Verifying Performance Equivalence for Timed Basic Parallel Processes

B. Bérard, A. Labroue, and Ph. Schnoebelen

Lab. Spécification & Vérification
ENS de Cachan & CNRS UMR 8643
61, av. Pdt. Wilson, 94235 Cachan Cedex France
email: {berard,labroue,phs}@lsv.ens-cachan.fr

Abstract. We address the problem of deciding performance equivalence for a timed process algebra in which actions are urgent and durational, and where parallel components have independent local clocks.
This process algebra can be seen as a timed extension of BPP, a process algebra giving rise to infinite-state processes. While bisimulation was known to be decidable for BPP with a non elementary complexity, our main and surprising result is that, for the timed extension, performance equivalence is decidable in polynomial time.

1 Introduction

Performance of processes. In the field of concurrency semantics, there exists a well-developed and widely accepted approach based on equivalences that relate processes having the same behaviour [Mil89,Gla90]. This framework has been extended in many directions in order to take various aspects into consideration: timing, causality, probability, locality, etc.

In the timed framework, some efforts have been directed toward defining a robust notion of "*performance*", that would allow comparing the efficiency of systems that have the same functional behaviour (what they do) but different speeds (how fast they do it). See, e.g., [MT91,AH92,FM95,GRS95,CGR97,Cor98].

Durational urgent actions. The efficiency preorders and equivalences considered in [GRS95,AM96,CGR97,Cor98] apply to process algebras where parallel components have their own independent local clocks, where actions have a *duration* and are *urgent*. Urgent actions take place as soon as possible and can only be delayed when one process must wait until synchronization with another process becomes possible. When the process algebra does not allow synchronization, this gives rise to a nice theory where performance equivalence is a congruence for all process constructors [CGR97].

Verification. These earlier works mainly focused on semantics. However, verification issues have been addressed in this framework:
(1) In [CP96], *lazy performance equivalence* is shown decidable over a class of

systems having only finite control but allowing for an infinite number of configurations when taking into account the values of the local clocks.
(2) In [CC97], a model checking problem for TAL (a modal logic with time) is shown decidable over another class of systems with finite control.
In both cases, the decision method relies on building a *finite* approximation of the system, on which the original problem can be solved with standard finite-state methods. This induces algorithms with exponential running time since the finite approximation has exponential size [1]. These results can probably be implemented with only polynomial-space requirements, but the issue is not addressed and no lower bounds for the structural complexity of the problems are given.

Removing the finite control assumption. To the best of our knowledge, when systems have a potentially infinite number of control states (disregarding clock values), nothing is known about verification issues for these processes algebra with urgent actions and local clocks [2]. This is probably because the problem combines two difficulties as it lies at the intersection of two recent fields: verification of timed systems and verification of infinite untimed systems.

Our contribution. In this paper we investigate the decidability of the *performance equivalence* introduced in [CGR97] when no finite-state restriction is made. Because no synchronization is considered in this framework, the resulting systems have a "BPP + Time" flavor [3], in a setting with local clocks. Hence our use of "TBPP" to denote this algebra.

Decidability of bisimulation for (untimed) BPP is known, via an elegant algorithm (alas with non-elementary complexity) [CHM93]. The connection with BPP is what motivated our study: we wanted to see whether local clocks could be dealt with.

Our main result is that performance equivalence is decidable for TBPP, and can be decided in polynomial-time (it is in fact PTIME-complete). Surprisingly, the addition of local clocks does not make the problem harder: they allow decomposing systems in a way not unlike what happens for normed processes [HJM96].

This is good news since algorithms for the analysis of well-behaved infinite-state systems have important applications, ranging from static analysis [EK99] to modeling and verification of communication protocols [CFP95]. This also justifies our view that negative results about basic process algebra are not always the last word, and that the field still contains many unexplored paths.

---

[1] Since the approximation is based on the idea that exact clock values (or differences between then) can be forgotten when they are large enough, this has similarities with the region graph technique of [ACD93].

[2] In the better known global clock framework, we are aware of [AJ98] where the systems may have infinitely many distinct states. In addition, there exists a large body of literature on Timed Petri Nets, but most of these works do not offer decidability results for unbounded nets.

[3] BPP is the algebra of Basic Parallel Processes [Chr93].

Plan of the paper. Section 2 introduces our notation for TBPP, and the operational semantics while Section 3 introduces the performance equivalences we consider. The main technical part starts with the introduction of the syntactic congruence (Section 4) and the cancellation lemmas (Section 5) that allow us to prove the main result in Section 6.

Our presentation of TBPP is mostly orientated towards the proof of the main result: we refer to [CGR97] for motivations, examples, and further discussion of this process algebra.

