Decidable First-Order Transition Logics for PA-Processes ⋆

Denis Lugiez

Lab. d'Informatique Fondamentale de Marseille (LIF)
Univ. Aix-Marseille 1 & CNRS, France

Philippe Schnoebelen [1]

Lab. Spécification et Vérification (LSV)
ENS de Cachan & CNRS, France

Abstract

We show the decidability of model checking PA-processes against several first-order logics based upon the reachability predicate. The main tool for this result is the recognizability by tree automata of the reachability relation. The tree automata approach and the transition logics we use allow a smooth and general treatment of parameterized model checking for PA. This approach is extended to handle a quite 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.

Key words: Regular model checking, Verification of recursive parallel systems, Transition logics, Regular tree languages

---

1 Introduction

Verification of infinite-state systems is a very active field of research where one studies how the decidability results that underly the successful technology of

⋆ This article is an extended version of a paper with same title that was presented at 27th ICALP, Geneva, Switzerland, July 2000.

Email addresses: `lugiez@lif.univ-mrs.fr` (Denis Lugiez), `phs@lsv.ens-cachan.fr` (Philippe Schnoebelen).

[1] The second author is supported by the ACI Sécurité & Informatique (project Persée) funded by the French Ministry of Research.

model checking for finite state systems can be extended to more expressive computational models. In this field, many different models have been studied, ranging from infinite-data models (like channel systems) to infinite-control models (like process algebras), including timed automata and hybrid systems.

Infinite-state process algebras can have decidable verification problem, as was first shown by Baeten *et al.* [3]. Since then many results have been obtained, applying to more and more complex process algebras (see [8] for a recent survey). These results have applications to the static analysis of programs, in situations where one is willing to abstract from the data and concentrate on control where complex recursive behavior involving parallelism is at hand [16,17].

Among such process algebras, PA is one of the most expressive: PA [4] allows recursive definitions, sequential and parallel compositions. Actions are uninterpreted and do not synchronize. Recently, several verification problems have been shown decidable for PA (or extensions of PA) using a variety of fairly involved techniques (see, e.g., [6,5,24,25,22,21,30]).

An example PA process. Fig. 1 shows a toy program (in some imperative syntax) that computes the weight of a binary tree.

```
     function W (T)
l0:  if T->size > 100
     then
       cobegin
l1:       w1 := W (T->left)
l2:    [] w2 := W (T->right)
       coend
l3:    return T->val + w1 + w2
     else
l4:    return W_seq (T)
```

```
     function W_seq (T)
l5:  if T->size > 1
     then
l6:    w1 := W_seq (T->left)
l7:    w2 := W_seq (T->right)
l8:    return T->val + w1 + w2
     else
l9:    return T->val
```

Fig. 1. A divide-and-conquer program that uses coroutines

The program applies a simple divide-and-conquer strategy based on $\mathit{weight}(\mathtt{T}) = \mathit{weight}(\mathtt{T\text{->}left}) + \mathit{weight}(\mathtt{T\text{->}right})$. There is a twist however: the recursive calls are made in parallel (using coroutines in a "`cobegin .. [] .. coend`" block) for large trees, and in sequence for small trees (say, because one considers that for small trees the overhead for parallelism is discouraging).

PA is a natural formalism for modeling the behavior of programs like `Weight` at an abstract level. Typically, the PA process associated with such a program abstracts away from the data (so that `if .. then .. else` constructs reduce to non-determinism) but it keeps track of the flow of control, even when coroutines run concurrently.

For example, we can model the Weight program with the following PA system (the semantics of which is defined in section 2):

$$\begin{aligned} r_1 &: X_{10} \to (X_{11} \parallel X_{12}).X_{13}, & r_7 &: X_{15} \to X_{16}.(X_{17}.X_{18}), \\ r_2 &: X_{10} \to X_{14}, & r_8 &: X_{15} \to X_{19}, \\ r_3 &: X_{11} \to X_{10}, & r_9 &: X_{16} \to X_{15}, \\ r_4 &: X_{12} \to X_{10}, & r_{10} &: X_{17} \to X_{15}, \\ r_5 &: X_{13} \to 0, & r_{11} &: X_{18} \to 0, \\ r_6 &: X_{14} \to X_{15}, & r_{12} &: X_{19} \to 0. \end{aligned} \qquad (\Delta_{\mathtt{Weight}})$$

This PA system uses 9 constant names, one per statement in the Weight program, and 12 rewrite rules (named $r_1, \ldots, r_{12}$ for further reference).

Now, the behavior of the Weight program is mimicked by the PA term X_{10}. Of course, since we abstracted away from the data in a drastic way, the PA term X_{10} exhibit "more" behavior. Still, any safety property of X_{10} also holds for the Weight program.

Examples of such properties are (stated informally) "at any time, the degree of parallelism (largest number of parallel active subterms) is smaller than the number of pending X_{13}". Later we describe logics in which to express such properties, and methods to check them algorithmically.

Regular model checking for PA systems. In [27], we advocated *regular tree languages* and *tree automata* as an easy-to-use tool for tackling (some) problems about PA, and we proved that the reachability sets (both forward and backward) of a PA process (and more generally of a regular set of 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 (see [16,17] for applications in data-flow analysis). Our approach has been applied to other process algebras [20,26]. These automata techniques have also been applied to a subset of PRS [7].

Here we extend our previous work in several ways:

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

First-order transition logic: Recognizability of $\xrightarrow{*}$ immediately gives a decision method for the first-order transition logic of PA-processes, i.e., the first-order logic having $\to$, $\xrightarrow{*}$, and equality as basic predicates (plus any other recognizable predicates). The method actually 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 (the sequence of action names). These costs allow to encode various measures (e.g., degree of parallelism) 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 so that it can handle decomposable cost predicates and show several applications (e.g., decidability for various 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. Over PA-processes, TLC is not decidable but we isolate a rich fragment which is. This last result relies on *regular tree grammar with costs.* A similar logic $PTTL$ with (parameterized) costs encoding timing measures is shown to have similar properties.

Related work. Dauchet and Tison introduced the idea of using recognizable related in the study of ground rewrite systems [12]. They show that the first-order theory of the (iterated) rewrite relation of a ground rewrite system is decidable, using ground tree transducers (or GTT's, a class of tree automata that accept pairs of terms). PA processes are defined via ground rewrite rules, but there are restrictions on the application of transitions to sequential compositions, and this forbids using GTT's: indeed, the relation induced by a ground rewrite system is stable under context, which is not the case of the relation induced by PA processes. But the strong similarity between PA processes and ground rewrite systems suggested looking further into recognizable relations for PA.

Several temporal logics with cost constraints have been proposed for finite state systems (see [14,1] for recent proposals). Bouajjani *et al.* exhibit some decidable (fragments of) temporal logics over PA-processes [6,5], 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 6.1).

For costs that are Parikh images of traces, [15] 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. [11] 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 [15] handles tuples of integers (markings of BPP's) and [11] handles tuples of reals (clock values). For prefix word rewriting, [9] shows several applications of the recognizability of the transition relation.

Reachability in PA is investigated in [28,30]. The underlying methods apply to more general systems (like PRS [29]) but they are quite complex since they view terms modulo structural congruence. As explained in [27], we believe it is better to only introduce structural congruence at a later stage.

The combination of costs and tree automata has been studied by Seidl [36] with compiler optimization in mind (involving more general costs than ours), not decidability of logics on trees (where the relevant problem is the combination of cost automata). Actually our construction for decomposable predicates coincides with his construction for embedded costs in automata.

Plan of the article. We define PA in Section 2 and show simple recognizability results in Section 3. Recognizability of the reachability relation is dealt with in Section 4, opening the way to the decidability of several transition logics (Sections 6 and 7). We finally introduce TLC, a rich transition logic with Parikh costs in Section 8, and show decidability (of fragments) via regular tree grammars with costs. Then we describe $PTTL$ a transition logic with costs measuring time (Section 9).

2 The PA process algebra

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

(1) uses rewrite rules *à la* Moller [32],
(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 [34]).

(1) is now quite common in the field of infinite state systems. In [27], we showed the usefulness of (2) in the tree-automata view of PA. (3) is a general concept that allows measuring PA steps in decidable logics. (4) is required by our definition of costs (and makes things clearer as seen, e.g., in [17]).

2.1 Syntax

Assume $M = \{c_1, c_2, \ldots\}$ is a finite set of constant names called *cost units*. We write T_M to denote the set $\{c, c', \ldots\}$ of *cost terms* over M, given by the following abstract syntax

$$c, c' ::= 0_M \;\mid\; c \oplus c' \;\mid\; c \otimes c' \;\mid\; c_1 \;\mid\; c_2 \;\mid\; \cdots$$

We say a cost term is *null* if it is only made of 0_M's, $\oplus$'s and $\otimes$'s (i.e., contains no c_i from M).

Given a set $Const = \{X, Y, Z, \ldots\}$ of *process constants*, $\mathcal{T}_{Const}$, or $\mathcal{T}$ when the underlying $Const$ is clear, is the set $\{s, t, \ldots\}$ of *PA-terms*, given by the following abstract syntax

$$s, t ::= s.t \;\mid\; s \parallel t \;\mid\; 0 \;\mid\; X \;\mid\; Y \;\mid\; Z \;\mid\; \cdots$$

A PA *declaration* is a finite $Const$ with a finite set $\Delta \subseteq Const \times T_M \times \mathcal{T}$ of *process rewrite rules*. A rule $(X, c, t) \in \Delta$ is written $X \xrightarrow{c} t$. For technical convenience, we require that all $X \in Const$ appear in the left-hand side of at least one rule of Δ. Similarly, we assume that for any rule $X \xrightarrow{c} t$, the cost $c \in T_M$ is not null.

For $t \in \mathcal{T}$, we let $Const(t)$ denote the set of process constants occurring in t, and $Sub(t)$ 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 $|t| \overset{\text{def}}{=} Card(Sub(t))$ (i.e., the number of its subterms), and the size of a PA declaration is $|\Delta| \overset{\text{def}}{=} Card(Sub(\Delta))$.

2.2 Semantics

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

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

$$(\mathrm{R'}_\varepsilon)\quad \frac{}{X \xrightarrow{0_M} X} \qquad\qquad (\mathrm{R_P})\quad \frac{t_1 \xrightarrow{c_1} t'_1 \;\; t_2 \xrightarrow{c_2} t'_2}{t_1 \parallel t_2 \xrightarrow{c_1 \otimes c_2} t'_1 \parallel t'_2}$$

$$(\mathrm{R_S})\quad \frac{t_1 \xrightarrow{c_1} t'_1 \;\; t_2 \xrightarrow{c_2} t'_2}{t_1.t_2 \xrightarrow{c_1 \oplus c_2} t'_1.t'_2} \text{ if } Const(t'_1) = \varnothing \text{ or } c_2 \text{ is null}$$

where the condition "$Const(t'_1) = \varnothing$" is a syntactic way of stating that t'_1 is terminated, as can be understood from the following useful lemmas.

Lemma 2.1 *Assume $t \xrightarrow{c} t'$ and c is null. Then $t' = t$.*

PROOF (Sketch). By structural induction on the derivation of the step $t \xrightarrow{c} t$. There ($\mathrm{R_C}$) cannot be used since we assumed that null costs do not appear in rules in Δ. □

Lemma 2.2 *For all t, there is exactly one transition $t \xrightarrow{c} t'$ such that c is null.*

PROOF (Sketch). By structural induction on t. □

Lemma 2.3 *Assume $t \xrightarrow{c} t'$ and $Const(t) = \varnothing$. Then c is null.*

PROOF (Sketch). By structural induction on t. □

Lemma 2.4 *Assume $Const(t) \neq \varnothing$. Then there exists a step $t \xrightarrow{c} t'$ with a non-null c.*

PROOF (Sketch). By structural induction on t. Here we use the assumption that each $X \in Const$ appears in the left-hand side of at last one rule in Δ. □

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

We write $s \xrightarrow{*} t$ when $s \xrightarrow{c} t$ for some c. 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.

Remark 2.5 *Our definition is a big-steps semantics in which one cannot express directly the usual one-step transitions classically used for process algebra. Small-steps can usually be recovered through the cost labels. For example, if all rules in Δ carry a same cost unit $c_u \in M$, then a step $s \xrightarrow{c} t$ is a small-step iff c_u occurs exactly once in c.*

The cost c in a PA step $s \xrightarrow{c} t$ is an uninterpreted term. In later sections, we shall interpret cost terms in some semantic domain $\mathbb{M}$. Each cost unit is interpreted by some element of $\mathbb{M}$, and $\oplus$, $\otimes$ are interpreted by operations in $\mathbb{M}$ that admit as neutral element the interpretation of 0_M. This is extended in the standard way so that every cost term $c \in T_M$ denotes some $\mathbf{c} \in \mathbb{M}$, written $[\![c]\!] = \mathbf{c}$. It is then possible to define a new transition relation where costs are interpreted: we write $s \overset{\mathbf{c}}{\Rightarrow} t$ when $s \xrightarrow{c} t$ for some c with $[\![c]\!] = \mathbf{c}$.

In such interpretations for costs, $\oplus$ and $\otimes$ will usually be associative and commutative. One may want that, e.g., corresponding steps from $(t_1 \parallel t_2) \parallel t_3$ and from $t_1 \parallel (t_2 \parallel t_3)$ have the same costs.

Interpretations for cost terms will be not used until section 5. However, in order to illustrate the potential of this mechanism, we now provide a few example possibilities.

Example 2.6 (Traces as costs) *The usual definition of PA has transitions labeled with action names from some $Act = \{a, b, \ldots\}$ (and big-steps labeled with traces, i.e., sequences of action names).*

This can be recovered through cost labels: take finite non-empty subsets of Act^* as concrete costs, with $\oplus$ interpreted as language concatenation, $\otimes$ as language shuffle (both having $(0 =)\{\varepsilon\}$ as neutral element). If rules in Δ have the form $X \xrightarrow{\{a\}} t$ then $t \xrightarrow{w} t'$ in the usual PA semantics iff $w \in L$ for some L s.t. $t \xrightarrow{L} t'$ in our semantics with costs.

Example 2.7 (Parikh costs) *By interpreting costs in $\mathbb{M} = \mathbb{N}^p$ with both $\oplus$ and $\otimes$ denoting vector addition, we obtain another set of concrete costs. This can be used for counting occurrences of actions of each type (assuming $Act = \{a_1, \ldots, a_p\}$ contains p distinct actions). Here a rule $X \xrightarrow{\mathbf{c}} t \in \Delta$, where $\mathbf{c} = (0, \ldots, 0, 1, 0, \ldots, 0)$ has a 1 in position i, records one occurrence of a_i.*

Example 2.8 (Costs for timing) *It is natural to choose $\mathbb{M}$ to be a* time domain *$\mathbb{T}$ that can be $\mathbb{N}$, or $\mathbb{Q}^+$, or $\mathbb{R}^+$, ... A rule $X \xrightarrow{\mathbf{c}} t$ from Δ is labeled with its* duration *$\mathbf{c} \in \mathbb{T}$ and one can, for example, interpret $\oplus$ as addition and $\otimes$ as* **max**. *This interpretations assumes that the time it takes for the parallel composition of $t_1 \xrightarrow{*} s_1$ and $t_2 \xrightarrow{*} s_2$ is the maximum of the times any one of them takes.*

3 Tree languages, regular tree grammars, and regular cost grammars

We assume familiarity with finite trees (or terms) and only recall the basic notions from regular tree languages and tree grammars. We refer to [10] for more details.

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 $T_{\mathcal{F}}$.

Example 3.1 *With $\mathcal{F}_0 = \{a, b\}$, $\mathcal{F}_1 = \{g, h\}$ and $\mathcal{F}_2 = \{f\}$, $T_{\mathcal{F}}$ contains terms like a, $f(a, b)$, and $f(g(f(h(b), a)), b)$.*

The sets T_M and $\mathcal{T}$ from section 2.1 are two more examples.

3.1 Regular tree grammars and regular cost grammars

A *regular tree grammar* is a tuple $\mathcal{G} = \langle \mathcal{F}, \mathcal{Q}, Q_{Ax}, \delta \rangle$ where $\mathcal{F}$ is a finite ranked alphabet, $\mathcal{Q} = \{Q_1, \ldots\}$ is a finite set of *non-terminals*, $Q_{Ax} \in \mathcal{Q}$ is the *axiom*, and $\delta \subseteq \left(\mathcal{Q} \times \bigcup_{n \in \mathbb{N}} (\mathcal{F}_n \times \mathcal{Q}^n)\right) \cup \mathcal{Q} \times \mathcal{Q}$ is a finite set of *derivation rules*. A rule of the form $\langle Q, f, Q_1, \ldots, Q_n \rangle$ (where f has arity n) is usually denoted by $Q \longrightarrow f(Q_1, \ldots, Q_n)$, while a rule of the form $\langle Q, Q' \rangle$ is denoted by $Q \longrightarrow Q'$ (and is called an *ε-rule*). Below we follow the standard practice, when writing down a grammar, of grouping rules that share a same left-hand side and list the right-hand sides separated by a "|".

