S. Lasota

A polynomial–time algorithm for deciding true concurrency equivalences of Basic Parallel Processes

Research Report LSV–02–13, Sep. 2002

Laboratoire Spécification et Vérification

Ecole Normale Supérieure de Cachan
61, avenue du Président Wilson
94235 Cachan Cedex France

http://www.lsv.ens–cachan.fr/Publis/
Research Report LSV–02–13, Lab. Spécification et Vérification, CNRS & ENS de Cachan, France, Sep. 2002

A polynomial-time algorithm for deciding true concurrency equivalences of Basic Parallel Processes

Sławomir Lasota*

LSV, CNRS UMR 8643 & ENS Cachan, France
and
Institute of Informatics, Warsaw University, Poland
sl@mimuw.edu.pl

Abstract. A polynomial-time algorithm is presented to decide distributed bisimilarity of Basic Parallel Processes. As a direct conclusion, several other non-interleaving semantic equivalences are also decidable in polynomial time for this class of process, since they coincide with distributed bisimilarity.

Keywords: verification of distributed programs, equivalence checking, distributed bisimilarity, process rewrite systems.

1 Introduction

One important problem in verification of concurrent systems is to check whether two given systems P and Q are equivalent under a chosen notion of equivalence. For process algebras generating infinite-state systems the equivalence checking problem cannot be decidable in general, therefore restricted classes of processes have been defined and investigated. We study here the class of Basic Parallel Processes [9] (BPP), an extension of recursively defined finite-state systems by parallel composition. Strong bisimilarity [26] is a well accepted behavioural equivalence, which often remains decidable for infinite-state systems. An elegant proof of decidability of bisimilarity for BPP and even for BPP_τ, extension of BPP by communication, was given in [10]. The PSPACE lower bound has been recently proved in [28]. On the other hand, all other equivalences in van Glabbeek's spectrum are undecidable [18, 20].

BPP is a very natural class of processes to investigate non-interleaving equivalences, intended to capture true concurrent computations of a system. One of the bisimulation-like non-interleaving equivalences is distributed bisimilarity [6], taking into account spatial distribution of a process. Already in [8] distributed bisimilarity was shown to be decidable on BPP_τ by means of a sound and complete tableau proof system. Concerning complexity, the tableau depth was only bounded exponentially.

* This work has been supported by the fellowship of the Foundation for Polish Science, during the author's post-doc stay at Laboratoire Specification et Verification, ENS Cachan. Partially supported by the KBN grant 7 T11C 002 21 and the Research Training Network *Games*.

In this paper we design a polynomial-time decision procedure for distributed bisimilarity. It strongly uses a polynomial-time algorithm for deciding strong bisimilarity on *normed* BPP processes proposed in [19]. Distributed bisimilarity is therefore very likely to be computationally more feasible than interleaving bisimilarity, in the light of the recent PSPACE lower bound for the latter one by Srba [28].

Further interesting conclusions follow from the fact that many non-interleaving equivalences coincide on BPP. As mentioned in [13], Kiehn proved in her unpublished draft [24] that *location equivalence* [7], *causal equivalence* [11] and distributed bisimilarity all coincide on CPP, a sublanguage of BPP_τ without explicit τ. Further, causal equivalence and *history preserving bisimilarity* [12, 14, 27] coincide on BPP by the result of Aceto [1]; moreover, Fröschle showed coincidence of distributed and history preserving bisimilarity on BPP [13]. Furthermore, the coincidence with *performance equivalence*, a timed bisimulation equivalence proposed by [15], can be easily proved for BPP [25].[1]

As a direct conclusion from all these results, all the mentioned equivalences can be also decided in polynomial time on BPP. Related results are in [17], where decision procedures were given for causal equivalence, location equivalence and ST-bisimulation equivalence of BPP_τ as well as for their weak versions on a subset of BPP_τ. However, complexity issues were not addressed there. Furthermore, polynomial-time complexity of performance equivalence extends the result of [5] shown only for timed BPP in a full standard form [9]. Surprisingly, polynomial-time complexity of history preserving bisimilarity on BPP can be contrasted with the EXPTIME-completeness on finite-state systems (finite 1-safe nets) [21]. Similarly, decidability of hereditary history preserving bisimilarity [4, 22] on BPP, proved in [13], can be contrasted with undecidability on finite 1-safe nets shown by Jurdziński in [23].

We start by Section 2 containing basic definitions and then we outline our algorithm in Section 3. The algorithm works for BPP processes in *standard form*, similarly as in [9]. A polynomial-time preprocessing procedure transforming a process into standard form is described in Appendix.

2 Definitions and basic facts

Let $\mathtt{Act}$ be a finite set of *actions*, ranged over by a, b, etc. and let $\mathtt{Const}$ be a finite set of *process constants*, ranged over by X, Y, etc. The set of BPP process expressions [9] over $\mathtt{Act}$ and $\mathtt{Const}$ is given by:

$$P \ ::= \ \mathbf{0} \mid X \mid a.P \mid \sum_{i\in I} P_i \mid P \| P \mid P \lfloor P$$

[1] A related result is [2], where Aceto proved that distributed bisimilarity coincides with *timed bisimilarity* [16] on BPP without recursion.

where $\mathbf{0}$ denotes the *empty* process, $a._$ is an action prefix, $\sum_{i\in I} P_i$ is a finite nondeterministic choice for a finite nonempty set I and $_\|_$ stands for a parallel composition. The only operator not present in CCS [26] is the left merge $_\lfloor_$, which differs from the parallel composition only in that the very first action must be performed in the left argument.