2 Timed Basic Parallel Processes

In this section, we define the timed process algebra TBPP as a timed extension of BPP. This definition is based on the features proposed in [GRS95,AM96,CGR97]:

- The time domain is the set $\mathbb{N}$ of natural numbers.
- We consider urgent and *durational* actions: a duration function associates its duration (number of time units taken for execution) with each action. This mapping is external to the syntax.
- Parallel components have independent clocks and executions are asynchronous and *ill-timed but well-caused.*

[AM96] showed the technical advantages of the "ill-timed but well-caused" viewpoint (which admits an intuitive understanding in terms of external observation). In this framework, time is not used to enforce a synchronous view of the system.

We mainly deviate from [GRS95,CGR97] by two technical points that do not bring any real semantical change:

- The date n in a step $u \xrightarrow{a,n} v$ denotes the beginning time for a, not the completing time.
- Instead of defining processes through recursive equations (as is traditional in process algebra), we adopt Moller's approach where behaviour is defined via a set of rewrite rules [Mol96]. This is for technical convenience only.

2.1 Syntax

We consider a set of *action names* Act ranged over by $a, b, \dots$ and a set of *process variables* $\mathcal{X}$ ranged over by $X, Y, \dots$.

Definition 2.1. *The set* $\mathcal{T}$ *of* TBPP-terms *is given by the following abstract syntax:*

$$t, u ::= Nil \mid X \mid t \parallel u \mid 1 \triangleright t.$$

As usual, Nil denotes the empty process which cannot proceed with any action, and $t \parallel u$ is the parallel combination of t and u (no synchronization is possible). $1 \triangleright t$ denotes the process which behaves like t, but with a one time unit delay.

We write $n \triangleright t$ for $\overbrace{1 \triangleright (1 \triangleright (\cdots (1 \triangleright}^{n} t) \cdots))$ and X^n for $\overbrace{X \parallel X \parallel \cdots \parallel X}^{n}$. By convention, $0 \triangleright t$ stands for t and X^0 stands for Nil.

For a term t, we denote by $Var(t)$ the set of process variables occurring in t, e.g., $Var(X \parallel 1 \triangleright (X \parallel Y)) = \{X, Y\}$.

Definition 2.2. *A* TBPP declaration *is a finite set* $\Delta \subseteq \mathcal{X} \times Act \times \mathcal{T}$ *of* process rewrite rules*, written* $\{X_i \xrightarrow{a}_\Delta t_i \mid i = 1, \ldots, n\}$*, such that* $Var(t_i) \subseteq \{X_1, \ldots, X_n\}$ *for any* i.

Note that the X_i's need not be distinct. Additionally, we require that any variable X_i used in Δ appears in the left-hand side of at least one rule from Δ (this is for technical convenience only).

In the examples, we often use the convenient CCS-like notations with action-prefixing, non-deterministic choice (denoted by $+$) and guarded recursion. E.g., the definition

$$\begin{aligned} X_1 &\stackrel{\text{def}}{=} a.(1 \triangleright (a \parallel a \parallel a)) + a.(1 \triangleright (a.a.a)), \\ X_2 &\stackrel{\text{def}}{=} a.(1 \triangleright (a \parallel a \parallel a)) + a.(1 \triangleright (a.a) \parallel 1 \triangleright a) + a.(1 \triangleright (a.a.a)) \end{aligned}$$

is just a shorthand for

$$\Delta \stackrel{\text{def}}{=} \left\{ \begin{array}{l} X_1 \xrightarrow{a} 1 \triangleright (Z_a \parallel Z_a \parallel Z_a),\ X_1 \xrightarrow{a} 1 \triangleright Z_{a.a.a}, \\ X_2 \xrightarrow{a} 1 \triangleright (Z_a \parallel Z_a \parallel Z_a),\ X_2 \xrightarrow{a} 1 \triangleright (Z_{a.a} \parallel 1 \triangleright Z_a),\ X_2 \xrightarrow{a} 1 \triangleright Z_{a.a.a}, \\ Z_a \xrightarrow{a} \mathit{Nil},\ Z_{a.a} \xrightarrow{a} Z_a,\ Z_{a.a.a} \xrightarrow{a} Z_{a.a}. \end{array} \right\}$$

2.2 Operational semantics

The evolution of a TBPP process is represented by a transition system where the steps carry visible labels of the form (a, n), where $a \in Act$ is an action and $n \in \mathbb{N}$ is the time at which the step occurs. Actually, n is the time at which the step starts, and knowing when it finishes requires knowing the duration of a (the time it takes to perform an a).

Definition 2.3. *A* duration function f *is a mapping from* Act *to* $\mathbb{N} \setminus \{0\}$.
$\mathbf{1}$ *is the constant duration function s.t.* $\mathbf{1}(a) \stackrel{def}{=} 1$ *for any* a.

Having $f(a) = 3$ means that a takes 3 time units. Here a duration function may represent for instance the performance of a particular machine. Thus this framework makes it possible to clearly distinguish the *functional* definition Δ and the *performance* definition f.

