Decidable First-Order Transition Logics for PA-processes

D. Lugiez[1] and Ph. Schnoebelen[2]

[1] Lab. d'Informatique de Marseille, Univ. Aix-Marseille 1 & CNRS URA 1787,
39, r. Joliot-Curie, 13453 Marseille Cedex 13, France
email: `lugiez@lim.univ-mrs.fr`

[2] Lab. Spécification et Vérification, ENS de Cachan & CNRS UMR 8643,
61, av. Pdt Wilson, 94235 Cachan, France
email: `phs@lsv.ens-cachan.fr`

Abstract. We show decidability of several first-order logics based upon the reachability predicate in PA. The main tool we use is the recognizability by tree automata of the reachability relation between PA-processes. This approach and the transition logics we use allow a smooth and general treatment of parameterized model checking for PA. Then the logic is extended to handle a general notion of costs of PA-steps. In particular, when costs are Parikh images of traces, we show decidability of a transition logic extended by some form of first-order reasoning over costs.

1 Introduction

PA [BW90] is the formal model of concurrency allowing recursive definitions, sequential and parallel compositions, but where actions are uninterpreted and do not synchronize. PA gives rise to infinite-state systems and is quite expressive. Recently, several verification problems have been shown decidable for PA using a variety of fairly involved techniques [1]. Decidability results for PA are mainly theoretical, and they help delineate some important frontiers in the field of verification for infinite-state processes. In [LS99], we advocated *regular tree languages* and *tree automata* as an easy-to-use tool for tackling problems about PA, and we proved that the reachability sets of a regular set of PA processes is regular. Our proofs are effective and give simple polynomial-time algorithms which can be used for a variety of problems based on reachability among PA-processes. [EK99,EP00] showed how to use and adapt them for problems in static analysis.

Contents of the paper. Here we extend our previous work in several ways:

Recognizable tree relations: We replace automata for tree languages by automata for tree relations and show that $\xrightarrow{*}$ over PA-processes is recognizable.

First-order transition logic: This gives a decision method for the first-order transition logic, i.e. the first-order logic having $\to$, $\xrightarrow{*}$, and equality as basic

[1] See, e.g., [BEH95b,BEH95a,Kuč96,Kuč97,JKM98,HJ99,May99].

predicates (plus any other recognizable predicates). The method computes the set of solutions of a given formula, and thus allows parameterized model checking, model measuring, ...

Costs: We enrich PA with a notion of "cost of steps" which is more general than traces. These costs can encode various measures and view PA as a truly concurrent model (e.g., costs can encode timing measures where parallelism is faster than interleaving). We extend the transition logic with decomposable cost predicates and show the decidability of several timed transition logics.

Parameterized constraints over $\xrightarrow{*}$: Finally, we define TLC, the transition logic where costs are the Parikh images of traces and where integer variables and Presburger formulas are freely used to state constraints on reachability. TLC is not decidable but we isolate a rich fragment which is.

Related work. Several temporal logics with cost constraints have been proposed for finite state systems (see [AELP99,ET99] for recent proposals). Some fragments of the temporal logics from [BEH95b,BEH95a] apply to PA but temporal logics deal with paths and are quite different from transition logics where a first-order theory of *states* is available (more explanations in section 7.1).

For costs that are Parikh images of traces, [Esp97] shows recognizability of the ternary relation $s \xrightarrow{c} t$ over BPP (PA without sequential composition) but does not consider applications to the first-order transition logic. [CJ99] shows recognizability of the reachability relation between configurations of timed automata, introduces the transition logic and uses it for model measuring and parameterized model checking. An important technical difference is that our automata recognize pairs of trees (PA-processes) while [Esp97] handles tuples of integers (markings of BPP's) and [CJ99] handles tuples of reals (clock values).

Reachability in PA is investigated in [May97,May99]. The underlying methods apply to more general systems (like PRS [May00]) but they are quite complex since they view terms modulo structural congruence. As explained in [LS99], we believe it is better to only introduce structural congruence at a later stage.

The combination of costs and tree automata is studied by Seidl [Sei94] for compiler optimization (using more general costs than ours), not decidability of logics on trees (where the relevant problem is the combination of cost automata).

2 The PA process algebra

PA may be defined in several (equivalent) ways. Our definition:

(1) uses rewrite rules *à la* Moller [Mol96],
(2) does not identify terms modulo structural congruence,
(3) incorporates a notion of costs for steps,
(4) is a *big-steps* semantics (in the sense of [Plo81]).

Syntax. We assume $Act = \{a, b, \dots\}$ is a finite set of *action names* and write $Act^* = \{w, \dots\}$ for the set of words over Act (with empty string denoted ε).

For $w_1, w_2 \in Act^*$, we let $w_1 \parallel w_2$ denote their shuffle product, i.e. the set of all their interleavings.

We assume $M = \{c, \dots\}$ is a set of values, called *costs*, equipped with two binary associative-commutative operations $\oplus$ and $\otimes$ with same neutral element 0_M (we see several different examples of cost sets in sections 6, 7). Given a set $Const = \{X, Y, Z, \dots\}$ of *process constants* (or names), $\mathcal{T}_{Const}$, or $\mathcal{T}$ when the underlying $Const$ is clear, is the set $\{s, t, \dots\}$ of *PA-terms*, given by the following abstract syntax

$$s, t ::= s.t \;\mid\; s \parallel t \;\mid\; O \;\mid\; X \;\mid\; Y \;\mid\; Z \;\mid\; \cdots$$

For $t \in \mathcal{T}$, $Const(t)$ denotes the set of process constants occurring in t. A PA *declaration* is a finite $Const$ with a finite set $\Delta \subseteq Const \times Act \times M \times \mathcal{T}$ of *process rewrite rules.* A rule $(X, a, c, t) \in \Delta$ is written $X \xrightarrow{a,c} t$. For simplicity, we require that all $X \in Const$ appear in the left-hand side of at least one rule.

