AVIS 2005 Preliminary Version

On distributed bisimilarity over Basic Parallel Processes [1]

Petr Jančar [2], Zdeněk Sawa [3]

Center for Applied Cybernetics,
Dept of Computer Science, Technical University of Ostrava
17. listopadu 15, 708 33 Ostrava-Poruba, Czech Republic

Abstract

Distributed bisimilarity is one of non-interleaving equivalences studied on concurrent systems; it refines the classical bisimilarity by taking also the spatial distribution of (sub)components into account. In the area of verification of infinite-state systems, one of the simplest (most basic) classes is the class of Basic Parallel Processes (BPP); here distributed bisimilarity is known to coincide with many other non-interleaving equivalences. While the classical (interleaving) bisimilarity on BPP is known to be PSPACE-complete, for distributed bisimilarity a polynomial time algorithm was shown by Lasota (2003). Lasota's algorithm is technically a bit complicated, and uses the algorithm by Hirshfeld, Jerrum, Moller (1996) for deciding bisimilarity on normed BPP as a subroutine. Lasota has not estimated the degree of the polynomial for his algorithm, and it is not an easy task to do. In this paper we show a direct and conceptually simpler algorithm, which allows to bound the complexity by $O(n^3)$ (when starting from the normal form used by Lasota).

Key words: verification, equivalence checking, distributed bisimilarity, basic parallel processes

1 Introduction

Finding efficient algorithms for various verification problems is of ever growing importance. One category of such problems is 'equivalence checking', where the aim is to verify that two systems (processes) are behaviorally equivalent. The notion of behaviour depends heavily on the context, and a plethora of behavioural equivalences have been proposed and studied. We can refer

[1] Supported by the Czech Ministry of Education, Grant No. 1M6840770004, and by the Grant Agency of the Czech Republic, Grant No. 201/03/1161.
[2] e-mail: `petr.jancar@vsb.cz`
[3] e-mail: `zdenek.sawa@vsb.cz`

to [10] for a survey of so called interleaving equivalences (where concurrency is modelled by interleaving and nondeterministic choice). If we have reasons to take 'true concurrency' into account, we can resort to various *non-interleaving* equivalences. As examples of these equivalences we can name distributed bisimilarity [1], location equivalence [2], causal equivalence [4], history preserving bisimilarity [11], or performance equivalence [5].

The properties and decision problems for behavioural equivalences have been studied on various classes of systems, including infinite-state systems. One of the simplest (most basic) classes of infinite state systems is the class of Basic Parallel Processes (BPP for short), introduced by Christensen in [3].

Since (interleaving) bisimilarity is known to be PSPACE-complete [7], the question of polynomial time algorithms for non-interleaving semantics is of particular interest. One can concentrate on distributed bisimilarity since it is known to coincide with many other non-interleaving equivalences on BPP (see [9] for references).

Already Christensen [3] showed that the problem of deciding distributed bisimilarity on BPP is decidable. The first polynomial time algorithm for the problem was shown by Lasota in [9], which has been the main motivating paper for us. Lasota's algorithm is technically a bit complicated, and uses the algorithm from [6] for deciding bisimilarity on normed BPP as a subroutine. Lasota has not estimated the degree of the polynomial for his algorithm, and it is not an easy task to do.

Remark: No complexity bound was given in [6] (for the problem of deciding strong bisimilarity on normed BPP) but it seems to be $\Omega(n^5)$. Even if we use the more efficient algorithm with time complexity in $O(n^3)$ [8] as a subroutine, it seems to result (just) in the estimation $O(n^5)$ for the whole Lasota's algorithm.

In this paper we show a direct and conceptually simpler algorithm, which allows to bound the complexity by $O(n^3)$, when starting from the normal form used by Lasota. The improvement is mainly inspired by the technique introduced in [7].

Section 2 contains basic definitions, Section 3 presents some remarks to the normal form (used by Lasota) and is completed by an Appendix; Section 4 presents the main result of the paper – a new efficient polynomial time algorithm for deciding distributed bisimilarity on BPP. Section 5 contains some final remarks.

2 Definitions

Let $Act = \{a, b, c, \ldots\}$ be a countably infinite set of atomic actions and let $Var = \{X, Y, Z, \ldots\}$ be a countably infinite set of process variables. The class of BPP expressions over Act and Var is defined by the following abstract syntax:

$$P \ ::= \ \mathbf{0} \ \mid \ X \ \mid \ a.P \ \mid \ P_1 + P_2 \ \mid \ P_1 \parallel P_2$$

where $\mathbf{0}$ denotes the empty process, X is a process variable, $a.\_$ is an action prefix, $\_ + \_$ denotes non-deterministic choice, and $\_ \parallel \_$ parallel composition.

A *BPP process definition* is a finite family of recursive equations

$$\Delta = \{X_i \overset{\text{def}}{=} P_i \mid 1 \leq i \leq n\}$$

where all X_i are distinct and all P_i are BPP expressions where every occurrence of a variable in P_i is *guarded*, i.e., it is within the scope of an action prefix. (This guarantees that the transition system induced by the rules below is finitely branching.) The sets of actions and variables occurring in Δ are denoted $Act(\Delta)$ and $Var(\Delta)$, respectively. A *BPP process* is a pair (P, Δ) where Δ is a BPP process definition and P is a process expression containing only actions and variables from $Act(\Delta)$ and $Var(\Delta)$. We usually write just P instead of (P, Δ) when Δ is obvious from the context.