Remark 2.4. It is possible to generalise duration "functions" so that (possibly infinite) *sets* of values are associated with actions. Our main decidability result is still valid in this framework (assuming that f is given effectively, for example by having $f(a)$ be a recognizable set of natural numbers) but the complexity measures are affected. □

A pair (Δ, f) where Δ is a TBPP declaration and f a duration function defines a labeled transition relation $\rightarrow_f \subseteq \mathcal{T} \times (Act \times \mathbb{N}) \times \mathcal{T}$, where $\rightarrow_f$ is given inductively via the following SOS rules:

$$\frac{}{X \xrightarrow{a,0}_f f(a) \triangleright t}\ (X \xrightarrow{a}_\Delta t) \in \Delta \qquad\qquad \frac{t \xrightarrow{a,n}_f t'}{t \parallel u \xrightarrow{a,n}_f t' \parallel u}$$

$$\frac{t \xrightarrow{a,n}_f t'}{1 \triangleright t \xrightarrow{a,n+1}_f 1 \triangleright t'} \qquad\qquad \frac{t \xrightarrow{a,n}_f t'}{u \parallel t \xrightarrow{a,n}_f u \parallel t'}$$

We use the usual standard abbreviations: $t \xrightarrow{w} t'$ (with $w \in (Act \times \mathbb{N})^*$), $t \xrightarrow{*} t'$, ... and omit the f subscript when it is clear from the context.

A *run* of t is a finite or infinite sequence $(t =)\ t_0 \xrightarrow{a_1,n_1} t_1 \xrightarrow{a_2,n_2} t_2 \cdots \xrightarrow{a_k,n_k} t_k \cdots$. The *trace* of such a run is the sequence $w = (a_1, n_1)(a_2, n_2) \ldots (a_k, n_k) \ldots$

A run is *ill-timed* if there are two positions $i > j$ s.t. $n_i < n_j$. TBPP allows ill-timed runs, but [AM96] argues convincingly that (1) this brings no semantical problem since "the ill-timed runs are well-caused" (i.e. local, causaly related, clock values do increase along a run), and (2) this greatly simplifies the technical treatment (see also [CGR97]).

Example 2.5. Consider $f = \mathbf{1}$ and the term X given by $X \stackrel{\text{def}}{=} a(bb \parallel c) + ac(b \parallel b)$. The maximal traces of X are $(a, 0)(b, 1)(b, 2)(c, 1)$, $(a, 0)(b, 1)(c, 1)(b, 2)$, $(a, 0)(c, 1)(b, 1)(b, 2)$ and $(a, 0)(c, 1)(b, 2)(b, 2)$. The first one is ill-timed.

2.3 Timing measures

Two structural measures can be associated with a term: $minclock(u) \in \mathbb{N} \cup \{\infty\}$ is the earliest time at which u can start an action, while $maxclock(u) \in \mathbb{N} \cup \{-\infty\}$ is the latest time.

We assume ordering and addition over $\mathbb{N}$ are extended in the obvious way to ∞ and $-\infty$, and we define the two measures by structural induction over terms:

$$\begin{aligned}
minclock(Nil) &\stackrel{\text{def}}{=} \infty & maxclock(Nil) &\stackrel{\text{def}}{=} -\infty \\
minclock(X) &\stackrel{\text{def}}{=} 0 & maxclock(X) &\stackrel{\text{def}}{=} 0 \\
minclock(1 \triangleright u) &\stackrel{\text{def}}{=} 1 + minclock(u) & maxclock(1 \triangleright u) &\stackrel{\text{def}}{=} 1 + maxclock(u)
\end{aligned}$$

$$\begin{aligned}
minclock(u \parallel v) &\stackrel{\text{def}}{=} \min(minclock(u), minclock(v)) \\
maxclock(u \parallel v) &\stackrel{\text{def}}{=} \max(maxclock(u), maxclock(v))
\end{aligned}$$

Example 2.6. For $u = 1 \triangleright (X \parallel 2 \triangleright X)$ we have $minclock(u) = 1$ and $maxclock(u) = 3$, and indeed if Δ contains $X \xrightarrow{a} t$, then $u \xrightarrow{a,1} \ldots$ and $u \xrightarrow{a,3} \ldots$

More generally:

Lemma 2.7. *For any u, $u \xrightarrow{a,n} v$ implies $minclock(u) \leq n \leq maxclock(u)$ and $minclock(u) \leq minclock(v)$.*
In the other direction, if u can make a move, then there exists a move $u \xrightarrow{a,n} v$ with $n = minclock(u)$ and a $u \xrightarrow{a',n'} v'$ with $n' = maxclock(u)$.