A BPP *process definition* Δ consists of a finite set $\mathtt{Act}(\Delta)$ of actions, a finite set $\mathtt{Const}(\Delta)$ of constants and a finite number of recursive process equations

$$X \stackrel{\text{def}}{=} P,$$

one for each constant $X \in \mathtt{Const}(\Delta)$, where P is a process expression over $\mathtt{Act}(\Delta)$ and $\mathtt{Const}(\Delta)$. Sets $\mathtt{Const}(\Delta)$ and $\mathtt{Act}(\Delta)$ are often even not mentioned explicitly, as they can be deduced from process equations. In the sequel we shall only consider *guarded* process definitions, i.e., such that each occurrence of a constant on the right-hand side of a defining equation is within the scope of action prefix. By a BPP *process* we mean a pair (P, Δ), where Δ is a BPP process definition and P is a BPP process expression over $\mathtt{Act}(\Delta)$ and $\mathtt{Const}(\Delta)$. When Δ is evident in the context, P itself is called a *process* too.

Distributed bisimilarity was introduced in [6], but here we follow the presentation of [9]. Given a BPP process definition Δ, consider the following SOS transition rules for BPP process expressions:

$$\begin{array}{ll}
\dfrac{(X \stackrel{\text{def}}{=} P) \in \Delta \qquad P \stackrel{a}{\to} [P', P'']}{X \stackrel{a}{\to} [P', P'']} & \dfrac{}{a.P \stackrel{a}{\to} [P, \mathbf{0}]} \\[2ex]
\dfrac{P_j \stackrel{a}{\to} [P', P''] \quad \text{for some } j\in I}{\sum_{i\in I} P_i \stackrel{a}{\to} [P', P'']} & \dfrac{P \stackrel{a}{\to} [P', P'']}{P \lfloor Q \stackrel{a}{\to} [P', P''\|Q]} \\[2ex]
\dfrac{P \stackrel{a}{\to} [P', P'']}{P\|Q \stackrel{a}{\to} [P', P''\|Q]} & \dfrac{Q \stackrel{a}{\to} [Q', Q'']}{P\|Q \stackrel{a}{\to} [Q', P\|Q'']}
\end{array} \tag{1}$$

We write $P \stackrel{a}{\to} [P', P'']$ if this transition can be derived from the above rules. The rules reflect a view on a process as distributed in space. Each transition $P \stackrel{a}{\to} [P', P'']$ gives rise to a *local* derivative P', which intuitively records a *location* at which the action is observed, and a *concurrent* derivative P'', recording the part of the process separated from the local component.

Two BPP processes (P_1, Δ_1) and (P_2, Δ_2) are *distributed bisimilar*, denoted $(P_1, \Delta_1) \sim (P_2, \Delta_2)$ if they are related by some *distributed bisimulation* R, that is a binary relation over BPP process expressions such that whenever $(P, Q) \in R$, for each $a \in \mathtt{Act}$,

- if $P \stackrel{a}{\to} [P', P'']$ then $Q \stackrel{a}{\to} [Q', Q'']$ for some Q', Q'' such that $(P', Q') \in R$ and $(P'', Q'') \in R$,

– if $Q \xrightarrow{a} [Q', Q'']$ then $P \xrightarrow{a} [P', P'']$ for some P', P'' such that $(P', Q') \in R$ and $(P'', Q'') \in R$.

In the next section we prove polynomial-time complexity of the problem of checking distributed bisimilarity for a given pair of constants. We do not lose generality, as checking $P \sim Q$ for arbitrary P, Q is equivalent to checking $X_P \sim X_Q$, where X_P and X_Q are new fresh constants and their defining equations are: $X_P \stackrel{\text{def}}{=} a.P$ and $X_Q \stackrel{\text{def}}{=} a.Q$, for arbitrary a. Moreover, w.l.o.g. we assume that both constants share a process definition.

Problem:	DISTRIBUTED BISIMILARITY FOR BPP
Instance:	A BPP process definition Δ and $X, Y \in \mathtt{Const}(\Delta)$
Question:	$(X, \Delta) \sim (Y, \Delta)$?

Christensen presented in [8] a sound and complete tableau proof system for $\sim$ on BPP_τ and proved an exponential upper bound for the depth of a tableau.

Theorem 1 ([8, 9]). *Distributed bisimilarity is decidable on* BPP.

Christensen [9] showed also that each BPP process definition Δ can be effectively transformed into an equivalent process definition Δ' in *standard form*, i.e., consisting exclusively of process equations in the restricted form

$$X \stackrel{\text{def}}{=} \sum_{i \in I} (a_i.P_i) \lfloor Q_i, \tag{2}$$

where all P_i and Q_i are merely a parallel composition of constants, i.e., expressions of the form

$$X_1 \| X_2 \| \ldots \| X_n, \quad \text{for } n > 0 \text{ and } X_1, \ldots, X_n \in \mathtt{Const}(\Delta'). \tag{3}$$

We omit brackets in (3) as $_\|_$ is associative and commutative w.r.t. $\sim$ (and w.r.t. any other known semantical equivalence, in fact). A parallel composition of constants (3) is called *basic process (expression)* in the sequel. Observe that processes P_i in (2) are precisely local derivatives of X and processes Q_i are precisely its concurrent derivatives. Hence the left merge operator allows here to syntactically separate both derivatives. Consequently, in the next section we will only consider basic processes as both local and concurrent derivatives of a basic process are basic again. Δ' is equivalent to Δ in the following sense: all constants of Δ are also constants of Δ' and for each $X \in \mathtt{Const}(\Delta)$, $(X, \Delta) \sim (X, \Delta')$. Moreover, since Δ is guarded, Δ' satisfies the *concurrent finiteness property*, crucial in the next section. To formulate the property, define a binary relation over $\mathtt{Const}(\Delta')$ as follows: $Y \prec_1 X$ iff Y appears in some concurrent derivative of X. Then the transitive closure $\prec_1^+$ of $\prec_1$ is irreflexive, i.e., no constant satisfies $X \prec_1^+ X$.

Christensen's transformation produces Δ' that contains only basic processes consisting of *at most* three constants. The price for this constant bound is the exponential size of Δ' w.r.t. the size of Δ, defined as the number of process equations in Δ plus the sum of lengths of right-hand sides. This is why we prove in Appendix that the transformation to the standard form (2) can be done in polynomial time. The process definition produced by our transformation also satisfies the concurrent finiteness property.