Distributed bisimilarity, which we want to define as a binary relation over BPP processes, should take the (spatial) distribution of (sub)processes into account. So we have to look for a suitable modification of the standard (interleaving) semantics. A natural candidate for such a modification is to view each transition (from a process) as going to a *pair* of processes, the *local derivative* and the *concurrent derivative.*
Remark: The intuition behind this definition is that processes are distributed in space, and local and concurrent derivatives are two parts of the whole process. Local derivative records a location at which the action is observed, and concurrent derivative records the rest of the process.

We write transitions as $P \xrightarrow{a} [P', P'']$ where P is the original process, P' and P'' are its local and concurrent derivatives, and a is the performed action.

Given a fixed BPP process definition Δ, the transitions are induced by the following SOS (structural operational semantics)-rules

$$\frac{}{a.P \xrightarrow{a} [P, \mathbf{0}]} \qquad \frac{P_j \xrightarrow{a} [P', P''] \text{ for some } j \in I}{\sum_{i \in I} P_i \xrightarrow{a} [P', P'']}$$

$$\frac{P \xrightarrow{a} [P', P'']}{P \parallel Q \xrightarrow{a} [P', P'' \parallel Q]} \qquad \frac{Q \xrightarrow{a} [Q', Q'']}{P \parallel Q \xrightarrow{a} [Q', P \parallel Q'']}$$

$$\frac{P \xrightarrow{a} [P', P'']}{X \xrightarrow{a} [P', P'']} ((X \overset{\text{def}}{=} P) \in \Delta)$$

A relation $\mathcal{R}$ (on the set of processes induced by Δ) is a *distributed bisimulation* iff for each $(P, Q) \in \mathcal{R}$ and each $a \in Act$ two following conditions hold:

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

Processes P and Q are *distributed bisimilar*, denoted $P \sim Q$, iff there is a distributed bisimulation $\mathcal{R}$ such that $(P, Q) \in \mathcal{R}$. The relation $\sim$ is called *distributed bisimulation equivalence* or *distributed bisimilarity.*

We consider the following problem called BPP-DBISIM in this paper:

INSTANCE: A BPP process definition Δ and two variables $X, Y \in \mathit{Var}(\Delta)$.
QUESTION: Is $X \sim Y$?

Remark: We can use this definition without losing generality because a more general problem where we have two process expressions P_1 and P_2 over a process definition Δ, and the question is whether $P_1 \sim P_2$, can be easily transformed to BPP-DBISIM.

3 Normal form

Lasota [9] uses a special normal form for BPP processes, a modification of the form used by Christensen [3]. It is useful for us as well.

We first observe that non-deterministic choice $(\_ + \_)$ and parallel composition $(\_ \parallel \_)$ are associative and commutative with respect to $\sim$, so we can work with them modulo associativity and commutativity.

Processes of the form $X_1 \parallel X_2 \parallel \cdots \parallel X_n$, where $n \geq 0$ and each $X_i \in \mathit{Var}$, are called *basic processes*. We can view $\mathbf{0}$ as the basic process with $n = 0$.

For the normal form we also need an auxiliary new process operator, called *left merge*:

$$P_1 \mathbin{\|\!\!\lfloor} P_2$$

is similar to parallel composition $P_1 \parallel P_2$ but P_1 (the left component) must perform an action first. The relevant SOS rule is:

$$\frac{P \xrightarrow{a} [P', P'']}{P \mathbin{\|\!\!\lfloor} Q \xrightarrow{a} [P', P'' \parallel Q]}$$

The point is that every BPP process definition Δ can be transformed into equivalent (distributed bisimilar) *normal form* Δ' with all equations of the form

$$X \stackrel{\text{def}}{=} \sum_{i \in I} ((a_i.P_i) \mathbin{\|\!\!\lfloor} Q_i)$$

where all P_i and Q_i are basic processes. We note that the variables in Q_i are syntactically unguarded. But since we start from a guarded (standard) BPP process definition, it will be guaranteed that the following relation $\prec$ is *irreflexive* (and is thus a strict order):

We first define relation $\prec_1 \subseteq \mathit{Var}(\Delta') \times \mathit{Var}(\Delta')$ such that $Y \prec_1 X$ holds iff $X \stackrel{\mathrm{def}}{=} \sum_{i\in I}((a.P_i) \mathbin{|\!\lfloor} Q_i)$ and Y occurs in some Q_i; then we take $\prec$ as the transitive closure of $\prec_1$. For technical convenience, we can extend $\prec$ to some arbitrary (but further fixed) linear order. In the next section relation $\prec$ represents such a linear order.

Lasota sketched a transformation into normal form. In the Appendix, we provide a more detailed description of such a transformation, which is more suitable for complexity analysis. It can be shown that the transformation can be done in $O(n^4)$, but unfortunately the size of the process definition is $\Theta(n^4)$ in the worst case.

In the next section we provide an algorithm for BPP-DBISIM working in $O(n^3)$, assuming the input is in the normal form.

We finish this section by rewriting the normal form in a notation which is more convenient for description of the algorithm.

Due to associativity and commutativity of parallel composition $\_ \parallel \_$, the order of variables in a basic process is not important, and so we can identify a basic process with a multiset of variables. For example,

$$X \parallel X \parallel X \parallel X \parallel X \parallel X \parallel X \parallel Y \parallel Y \parallel Z \parallel Z \parallel Z$$

can be represented as $\{7\,X, 2\,Y, 3\,Z\}$.