Proof. Easy induction on u. $\square$

More fundamental is the following lemma, stating that *minclock* can be made arbitrarily large:

Lemma 2.8. *For any u and any $n \in \mathbb{N}$ there is a $u \xrightarrow{*} v$ s.t. $minclock(v) > n$.*

Proof. An easy induction on u shows that if $minclock(u) < \infty$ then that $u \xrightarrow{*} v$ for some v s.t. $minclock(v) > minclock(u)$. $\square$

3 Performance equivalences

In this section, we recall the definition of performance equivalence introduced in [GRS95,CGR97]: "*f-performance equivalence*" is associated with a duration function f while "*independent-performance equivalence*" abstracts from the particular duration function.

f-performance equivalence corresponds to strong bisimulation [Mil89] on TBPP transitions, taking timing information into account.

Definition 3.1. *A relation $\mathcal{R} \subseteq \mathcal{T} \times \mathcal{T}$ is called a f-*performance relation *if $u\mathcal{R}v$ implies that*

1. *for any $u \xrightarrow{a,n}_f u'$ there is a move $v \xrightarrow{a,n}_f v'$ s.t. $u'\mathcal{R}v'$,*
2. *and vice versa: for any $v \xrightarrow{a,n}_f v'$ there is a $u \xrightarrow{a,n}_f u'$ with $u'\mathcal{R}v'$.*

Definition 3.2. *Two TBPP terms u and v are f-*performance equivalent *(written $u \sim_f v$) if there is a f-performance relation $\mathcal{R}$ such that $u\mathcal{R}v$.*

Example 3.3. Assume $f(a) = 1$ and consider $X \stackrel{\text{def}}{=} a.(X \parallel X)$ and $Y \stackrel{\text{def}}{=} a.Y$. Then $X \not\sim_f Y$ because the steps $X \xrightarrow{a,0} 1 \triangleright (X \parallel X) \xrightarrow{a,1} 1 \triangleright (X \parallel 1 \triangleright (X \parallel X)) \xrightarrow{a,1} 1 \triangleright (1 \triangleright (X \parallel X) \parallel 1 \triangleright (X \parallel X))$ cannot be imitated by Y. (However X and Y are bisimilar when timing is not taken into account: they both behave as a^ω.)

As expected, $\sim_f$ is the largest f-performance relation, it is an equivalence, and a congruence for the $\parallel$ and $1\triangleright$ operators:

Proposition 3.4. *If $u \sim_f v$ and $u' \sim_f v'$ then $1 \triangleright u \sim_f 1 \triangleright v$ and $u \parallel u' \sim_f v \parallel v'$.*

Proof. A consequence of the fact that the SOS rules for $\rightarrow_f$ are in *tyft/tyxt*, or even De Simone's, format [GV92]. $\square$

Additionally, $u \sim_f v$ entails $minclock(u) = minclock(v)$ and $maxclock(u) = maxclock(v)$, as a consequence of Lemma 2.7.

f-performance equivalence enjoys the usual associativity, commutativity and nilpotence laws. The distributivity law, (Eq4), is called a *clock distribution equation* in [CGR97]:

Proposition 3.5. *For any terms t, u, v*

$$\begin{aligned} u \parallel t &\sim_f t \parallel u && \text{(Eq1)} \\ (u \parallel t) \parallel v &\sim_f u \parallel (t \parallel v) && \text{(Eq2)} \\ t \parallel \mathit{Nil} &\sim_f t && \text{(Eq3)} \end{aligned} \qquad \begin{aligned} 1 \triangleright (u \parallel v) &\sim_f (1 \triangleright u) \parallel (1 \triangleright v) && \text{(Eq4)} \\ 1 \triangleright \mathit{Nil} &\sim_f \mathit{Nil} && \text{(Eq5)} \end{aligned}$$

3.1 Performance not depending from f.

Our definitions followed [CGR97] in that we did not mix functional definitions (the rules in Δ, the program, ...) and timing definitions (the duration function f, the hardware, ...).

We may now define a notion of performance equivalence that does not depend on f:

Definition 3.6. *Two terms u and v are* independent-performance equivalent *(written $u \sim_{\mathrm{i}} v$) if $u \sim_f v$ for any duration function f.*

$\sim_{\mathrm{i}}$ is a congruence since it is an intersection of congruences.

Remark 3.7. A byproduct of our study is a proof that $\sim_f$ and $\sim_{\mathrm{i}}$ coincide for any f (Corollary 5.8), which we see as the reason why [CGR97] introduced both an f-performance and an independent-performance preorder (these two preorders do not coincide) but only one performance equivalence, and did not comment about this. However, since we cannot prove Corollary 5.8 without the technical developments of the next sections, we shall keep writing $\sim_f$ as long as necessary. $\square$

4 Structural congruence

Here we introduce a structural congruence for TBPP. It allows us to exhibit a normal form for the terms that generalizes the usual normal form for BPP [CHM93].

Definition 4.1. *We denote by $\equiv$ the smallest congruence induced by the five equations of Proposition 3.5.*

Clearly $u \equiv v$ implies $u \sim_f v$ since $\sim_f$ is a congruence and it satisfies the five equations. Also, since $\equiv$ does not depend on f, $u \equiv v$ entails $u \sim_{\mathrm{i}} v$.