For $t \in \mathcal{T}$, we let $Sub(t)$ denote the set of all subterms of t. Similarly, we write $Sub(\Delta)$ for the finite set of all subterms of (some term from) Δ. The size of a term is $|t| \stackrel{\text{def}}{=} Card(Sub(t))$ and the size of a PA declaration is $|\Delta| \stackrel{\text{def}}{=} Card(Sub(\Delta))$.

Semantics. A PA declaration Δ defines a labeled transition system $(\mathcal{T}, \rightarrow)$ with $\rightarrow \subseteq \mathcal{T} \times Act^* \times M \times \mathcal{T}$. We write $s \xrightarrow{w,c} t$ when $(s, w, c, t) \in \rightarrow$. The transition relation $\xrightarrow{w,c}$ is defined by the following SOS rules:

$$(\mathrm{R}_\varepsilon)\quad \frac{}{0 \xrightarrow{\varepsilon,0_M} 0} \qquad\qquad (\mathrm{R_C})\quad \frac{t \xrightarrow{w,c'} t'}{X \xrightarrow{aw,c\oplus c'} t'} \text{ if } X \xrightarrow{a,c} t \in \Delta$$

$$(\mathrm{R'}_\varepsilon)\quad \frac{}{X \xrightarrow{\varepsilon,0_M} X} \qquad\qquad (\mathrm{R_P})\quad \frac{t_1 \xrightarrow{w_1,c_1} t_1' \quad t_2 \xrightarrow{w_2,c_2} t_2'}{t_1 \parallel t_2 \xrightarrow{w,c_1\otimes c_2} t_1' \parallel t_2'} \text{ if } w \in w_1 \parallel w_2$$

$$(\mathrm{R_S})\quad \frac{t_1 \xrightarrow{w_1,c_1} t_1'}{t_1.t_2 \xrightarrow{w_1,c_1} t_1'.t_2} \qquad\qquad (\mathrm{R'_S})\quad \frac{t_1 \xrightarrow{w_1,c_1} t_1' \quad t_2 \xrightarrow{w_2,c_2} t_2'}{t_1.t_2 \xrightarrow{w_1w_2,c_1\oplus c_2} t_1'.t_2'} \text{ if } Const(t_1') = \varnothing$$

The intuition formalized by $s \xrightarrow{w,c} t$ is that s can evolve into t by performing the sequence of actions w and the cost of that derivation is c. In general there may exist several different derivations between a s and a t: they may have same cost or not, and use same sequence of actions or not. For instance, if $\Delta = \{X \xrightarrow{a,c} Y, X \xrightarrow{b,c'} Y\}$, then the cost of reaching Y from X is either c or c'.

We write $s \xrightarrow{w} t$ (or $s \xrightarrow{c} t$, or $s \xrightarrow{*} t$) when $s \xrightarrow{w,c} t$ for some c (resp. for some w, for some w, c). Even though we started with a big-steps semantics, it is convenient to have small-steps too and we write $s \rightarrow t$ when $s \xrightarrow{w} t$ for some w of length 1. We also use $\xrightarrow{+}$ defined as $\rightarrow \circ \xrightarrow{*}$.

For $t \in \mathcal{T}$, the set $Post^*(t) \stackrel{\text{def}}{=} \{t' \mid t \xrightarrow{*} t'\}$ and $Pre^*(t) \stackrel{\text{def}}{=} \{t' \mid t' \xrightarrow{*} t\}$ denote the set of *iterated successors* and (resp.) *iterated predecessors* of t. $Post^*(t)$ is also called the reachability set of t.

3 Tree Automata and Regular Cost Grammars

Given a finite ranked alphabet $\mathcal{F} = \mathcal{F}_0 \cup \mathcal{F}_1 \cup \cdots \cup \mathcal{F}_m$, $T_{\mathcal{F}}$ denotes the set of terms (or finite trees) built from $\mathcal{F}$. A *tree language* is any subset L of $T_{\mathcal{F}}$.

3.1 Tree Automata and Regular Tree Grammars

Tree automata recognize sets of trees (see [CDG$^+$99]). Formally, a *tree automaton* is a tuple $\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \delta \rangle$ where $\mathcal{F}$ is a finite ranked alphabet, $\mathcal{Q} = \{q_1, \ldots, \}$ is a finite set of states, and $\delta \subseteq \cup_{n\in\mathbb{N}}(\mathcal{F}_n \times \mathcal{Q}^n \times \mathcal{Q})$ is a finite set of *transition rules*. A rule $\langle f, q_1, \ldots, q_n, q\rangle \in \delta$ is usually written $f(q_1, \ldots, q_n) \longmapsto q$ and is read as "*if the subterms* $t_1, \ldots, t_n$ *of some* $t = f(t_1, \ldots, t_n)$ *have been labeled by* $q_1, \ldots, q_n$, *then* $\mathcal{A}$ *may label* t *with* q". Given a term t, the automaton labels the nodes of t by states according to the rules in a bottom-up way. We write $t \overset{*}{\longmapsto} q$ when $t \in T_{\mathcal{F}}$ may be rewritten into $q \in \mathcal{Q}$ using the rules of δ.