Theorem 2. *There exists a polynomial-time algorithm to transform a guarded process definition Δ into a process definition Δ' such that:*

1. *Δ' is in standard form (2),*
2. *Δ' satisfies the concurrent finiteness property,*
3. $\mathtt{Const}(\Delta) \subseteq \mathtt{Const}(\Delta')$ *and for each* $X \in \mathtt{Const}(\Delta)$, $(X, \Delta) \sim (X, \Delta')$.

(The proof of this theorem is given in Appendix.) Therefore, in the next section we assume a process definition to be in standard form and to satisfy the concurrent finiteness property.

The *strong bisimilarity* is defined w.r.t. single-derivative transitions $P \xrightarrow{a} P'$ obtained by the rules (1) when distribution of a process is ignored – for details we refer eg. to [26]. A remarkable result is that strong bisimilarity can be decided for BPP processes in polynomial time [19], but only when all the constants are *normed*. A constant, or generally a process P, is *normed* if an inactive process is reachable from P, i.e., if there is a finite sequence of transitions $P \xrightarrow{a_1} P_1 \xrightarrow{a_2} \ldots \xrightarrow{a_n} P_n$, $n \geq 0$, such that there are no further transition from P_n. The *norm* of P is the length of the shortest such sequence.

Strong bisimilarity is less restrictive than distributed bisimilarity. Hence a process definition Δ can be transformed in polynomial time into an equivalent w.r.t. strong bisimilarity process definition Δ' in a more restrictive than (2) *full standard form* [9], i.e., consisting exclusively of process equations in the form: $X \stackrel{\text{def}}{=} \sum_{i \in I} a_i.P_i$, where all P_i are basic again.

Theorem 3 ([19]). *There exists a polynomial-time algorithm to decide strong bisimilarity on normed* BPP *in full standard form.*

3 Algorithm

Throughout this section we fix Δ, and hence also sets of constants and actions. Δ is assumed to be in standard form and to satisfy the concurrent finiteness property. A more succinct representation is possible for Δ in standard form. Namely, due to associativity and commutativity of $_\|_$, it is sufficient to remember the number of occurrences of each constant in each basic expression

P_i and Q_i in right-hand side of all equations (2) in Δ, encoded in binary. In this section, complexities is measured w.r.t. the size of Δ defined as the number of process equations plus the sum of lengths of the succinct representations of the right-hand sides. Theorem 3 is still valid when this definition of size is chosen.

The reachability relation over process expressions is defined as the smallest transitive relation such that each P is related to all its local and concurrent derivatives, i.e, whenever $P \xrightarrow{a} [P', P'']$, the relation contains pairs (P, P') and (P, P''). We say that Q is *reachable* from P if the pair (P, Q) is in the reachability relation.

Let us denote by $\mathbb{P}$ the set of all process expressions reachable from the constants. As observed previously, all $P \in \mathbb{P}$ are basic. Unless stated otherwise, all the relations mentioned below are binary relations over $\mathbb{P}$. This includes also $\sim$, which is restricted now to $\mathbb{P}$ only. Unless explicitly stated otherwise, P, Q, etc. ranges over $\mathbb{P}$.

An exponential-time algorithm can be easily derived as follows. First define two monotonic operators as follows. The operator $\mathcal{B}_2$ acts on a pair of relations as follows: $(P, Q) \in \mathcal{B}_2(R', R'')$ iff for each $a \in Act$,

- if $P \xrightarrow{a} [P', P'']$ then $Q \xrightarrow{a} [Q', Q'']$ for some Q', Q'' such that $(P', Q') \in R'$ and $(P'', Q'') \in R''$,
- if $Q \xrightarrow{a} [Q', Q'']$ then $P \xrightarrow{a} [P', P'']$ for some P', P'' such that $(P', Q') \in R'$ and $(P'', Q'') \in R''$.

Then the operator $\mathcal{B}_1$ is defined by $\mathcal{B}_1(R) := \mathcal{B}_2(R, R)$. Now, define the approximating equivalences $\sim_i$ as follows:

- $P \sim_0 Q$ always,
- $\sim_{i+1} := \mathcal{B}_1(\sim_i)$, for $i \geq 0$.

Distributed bisimulations R are pre-fixed points of $\mathcal{B}_1$, i.e., are defined by $R \subseteq \mathcal{B}_1(R)$. Distributed bisimilarity being the union of all distributed bisimulations is then the greatest fixed point of $\mathcal{B}_1$, by the Knaster-Tarski theorem. Recall that BPP is image-finite, that is to say each process expression has only finitely many local and concurrent derivatives. Thus the decreasing chain $\{\sim_i\}$ converges to $\sim$:

$$\sim \; = \bigcap_{i \in \mathbb{N}} \sim_i . \tag{4}$$

Furthermore, observe that each local derivative of a basic process $X_1 \| \ldots \| X_n$ is a local derivative of some X_j, i.e., is equal to some process P_i appearing in a process equation (2) in Δ. In consequence, the number of local derivatives of all basic processes is polynomial. Let us denote the set of all those by $\mathbb{L}$. Moreover, there are only exponentially many processes reachable from each $P \in \mathbb{L}$ – this

follows easily from the concurrent finiteness property of Δ. Consequently, the cardinality N of the whole $\mathbb{P}$ is exponential. Hence $\sim$ can be computed over $\mathbb{P}$ in exponential time, e.g., as the limit of the sequence $\{\sim_i\}$ of equivalences, since the sequence stabilizes after at most $N-1$ steps.

We have not focused on details of the exponential-time algorithm, as in the rest of this section we argue that one can do better: the problem can be solved in polynomial time. Essentially, this is possible due to a "quicker" convergence to the greatest fixed point, as explained in Lemma 2 below and thereafter. Then, in the crucial Lemma 4 we reduce distributed bisimilarity to strong bisimilarity between normed processes. To this aim we incorporate local derivatives into actions and obtain single-derivative transitions. We start by a couple of definitions and simple facts.