We use $\mathit{Var}^{\oplus}$ to denote the set of all multisets of $\mathit{Var}(\Delta)$. For $P \in \mathit{Var}^{\oplus}$ and $X \in \mathit{Var}$ we use $P(X)$ to denote the number of occurrences of X in P. The relation $\geq$ on $\mathit{Var}^{\oplus}$ is defined as follows: $P \geq Q$ iff $P(X) \geq Q(X)$ for every $X \in \mathit{Var}(\Delta)$. We use $P \oplus Q$ to denote the union of $P, Q \in \mathit{Var}^{\oplus}$, i.e., $(P \oplus Q)(X) = P(X) + Q(X)$ for each $X \in \mathit{Var}(\Delta)$. We use $P \ominus Q$ to denote the difference of $P, Q \in \mathit{Var}^{\oplus}$ such that $P \geq Q$, i.e., $(P \ominus Q)(X) = P(X) - Q(X)$ for each $X \in \mathit{Var}(\Delta)$.

We represent a BPP process definition Δ in the normal form as a finite set of *rules* of the form

$$X \xrightarrow{a} (P, Q)$$

where $X \in \mathit{Var}(\Delta)$, $a \in \mathit{Act}$, and $P, Q \in \mathit{Var}^{\oplus}$. Each original rule

$$X \stackrel{\mathrm{def}}{=} \sum_{1\leq i\leq m} ((a_i.P_i) \mathbin{|\!\lfloor} Q_i)$$

gives rise to new rules $X \xrightarrow{a_1} (P_1, Q_1), X \xrightarrow{a_2} (P_2, Q_2), \ldots, X \xrightarrow{a_m} (P_m, Q_m)$.

For example, a BPP process definition

$$\begin{aligned}
X &\stackrel{\mathrm{def}}{=} (a.XY \mathbin{|\!\lfloor} \mathbf{0}) + (b.\mathbf{0} \mathbin{|\!\lfloor} \mathbf{0}) \\
Y &\stackrel{\mathrm{def}}{=} a.XZ \mathbin{|\!\lfloor} XX \\
Z &\stackrel{\mathrm{def}}{=} (a.YZ \mathbin{|\!\lfloor} XXXY) + (a.X \mathbin{|\!\lfloor} XY)
\end{aligned}$$

(where $X \prec Y \prec Z$) is represented by the following set of rules:

$$X \xrightarrow{a} (XY, \mathbf{0}) \qquad X \xrightarrow{b} (\mathbf{0}, \mathbf{0}) \qquad Y \xrightarrow{a} (XZ, XX)$$
$$Z \xrightarrow{a} (YZ, XXXY) \qquad\qquad Z \xrightarrow{a} (X, XY)$$

(For clarity we do not use multiset representations in examples.)

For each $t = (X \xrightarrow{a} (P, Q)) \in \Delta$, we define $\textsc{pre}(t) = X$ and $\lambda(t) = a$. We write $P \xrightarrow{t} [P', P'']$, for a rule $t = (X \xrightarrow{a} (P', Q'))$, iff $P'' = (P \ominus \{X\}) \oplus Q'$.

For example, if t is a rule $Z \xrightarrow{a} (YZ, XXXY)$ in Δ, then

$$XXXYZZ \xrightarrow{t} [YZ, XXXXXXYYZ]$$

is a possible transition.

We use $\mathbb{N}$ to denote the set of non-negative integers.

4 New algorithm

Let us assume that we have an input instance of BPP-DBISIM consisting of some BPP process definition Δ *in normal form* and two variables $X, Y \in \mathit{Var}(\Delta)$. We assume that Δ is given by a set of rules of the form

$$X \xrightarrow{a} (P, Q)\,.$$

(We suppose that P and Q are represented is such a way that the representation contains the numbers of occurrences of variables in P and Q encoded in binary. This is important for the complexity analysis.) We assume that $\mathit{Var}(\Delta) = \{X_1, X_2, \dots, X_k\}$ and $X_1 \prec X_2 \prec \cdots \prec X_k$ for the linear order $\prec$ discussed in the previous section.

The main idea of the algorithm is to construct a sequence of approximations of distributed bisimilarity $\sim$ from above. The algorithm is inspired by the ideas introduced in [7].

4.1 Preliminaries

Before going into details, we need some technical definitions. Let $\mathcal{D}$ be some (non-empty) set and let $f : \mathit{Var}^{\oplus} \to \mathcal{D}$ be a function. We say that f is *bisimulation invariant* iff for every $P, Q \in \mathit{Var}^{\oplus}$ $P \sim Q$ implies $f(P) = f(Q)$. The notion 'bisimulation invariant' can be used also for predicates over processes from $\mathit{Var}^{\oplus}$, because a predicate can be viewed as a function $f : \mathit{Var}^{\oplus} \to \mathcal{D}$ where $\mathcal{D} = \{0, 1\}$.

Rule $t \in \Delta$ is *disabled* in $P \in \mathit{Var}^{\oplus}$ iff $P(\textsc{pre}(t)) = 0$, and it is *enabled* in P iff $P(\textsc{pre}(t)) > 0$. For $T \subseteq \Delta$ we define predicates $\textsc{disabled}(T)$ and $\textsc{enabled}(T)$ on elements of $\mathit{Var}^{\oplus}$ such that $\textsc{disabled}(T)(P)$ holds iff every $t \in T$ is disabled in P, and $\textsc{enabled}(T)(P)$ iff $\neg\textsc{disabled}(T)(P)$.