Definition 4.2. *A term $u \in \mathcal{T}$ is in* normal form *if it is some $n_1 \triangleright X_1 \parallel \ldots \parallel n_k \triangleright X_k$ (where the X_i's need not be distinct, and where we allow $n_i = 0$ or $k = 0$).*

Using Proposition 3.5, any term can be rewritten to a structurally equivalent normal form. Moreover, this normal form is unique (modulo associativity and commutativity of $\parallel$). Sometimes we are only interested in the subterms "$0 \triangleright X_i$" in a normal form and write it $X_1 \parallel \ldots \parallel X_n \parallel 1 \triangleright u$.

The normal form of a term u displays all dates for which u can make an immediate step. A consequence is the very useful Lemma:

Lemma 4.3. *$u \sim_f Nil$ iff $u \equiv Nil$ iff $minclock(u) = +\infty$ iff $maxclock(u) = -\infty$.*

5 Cancellation for performance equivalence

In this section, we prove the surprising result that performance equivalence can be reduced to a notion of equality of normal forms. For this, we use a decomposition approach along the lines that have been pioneered by [MM93] and which often work nicely in timed or normed settings (see Prop. 30 in [AM96] or Prop. 2.2.8 in [Hen88]).

The following lemma is the converse of Proposition 3.4. It emphasizes the link between the behaviours of the terms u and $1 \triangleright u$.

Lemma 5.1. *$1 \triangleright u \sim_f 1 \triangleright v$ entails $u \sim_f v$.*

Proof. Standard: one checks that $\mathcal{R} \stackrel{\text{def}}{=} \{(u_1, u_2) \mid 1 \triangleright u_1 \sim_f 1 \triangleright u_2\}$ is an f-performance equivalence. □

Given two TBPP terms u and v, we say that u *is earlier than* v if $maxclock(u) < minclock(v)$ and $v \not\sim_f Nil$. A *separated product* is some $u \parallel v$ with u earlier than v. This syntactic notion is useful because when $u \parallel v$ makes a move at time n, it is possible to assign the move to u or v on the basis of n only.

Lemma 5.2. *Assume $u_1 \parallel u_2$ and $v_1 \parallel v_2$ are separated products s.t. u_1 and v_1 have same maxclock. Then $u_1 \parallel u_2 \sim_f v_1 \parallel v_2$ entails (1) $u_2 \sim_f v_2$ and (2) $u_1 \sim_f v_1$.*

Proof. (1) is easy to see with the separation hypothesis. Let $\mathcal{R}$ be the set of all pairs (u, v) s.t. $u_1 \parallel u \sim_f v_1 \parallel v$ and both $u_1 \parallel u$ and $v_1 \parallel v$ are separated. We show that $\mathcal{R} \cup \sim_f$ is an f-performance equivalence. Indeed, if $u \xrightarrow{a,n} u'$ then $u_1 \parallel u \xrightarrow{a,n} u_1 \parallel u'$ which is still separated (or $u' \sim_f Nil$). Now there is a $v \parallel v_1 \xrightarrow{a,n} t$ with $u_1 \parallel u' \sim_f t$ but this step can only come from v, so that t is some separated $v_1 \parallel v'$ (or $v' \sim_f Nil$). Since u_1 and v_1 have same *maxclock*, $u' \sim_f Nil$ iff $v' \sim_f Nil$ and we have $(u', v') \in \mathcal{R} \cup \sim_f$.

(2) We now prove $u_1 \sim_f v_1$. Let $\mathcal{R}$ be the set of all pairs (u, v) s.t. u and v have same *maxclock*, and there exists a t s.t. $u \parallel t \sim_f v \parallel t$ and $u \parallel t$ and $v \parallel t$ are separated. We show $\mathcal{R} \cup \sim_f$ is an f-performance equivalence. Consider a pair $(u, v) \in \mathcal{R}$ (via some t) and let K be the largest *maxclock* for all immediate successors of u and v. K is finite because TBPP has finite branching. Thanks to Lemma 2.8, there is a sequence w s.t. $t \xrightarrow{w} t'$ and $minclock(t') > K$.

Consider a step $u \xrightarrow{a,n} u'$. Now $u \parallel t \xrightarrow{w} u \parallel t' \xrightarrow{a,n} u' \parallel t'$. Then there must exist a $v \parallel t \xrightarrow{w} v \parallel t'' \xrightarrow{a,n} v' \parallel t''$ with $u \parallel t' \sim_f v \parallel t''$ and $u' \parallel t' \sim_f v' \parallel t''$. We have $t' \sim_f Nil$ iff $t'' \sim_f Nil$ (because they have same *maxclock*) so that (1) gives us $t' \sim_f t''$. Thanks to $minclock(t') > K$, we have $u' \sim_f Nil$ iff $v' \sim_f Nil$