Example 3.2 *Take $\mathcal{F}$ as in Example 3.1 and let $\mathcal{Q} = \{Q_o, Q_e\}$ be a set of non-terminals, Q_e being the axiom. The following 10 rules make a regular tree grammar*

$$Q_o \longrightarrow a \mid b \mid g(Q_e) \mid h(Q_e) \mid f(Q_e, Q_e) \mid f(Q_o, Q_o)$$

$$Q_e \longrightarrow g(Q_o) \mid h(Q_o) \mid f(Q_o, Q_e) \mid f(Q_e, Q_o)$$

Grammar rules are rewrite rules between *mixed terms*, i.e., terms built on $\mathcal{F} \cup \mathcal{Q}$ where non-terminals can appear as nullary symbols. A one-step derivation, written $s \vdash t$, is possible when t is obtained from s by replacing one occurrence of some non-terminal Q in s by a term u that comes from a rule $(Q \longrightarrow u) \in \delta$. A *derivation of t_n from t_0* is a sequence of steps $t_0 \vdash t_1 \vdash \cdots \vdash t_n$, denoted by $t_0 \vdash^* t_n$.

Example 3.3 *With the above grammar, a possible derivation is*

$$Q_e \vdash f(Q_e, Q_o) \vdash f(h(Q_o), Q_o) \vdash f(h(Q_o), a) \vdash f(h(b), a),$$

so that $Q_e \vdash^* f(h(b), a)$. More generally, one proves that $Q_e \vdash^* t$ (resp. $Q_o \vdash^* t$) iff t is a mixed term built with an even (resp. odd) number of symbols.

Below we shall also use regular tree grammars for cost terms, and call them *regular cost grammars*. This profits from the fact that cost terms are just trees over the signature $\mathcal{F}_M$ consisting of M augmented by 0, $\oplus$ and $\otimes$.

3.2 Regular tree languages

The *tree language* generated by a non-terminal Q (assuming some underlying grammar) is $L(Q) \stackrel{\text{def}}{=} \{t \in T_{\mathcal{F}} \mid Q \vdash^* t\}$. Note that $L(Q)$ has no mixed terms. The language $L(\mathcal{G})$ generated by the grammar $\mathcal{G}$ is $L(Q_{Ax})$, where Q_{Ax} is the axiom of $\mathcal{G}$.

The *size* $|\mathcal{G}|$ of a regular tree grammar $\mathcal{G}$ is defined in the usual way, as the number of symbols it takes to write the rules. One can decide if $L(\mathcal{G})$ is empty or not in time $O(|\mathcal{G}|)$.

We say that $L \subseteq T_{\mathcal{F}}$ is a *regular* tree language, if $L = L(\mathcal{G})$ for some regular tree grammar $\mathcal{G}$. For example, the language of $\mathcal{F}$-terms having an even number of symbols is regular, as explained by Example 3.3. It is well-known that regular tree languages are closed under union, intersection, and complementation.

Considering PA-processes as trees allows developing symbolic approaches based on regular languages, as in [23]. Regular sets of PA-terms are easy to manipulate symbolically, but they are expressive enough to denote many interesting sets. For example, by Lemmas 2.3 and 2.4, the set of terminated processes is $\mathit{Final} \stackrel{\text{def}}{=} \{t \mid \mathit{Const}(t) = \varnothing\}$, which is easily seen to be regular.

A more general result is given by the *Regularity Theorems* of [27]: if L is a regular set of PA-terms, then $\mathit{Post}^*(L)$ and $\mathit{Pre}^*(L)$ are regular too.

3.3 Regular tree grammars vs. bottom-up tree automata

Reversing the arrows in the derivation rules of a regular tree grammar transforms the generating device into an accepting device, i.e., a *bottom-up tree automaton*, and the *recognizable* tree languages are defined as the languages accepted by these automata. Both devices have the same power in the sense that a language is regular iff it is recognizable (see [10] for details). Actually, the algorithms used for combining regular languages, deciding the emptiness of $L(\mathcal{G})$, etc., have been designed for tree automata. However, we use regular tree grammars in this article because they fit our approach better. Since moving from a grammar to a tree automaton, and *vice versa*, is straightforward (simply reverse the arrows!) we still have at hand all the algorithmics designed for tree automata.

3.4 Regularity of $Post^*(X)$

We now show that, for any PA-declaration Δ, and any $X \in Const$, the set $Post^*(X)$ is a regular tree language. This result is already present in [27] but we recall it since the construction will be extended later in this paper (and the definitions in section 2 are slightly different from those in [27]).

Describing $Post^*(t)$ for a given t is easy after we make some observations. Consider some $t \in \mathcal{T}$ and write t under the form $C[X_1, \ldots, X_n]$ where $C[\,]$, the skeleton, is a n-holes context made out of "0"s, "$\|$"s and "."s, and where $X_1, \ldots, X_n$ are the n different occurrences of process constants in t (note that the X_i's need not be distinct).

Assume now $t \xrightarrow{*} u$. Then the skeleton $C[\,]$ of t has been preserved in u, and u is some $C[u_1, \ldots, u_n]$ where $X_i \xrightarrow{*} u_i$. This is illustrated in Fig. 2.

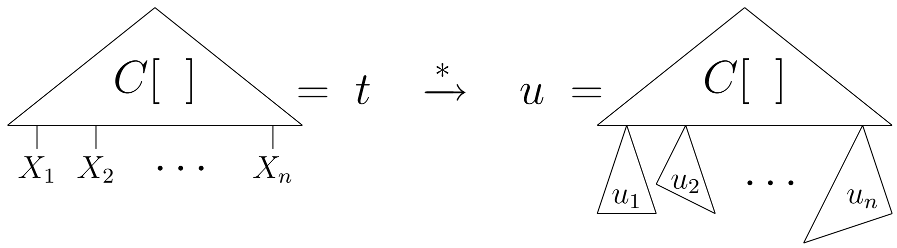

Fig. 2. A step from t to u

However, not every such $C[u_1, \ldots, u_n]$ is reachable from t: the priority rule for sequential composition must be obeyed, i.e., a X_i to the right of some sequential composition operator "." can only be transformed if the left-hand side of that "." is terminated. More formally, we have

Lemma 3.4 *$C[X_1, \ldots, X_n] \xrightarrow{*} C[u_1, \ldots, u_n]$ iff $X_i \xrightarrow{*} u_i$ for $i = 1, \ldots, n$ and, for any i, j such that X_i and X_j occur respectively to the left and to the right of a same occurrence of "." in $C[\,]$, either u_i is terminated, or $u_j = X_j$.*

For example, assume $t = (X_1 \parallel X_2).X_3$, $u_1 \in Post^*(X_1)$, $u_2 \in Post^*(X_2)$ and $u_3 \in Post^*(X_3)$. Then $(u_1 \parallel u_2).u_3 \in Post^*(t)$ iff $u_3 = X_3$ or $u_1 \parallel u_2$ is terminated (i.e., u_1 and u_2 are terminated).

It is now easy to write a regular grammar for $\mathit{Post}^*(X)$: the non-terminals are all Q_s, Q'_s and I_s for $s \in \mathit{Sub}(\Delta)$ and we give rules such that for any $s \in \mathit{Sub}(\Delta)$ we have:

$$\begin{aligned} L(I_s) &= \{s\}, \\ L(Q_s) &= \mathit{Post}^*(s), \\ L(Q'_s) &= \{t \in \mathit{Post}^*(s) \mid \mathit{Const}(t) = \varnothing\}. \end{aligned} \tag{1}$$

In other words, Q'_s generates all terminated processes in $\mathit{Post}^*(s)$.

The following rules ensure $L(I_s) = \{s\}$:

$$\begin{aligned} I_0 &\longrightarrow 0 && \text{if } 0 \in \mathit{Sub}(\Delta) \\ I_Y &\longrightarrow Y && \text{for all } Y \in \mathit{Sub}(\Delta) \\ I_{s_1 \| s_2} &\longrightarrow I_{s_1} \parallel I_{s_2} && \text{for all } s_1 \parallel s_2 \in \mathit{Sub}(\Delta) \\ I_{s_1.s_2} &\longrightarrow I_{s_1}.I_{s_2} && \text{for all } s_1.s_2 \in \mathit{Sub}(\Delta) \end{aligned} \tag{δ_1}$$

We now need rules for the Q_s and Q'_s non-terminals, ensuring (1). When s is not some $Y \in \mathit{Const}$, the rules keep track of the structure of s, relying on Lemma 3.4:

$$\begin{aligned} &\left.\begin{aligned} Q_0 &\longrightarrow 0 \\ Q'_0 &\longrightarrow 0 \end{aligned}\right\} \text{if } 0 \in \mathit{Sub}(\Delta) \\ &\left.\begin{aligned} Q_{s_1 \| s_2} &\longrightarrow Q_{s_1} \parallel Q_{s_2} \\ Q'_{s_1 \| s_2} &\longrightarrow Q'_{s_1} \parallel Q'_{s_2} \end{aligned}\right\} \text{for all } s_1 \parallel s_2 \in \mathit{Sub}(\Delta) \\ &\left.\begin{aligned} Q_{s_1.s_2} &\longrightarrow Q_{s_1}.I_{s_2} \mid Q'_{s_1}.Q_{s_2} \\ Q'_{s_1.s_2} &\longrightarrow Q'_{s_1}.Q'_{s_2} \end{aligned}\right\} \text{for all } s_1.s_2 \in \mathit{Sub}(\Delta) \end{aligned} \tag{δ_2}$$

When s is some $Y \in \mathit{Const}$, we rely on

$$\mathit{Post}^*(Y) = \{Y\} \cup \bigcup_{Y \xrightarrow{c} t \in \Delta} \mathit{Post}^*(t)$$

which leads to the following grammar rules (note that the last rules are ε-

rules):

$$\left.\begin{array}{lcl} Q_Y & \longrightarrow & Y \end{array}\right\}\text{for all } Y \in \mathit{Sub}(\Delta)$$

$$\left.\begin{array}{lcl} Q_Y & \longrightarrow & Q_s \\ Q'_Y & \longrightarrow & Q'_s \end{array}\right\}\text{for all } Y \xrightarrow{c} s \in \Delta \qquad (\delta_3)$$

Finally, the rules in $\delta_1 \cup \delta_2 \cup \delta_3$ ensure (1), and $\mathit{Post}^*(X)$ is the language generated by axiom Q_X.

3.5 A cost grammar for derivations

It is possible to provide a regular cost grammar that generates exactly the cost terms labeling possible PA steps. The cost grammar (called $\mathcal{C}_{\mathit{Post}^*}$) is obtained with the same techniques that provided a regular tree grammar (called $\mathcal{G}_{\mathit{Post}^*}$) for $\mathit{Post}^*(X)$, and the rules in the two grammars are in close correspondence.

Here we give the two grammars side by side:

$$\left.\begin{array}{rcl@{\qquad}rcl} I_0 & \longrightarrow & 0 & C^I_0 & \longrightarrow & 0_M \\ Q_0 & \longrightarrow & 0 & C^Q_0 & \longrightarrow & 0_M \\ Q'_0 & \longrightarrow & 0 & C^{Q'}_0 & \longrightarrow & 0_M \end{array}\right\}\text{if } 0 \in \mathit{Sub}(\Delta)$$

$$\left.\begin{array}{rcl@{\qquad}rcl} I_Y & \longrightarrow & Y & C^I_Y & \longrightarrow & 0_M \\ Q_Y & \longrightarrow & Y & C^Q_Y & \longrightarrow & 0_M \end{array}\right\}\text{for all } Y \in \mathit{Sub}(\Delta)$$

$$\left.\begin{array}{rcl@{\qquad}rcl} Q_Y & \longrightarrow & Q_s & C^Q_Y & \longrightarrow & c \oplus C^Q_s \\ Q'_Y & \longrightarrow & Q'_s & C^{Q'}_Y & \longrightarrow & c \oplus C^{Q'}_s \end{array}\right\}\text{for all } Y \xrightarrow{c} s \in \Delta$$

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

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

$\mathcal{G}_{Post^*}$ and $\mathcal{C}_{Post^*}$ describe the intended reachability sets and the associated sets of costs in the following sense:

Proposition 3.5 *For all $t \in \mathit{Sub}(\Delta)$:*

(1) $I_t \vdash^* s$ iff $s = t$ (and hence $t \xrightarrow{0_M} s$). Furthermore, $C^I_t \vdash^* c$ iff $c = 0_M$.
(2) $Q_t \vdash^* s$ iff $t \xrightarrow{*} s$. Furthermore, if $t \xrightarrow{c} s$ then $C^Q_t \vdash^* c$, and if $C^Q_t \vdash^* c$ then $t \xrightarrow{c} s'$ for some s'.
(3) $Q'_t \vdash^* s$ iff $t \xrightarrow{*} s$ and s is terminated. Furthermore, if $t \xrightarrow{c} s$ then $C^{Q'}_t \vdash^* c$, and if $C^{Q'}_t \vdash^* c$ then $t \xrightarrow{c} s'$ for some terminated s'.

The proof (omitted) is by structural induction and is similar to the proof of the regularity theorems in [27].

4 Regular tree grammars and n-ary relations

In this section, we explain how one can go beyond the regularity of $Post^*(t)$ and $Pre^*(t)$: the relation $\xrightarrow{*}$ itself can be generated by a regular tree grammar. But this development requires that we first define a notion of *recognizable tree relations.*

4.1 Recognizable tree relations

We follow [12]. Given two terms $s, t \in T_{\mathcal{F}}$, the pair (s, t) can be seen as a single term over a product alphabet $\mathcal{F}_\times \stackrel{\text{def}}{=} (\mathcal{F} \cup \{\perp\}) \times (\mathcal{F} \cup \{\perp\}) - \{(\perp, \perp)\}$ where $\perp$ is a new symbol with arity 0. A symbol (f, g) in $\mathcal{F}_\times$ is written shortly fg and its arity 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

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

For instance the product $f(a, g(b)) \times f(f(a, a), b)$ is $ff(af(\perp a, \perp a), gb(b\perp))$. 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 (seen as a set of terms built on the product alphabet).*

For instance $Id \stackrel{\text{def}}{=} \{\langle s, s\rangle \mid s \in T_{\mathcal{F}}\}$ is a recognizable relation and a tree grammar generating Id needs only one non-terminal I, and rules $I \longrightarrow ff(I,\ldots,I)$ for all $f \in \mathcal{F}$.

Intersections, unions, and complements of recognizable n-ary relations are also recognizable. For $1 \le i \le n$, the ith *projection* of a n-ary relation R is the $(n-1)$-ary relation obtained by suppressing the ith component in any n-tuple of R. For $0 \le i \le n$, the ith *cylindrification* (also called *inverse projection*) of R is the largest $(n + 1)$-ary relation whose $(i + 1)$th projection is R. Both the ith projection and the ith cylindrification of a recognizable relation R are recognizable.

The important corollary is that the first-order theory of recognizable relations over finite trees is decidable, or, more precisely:

Theorem 4.2 ([37, Lemma 19]) *Let $\varphi(x_1,\ldots,x_n)$ be a first-order formula over $\mathcal{F}$-trees where the predicates denote recognizable relations $R_1,\ldots$. Let $Sol(\varphi)$ denote $\{\langle t_1,\ldots,t_n\rangle \mid\ \models \varphi(t_1,\ldots,t_n)\}$, the set of n-tuples described by φ, or "the solutions of φ". Then $Sol(\varphi)$ is a recognizable subset of $T^n_{\mathcal{F}}$. Furthermore, from regular grammars $\mathcal{G}_1,\ldots$ generating $R_1,\ldots$, one can build a regular tree grammar $\mathcal{G}_\varphi$ recognizing $Sol(\varphi)$.*

Remark 4.3 *The construction of $\mathcal{G}_\varphi$ may require nested exponential steps (for each alternation of universal and existential quantifications in φ) but this probably cannot be avoided since the first-order logic of finite trees has a non-elementary decision problem [31].*

Once the notion of recognizable relations is understood, our earlier construction for the recognizability of $Post^*(X)$ is easily extended into a construction for the recognizability of $\xrightarrow{*}$. For non-terminals we consider all $I_{\perp,s}$, $Q_{X,s}$, $Q_{\perp,s}$, $Q'_{X,s}$ and $Q'_{\perp,s}$ for $X \in Const$ and $s \in Sub(\Delta)$, to which we add three specific non-terminals I, R and R'.

We shall give rules ensuring that for any $s, X \in Sub(\Delta)$

$$\begin{aligned} L(I_{\perp,s}) &= \{\perp \times s\} \\ L(Q_{X,s}) &= \{X \times t \mid s \xrightarrow{*} t\} \\ L(Q'_{X,s}) &= \{X \times t \mid s \xrightarrow{*} t \text{ and } t \text{ is terminated}\} \\ L(Q_{\perp,s}) &= \{\perp \times t \mid s \xrightarrow{*} t\} \\ L(Q'_{\perp,s}) &= \{\perp \times t \mid s \xrightarrow{*} t \text{ and } t \text{ is terminated}\}, \end{aligned} \tag{2}$$

and

$$\begin{aligned} L(I) &= \{s \times s\} \\ L(R) &= \{s \times t \mid s \xrightarrow{*} t\} \\ L(R') &= \{s \times t \mid s \xrightarrow{*} t \text{ and } t \text{ is terminated}\}. \end{aligned} \tag{3}$$

Hence a non-terminal like $Q_{X,s}$ recognizes $\{X \times t \mid Q_s \vdash^* t \text{ in } \mathcal{G}_{Post^*}\}$ so that the corresponding rules are small variants of the rules given in (δ_1–δ_3) for $\mathcal{G}_{Post^*}$ (we give them in full in Section 4.3 when costs are accounted for).

