Partially-commutative context-free processes[⋆]

Wojciech Czerwiński[1], Sibylle Fröschle[2], and Sławomir Lasota[1]

[1] Institute of Informatics, University of Warsaw
[2] University of Oldenburg

Abstract. Bisimulation equivalence is decidable in polynomial time for both sequential and commutative normed context-free processes, known as BPA and BPP, respectively. Despite apparent similarity between the two classes, different techniques were used in each case. We provide one polynomial-time algorithm that works in a superclass of both normed BPA and BPP. It is derived in the setting of *partially-commutative context-free processes*, a new process class introduced in the paper. It subsumes both BPA and BPP and seems to be of independent interest.

We investigate the bisimulation equivalence of the *context-free processes*, i.e., the process graphs defined by a context-free grammar in Greibach normal form. In process algebra, there are two essentially different ways of interpreting such grammars, depending on whether the concatenation is understood as the sequential or parallel composition of processes. These two process classes are known as BPA (Basic Process Algebra) and BPP (Basic Parallel Processes) [1].

The bisimulation equivalence is decidable both in BPA and BPP [2, 7]. Under the assumption of *normedness* the polynomial-time algorithms exist [5, 6]. These surprising results were obtained basing on the strong *unique decomposition property* enjoyed by both classes. Despite the apparent similarity of BPA and BPP, the algorithms are fundamentally different; cf. [1] (Chapt. 9, p. 573):

> "These algorithms are both based on an exploitation of the decomposition properties enjoyed by normed transition systems; however, despite the apparent similarity of the two problems, different methods appear to be required."

In [4] a decision procedure was given for normed PA, a superclass of both BPA and BPP. It is however very complicated and has doubly exponential complexity. In [8] a polynomial-time algorithm was proposed for the normed BPA vs. BPP problem. It transforms a BPP process into BPA, if possible, and then refers to a BPA algorithm.

This paper contains a polynomial-time algorithm for a superclass of normed BPA and BPP. The algorithm simultaneously applies to BPA and BPP, thus confirming the similarity of the two classes. Our contributions are as follows.

In Section 1 we introduce a new class of *partially-commutative* context-free processes, called BPC, build on the underlying concept of *(in)dependence* of

[⋆] The first and the last author acknowledge a partial support by Polish government grants no. N206 008 32/0810 and N N206 356036.

elementary processes. BPA (no independence) and BPP (full independence) are special cases. Our main motivation was to introduce a common setting for both BPA and BPP; however, the BPC class seems to be of independent interest and may be applied, e.g., as an abstraction in program analysis.

Our first main result is the proof of the unique decomposition property for normed BPC with a transitive dependence relation (Thm 1 in Section 2).

Then in Sections 3–5 we work out our second main result: a polynomial-time algorithm for the bisimulation equivalence in the *feasible* fragment of normed BPC, to be explained below. It clearly subsumes both normed BPA and BPP but allows also for expressing, e.g., a parallel composition of inter-dependent BPA processes. It seems thus suitable for applications, e.g., for modeling of multi-core recursive programs. We sketch the main technical points below.

Recall the classical idea of approximating the bisimulation equivalence from above, by consecutive refinements $R \mapsto R \cap \exp(R)$, where $\exp(R)$ denotes the bisimulation expansion wrt. the relation R. The crucial idea underlying the BPP algorithm [6] was to ensure that the approximant R is always a congruence, and to represent it by a finite *base*; the latter requires a further additional refinement step. Our starting point was an insight of [3] that this latter step yields *the greatest norm-reducing bisimulation* contained in $R \cap \exp(R)$.

The feasibility condition requires the bisimulation expansion to preserve congruences. It appears sufficient for the above scheme to work, after a suitable adaptation, in the general setting of BPC. Roughly speaking, we demonstrate in particular that the BPP algorithm works, surprisingly, for BPA just as well!

Our algorithm efficiently processes both multisets and strings over the set of elementary processes, of pessimistically exponential size. One of technical contributions of this paper is to devise a way of combining the BPP base refinement of [6] with the BPA procedure based on compressed string algorithms [10].

1 Partially commutative context-free processes

BPA and BPP are defined by a context-free grammar in Greibach normal form. The former class is built on sequential composition of processes, while the latter one on parallel composition. Thus BPA processes are elements of the free monoid generated by non-terminal symbols; and BPP processes correspond to the free commutative monoid. Our aim in this section is to define a process class corresponding to the free *partially-commutative* monoid.

A grammar in Greibach normal form consists of a set of non-terminal symbols $\mathtt{V}$, which we call *variables* or *elementary processes*, and a finite set of productions, which we call *rules*, of the form

$$X \xrightarrow{a} \alpha, \tag{1}$$

where $\alpha \in \mathtt{V}^*$, $X \in \mathtt{V}$, and a is an alphabet letter. Additionally assume a symmetric irreflexive relation $\mathtt{I} \subseteq \mathtt{V} \times \mathtt{V}$, called the *independence relation.* For convenience we will also use the *dependence relation* $\mathtt{D} \subseteq \mathtt{V} \times \mathtt{V}$ defined as $\mathtt{D} = (\mathtt{V} \times \mathtt{V}) \setminus \mathtt{I}$. $\mathtt{D}$ is thus symmmetric and reflexive.