Recognizability. When $t \overset{*}{\longmapsto} q$, we say t is *recognized* (also *accepted*) by state q. We write $L(q)$ for $\{t \mid t \overset{*}{\longmapsto} q\}$ and say it is the tree language recognized by q. If we add a set $F \subseteq \mathcal{Q}$ of *final states* to some tree automaton $\mathcal{A}$, then the language recognized by $\mathcal{A}$ is $L(\mathcal{A}) \overset{\text{def}}{=} \cup_{q\in F} L(q) = \{t \mid \exists q \in F, t \overset{*}{\longmapsto} q\}$. We say that $L \subseteq T_{\mathcal{F}}$ is *recognizable* if $L = L(\mathcal{A})$ for some tree automaton. Recognizable tree languages are closed under union, intersection, complementation.

ε-rules. Tree automata with ε-rules further allow rules of the form $q \longmapsto q'$. These rules may be used anywhere inside $t \overset{*}{\longmapsto} q''$. With the classical subset construction, tree automata with ε-rules can be transformed into equivalent deterministic tree automata without ε-rules.

Top-down tree automata are structurally bottom-up tree automata but label trees in a top-down way. The difference is that a rule is now written $q \to f(q_1, \ldots, q_n)$ (or $q' \to q$ for an ε-rule) and that the final states are now called *initial states*. The most important difference between top-down and bottom-up automata is a difference in viewpoint. Top-down tree automata are often seen as regular tree grammars, i.e. as generators rather than acceptors.

3.2 Recognizability of $\boldsymbol{Post^*(X)}$

We now show that, for any PA-declaration Δ, and any $X \in \mathit{Const}$, the set $\mathit{Post}^*(X)$ is a regular tree language and give at the same time a grammar for the costs of the derivations. This generalizes the result of [LS99] to costs and uses a slightly different approach which yields the proof that the relation $\xrightarrow{*}$ is recognizable.

Given a cost set $\langle M, \oplus, \otimes\rangle$, a *regular cost grammar* over M is a set $\mathcal{C}$ of non-terminals $C_1, \ldots, C_n$ together with cost rules $C_{n_i} \to E_{n_i}$ where the E's are simple right-hand sides of the form "$C \oplus C'$" or "$C \otimes C'$" (where C, C' are non-terminals), or "c" (where $c \in M$). Cost grammars describe subsets of M: the rules define a set of recursive inclusions that admit a least fixpoint solution. For simplicity we write C for the subset of M defined by the non-terminal C and use shortcuts like $C \to C'$ (ε-rules) or $C \to c \otimes C'$ for grammar rules.

A regular tree grammar and a regular cost grammar for $Post^*(X)$

$$\left.\begin{array}{rl} I_0 & \to 0 \\ Q_0 & \to 0 \\ Q'_0 & \to 0 \end{array} \quad \begin{array}{rl} C_0^I & \to 0_M \\ C_0^Q & \to 0_M \\ C_0^{Q'} & \to 0_M \end{array}\right\}\text{if } 0 \in \mathit{Sub}(\Delta)$$

$$\left.\begin{array}{rl} I_Y & \to Y \\ Q_Y & \to Y \end{array} \quad \begin{array}{rl} C_Y^I & \to 0_M \\ C_Y^Q & \to 0_M \end{array}\right\}\text{for all } Y \in \mathit{Sub}(\Delta)$$

$$\left.\begin{array}{rl} Q_Y & \to Q_s \\ Q'_Y & \to Q'_s \end{array} \quad \begin{array}{rl} C_Y^Q & \to c \oplus C_s^Q \\ C_Y^{Q'} & \to c \oplus C_s^{Q'} \end{array}\right\}\text{for all } Y \xrightarrow{a,c} s \in \Delta$$

$$\left.\begin{array}{rl} I_{s_1\|s_2} & \to I_{s_1} \parallel I_{s_2} \\ Q_{s_1\|s_2} & \to Q_{s_1} \parallel Q_{s_2} \\ Q'_{s_1\|s_2} & \to Q'_{s_1} \parallel Q'_{s_2} \end{array} \quad \begin{array}{rl} C^I_{s_1\|s_2} & \to C^I_{s_1} \otimes C^I_{s_2} \\ C^Q_{s_1\|s_2} & \to C^Q_{s_1} \otimes C^Q_{s_2} \\ C^{Q'}_{s_1\|s_2} & \to C^{Q'}_{s_1} \otimes C^{Q'}_{s_2} \end{array}\right\}\text{for all } s_1 \parallel s_2 \in \mathit{Sub}(\Delta)$$

$$\left.\begin{array}{rl} I_{s_1.s_2} & \to I_{s_1}.I_{s_2} \\ Q_{s_1.s_2} & \to Q_{s_1}.I_{s_2} \\ & \mid\ Q'_{s_1}.Q_{s_2} \\ Q'_{s_1.s_2} & \to Q'_{s_1}.Q'_{s_2} \end{array} \quad \begin{array}{rl} C^I_{s_1.s_2} & \to C^I_{s_1} \oplus C^I_{s_2} \\ C^Q_{s_1.s_2} & \to C^Q_{s_1} \oplus C^I_{s_2} \\ & \mid\ C^{Q'}_{s_1} \oplus C^Q_{s_2} \\ C^{Q'}_{s_1.s_2} & \to C^{Q'}_{s_1} \oplus C^{Q'}_{s_2} \end{array}\right\}\text{for all } s_1.s_2 \in \mathit{Sub}(\Delta)$$