Definition 1. *Given a binary relation* S*, a* distributed bisimulation w.r.t. S *is any binary relation* R *such that* $R \subseteq \mathcal{B}_2(S,R)$*. Expanding the definition of* $\mathcal{B}_2$*, we have: whenever* $(P,Q) \in R$*, for each* $a \in Act$*,*

- *if* $P \xrightarrow{a} [P',P'']$ *then* $Q \xrightarrow{a} [Q',Q'']$ *for some* Q', Q'' *such that* $(P',Q') \in S$ *and* $(P'',Q'') \in R$*,*
- *if* $Q \xrightarrow{a} [Q',Q'']$ *then* $P \xrightarrow{a} [P',P'']$ *for some* P', P'' *such that* $(P',Q') \in S$ *and* $(P'',Q'') \in R$*.*

P *and* Q *are* distributed bisimilar w.r.t. S*, denoted* $P \sim^S Q$*, if they are related by some distributed bisimulation w.r.t.* S*.*

Definition 2. *We say that a relation* R *is a* distributed bisimilarity w.r.t. itself *if* $R = \sim^R$*. Let* $\approx$ *denote the greatest distributed bisimilarity w.r.t. itself.*

A relation is a distributed bisimilarity w.r.t. itself precisely if it is a fixed point of the monotonic mapping $R \mapsto \sim^R$. Hence the greatest distributed bisimilarity w.r.t. itself always exists.

Lemma 1. $\sim$ *and* $\approx$ *coincide.*

Proof. For one inclusion, recall that $\sim$ is the union of all distributed bisimulations while $\sim^{\sim}$ is the union of all distributed bisimulations w.r.t. $\sim$. Since each distributed bisimulation is a distributed bisimulation w.r.t. $\sim$, we have $\sim \subseteq \sim^{\sim}$, i.e., $\sim$ is a pre-fixed point of the mapping $R \mapsto \sim^R$. As $\approx$ is the greatest fixed point of that mapping, we obtain $\sim \subseteq \approx$.

For the other inclusion, assume a relation S is a distributed bisimulation w.r.t. itself, $S = \sim^S$. Since $\sim^S$ is the union of all distributed bisimulations w.r.t. S, it is the (greatest) fixed point of the monotonic mapping $R \mapsto \mathcal{B}_2(S,R)$, i.e., $\sim^S = \mathcal{B}_2(S, \sim^S)$. Substituting S in place of $\sim^S$ we get $S = \mathcal{B}_2(S,S)$, i.e., S is a

fixed point of $\mathcal{B}_1$. Hence $S \subseteq \sim$. As S was chosen arbitrary, we showed that each distributed bisimulation w.r.t. itself is included in $\sim$. Hence also the greatest one does, i.e., $\approx \subseteq \sim$. □

Lemma 2. *$\approx = \bigcap_{i\in\mathbb{N}} \approx_i$, where the sequence $\{\approx_i\}$ is defined by:*

- $P \approx_0 Q$ *always,*
- $\approx_{i+1} := \sim^{\approx_i}$.

Proof. Obviously $\approx \subseteq \bigcap_{i\in\mathbb{N}} \approx_i$, so we only need to show the opposite inclusion. Similarly as $\sim$, $\sim^S$ is the greatest fixed point of the monotonic mapping $R \mapsto \mathcal{B}_2(S, R)$, for any fixed S. So $\sim^S$ is also a limit of a decreasing sequence of approximations:

$$\sim^S = \bigcap_{i\in\mathbb{N}} \sim^S_i, \tag{5}$$

where relations $\sim^S_i$ are defined by:

- $P \sim^S_0 Q$ always,
- $\sim^S_{i+1} := \mathcal{B}_2(S, \sim^S_i)$.

Having this, by an easy induction we show $\approx_i \subseteq \sim_i$, for all $i \geq 0$. As the induction assumption suppose $\approx_i \subseteq \sim_i$, for a fixed $i \geq 0$. Now substitute $\approx_i$ in place of S in (5) and in the definition of $\sim^S_j$, $j \geq 0$. Due to monotonicity of $\mathcal{B}_2$ we derive by another easy induction on $j \leq i$ that $\sim^{\approx_i}_{j+1} = \mathcal{B}_2(\approx_i, \sim^{\approx_i}_j) \subseteq \mathcal{B}_2(\sim_j, \sim_j) = \sim_{j+1}$, since $\approx_i \subseteq \sim_i \subseteq \sim_j$ by the induction assumption and by $j \leq i$. Hence $\sim^{\approx_i}_{i+1} \subseteq \sim_{i+1}$. By (5) we know that $\approx_{i+1} = \sim^{\approx_i} \subseteq \sim^{\approx_i}_{i+1}$, hence we conclude $\approx_{i+1} \subseteq \sim_{i+1}$. This completes the induction step.

Having $\approx_i \subseteq \sim_i$, for all $i \geq 0$, we apply Lemma 1 and (4). □

Equipped with Lemmas 1 and 2, we are ready to describe the polynomial-time algorithm. Recall that $\mathbb{L}$ denotes the set of all local derivatives. The algorithm consists of two phases (by $R \cap (\mathbb{L}\times\mathbb{L})$ we mean the restriction of a relation R to pairs from $\mathbb{L}$):

PHASE 1:
  let $\hat{=}_0 := \mathbb{L}\times\mathbb{L}$
  REPEAT FOR $n = 0, 1, \ldots$
    compute $\hat{=}_{n+1} \subseteq \mathbb{L}\times\mathbb{L}$ as follows: $\hat{=}_{n+1} := \sim^{\hat{=}_n} \cap (\mathbb{L}\times\mathbb{L})$
  UNTIL $\hat{=}_n = \hat{=}_{n+1}$
PHASE 2:
  decide whether $X \sim^{\hat{=}_n} Y$

The first phase is a crucial one, but amounts simply to computing an initial part of $\{\approx_i\}$ up to the position n where it eventually stabilizes. The only trick is that $\approx_i$ is computed only for local derivatives. Then, in the second phase we only need to check whether the input pair (X, Y) belongs to $\sim^{\approx_n}$. Assumed that the first phase of the algorithm terminates, the outcome $\sim^{\hat{=}_n}$ coincides with $\approx$:

Lemma 3. *If* $\hat{=}_n = \hat{=}_{n+1}$ *then* $\sim^{\hat{=}_n} = \approx$.