The independence induces an equivalence in $\mathtt{V}^*$ in a standard way: two strings over $\mathtt{V}$ are equivalent, if one can transform one into another by a sequence of transpositions of independent variables. Formally, the equivalence $\sim_{\mathtt{I}} \subseteq \mathtt{V}^* \times \mathtt{V}^*$ is the reflexive-transitive closure of the relation containing all pairs $(wXYv, wYXv)$, for $w, v \in \mathtt{V}^*$, $(X, Y) \in \mathtt{I}$; or equivalently, $\sim_{\mathtt{I}}$ is the smallest congruence in $\mathtt{V}^*$ relating all pairs (XY, YX) where $(X, Y) \in \mathtt{I}$. We work in the monoid $\mathtt{V}_{\mathtt{I}}^{\diamondsuit} = \mathtt{V}^*/{\sim_{\mathtt{I}}}$ from now on; we call $\mathtt{V}_{\mathtt{I}}^{\diamondsuit}$ the *free partially-commutative monoid* generated by $\mathtt{I}$. The subscript is usually omitted when $\mathtt{I}$ is clear from the context. Elements of $V^{\diamondsuit}$ will be called *partially-commutative processes*, or *processes* in short, and usually denoted by Greek letters α, β, ... Composition of α and β in $\mathtt{V}^{\diamondsuit}$ is written $\alpha\beta$. Empty process (identity in $\mathtt{V}^{\diamondsuit}$) will be written as ϵ. Our development is based on the decision to interpret right-hand sides α of productions (1) as elements of $\mathtt{V}^{\diamondsuit}$, instead of as words or multisets over $\mathtt{V}$. The induced class of processes we will call BPC (Basic Partially-Commutative algebra).

Formally, a *BPC process definition Δ* consists of a finite set $\mathtt{V}$ of variables, a finite alphabet $\mathcal{A}$, an independence relation $\mathtt{I} \subseteq \mathtt{V} \times \mathtt{V}$, and a finite set of rules (1), where $\alpha \in \mathtt{V}^{\diamondsuit}$, $X \in \mathtt{V}$, and $a \in \mathcal{A}$. The induced transition system has processes as states, and transitions derived according to the following rule:

$$X\beta \xrightarrow{a} \alpha\beta \quad \text{whenever} \quad (X \xrightarrow{a} \alpha) \in \Delta,\ \beta \in \mathtt{V}^{\diamondsuit}.$$

Note that when $(X, Y) \in \mathtt{I}$ it may happen that $X\beta = Y\beta'$ in $\mathtt{V}^{\diamondsuit}$. In such case the rules of both X and Y contribute to the transitions of $X\beta = Y\beta'$. Particular special cases are BPA ($\mathtt{I}$ is empty), and BPP ($\mathtt{D}$ is the identity relation).

Example 1. Let $\mathtt{I}$ contain the pairs (B, C), (T, C), (B, U), (T, U), and the symmetric ones. In the transition system induced by the rules:

$$\begin{array}{llll} P \xrightarrow{a} WBCT \qquad & W \xrightarrow{a} WBC \qquad & T \xrightarrow{t} \epsilon \qquad & B \xrightarrow{b} \epsilon \\ & W \xrightarrow{s} U & U \xrightarrow{u} \epsilon & C \xrightarrow{c} \epsilon \end{array}$$

there are, among the others, the following transitions:

$$P \xrightarrow{a^3} W(BC)^3T \xrightarrow{s} U(BC)^3T \sim_{\mathtt{I}} B^3TUC^3 \xrightarrow{b^3} TUC^3 \xrightarrow{tu} C^3 \xrightarrow{c^3} \epsilon.$$

Definition 1. A *bisimulation* is any binary relation R over processes such that $R \subseteq \exp(R)$, where $\exp(R)$, the *bisimulation expansion wrt. R*, contains all pairs (α, β) of processes such that for all $a \in \mathcal{A}$:

1. whenever $\alpha \xrightarrow{a} \alpha'$, there is β' with $\beta \xrightarrow{a} \beta'$ and $(\alpha', \beta') \in R$,
2. the symmetric condition holds,

The *bisimulation equivalence*, written as $\sim$, is the union of all bisimulations.

An equivalence $\approx \subseteq \mathtt{V}^{\diamondsuit} \times \mathtt{V}^{\diamondsuit}$ is a congruence if it is preserved by composition: $\alpha \approx \alpha'$ and $\beta \approx \beta'$ implies $\alpha\beta \approx \alpha'\beta'$. Bisimulation equivalence is a congruence both in BPA and BPP; however it needs not be so in BPC, as the following simple example shows:

Example 2. Consider $\mathtt{D} = \{(A, B), (B, A)\}$ (plus identity pairs) and the rules below; $AB \not\sim A'B'$, despite that $A \sim A'$ and $B \sim B'$:

$$A \xrightarrow{a} \epsilon \qquad A' \xrightarrow{a} \epsilon \qquad B \xrightarrow{b} \epsilon \qquad B' \xrightarrow{b} \epsilon.$$

2 The unique decomposition for normed processes

Assume that Δ is *normed*, i.e., for every variable $X \in \mathtt{V}$, there is a sequence of transitions $X \xrightarrow{a_1} \alpha_1 \ldots \xrightarrow{a_k} \alpha_k = \epsilon$ leading to the empty process ϵ. The length of the shortest such sequence is *the norm* of X, written $|X|$. Norm extends additively to all processes. By the very definition of norm, a transition may decrease norm by at most 1. Those transitions that do decrease norm will be called *norm-reducing* (n-r-transitions, in short). The rules of Δ that induce such transitions will be called norm-reducing as well.