These rules denote both a tree automaton $\mathcal{A}_{Post^*}$, and a cost grammar $\mathcal{C}_{Post^*}$. Their relationship is stated in the following proposition:

Proposition 3.1. *For all $t \in \mathit{Sub}(\Delta)$:*

1. *$I_t \xrightarrow{*} s$ iff $s = t$ and $t \xrightarrow{0_M} s$. Furthermore, $C_t^I = \{0_M\}$.*
2. *$Q_t \xrightarrow{*} s$ iff $t \xrightarrow{*} s$. Furthermore, if $t \xrightarrow{c} s$ then $c \in C_t^Q$, and if $c \in C_t^Q$ then $t \xrightarrow{c} s'$ for some s' such that $Q_t \xrightarrow{*} s'$.*
3. *$Q'_t \xrightarrow{*} s$ iff $t \xrightarrow{*} s$ and s is terminated. Furthermore, if $t \xrightarrow{c} s$ then $c \in C_t^{Q'}$, and if $c \in C_t^{Q'}$ then $t \xrightarrow{c} s'$ for some terminated s' such that $Q'_t \xrightarrow{*} s'$.*

4 Tree automata and n-ary relations

Products of trees. We follow [DT90]. Given two terms $s, t \in T_{\mathcal{F}}$, the pair (s,t) can be seen as one term over a product alphabet $\mathcal{F}_\times \stackrel{\text{def}}{=} (\mathcal{F} \cup \{\bot\}) \times (\mathcal{F} \cup \{\bot\}) - \{\bot\bot\}$ where $\bot$ is a new symbol with arity 0. In $\mathcal{F}_\times$ the arity of fg is the maximum of the arities of f and g. Formally we define $s \times t$ as the term in $T_{\mathcal{F}_\times}$ given recursively by

$$f(s_1,\dots,s_n) \times g(t_1,\dots,t_m) \stackrel{\text{def}}{=} \begin{cases} fg(s_1 \times t_1, \dots, s_n \times t_n, \bot \times t_{n+1}, \dots, \bot \times t_m) \\ \qquad \text{if } n < m, \\ fg(s_1 \times t_1, \dots, s_n \times t_n, s_{n+1} \times \bot, \dots, s_m \times \bot) \\ \qquad \text{otherwise.} \end{cases}$$

For instance the product $f(a, g(b)) \times f(f(a,a), b)$ is $ff(af(\bot a, \bot a), gb(b\bot))$. This definition is extended to products of n terms $s_1 \times \ldots \times s_n$ in the obvious way.

Definition 4.1. *A n-ary relation $R \subseteq T_{\mathcal{F}} \times \cdots \times T_{\mathcal{F}}$ is* recognizable *iff the set of all $s_1 \times \ldots \times s_n$ for $\langle s_1, \ldots, s_n \rangle \in R$ is a regular tree language.*

For instance $Id \stackrel{\text{def}}{=} \{(s, s) \mid s \in T_{\mathcal{F}}\}$ is a recognizable relation and an automaton accepting Id needs only one state q (which is final) and rules $ff(q, \ldots, q) \longmapsto q$ for all $f \in \mathcal{F}$. The intersection, the union and the complement of recognizable n-ary relations are also recognizable. The main consequence is that the first-order theory of recognizable relations over finite trees is decidable, or, more precisely:

Theorem 4.2. *Let $\varphi(x_1, \ldots, x_n)$ be a first-order formula involving recognizable relations $R_1, \ldots$ and $Sol(\varphi)$ denote $\{\langle t_1, \ldots, t_n \rangle \mid \models \varphi(t_1, \ldots, t_n)\}$. Then $Sol(\varphi)$ is a recognizable subset of $T^n_{\mathcal{F}}$. Furthermore, from automata $\mathcal{A}_1, \ldots$ recognizing $R_1, \ldots,$ one can build an automaton $\mathcal{A}_\varphi$ recognizing $Sol(\varphi)$.*

4.1 Recognizability of the reachability relation

We can extend the proof that all $Post^*(X)$ are recognizable languages into a construction showing that the relation $\xrightarrow{*}$ is recognizable, providing a top-down tree automaton for $\xrightarrow{*}$. The main feature of the construction is to add three specific non-terminals I (for identity), R (for rewrite) and R' (for rewrite and termination) and other non-terminals $I_{\perp,s}$, $Q_{X,s}$, $Q_{\perp,s}$, $Q'_{X,s}$ and $Q'_{\perp,s}$ for X, s in $Sub(\Delta)$. The complete automaton is given in the full version of the paper.

Proposition 4.3. *For any Δ, the relation $\xrightarrow{*}$ between PA terms is a recognizable relation, and there is automaton with size $O(|\Delta|)$ recognizing it. The relations $\xrightarrow{+}$, $\rightarrow$ and $\xrightarrow{w}$ for any $w \in Act^*$ are recognizable.*

Since the image and the inverse image of a recognizable language via a recognizable relation is recognizable, the recognizability of $\xrightarrow{*}$ implies the regularity theorems of [LS99] as a byproduct. However, having recognizable $Post^*(L)$ and $Pre^*(L)$ does not necessarily entail the recognizability of $\xrightarrow{*}$. This can be illustrated in the PA framework, as the following remark shows.

