}$ as follows. We let $S_0 := \{T\}$. For any $i : 0 \leq i \leq |N''| - 1$, we define S_{i+1} in terms of S_i by three case analysis reflecting the nature of the node n_{i+1}.

- If $n_{i+1} \in \mathit{leaves}(N'')$, then we let

$$S_{i+1} := \bigcup_{T'' \in S_i} T'' \overset{below}{\oplus_f} (n_{i+1}, T').$$

- If there is $b \in \{0,1\}$ such that $n_{i+1} \bullet b \in N''$ and $n_{i+1} \bullet (1-b) \notin N''$, then we define

$$S_{i+1} := \bigcup_{T'' \in S_a} T'' \overset{below}{\oplus_f} (n_{i+1}, 1-b, T') \text{ where}$$
$$S_a := \bigcup_{T'' \in S_i} \left\{ T'' \overset{between}{\oplus_f} (n_{i+1}, n_{i+1} \bullet b, T') \right\}.$$

  Observe that this is the case where n_{i+1} has only one child in N''.
- Otherwise, we define

$$S_{i+1} := \bigcup_{T'' \in S_a} \left\{ T'' \overset{between}{\oplus_f} (n_{i+1}, n_{i+1} \bullet 1, T') \right\} \text{ where}$$
$$S_a := \bigcup_{T'' \in S_i} \left\{ T'' \overset{between}{\oplus_f} (n_{i+1}, n_{i+1} \bullet 0, T') \right\}.$$

Finally, we let

$$T \oplus_f T' := \bigcup_{T'' \in S_{|N''|}} \left\{ T'' \overset{over}{\oplus_f} (n_{|N''|}, T') \right\}.$$

Observe that each tree $T'' \in T \oplus_f T'$ satisfies (i) $T \preceq T''$, and (ii) there is a unique inclusion denoted by $\overline{f}$ of T' in T'' such that $\overline{f}(\varepsilon) = \mathit{root}(N'')$ (see Figure 8).

We are now ready to describe Pre.

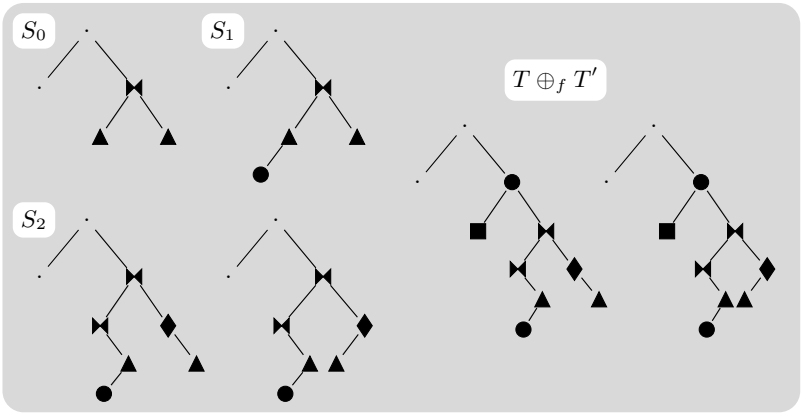


Fig. 8. The figures shows the sequence of sets of trees in the computation of $T \oplus_f T'$.

Computing Pre Assume that we are given a parameterized tree system $\mathcal{P} = (Q, R)$. For a constraint ϕ', we define $\mathrm{Pre}_R(\phi') = \bigcup_{r \in R} \mathrm{Pre}_r(\phi')$, i.e. we compute the set of predecessor constraints with respect to each rewrite rule $r \in R$.

Consider a constraint ϕ' and a rule $r \in R$. We define $\mathrm{Pre}_r(\phi')$ to be the set of constraints satisfying the following: A constraint ϕ belongs to $\mathrm{Pre}_r(\phi')$ iff there is a partial embedding f of ϕ' in $rhs(r)$ such that

$$\phi \in (\phi' \oplus_f rhs(r)) \odot_{\overline{f}} lhs(r).$$

Proof of Lemma 2 In order to prove Lemma 2, we rely on the following auxiliary result related to the addition operation.

Lemma A3. *For any $T_0, T_1, T_2 \in Trees(\Sigma)$ and any mappings f and g satisfying $T_0 \subseteq_f T_2$ and $T_1 \preceq_g T_2$, the following holds: If $Img(f) \cap Img(g) \neq \emptyset$, then there is a partial embedding h of T_1 in T_0 such that for any $T \in T_1 \oplus_h T_0$ we have $T \preceq T_2$.*

Proof. Assume that T_0, T_1, T_2 are of the forms $(N_0, \lambda_0), (N_1, \lambda_1), (N_2, \lambda_2)$ respectively. We first define the following set of nodes $N'' := \{n \in N_1 |\ g(n) \in \mathrm{Img}(f)\}$. Observe that $\mathrm{Img}(f) \cap \mathrm{Img}(g) \neq \emptyset$ iff $N'' \neq \emptyset$. We now show that N'' is almost prefix closed.