We will need a concept of norm-reducing bisimulation (n-r-bisimulation, in short), i.e., a bisimulation over the transition system restricted to only norm-reducing transitions. The appropriate norm-reducing expansion wrt. R will be written as n-r-exp(R). Every bisimulation is a n-r-bisimulation (as a norm-reducing transition must be matched in a bisimulation by a norm-reducing one) but the converse does not hold in general.

Proposition 1. *Each n-r-bisimulation, and hence each bisimulation, is norm-preserving, i.e., whenever α and β are related then $|\alpha| = |\beta|$.*

Assume from now on that variables $\mathtt{V} = \{X_1, \ldots, X_n\}$ are ordered according to non-decreasing norm: $|X_i| \leq |X_j|$ whenever $i < j$. We write $X_i < X_j$ if $i < j$. Note that $|X_1|$ is necessarily 1, and that norm of a variable is at most exponential wrt. the size of Δ, understood as the sum of lengths of all rules.

We write $X^\diamondsuit$, for a subset $X \subseteq \mathtt{V}$ of variables, to mean the free partially-commutative monoid generated by X and the independence relation restricted to pairs from X. Clearly, $X^\diamondsuit$ inherits composition and identity from $\mathtt{V}^\diamondsuit$.

Let $\equiv$ be an arbitrary norm-preserving congruence in $\mathtt{V}^\diamondsuit$. Intuitively, an elementary process X_i is *decomposable* if $X_i \equiv \alpha\beta$ for some $\alpha, \beta \neq \epsilon$. Note that $|\alpha|$, $|\beta| < |X_i|$ then. For technical convenience we prefer to apply a slightly different definition. We say that X_i is *decomposable* wrt. $\equiv$, if $X_i \equiv \alpha$ for some process $\alpha \in \{X_1, \ldots, X_{i-1}\}^\diamondsuit$; otherwise, X_i is called *prime* wrt. $\equiv$. In particular, X_1 is always prime.

Denote by $\mathtt{P}$ the set of primes wrt. $\equiv$. It is easy to show by induction on norm that for each process α there is some $\gamma \in \mathtt{P}^\diamondsuit$ with $\alpha \equiv \gamma$; in such case γ is called a *prime decomposition* of α. Note that a prime decomposition of X_i is either X_i itself, or it belongs to $\{X_1, \ldots, X_{i-1}\}^\diamondsuit$. We say that $\equiv$ has the *unique decomposition property* if each process has precisely one prime decomposition. While the set $\mathtt{P}$ of primes depends on the chosen ordering of variables (in case $X_i \equiv X_j$, $i \neq j$), the unique decomposition property does not.

In general $\sim$, even if it is a congruence, needs not to have the unique decomposition property, as the following example shows:

Example 3. Let $\mathtt{I} = \{(B, C), (C, B)\}$ and the rules be as follows:

$$A \xrightarrow{a} B \qquad A' \xrightarrow{a} C \qquad B \xrightarrow{b} \epsilon \qquad C \xrightarrow{c} \epsilon.$$

Consider two equivalent processes $AC \sim A'B$. As all four variables are prime wrt. $\sim$, we have thus a process with two different prime decompositions wrt. $\sim$.

The situation is not as bad as Example 3 suggests. We can prove the unique decomposition property (Thm 1 below) assumed that $\mathtt{D}$ is transitive (hence $\mathtt{D}$ is an equivalence). Abstraction classes of $\mathtt{D}$ we call *threads*. Intuitively, a process (i.e., an abstraction class of $\sim_{\mathtt{I}}$) may be seen as a collection of strings over $\mathtt{V}$, one for each thread. For convenience, each of the strings will be called thread as well. This concrete representation will be extensively exploited in the sequel.

For $\alpha, \gamma \in \mathtt{V}^{\diamondsuit}$ we say that α *masks* γ if any thread nonempty in γ is also nonempty in α. A binary relation $\approx$ is called:

– *strongly right-cancellative* if whenever $\alpha\gamma \approx \beta\gamma$ then $\alpha \approx \beta$;
– *right-cancellative* if whenever $\alpha\gamma \approx \beta\gamma$ and both α and β mask γ then $\alpha \approx \beta$.

Proposition 2. *If a congruence has the unique decomposition property then it is strongly right-cancellative.*

Proposition 3. *If $\approx$ is strongly right-cancellative then exp($\approx$) and n-r-exp($\approx$) are right-cancellative.*

A counterexample to the unique decomposition property of $\equiv$, if any, is a pair (α, β) of processes from $\mathtt{P}^{\diamondsuit}$ with $\alpha \equiv \beta$, $\alpha \neq \beta$. If there is a counterexample, there is one of minimal norm; we call it a *minimal $\equiv$-counterexample.* A congruence $\equiv$ is *weakly right-cancellative* if whenever $\alpha\gamma \equiv \beta\gamma$ and both α and β mask γ and $(\alpha\gamma, \beta\gamma)$ is a minimal $\equiv$-counterexample then $\alpha \equiv \beta$.

Theorem 1. *Assume $\mathtt{D}$ to be transitive. Then each weakly right-cancellative congruence that is a n-r-bisimulation has the unique decomposition property.*

It is a generalization of the unique decomposition in BPA and BPP (cf. [5] and [6], resp.), in two respects. Firstly, we consider a wider class of processes. Secondly, we treat every n-r-bisimulation that is a weak right-cancellative congruence, while in the two cited papers the result was proved only for the bisimulation equivalence $\sim$ (cf. Prop. 10 in Section 3.2, where we consider the unique decomposition property of $\sim$). The rest of this section is devoted to the proof.