Proof. Assumed $\hat{=}_n = \hat{=}_{n+1}$, we will show $\sim^{\hat{=}_n} \subseteq \approx$ and $\sim^{\hat{=}_n} \supseteq \approx$. For both inclusions we will silently use an obvious fact: when two relations S_1 and S_2 coincide on $\mathbb{L} \times \mathbb{L}$, i.e., $S_1 \cap (\mathbb{L}\times\mathbb{L}) = S_2 \cap (\mathbb{L}\times\mathbb{L})$, then $\sim^{S_1} = \sim^{S_2}$.

$\sim^{\hat{=}_n} \subseteq \approx$: It is sufficient to show that $\sim^{\hat{=}_n}$ is a distributed bisimilarity w.r.t. itself. We have $\sim^{\sim^{\hat{=}_n}} = \sim^{\sim^{\hat{=}_n} \cap (\mathbb{L}\times\mathbb{L})} = \sim^{\hat{=}_{n+1}} = \sim^{\hat{=}_n}$.

$\sim^{\hat{=}_n} \supseteq \approx$: By induction we show that for all $i \geq 0$,

(a) $\hat{=}_i = \approx_i \cap (\mathbb{L}\times\mathbb{L})$,
(b) $\sim^{\hat{=}_i} = \approx_{i+1}$.

First, for any $i \geq 0$, (b) follows from (a): $\sim^{\hat{=}_i} \stackrel{(a)}{=} \sim^{\approx_i \cap (\mathbb{L}\times\mathbb{L})} = \sim^{\approx_i} = \approx_{i+1}$. Second, we prove (a). For $i = 0$ it is obvious, so assume $i > 0$ and (a) (and (b) as a consequence) holds for $i-1$. Then $\hat{=}_i = \sim^{\hat{=}_{i-1}} \cap (\mathbb{L}\times\mathbb{L}) \stackrel{(b)}{=} \approx_i \cap (\mathbb{L}\times\mathbb{L})$. Now $\sim^{\hat{=}_n} \supseteq \approx$ follows from (b) since $\approx_i \supseteq \approx$, for $i \geq 0$ by Lemma 2. □

Termination of the first phase of the algorithm after a polynomial number of iterations of the main loop is guaranteed as the sequence $\{\hat{=}_i\}$ is non-increasing: $\hat{=}_0 \supseteq \hat{=}_1 \supseteq \ldots$, and each $\hat{=}_i$ contains only polynomially many pairs. What we still need to show is that the single iteration of the loop body, i.e., computation of $\hat{=}_{i+1}$ from $\hat{=}_i$, can be done in polynomial time. To this aim we will prove the following Lemma 4. In the proof we will profit from Theorem 3.

Lemma 4. *Let* $S \subseteq \mathbb{L}\times\mathbb{L}$ *be an equivalence such that there exists a polynomial-time algorithm (w.r.t. the size of* Δ*) to decide whether* $(P, Q) \in S$*, for* $P, Q \in \mathbb{L}$*. Then there exists a polynomial-time algorithm to decide* $P \sim^S Q$*, for* $P, Q \in \mathbb{P}$.

Proof. As the first stage, we construct from Δ a new process definition Δ' in the full standard form, equivalent to Δ in the following sense: $P, Q \in \mathbb{P}$,

$$(P, \Delta) \sim^S (Q, \Delta) \quad \text{iff} \quad (P, \Delta') \text{ is strongly bisimilar to } (Q, \Delta'). \tag{6}$$

The construction of Δ' is as follows:

$$Const(\Delta') := Const(\Delta) \qquad Act(\Delta') := Act(\Delta) \times L/S,$$

where L/S denotes the set of abstraction classes of S and can be computed in polynomial time. Furthermore, whenever Δ contains a process equation

$$X \stackrel{\text{def}}{=} \sum_{i\in I}(a_i.P_i)\lfloor Q_i, \tag{7}$$

Δ' contains

$$X \stackrel{\text{def}}{=} \sum_{i\in I}(a_i,[P_i]_S).Q_i, \tag{8}$$

where $[P_i]_S$ denotes the abstraction class of P_i in S. Having this, (6) is clear, by the very definitions of the two bisimilarities involved. Now, the crucial point is that Δ' is always normed, due to the concurrent finiteness property of Δ. We have $Y \prec_1 X$ (cf. Section 2) iff Y appears in some Q_i on the right-hand side of process equation (7) defining X. As the transitive closure $\prec_1^+$ is irreflexive, the following equations (9) and (10) are well-defined and give the norm of all the constants. First, for each constant X defined by equation (8) in Δ',

$$\mathrm{norm}(X) = 1 + \min_{i\in I}\{\mathrm{norm}(Q_i)\}. \tag{9}$$

Second, the norm is additive w.r.t. parallel composition:

$$\mathrm{norm}(P\|Q) = \mathrm{norm}(P) + \mathrm{norm}(Q) \tag{10}$$

and this implies that the norm of each concurrent derivative Q_i in equation (9) is the sum of norms of all parallel components. Now we apply Theorem 3 to Δ' and by (6) get a polynomial-time procedure to decide $\sim^S$. □

Evidently each $\doteq_i$ is an equivalence, hence the lemma applies and we conclude that the body of the main loop of the first phase requires only polynomial time: it amounts to invoking the decision procedure for strong bisimilarity on normed BPP polynomially many times, since the set $\mathbb{L}\times\mathbb{L}$ has polynomial cardinality. By Lemma 4 the second phase can be computed in polynomial time as well. Correctness of the algorithm follows by Lemmas 3 and 1. This completes the proof of the following:

Theorem 4. *There exists a polynomial-time algorithm to decide distributed bisimilarity for* BPP *processes.*

4 Final remarks

A polynomial-time decision procedure was proposed for distributed bisimilarity on BPP. As mentioned in the introduction, many non-interleaving equivalences coincide on BPP. Therefore, we directly conclude from Theorem 4:

Corollary 1. *There exists a polynomial-time algorithm to decide the following equivalences for* BPP *processes: location equivalence, causal equivalence, history preserving equivalence, performance equivalence.*

Note that Theorem 4 does not hold for BPP_τ, extension of BPP with communication between parallel components expressed by one additional rule:

$$\frac{P \xrightarrow{a} [P', P''] \qquad Q \xrightarrow{\bar{a}} [Q', Q'']}{P \| Q \xrightarrow{\tau} [P' \| Q', P'' \| Q'']}.$$

The reduction of distributed bisimilarity to strong bisimilarity in the proof of Lemma 4 fails for BPP_τ. In the case of a τ-transition, a local derivative can be composed of two local derivatives of the parallel components. Hence local derivatives cannot be encoded directly into actions as it was done in the proof of Lemma 4.