In order to do that, we assume the opposite and derive a contradiction. We suppose that N'' is not almost prefix closed. By definition, this implies the existence of two nodes $n, n' \in N''$ such one the following conditions is satisfied.

1. $n \leq n_c \leq n'$ for some $n_c \notin N''$.
2. $n \not\leq n' \wedge n' \not\leq n$ and there is no $n_p \in N''$ such that $n_p \leq n \wedge n_p \leq n'$.

If condition (1) holds then this implies that $g(n_c) \notin \mathrm{Img}(f)$. Since $g(n), g(n') \in \mathrm{Img}(f)$ and $g(n) \leq g(n')$, it holds by definition of an inclusion that any $n_2 \in N_2 : g(n) \leq n_2 \leq g(n')$ satisfies $n_2 \in \mathrm{Img}(f)$. This is a contradiction since by

definition of an embedding $g(n) \leq g(n_c) \leq g(n')$ and by assumption $g(n_c) \notin \mathrm{Img}(f)$.

If condition (2) holds, then we let $n_p \in N_0$ be the node satisfying $n_p \bullet b \leq n$ and $n_p \bullet (1-b) \leq n'$ for some $b \in \{0,1\}$. Observe that such a node exists since N_0 is prefix closed. We recall here that by assumption $n_p \notin N$. By definition of an embedding $g(n_p) \bullet b \leq g(n)$ and $g(n_p) \bullet (1-b) \leq g(n')$. Since $g(n), g(n') \in \mathrm{Img}(f)$, it follows by definition of an inclusion that $g(n_p) \in \mathrm{Img}(f)$. This is also a contradiction.

So far, we have shown that N'' is almost prefix closed. We let $h := f^{-1} \circ g$ denote the function defined over N''. Observe that by definition of g and f, and since N'' is almost prefix closed, it follows that h is a partial embedding of T_1 in T_0.

Consider now a tree $T = (N, \lambda) \in T_1 \oplus_h T_0$. We first recall that $T_0 \subseteq_{\overline{h}} T$. Then, we provide below a mapping e that guarantees $T \preceq_e T_2$. We define e by case analysis depending on the membership of $n \in N$.

- If $n \in \mathrm{Img}(\overline{h})$, then we let $e(n) := f(\overline{h}^{-1}(n))$.
- If $\overline{h}(\varepsilon) \not\leq n$, we define $e(n) := g(n)$.
- Otherwise, we denote by n_h the longest prefix in $\mathrm{Img}(\overline{h})$ such that and $n_h \bullet b \leq n$ for some $b \in \{0,1\}$. Two cases follow depending on the membership of $f(\overline{h}^{-1}(n_h))$ in $\mathrm{Img}(g)$.
  - If $f(\overline{h}^{-1}(n_h)) \in \mathrm{Img}(g)$, then we define $e(n) := g(g^{-1}(f(\overline{h}^{-1}(n_h))) \bullet (n - n_h))$.
  - Otherwise, we denote by n_g, the longest word in $\mathrm{Img}(g) \cap \mathrm{Img}(\overline{h})$ such that $n_g \bullet b' \leq n_h$ for some $b' \in \{0,1\}$. We let $e(n) := g(g^{-1}(f(\overline{h}^{-1}(n_g))) \bullet b' \bullet (n - n_h \bullet b))$.

□

In what follows, we consider a weaker statement of Lemma 2 which is more convenient for our definition of Pre. Nevertheless, the original statement can be proven similarly but in a more tedious manner. We have considered this variant hoping that the presentation would be more clear. Observe, that the result below suffices for correctness of our algorithm.

Weaker Variant of Lemma 2. *For any constraint ϕ:*

- *$Pre_R(\phi)$ is computable and finite.*
- *$[\![Pre(\phi)]\!] \subseteq [\![\phi]\!] \cup [\![Pre_R(\phi)]\!]$.*

Proof. Observe that Finiteness and computability of $\mathrm{Pre}_R(\phi)$ follows by definition of the $\oplus$ operation.

In order to show correctness (the second item), we consider a constraint ϕ' and a configuration $c' \in [\![\phi']\!]$ and show that for any constraint c with $c \rightsquigarrow c'$ it is the case that $c \in [\![\phi']\!] \cup [\![\mathrm{Pre}_R(\phi')]\!]$.