Let $a \in Act$ be an action and let $T_a = \{t \in \Delta \mid \lambda(t) = a\}$. Note that $\textsc{disabled}(T_a)$ is bisimulation invariant for every $a \in Act$.

Given a set $T \subseteq \Delta$, the *norm* of T, denoted $\textsc{norm}(T)$, is a function $\textsc{norm}(T) : Var^{\oplus} \to \mathbb{N}$ defined for $P \in Var^{\oplus}$ so that the value of $\textsc{norm}(T)(P)$ is the length of the shortest sequence $Q_0, Q_1, \ldots, Q_m$ of basic processes (the length of the sequence is m) where $Q_0 = P$, $\textsc{disabled}(T)(Q_m)$, and for each $1 \le i \le m$ there are some a_i and P_i such that $Q_{i-1} \xrightarrow{a_i} [P_i, Q_i]$.

It will be shown that if $\textsc{disabled}(T)$ is bisimulation invariant then also $\textsc{norm}(T)$ is bisimulation invariant. Moreover, it will be shown that for any $T \subseteq \Delta$, the function $\textsc{norm}(T)$ can be expressed as a *linear function*, i.e., as a function $L : Var^{\oplus} \to \mathbb{N}$ of the form

$$L(P) = \sum_{i=1}^{k} c_i \cdot P(X_i)$$

for $P \in Var^{\oplus}$, where each *coefficient* $c_i \in \mathbb{N}$ can be computed efficiently.

4.2 Algorithm

The algorithm creates a set of linear functions $\mathcal{L}$ such that each linear function L from $\mathcal{L}$ will be the norm of some set of rules $T \subseteq \Delta$ such that $\textsc{disabled}(T)$ is bisimulation invariant. Because of this, each L from $\mathcal{L}$ will be also bisimulation invariant, and therefore $L(P) \neq L(Q)$ implies $P \not\sim Q$. Moreover, $\mathcal{L}$ will be constructed in such a way that if $\forall L \in \mathcal{L} : L(P) = L(Q)$ then $P \sim Q$. More formally we define for $\mathcal{L}$ an equivalence relation $\equiv_{\mathcal{L}}$ on $Var^{\oplus}$ such that $P \equiv_{\mathcal{L}} Q$ iff $\forall L \in \mathcal{L} : L(P) = L(Q)$, and we show that $P \sim Q$ iff $P \equiv_{\mathcal{L}} Q$.

The algorithm starts with $\mathcal{L} = \emptyset$ and successively adds functions to $\mathcal{L}$ until no more function can be added.

For each linear function $L(P) = \sum_{i=1}^{k} c_i \cdot P(X_i)$ and rule $t \in \Delta$ we can compute the value δ_t^L representing the 'change' on value of L caused by t (where t is of the form $X_\ell \xrightarrow{a} (Q, Q')$) as

$$\delta_t^L = -c_\ell + \sum_{i=1}^{k} c_i \cdot Q'(X_i)\,.$$

Note that δ_t^L does not depend on the actual value of $L(P)$ and that δ_t^L can be easily computed when we know coefficients c_i of L; also note that $L(P'') = L(P) + \delta_t^L$ when $P \xrightarrow{t} [P', P'']$.

For each L we can define the equivalence $=_L$ on rules from Δ. Let us have rules $t = (X \xrightarrow{a} (P, P'))$ and $t' = (Y \xrightarrow{a'} (Q, Q'))$. We define $t =_L t'$ iff $a = a'$, $L(P) = L(Q)$, and $\delta_t^L = \delta_{t'}^L$. For a set of linear functions $\mathcal{L}$ we define the equivalence $=_{\mathcal{L}}$ such that $t =_{\mathcal{L}} t'$ iff $\forall L \in \mathcal{L} : t =_L t'$.

The algorithm maintains a partition of (rules of) Δ, denoted $\mathcal{T}$, and successively refines it. For each class T of $\mathcal{T}$ the algorithm adds to $\mathcal{L}$ a function

$L = \textsc{norm}(T)$. The algorithm also maintains a 'queue' $\mathcal{Q}$ of unprocessed classes of $\mathcal{T}$.

The algorithm starts with $\mathcal{Q} = \mathcal{T} = \{T_a \mid a \in Act\}$ where $T_a = \{t \in \Delta \mid \lambda(t) = a\}$, and proceeds as follows.

While $\mathcal{Q} \neq \emptyset$:

(i) Take some $T \in \mathcal{Q}$ and remove T from $\mathcal{Q}$.

(ii) Compute coefficients of $L = \textsc{norm}(T)$ and add L to $\mathcal{L}$. (The computation of coefficients is described in more detail in Subsection 4.4.)

(iii) For each $t \in \Delta$ where t is of the form $(X \xrightarrow{a} (P, Q))$ compute values $L(P)$, and δ_t^L, and refine $\mathcal{T}$ according to the relation $=_L$. (Put transitions t, t' such that $t \neq_L t'$ to different classes of the refinement of $\mathcal{T}$.)

(iv) Add to $\mathcal{Q}$ each new class of $\mathcal{T}$ created in the previous step.

4.3 Correctness of the Algorithm

Now we show that the set $\mathcal{L}$ computed by the algorithm really represents distributed bisimilarity in the sense that for each $P, Q \in Var^{\oplus}$ we have $P \sim Q$ iff $P \equiv_{\mathcal{L}} Q$.