Now, if we assume (2), the rules for R, R' and I are straightforward [2]:

$$\begin{aligned} I &\longrightarrow \|\|\ (I, I) \mid ..(I, I) \mid 00 \mid XX \mid YY \mid \ldots \\ R &\longrightarrow \|\|\ (R, R) \mid ..(R, I) \mid ..(R', R) \mid 00 \mid Q_{X,X} \mid Q_{Y,Y} \mid \ldots \\ R' &\longrightarrow \|\|\ (R', R') \mid ..(R', R') \mid 00 \mid Q'_{X,X} \mid Q'_{Y,Y} \mid \ldots \end{aligned} \tag{δ_4}$$

The interesting consequence is:

Proposition 4.4 *For any Δ, the relation $\xrightarrow{*}$ between PA terms is a recognizable relation, and there is a regular tree grammar with size $O(|\Delta|^2)$ that generates it.*

[2] The reader must understand that symbols like "$\|\|$", "..", "00", etc., belong to the product alphabet $\mathcal{F}_\times$. The full grammar (see Fig. 3) uses further product symbols like "$\perp 0$", "$Y\ \|$", "$X.$", etc.

Since the image of a recognizable language via a recognizable relation is recognizable, the regularity theorems of [27] are direct corollaries of Proposition 4.4. Since the non-terminals of the grammar are indexed by pairs u, v where u is a name of Δ or $\perp$ and v a subterm of Δ or $\perp$, the size of the grammar is $O(|\Delta|^2)$.

Remark 4.5 *The recognizability of $\xrightarrow{*}$ is a stronger result than the regularity of $Post^*(L)$ and $Pre^*(L)$ for regular L. In particular, there exist alternative ways of defining PA, where regularity theorems hold but where $\xrightarrow{*}$ is not recognizable.*

For example, an alternative definition of PA is obtained by replacing the rule $(\mathrm{R}'_{\mathrm{S}})$ from Section 2 with

$$(\mathrm{R}''_{\mathrm{S}}) \qquad \frac{t_1 \xrightarrow{c_1} t'_1 \quad t_2 \xrightarrow{c_2} t'_2}{t_1.t_2 \xrightarrow{c_1 \otimes c_2} t'_2} \text{ if } Const(t'_1) = \varnothing$$

(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 (see Appendix A). This is one more justification for our choice of SOS rules for PA.

4.3 Costs for reachability

It turns out we can easily write a cost grammar side by side with the regular tree grammar for $\xrightarrow{*}$. The whole construction is given in Fig. 3, where the usual quantifications "for all $Y \xrightarrow{c} s \in \Delta$, etc." have been collected at the bottom of the page but are exactly like everywhere else in this work.

The fundamental property of this tree grammar and the associated cost grammar is:

Proposition 4.6 *For any $s, t \in \mathcal{T}$:*

i. $R \vdash^* s \times t$ *iff* $s \xrightarrow{*} t$. *Furthermore, if* $s \xrightarrow{c} t$ *then* $C^R \vdash^* c$, *and if* $C^R \vdash^* c$ *then* $s \xrightarrow{c} t$ *for some* s, t *such that* $R \vdash^* s \times t$.
ii. $R' \vdash^* s \times t$ *iff* $s \xrightarrow{*} t$ *and* t *is terminated. Furthermore, if* $s \xrightarrow{c} t$ *then* $C^{R'} \vdash^* c$, *and if* $C^{R'} \vdash^* c$ *then* $s \xrightarrow{c} t$ *for some* s, t *such that* $R' \vdash^* s \times t$.

PROOF (Idea). One completes *i.* and *ii.* with other similar statements that cover the other non-terminals of the grammar. This gives nine statements

$$
\begin{array}{rrcl@{\qquad}rcl}
(1) & I & \longrightarrow & \|\|\| \, (I, I) & C^I & \longrightarrow & C^I \otimes C^I \\
(2) & & | & ..(I, I) & & | & C^I \oplus C^I \\
(3) & & | & 00 & & | & 0_M \\
(4) & & | & YY & & | & 0_M \\
(5) & I_{\perp, s_1 \| s_2} & \longrightarrow & \perp \| \, (I_{\perp, s_1}, I_{\perp, s_2}) & C^I_{\perp, s_1 \| s_2} & \longrightarrow & C^I_{\perp, s_1} \otimes C^I_{\perp, s_2} \\
(6) & I_{\perp, s_1 . s_2} & \longrightarrow & \perp . (I_{\perp, s_1}, I_{\perp, s_2}) & C^I_{\perp, s_1 . s_2} & \longrightarrow & C^I_{\perp, s_1} \oplus C^I_{\perp, s_2} \\
(7) & I_{\perp, 0} & \longrightarrow & \perp 0 & C^I_{\perp, 0} & \longrightarrow & 0_M \\
(8) & I_{X, 0} & \longrightarrow & X0 & C^I_{X, 0} & \longrightarrow & 0_M \\
(9) & I_{\perp, Y} & \longrightarrow & \perp Y & C^I_{\perp, Y} & \longrightarrow & 0_M \\
(10) & R & \longrightarrow & \|\|\| \, (R, R) & C^R & \longrightarrow & C^R \otimes C^R \\
(11) & & | & ..(R, I) & & | & C^R \oplus C^I \\
(12) & & | & ..(R', R) & & | & C^{R'} \oplus C^R \\
(13) & & | & 00 & & | & 0_M \\
(14) & & | & Q_{X,X} & & | & C^Q_{X,X} \\
(15) & Q_{\perp, s_1 \| s_2} & \longrightarrow & \perp \| \, (Q_{\perp, s_1}, Q_{\perp, s_2}) & C^Q_{\perp, s_1 \| s_2} & \longrightarrow & C^Q_{\perp, s_1} \otimes C^Q_{\perp, s_2} \\
(16) & Q_{\perp, s_1 . s_2} & \longrightarrow & \perp . (Q'_{\perp, s_1}, Q_{\perp, s_2}) & C^Q_{\perp, s_1 . s_2} & \longrightarrow & C^{Q'}_{\perp, s_1} \oplus C^Q_{\perp, s_2} \\
(17) & & | & \perp . (Q_{\perp, s_1}, I_{\perp, s_2}) & & | & C^Q_{\perp, s_1} \oplus C^I_{\perp, s_2} \\
(18) & Q_{X, s_1 . s_2} & \longrightarrow & X . (Q'_{\perp, s_1}, Q_{\perp, s_2}) & C^Q_{X, s_1 . s_2} & \longrightarrow & C^{Q'}_{\perp, s_1} \oplus C^Q_{\perp, s_2} \\
(19) & & | & X . (Q_{\perp, s_1}, I_{\perp, s_2}) & & | & C^Q_{\perp, s_1} \oplus C^I_{\perp, s_2} \\
(20) & Q_{\perp, 0} & \longrightarrow & \perp 0 & C^Q_{\perp, 0} & \longrightarrow & 0_M \\
(21) & Q_{X, 0} & \longrightarrow & X0 & C^Q_{X, 0} & \longrightarrow & 0_M \\
(22) & Q_{\perp, Y} & \longrightarrow & \perp Y & C^Q_{\perp, Y} & \longrightarrow & 0_M \\
(23) & & | & Q_{\perp, s} & & | & c \oplus C^Q_{\perp, s} \\
(24) & Q_{X, Y} & \longrightarrow & XY & C^Q_{X, Y} & \longrightarrow & 0_M \\
(25) & & | & Q_{X, s} & & | & c \oplus C^Q_{X, s} \\
(26) & R' & \longrightarrow & \|\|\| \, (R', R') & C^{R'} & \longrightarrow & C^{R'} \otimes C^{R'} \\
(27) & & | & ..(R', R') & & | & C^{R'} \oplus C^{R'} \\
(28) & & | & 00 & & | & 0_M \\
(29) & & | & Q'_{X,X} & & | & C^{Q'}_{X,X}
\end{array}
$$

$$
\begin{array}{lrclrcl}
(30) & Q'_{\perp,s_1\|s_2} & \longrightarrow & \perp \parallel (Q'_{\perp,s_1}, Q'_{\perp,s_2}) & C^{Q'}_{\perp,s_1\|s_2} & \longrightarrow & C^{Q'}_{\perp,s_1} \otimes C^{Q'}_{\perp,s_2} \\
(31) & Q'_{X,s_1\|s_2} & \longrightarrow & X \parallel (Q'_{\perp,s_1}, Q'_{\perp,s_2}) & C^{Q'}_{X,s_1\|s_2} & \longrightarrow & C^{Q'}_{\perp,s_1} \otimes C^{Q'}_{\perp,s_2} \\
(32) & Q'_{\perp,s_1.s_2} & \longrightarrow & \perp.(Q'_{\perp,s_1}, Q'_{\perp,s_2}) & C^{Q'}_{\perp,s_1.s_2} & \longrightarrow & C^{Q'}_{\perp,s_1} \oplus C^{Q'}_{\perp,s_2} \\
(33) & Q'_{X,s_1.s_2} & \longrightarrow & X.(Q'_{\perp,s_1}, Q'_{\perp,s_2}) & C^{Q'}_{X,s_1.s_2} & \longrightarrow & C^{Q'}_{\perp,s_1} \oplus C^{Q'}_{\perp,s_2} \\
(34) & Q'_{\perp,0} & \longrightarrow & \perp 0 & C^{Q'}_{\perp,0} & \longrightarrow & 0_M \\
(35) & Q'_{X,0} & \longrightarrow & X0 & C^{Q'}_{X,0} & \longrightarrow & 0_M \\
(36) & Q'_{\perp,Y} & \longrightarrow & Q'_{\perp,s} & C^{Q'}_{\perp,Y} & \longrightarrow & c \oplus C^{Q'}_{\perp,s} \\
(37) & Q'_{X,Y} & \longrightarrow & Q'_{X,s} & C^{Q'}_{X,Y} & \longrightarrow & c \oplus C^{Q'}_{X,s}
\end{array}
$$

Fig. 3. Regular tree grammar for $\xrightarrow{*}$ and associated cost grammar. The rules are given for all $X, s_1.s_2, s_1 \parallel s_2 \in \mathit{Sub}(\Delta)$, and for all $Y \xrightarrow{c} s \in \Delta$.

(omitted, but they follow the pattern of Equations (2–3)) that we prove simultaneously.

Then the proof is in several steps:

1. We first prove all statements of the form "$R\vdash^* s \times t$ implies $s \xrightarrow{c} t$ and $C^R\vdash^* c$" by induction over the derivation of $R\vdash^* s \times t$. There are 37 cases to consider, depending on which grammar rule is used first. Let us mention a few typical cases:

rule (11): the derivation is some $R \vdash ..(R, I)\vdash^* s \times t$. Then $s \times t$ has the form $(s_1.s_2) \times (t_1.t_2)$, with $R\vdash^* s_1 \times t_1$ and $I\vdash^* s_2 \times t_2$. By ind. hyp. $s_1 \xrightarrow{c_1} t_1$ for some c_1 with $C^R\vdash^* c_1$, and $s_2 = t_2$ with $s_2 \xrightarrow{c_2} t_2$ for a null c_2 s.t. $C^I\vdash^* c_2$. Then, writing c for $c_1 \oplus c_2$, we deduce $s \xrightarrow{c} t$, by PA rule $\mathrm{R_S}$, and $C^R\vdash^* c$ by rule (11).

rule (25): the derivation is some $Q_{X,Y} \vdash Q_{X,s}\vdash^* u \times t$. Then by ind. hyp. $u \times t$ has the form $X \times t$ and $s \xrightarrow{c'} t$ with $C^Q_{X,s}\vdash^* c'$. Since $Y \xrightarrow{c} s$ is a rule in Δ, $Y \xrightarrow{c \oplus c'} t$ is a valid step, and rule (25) entails $C^Q_{X,Y}\vdash^* c \oplus c'$.

2. We prove all statements of the form "$s \xrightarrow{*} t$ implies $R\vdash^* s \times t$". This is done simultaneously for all non-terminals by induction over the derivation of $s \xrightarrow{*} t$. We only mention two typical cases:

non-terminals R and rule $\mathrm{R_C}$: the derivation $s \xrightarrow{*} t$ is $X \xrightarrow{*} t$ deduced from a derivation of $u \xrightarrow{*} t$ and the existence of a rule $X \rightarrow u$ in Δ. Then $Q_{X,u}\vdash^* X \times t$ by ind. hyp., so that $R \vdash Q_{X,X}\vdash^* X \times t$ using rule (14) and then (25).

non-terminal $Q_{\perp,s}$ and PA rule $\mathrm{R_S}$: the derivation $s \xrightarrow{*} t$ is $s_1.s_2 \xrightarrow{*} t_1.t_2$ deduced from a $s_1 \xrightarrow{c_1} t_1$ and $s_2 \xrightarrow{c_2} t_2$ with null c_2, entailing $t_2 = s_2$

(Lemma 2.1). Then, by ind. hyp., $Q_{\perp,s_1} \vdash^* \perp \times t_1$ and $I_{\perp,s_2} \vdash^* \perp \times s_2$. So that $Q_{\perp,s_1.s_2} \vdash \perp.(Q_{\perp,s_1}, I_{\perp,s_2}) \vdash^* \perp \times t_1.s_2$ using rule (15).

3. We finally prove all statements of the form "$C^R \vdash^* c$ implies that there are some s and t s.t. $s \xrightarrow{c} t$" (and then $R \vdash^* s \times t$ has already been proven). This is done by induction on the derivation of $C^R \vdash^* c$, by case analysis of the 37 rules. We only mention one typical case:

rule (37): we have $C^{Q'}_{X,Y} \vdash^* c$ from $c = c_1 \oplus c_2$ where c_1 appears in a rule $Y \xrightarrow{c_1} s$ from Δ and $C^{Q'}_{X,s} \vdash^* c_2$. By ind. hyp. there is a t s.t. $s \xrightarrow{c_2} t$ and t is terminated. Then we deduce that $Y \xrightarrow{c} t$ thanks to rule $\mathrm{R_C}$. □

5 Regular tree grammar with costs

Inspecting Fig. 3, one realizes that rules for trees and rules for costs are different enough to prevent recognizability of the ternary relation defined by $s \xrightarrow{c} t$ in $\mathcal{T} \times T_M \times \mathcal{T}$. For instance, rules (25) combine an ε-rule for trees with a normal rule for costs, thereby allowing a structural difference between PA-terms and the associated costs. This slight difference is enough to cause the undecidability of transition logics with costs (see Prop. 8.1).

In order to better capture the relationship between grammar rules for PA-terms and grammar rules for the associated costs, we now introduce *tree grammars with costs*, a new device designed to deal with the ternary relation $s \xrightarrow{c} t$. These grammars aim at capturing simultaneously the derivation on terms and the derivation on costs in a synchronous way.

5.1 Basic definitions

Operation on terms. Positions are sequences of integers used to point inside terms: the subterm of t at position i is denoted by $t|i$, and the term obtained from t by replacing the subterm of t at position i by s is denoted $t[i \leftarrow s]$. For instance, if $t = X \parallel (Y.Z)$ then $t|2.1 = Y$ and $t[2.1 \leftarrow X.X] = X \parallel ((X.X).Z)$. For more details on these classical notions, the reader is referred to [13].

Costs. For tree grammar with costs, we assume that the interpretation domain for costs is $\mathbb{M} = \mathbb{N}^p$ for some p, i.e., each $c \in T_M$, is interpreted as some $[\![c]\!]$ in $\mathbb{N}^p$. Furthermore, and since we shall have to combine costs in various ways, we consider a more general notion of cost terms where we have a larger set of binary function symbols instead of only $\oplus$ and $\otimes$.

Formally, let p be some fixed positive integer, let $M = \{c_1, c_2, \ldots\}$ be a finite set of cost names. The set T_M of cost terms is given by the abstract syntax

$$c, c' ::= 0_M \ \mid\ c \odot_I c' \ \mid\ c_1 \ \mid\ c_2 \ \mid\ \cdots$$

where I is any subset of $\{1, \ldots, p\}$. These costs terms are interpreted in $\mathbb{M} = \mathbb{N}^p$ in the following way: $[\![0_M]\!] = \langle 0, \ldots, 0\rangle$, each constant name is interpreted by some constant tuple of $\mathbb{N}^p$, and the $\odot_I$ operations are interpreted as follows (we use the same notation $\odot_I$ for the interpretation): $\langle x_1, \ldots, x_p\rangle \odot_I \langle y_1, \ldots, y_p\rangle = \langle z_1, \ldots, z_p\rangle$ with, for $i = 1, \ldots, p$, $z_i = \mathbf{max}(x_i, y_i)$ if $i \in I$, and $z_i = x_i + y_i$ otherwise. Observe that all $\odot_I$ are associative-commutative and admit $\langle 0, \ldots, 0\rangle$ as neutral element. In this setting, we often write "+" instead of "$\odot_\emptyset$".