A crucial ingredient of our decision procedure is the polynomial-time transformation of a process definition to the standard form, described in Appendix. It is different from the transformation proposed by Christensen in [9], since the process definition in standard form yielded by the latter is of exponential size.

Our algorithms needs $\Theta(n^2)$ calls to the polynomial-time algorithm of [19] in each iteration in the first phase, where n stands for the size of Δ. At most n iterations are needed, since all $\doteq_i$ are equivalence, and therefore total cost is $\Theta(n^3)$ calls to the procedure of [19]. On the other hand, P-completeness of the problem follows easily since it subsumes strong bisimilarity for finite-state systems and the latter is P-complete [3].

Acknowledgements The author is very grateful to Philippe Schnoebelen for many fruitful discussions.

References

1. L. Aceto. History preserving, causal and mixed-ordering equivalence over stable event structures. *Fundamenta Informaticae*, 17:319–331, 1992.
2. L. Aceto. Relating distributed, temporal and causal observations of simple processes. *Fundamenta Informaticae*, 17:369–397, 1992.
3. J. Balcázar, J. Gabarró, and M. Sántha. Deciding bisimilarity is P-complete. *Formal Aspects of Computing*, (6A):638–648, 1992.
4. M. Bednarczyk. Hereditary history preserving bisimulation or what is the power of the future perfect in program logics. Technical report, Polish Academy of Sciences, Gdańsk, 1991.
5. B. Bérard, A. Labroue, and P. Schnoebelen. Verifying performance equivalence for timed Basic Parallel Processes. In *Proc. 3rd Int. Conf. Foundations of Software Science and Computation Structures (FOSSACS'2000), LNCS 1784*, pages 35–47, 2000.
6. I. Castellani. *Bisimulations for Concurrency*. PhD thesis, University of Edinburg, 1988.

7. I. Castellani. Process algebras with localities. In *J. Bergstra, A. Ponse, S. Smolka, eds., Handbook of Process Algebra, chapter 15*, pages 945–1046, 2001.
8. S. Christensen. Distributed bisimilarity is decidable for a class of infinite state systems. In *Proc. 3th Int. Conf. Concurrency Theory (CONCUR'92), LNCS 630*, pages 148–161, 1992.
9. S. Christensen. *Decidability and Decomposition in process algebras*. PhD thesis, Dept. of Computer Science, University of Edinbourg, UK, 1993. PhD Thesis CST-105-93.
10. S. Christensen, Y. Hirshfeld, and F. Moller. Bisimulation equivalence is decidable for Basic Parallel Processes. In *Proc. 4th Int. Conf. Concurrency Theory (CONCUR'93), LNCS 713*, pages 143–157, 1993.
11. P Darondeau and P. Degano. Causal trees. In *Proc. ICALP'89, LNCS 372*, pages 234–248, 1989.
12. P. Degano, R. De Nicola, and U. Montanari. Partial ordering descriptions and observations of nondeterministic concurrent processes. In *Proc. REX School/Workshop on Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency, LNCS 354*, pages 438–466, 1989.
13. S. Fröschle. Decidability of plain and hereditary history-preserving bisimulation for BPP. In *Proc. 6th International Workshop on Expressiveness in Concurrency, EXPRESS'99, volume 27 of ENTCS*, 1999.
14. R. van Glabbeek and U. Goltz. Equivalence notions for concurrent systems and refinement of actions. In *Proc. Mathematical Foundations of Computer Science (MFCS'89), LNCS 379*, pages 237–248, 1989.
15. R. Gorrieri, M. Roccetti, and E. Stancampiano. A theory of processes with durational actions. *Theoretical Computer Science*, 140(1):73–94, 1995.
16. M. Hennesy. Axiomatizing finite concurrent processes. *SIAM Journal on Computing*, 17(5):997–1017, 1988.
17. M. Hennesy and A. Kiehn. On the decidability of non-interleaving process equivalences. In *Proc. 5th Int Conf. Concurrency Theory (CONCUR'94)*, pages 18–33, 1994.
18. Y. Hirshfeld. Petri nets and the equivalence problem. In *Proc. Computer Science Logic (CSL'93), LNCS 832*, pages 165–174, 1993.
19. Y. Hirshfeld, M. Jerrum, and F. Moller. A polynomial time algorithm for deciding bisimulation equivalence of normed basic parallel processes. *Mathematical Structures in Computer Science*, 6:251–259, 1996.
20. H. Hüttel. Undecidable equivalences for basic parallel processes. In *Proc. TACS'94, LNCS 789*, pages 454–464, 1994.
21. L Jategaonkar and A. R. Meyer. Deciding true concurrency equivalences on safe, finite nets. *Theoretical Computer Science*, 154:107–143, 1996.
22. A. Joyal, M. Nielsen, and G. Winskel. Bisimulation from open maps. *Information and Computation*, 127:164–185, 1996.
23. M. Jurdziński and M. Nielsen. Hereditary history preserving bisimilarity is undecidable. In *Proc. 17th International Symposium on Theoretical Aspects of Computer Science (STACS'00), LNCS 1770*, 358-369.
24. A. Kiehn. A note on distributed bisimulations. Unpublished draft, 1999.
25. S. Lasota. On coincidence of distributed and performance equivalence for Basic Parallel Processes. `http://www.mimuw.edu.pl/~sl/papers/distperf.ps.gz`, unpublished draft, 2002.
26. R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.
27. A. Rabinowich and B. A. Trakhtenbrot. Behaviour structures and nets. *Fundamenta Informaticae*, 11:357–403, 1988.
28. J. Srba. Strong bisimilarity and regularity of Basic Parallel Processes is PSPACE-hard. In *Proc. 19th International Symposium on Theoretical Aspects of Computer Science (STACS'02), LNCS 2285*, 2002.