Remark 4.4. An alternative definition of the reachability relation for PA is obtained by replacing the rule $(\mathrm{R}'_{\mathrm{S}})$ from section 2 with

$$(\mathrm{R}''_{\mathrm{S}}) \qquad \frac{t_1 \xrightarrow{w_1} t'_1 \quad t_2 \xrightarrow{w_2} t'_2}{t_1.t_2 \xrightarrow{w_1.w_2} t'_2} \text{ if } Const(t'_1) = \varnothing$$

(indeed, why not get rid of these useless terminated processes?). With this new definition, it is still true that, for regular $L \subseteq \mathcal{T}$, $Pre^*(L)$ and $Post^*(L)$ are regular tree languages, but the relation $\xrightarrow{*}$ is in general not recognizable. □

4.2 Costs for reachability

We can easily write simultaneously a cost grammar associated to the top-down automaton for $\xrightarrow{*}$ which satisfies the following property:

Proposition 4.5. *For any $s, t \in \mathcal{T}$:*

1. $R \overset{*}{\longmapsto} s \times t$ *iff* $s \xrightarrow{*} t$*. Furthermore, if* $s \xrightarrow{c} t$ *then* $c \in C^R$*, and if* $c \in C^R$ *then* $s \xrightarrow{c} t$ *for some* s, t *such that* $R \overset{*}{\longmapsto} s \times t$*.*
2. $R' \overset{*}{\longmapsto} s \times t$ *iff* $s \xrightarrow{*} t$ *and* t *is terminated. Furthermore, if* $s \xrightarrow{c} t$ *then* $c \in C^{R'}$*, and if* $c \in C^{R'}$ *then* $s \xrightarrow{c} t$ *for some* s, t *such that* $R' \overset{*}{\longmapsto} s \times t$*.*

5 *TL*, the first-order transition logic

Assume Δ is fixed. The first-order transition logic TL is the first-order logic with process variables $(u, v, \dots)$, the binary predicates $=$, $\xrightarrow{*}$ and $\rightarrow$, any other recognizable relation like, for instance, $u \in P$ for P a regular tree language. Observe that whether $t_1, \dots, t_n$ satisfies $\varphi(u_1, \dots, u_n)$ depends on the underlying PA declaration Δ. Theorem 4.2 yields the decidability of TL, or more precisely:

Corollary 5.1. *For any Δ and any TL formula $\varphi(u_1, \dots, u_n)$, we can build an automaton recognizing $Sol(\varphi)$ (a subset of $\mathcal{T}^n$).*

Expressivity Since quantifiers can be used freely, and since equality and other predicates are available, TL is more expressive than the modal logic EF handled in [May99,LS99]. For instance, the confluence of $\xrightarrow{*}$ is expressed by the TL formula $\forall u, v, v' \Big[\big(u \xrightarrow{*} v \wedge u \xrightarrow{*} v' \big) \Rightarrow \exists v'' \big(v \xrightarrow{*} v'' \wedge v' \xrightarrow{*} v'' \big) \Big]$.

Furthermore, the logic TL could be extended so that several PA declarations may be used simultaneously. E.g., we can state that the term reachable from X via Δ are the terms reachable via Δ_1 followed by Δ_2 by the formula

$$\forall u \Big[X \xrightarrow{*}_{\Delta} u \;\Leftrightarrow\; \exists v \big(X \xrightarrow{*}_{\Delta_1} v \wedge v \xrightarrow{*}_{\Delta_2} u \big) \Big].$$

Process terms with free variables, e.g., $(X.u) \parallel (v.0)$, can also be used since the predicates encoding the functions symbols are recognizable.

Parameterized verification and model measuring

Computing $Sol(\varphi)$ is more general than deciding validity, satisfiability, or model checking (telling whether $t \models \varphi(u)$ for a given t and φ). In model checking applications, being able to compute $Sol(\varphi)$ under a suitable symbolic representation (a tree automaton $\mathcal{A}_\varphi$ in our case) gives a general approach that smoothly integrate and generalize *parameterized verification* and *model measuring*

Example 5.2. Write X^n for $\overbrace{X \parallel (X \parallel (X \cdots \parallel X) \dots)}^{n \text{ copies of } X}$: the sets $L \overset{\text{def}}{=} \{X^n \mid n = 0, 1, 2, \dots\}$ and $L' \overset{\text{def}}{=} \{Y^n \mid n = 0, 1, 2, \dots\}$ are regular tree languages. Let $\varphi(u, v)$ be some TL formula comparing the behaviors of u and v (e.g. $\varphi \overset{\text{def}}{=} \forall z \big((v \xrightarrow{*} z \wedge z \xrightarrow{+} z) \Rightarrow u \xrightarrow{*} z \big)$, "all loops of v are loops of u"). By computing $Sol(u \in L \wedge v \in L' \wedge \varphi(u, v))$ we can find which values of n make X^n relate to some Y^m (or to all of them) and we can also find which values of m make the property $\varphi(u, Y^m)$ satisfiable (or valid) for the X^n's. □

6 Reachability under (decomposable) constraints

In this section, we enrich the basic reachability predicate used in TL and allow to write "decomposable" constraints upon the derivation costs.