Grammars. A *regular tree grammar with costs* is a tuple $\mathcal{G} = \langle \mathcal{F}, \mathcal{Q}, \mathcal{C}, Q_{Ax}, \delta\rangle$ where $\mathcal{Q}$ is a set of non-terminals for trees, $\mathcal{C} = \{C_Q \mid Q \in \mathcal{Q}\}$ is a set of non-terminals for costs, $Q_{Ax} \in \mathcal{Q}$ is the axiom, and δ is a set of rules r of the form $Q \xrightarrow{c_r, \odot_{I_r}} f(Q_1, \ldots, Q_n)$ (when $f \in \mathcal{F}$ has arity $n > 0$), $Q \xrightarrow{c_r} f$ (when $f \in \mathcal{F}$ has arity $n = 0$), or $Q \xrightarrow{c_r, \odot_{I_r}} Q'$ (called ε-rules).

Example 5.1 *Let $\mathcal{F} = \{\|, ., X\}$, then $\mathcal{G} = \langle \mathcal{F}, \{Q, Q'\}, \{C_Q, C_{Q'}\}, Q, \delta\rangle$ with δ given by*

$$Q \xrightarrow{c,+} Q \parallel Q' \qquad\qquad Q \xrightarrow{0_M} X \qquad\qquad Q' \xrightarrow{c,+} Q$$

is a regular tree grammar with costs.

As appears in this example, the rules carry a cost term c and (sometimes) a cost operation $\odot_I$. The intuition is that applying such a rule incurs a given amount, namely c, to be combined using $\odot_I$ with the cost of the rest of the derivation.

The derivation relation. Before defining formally what are derivations for tree grammars with costs, we consider one example derivation, based on the grammar of Example 5.1:

$$(Q, C_Q) \vdash (Q \parallel Q', c + (C_Q + C_{Q'})) \vdash (Q \parallel Q, c + (C_Q + (c + C_Q)))$$

Observe that mixed terms and mixed cost terms are rewritten alongside, in pairs. Here, the first step has replaced Q by a parallel composition $Q \parallel Q'$ and the cost has been updated to $c + (C_Q + C_{Q'})$ (i.e., the cost c of parallelizing, plus the cost for the future derivations from Q plus the cost for the future derivations from Q'.

For the second step, we choose a non-terminal in $Q \parallel Q'$, say Q', that is derived into Q using the rule $Q' \xrightarrow{c,+} Q$. This means that the cost $C_{Q'}$ corresponding to Q' must be replaced by $c + C_Q$. In the result $(Q \parallel Q, c + (C_Q + (c + C_Q)))$ we have now two occurrences of Q and two occurrences of C_Q, but the first occurrence of Q is related to the first occurrence of C_Q and the second occurrence of Q is related to the second occurrence of C_Q. These dependencies must be remembered in further derivations, which makes the notion of derivation more complex.

We now define derivations more formally. Firstly, we extend the signature for terms with $\mathcal{Q}$ and the signature for cost terms with $\mathcal{C}$. Given a term t (resp. a cost term c), we define $Pos_{\mathcal{Q}}(t)$ (resp. $Pos_{\mathcal{C}}(c)$) to be the set of positions of occurrences of symbols of $\mathcal{Q}$ in t (resp. $\mathcal{C}$ in c). For instance $Pos_{\mathcal{Q}}(X \parallel (Q_1.Q_2)) = \{2.1, 2.2\}$.

Secondly, we consider triples (t, c, σ) where σ is a 1-to-1 mapping from $Pos_{\mathcal{Q}}(t)$ to $Pos_{\mathcal{C}}(c)$ such that $t|i = Q$ iff $c|\sigma(i) = C_Q$: in other words for each occurrence of the non-terminal Q in t, there is a corresponding unique occurrence of the non-terminal C_Q.

By definition $(t, c, \sigma) \vdash (t', c', \sigma')$ iff

- either there exists a position $i \in Pos_{\mathcal{Q}}(t)$ and a rule $Q \xrightarrow{c_r, \odot_{I_r}} f(Q_1, \ldots, Q_n)$ s.t.
$$\begin{cases} t|i = Q \ and \ t' = t[i \leftarrow f(Q_1, \ldots, Q_n)] \\ c' = c[\sigma(i) \leftarrow c_r \odot_{I_r} (C_{Q_1} \odot_{I_r} \ldots \odot_{I_r} C_{Q_n})] \\ \sigma'(i.l) = \sigma(i).2.l \quad for \ l = 1, \ldots, n \\ \sigma'(j) = \sigma(j) \ otherwise \end{cases}$$
- or there exists a position $i \in Pos_{\mathcal{Q}}(t)$ and a rule $Q \xrightarrow{c_r} f$ s.t.
$$\begin{cases} t|i = Q \ and \ t' = t[i \leftarrow f] \\ c' = c[\sigma(i) \leftarrow c_r] \\ \sigma'(j) = \sigma(j) \ for \ j \neq i \end{cases}$$
- or there exists a position $i \in Pos_{\mathcal{Q}}(t)$ and a rule $Q \xrightarrow{c_r, \odot_{I_r}} Q'$ s.t.

$$\begin{cases} t|i = Q \text{ and } t' = t[i \leftarrow Q'] \\ c' = c[\sigma(i) \leftarrow c_r \odot_{I_r} C_{Q'}] \\ \sigma'(i) = \sigma(i).2 \\ \sigma'(j) = \sigma(j) \quad \text{otherwise} \end{cases}$$

For simplicity, we usually drop the third component which is merely a technical way to relate occurrences of Q and C_Q and we simply write $(t, c) \vdash (t', c')$. Given a sequence $(Q, C_Q) \vdash (t_1, c_1) \vdash \ldots \vdash (t_n, c_n)$, we write $(Q, C_Q) \vdash^* (t_n, c_n)$ and $(t_n, c_n) \in L(Q)$. The language generated by $\mathcal{G}$ is $L(\mathcal{G}) = L(Q_{Ax})$.

Since we are interested in interpreted costs, we define $\mathcal{L}(Q) \stackrel{\text{def}}{=} \{(t, [\![c]\!]) \mid (t, c) \in L(Q)\}$ and $\mathcal{L}_{\mathcal{C}}(Q) \stackrel{\text{def}}{=} \{[\![c]\!] \mid (t, c) \in L(Q)\}$. We let $\mathcal{L}(\mathcal{G}) \stackrel{\text{def}}{=} \mathcal{L}(Q_{Ax})$.

Example 5.2 *Let $\mathcal{G}$ be as in example 5.1, let $+$ be the cost operation associated to $\|$. A sequence of derivation is*

$$\begin{aligned} (Q, C_Q) &\vdash (Q \parallel Q', c + (C_Q + C_{Q'})) \vdash (Q \parallel Q, c + (C_Q + (c + C_Q))) \\ &\vdash (X \parallel Q, c + (0_M + (c + C_Q))) \vdash (X \parallel X, c + (0_M + (c + 0_M))) \end{aligned}$$

Therefore $(Q, C_Q) \vdash^* (X \parallel X, c + (0_M + (c + 0_M)))$ and $(X \parallel X, \boldsymbol{d}) \in \mathcal{L}(Q)$ where $\boldsymbol{d} = [\![c + (0_M + (c + 0_M))]\!] = [\![c]\!] + [\![c]\!] \in \mathbb{M}$.

Let $\mathcal{G} = \langle \mathcal{Q}, \mathcal{C}, Q_{Ax}, \delta \rangle$ be a regular tree grammar with costs. $\mathcal{G}$ can be seen as the blending of a tree grammar and a cost grammar, and it is easy to extract these components:

- The *regular tree grammar induced by $\mathcal{G}$* is $\mathcal{G}_R = \langle \mathcal{F}, \mathcal{Q}, Q_{Ax}, \delta_R \rangle$ where δ_R is the set of rules $Q \longrightarrow f(Q_1, \ldots, Q_n)$ for $Q \xrightarrow{c, \odot_I} f(Q_1, \ldots, Q_n) \in \delta$, $Q \longrightarrow f$ for $Q \xrightarrow{c} f \in \delta$, and rules $Q \longrightarrow Q'$ for $Q \xrightarrow{c, \odot_I} Q' \in \delta$.
- The *regular cost grammar $\mathcal{G}_C$ induced by $\mathcal{G}$* has $\mathcal{C}_Q = \{C_Q \mid Q \in \mathcal{Q}\}$ as set of non-terminals and the rules $C_Q \longrightarrow c \odot_I (C_{Q_1} \odot_I \ldots \odot_I C_{Q_n})$ for $Q \xrightarrow{c, \odot_I} f(Q_1, \ldots, Q_n) \in \delta$, $C_Q \longrightarrow c$ for $Q \xrightarrow{c} f \in \delta$ and the rules $C_Q \longrightarrow c \odot_I C_{Q'}$ for $Q \xrightarrow{c, \odot_I} Q' \in \delta$.

The link between these grammars is established by the next proposition:

Proposition 5.3 *If $(t, c) \in L(\mathcal{G})$ then $t \in L(\mathcal{G}_R)$ and $c \in L(\mathcal{G}_C)$. Conversely, for any $t \in L(\mathcal{G}_R)$ there is some $c \in L(\mathcal{G}_C)$ such that $(t, c) \in L(\mathcal{G})$, and for any $c \in L(\mathcal{G}_C)$ there is some $t \in L(\mathcal{G}_R)$ such that $(t, c) \in L(\mathcal{G})$.*

PROOF. ($\Rightarrow$:) By construction $(Q, C_Q) \vdash^* (t, c)$ implies that $t \in L_{\mathcal{G}_R}(Q)$ and $c \in L_{\mathcal{G}_C}(C_Q)$.

($\Leftarrow$:) We prove it for costs and leave the proof for terms to the reader. Let $C_Q \vdash^* \bar{c}$ in the grammar $\mathcal{G}_C$. We prove the result by induction on the length of the derivation.

- Base case: the derivation uses a rule $C_Q \longrightarrow c$ (c a constant cost, possibly 0_M). By definition of $\mathcal{G}_C$ and $\mathcal{G}_R$ there exists a rule $Q \xrightarrow{c} f \in \mathcal{G}$ and a rule $Q \longrightarrow f \in \mathcal{G}_R$. This yields a derivation $(Q, C_Q) \vdash (f, c)$ and a derivation $Q \vdash f$.
- For the induction step, assume the derivation has the form $C_Q \vdash c \odot_I C_{Q'} \vdash^* \bar{c}$, using a rule $C_Q \longrightarrow c \odot_I C_{Q'}$ in $\mathcal{G}_C$. Then $\bar{c}$ is some $c \odot_I c'$ and $C_{Q'} \vdash^* c'$. By definition of $\mathcal{G}_C$ there exists a rule $Q \xrightarrow{c, \odot_I} Q'$ in $\mathcal{G}$. By induction hypothesis, $(Q', C_{Q'}) \vdash^* (t, c')$ for some t. Therefore $(Q, C_Q) \vdash^* (t, c \odot_I c')$, i.e., $(Q, C_Q) \vdash^* (t, \bar{c})$.

  Otherwise the derivation has the form $C_Q \vdash c \odot_I (C_{Q_1} \odot_I \ldots \odot_I C_{Q_n}) \vdash^* \bar{c}$ for a rule $Q \longrightarrow c \odot_I C_{Q_1} \odot_I \ldots \odot_I C_{Q_n}$ in $\mathcal{G}_C$. Then $C_{Q_i} \vdash^* c_i$ for $i = 1, \ldots, c_n$ and $\bar{c} = c \odot_I (c_1 \odot_I \ldots \odot_I c_n)$. By definition there exists a rule $Q \xrightarrow{c, \odot_I} f(Q_1, \ldots, Q_n)$ in $\mathcal{G}$. By induction hypothesis, $(Q_i, C_{Q_i}) \vdash^* (t_i, c_i)$ for some t_i. Therefore $(Q, C_Q) \vdash^* (f(t_1, \ldots, t_n), c \odot_I (c_1 \odot_I \ldots \odot_I c_n))$, i.e., $(Q, C_Q) \vdash^* (t, \bar{c})$. □

In general the sets $\mathcal{L}_\mathcal{C}(Q)$ can be quite arbitrary and are usually not decidable subsets of $\mathbb{N}^p$. However, with Parikh costs (only use the + operation), these sets are semilinear sets [3] according to the next proposition:

Proposition 5.4 *Let $\mathcal{G} = (\mathcal{F}, \mathcal{Q}, \mathcal{C}, Q_{Ax}, \delta)$ be a regular tree grammar with Parikh costs. Then the set $\mathcal{L}_\mathcal{C}(Q)$ is an effectively computable semilinear set of $\mathbb{N}^p$ for any $Q \in \mathcal{Q}$.*

PROOF (Idea). The $\mathcal{L}_\mathcal{C}(Q)$'s are the Parikh images of a context-free language for which a grammar can be read out of the cost rules, and hence is an effectively computable semilinear set of $\mathbb{N}^p$ [33]. □

5.2 Operations on tree grammars with costs

We shall later need *closure* results, stating that sets defined by tree grammars with costs are closed under product, conjunction and projection. These results are proved by exhibiting the constructions on tree grammars with costs that realize these operations. In this subsection we assume $\mathcal{G}$ and $\mathcal{G}'$ are two tree

[3] A subset $L \subseteq \mathbb{N}^p$ is *linear* if it is has the form $L = \{\mathbf{c}_0 + a_1\mathbf{c}_1 + \cdots + a_l\mathbf{c}_l \mid a_1, \ldots, a_l \in \mathbb{N}\}$ for a *base* $\mathbf{c}_0 \in \mathbb{N}^p$ and some *periods* $\mathbf{c}_1, \ldots, \mathbf{c}_l \in \mathbb{N}^p$. It is *semilinear* if it is a finite union of linear sets. Semilinear subsets of $\mathbb{N}^p$ are exactly the sets definable in Presburger arithmetic. See [18] for more details.

grammars with costs recognizing, respectively, $\mathcal{F}$-terms with costs in $\mathbb{M} = \mathbb{N}^p$, and $\mathcal{F}'$-terms with costs in $\mathbb{L} = \mathbb{N}^q$.

Product. We show how to construct a product grammar $\mathcal{G} \times \mathcal{G}'$ that recognizes terms built over the signature $\mathcal{F} \times \mathcal{F}'$, with costs interpreted in $\mathbb{M} \times \mathbb{L} = \mathbb{N}^{p+q}$.

Firstly, we define products of cost terms and we show that their interpretations fits our previous definition. Let T_M be a set of cost terms interpreted in $\mathbb{M} = \mathbb{N}^p$, let T_L be a set of cost terms interpreted in $\mathbb{L} = \mathbb{N}^q$. The set $T_{M\times L}$ is the set of terms build from $\langle 0_M, 0_L\rangle$, $\langle c, 0_L\rangle$ for $c \in M$, $\langle 0_M, d\rangle$ for $d \in L$, and functions $\odot_{i,j}$ for all pairs $\odot_i$ of T_M and $\odot_j$ of T_L. The interpretation of these costs is done in $\mathbb{N}^{p+q}$ with the obvious meaning for the constants, and $\odot_{i,j}$ is interpreted as

$$\langle c, d\rangle \odot_{i,j} \langle c', d'\rangle \stackrel{\text{def}}{=} \langle c \odot_i c', d \odot'_j d'\rangle.$$

By construction an interpretation $\langle c, d\rangle$ is a tuple such that the first p components are the interpretation of a cost term of T_M and the last q components are the interpretation of a cost term of T_L.

Secondly, we define product of terms (not cost terms) as in Section 4 on the product signature

$$\mathcal{F} \times \mathcal{F}' \stackrel{\text{def}}{=} (\mathcal{F} \cup \{\perp\}) \times (\mathcal{F}' \cup \{\perp\}) - \{\perp\perp\}$$

The non-terminals of the product grammar $\mathcal{G} \times \mathcal{G}'$ are all (Q, Q') with $Q \in \mathcal{Q} \cup \{\perp\}$ and $Q' \in \mathcal{Q}' \cup \{\perp\}$. We usually write QQ' instead of (Q, Q'). The axiom is $Q_{Ax}Q'_{Ax}$. The rules of $\mathcal{G} \times \mathcal{G}'$ are defined as follows:

- For every rule $r = Q \xrightarrow{c,\odot_i} f(Q_1, \ldots, Q_n)$ in δ and $r' = Q' \xrightarrow{c',\odot_j} g(Q'_1, \ldots, Q'_m)$ in δ' s.t. $m \geq n$, $\mathcal{G} \times \mathcal{G}'$ has a rule
$$rr' = QQ' \xrightarrow{\langle c,c'\rangle\odot_{i,j}} fg(Q_1Q'_1, \ldots, Q_nQ'_n, \perp Q'_{n+1}, \ldots, \perp Q'_m)$$
- We have similar rules when $m < n$.
- For every rule $r = Q \xrightarrow{c,\odot_i} f(Q_1, \ldots, Q_n)$ in δ and $r' = Q' \xrightarrow{c'} g$ in δ', $\mathcal{G} \times \mathcal{G}'$ has a rule
$$rr' = QQ' \xrightarrow{\langle c,c'\rangle\odot_{i,\emptyset}} fg(Q_1Q_\perp, \ldots, Q_nQ_\perp)$$
- We have similar rules for the symmetric case.
- For every rule $r = Q \xrightarrow{c,\odot_i} f(Q_1, \ldots, Q_n)$ in δ, $\mathcal{G} \times \mathcal{G}'$ has a rule[4].
$$r\perp = Q\perp \xrightarrow{\langle c,0_L\rangle\odot_{i,\emptyset}} f\perp(Q_1\perp, \ldots, Q_n\perp)$$

[4] We use $\odot_{i,\emptyset}$ but any other $\odot_{i,j}$ is possible since 0_L is neutral for all $\odot'_j$'s.