By definition of the approximate transition relation, $c \rightsquigarrow c'$ implies that there is a rule $r \in R$ and an embedding f of $lhs(r)$ in c such that

$$(c \ominus_f lhs(r)) \odot_{\widehat{f}} rhs(r) = c'. \tag{1}$$

By assumption $c' \in [\![\phi']\!]$, therefore there is a function g such that $\phi' \preceq_g c'$. By Equation (1) and the definition of $\widehat{f}$, we have $rhs(r) \subseteq_{\widehat{f}} c'$. Below, we consider two cases depending on emptiness of $\mathrm{Img}(\widehat{f}) \bigcap \mathrm{Img}(g)$.

In case $\mathrm{Img}(\widehat{f}) \bigcap \mathrm{Img}(g) = \emptyset$, it follows directly that $\phi' \preceq_g (c \ominus_f lhs(r)) \preceq c$ where the last embedding relation holds by Lemma A1(1). As a consequence, $c \in [\![\phi']\!]$.

In case $\mathrm{Img}(\widehat{f}) \cap \mathrm{Img}(g) \neq \emptyset$, then this combined with the fact that $\phi' \preceq_g c'$ and Lemma A3 yields the following: There is a partial embedding h of ϕ' in $rhs(r)$ such that for any $\phi" \in \phi' \oplus_h rhs(r)$, it is the case that $rhs(r) \subseteq_{\overline{h}} \phi" \preceq c'$.

We fix a constraint $\phi"$ in $\phi' \oplus_h rhs(r)$ and we recall from the proof of Lemma A3 the definition of the embedding e satisfying $\phi" \preceq_e c'$. Observe that by definition of e, $\widehat{f}$ and $\overline{h}$, we have $\widehat{f} = e \circ \overline{h}$. This combined with Lemma A2(1) and the fact that $\phi" \preceq_e c'$ yields the following:

$$\phi" \odot_{\overline{h}} lhs(r) \preceq c' \odot_{e \circ \overline{h}} lhs(r) = c' \odot_{\widehat{f}} lhs(r) = c \ominus_f lhs(r),$$

where the last equality follows from the definition of c' and the renaming operation.

By Lemma A1(1), $c \ominus_f lhs(r) \preceq c$ and hence by the above equations $\phi" \odot_{\overline{h}} lhs(r) \preceq c$. Finally, since by definition of $\mathrm{Pre}_r(\cdot)$ we have

$$\phi" \odot_{\overline{h}} lhs(r) \in (\phi' \oplus_h rhs(r)) \odot_{\overline{h}} lhs(r) \subseteq \mathrm{Pre}_r(\phi') \subseteq \mathrm{Pre}_R(\phi'),$$

it follows that $c \in [\![\mathrm{Pre}_R(\phi')]\!]$. □

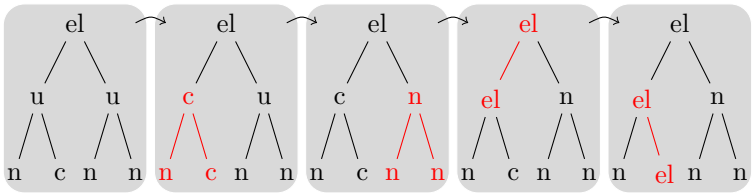
B Appendix – Description of the Case Studies

In the following, we give detailed description of the case studies. For each example, we describe how we instantiate the algorithm by specifying (i) the initial configuration, (ii) the set of rewriting rules , and (iii) the set of bad constraints.

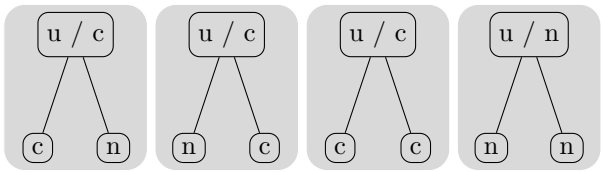
B.1 Leader Election Protocol

The protocol operates on binary trees to elect a leader among processes which reside in the leaves and which are candidates. A leaf process can be labeled as candidate c or non-candidate n. An inner node is initially labeled as undefined u and will be labeled as candidate if at least one of its children is candidate, and non-candidate otherwise.

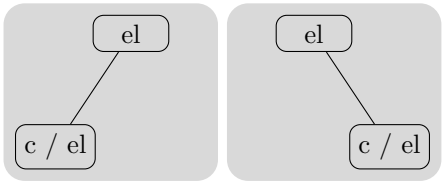
In a first phase, the information that a node is candidate or not travels up to reach the root. In a second phase, the decision el, initially in the root, travels down from candidate parent to candidate child. (If several children are candidates, the parent chooses one undeterministicly). Once the decision reaches a leaf, this leaf process is elected as leader. Below follows a small scenario:



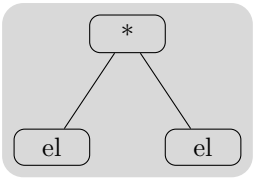
The rules for the upwards propagation of candidate information are:



The rules for the downwards propagation of election decision are:



The set of bad constraints F is represented by trees where at least two nodes in different branches are elected.