Proof of Thm 1. Fix a weakly right-cancellative congruence $\equiv$ that is a n-r-bisimulation; $\equiv$ is thus norm-preserving. Let $\mathtt{P} \subseteq \mathtt{V}$ denote primes wrt. $\equiv$, ordered consistently with the ordering $\leq$ of $\mathtt{V}$. For the sake of contradiction, suppose that the unique decomposition property does not hold, and consider a minimal $\equiv$-counterexaple (α, β).

We say that a transition of one of α, β is *matched* with a transition of the other if the transitions are equally labelled and the resulting processes are related by $\equiv$. Clearly, each norm-reducing transition of one of α, β may be matched with a transition of the other. Due to the minimality of the counterexample (α, β), *any* prime decompositions of the resulting processes, say α' and β', are necessarily identical. For convenience assume that each right-hand side of Δ was replaced by a prime decomposition wrt. $\equiv$. Thus α', β' must be identical.

Let t be the number of threads and let $\mathtt{V} = \mathtt{V}_1 \cup \ldots \cup \mathtt{V}_t$ be the partition of $\mathtt{V}$ into threads. A process α restricted to the ith thread we denote by $\alpha_i \in {\mathtt{V}_i}^*$. Hence $\alpha = \alpha_1 \ldots \alpha_t$ and the order of composing the processes α_i is irrelevant.

A (n-r-)transition of α, or β, is always a transition of the first variable in some α_i, or β_i; such variables we call *active*. Our considerations will strongly rely on the simple observation: a n-r-transition of an active variable X may 'produce' only variables of strictly smaller norm than X, thus smaller than X wrt. $\leq$.

In Claims 1–6, to follow, we gradually restrict the possible form of (α, β).

Claim 1. For each $i \leq t$, one of α_i, β_i is a suffix of the other.

Proof. Suppose that some thread i does not satisfy the requirement, and consider the longest common suffix γ of α_i and β_i. Thus γ is masked in α and β. As $\equiv$ is weakly right-cancellative, γ must be necessarily empty – otherwise we would obtain a smaller counterexample. Knowing that the last letters of α_i and β_i, say P_α, P_β, are different, we perform a case-analysis to obtain a contradiction. The length of a string w is written $\|w\|$.

CASE 1: $\|\alpha_i\| \geq 2$, $\|\beta_i\| \geq 2$. After performing any pair of matching n-r-transitions, the last letters P_α, P_β will still appear in the resulting processes α', β', thus necessarily $\alpha' \neq \beta'$ – a contradiction to the minimality of (α, β).

CASE 2: $\|\alpha_i\| = 1$, $\|\beta_i\| \geq 2$ (or a symmetric one). Thus $\alpha_i = P_\alpha$. As P_α is prime, some other thread is necessarily nonempty in α. Perform any n-r-transition from that other thread. Irrespective of a matching move in β, the last letters P_α and P_β still appear in the resulting processes – a contradiction.

CASE 3: $\|\alpha_i\| = \|\beta_i\| = 1$. Thus $\alpha_i = P_\alpha$, $\beta_i = P_\beta$. Similarly as before, some other thread must be nonempty both in α and β. Asssume wlog. $|P_\alpha| \geq |P_\beta|$. Perform any n-r-transition in α from a thread different than i. Irrespective of a matching move in β, in the resulting processes α', β' the last letter P_α in ${\alpha'}_i$ is different from the last letter (if any) in ${\beta'}_i$ – a contradiction. □

Claim 2. For each $i \leq t$, either $\alpha_i = \beta_i$, or $\alpha_i = \epsilon$, or $\beta_i = \epsilon$.

Proof. By minimality of (α, β). If α_i, say, is a proper suffix of β_i, then a n-r-transition of α_i may not be matched in β. □

A thread i is called *identical* if $\alpha_i = \beta_i \neq \epsilon$.

Claim 3. A n-r-transition of one of α, β from an identical thread may be matched only with a transition from the same thread.

Proof. Consider an identical thread i. A n-r-transition of α_i decreases $|\alpha_i|$. By minimality of (α, β), $|\beta_i|$ must be decreased as well. □

Claim 4. There is no identical thread.

Proof. Assume thread i is identical. Some other thread j is not as $\alpha \neq \beta$; wlog. assume $|\alpha_j| > |\beta_j|$, using Claim 1. Consider a n-r-transition of the active variable in $\alpha_i = \beta_i$ that maximises the increase of norm on thread j. This transition, performed in α, may not be matched in β, due to Claim 3, so that the norms of α_j and β_j become equal. □

Claim 5. One of α, β, say α, has only one nonempty thread.

Proof. Consider the greatest (wrt. $\leq$) active variable and assume wlog. that it appears in α. We claim α has only one nonempty thread. Indeed, if some other thread is nonempty, a n-r-transition of this thread can not be matched in β. □

Let α_i be the only nonempty thread in α, and let P_i be the active variable in that thread, $\alpha_i = P_i\gamma_i$. The process γ_i is nonempty by primality of P_i.

Claim 6. $|P_i|$ is greater than norm of any variable appearing in γ_i.