- We have similar rules from $r' \in \delta'$.
- For every rule $r = Q \xrightarrow{c} f$ in δ, $\mathcal{G} \times \mathcal{G}'$ has a rule

$$r\bot = Q\bot \xrightarrow{\langle c, 0_L \rangle} f\bot$$

- We have similar rules from $r' \in \delta'$.
- For every ε-rule $r = Q_1 \xrightarrow{c, \odot_i} Q_2$ in δ, and every non-terminal $Q' \in \mathcal{Q}'$, $\mathcal{G} \times \mathcal{G}'$ has the ε-rules

$$rQ' = Q_1 Q' \xrightarrow{\langle c, 0_L \rangle \odot_{i,1}} Q_2 Q' \text{ and } r\bot = Q_1 \bot \xrightarrow{\langle c, 0_L \rangle \odot_{i,1}} Q_2 \bot$$

- We have similar rules from ε-rules in δ'.

All this is made to ensure that:

Proposition 5.5 *Let $s \in T_{\mathcal{F}}$ and $t \in T_{\mathcal{F}'}$, let $c \in \mathbb{M}$, $c' \in \mathbb{L}$. Then $(s \times t, \langle c, c' \rangle) \in \mathcal{L}(QQ')$ iff $(s, c) \in \mathcal{L}(Q)$ and $(t, c') \in \mathcal{L}(Q')$.*

The proof is omitted and is similar to the proof for product of regular tree grammars (without cost). It uses two additional properties:

- The interpretation of any null term of T_M (resp. T_L) is the tuple $\langle 0, \ldots, 0 \rangle$ in $\mathbb{M}$ (resp. $\mathbb{L}$) which is the neutral element of all operations of $\mathbb{M}$ involved.
- Derivations involving $\bot$ on one component have a null cost on this component by definition of the rules.

The reader may notice that the proposition is false for uninterpreted costs (because extraneous null cost terms occur).

The same construction can be used in a slightly more general framework, where the product $\mathcal{G} \times \mathcal{G}'$ of a grammar $\mathcal{G}$ generating an x-ary relation (instead of only a ternary relation) with a grammar $\mathcal{G}'$ generating an y-ary relation gives an grammar generating an $(x + y)$-ary relation.

Similarly, the product of a grammar with costs over $\mathbb{M}$ with a normal regular tree grammar (without costs) $\mathcal{G}'$ gives a regular tree grammar with costs $\mathcal{G} \times \mathcal{G}'$ with costs over $\mathbb{M}$ since $\mathcal{G}'$ can be seen as a tree grammar with a trivial cost set.

Projection. Given a product $\mathcal{G} = \mathcal{G}_1 \times \ldots \times \mathcal{G}_n$ of grammars with costs generating the language $\mathcal{L}(\mathcal{G})$, the i-th projection of this language is $\{(t_1 \times \ldots \times t_{i-1} \times t_{i+1} \times \ldots \times t_n, \langle c_1, \ldots, c_{i-1}, c_{i+1}, \ldots, c_n \rangle) \mid \exists t_i, c_i \ s.t.(t_1 \times \ldots \times t_n, \langle c_1, \ldots, c_n \rangle) \in \mathcal{L}(\mathcal{G})\}$. This language can be generated by the regular tree grammar with costs obtained by erasing components i of symbols $f_1 f_2 \ldots f_n$ and symbols $\odot_{j_1 j_2 \ldots j_n}$ in the grammars rules of $\mathcal{G}$. The proof that this grammar generates indeed the i-th projection of $\mathcal{G}$ is similar to the proof for regular tree languages and is omitted.

Conjunction. The conjunction of $\mathcal{G}$ (with costs in $\mathbb{M}$) and $\mathcal{G}'$ (with costs in $\mathbb{L}$) is obtained by computing $\mathcal{G} \times \mathcal{G}'$, discarding all rules with symbols fg for $f \neq g$ and replacing symbols ff by f. The resulting grammar is a grammar $\mathcal{G}_\wedge$ for terms of $T_\mathcal{F}$ and costs in $\mathbb{M} \times \mathbb{L}$. We have that $(t, \langle c, c' \rangle) \in \mathcal{L}(\mathcal{G}_\wedge)$ iff $(t, c) \in \mathcal{L}(\mathcal{G})$ and $(t, c') \in \mathcal{L}(\mathcal{G}')$. The proof of this equivalence is similar to the correctness proof done for the construction computing the intersection of two regular tree languages. The main change lies in the treatment of costs since we keep track of costs related to each grammar.

Concluding remark. The cost grammars given in Fig. 3 for $\mathcal{T}$ use rules of the form $C \longrightarrow C' \oplus C''$ that yield grammars with cost that have rules of the form $Q \xrightarrow{\odot_I} f(Q_1, \ldots, Q_n)$ or $Q \xrightarrow{\odot_I} Q'$. Such rules don't fit our framework since there is no cost term c involved. We could easily adapt our definition to allow this rules but this leads to a lot of additional cases in the product construction and, in some case, we need to introduce some extraneous 0_M's terms.

Since we use tree grammar with cost with interpreted costs in mind, we shall merely replace such rules by rules $Q \xrightarrow{0_M \odot_I} f(Q_1, \ldots, Q_n)$ or $Q \xrightarrow{0_M \odot_I} Q'$. Since 0_M is neutral for all operations, this preserves the set $\mathcal{L}(\mathcal{G})$.

6 *TL*, the first-order transition logic

The results of Sections 4 and 5 have applications to the decision of process logics over PA.

Assume Δ is fixed. The *(first-order) transition logic TL* is the first-order logic of the structure $\langle \mathcal{T}; =, \xrightarrow{*}, P_1, \ldots \rangle$ where the binary predicates $=$ (equality) and $\xrightarrow{*}$ (reachability) have the obvious interpretation, and where $P_1, P_2, \ldots$ are any additional predicates provided they are *recognizable* (i.e., their interpretation is a recognizable relation over $\mathcal{T}$: in particular, membership predicates "$\in L$" with L a regular tree language are recognizable predicates). Since in our PA framework $\xrightarrow{*}$ and $=$ are recognizable predicates, the transition logic is "just" a first-order logic of trees with recognizable predicates.

We use $u, v, \ldots$ to denote variables, and $s, t, \ldots$ to denote trees in $\mathcal{T}$. The satisfaction relation $t_1, \ldots, t_n \models \varphi(u_1, \ldots, u_n)$ is defined as usual in first-order logic. Observe that the relation $\models$ depends on the underlying PA declaration Δ (since $\xrightarrow{*}$ does).

6.1 The difference between TL and temporal logics

Because quantifiers can be used freely, and because equality and other predicates are available, transition logics are more expressive than the modal logic EF handled in [30,27].

The ability to refer to states is the specific feature of transition logics: these logics can distinguish between otherwise bisimilar processes. For instance, it is possible to state the confluence of $\xrightarrow{*}$ through the *TL* formula

$$\forall u, v, v' \Big[\big(u \xrightarrow{*} v \wedge u \xrightarrow{*} v' \big) \Rightarrow \exists v'' \big(v \xrightarrow{*} v'' \wedge v' \xrightarrow{*} v'' \big) \Big] \qquad \text{(Confl)}$$

Temporal logics do not have such 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. This is not possible with transition logics: writing $u \xrightarrow{*} v$, one states that u may go to v via some path, but one cannot isolate this path and refer to it again. Additionally, temporal modalities are recursive by nature, while transition logics only have some built-in fixed points like $\xrightarrow{*}$. As a consequence, simple temporal modalities like EF can be expressed in the transition logic but more complex constructions like $\mathsf{E\_U}$ cannot.

In our PA framework, *TL* can deal with several simultaneous PA declarations (since any $\xrightarrow{*}_\Delta$ is a recognizable relation). E.g., one can states that all the terms reachable from X via $\Delta = \Delta_1 \cup \Delta_2$ are also reachable using rules from Δ_1 first, followed by rules from Δ_2. One uses the following formula:

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

TL does not explicitly allow using constants like "$X \parallel 0$", or terms with process variables like "$u \parallel u$". However, such terms could be allowed at no extra cost since they can be eliminated via simple transformations based on recognizable predicates encoding the function symbols. For instance, the following formula

$$\exists u (u \parallel u \xrightarrow{*} X \parallel 0)$$

is rewritten into

$$\exists u\, \exists v, v' (P_\parallel(v) \wedge P_l(u, v) \wedge P_r(u, v) \wedge P_{X\parallel 0}(v') \wedge v \xrightarrow{*} v')$$

where $P_\|$ is a (recognizable) unary predicate stating that its argument is a term with "$\|$" as its root, P_l and P_r are (recognizable) binary predicates stating that their first argument is the left-hand side (resp. right-hand) of its second argument, and where $P_{X\|0}$ states that its argument is $X \parallel 0$. Observe that this encoding only requires a finite number of predefined predicates: $P_{X\|0}$ can be defined by combining $P_\|$, P_X, P_0 with P_l and P_r.

6.2 Solving TL formulas

Theorem 4.2 immediately gives decidability of *TL*, or more precisely:

Corollary 6.1 *There is an effective procedure that, given Δ and a TL formula $\varphi(u_1, \ldots, u_n)$, computes a regular tree grammar that generates $Sol(\varphi)$.*

Being able to compute $Sol(\varphi)$ is more general than deciding validity or satisfiability of φ, or than model checking (telling whether $t \models \varphi(u)$ for a given t and φ). In particular, this allows the verification of parameterized systems, i.e., verifying that all instances of a parameterized system satisfy a given property. In our framework, this assumes that the set of instances is a regular language.

Example 6.2 *Let's write $t^{\|n}$ for $\overbrace{t \parallel (t \parallel (t \cdots \parallel t) \ldots))}^{n \text{ copies of } t}$ where $t \in \mathcal{T}$ and $n \in \mathbb{N}$. The set $L \stackrel{def}{=} \{t^{\|n} \mid n = 0, 1, 2, \ldots\}$ is regular and one checks that $t^{\|n} \models \varphi(u)$ for every n by checking $L \subseteq Sol(\varphi)$. What is more, the set of all n s.t. $t^{\|n} \models \varphi$ can be computed simply by building the grammar for $L \cap Sol(\varphi)$.*

7 *DTL* and decomposable constraints

In this section we extend *TL* with "decomposable" predicates that can express properties of the cost c in steps $s \xrightarrow{c} t$.

7.1 Decomposable cost predicates

A *cost predicate* P is a unary predicate over cost terms, or equivalently a subset of T_M. We write $P(c)$ when P holds for c.

Decomposable cost predicates generalize the notion of "decomposable regular languages" we introduced in [27] (see also [35,29,19]).

Definition 7.1 *A 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 of predicates* $\{P_i^1, P_i^2 \in \mathcal{DP} \mid i \in I\}$ *such that for all* $c, c' \in T_M$, $P(c \oplus c')$ *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 of predicates* $\{P_i^1, P_i^2 \in \mathcal{DP} \mid i \in I\}$ *such that for all* $c, c' \in T_M$, $P(c \otimes c')$ *iff* $\bigvee_{i\in I} P_i^1(c) \wedge P_i^2(c')$.

unit-decompositions: *for all* $P \in \mathcal{DP}$ *and all cost terms* c *appearing in* Δ, *there is a finite family of predicates* $\{P_i^c \in \mathcal{DP} \mid i \in I\}$ *such that for all* $c' \in T_M$, $P(c \oplus c')$ *iff* $\bigvee_{i\in I} P_i^c(c')$.

A predicate P is *decomposable* if it belongs to a *finite* decomposable family.

Example 7.2 (Counting constraints) *With Parikh costs (see Example 2.7)* $c \in T_M$ *denotes a* p*-tuple of integers* $[\![c]\!] = \langle x_1, \dots, x_p\rangle \in \mathbb{N}^p$ *and we have* $[\![c_1 \oplus c_2]\!] = [\![c_1]\!] + [\![c_2]\!]$ *(and a similar property for* $\otimes$*). Useful decomposable predicates for this cost set are, for example, all Boolean combinations of the basic predicates* $x_i = k$ *and* $x_i > k$ *(for* $k \in \mathbb{N}$*), and* $x_i \equiv k\ (m)$ *(congruence modulo some integer* m*).*

Example 7.3 (Timing constraints) *With costs for timing (cf. Example 2.8)* $c \in T_M$ *denotes some duration* $\boldsymbol{c}$ *in a time domain* $\mathbb{T}$. *One obtains a decomposable family of costs predicates by fixing a maximal duration* $K \in \mathbb{T}$ *beyond which precise values are not important. Let* $D \subseteq \mathbb{T}$ *be the set of costs that appear in* Δ *and write* Ω *for the set* $\{\alpha_1, \alpha_2, \dots\}$ *of all linear combinations (with coefficients from* $\mathbb{N}$*) of costs from* D *such that* $\alpha_i \leq K$*: then* Ω *is finite and the predicates "*$c = \alpha$*", "*$c < \alpha$*" and "*$c > \alpha$*" for* $\alpha \in \Omega$*, form a decomposable family in a straightforward way.*

7.2 The decomposable transition logic DTL

DTL ("Decomposable" Transition Logic) is the first-order logic that extends *TL* by allowing all binary predicates of the form "$\xrightarrow{\exists c\ P(c)}$", where P is any decomposable cost predicate.

$u \xrightarrow{\exists c\ P(c)} v$ is short for "$\exists c \big(u \xrightarrow{c} v \wedge P(c)\big)$" and holds iff there is some derivation $u \xrightarrow{c} v$ with $P(c)$. Observe that *DTL* is only a fragment of a first-order logic with two sorts and a ternary $\rightarrow$ predicate, where cost variables cannot be freely used. They are quantified upon whenever they are introduced (as with the *freeze quantification* of [2]). This explains why we often shortly write $u \xrightarrow{\exists P} v$ for $u \xrightarrow{\exists c\ P(c)} v$.

The negation of an atom $u \xrightarrow{\exists P} v$ states that for any path $u \xrightarrow{c} v$ between u and v, $P(c)$ is false. This will be later written $\forall c\ (u \xrightarrow{c} v \Rightarrow \neg P(c))$.

Proposition 7.4 *For any decomposable predicate P, the relation $\xrightarrow{\exists P}$ is recognizable.*

PROOF (Sketch). The construction of the required grammar (called $\mathcal{C}'$) is just an elaboration of the grammar $\mathcal{C}$ for $\xrightarrow{*}$ (Prop. 4.4 and Fig. 3).

Assume $\mathcal{DP}$ is a decomposable family. The non-terminals of $\mathcal{C}'$ are the non-terminals of $\mathcal{C}$ decorated with a predicate $P \in \mathcal{DP}$ (except that we keep I, $I_{\perp,s}$ and $I_{X,s}$ without any decoration). The intention is that R_P recognize all products $t \times t'$ s.t. $t \xrightarrow{\exists P} t'$.

The rules for $\mathcal{C}'$ are similar to the rules for $\mathcal{C}$. For example, the rules for R

$$R \longrightarrow \|\|\ (R,R) \mid ..(R,I) \mid ..(R',R) \mid 00 \mid Q_{X,X} \mid Q_{Y,Y} \mid \ \ldots$$

found in (δ_4) are replaced by all rules of the following form, where P is any predicate from $\mathcal{DP}$:

$$\left.\begin{array}{rcl} R\_P & \longrightarrow & \|\|\ (R\_P_i, R\_P'_i) \\ R'\_P & \longrightarrow & \|\|\ (R'\_P_i, R'\_P'_i) \end{array}\right\} \text{for all } (P_i, P'_i) \text{ in the par-decomposition of } P,$$

$$\left.\begin{array}{rcl} R\_P & \longrightarrow & ..(R\_P, I) \\ R\_P & \longrightarrow & ..(R'\_P_i, R\_P'_i) \\ R'\_P & \longrightarrow & ..(R'\_P_i, R'\_P'_i) \end{array}\right\} \text{for all } (P_i, P'_i) \text{ in the seq-decomposition of } P,$$

$$\left.\begin{array}{rcl} R\_P & \longrightarrow & 00 \\ R'\_P & \longrightarrow & 00 \end{array}\right\} \text{when } P(0),$$

$$\left.\begin{array}{rcl} R\_P & \longrightarrow & Q_{Y,Y}\_P \\ R'\_P & \longrightarrow & Q'_{Y,Y}\_P \end{array}\right\} \text{for all } Y \in \mathit{Const}.$$

The rules for the $Q_{X,s}\_P$ etc. are built similarly, extending the corresponding rules in Fig. 3. For instance the rule $Q_{\perp,Y} \longrightarrow Q_{\perp,s}$, associated with a rule $Y \xrightarrow{c} s \in \Delta$, is replaced by all $Q_{\perp,Y}\_P \longrightarrow Q_{\perp,s}\_P_i^c$ where the P_i^c are obtained by unit-decomposition of P w.r.t. c.