Claim 4.1 *Let $T \subseteq \Delta$ be a set of rules such that* $\textsc{disabled}(T)$ *is bisimulation invariant. Then* $\textsc{norm}(T)$ *is bisimulation invariant.*

Proof. Let $L = \textsc{norm}(T)$. We show that if $P \sim Q$ and $L(P) = m$, then $L(Q) = m$, which proves the result. We proceed by induction on m. If $m = 0$, the $L(Q) = 0$ follows from the assumption that $\textsc{disabled}(T)$ is bisimulation invariant. Consider $m > 0$ and let us assume $L(Q) = m'$ where $m' \neq m$. Without loss of generality we can assume that $m < m'$. There must be a transition $P \xrightarrow{a} [P', P'']$ such that $L(P'') = m - 1$. Since $P \sim Q$, there must be a matching transition $Q \xrightarrow{a} [Q', Q'']$ such that $P' \sim Q'$ and $P'' \sim Q''$, but obviously $L(Q'') \geq m' - 1 > m - 1$, but on the other hand $P'' \sim Q''$ and $L(P'') = m - 1$ imply $L(Q'') = m - 1$ by induction hypothesis, a contradiction. $\square$

Lemma 4.2 *If $P \sim Q$ then $P \equiv_{\mathcal{L}} Q$.*

Proof. It is sufficient to show that each L added to $\mathcal{L}$ in step (ii) of the algorithm is bisimulation invariant. Because L is computed as $\textsc{norm}(T)$ from some $T \subseteq \Delta$, due to Claim 4.1 it is sufficient to show that $\textsc{disabled}(T)$ is bisimulation invariant. We show that that following invariant holds in every step of the algorithm: For every class T of $\mathcal{T}$, $\textsc{disabled}(T)$ is bisimulation invariant. To show it, we proceed by induction of the number of steps of the algorithm. The invariant obviously holds at the start of the algorithm when $\mathcal{T}$ contains classes T_a for each $a \in Act$.

Now consider T created in step (iii) of the algorithm. Let us assume $P \sim Q$ where $\textsc{enabled}(T)(P)$ and $\textsc{disabled}(T)(Q)$, so there is some $t \in T$

such that $P \xrightarrow{t} [P', P'']$ and there must be some t' such that $Q \xrightarrow{t'} [Q', Q'']$ such that $\lambda(t) = \lambda(t')$ and $P' \sim Q'$ and $P'' \sim Q''$. Obviously $t' \notin T$, and so $t \neq_L t'$ for some $L \in \mathcal{L}$ which is bisimulation invariant by induction hypothesis and Claim 4.1. From $P \sim Q$ we have $L(P) = L(Q)$, $L(P') = L(Q')$ and $L(P'') = L(Q'')$, but $t \neq_L t'$ implies that $L(P') \neq L(Q')$ or $\delta_t^L \neq \delta_{t'}^L$. Because $L(P'') = L(P) + \delta_t^L$ and $L(Q'') = L(Q) + \delta_{t'}^L$, we obtain that either $L(P') \neq L(Q')$ or $L(P'') \neq L(Q)''$, so $P' \not\sim Q'$ or $P'' \not\sim Q''$, a contradiction. □

Lemma 4.3 *If $P \equiv_{\mathcal{L}} Q$ then $P \sim Q$.*

Proof. We just need to show that $\equiv_{\mathcal{L}}$ is distributed bisimulation. Let us have $P, Q \in \mathit{Var}^{\oplus}$ such that $P \equiv_{\mathcal{L}} Q$ and a rule t such that $P \xrightarrow{t} [P', P'']$. Let T be the set of all rules t' such that $t' =_{\mathcal{L}} t$. Obviously $\textsc{enabled}(T)(P)$ and $\textsc{enabled}(T)(Q)$. So let t' be a rule from T enabled in Q, such that $Q \xrightarrow{t'} [Q', Q'']$ From $t =_{\mathcal{L}} t'$ we obtain $L(P') = L(Q')$ for every $L \in \mathcal{L}$, and so $P' \equiv_{\mathcal{L}} Q'$. $t =_{\mathcal{L}} t'$ also implies $\delta_t^L = \delta_{t'}^L$ for every $L \in \mathcal{L}$, and because $L(P) = L(Q)$ for every L, and $L(P'') = L(P) + \delta_t^L$ and $L(Q'') = L(Q) + \delta_{t'}^L$, we obtain $L(P'') = L(Q'')$ for every L, and so $P'' \equiv_{\mathcal{L}} Q''$. □

4.4 Computation of Coefficients of Linear Functions

Now we show that $\textsc{norm}(T)$ for $T \subseteq \Delta$ can be really expressed as a linear function

$$L(P) = \sum_{i=1}^{k} c_i \cdot P(X_i)$$

and that coefficients c_i in L can be computed efficiently.

Recall the assumed order $X_1 \prec X_2 \prec \cdots \prec X_k$. The subroutine that computes coefficients $c_1, c_2, \ldots, c_k$, computes them according to this order. Each coefficient c_i can be computed as $\textsc{norm}(T)(\{X_i\})$. If $\textsc{disabled}(T)(\{X_i\})$ then $c_i = 0$. Otherwise consider the set T_i of all rules t from T such that $\textsc{pre}(t) = X_i$. For each $t \in T_i$ where $t = (X_i \xrightarrow{a} (P, Q))$ we can compute the value $d_t = \textsc{norm}(T)(Q)$ as

$$d_t = 1 + \sum_{j=1}^{i-1} c_j \cdot Q(X_j)\,.$$

Intuitively, d_t represents the distance to disabling T if t is chosen as the rule used for the first transition. Note that d_t can be computed this way, since variables X_j such that $j \geq i$ do not occur in Q, and coefficients c_j where $j < i$ were already computed. We then compute c_i simply as $\min\{d_t \mid t \in T_i\}$. Obviously $c_i = \textsc{norm}(T)(\{X_i\})$.