Proof. Consider any thread $\beta_j = P_j\gamma_j$ nonempty in β. We know that $|P_i| \geq |P_j|$. As the thread i is empty in β, the norm of P_j must be sufficiently large to "produce" all of γ_i in one n-r-transition, i.e., $|P_j| > |\gamma_i|$. Thus $|P_i| > |\gamma_i|$. □

Now we easily derive a contradiction. Knowing that P_i has the greatest norm in α, consider the processes $P_i\alpha \equiv P_i\beta$, and an arbitrary sequence of $|P_i|{+}1$ norm-reducing transitions from $P_i\beta$. We may assume that this sequence does not touch P_i as $|\beta| = |\alpha| > |P_i|$. Let β' be the resulting process, and let α' denote the process obtained by performing some matching transitions from $P_i\alpha = P_iP_i\gamma_i$. The variable P_i may not appear in α' while it clearly appears in β'. Thus $\alpha' \equiv \beta'$, $\alpha' \neq \beta'$ and $|\alpha'| = |\beta'|$ is smaller than $|\alpha| = |\beta|$ – a contradiction to the minimality of the counterexample (α, β). This completes the proof of the theorem.

3 The algorithm

From now on we only consider normed BPC process definitions Δ with a transitive dependence relation $\mathtt{D}$. Such Δ is called *feasible* if it satifies the following:

Assumption 1 (Feasibility). Whenever $\equiv$ is a congruence, then the relations $\equiv \cap \exp(\equiv)$ and $\equiv \cap$ n-r-exp$(\equiv)$ are congruences as well.

Clearly, not all normed BPC process definitions are feasible (cf., e.g., Example 2 and the smallest congruence $\equiv$ such that $A \equiv A'$ and $B \equiv B'$).

Proposition 4. *Every normed BPA or BPP process definition is feasible.*

We prefer to separate description of the algorithm from the implementation details. In this section we provide an outline of the algorithm only. In Section 4 we explain how each step can be implemented. Without further refinement, this would give an exponential-time procedure. Finally, in Section 5 we provide the polynomial-time implementation of crucial subroutines. Altogether, Sections 3–5 contain the proof of our main result:

Theorem 2. *The bisimulation equivalence $\sim$ is decidable in polynomial time for feasible BPC process definitions.*

The algorithm will compute a finite representation of $\sim$. From now on let Δ be a fixed feasible process definition with variables $\mathtt{V}$ and dependence $\mathtt{D}$; we also fix an ordering of variables $\mathtt{V} = \{X_1, \dots, X_n\}$.

3.1 Bases

A *base* will be a finite representation of a congruence having the unique decomposition property. A base $\mathtt{B} = (\mathtt{P}, \mathtt{E})$ consists of a subset $\mathtt{P} \subseteq \mathtt{V}$ of variables, and a set $\mathtt{E}$ of equations $(X_i = \alpha)$ with $X_i \notin \mathtt{P}$, $\alpha \in (\mathtt{P} \cap \{X_1, \dots, X_{i-1}\})^\diamondsuit$ and $|X_i| = |\alpha|$. We assume that there is precisely one equation for each $X_i \notin \mathtt{P}$.

An equation $(X_i = \alpha) \in \mathtt{E}$ is thought to specify a decomposition of a variable $X_i \notin \mathtt{P}$ in $\mathtt{P}^\diamondsuit$. Put $d_\mathtt{B}(X_i) = \alpha$ if $(X_i = \alpha) \in \mathtt{E}$ and $d_\mathtt{B}(X_i) = X_i$ if $X_i \in \mathtt{P}$. We want $d_\mathtt{B}$ to unambiguously extend to all processes as a homomorphism from $\mathtt{V}^\diamondsuit$ to $\mathtt{P}^\diamondsuit$. This is only possible when, intuitively, decompositions of independent variables are independent. Formally, we say that a base $\mathtt{B}$ is $\mathtt{I}$-*preserving* if whenever $(X_i, X_j) \in \mathtt{I}$, and $d_\mathtt{B}(X_i) = \alpha$, $d_\mathtt{B}(X_j) = \beta$, then $\alpha\beta = \beta\alpha$ in $\mathtt{P}^\diamondsuit$.

The elements of $\mathtt{P}$ are, a priori, arbitrarily chosen, and not to be confused with the primes wrt. a given congruence. However, an $\mathtt{I}$-preserving base $\mathtt{B}$ naturally induces a congruence $=_\mathtt{B}$ on $\mathtt{V}^\diamondsuit$: $\alpha =_\mathtt{B} \beta$ iff $d_\mathtt{B}(\alpha) = d_\mathtt{B}(\beta)$. It is easy to verify that primes wrt. $=_\mathtt{B}$ are precisely variables from $\mathtt{P}$ and that $=_\mathtt{B}$ has the unique decomposition property. Conversely, given a congruence $\equiv$ with the latter property, one easily obtains a base $\mathtt{B}$: take primes wrt. $\equiv$ as $\mathtt{P}$, and the (unique) prime decompositions of decomposable variables as $\mathtt{E}$. $\mathtt{B}$ is guaranteed to be $\mathtt{I}$-preserving, by the uniqueness of decomposition of $XY \equiv YX$, for $(X,Y) \in \mathtt{I}$. As these two transformations are mutually inverse, we have just shown:

Proposition 5. *A norm-preserving congruence in* $\mathtt{V}^\diamondsuit$ *has unique decomposition property iff it equals* $=_\mathtt{B}$, *for an* $\mathtt{I}$-*preserving base* $\mathtt{B}$.

This allows us, in particular, to speak of *the base of* a given congruence, if it exhibits the unique decomposition property; and to call elements of $\mathtt{P}$ *primes*.

3.2 Outline of the algorithm

For an equivalence $\equiv$ over processes, let gnrb($\equiv$) denote the greatest n-r-bisimulation that is contained in $\equiv$, defined as the union of all n-r-bisimulations contained in $\equiv$. It admits the following fix-point characterization:

Proposition 6. $(\alpha, \beta) \in$ *gnrb*$(\equiv)$ *iff* $\alpha \equiv \beta$ *and* $(\alpha, \beta) \in$ *n-r-exp*(*gnrb*$(\equiv)$).