(because $u' \parallel t'$ and $v' \parallel t'$ have same *minclock*). If $u' \not\sim_f Nil$ then both $u' \parallel t'$ and $v' \parallel t''$ are separated, so that $(u', v') \in \mathcal{R}$. Otherwise $u' \sim_f Nil \sim_f v'$. $\square$

Of course, normal forms are separated in an obvious way. Hence:

Lemma 5.3. *Assume $X_1 \parallel \ldots \parallel X_m \sim_f X'_1 \parallel \ldots \parallel X'_{m'}$. Then $m = m'$ and to any X_i we can associate a X'_j s.t. $X_i \sim_f X'_j$.*

Proof. Obviously $m = m'$ since any maximal execution of $X_1 \parallel \ldots \parallel X_m$ has exactly m steps with date 0. Now pick actions a_i's s.t. $X_i \xrightarrow{a_i,0} u_i$. We have $X_1 \parallel \ldots \parallel X_m \xrightarrow{a_2,0}\xrightarrow{a_3,0} \cdots \xrightarrow{a_m,0} X_1 \parallel 1 \triangleright u$. Then there is $X'_1 \parallel \ldots \parallel X'_m \xrightarrow{a_2,0}\xrightarrow{a_3,0} \cdots \xrightarrow{a_m,0} v$ with $X_1 \parallel 1 \triangleright u \sim_f v$. But v is reached by $m-1$ steps at date 0 from $X'_1 \parallel \ldots \parallel X'_m$, hence it has the form $X'_j \parallel 1 \triangleright u'$. The previous lemmas entail $X_1 \sim_f X'_j$ (and $u \sim_f u'$), which conclude the proof. $\square$

Lemma 5.4. *Assume $X_1 \parallel \ldots \parallel X_m \sim_f X'_1 \parallel \ldots \parallel X'_m$. Then there is a bijective $h : [1..m] \to [1..m]$ s.t. $X_i \sim_f X'_{h(i)}$ for all i.*

Proof. We split the multiset $\{X_1, \ldots, X_m, X'_1, \ldots, X'_m\}$ into the equivalence classes induced by $\sim_f$. If every class contains exactly as many X_i's as X'_j's, then h is easy to build. Otherwise we can assume w.l.o.g. that one class is $\{X_1, X_2, \ldots, X_p, X'_1, X'_2, \ldots, X'_q\}$ with $p < q$. Assume $X_i \xrightarrow{a_i,0}$ for all i's, and consider $w = (a_1, 0) \ldots (a_p, 0)$. We have a move $X_1 \parallel \ldots \parallel X_m \xrightarrow{w} 1 \triangleright u \parallel X_{p+1} \parallel \ldots \parallel X_m$. This is imitated by $X'_1 \parallel \ldots \parallel X'_m \xrightarrow{w} 1 \triangleright u' \parallel X'_{i_{p+1}} \parallel \ldots \parallel X'_{i_m}$. Lemma 5.2 entails that $X_{p+1} \parallel \ldots \parallel X_m \sim_f X'_{i_{p+1}} \parallel \ldots \parallel X'_{i_m}$. Now one index (say j) in $\{i_{p+1}, \ldots, i_m\}$ must belong to $\{1, \ldots, q\}$. This contradicts Lemma 5.3 because we assumed X'_j has no match in $X_{p+1}, \ldots, X_m$. $\square$

As a consequence, we now have the following important result, reducing $\sim_f$ to "equality" of normal forms:

Theorem 5.5. *Assume $u \equiv n_1 \triangleright X_1 \parallel \ldots \parallel n_m \triangleright X_m$ and $v \equiv n'_1 \triangleright X'_1 \parallel \ldots \parallel n'_{m'} \triangleright X'_{m'}$. Then $u \sim_f v$ iff there is a bijective $h : [1..m] \to [1..m']$ s.t. $n_i = n'_{h(i)}$ and $X_i \sim_f X'_{h(i)}$ for all i.*

Hence f-performance equivalence of u and v can be reduced to a combination of f-performance equivalence of variables.

An equivalence relation $\approx$ between variables of $\mathcal{X}$ can be extended to terms: we say $u \approx v$ when the normal forms $n_1 \triangleright X_1 \parallel \ldots$ and $n'_1 \triangleright X'_1 \parallel \ldots$ of u and v can be related by a bijective h s.t. $n_i = n'_{h(i)}$ and $X_i \approx X'_{h(i)}$.