To analyze the complexity, we need the following definitions. For $x \in \mathbb{N}$, $\mathit{size}(x)$ denotes the number of bits of x when encoded in binary. We suppose that $\mathit{size}(x+y) = 1+\max\{\mathit{size}(x), \mathit{size}(y)\}$, and $\mathit{size}(x \cdot y) = \mathit{size}(x) + \mathit{size}(y)$.

For $t \in \Delta$, $\mathit{size}(t)$ denotes the number of bits in the representation of t where numbers of occurrences of variables on right-hand sides are encoded in binary.

We use n to denote the size of Δ, i.e., $n = \sum_{t \in \Delta} \mathit{size}(t)$.

Proposition 4.4 *$\mathit{size}(c_i) \in O(n)$ for every coefficient c_i of L.*

Proof. Let $X'_1, X'_2, \ldots, X'_{k'}$ be a subset of variables where $c_i > 0$, and let t_i be the rule (with X'_i is on the left-hand side) that was used in the computation of c_i. We show by induction on i the following proposition from which the result directly follows:

$$\mathit{size}(c_i) \leq \sum_{j=1}^{i} \mathit{size}(t_j)$$

This holds trivially for $i = 1$ because c_1 is always 1, so suppose $i > 1$. Let $t_i = (X \xrightarrow{a} (P, Q))$. Note that

$$c_i = 1 + \sum_{j=1}^{i-1} c_j \cdot Q(X_j)$$

and $\mathit{size}(c_j \cdot Q(X_j)) = \mathit{size}(c_j) + \mathit{size}(Q(X_j))$. The sum of all such products can be written in the size of maximal of them plus some number less then their count (overflow caused by addition). This size is less then

$$\mathit{size}(\max\{c_j \mid 1 \leq j < i\}) + \sum_{j=1}^{i-1} \mathit{size}(Q(X_j)).$$

The second summand (the sum) is less then $\mathit{size}(t_i)$. By induction hypothesis maximal c_j can be written in the count of bits needed for $t_1, t_2, \ldots, t_{i-1}$. Therefore c_i can be written in the space needed for representations of $t_1, t_2, \ldots, t_i$.□

Proposition 4.5 *Coefficients of a norm (i.e., linear function) L can be computed in time $O(n^2)$.*

Proof. The most time-consuming step is computation of all d_t. In computation of this, multiplications are more time-consuming than additions. Hence it suffices to show that aggregated complexity of all multiplications is in $O(n^2)$.

In our algorithm, the value of d_t is computed at most once for each t (where t is of the form $X \xrightarrow{a} (P, Q)$). During computation of d_t we need to determine all products $c_j \cdot Q(X_j)$ where $Q(X_j) > 0$. From Proposition 4.4 we know that $\mathit{size}(c_j)$ is in $O(n)$ for every c_j. Hence one product is computed in $O(n \cdot \mathit{size}(Q(X_j)))$. If we sum complexities of such products for all rules $t_i \in \Delta$ (of the form $X_i \xrightarrow{a_i} (P_i, Q_i)$) and all variables, we get the aggregated complexity of all multiplications

$$O(\sum_{t_i \in \Delta} \sum_{\substack{X_j \in \mathit{Var} \\ (Q_i(X_j) > 0)}} (n \cdot \mathit{size}(Q_i(X_j)))) = O(n \cdot \sum_{t_i \in \Delta} \sum_{\substack{X_j \in \mathit{Var} \\ (Q_i(X_j) > 0)}} \mathit{size}(Q_i(X_j))) = O(n^2).$$

□

Proposition 4.6 *Values of δ_t^L for a linear function L and all rules $t \in \Delta$ can be computed in time $O(n^2)$.*

Proof. Again the complexity of additions is dominated by the complexity of multiplications $c_j \cdot Q_i(X_j)$ for t_i of the form $X_i \xrightarrow{a_i} (P_i, Q_i)$. Each such product is computed only once. From Proposition 4.4 we know that each c_j is in $O(n)$. Each $Q_i(X_j)$ is used only once and is part of Δ. Hence we can similarly as in the previous case for coefficients deduce that aggregated complexity of all multiplications is in $O(n^2)$, from which the result follows. □

4.5 Overall Complexity of the Algorithm

In the analysis of the complexity of the algorithm we use the following well known fact.