6.1 The decomposable transition logic DTL

In [LS99], we proved that reachability via a trace constrained by a regular word language is undecidable but it is decidable for "decomposable" languages (see also [Sch99,May00]). Here we define *decomposable cost predicates* along the same lines for unary *cost predicate* P. We write $P(c)$ when P holds for c.

Definition 6.1. *A finite set $\mathcal{DP}$ of cost predicates is a* decomposable family *if*

seq-decompositions: *for all $P \in \mathcal{DP}$ there is a finite index set I and a family $\{P_i^1, P_i^2 \in \mathcal{DP} \mid i \in I\}$ s.t. for all $c, c' \in M$, $P(c \oplus c')$ iff $\bigvee_{i \in I} P_i^1(c) \wedge P_i^2(c')$.*

par-decompositions: *for all $P \in \mathcal{DP}$ there is a finite family $\{P_i^1, P_i^2 \in \mathcal{DP} \mid i \in I\}$ s.t. for all $c, c' \in M$, $P(c \otimes c')$ iff $\bigvee_{i \in I} P_i^1(c) \wedge P_i^2(c')$.*

unit-decompositions: *for all $P \in \mathcal{DP}$ and all costs c appearing in Δ, there is a finite family $\{P_i^c \in \mathcal{DP} \mid i \in I\}$ s. t. for all $c' \in M$, $P(c \oplus c')$ iff $\bigvee_{i \in I} P_i^c(c')$.*

A predicate P is *decomposable* if it belongs to a decomposable family. DTL (Decomposable Transition Logic) is the first-order logic that extends TL by allowing all atoms $u \xrightarrow{\exists c\ P(c)} v$ where P is any decomposable predicate.

$u \xrightarrow{\exists c\ P(c)} v$ is short for "$\exists c, u \xrightarrow{c} v \wedge P(c)$" and holds iff there is a derivation $u \xrightarrow{c} v$ such that $P(c)$ holds. Observe that we require that any cost variable be immediately quantified upon (like the freeze quantification of [AH94]), hence the cost variable c is always bound and we sometimes simply write $u \xrightarrow{\exists P} v$.

Using the tree automata approach, we can show that the relation $\xrightarrow{\exists P}$ is recognizable for any decomposable predicate P, which yields the decidability theorem:

Theorem 6.2. *The logic DTL is decidable.*

We now look at instances of DTL where costs measure some form of timing.

6.2 The timed transition logic *TTL*

TTL (Timed Transition Logic) is the first-order logic that extends TL by allowing all atoms $u \xrightarrow{\exists \tau} v$ where τ is a *time constraint* built according to the grammar:

$$\tau ::= c < C \mid \neg\tau \mid \tau \wedge \tau$$

where the C's can be any numerical constant from a time domain $\mathbb{T}$ that can be $\mathbb{N}$, or $\mathbb{Q}^+$, or $\mathbb{R}^+$ (and where c is the free cost variable of τ). Since these time constraints can be expressed by decomposable predicates, we have

Proposition 6.3. *For any time constraint τ, the relation $s \xrightarrow{\exists \tau} t$ is recognizable.*

allowing the following instantiation of Theorem 6.2

Theorem 6.4. *The logic TTL is decidable.*

TTL can be enriched with the predicate $u \xrightarrow{Unbounded} v$, meaning that going from u to v may take arbitrarily long time:

Lemma 6.5. *The relation $\xrightarrow{Unbounded}$ is recognizable.*

7 The transition logic with constraints TLC

In this section, we add a first-order logic of costs to the transition logic. The resulting two-sorted logic, called TLC, is very expressive and allows parameterized verification with parameters ranging over costs rather than "processes". For TLC, costs are Parikh costs and the cost $c = (n_1, \dots, n_p)$ of a derivation $s \xrightarrow{*} t$ records the number of occurrences of each action of Act along the derivation. Since c is now a p-tuple of integers, we often write $x_1, \dots, x_p$ or $\bar{x}$ instead of c. The formula over costs are Presburger formula. This allows to state properties as "$s \xrightarrow{w} t$ with *as many actions a as actions b*".

TLC allows two kinds of atoms: all $R(u_1, \dots, u_n)$ for R a recognizable relation as in TL and all $u \xrightarrow{\exists \bar{x} \psi(\bar{x},\bar{y})} v$ where $\psi(\bar{x},\bar{y})$ is a Presburger formula whose free variables are partitioned into $\bar{x}$ a tuple of p integer variables, for the cost of the derivation and the arbitrary parameters $\bar{y}$. $u \xrightarrow{\exists \bar{x} \psi(\bar{x},\bar{y})} v$ is short for "$\exists c, u \xrightarrow{c} v \wedge \psi(c,\bar{y})$". Observe that only $u, v, \bar{y}$ are free in $u \xrightarrow{\exists \bar{x} \psi(\bar{x},\bar{y})} v$. The negation of $u \xrightarrow{\exists \bar{x} \psi(\bar{x},\bar{y})} v$ can be written $\forall \bar{x} \big(u \xrightarrow{\bar{x}} v \Rightarrow \psi'(\bar{x},\bar{y})\big)$ where ψ' is $\neg\psi$, another Presburger formula, and we shall use this notation freely. TLC formulas are given by the abstract syntax:

$$\varphi ::= Atom \mid \varphi \wedge \varphi \mid \neg\varphi \mid \exists u \varphi \mid \exists y \varphi$$

Since the full TLC is undecidable (Prop. 7.1) we introduce two fragments that will be shown decidable (Theo. 7.3):

the parameterized existential fragment: which is the set of closed formulas that can be written under the form $(\exists|\forall\ \bar{y})^*(\exists\ u)^*[\vee \wedge Atoms]$, and

the parameterized universal fragment: which is the set of closed formulas that can be written under the form $(\exists|\forall\ \bar{y})^*(\forall\ u)^*[\vee \wedge \neg Atoms]$.