Proposition 7. *(i) if* $\equiv$ *is a congruence then gnrb*$(\equiv)$ *is a congruence as well. (ii) if* $\equiv$ *is right-cancellative then gnrb*$(\equiv)$ *is weakly right-cancellative.*

Proof. (i) gnrb($\equiv$) is the intersection of the descending chain of relations $\equiv_1 := \equiv \cap$ n-r-exp($\equiv$), $\equiv_2 := \equiv_1 \cap$ n-r-exp($\equiv_1$), ... Due to Assumption 1 each $\equiv_i$ is a congruence, hence gnrb($\equiv$) is a congruence too.

(ii) Consider a minimal gnrb($\equiv$)-counterexample $(\alpha\gamma, \beta\gamma)$ such that γ is masked both by α and β. Hence each n-r-transition of α (β) is matched by a transition of β (α); the resulting processes have the same prime decompositions, due to minimality of $(\alpha\gamma, \beta\gamma)$, and thus are related by gnrb($\equiv$). This proves that $(\alpha, \beta) \in$ n-r-exp(gnrb($\equiv$)). Due to right-cancellativity of $\equiv$ we have also $\alpha \equiv \beta$. Now by the *if* implication of Prop. 6 we deduce $(\alpha, \beta) \in$ gnrb($\equiv$). □

Here is the overall idea. We start with the initial congruence $=_{\mathtt{B}}$ that relates processes of equal norm, and then perform the fixpoint computation by refining $=_{\mathtt{B}}$ until it finally stabilizes. The initial approximant has the unique decomposition property. To ensure that all consecutive approximants also have the property, we apply the refinement step: $=_{\mathtt{B}} \mapsto$ gnrb($=_{\mathtt{B}} \cap$ exp($=_{\mathtt{B}}$)). By Assumption 1, $=_{\mathtt{B}} \cap$ exp($=_{\mathtt{B}}$) is a congruence, and by Prop. 2 and 3 it is right-cancellative. Thus Prop. 7 applies to gnrb($=_{\mathtt{B}} \cap$ exp($=_{\mathtt{B}}$)) and in consequence of Thm. 1 we get:

Proposition 8. *gnrb($=_{\mathtt{B}} \cap$ exp($=_{\mathtt{B}}$)) is a congruence with the unique decomposition property.*

Outline of the algorithm:

(1) Compute the base B of 'norm equality'.
(2) If $=_{\mathtt{B}}$ is a bisimulation then halt and return B.
(3) Otherwise, compute the base of the congruence gnrb($=_{\mathtt{B}} \cap$ exp($=_{\mathtt{B}}$)).
(4) Assign this new base to B and go to step (2).

This scheme is a generalization of the BPP algorithm [6]. As our setting is more general, the implementation details, to be given in the next sections, will be necessarily more complex than in [6]. However, termination and correctness may be proved without inspecting the details of implementation:

Proposition 9 (termination). *The number of iterations is smaller than n.*

Proof. In each iteration the current relation $=_{\mathtt{B}}$ gets strictly finer (if B did not change in one iteration, then $=_{\mathtt{B}}$ would necessarily be a bisimulation). Therefore all prime variables stay prime, and at least one non-prime variable becomes prime. To prove this suppose the contrary. Consider the smallest X_i wrt. $\leq$ such that its prime decomposition changes during the iteration. X_i has thus two different prime decompositions (wrt. the 'old' relation $=_{\mathtt{B}}$), a contradiction. □

Proposition 10 (correctness). *The algorithm computes the base of $\sim$.*

Proof. The invariant $\sim \subseteq =_{\mathtt{B}}$ is preserved by each iteration. The opposite inclusion $=_{\mathtt{B}} \subseteq \sim$ follows when $=_{\mathtt{B}}$ is a bisimulation. □

The unique decomposition property of $\sim$ is thus only a corollary, as we did not have to prove it prior to the design of the algorithm! A crucial discovery is that the unique decomposition must only hold for the relations $\mathrm{gnrb}(=_{\mathtt{B}} \cap \exp(=_{\mathtt{B}}))$ and that these relations play a prominent role in the algorithm (cf. [3]).

4 Implementation

Step (1) is easy: recalling that $|X_1| = 1$, initialize $\mathtt{B}$ by $\mathtt{P} := \{X_1\}$, $\mathtt{E} := \{X_i = {X_1}^{|X_i|} : i = 2 \ldots n\}$. On the other hand implementations of steps (2) and (3) require some preparation. We start with a concrete characterization of $\mathtt{I}$-preserving bases $\mathtt{B} = (\mathtt{P}, \mathtt{E})$. Distinguish *monic threads* as those containing precisely one prime variable. $\mathtt{B}$ is called pure if for each decomposition $(X_i = \alpha) \in \mathtt{E}$, α contains only variables from the thread of X_i and from (other) monic threads.