With this we can prove a lemma similar to Prop. 4.6: for all $s, t \in \mathcal{T}$, $R\_P \vdash^* s \times t$

iff $s \xrightarrow{\exists P} t$. $\square$

The corollary is that DTL is decidable, or more precisely:

Theorem 7.5 *For any decomposable family of costs predicates, there is an effective procedure that, given Δ and a DTL formula $\varphi(u_1, \ldots, u_n)$, outputs a tree grammar generating $Sol(\varphi)$.*

7.3 The timed transition logic TTL

DTL and Theorem 7.5 can be instantiated in meaningful ways. In the rest of this section we consider a situation where costs denote some kind of durations.

More precisely, we adopt the framework of Example 2.8 and look at TTL (Timed Transition Logic), an instance of DTL based on the timing constraints from Example 7.3. Thus TTL extends TL by allowing all atoms $u \xrightarrow{\exists c\ \tau} v$ where τ is a *time constraint* built according to the following grammar:

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

where the C's can be any numerical constant from $\mathbb{T}$ (and where c is the free cost variable of τ). (Observe that $c < C$ is really a short way of writing $[\![c]\!] < C$.)

In TTL, one may write formulas expressing properties like *from any state reachable from an initial state in less than* 5 *time units, it is possible to reach a final state in less than* 10 *time units* as follows:

$$\forall s, t\ (s \in Initial \wedge s \xrightarrow{\exists c\ c \leq 5} t \Rightarrow \exists s' \ : \ s' \in Final \wedge t \xrightarrow{\exists c\ c \leq 10} s').$$

Then, because these time constraints are decomposable, we have

Proposition 7.6 *For any time constraint τ, the relation $s \xrightarrow{\exists c\ \tau} t$ is recognizable.*

We also have the following instantiation of Theorem 7.5:

Theorem 7.7 *The logic TTL is decidable.*

TTL can be enriched. Consider, for example, the binary predicate $u \xrightarrow{Unbounded} v$, meaning that there exists steps $u \xrightarrow{c} v$ with arbitrarily large costs c (i.e., going from u to v may take arbitrarily long time).

Lemma 7.8 *The relation $\xrightarrow{Unbounded}$ is recognizable.*

PROOF (Idea). $s \xrightarrow{\textit{Unbounded}} t$ iff it there is a derivation $s \xrightarrow{*} t$ that involves a loop $X \xrightarrow{c} X$ with $[\![c]\!] > 0$ (and $X \in \mathit{Const}$). Such loops are easy to compute and list, and it is then easy to adapt our automaton for $\xrightarrow{*}$ so that it keeps track of whether an opportunity for the loops has been encountered. $\square$

Example 7.9 (Two other "timed" logics for free) *Using the same time domain and the same time constraints, we may change the definition of $\otimes$ and interpret it as addition (like $\oplus$), thereby reducing parallelism to interleaving. Another variant is to exchange the roles of $\oplus$ and $\otimes$: then the costs measure the maximal degree of parallelism rather than the elapsed time. In both cases the constraints remain decomposable and we still have a decidable TTL.*

8 *TLC*, a parameterized transition logic with Parikh costs

Extending transition logics like we did in the previous section allows referring to the underlying costs in reachability predicates, but it only provides a limited way of stating properties of these costs. In this section, we consider adding a *first-order logic of costs* to transition logic. The resulting two-sorted logic, called *TLC*, is very expressive. In verification settings, it allows stating parameterized properties with parameters ranging over costs (and processes) rather than only processes as in section 6.2.

For *TLC*, we fix a precise notion of costs: we assume Parikh costs, as in Example 2.7. Here $\mathbb{M} = \mathbb{N}^p$ and the cost $[\![c]\!] = (n_1, \ldots, n_p)$ of a derivation $s \xrightarrow{*} t$ can be used to record the number of occurrences of each action of Act along the derivation. Since $[\![c]\!]$ is now a p-tuple of integers, we often write $x_1, \ldots, x_p$, or $\bar{x}$, instead of $[\![c]\!]$.

We also fix the cost constraints we shall allow: we use Presburger formulas over the components of costs. This allows to state properties such as "s reaches t using *as many actions a than actions b*".

Formally, *TLC* allows three kinds of atoms:

- all $R(u_1, \ldots, u_n)$ where R is a recognizable relation (and $u_1, \ldots, u_n$ are process variables),
- all $\psi(\bar{y})$ where ψ is a Presburger formula,
- 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 rest $\bar{y}$ (an arbitrary number of parameters).

$u \xrightarrow{\exists \bar{x} \psi(\bar{x}, \bar{y})} v$ is short for "$\exists \mathbf{c}, u \overset{\mathbf{c}}{\Rightarrow} v \wedge \psi(\mathbf{c}, \bar{y})$". Observe that only $u, v, \bar{y}$ are free in $u \xrightarrow{\exists\bar{x}\psi(\bar{x},\bar{y})} v$. In practice we omit writing the variables from $\bar{x}$ that are not used in ψ. 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.

Finally, *TLC* formulas are given by the abstract syntax:

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

Even with our restriction to Parikh costs and Presburger constraints, the full *TLC* is undecidable (see Prop. 8.1). Therefore we introduce two fragments that will be shown decidable (Theo. 8.2):

the parameterized existential fragment: which is the set of closed formulas that can be written under the form $(\exists|\forall\ \bar{y})^*(\exists\ u)^*[\vee \wedge Atoms]$,[5] and
the parameterized universal fragment: which is the set of closed formulas that can be written under the form $(\exists|\forall\ \bar{y})^*(\forall\ u)^*[\vee \wedge \neg Atoms]$.

Observe that the two fragments are dual: a formula in one fragment is equivalent to the negation of a formula in the other fragment.

Further, and since the complement of a recognizable relation is recognizable (similarly the negation of a Presburger formula is a Presburger formula), the restriction on the polarity of atoms only applies 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).

8.1 Expressing properties with TLC

We consider the PA example $\Delta_{\texttt{Weight}}$ that abstracts the divide-and-conquer `Weight` program from Fig. 1.

This system is decorated with Parikh costs by isolating the following actions: *sp*, *seq*, *sw* and *add*. *sp* is associated with rule r_1, and allows counting the number of times parallel coroutines are *sp*awned. Similarly *seq*, associated with rule r_7, allows counting recursive calls in the *seq*uential part. *sw* is associated with rule r_6, when computation *sw*itches from `W` to `W_seq`, and *add* is associated with rules r_5 and r_{11} where an *add*ition of subweights is performed. Formally, we consider $\mathbb{M} = \mathbb{N}^4$ and let rules in Δ carry costs of the

[5] That is, process variables may only be existential, and the corresponding quantifications must occur under the quantification over integer parameters. Negations of atoms are not allowed.

form $\mathbf{c}_{sp} = (1,0,0,0)$, $\mathbf{c}_{seq} = (0,1,0,0)$, $\mathbf{c}_{sw} = (0,0,1,0)$, $\mathbf{c}_{add} = (0,0,0,1)$, or $\bar{0} = (0,0,0,0)$.

TLC can now be used to state properties that refer to the number of times the given actions have been used. For example, the following formula

$$\forall u\, \neg\Big(X_{10} \xrightarrow{\exists \bar{x}. x_{sw}=0 \wedge x_{add}>0} u\Big)$$

states that "runs from X_{10} never do *add*'s without doing a *sw*". The formula

$$\forall u\, \neg\Big(X_{10} \xrightarrow{\exists \bar{x}. x_{sp}+x_{seq}<x_{add}} u\Big)$$

states that "runs from X_{10} never use less *sp*'s and *seq*'s than *add*'s". Finally, the formula

$$\forall u\, \Big(X_{10} \xrightarrow{\exists \bar{x}. x_{sp}+x_{seq}=x_{add}} u \;\Rightarrow\; u \in \mathit{Final}\Big)$$

further states that these figures coincide only when we reach "terminated" situations.

Observe that these three *TLC* formulas belong to the parameterized universal fragment and do not even use parameters.

An example illustrating the usefulness of parameters is:

$$\forall y_1, y_2 \forall u, v\, \Big[\Big(u \xrightarrow{\exists \bar{x}. y_1=x_{add}} v \wedge u \xrightarrow{\exists \bar{x}. y_2=x_{add}} v\Big) \Rightarrow\; y_1 = y_2\Big]$$

stating that all paths from a same u to a same v contain the same number of *add*'s. This last formula belongs to the parameterized universal fragment.

8.2 Decidability for TLC

We are interested in whether, given a PA declaration Δ, a given closed *TLC* formula is valid.

The problem is undecidable in general:

Proposition 8.1 (see Appendix B) *The problem whether, given some Δ, a closed TLC formula φ is valid, is undecidable, even when restricted to formulas φ of the fragment without parameters and with only $\exists\forall$ quantification for process variables.*

This result motivated the isolation of the parameterized fragments of *TLC*, for which we get the following decidability results:

Theorem 8.2 *The parameterized existential and the parameterized universal fragments of TLC are decidable.*

The proof of Theorem 8.2 relies on the classical result that semilinear sets are exactly the sets definable in Presburger arithmetic (see [18]). This is used to prove the next proposition for which we introduce the following definition. Let Φ be a parameterized existential TLC formula. Φ is some $(\exists|\forall\ \bar{y})^*(\exists\ u)^*\varphi$: we write $\phi(y_1,\ldots,y_k)$ for the "$(\exists\ u)^*\varphi$" part and let $Sol(\phi) = \{\langle n_1,\ldots,n_k\rangle \mid\ \models \phi(n_1,\ldots,n_k)\}$.

Theorem 8.3 *Let Φ be a parameterized existential TLC formula, then $Sol(\phi)$ is an effectively computable semilinear set.*

For proving this, we begin by linking the atomic predicates of TLC to tree grammars with costs: this is easy to do since the grammars we used in Prop. 4.4 already formed a cost grammar. To fit precisely our framework, we simply consider the cost rules given in Fig. 3 where we replace rules $C \longrightarrow C' \odot C''$ by rules $C \longrightarrow 0_M \oplus (C' \odot C'')$ and similarly for $\otimes$ (which doesn't change the set of interpreted costs, see the last remark of section 5). Then we merge the resulting grammar with the regular tree grammar given in Fig. 3 in order to obtain a regular tree grammar with costs $\mathcal{G}$ that generates $\stackrel{*}{\Rightarrow}$. The grammar has the following properties:

Lemma 8.4 *For any $s,t,u,v \in (\mathcal{T} \cup \{\bot\})$,*

(1) $(s \times t, \boldsymbol{c}) \in \mathcal{L}(R)$ *iff* $s \stackrel{\boldsymbol{c}}{\Rightarrow} t$,
(2) $(s \times t, \boldsymbol{c}) \in \mathcal{L}(R')$ *iff* $s \stackrel{\boldsymbol{c}}{\Rightarrow} t$ *and t is terminated,*
(3) $(s \times t, \boldsymbol{c}) \in \mathcal{L}(Q_{u,v})$ *iff* $s = u$ *and* $v \stackrel{\boldsymbol{c}}{\Rightarrow} t$,
(4) $(s \times t, \boldsymbol{c}) \in \mathcal{L}(Q'_{u,v})$ *iff* $s = u$ *and* $v \stackrel{\boldsymbol{c}}{\Rightarrow} t$ *and t is terminated.*

Furthermore,

(1) for each $\boldsymbol{c} \in \mathcal{L}_{\mathcal{C}}(R)$ (resp. $L_{\mathcal{C}}(R')$) there are some s and t and a derivation $s \stackrel{\boldsymbol{c}}{\Rightarrow} t$ s.t. $(s \times t, \boldsymbol{c}) \in \mathcal{L}(R)$ (resp. R'),
(2) for each $\boldsymbol{c} \in \mathcal{L}_{\mathcal{C}}(Q_{u,v})$ (resp. $\mathcal{L}_{\mathcal{C}}({Q'}_{u,v})$) there is some t and a derivation $v \stackrel{\boldsymbol{c}}{\Rightarrow} t$ s.t. $(u \times t, \boldsymbol{c}) \in \mathcal{L}(Q_{u,v})$ (resp. $\in \mathcal{L}(Q'_{u,v})$).

(The proof is just a rehash of the proof of Prop. 4.6, where the use of interpretation allows to get rid of the extraneous 0_M's).

This lemma must be handled cautiously since, as with Prop. 5.3, we cannot simultaneously choose a $s \times t \in L(R)$, a $\mathbf{c} \in \mathcal{L}_{\mathcal{C}}(R)$, and assume $s \stackrel{\mathbf{c}}{\Rightarrow} t$.

We now turn to the proof of Theorem 8.3. It goes through a succession of steps:

Input formula. Consider the *TLC* formula $\phi(y_1, \ldots, y_k)$. It has the form $(\exists\ u)^*\varphi$. For simplicity, we assume φ only contains reachability atoms (the proof is the same when $R(\bar{u})$ atoms for recognizable R's are allowed) and has no disjunctions (this is no loss of generality since ϕ is an existential formula). Hence φ is some $\exists\bar{u},\bar{v}\ \left(\bigwedge_{i=1}^n u_i \xrightarrow{\exists\bar{x}_i\psi_i(\bar{x}_i,\bar{y}_i)} v_i\right)$ where, after some renaming, we may assume that the $\bar{x}_i$'s are all disjoint.

Linearization. The linearization φ_L of φ is the formula computed from φ by renaming all term variables such that they occur only once in φ_L. We still write u_i, v_i for the renamed variables (and let u_i^r, v_i^r denote the original variables in φ).

The resulting φ_L is not equivalent to φ, but if we conjunct φ_L with Id_φ (a formula stating equality of the relevant components) and apply a projection to get rid of the extra components, we obtain a formula that is equivalent to φ.

For instance a formula $\varphi \equiv u \xrightarrow{\exists x\psi_1(x)} v \wedge v \xrightarrow{\exists x\psi_2(x)} u$ is linearized in $\varphi_L \equiv u_1 \xrightarrow{\exists x\psi_1(x)} v_1 \wedge u_2 \xrightarrow{\exists x\psi_2(x)} v_2$. The formula Id_φ is $u_2 = v_1 \wedge u_1 = v_2$ which defines a recognizable relation. Projecting the solutions of $\varphi_L \wedge Id_\varphi$ on the first and third component yields the solution of φ.

The associated tree grammar with costs. To each reachability atom $u_i \xrightarrow{\exists\bar{x}_i\psi_i(\bar{x}_i,\bar{y}_i)} v_i$ from φ_L, we associate a copy $\mathcal{G}_i$ of the grammar with costs from Lemma 8.4. Now let $\mathcal{G} = (\prod_{i=1}^n \mathcal{G}_i) \wedge \mathcal{G}_{Id_\varphi}$, where $\mathcal{G}_{Id_\varphi}$ is the grammar generating all tuples satisfying Id_φ (here $\wedge$ denotes the conjunction of the grammars defined in section 5, furthermore we discard the costs coming from $\mathcal{G}_{Id_\varphi}$). By construction this grammar accepts the solutions of $\left(\bigwedge_{i=1}^n u_i \xrightarrow{\exists\bar{x}_i\ true} v_i\right) \wedge Id_\varphi$ (a set of products $\bar{s} \times \bar{t}$, or $s_1 \times \cdots \times s_n \times t_1 \times \cdots \times t_n$) and the cost set $\mathcal{L}_\mathcal{C}(\mathcal{G})$ records all $\langle \bar{z}_1, \ldots, \bar{z}_n\rangle \in \mathbb{N}^p \times \cdots \times \mathbb{N}^p$ s.t. $s_i \stackrel{\bar{z}_i}{\Rightarrow} t_i$ for all i. The set $\mathcal{L}_\mathcal{C}(\mathcal{G})$ is an effectively computable semilinear set (by proposition 5.4).

Computation of the solutions of arithmetical constraints. Define

$$\varphi_{\mathcal{G},\mathrm{PA}}(\bar{y}) \stackrel{\text{def}}{=} \exists\bar{x}_1 \ldots \bar{x}_n \left(\langle \bar{x}_1, \ldots, \bar{x}_n\rangle \in \mathcal{L}_\mathcal{C}(\mathcal{G}) \wedge \bigwedge_{i=1}^n \psi_i(\bar{x}_i, \bar{y}_i)\right)$$

This is a Presburger formula that can be built effectively.

Lemma 8.5 $Sol(\phi) = \{\bar{z} \mid\ \models \varphi_{\mathrm{PA}}(\bar{z})\}$.