The restriction on the polarity of atoms only applies only to reachability atoms "$u \xrightarrow{\exists \bar{x} \psi(\bar{x},\bar{y})} v$" with some non-empty $\bar{y}$ hence $u \xrightarrow{*} v$, etc., can be negated freely. Observe that one fragment only contains formulas that are (equivalent to) the negation of a formula of the other fragment.

7.1 The difference between TLC and temporal logics

Temporal logics do not have a mechanism for identifying states precisely and relate them. On the other hand, they can refer to a given path, state properties that hold along this path, and relate paths. When we write $u \xrightarrow{\exists \bar{x} \psi(\bar{x},\bar{y})} v$ in TLC, we state that u may go to v via a path having some cost properties, but we cannot isolate this path and refer to it again. Additionally, temporal modalities are recursive by nature, while transition logics only have the fixpoint built-in $\xrightarrow{*}$. As a consequence, simple temporal modalities like EF can be expressed in the transition logic but more complex constructions like E_U cannot.

The ability to refer to states is the specific feature of transition logics: these logics can distinguish bisimilar processes. And the rich language for constraints allows to express many counting properties that occur naturally in verification.

7.2 Expressing properties with *TLC*

We consider the communication protocol example used in [BEH95b], where they focus on the two actions *req* (for requests) and *ack* (for acknowledgments).

TLC can state that, between an initial state and a final state, a protocol sends as many acknowledgments *ack* as requests *req* :

$$\forall u, v \ (u \in \mathit{Init} \wedge v \in \mathit{Final}) \Rightarrow \big(\forall x_{ack}, x_{req}((u \xrightarrow{x_{ack}, x_{req}} v) \Rightarrow x_{ack} = x_{req}\big)$$

where "$x_{ack} = x_{req}$" is the $\psi(\bar{x})$ constraint. (We assumed that Init and Final are regular tree languages.)

TLC can also state that at any position along a path from an initial state to a final state, the number of sent requests is always greater than or equal to the number of received acknowledgments. This is specified as follows:

$$\forall u, v, w \left(\begin{array}{c} u \in \mathit{Init} \wedge v \in \mathit{Final} \wedge u \xrightarrow{*} w \wedge w \xrightarrow{*} v \\ \Rightarrow \forall x_{ack}, x_{req}\big(u \xrightarrow{x_{ack}, x_{req}} w \Rightarrow x_{req} \geq x_{ack}\big) \end{array} \right)$$

TLC can further impose that all traces belong to $req^* ack^*$. We add to the previous formula the formula stating that if a state is reached by emitting an acknowledgment, then all paths from this state to a final state contain no request.

$$\forall u, v, w \left(\begin{array}{c} u \in \mathit{Init} \wedge v \in \mathit{Final} \wedge (u \xrightarrow{\exists x_{ack}, x_{ack}=1} w) \\ \Rightarrow \forall x_{ack}, x_{req}\big(w \xrightarrow{x_{ack}, x_{req}} v \Rightarrow x_{req} = 0\big) \end{array} \right)$$

Observe that these three examples are all in the parameterized universal fragment of TLC and could be enriched by using parameters.

7.3 Decidability issues for *TLC*

The first result is a negative one.

Proposition 7.1. *TLC is undecidable, even when restricted to the fragment without parameters and with only $\exists\forall$ quantification for process variables.*

This result prompted the introduction of the parameterized fragments of TLC. Let $\Phi \equiv (\exists|\forall\ \bar{y})^*(\exists\ u)^*\varphi$ be a parameterized existential TLC formula. Let $\phi(y_1, \ldots, y_k)$ denote the "$(\exists\ u)^*\varphi$" part and $Sol(\phi) = \{\langle n_1, \ldots, n_k\rangle \mid\ \models \phi(n_1, \ldots, n_k)\}$.

Theorem 7.2. *$Sol(\phi)$ is an effectively computable semilinear set.*

The proof of this result uses an extension of tree automata which combines a cost grammar with the usual tree automata rules. This class of automata is closed under product and union and the reachability of a state is decidable. Moreover in our case, the set of costs associated to a state is an effectively computable semilinear set and the relation $\xrightarrow{c}$ is accepted by an automaton with cost.

Theorem 7.3. *The parameterized existential and the parameterized universal fragments of TLC are decidable.*

8 Conclusion

The recognizability of $\xrightarrow{*}$ extends our earlier results on reachability sets. This also opens new directions for automata-theoretic approaches to the verification of PA-processes, since being able to compute the set of solutions of a transition logic formula allows a smooth and general approach to the verification of parameterized properties for parameterized systems. Additionally, the automata-theoretic approach relies on quite simple constructions. The consequence is that we can easily extend it in various ways, as we demonstrated with reachability under decomposable cost predicates, with various timed extensions of the transition logic, and with TLC where both PA-processes and Parikh costs can be constrained via parameterized formulas. An important goal for future work is to analyze the computational complexity of the various ideas we proposed. This should help understand what cost sets and what decomposable predicates can be handled in practice, and what restrictions may be fruitfully imposed on transition logics so that they remain computationally tractable.