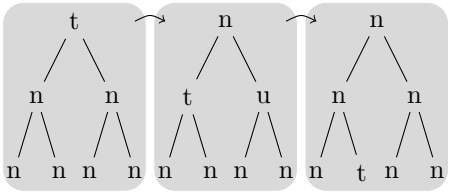


The set of initial constraints *Init* is represented by trees where leaves are either candidates or non-candidates, inner nodes are labeled undefined and the root is labeled *el*.

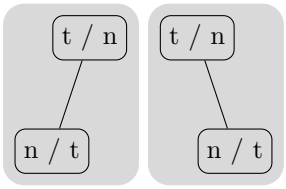
We have been able to prove *Init* is not backward reachable from F (i.e. that $Init \xrightarrow{*} F$ does not hold).

B.2 Token Protocol

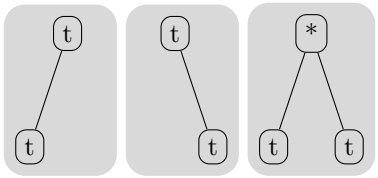
The protocol operates on binary trees to transmit a token from the root to the leaves. A node can be labeled as having the token t, or not having the token n. As an example:



The rules for the propagation of the token are:



The set of bad constraints F is represented by trees where at least two nodes contain the token.

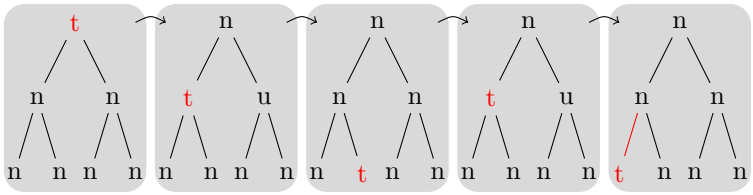


Initially, the token is in the root.

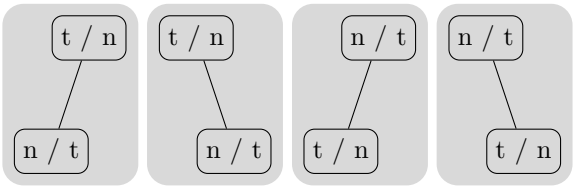
We have been able to prove *Init* is not backward reachable from F (i.e. that $Init \xrightarrow{*} F$ does not hold).

B.3 Two-way Token Protocol

This protocol is a generalization of the token protocol by allowing the token to both move upwards and downwards. As an example:



The rules for the propagation of the token are:



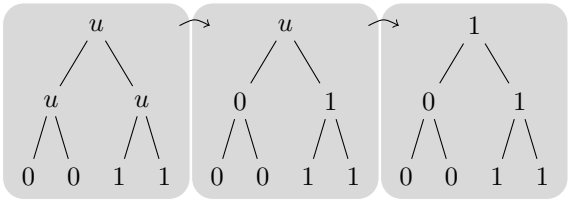
The set of bad constraints F is represented by trees where at least two nodes contain the token, similarly to the simple token protocol.

Initially, the token is in the root.

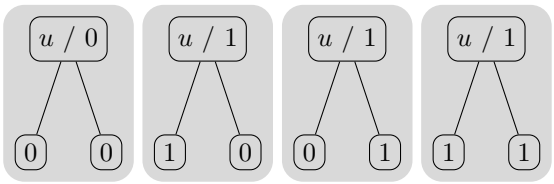
We have been able to prove *Init* is not backward reachable from F (i.e. that $\mathit{Init} \xrightarrow{*} F$ does not hold).

B.4 Percolate Protocol

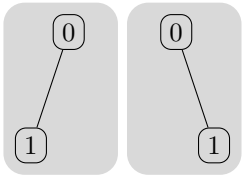
The protocol [17] operates on binary trees of processes and evaluates the disjunction of the values in the leaves up to the root. Initially, the leaves contain either a 0 or a 1 and all other nodes are labeled as undefined u. An still undefined inner node will be labeled as 1 if at least one of its children contains a 1, and 0 otherwise. As an example:



The corresponding rules are:



The set of bad constraints F is represented by trees where a 0 get propagated upwards while there is a 1 below.



We have been able to prove *Init* is not backward reachable from F (i.e. that $\mathit{Init} \xrightarrow{*} F$ does not hold).