PROOF. ($\subseteq$:) Assume $\bar{z} \in Sol(\phi)$. Then there exist some $\bar{s}, \bar{t}$ s.t. $\varphi(\bar{z}, \bar{s}, \bar{t})$ holds, i.e., for any i there is a cost term $\bar{c}_i$ s.t. $s_i \xrightarrow{\bar{c}_i} t_i \wedge \psi_i(\bar{\mathbf{c}}_i, \bar{z}_i)$ (remember that $\bar{\mathbf{c}}_i$ denotes the interpretation of $\bar{c}_i$). By construction of $\mathcal{G}$, this means that $(\bar{s} \times \bar{t}, \bar{\mathbf{c}}) \in \mathcal{L}(\mathcal{G})$ for $\bar{\mathbf{c}} = \langle \bar{\mathbf{c}}_1, \ldots, \bar{\mathbf{c}}_n \rangle$, therefore $\bar{\mathbf{c}} \in \mathcal{L}_{\mathcal{C}}(\mathcal{G})$. Hence $\bar{\mathbf{c}}$ is a witness for $\exists \bar{x}_1 \ldots \bar{x}_n \left(\langle \bar{x}_1, \ldots, \bar{x}_n \rangle \in \mathcal{L}_{\mathcal{C}}(\mathcal{G}) \wedge \bigwedge_{i=1}^{n} \psi_i(\bar{x}_i, \bar{y}_i) \right)$, which proves that $\bar{z}$ satisfies φ_{PA}.
($\supseteq$:) Assume that $\varphi_{\mathrm{PA}}(\bar{z})$, i.e., $\varphi_{\mathcal{G},\mathrm{PA}}(\bar{z})$, holds. Then there exists some $\bar{\mathbf{c}} = (\bar{\mathbf{c}}_1, \ldots, \bar{\mathbf{c}}_n) \in \mathcal{L}_C(\mathcal{G})$ s.t. for any $i = 1, \ldots, n$, $\psi_i(\bar{\mathbf{c}}_i, \bar{z}_i)$ holds. Since $\bar{\mathbf{c}} \in \mathcal{L}_{\mathcal{C}}(\mathcal{G})$ there exist $\bar{s}, \bar{t}$ s.t. $(\bar{s} \times \bar{t}, \bar{\mathbf{c}}) \in \mathcal{L}(\mathcal{G})$. By definition of $\mathcal{G}$ as a product grammar, this implies that for all i, there exists a cost term $\bar{c}_i$ with interpretation $\bar{\mathbf{c}}_i$ s.t. $s_i \xrightarrow{\bar{c}_i} t_i \wedge \psi(\bar{\mathbf{c}}_i, \bar{z}_i)$. Thus $s_i \xrightarrow{\exists \bar{x}_i \psi_i(\bar{x}_i, \bar{z}_i)} t_i$. Finally $\phi(\bar{s}, \bar{t}, \bar{z})$ holds, hence $\bar{z} \in Sol(\phi)$. $\Box$

The results on the universal fragment is a direct consequence since the complement of a semilinear set is a semilinear set (see [18]).

9 *PTTL*, a parameterized timed transition logic

In this section we consider *PTTL*, a transition logic that combines parameters (as in section 8) with an interpretation of costs as durations in $\mathbb{N}$, as in *TTL* (from section 7.3). *PTTL* allows atoms like $s \xrightarrow{\exists c\ c > x+2} t$ where x represents some unknown duration.

The main difference between *PTTL* and *TLC* is that *PTTL* interprets costs as (integer) durations, instead of seeing them more abstractly as counting the number of occurrences of actions. As in Example 2.8, the duration of a sequential composition (resp. a parallel composition) of steps is the sum (resp. the maximum) of the durations of each component.

Formally, *PTTL* is the transition logic build from atoms $u \xrightarrow{\exists x \phi(x, \bar{y})} v$ where

- x is the cost of the derivation,
- $y_1, \ldots, y_n$ are parameters ranging over $\mathbb{N}$,
- $\phi(x, \bar{y})$ is a time constraint of the form $x \geq \sum_{i=1}^{m} a_i y_i$ with $a_i \in \mathbb{N}$,

and formulas of *PTTL* are given by the abstract syntax:

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

As before, it is possible to allow predicates given by arbitrary recognizable relations, for example all relations defined by a *TTL* formula (the parameter-free fragment of *PTTL*).

9.1 *Decidability for PTTL*

As in section 8, we show decidability for restricted fragments of *PTTL*:[6]

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

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

The following theorem is the counterpart for *PTTL* of Theorem 8.2 for *TLC*:

Theorem 9.1 *The parameterized existential (resp. universal) fragment of PTTL is decidable.*

The rest of the section is devoted to a proof of Theorem 9.1. Our approach follows the earlier proof for *TLC* (tree grammars with costs are associated with basic atoms, and then combined and/or transformed). There is a difference however: we are in a situation where we do not know how to compute the cost sets generated by cost grammars. Thus our method uses approximations of costs sets and relies on the fortunate fact that, for the fragments we consider, cost sets can be safely replaced by their approximations.

9.2 *Approximating the language generated by cost grammars*

We consider cost terms on a cost set M with operations $\odot_I$ for all $I \subseteq \{1,\ldots,p\}$. As in section 5, the interpretation domain is $\mathbb{M} = \mathbb{N}^p$ and the interpretation of $\odot_I$ is given by $\langle z_1,\ldots,z_p\rangle = \langle x_1,\ldots,x_p\rangle \odot_I \langle y_1,\ldots,y_p\rangle$ iff $z_i = \mathbf{max}(x_i, y_i)$ when $i \in I$ and $z_i = x_i + y_i$ otherwise.

For each $I \subseteq \{1,\ldots,p\}$, for each $\mathbf{c} = \langle x_1,\ldots,x_p\rangle \in \mathbb{N}^p$ the projection $\Pi_I\mathbf{c}$ is defined by $\Pi_I\langle x_1,\ldots,x_p\rangle \stackrel{\text{def}}{=} \langle y_1,\ldots,y_p\rangle$ where $y_i = 0$ when $i \in I$ and $y_i = x_i$ otherwise. For $L \subseteq \mathbb{N}^p$, $\Pi_I L \stackrel{\text{def}}{=} \{\Pi_I\mathbf{c} \mid \mathbf{c} \in L\}$.

We shall use another set of cost names $M' = \{\Pi_I c \mid I \subseteq \{1,\ldots,p\}, c \in M\}$ and the corresponding set $T_{M'}$. The interpretation is done in $\mathbb{M}' = \mathbb{N}^p$ by $[\![\Pi_I c]\!] = \Pi_I([\![c]\!])$, and the $\odot_I$ are interpreted as previously.

Let $\mathcal{G}$ be a cost grammar generating a subset $L(\mathcal{G}) \subseteq T_M$ yielding an interpretation $\mathcal{L}(\mathcal{G}) = \{[\![c]\!] \mid c \in L(\mathcal{G})\} \subseteq \mathbb{M}$. We show how to transform the cost grammar $\mathcal{G}$ into an *approximated* $\mathcal{G}_A$ that generates a larger (interpreted) cost

[6] However, contrary to the *TLC* case, the decidability of full *PTTL* is open.

set $\mathcal{L}(\mathcal{G}_A)$ but such that any cost $\mathbf{d} \in \mathcal{L}(\mathcal{G}_A)$ is less than or equal to a cost $\mathbf{c} \in \mathcal{L}(\mathcal{G})$.

Assume $\mathcal{G}$ has non-terminals $C_1, \ldots, C_n$ and rules of the form $C \longrightarrow C_1 \odot_J C_2$, or $C \longrightarrow C'$, or $C \longrightarrow c$. We denote by $L(C)$ the language of T_M generated by any non-terminal C and by $\mathcal{L}(C)$ its interpretation in $\mathbb{M}$. The approximation process proceeds in two steps. We first compute a new grammar $\mathcal{G}_\Pi$ from $\mathcal{G}$ that will generate $\Pi_I \mathcal{L}(C)$ for all $I \subseteq \{1, \ldots, p\}$ and non-terminal C of $\mathcal{G}$. The grammar $\mathcal{G}_\Pi$ is defined as follows:

- the non-terminals are all C^I for C a non-terminal of $\mathcal{G}$ and I a subset of $\{1, \ldots, p\}$,
- it contains the rules $C^I \longrightarrow C_1^I \odot_J C_2^I$ for $C \longrightarrow C_1 \odot_J C_2$ a rule of $\mathcal{G}$ and $I \subseteq \{1, \ldots, p\}$,
- similarly, it contains all rules $C^I \longrightarrow C'^I$ for $C \longrightarrow C'$ an (epsilon-)rule of $\mathcal{G}$ and $I \subseteq \{1, \ldots, p\}$,
- and all rules $C^I \longrightarrow \Pi_I c$ for $C \longrightarrow c$ a rule of $\mathcal{G}$ and $I \subseteq \{1, \ldots, p\}$.

We let the reader check that $\mathcal{G}$ is embedded in $\mathcal{G}_\Pi$ (when renaming C by $C^\emptyset$) and that $(\mathcal{G}_\Pi)_\Pi$ is embedded in $\mathcal{G}_\Pi$ (when renaming $(C^I)^J$ by $C^{I \cup J}$).

The following proposition is proved by a straightforward induction on derivation length:

Proposition 9.2 *For each non-terminal C of $\mathcal{G}$, and subset I of $\{1, \ldots, p\}$, the following properties hold:*
– For each derivation $C \vdash^* c$ in $\mathcal{G}$ (resp. $C^I \vdash^* \Pi_I c$ in $\mathcal{G}_\Pi$) there is a derivation $C^I \vdash^* \Pi_I c$ in $\mathcal{G}_\Pi$ (resp. $C \vdash^* c$ in $\mathcal{G}$) having the same length.
– $\mathcal{L}(C^I) = \Pi_I \mathcal{L}(C)$.

After dealing with projections, we now deal with the effective approximation step. Our goal is to transform a cost grammar $\mathcal{G}$ with operations $\odot_I$ that mix additions and maximums to a cost grammar $\mathcal{G}_A$ with Parikh costs. The set $\mathcal{L}(\mathcal{G}_A)$ is a superset of $\mathcal{L}(\mathcal{G})$ but the approximation is "safe" in that, for every cost $\mathbf{c}$ in $\mathcal{L}(\mathcal{G}_A) \setminus \mathcal{L}(\mathcal{G})$, we can find a $\mathbf{c}' \in \mathcal{L}(\mathcal{G})$ with $\mathbf{c} \leq \mathbf{c}'$.

The idea underlying the construction of $\mathcal{G}_A$ is to replace a rule $C \longrightarrow \mathbf{max}(C_1, C_2)$ by two rules $C \longrightarrow C_1$ and $C \longrightarrow C_2$ and see that the safety criterion is preserved through rules involving maxima or sums. However, while this simple scheme works in dimension 1, the general case has to deal with mixed operations $\odot_I$ that take maxima on positions $i \in I$ and sums on positions $i \notin I$.

We now describe a construction that generalizes the previous idea in higher dimensions. Let $\mathcal{G}_\Pi$ be constructed from $\mathcal{G}$ as seen above. If $X = C^I$ is a non-terminal of $\mathcal{G}_\Pi$, we denote by X^J the non-terminal $C^{I \cup J}$. $\mathcal{G}_A$ is defined by:

- its non-terminals are exactly the non-terminals of $\mathcal{G}_\Pi$,
- the rules $X \longrightarrow X_1 \odot_I X_2$ with $I \neq \emptyset$ in $\mathcal{G}_\Pi$ are replaced by all rules $X \longrightarrow X_1^J + X_2^K$ where $J \cup K = I$ and $J \cap K = \emptyset$ (here "+" denotes $\odot_\emptyset$).

By construction, $\mathcal{G}_A$ is a cost grammar with Parikh costs, therefore the language $\mathcal{L}(X)$ generated by any non-terminal X of $\mathcal{G}_A$ is a semilinear set.

The next two propositions relate $\mathcal{G}$ and $\mathcal{G}_A$.

Proposition 9.3 *$\mathcal{L}(C_{(\mathcal{G})}) \subseteq \mathcal{L}(C_{(\mathcal{G}_A)})$ for any non-terminal C of $\mathcal{G}$.*

PROOF. Actually we prove that $\mathcal{L}(C^I_{(\mathcal{G}_\Pi)}) \subseteq \mathcal{L}(C^I_{(\mathcal{G}_A)})$ for all C and I. The proof is by induction on the length of minimal derivations $C^I_{(\mathcal{G}_\Pi)} \vdash^* c$ for $[\![c]\!] \in \mathcal{L}(C^I_{(\mathcal{G}_\Pi)})$. Assume $\mathbf{c} \in \mathcal{L}(C^I_{(\mathcal{G}_\Pi)})$. By definition $\mathbf{c} = [\![c]\!]$ with $c \in L(C^I_{(\mathcal{G}_\Pi)})$ and we choose c such that the length of the derivation is minimal.

- Assume $X \vdash X_1 \odot_I X_2 \vdash^* c$ is a derivation in $\mathcal{G}_\Pi$. Then c is some $c_1 \odot_I c_2$ and, for $i = 1, 2$, $X_i \vdash^* c_i$ is a sub-derivation in $\mathcal{G}_\Pi$.
  The derivations $X_i \vdash^* c_i$ induce derivations $X_1^J \vdash^* \Pi_J c_1$ and $X_2^K \vdash^* \Pi_K c_2$. Furthermore, the new derivations have the same lengths (Prop. 9.2).
  For $i = 1, 2$, write $[\![c_i]\!]$ under the form $(x_i^1, \ldots, x_i^p)$.
  Let $J = \{j \in I \mid x_1^j < x_2^j\}$ and $K = I \setminus J$: this entails $\mathbf{c} = \Pi_J \mathbf{c}_1 + \Pi_K \mathbf{c}_2$.
  By construction, the rule $X \longrightarrow X_1^J + X_2^K$ is in $\mathcal{G}_A$.
  By induction hypothesis, $X_1^J \vdash^* \Pi_J c_1$ and $X_2^K \vdash^* \Pi_K c_2$ are derivations in $\mathcal{G}_A$ (where $\mathbf{c}_1 = [\![c_1]\!], \mathbf{c}_2 = [\![c_2]\!]$).
  Therefore $X \vdash^* \Pi_J c_1 + \Pi_K c_2$ in $\mathcal{G}_A$ and $[\![c_1]\!] + [\![c_2]\!] = \mathbf{c}$.
- In case the derivation starts with a rule of the form $C \longrightarrow c$ or $C \longrightarrow C'$, the proof is similar (even simpler). $\square$

Proposition 9.4 *For any non-terminal C of $\mathcal{G}$ and cost $\mathbf{c} \in \mathcal{L}(C_{(\mathcal{G}_A)})$ there exists some $\mathbf{d} \in \mathcal{L}(C_{(\mathcal{G})})$ s.t. $\mathbf{c} \leq \mathbf{d}$.*

PROOF. Actually we prove the result for all non-terminals of $\mathcal{G}_\Pi$. Assume $\mathbf{c} \in \mathcal{L}(X_{(\mathcal{G}_A)})$. We prove $\mathbf{c} \leq \mathbf{d}$ for some $\mathbf{d} \in \mathcal{L}(X_{(\mathcal{G}_\Pi)})$ by induction on the length of minimal derivations of cost terms c such that $[\![c]\!] = \mathbf{c}$.

- Assume $X \vdash X_1^J + X_2^K \vdash^* c$ is a derivation in $\mathcal{G}_A$, where the rule $X \longrightarrow X_1^J + X_2^K$ comes from a rule $X \longrightarrow X_1 \odot_{J \cup K} X_2$ in $\mathcal{G}_\Pi$.
  Then there exists c_1, c_2 such that $\mathbf{c} = [\![c_1]\!] + [\![c_2]\!]$ and two sub-derivations $X_1^J \vdash^* c_1$ and $X_2^K \vdash^* c_2$.
  By induction hypothesis, there are $\mathbf{d}_1 \geq \mathbf{c}_1 = [\![c_1]\!]$ and $\mathbf{d}_2 \geq \mathbf{c}_2 = [\![c_2]\!]$ s.t. $\mathbf{d}_1 \in \mathcal{L}(X_1^J)$ and $\mathbf{d}_2 \in \mathcal{L}(X_2^K)$.

By definition there exists $d_1 \in \mathcal{L}(X_1^J)$ and $d_2 \in L(X_2^K)$ s.t. $X_1^J \vdash^* d_1$ and $X_2^K \vdash^* d_2$ are derivations in $\mathcal{G}_\Pi$.

By Prop. 9.2, $d_1 = \Pi_J d_1'$ and $d_2 = \Pi_K d_2'$ for some derivations $X_1 \vdash^* d_1'$ and $X_2 \vdash^* d_2'$ in $\mathcal{G}_\Pi$: we deduce $[\![d_1]\!] + [\![d_2]\!] \leq [\![d_1']\!] \odot_{J\cup K} [\![d_2']\!]$.

Let us write d for $d_1' \odot_{J\cup K} d_2'$. We have $X \vdash^* d$ in $\mathcal{G}_\Pi$ and $\mathbf{d} = [\![d]\!] \geq [\![d_1]\!] + [\![d_2]\!] \geq \mathbf{c}_1 + \mathbf{c}_2 = \mathbf{c}$.