Definition 5.6. *An equivalence relation $\approx$ between variables of $\mathcal{X}$ has the* transfer property *if for any $X \approx Y$ and for any $X \xrightarrow{a}_\Delta u$ there is a $Y \xrightarrow{a}_\Delta v$ s.t. $u \approx v$.*

Clearly, if $\approx$ has the transfer property, then its extension to terms is an f-performance equivalence. Conversely, Theorem 5.5 implies that $\sim_f \cap (\mathcal{X} \times \mathcal{X})$ has the transfer property. But the transfer property for some $\approx$ does not depend on f. Hence

Lemma 5.7. *Let f and g be two duration functions. Then $\sim_f$ and $\sim_g$ coincide.*

Corollary 5.8. *$u \sim_{\mathrm{i}} v$ iff there is a duration function f such $u \sim_f v$ iff $u \sim_{\mathbf{1}} v$.*

Remark 5.9. Corollary 5.8 calls for comments. It is not a paradox and can be compared, e.g., with Prop. 13 from [AM96]. Still, we see no easy way to prove it without going through the analysis required for our Theorem 5.5.

Observe that it does not hold if we allow duration functions taking the value zero (which is rather meaningless in our framework). E.g., the terms from Example 3.3 become performance equivalent when $f(a) = 0$.

Similarly, it does not hold in a framework where we associate several values to a same action (cf. Remark 2.4). E.g., with

$$\Delta = \{X \xrightarrow{a} X, X \xrightarrow{a} 2 \triangleright X, \quad Y \xrightarrow{a} X, Y \xrightarrow{a} 1 \triangleright X, Y \xrightarrow{a} 2 \triangleright X\}$$

we have $X \not\sim_{\mathrm{i}} Y$ but $X \sim_f Y$ when $f(a) = \{1, 2\}$. $\square$

As a consequence, we may write indistinctly $\sim$ for any $\sim_f$ (and for $\sim_{\mathrm{i}}$). We do that in the rest of the paper, where we assume additionally that f is the constant duration function $\mathbf{1}$.

6 Decidability of performance equivalence

With the results from Section 5, deciding performance equivalence is simple since it amounts to computing the largest equivalence on variables that has the transfer property.

Proposition 6.1. *Computing $\sim \cap (\mathcal{X} \times \mathcal{X})$ can be done in time polynomial in $|\Delta|$.*

(Where $|\Delta|$ is the number of rules plus the sum of the sizes of the left-hand sides.)

Proof. Given Δ we partition the set $\mathcal{X}$ of variables into equivalence classes. This is done in the usual way, starting with $\approx_0 = \mathcal{X} \times \mathcal{X}$ and refining $\approx_i$ into $\approx_{i+1}$ until stabilization. The refinement step removes a pair (X, Y) from $\approx_i$ whenever there is a $X \xrightarrow{a} u$ in Δ s.t. no $Y \xrightarrow{a} v$ has $u \approx_i v$ (which can be checked easily by a sorting algorithm when u and v are in normal form). Stabilization is reached after at most $|\mathcal{X}| - 1$ refinement steps. $\square$

Hence deciding whether $u \sim v$ can be done in time polynomial in $|u| + |v| + |\Delta|$. Finally we have

Theorem 6.2. *Deciding performance equivalence over TBPP is P-complete.*

Proof. We already know membership in P and only prove P-hardness.

When no parallel composition is involved, TBPP terms behave like finite-state processes where the single local clock just records the length of the history of the computation. Hence performance equivalence of these sequential terms reduces to strong untimed bisimilarity of the underlying unfolded trees, which is just strong bisimilarity of untimed finite state processes, entailing P-hardness [BGS92]. $\square$

7 Conclusion

In this paper we investigated TBPP, a timed extension of the BPP. TBPP is essentially equivalent to the algebra of [CGR97], itself obtained by forbidding synchronization in earlier process algebra with urgent durational actions.

In this framework, [CGR97] introduced *performance equivalence* as a way to relate processes having the same behaviour and the same efficiency.

Our main result is a polynomial-time method for deciding performance equivalence over this class where systems can have an infinite number of different states (even disregarding time). Thus, BPP + Time turns out to be simpler than plain BPP, which is a surprising result. This suggests that timed extensions of related infinite-state algebra should be investigated and could well turn out to be simpler than their better-known untimed counterpart. Let us suggests some directions:

1. Bisimulation of normed PA processes is decidable [HJ99] but appears quite complex. What about performance equivalence for PA+Time?
2. Decidability of observational equivalence (a.k.a. τ-bisimulation) of BPP processes is an important open problem [Esp97,KM99]. What about observational performance equivalence? (Adding τ's to TBPP can be done in several ways: e.g., they can model internal actions with null duration instead of abstracted-away actions with positive duration.)
3. Most behavioural equivalences are undecidable on BPP processes [Hüt94]. What about BPP+Time?

References

[ACD93] R. Alur, C. Courcoubetis, and D. Dill. Model-checking in dense real-time. *Information and Computation*, 104(1):2–34, 1993.

[AH92] S. Arun-Kumar and M. Hennessy. An efficiency preorder for processes. *Acta Informatica*, 29(8):737–760, 1992.

[AJ98] P. A. Abdulla and B. Jonsson. Verifying networks of timed processes. In *Proc. Int. Conf. Tools and Algorithms for the Construction and Analysis of Systems (TACAS'98), Lisbon, Portugal, March 1998*, volume 1384 of *Lecture Notes in Computer Science*, pages 298–312. Springer, 1998.