Fact 4.7 *Let U be a non-empty finite set, and let $\mathcal{U}_1, \mathcal{U}_2, \ldots$ be a sequence of partitions of U such that each $\mathcal{U}_{i+1}$ is a refinement of $\mathcal{U}_i$. Then the total number of different classes in all these partitions is less then $2 \cdot |U|$.*

Proof idea. Use induction on $|U|$. □

Theorem 4.8 *Assuming the input processes in the normal form, there is an algorithm deciding distributed bisimilarity on BPP in time $O(n^3)$.*

Proof. We can use the algorithm described above to compute $\mathcal{L}$ and then check whether $\{X\} \equiv_{\mathcal{L}} \{Y\}$ where X and Y are variables from the instance of BPP-DBISIM. The correctness of the algorithm follows from Lemmas 4.2 and 4.3. The number of subsets of Δ inserted into $\mathcal{Q}$ (and so the the number of functions in $\mathcal{L}$) is $O(n)$ as follows from Fact 4.7. As follows from Propositions 4.5 and 4.6, each such subset can be processed in time $O(n^2)$, and hence the overall time complexity of the algorithm is $O(n^3)$. □

5 Conclusion

We have presented a new algorithm for deciding distributed bisimilarity on BPP. Time complexity of the algorithm is $O(n^3)$, in the case of inputs in normal form, and so it improves the previous result by Lasota [9]. The algorithm presented here is also conceptually simpler. As distributed bisimilarity coincides on BPP with many other non-interleaving equivalences, the algorithm can be used to decide any such equivalence.

Since the transformation of a general BPP process into normal form requires time (and the size of output) $\Theta(n^4)$ in the worst case, it would be interesting to explore the possibility of a direct algorithm which would avoid the transformation into normal form.

References

[1] Castellani, I., "Bisimulations for Concurrency," Ph.D. thesis, University of Edinburgh (1988).

[2] Castellani, I., *Process algebras with localities, chapter 15*, Handbook of Process Algebra (2001), pp. 945–1046.

[3] Christensen, S., "Decidability and Decomposition in Process Algebras," Ph.D. thesis, The University of Edinburgh (1993).

[4] Darondeau, P. and P. Degano, *Causal trees*, in: *Automata, Languages and Programming (ICALP '89)* (1989), pp. 234–248.

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

[6] Hirshfeld, Y., M. Jerrum and F. Moller, *A polynomial-time algorithm for deciding bisimulation equivalence of normed basic parallel processes*, Mathematical Structures in Computer Science **6** (1996), pp. 251–259.

[7] Jančar, P., *Strong bisimilarity on basic parallel processes is PSPACE-complete*, in: *Proceedings of the Eighteenth Annual IEEE Symposium on Logic in Computer Science (LICS-03)* (2003), pp. 218–227.

[8] Jančar, P. and M. Kot, *Bisimilarity on normed basic parallel processes can be decided in time $O(n^3)$*, in: R. Bharadwaj, editor, *Proceedings of the Third International Workshop on Automated Verification of Infinite-State Systems – AVIS 2004*, 2004.

[9] Lasota, S., *A polynomial-time algorithm for deciding true concurrency equivalences of Basic Parallel Processes*, in: *MFCS: Symposium on Mathematical Foundations of Computer Science*, 2003.

[10] van Glabbeek, R., *The linear time—branching time spectrum*, Handbook of Process Algebra (1999), pp. 3–99.

[11] van Glabbeek, R. J. and U. Goltz, *Equivalence notions for concurrent systems and refinement of actions*, in: A. Kreczmar and G. Mirkowska, editors, *Proc. Conf. on Mathematical Foundations of Computer Science*, Lecture Notes in Computer Science **379** (1989), pp. 237–248.

Appendix: Transformation to normal form

Let us assume Δ is a family of equations of the form

$$\Delta = \{X_i \overset{\text{def}}{=} P_i \mid 1 \le i \le k\}$$

where each P_i is defined by the following abstract syntax

$$P \ ::= \ \mathbf{0} \ \mid \ X \ \mid \ a.P \ \mid \ P_1 + P_2 \ \mid \ P_1 \parallel P_2$$

where each occurrence of a variable in P_i is *guarded*, i.e., in a scope of an action prefix.

Each expression Q has a corresponding syntax tree with leaves labelled with $\mathbf{0}$ or variables, and with inner nodes labelled with $a.$, $+$ and $\|$. We can identify Q with this syntax tree. (We assume that subexpressions of Q are parenthesized in Q.) Each subexpression Q' of Q has a corresponding node in the tree representing Q. The size of Q, denoted $\mathit{size}(Q)$, is the number of nodes of the tree that represents Q, the size of an equation $X_i \overset{\text{def}}{=} P_i$ is $\mathit{size}(P_i) + O(1)$, and the size of Δ is the sum of sizes of its equations. In the rest of Appendix we use n to denote the size of Δ. We show that the transformation of Δ to normal form can be done in $O(n^4)$.

We use $\mathcal{S}(Q)$ to denote the set of subexpressions of Q (including Q).

The transformation to normal form is done in two steps.

Step 1: Create Δ' as follows: Start with $\Delta' = \emptyset$ and for each definition $(X_i \overset{\text{def}}{=} P_i) \in \Delta$ and each $Q \in \mathcal{S}(P_i)$ add a new variable Y_Q to Δ' with the definition

$$\begin{array}{ll} Y_Q \overset{\text{def}}{=} Q & \text{if } Q = \mathbf{0} \\ Y_Q \overset{\text{def}}{=} a.Y_{Q_1} & \text{if } Q = a.Q_1 \\ Y_Q \overset{\text{def}}{=} Y_{Q_1} + Y_{Q_2} & \text{if } Q = Q_1 + Q2 \\ Y_Q \overset{\text{def}}{=} Y_{Q_1} \parallel Y_{Q_2} & \text{if } Q = Q_1 \parallel Q2 \end{array}$$

For $Q = X$ where $X \in \mathit{Var}(\Delta)$, identify Y_Q with X. For each definition $(X_i \overset{\text{def}}{=} P_i) \in \Delta$ add to Δ' also the definition $X_i \overset{\text{def}}{=} Y_{P_i}$.