- In case the derivation start with a rule of the form $C \longrightarrow c$ or $C \longrightarrow C'$, the proof is similar (even simpler). □

PROOF (of Theorem 9.1). The proof is similar to the proof of theorem 8.3. To a parameterized formula φ of the form $\exists \bar{u}, \bar{v} \left(\bigwedge_{i=1}^n u_i \xrightarrow{\exists x_i > \psi_i(\bar{y}_i)} v_i\right)$ we associate $\mathcal{G}_\varphi$, a regular tree grammar with costs which generates the solution of $\exists \bar{u}, \bar{v} \left(\bigwedge_{i=1}^n u_i \xrightarrow{\exists true} v_i\right)$. According to Propositions 9.3 and 9.4, for each set of costs $\mathcal{L}(C_Q)$ of $\mathcal{G}_\varphi$, we can compute a semilinear set $\mathcal{L}(C_Q^A)$ which approximates $\mathcal{L}(C_Q)$.

We claim that φ is satisfiable iff the Presburger formula

$$\exists \langle x_1', \ldots, x_n' \rangle \in \mathcal{L}(C_{Q_{Ax}}^A) \wedge \bigwedge_{i=1}^{i=n} x_i' > \psi_i(\bar{y}_i)$$

is satisfiable:

- $\Rightarrow$ direction: since $\mathcal{L}(C_{Q_{Ax}}) \subseteq \mathcal{L}(C_{Q_{Ax}}^A)$, then $x_i \in \mathcal{L}(C_{Q_{Ax}})$ implies that $x_i \in \mathcal{L}(C_{Q_{Ax}}^A)$. Therefore the Presburger's formula holds.
- $\Leftarrow$ direction: If $\langle x_1', \ldots, x_n' \rangle \in \mathcal{L}(C_{Q_{Ax}}^A)$ and $x_i' > \psi_i(\bar{y}_i)$ for $1, \ldots, n$, there exists $\langle x_1, \ldots, x_n \rangle \in \mathcal{L}(C_{Q_{Ax}})$ such that $x_i \geq x_i' > \psi_i(\bar{y}_i)$ for $i = 1, \ldots, n$. Therefore φ is satisfiable. □

10 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 open problem is the complexity of the decision problem for the logic *TL*. The decision procedure that we gave is non-elementary since each quantifier alternation yields an exponential blowup but we don't know whether a lower complexity can be obtained. 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

[1] R. Alur, K. Etessami, S. La Torre, and D. Peled. Parametric temporal logic for "model measuring". *ACM Trans. Computational Logic*, 2(3):388–407, 2001.

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

[3] J. C. M. Baeten, J. A. Bergstra, and J. W. Klop. Decidability of bisimulation equivalence for processes generating context-free languages. *Journal of the ACM*, 40(3):653–682, 1993.

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

[5] A. Bouajjani, R. Echahed, and P. Habermehl. On the verification problem of nonregular properties for nonregular processes. In *Proc. 10th IEEE Symp. Logic in Computer Science (LICS '95), San Diego, CA, USA, June 1995*, pages 123–133, 1995.

[6] A. Bouajjani, R. Echahed, and P. Habermehl. Verifying infinite state processes with sequential and parallel composition. In *Proc. 22nd ACM Symp. Principles of Programming Languages (POPL '95), San Francisco, CA, USA, Jan. 1995*, pages 95–106, 1995.

[7] A. Bouajjani and T. Touili. Reachability analysis of Process Rewrite Systems. In *Proc. 23rd Conf. Found. of Software Technology and Theor. Comp. Sci. (FST&TCS 2003), Mumbai, India, Dec. 2003*, volume 2914 of *Lecture Notes in Computer Science*, pages 74–87. Springer, 2003.

[8] O. Bukart, D. Caucal, F. Moller, and B. Steffen. Verification on infinite structures. In J. A. Bergstra, A. Ponse, and S. A. Smolka, editors, *Handbook of Process Algebra*, chapter 9, pages 545–623. Elsevier Science, 2001.

[9] D. Caucal. On word rewriting systems having a rational derivation. In *Proc. 3rd Int. Conf. Foundations of Software Science and Computation Structures (FOSSACS 2000), Berlin, Germany, Mar.-Apr. 2000*, volume 1784 of *Lecture Notes in Computer Science*, pages 48–62. Springer, 2000.

[10] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available on: `http://www.grappa.univ-lille3.fr/tata`, 1997. release Oct., 1st 2002.

[11] H. Comon and Y. Jurski. Timed automata and the theory of real numbers. In *Proc. 10th Int. Conf. Concurrency Theory (CONCUR '99), Eindhoven, The Netherlands, Aug. 1999*, volume 1664 of *Lecture Notes in Computer Science*, pages 242–257, Eindhoven, The Netherlands, 1999. Springer.

[12] M. Dauchet and S. Tison. The theory of ground rewrite systems is decidable. In *Proc. 5th IEEE Symp. Logic in Computer Science (LICS '90), Philadelphia, PA, USA, June 1990*, pages 242–248, 1990.

[13] N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, chapter 6, pages 243–320. Elsevier Science, 1990.

[14] E. A. Emerson and R. J. Trefler. Parametric quantitative temporal reasoning. In *Proc. 14th IEEE Symp. Logic in Computer Science (LICS '99), Trento, Italy, July 1999*, pages 336–343, 1999.

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

[16] J. Esparza and J. Knoop. An automata-theoretic approach to interprocedural data-flow analysis. In *Proc. 2nd Int. Conf. Foundations of Software Science and Computation Structures (FOSSACS '99), Amsterdam, The Netherlands, Mar. 1999*, volume 1578 of *Lecture Notes in Computer Science*, pages 14–30. Springer, 1999.

[17] 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 (POPL 2000), Boston, MA, USA, Jan. 2000*, pages 1–11, 2000.

[18] S. Ginsburg. *The Mathematical Theory of Context-Free Languages*. McGraw-Hill, 1966.

[19] A. C. Gómez and J.-E. Pin. On a conjecture of Schnoebelen. In *Proc. 7th Int. Conf. Developments in Language Theory (DLT 2003), Szeged, Hungary, July 2003*, volume 2710 of *Lecture Notes in Computer Science*, pages 35–54. Springer, 2003.

[20] V. Gouranton, P. Réty, and H. Seidl. Synchronized tree languages revisited and new applications. In *Proc. 4th Int. Conf. Foundations of Software Science and Computation Structures (FOSSACS 2001), Genova, Italy, Apr. 2001*, volume 2030 of *Lecture Notes in Computer Science*, pages 214–229. Springer, 2001.

[21] Y. Hirshfeld and M. Jerrum. Bisimulation equivalence is decidable for normed process algebra. In *Proc. 26th Int. Coll. Automata, Languages, and Programming (ICALP '99), Prague, Czech Republic, July 1999*, volume 1644 of *Lecture Notes in Computer Science*, pages 412–421. Springer, 1999.

[22] P. Jančar, A. Kučera, and R. Mayr. Deciding bisimulation-like equivalences with finite-state processes. *Theoretical Computer Science*, 258(1–2):409–433, 2001.

[23] Y. Kesten, O. Maler, M. Marcus, A. Pnueli, and E. Shahar. Symbolic model checking with rich assertional languages. *Theoretical Computer Science*, 256(1–2):93–112, 2001.

[24] A. Kučera. Regularity is decidable for normed PA processes in polynomial time. In *Proc. 16th Conf. Found. of Software Technology and Theor. Comp. Sci. (FST&TCS '96), Hyderabad, India, Dec. 1996*, volume 1180 of *Lecture Notes in Computer Science*, pages 111–122. Springer, 1996.

[25] A. Kučera. How to parallelize sequential processes. In *Proc. 8th Int. Conf. Concurrency Theory (CONCUR '97), Warsaw, Poland, Jul. 1997*, volume 1243 of *Lecture Notes in Computer Science*, pages 302–316. Springer, 1997.

[26] A. Labroue and Ph. Schnoebelen. An automata-theoretic approach to the reachability analysis of RPPS systems. *Nordic Journal of Computing*, 9(2):118–144, 2002.

[27] D. Lugiez and Ph. Schnoebelen. The regular viewpoint on PA-processes. *Theoretical Computer Science*, 274(1–2):89–115, 2002.

[28] R. Mayr. Tableaux methods for PA-processes. In *Proc. Int. Conf. Automated Reasoning with Analytical Tableaux and Related Methods (TABLEAUX '97), Pont-à-Mousson, France, May 1997*, volume 1227 of *Lecture Notes in Artificial Intelligence*, pages 276–290. Springer, 1997.

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

[30] R. Mayr. Decidability of model checking with the temporal logic EF. *Theoretical Computer Science*, 256(1–2):31–62, 2001.

[31] P. Mielniczuk. Basic theory of feature trees. *ACM Trans. Computational Logic*, 5(3):385–402, 2004.

[32] F. Moller. Infinite results. In *Proc. 7th Int. Conf. Concurrency Theory (CONCUR '96), Pisa, Italy, Aug. 1996*, volume 1119 of *Lecture Notes in Computer Science*, pages 195–216. Springer, 1996.

[33] R. J. Parikh. On context-free languages. *Journal of the ACM*, 13(4):570–581, 1966.

[34] G. D. Plotkin. A structural approach to operational semantics. *Journal of Logic and Algebraic Programming*, 60–61:17–139, 2004. First published as Aarhus Univ. Lect. Notes in 1981.

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

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

[37] J. W. Thatcher and J. B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical Systems Theory*, 2(1):57–81, 1968.

A About Remark 4.5

We provide an example Δ where, assuming the SOS rules given in Remark 4.5, $\xrightarrow{*}$ is not regular.

Let $\mathit{Const} \stackrel{\text{def}}{=} \{X, Y\}$ and $\Delta \stackrel{\text{def}}{=} \{X \to X, Y \to 0\}$. Write X^n for $\underbrace{X.(X.(X \ldots X))}_{n}$ and $Y^m X^n$ for $\underbrace{Y.(Y.(\ldots Y}_{m} .X^n))$.

For any $n, m > 0$, the pair $\langle Y^m X^n, X^n \rangle$ is in $\xrightarrow{*}$ so that any regular grammar generating $\xrightarrow{*}$ must generate the products $Y^m X^n \times X^n$. For $m > n$, this product has the form

$$\underbrace{YX..(YX..(\cdots(YX}_{n}..(Y\bot..(\cdots Y\bot..(Y\bot..\underbrace{(YX..(X\bot..(\cdots X\bot)}_{n})))))))))$$

Here a standard pumping argument shows that, for n large enough, there must be some k s.t. the product term

$$\underbrace{YX..(YX..(\cdots(YX}_{n+k}..(Y\bot..(\cdots Y\bot..(Y\bot..\underbrace{(YX..(X\bot..(\cdots X\bot)}_{n})))))))))$$

is also accepted.

This would imply $Y^{m+k} X^n \xrightarrow{*} X^{n+k}$, in contradiction with Δ. Hence no regular tree grammar $\mathcal{G}$ can generate $\xrightarrow{*}$.

B Proof of Proposition 8.1

The proof is a reduction from PCP, the Post Correspondence Problem: let $P = \langle \alpha_1, \ldots, \alpha_m, \beta_1, \ldots, \beta_m \rangle$ be an instance of PCP. W.l.o.g. we can assume the α_i's and β_i's are non-empty words over the two-letters alphabet $\Sigma = \{a, b\}$.

We define a PA system that can look for solutions to P. The set of actions has two copies of a and b and two copies of all numbers from 1 to m, plus a special purpose marker $\#$: $Act \stackrel{\text{def}}{=} \{\#, a^+, b^+, 1^+, \dots, m^+, a^-, b^-, 1^-, \dots, m^-\}$.

Δ_P has a rule $X^+ \xrightarrow{\#} 0$ and, for every α_i of the form $a_1 \cdots a_{n_i}$, there are rules $X^+ \xrightarrow{a_1^+} Z_{i,1}^+$, $Z_{i,1}^+ \xrightarrow{a_2^+} Z_{i,2}^+$, $\dots$, $Z_{i,j-1}^+ \xrightarrow{a_j^+} Z_{i,j}^+$, $\dots$, $Z_{i,n_i-1}^+ \xrightarrow{a_{n_i}^+} X^+.Y_i^+$, and finally $Y_i^+ \xrightarrow{i^+} 0$.

X^+ is then able to perform a sequence of α_i's (all the while storing the i's), then stop with $\#$, and then emit the i's in reverse order. Formally, for any sequence $i_1 \dots i_p$ of indexes in $\{1, \dots, m\}$ there is one way to perform

$$X^+ \xrightarrow{u_{i_1}^+ \dots u_{i_p}^+} t \xrightarrow{\#} t' \xrightarrow{i_p^+ \dots i_1^+} t''$$

Furthermore this behavior is completely determined by $i_1 \dots i_p$: t, t' and t'' are unique, e.g., t' is $(\cdots((O.Y_{i_1}^+).Y_{i_2}^+)\cdots).Y_{i_p}^+)$, and t'' is now a terminated process.

We define X^- with similar rules, this time using the β_i's instead of the α_i's, and using the letters with "$-$" superscripts replacing the $+$'s. Now if P has a solution $i_1 \dots i_p$, then $X^+ \parallel X^-$ may display it via some

$$X^+ \parallel X^- \xrightarrow{w} t_1 \xrightarrow{\#\#} t_2 \xrightarrow{w'} s$$

where w is a sequence of matched $a_i^+.a_i^-$, w' is a sequence of matched $i^+.i^-$, and s is terminated. This is obtained by shuffling behaviors from X^+ and X^-. Conversely, $X^+ \parallel X^-$ has such a behavior only if P admits a solution.

We now write the *TLC* formula

$$\varphi(t_0) \stackrel{\text{def}}{=} \exists t_1 \exists t_2 \exists t_3 \left[\begin{array}{l} t_0 \xrightarrow{\exists \bar{x} \psi(\bar{x})} t_1 \ \wedge \ t_1 \xrightarrow{\#\#} t_2 \ \wedge \ t_2 \xrightarrow{\exists \bar{x} \psi(\bar{x})} t_3 \ \wedge \ t_3 \in \mathit{Final} \\ \wedge \ \forall s \left(\begin{array}{rl} & \neg(t_0 \xrightarrow{*} s) \vee \neg(s \xrightarrow{*} t_1) \vee t_0 \xrightarrow{\exists \bar{x} \psi'(\bar{x})} s \\ \wedge & \neg(t_2 \xrightarrow{*} s) \vee \neg(s \xrightarrow{*} t_3) \vee t_2 \xrightarrow{\exists \bar{x} \psi'(\bar{x})} s \end{array} \right) \end{array} \right]$$

where $\psi(\bar{x})$ is a Presburger formula stating that the number of $+$ letters equals the number of $-$ letters, and $\psi'(\bar{x})$ states that if $\psi(\bar{x})$ then, for each individual letter, the number of $+$'s agrees with the number of $-$'s. Formally, if $\bar{x} = \langle n_1, \dots, n_k, m_1, \dots, m_k \rangle$ is the cost (omitting the $\#$'s) ψ is $n_1 + \cdots + n_k = m_1 + \cdots + m_k$ and ψ' is $\psi(\bar{x}) \Rightarrow (n_1 = m_1 \wedge \cdots \wedge n_k = m_k)$.

Observe that the negations in φ only occur inside atoms "$\neg(s \to t)$" denoting some recognizable $R(s,t)$ since the complements of $\xrightarrow{*}$ and $\to$ are recognizable relations. Hence φ is a *TLC* formula.

Lemma B.1 *P admits a solution iff $\varphi(X^+ \parallel X^-)$ is true (in the context of Δ_P).*

PROOF. ($\Rightarrow$:) We choose t_1, t_2 and t_3 corresponding to the sequence $i_1 \dots i_p$ solving P. Any s between $X^+ \parallel X^-$ and t_1 (resp. between t_2 and t_3) is reached by interleaving some prefix of some w^+ and some other prefix of the corresponding w^-, hence the two "$t_{\dots} \xrightarrow{\exists \bar{x} \psi'(\bar{x})} s$" hold since if the prefix have same length, i.e., "$\psi(\bar{x})$", then they match.
($\Leftarrow$:) Assume $\varphi(X^+ \parallel X^-)$ is true. Then t_1, t_2 and t_3 are reached by interleaving $X^+ \xrightarrow{w_1^+} t_1^+ \xrightarrow{\#} t_2^+ \xrightarrow{w_2^+} t_3^+$ and $X^- \xrightarrow{w_3^-} t_1^- \xrightarrow{\#} t_2^- \xrightarrow{w_4^-} t_3^-$. By construction of Δ_P, w_2 and w_4 are sequences of indexes from $\{1, \dots, m\}$, and w_1 (resp. w_3) is the corresponding concatenation of α_i's (resp. of β_i). We show $w_1 = w_3$ and $w_2 = w_4$, i.e., P admits a solution: since all interleaving are allowed when moving from $X^+ \parallel X^-$, φ states that every prefix w_1' and w_2' of, resp., w_1 and w_2 with same length have same Parikh image. Since this holds for every prefix, this entails that they have the same letters at the same position, and hence are equal. The same holds for w_3 and w_4. $\square$

Hence PCP can be reduced to satisfiability of TLC, concluding the proof of Prop. 8.1.