References

[AELP99] R. Alur, K. Etessami, S. La Torre, and D. Peled. Parametric temporal logic for "model measuring". In *Proc. 26th Int. Coll. Automata, Languages, and Programming* , vol. 1644 of *L.N.C.S.*, pages 159–168. Springer, 1999.

[AH94] R. Alur and T. A. Henzinger. A really temporal logic. *J. of the ACM*, 41(1):181–203, 1994.

[BEH95a] A. Bouajjani, R. Echahed, and P. Habermehl. On the verification problem of nonregular properties for nonregular processes. In *Proc. 10th IEEE Symp. Logic Comp. Science,* , pages 123–133, 1995.

[BEH95b] A. Bouajjani, R. Echahed, and P. Habermehl. Verifying infinite state processes with sequential and parallel composition. In *Proc. 22nd ACM Symp. Princ. of Programming Languages* , pages 95–106, 1995.

[BW90] J. C. M. Baeten and W. P. Weijland. *Process Algebra*, vol. 18 of *Cambridge Tracts in Theoretical Computer Science.* Cambridge Univ. Press, 1990.

[CDG+99] H. Comon, M. Dauchet, R. Gilleron, D. Lugiez, S. Tison, and M. Tommasi. Tree Automata Techniques and Applications, 1997–99. Available at http://www.grappa.univ-lille3.fr/tata.

[CJ99] H. Comon and Y. Jurski. Timed automata and the theory of real numbers. In *Proc. 10th Int. Conf. Concurrency Theory* , vol. 1664 of *L.N.C.S.*, pages 242–257. Springer, 1999.

[DT90] M. Dauchet and S. Tison. The theory of ground rewrite systems is decidable. In *Proc. 5th IEEE Symp. Logic in Computer Science* , pages 242–248, 1990.

[EK99] J. Esparza and J. Knoop. An automata-theoretic approach to interprocedural data-flow analysis. In *Proc. 2nd Int. Conf. Found. of Soft. Sci. and Comp. Struct.* , vol. 1578 of *L.N.C.S.*, pages 14–30. Springer, 1999.

[EP00] J. Esparza and A. Podelski. Efficient algorithms for $pre*$ and $post*$ on interprocedural parallel flow graphs. In *Proc. 27th ACM Symp. Principles of Programming Languages* , 2000.

[Esp97] J. Esparza. Petri nets, commutative context-free grammars, and basic parallel processes. *Fundamenta Informaticae*, 31(1):13–25, 1997.

[ET99] E. A. Emerson and R. J. Trefler. Parametric quantitative temporal reasoning. In *Proc. 14th IEEE Symp. Logic in Comp. Sci.* , pp 336–343, 1999.

[HJ99] Y. Hirshfeld and M. Jerrum. Bisimulation equivalence is decidable for normed process algebra. In *Proc. 26th Int. Coll. Automata, Languages, and Programming* , vol. 1644 of *L.N.C.S.*, pages 412–421. Springer, 1999.

[JKM98] P. Jančar, A. Kučera, and R. Mayr. Deciding bisimulation-like equivalences with finite-state processes. In *Proc. 25th Int. Coll. Automata, Languages, and Programming* , vol. 1443 of *L.N.C.S.*, pages 200–211. Springer, 1998.

[Kuč96] A. Kučera. Regularity is decidable for normed PA processes in polynomial time. In *Proc. 16th Conf. Found. of Software Technology and Theor. Comp. Sci.* , vol. 1180 of *L.N.C.S.*, pages 111–122. Springer, 1996.

[Kuč97] A. Kučera. How to parallelize sequential processes. In *Proc. 8th Int. Conf. Concurrency Theory* , vol. 1243 of *L.N.C. S.*, pages 302–316. Springer, 1997.

[LS99] D. Lugiez and Ph. Schnoebelen. The regular viewpoint on PA-processes. September 1999. To appear in *Theor. Comp. Sci.*

[May97] R. Mayr. Tableaux methods for PA-processes. In *Proc. TABLEAUX'97* , vol. 1227 of *L.N.A.I.*, pages 276–290. Springer, 1997.

[May99] R. Mayr. Decidability of model checking with the temporal logic EF. May 1999. To appear in *Theor. Comp. Sci.*

[May00] R. Mayr. Process rewrite systems. *Information and Computation*, 156(1/2):264–286, 2000.

[Mol96] F. Moller. Infinite results. In *Proc. 7th Int. Conf. Concurrency Theory* , vol. 1119 of *L.N.C.S.*, pages 195–216. Springer, 1996.

[Par66] R. J. Parikh. On context-free languages. *J.A.C.M*, 13(4):570–581, 1966.

[Plo81] G. D. Plotkin. A structural approach to operational semantics. Lect. Notes, Aarhus University, Aarhus, DK, 1981.

[Sch99] Ph. Schnoebelen. Decomposable regular languages and the shuffle operator. *EATCS Bull.*, 67:283–289, 1999.

[Sei94] H. Seidl. Finite tree automata with cost function. *Theoretical Computer Science*, 126(1):113–142, 1994.