A Standard form

We show that each guarded BPP process definition Δ can be effectively transformed into an equivalent process definition Δ' consisting exclusively of process equations in the following form

$$X \stackrel{\text{def}}{=} \sum_{i \in I} (a_i.X_i) \lfloor (X_{i,1} \| X_{i,2} \| \ldots \| X_{i,n_i}), \tag{11}$$

$X_i, X_{i,j} \in \mathtt{Const}(\Delta')$. Thus, each summand of the right-hand side of each equation in Δ' is left merge applied to a prefixed constant and a basic process. The transformation requires only polynomial time w.r.t. the size of Δ, defined as the number of process equations in Δ plus the sum of lengths of right-hand sides. Moreover, Δ' satisfies the concurrent finiteness property and is equivalent to Δ in the following sense: $\mathtt{Const}(\Delta) \subseteq \mathtt{Const}(\Delta')$ and for each $X \in \mathtt{Const}(\Delta)$, $(X, \Delta) \sim (X, \Delta')$.

Before focusing on details, let us inspect a relationship between (11) and the standard form of Christensen [9]. First, both are special cases of the standard form (2) from Section 2, that is sufficient for the development of the paper. Furthermore, (11) is slightly more restrictive: local derivatives are occurrences of a single constant instead of arbitrary basic processes as in (2). The Christensen's standard form is also more restrictive than (2): both local and concurrent derivatives are basic processes consisting of *at most* three constants. Moreover, the latter standard form works for BPP_τ as well. The price to pay for the constant bound on the size of local and concurrent derivatives is that the size of the process definition in the standard form is exponential w.r.t. the size of Δ.

The overall idea of the transformation comes from [9], but is essentially modified here. The transformation consist of several phases, and in each phase one more constant is given its new defining equation. Initially, all process equations from Δ belong to an auxiliary process definition $\widehat{\Delta}$ and Δ' is empty:

$$\widehat{\Delta} := \Delta \qquad\qquad \Delta' := \emptyset.$$

In each phase, one equation $X \stackrel{\text{def}}{=} P$ will be removed from $\widehat{\Delta}$ and a new equation for constant X of the form (11) will be added to Δ'. Additionally, possibly a number of new constants X_P will be introduced and equations $X_P \stackrel{\text{def}}{=} P$ will be added to $\widehat{\Delta}$. In such case P is always a subexpression of the right-hand side of some equation from the original process definition Δ. The equation $X_P \stackrel{\text{def}}{=} P$ will be added to $\widehat{\Delta}$ only if constant X_P is not yet defined in $\widehat{\Delta}$ or Δ' – this guarantees termination, when $\widehat{\Delta} = \emptyset$, after a polynomial number of phases.

We deliberately do not distinguish between processes related by a *structural congruence*, i.e., the smallest congruence induced by:

- commutativity and associativity of parallel composition: $P\|Q = Q\|P$, $P\|(Q\|R) = (P\|Q)\|R$,
- nilpotency of $\mathbf{0}$ w.r.t. the parallel composition: $\mathbf{0}\|P = P$.

From now on, equality of processes in understood up to the structural congruence.

Before explaining a single phase in detail, we need some notation. For each process expression P we will define a set $\mathrm{deriv}(P)$ containing triples of the form $\langle a, X_Q, X_{Q_1}\|\ldots\|X_{Q_n}\rangle$ such that the following two implications hold:

$$\begin{aligned} \langle a, X_Q, X_{Q_1}\|\ldots\|X_{Q_n}\rangle \in \mathrm{deriv}(P) \;\Rightarrow\; P \xrightarrow{a} [Q, Q_1\|\ldots\|Q_n] \\ P \xrightarrow{a} [Q, Q'] \Rightarrow \text{ there are } Q_1,\ldots,Q_n \text{ such that } Q' = Q_1\|\ldots\|Q_n \wedge \\ \langle a, X_Q, X_{Q_1}\|\ldots\|X_{Q_n}\rangle \in \mathrm{deriv}(P) \end{aligned} \tag{12}$$

This simply means that $\mathrm{deriv}(P)$ contains precisely the full information about all derivatives of P. We define $\mathrm{deriv}(_)$ by structural induction:

$$\mathrm{deriv}(\mathbf{0}) := \emptyset \tag{13}$$

$$\mathrm{deriv}(a.P) := \{\, \langle a, X_P, \mathbf{0}\rangle \,\} \tag{14}$$

$$\mathrm{deriv}(\textstyle\sum_{i\in I} P_i) := \bigcup_{i\in I} \mathrm{deriv}(P_i) \tag{15}$$

$$\mathrm{deriv}(P\lfloor Q) := \{\, \langle a, P', P''\|X_Q\rangle \;:\; \langle a,\ P', P''\rangle \in \mathrm{deriv}(P) \,\} \tag{16}$$

$$\mathrm{deriv}(P\|Q) := \mathrm{deriv}(P\lfloor Q) \cup \mathrm{deriv}(Q\lfloor P) \tag{17}$$

$$\mathrm{deriv}(X) := \mathrm{deriv}(P), \quad \text{when } (X \stackrel{\mathrm{def}}{=} P) \in \Delta \tag{18}$$

We note that in (18) a process equation $(X \stackrel{\mathrm{def}}{=} P)$ from Δ is used, even if X has already its defining equation in Δ'. New constants X_P are introduced in (14) and (16) to replace subexpression P of the right-hand side of some equation in Δ. To keep the algorithm below uniform and simple, this includes also the "degenerate" cases: $P = \mathbf{0}$ or $P = Y$, for some $Y \in \mathtt{Const}(\Delta)$.