Proposition 11. *A base* $\mathtt{B}$ *is* $\mathtt{I}$*-preserving if and only if it is pure.*

Proof. The *if* implication is immediate: if $\mathtt{B}$ is pure and the decompositions $\alpha = d_{\mathtt{B}}(X_i)$, $\beta = d_{\mathtt{B}}(X_j)$ of independent variables X_i, X_j both contain a prime from some thread, then the thread is necessarily monic. Thus $\alpha\beta = \beta\alpha$. This includes the case when any of X_i, X_j is prime. The *only if* implication is shown as follows. Consider a decomposition $(X_i = \alpha) \in \mathtt{E}$ and any prime X_j, appearing in α, from a thread different than that of X_i. As $\mathtt{B}$ is $\mathtt{I}$-preserving, $X_j\alpha = \alpha X_j$. Hence α, restricted to the thread of X_j, must be a monomial ${X_j}^k$. As X_j was chosen arbitrary, we deduce that this thread must be a monic one. □

Let $\mathtt{B} = (\mathtt{P}, \mathtt{E})$ be a pure base. Two variables $X_i, X_j \notin \mathtt{P}$ are *compatible* if either they are independent, or $(X_i, X_j) \in \mathtt{D}$, $(X_i = \alpha)$, $(X_j = \beta) \in \mathtt{E}$ and α and β contain primes from the same threads. That is, α contains a prime from a thread iff β contains a prime from that thread. Note that it must be the same prime only in case of a (necessarily monic) thread different from the thread of X_i and X_j. $\mathtt{B}$ is compatible if all pairs of non-prime variables are compatible.

Proposition 12. *Let* $\mathtt{B}$ *be pure. If* $=_{\mathtt{B}}$ *is a n-r-bisimulation then* $\mathtt{B}$ *is compatible.*

Proof. Assume $(X_i, X_j) \in \mathtt{D}$, $(X_i = \alpha)$ and $(X_j = \beta) \in \mathtt{E}$. For the sake of contradiction, suppose that some thread t is nonempty in α but empty in β: $\alpha_t \neq \epsilon$, $\beta_t = \epsilon$. We know that $(X_j X_i, \beta\alpha) \in \text{n-r-exp}(=_{\mathtt{B}})$. Hence a norm-reducing transition of $(\beta\alpha)_t = \alpha_t$ is matched by a transition of X_j, so that the decompositions of the two resulting processes are equal. In the decomposition of the left process the norm of thread t is at least $|\alpha_t|$, as X_i was not involved in the transition. On the right side, the norm decreased due to the fired transition, and is thus smaller than $|\alpha_t|$ – a contradiction. □

Step (2) may be implemented using the fact stated below. It is essentially an adaptation of the property of *Caucal base* [1] to the setting of partially-commutative processes, but the proof requires more care than in previously studied settings.

Proposition 13. *Let* $\mathtt{B} = (\mathtt{P}, \mathtt{E})$ *be pure and compatible. Then* $=_{\mathtt{B}}$ *is a bisimulation if and only if* $(X, \alpha) \in exp(=_{\mathtt{B}})$ *for each* $(X = \alpha) \in \mathtt{E}$.

Proof. We only need to consider the *if* direction. For any pair α, β of processes such that $\alpha =_{\mathtt{B}} \beta$, we should show that $(\alpha, \beta) \in \exp(=_{\mathtt{B}})$. Let $\gamma := d_{\mathtt{B}}(\alpha) = d_{\mathtt{B}}(\beta)$ be the prime decomposition of α and β. It is sufficient to prove that $(\alpha, \gamma) \in \exp(=_{\mathtt{B}})$, as $\exp(=_{\mathtt{B}})$ is symmetric and transitive. We will analyse the possible transitions of α and γ, knowing that all decompositions in $\mathtt{E}$ are pure.

First consider the possible transitions of α. Let X_i be an active variable in α, i.e., $\alpha = X_i \alpha'$ for some α', and let $\delta := d_{\mathtt{B}}(X_i)$. Then γ may be also split into $\gamma = \delta\gamma'$, where $\gamma' = d_{\mathtt{B}}(\alpha')$. Thus, any transition of α may be matched by a transition of γ, as we know, by assumption, that $(X_i, \delta) \in \exp(=_{\mathtt{B}})$.

Now we consider the possible transitions of γ. Let a prime X_j be active in γ. Choose a variable X_i such that X_j appears in the decomposition $\delta = d_{\mathtt{B}}(X_i)$. Due to compatibility of $\mathtt{B}$ we may assume that the chosen X_i is active in α, i.e., $\alpha = X_i \alpha'$, for some α'. Similarly as above we have $\gamma = \delta\gamma'$. A transition of X_j is necessarily a transition of δ, hence may be matched by a transition of X_i, by the assumption that $(X_i, \delta) \in \exp(=_{\mathtt{B}})$. □

Proposition 14. *The base* $\mathtt{B}$ *is pure and compatible in each iteration.*

Proof. Initially $\mathtt{B}$ is pure and compatible. After each iteration $\mathtt{B}$, being the base of $\text{gnrb}(=_{\mathtt{B}} \cap \exp(=_{\mathtt{B}}))$, is pure by Prop. 11, 8 and 5, and compatible by Prop. 12. □

Therefore in step (2), the algorithm only checks the condition of Prop. 13.

Implementation of step (3). We compute the base $\mathtt{B}' = (\mathtt{P}', \mathtt{E}')$ of the greatest n-r-bisimulation contained in $=_{\mathtt{B}} \cap \exp(=_{\mathtt{B}})$. As only norm-reducing transitions are concerned, the base is obtained in a sequence of consecutive extensions, by inspecting the variables according to their ordering, as outlined below.

In the following let $\equiv$ denote the relation $=_{\mathtt{B}} \cap \exp(=_{\mathtt{B}})$. The algorithm below is an implementation of the fix-point characterization of $\text{gnrb}(\equiv)$ (cf. Prop. 6).