Note that Δ' is in fact only a different representation of the syntax trees of expressions in Δ. The size of Δ' is in $O(n)$. Note also that Δ' is not correct BPP process definition since it can contain expressions that are not guarded.

Step 2: In this step we construct Δ'' in normal form represented as a set of rules of the form $X \xrightarrow{a} (Y, Q)$. (Note that Y is one variable, not a multiset of variables.)

The set of variables of Δ'' is the same as of Δ', i.e., $\mathit{Var}(\Delta'') = \mathit{Var}(\Delta')$. We use $\mathcal{V}$ to denote this set of variables and $\mathcal{V}^{\oplus}$ to denote the set of multisets of $\mathcal{V}$.

Let $\mathit{RRules} = \mathit{Act} \times \mathcal{V} \times \mathcal{V}^{\oplus}$. Intuitively, elements of RRules are rules without left hand sides (Right parts of Rules). For example if $X \xrightarrow{a} (Y, Q)$ is a rule then $\langle a, Y, Q \rangle$ is its corresponding right part.

For each $Z \in \mathcal{V}$ we compute the value $\mathit{deriv}(Z)$ of the function $\mathit{deriv} : \mathcal{V} \to \mathcal{P}(\mathit{RRules})$ (which will be defined below), and add to Δ'' the rules created from elements of $\mathit{deriv}(Z)$ by adding Z as the left hand side to them, i.e.,

$$\Delta'' = \{ Z \xrightarrow{a} (X, Q) \mid Z \in \mathcal{V},\ \langle a, X, Q \rangle \in \mathit{deriv}(Z) \}.$$

The value of $deriv(Z)$ is defined inductively as follows when $(Z \overset{\text{def}}{=} R) \in \Delta'$:

$$\begin{array}{rl}
\emptyset & \text{if } R = \mathbf{0} \\
\{\langle a, Y, \emptyset\rangle\} & \text{if } R = a.Y \\
deriv(Y) \cup deriv(Y') & \text{if } R = Y + Y' \\
extend(deriv(Y), Y') \cup extend(deriv(Y'), Y) & \text{if } R = Y \parallel Y' \\
deriv(Y) & \text{if } R = Y
\end{array}$$

where the function $extend : \mathcal{P}(RRules) \times \mathcal{V} \to \mathcal{P}(RRules)$ is defined as follows:

$$extend(U, Y) = \{\langle a, X, P \cup \{Y\}\rangle \mid \langle a, X, P\rangle \in U\}\,.$$

For example, if $U = \{\langle a, X, XYYZ\rangle, \langle b, Y, X\rangle\}$, then

$$extend(U, Y) = \{\langle a, X, XYYYZ\rangle, \langle b, Y, XY\rangle\}\,.$$

In the definition of $deriv$ we must specify the order in which the values are computed to ensure the correctness of the definition. To specify the order we need some auxiliary definitions.

For an expression Q of Δ we define $top(Q)$ as the expression that we obtain from Q by replacing each occurrence of a subexpression of a form $a.Q'$ with $a.Y_{Q'}$. For example, $top(a.(X + c.Z) + \mathbf{0}) = a.Y_{Q'} + \mathbf{0}$ where $Q' = X + c.Z$.

Consider the syntax tree for $top(P_i)$ where $(X_i \overset{\text{def}}{=} P_i) \in \Delta$. Note that all its branches end with $\mathbf{0}$ or with $a.Y_Q$ for some Q, and all its remaining nodes are labelled with $+$ or $\parallel$. Since the cases $R = \mathbf{0}$ and $R = a.Y$ are the elementary cases in the definition of $deriv$, we can compute $deriv(Y_Q)$ for each Y_Q corresponding to some subexpression Q of $top(P_i)$ (with the exception of leaves of $top(P_i)$ which are labelled with variables). In this computation we proceed in a bottom-up fashion (children at first, then their parents in the syntax tree).

This way we compute $deriv(Y_{P_i})$ for each P_i such that $(X_i \overset{\text{def}}{=} P_i) \in \Delta$, and so we can set $deriv(X_i) = deriv(Y_{P_i})$ for each $X_i \in Var(\Delta)$. Now we can compute the values of $deriv$ for the rest of variables in $\mathcal{V}$, again proceeding in a bottom-up fashion with respect to the structure of the trees of expressions of Δ.

To analyze the complexity of the transformation to normal form, we must analyze the number of elements in $deriv(Y)$ for each $Y \in \mathcal{V}$, and the sizes of these elements. Note that when we have an expression Q where the sum of numbers of elements of $deriv$ at its leaves is m, then the number of elements of $deriv(Y_Q)$ is at most m, and if m' is the maximal size of an element at leaves of Q, then the size of the maximal element in $deriv(Y_Q)$ is at most $m' + |Q|$.

Using these facts we easily obtain that the total number of elements in all $deriv(Y_{P_i})$ where $(X_i \overset{\text{def}}{=} P_i) \in \Delta$ is in $O(n)$ and the size of these elements is also in $O(n)$. From this we obtain that $|deriv(Z)|$ is in $O(n^2)$ for any $Z \in \mathcal{V}$, and that the maximal size of elements of $deriv(Z)$ is in $O(n)$. Since we have $O(n)$ variables in $\mathcal{V}$, we obtain that the total size of Δ'' is in $O(n^4)$.

The computation of the values of *deriv* according to its inductive definition is straightforward, and so the running time of the algorithm is proportional to the size of Δ'' and is also in $O(n^4)$.