Note that (12) always holds, whenever P is a subexpression of the right-hand side of some equation in Δ, but only up to the structural congruence defined above.

Now we are ready to outline the algorithm of the transformation. A single phase corresponds to execution of the body of the main loop below. Note that the constant X whose defining equation is removed from $\widehat{\Delta}$ in $(*)$ below can be either a constant from Δ or some of constants X_P added to $\widehat{\Delta}$ in previous iterations. In the latter case, the equation removed in $(*)$ is always of the form $X_P \stackrel{\mathrm{def}}{=} P$.

$\hat{\Delta} := \Delta, \quad \Delta' := \emptyset$
REPEAT
choose an arbitrary equation $(X \stackrel{\text{def}}{=} P) \in \hat{\Delta}$ and remove it from $\hat{\Delta}$ $(*)$
compute deriv(P)
add $(X \stackrel{\text{def}}{=} \sum_{\langle a, P', P'' \rangle \in \text{deriv}(P)} (a.P') \lfloor P'')$ to Δ'
FOR EACH Q such that X_Q appears in deriv(P) and $X_Q \notin \mathtt{Const}(\hat{\Delta}) \cup \mathtt{Const}(\Delta')$
add $(X_Q \stackrel{\text{def}}{=} Q)$ to $\hat{\Delta}$
UNTIL $\hat{\Delta} = \emptyset$

Theorem 5. *There exists a polynomial-time algorithm to transform a guarded process definition Δ into a process definition Δ' such that:*

1. *Δ' is in standard form (11),*
2. *Δ' satisfies the concurrent finiteness property (cf. Section 2),*
3. *$\mathtt{Const}(\Delta) \subseteq \mathtt{Const}(\Delta')$ and for each constant $X \in \mathtt{Const}(\Delta)$, it holds $(X, \Delta) \sim (X, \Delta')$.*

Proof. For the correctness of the structural rules (13-18) underlying the algorithm, observe that for any newly introduced constant X_Q, it is never required to compute deriv(X_Q). This is due to the following invariant that holds continuously after each phase in the course of computation: each right-hand side of any equation in $\hat{\Delta}$ is a subexpression of the right-hand side of some equation in Δ. The invariant can be easily proved by induction on the number of phases performed, since only process equations from Δ are relevant in (18).

Concerning complexity, the algorithm terminates when $\hat{\Delta} = \emptyset$. The number of phases is polynomial, since the number of different X_Q, being equal to the number of different subexpressions of right-hand sides of Δ, is polynomially bounded. Moreover, each set deriv(P) computed in each phase has polynomial cardinality. This can be easily shown by two easy structural induction proofs directly from the definition (13-18). We remind that deriv$(_)$ is applied during the computation only to subexpressions of the right-hand sides of equations in Δ. The first induction is to show that deriv(X) is always polynomial, for $X \in \mathtt{Const}(\Delta)$. Note that in the induction step we don't need to consider a constant since Δ is guarded by assumption. The second induction proof is to show that deriv(P) is polynomial for arbitrary P, and now as the induction assumption we use the already-proved fact that deriv(X) is polynomial.

Therefore each new process equation added to Δ' is of polynomial size and each phase requires only polynomial time.

Now we prove points 1-3 for the process definition Δ' computed by the algorithm. First, Δ' is in standard form by the very construction. The concurrent

finiteness property can be easily proved from guardedness of Δ as follows. First observe that whenever $Y \prec_1 X$, necessarily $Y = X_Q$ for some Q. Hence $X \prec_1^+ X$ is impossible for $X \in \mathit{Const}(\Delta)$. For $X, Y \in \mathit{Const}(\Delta')$, if $Y \prec_1 X$ holds then

1. either $X \in \mathit{Const}(\Delta)$, $Y = X_Q$, for some Q, and Q is an unguarded subexpression of the defining equation of X in Δ,
2. either $X = X_P$, $Y = X_Q$, for some P, Q, and Q is an unguarded subexpression of P,
3. or $X = X_P$, $Y = X_Q$, for some P, Q, and there is $Z \in \mathit{Const}(\Delta)$ occurring unguarded in P such that Q is an unguarded subexpression of the defining equation of Z.

Assume $X_Q \prec_1^+ X_Q$, i.e., X_Q is related to itself by a finite nonempty sequence of pairs related by $\prec_1$. At least one of these pairs must satisfy point (3) above. Assume first that there is precisely one such pair. This means that some intermediate $Z \in \mathit{Const}(\Delta)$ appears unguarded in the defining equation of X_Q, i.e., in Q, and also Q is an unguarded subexpression of the defining equation of Z in Δ. But this would clearly lead to a contradiction, i.e., Z would occur unguarded in its own defining equation in Δ. Moreover, if there are two such pairs, this would mean that one of the intermediate constants appears unguarded in the the definition of the other intermediate constant in Δ, which leads also to a contradiction.

Concerning point 3, it is routine to check by the very definition of $\mathrm{deriv}(_)$ that (12) holds in each phase, for P being the right-hand side of the equation removed in $(*)$. Thus, it can be proved by induction on the number of phases that it continuously holds after each phase:

- $(X, \Delta) \sim (X, \widehat{\Delta} \cup \Delta')$, for each constant X from Δ,
- $(P, \Delta) \sim (X_P, \widehat{\Delta} \cup \Delta')$, for each constant $X_P \in \mathit{Const}(\widehat{\Delta}) \cup \mathit{Const}(\Delta')$.

(Note that it never happens in the course of computation that a constant has a defining equation both in $\widehat{\Delta}$ and Δ'. Therefore $\widehat{\Delta} \cup \Delta'$ is a correct process definition.) Hence after termination, i.e., when $\widehat{\Delta} = \emptyset$, we get $(X, \Delta) \sim (X, \Delta')$, for each $X \in \mathit{Const}(\Delta)$. $\square$