[AM96] L. Aceto and D. Murphy. Timing and causality in process algebra. *Acta Informatica*, 33(4):317–350, 1996.

[BGS92] J. Balcázar, J. Gabarró, and M. Sántha. Deciding bisimilarity is P-Complete. *Formal Aspects of Computing*, 4(6A):638–648, 1992.

[CC97] Xiao Jun Chen and F. Corradini. On the specification and verification of performance properties for a timed process algebra. In *Proc. 6th Int. Conf. Algebraic Methodology and Software Technology (AMAST'97), Sydney, Australia, Dec. 1997*, volume 1349 of *Lecture Notes in Computer Science*, pages 123–137, 1997.

[CFP95] G. Cécé, A. Finkel, and S. Purushothaman Iyer. Unreliable channels are easier to verify than perfect channels. *Information and Computation*, 124(1):20–31, 1995.

[CGR97] F. Corradini, R. Gorrieri, and M. Roccetti. Performance preorder and competitive equivalence. *Acta Informatica*, 34(11):805–835, 1997.

[CHM93] S. Christensen, Y. Hirshfeld, and F. Moller. Bisimulation equivalence is decidable for basic parallel processes. In *Proc. 4th Int. Conf. Concurrency Theory (CONCUR'93), Hildesheim, Germany, Aug. 1993*, volume 715 of *Lecture Notes in Computer Science*, pages 143–157. Springer, 1993.

[Chr93] S. Christensen. Decidability and decomposition in process algebras. PhD thesis CST-105-93, Dept. of Computer Science, University of Edinburgh, UK, 1993.

[Cor98] F. Corradini. On performance congruences for process algebras. *Information and Computation*, 145(2):191–230, 1998.

[CP96] F. Corradini and M. Pistore. Specification and verification of timed lazy systems. In *Proc. 21st Int. Symp. Math. Found. Comp. Sci. (MFCS'96), Cracow, Poland, Sep. 1996*, volume 1113 of *Lecture Notes in Computer Science*, pages 279–290, 1996.

[EK99] J. Esparza and J. Knoop. An automata-theoretic approach to interprocedural data-flow analysis. In *Proc. 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.

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

[FM95] G. Ferrari and U. Montanari. Dynamic matrices and the cost analysis of concurrent programs. In *Proc. 4th Int. Conf. Algebraic Methodology and Software Technology (AMAST'95), Montreal, Canada, July 1995*, volume 936 of *Lecture Notes in Computer Science*, pages 307–321, 1995.

[Gla90] R. J. van Glabbeek. The linear time – branching time spectrum. In *Proc. Theories of Concurrency (CONCUR'90), Amsterdam, NL, Aug. 1990*, volume 458 of *Lecture Notes in Computer Science*, pages 278–297. Springer, 1990.

[GRS95] R. Gorrieri, M. Roccetti, and E. Stancampiano. A theory of processes with durational actions. *Theoretical Computer Science*, 140(1):73–94, 1995.

[GV92] J. F. Groote and F. W. Vaandrager. Structured operational semantics and bisimulation as a congruence. *Information and Computation*, 100(2):202–260, 1992.

[Hen88] M. Hennessy. Axiomatising finite concurrent processes. *SIAM J. Comput.*, 17(5):997–1017, 1988.

[HJ99] 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.

[HJM96] Y. Hirshfeld, M. Jerrum, and F. Moller. A polynomial-time algorithm for deciding bisimulation equivalence of normed Basic Parallel Processes. *Math. Struct. in Comp. Science*, 6(3):251–259, 1996.

[Hüt94] H. Hüttel. Undecidable equivalences for Basic Parallel Processes. In *Proc. Int. Symp. Theoretical Aspects of Computer Software (TACS'94), Sendai, Japan, Apr. 1994*, volume 789 of *Lecture Notes in Computer Science*, pages 454–464. Springer, 1994.

[KM99] A. Kučera and R. Mayr. Weak bisimilarity with infinite-state systems can be decided in polynomial time. In *Proc. 10th Int. Conf. Concurrency The-*

ory (CONCUR'99), Eindhoven, The Netherlands, Aug. 1999, volume 1664 of *Lecture Notes in Computer Science*, pages 368–382. Springer, 1999.

[Mil89] R. Milner. *Communication and Concurrency.* Prentice Hall Int., 1989.

[MM93] R. Milner and F. Moller. Unique decomposition of processes. *Theoretical Computer Science*, 107(2):357–363, 1993.

[Mol96] 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.

[MT91] F. Moller and C. Tofts. Relating processes with respect to speed. In *Proc. 2nd Int. Conf. Theory of Concurrency (CONCUR'91), Amsterdam, NL, Aug. 1991*, volume 527 of *Lecture Notes in Computer Science*, pages 424–438. Springer, 1991.