Implementation of step (3):

> Start with the set $\mathtt{P}' = \{X_1\}$ of primes and the empty set $\mathtt{E}'$ of decompositions. Then for $i := 2, \ldots, n$ do the following:
>
> > Check if there is some $\alpha \in \mathtt{P}'^{\diamondsuit}$ such that
> >
> > (a) $(X_i, \alpha) \in \text{n-r-exp}(=_{\mathtt{B}'})$, and (b) $X_i \equiv \alpha$.
> >
> > If one is found, add $(X_i = \alpha)$ to $\mathtt{E}'$. Otherwise, add X_i to $\mathtt{P}'$ and thus declare X_i prime in $\mathtt{B}'$.

Before explaining how searching for a decomposition α of X_i is implemented, we consider the correctness issue.

Proposition 15 (correctness of step (3)). *The base* $\mathtt{B}'$ *computed in step (3) coincides with the base of gnrb*$(\equiv)$.

Now we return to the implementation of step (3). Seeking $\alpha \in (\mathtt{P}')^\diamondsuit$ appropriate for the decomposition of X_i is performed by an exhaustive check of all 'candidates' computed according to the procedure described below. The computation implements a necessary condition for (a) to hold: if $(X_i, \alpha) \in$ n-r-exp$(=_{\mathtt{B}'})$ then α is necessarily among the candidates.

Computing candidates α*:*

> Fix an arbitrarily chosen norm-reducing rule $X_i \xrightarrow{a} \beta$ (hence $\beta \in \{X_1, \ldots, X_{i-1}\}^\diamondsuit$) and let $\beta' := d_{\mathtt{B}'}(\beta)$ be a decomposition of β wrt. $\mathtt{B}'$ (hence $\beta' \in (\mathtt{P}')^\diamondsuit$). For any $j < i$ such that $X_j \in \mathtt{P}'$, for any norm reducing rule $X_j \xrightarrow{a} \gamma$, do the following: let $\gamma' := d_{\mathtt{B}'}(\gamma)$; if $\beta' = \gamma'\gamma''$, for some γ'', then let $\alpha := X_j\gamma'' \in (\mathtt{P}')^\diamondsuit$ be a candidate.

We will write $\gamma \leq_{\mathtt{B}'} \beta$ to mean that $d_{\mathtt{B}'}(\gamma)$ is a prefix of $d_{\mathtt{B}'}(\beta)$.

5 Polynomial-time implementation

The algorithm performs various manipulations on processes. The most important are the following 'subroutines', invoked a polynomial number of times:

(i) Given α, $\beta \in \mathtt{V}^\diamondsuit$ and $\mathtt{B}$, check if $\alpha =_{\mathtt{B}} \beta$.
(ii) Given α, $\beta \in \mathtt{V}^\diamondsuit$ and $\mathtt{B}$, check if $\alpha \leq_{\mathtt{B}} \beta$. If so, compute γ such that $\alpha\gamma =_{\mathtt{B}} \beta$.

Recall that the processes involved in the algorithm are tuples of strings over prime variables, one string for each thread, of pessimistically exponential length and norm. We need thus to consider two inter-related issues:

- a succint representation of processes in polynomial space; and
- polynomial-time implementations of all manipulations, including subroutines (i) and (ii), that preserve the succint representation of manipulated data.

The special case of BPP is straightforward: the commutative processes are essentially multisets, may be thus succintly represented by storing exponents in binary, and effectively manipulated in polynomial time.

In the general case of BPC, and even in BPA, we need a more elaborate approach. To get a polynomial-time implementation, we need to use a method of 'compressed' representation of strings. Moreover, all the operations performed will have to be implemented on compressed representations, without 'decompressing' to the full exponential-size strings. After preparatory Section 5.1, in Section 5.2 we explain how to implement steps (1)–(3) in polynomial time.

5.1 Compression by an acyclic morphism

Let $\mathtt{A}$ be a finite alphabet and $\mathtt{S} = \{z_1, \ldots, z_m\}$ a finite set of non-terminal symbols. An *acyclic morphism* is a mapping $h : \mathtt{S} \to (\mathtt{S} \cup \mathtt{A})^*$ such that

$$h(z_i) \in (\mathtt{A} \cup \{z_1, \ldots, z_{i-1}\})^*.$$

We assume thus a numbering of symbols such that in string $h(z_i)$, only z_j with smaller index $j < i$ are allowed. Due to this acyclicity requirement, h induces a monoid morphism $h^* : \mathtt{S}^* \to \mathtt{A}^*$, as the limit of compositions $h, h^2 = h \circ h, \ldots$. Formally, $h^*(z_i) = h^k(z_i)$, for the smallest k with $h^k(z_i) \in \mathtt{A}^*$. Then the extension of h^* to all strings in $\mathtt{S}^*$ is as usual. Therefore each symbol z_i *represents* a nonempty string over A. Its length $\|h^*(z_i)\|$ may be exponentially larger than the size of h, defined as the sum of lengths of all strings $h(z_i)$.

Action of h^* on a symbol z may be presented by a finite tree, that we call the *derivation tree* of z. The root is labeled by z. If a node is labeled by some z', then the number of its children is equal to the length of $h(z')$. Their labels are consecutive letters from $h(z')$ and their ordering is consistent with the ordering of letters in $h(z')$. Nodes labeled by an alphabet letter are leaves. By acyclicity of h the tree is necessarily finite; the labels of its leaves store $h^*(z)$.

Lemma 1 ([9]). *Given an acyclic morphism h and two symbols $z, z' \in \mathtt{S}$, one may answer in polynomial-time (wrt. the size of h) the following questions:*

- *is $h^*(z) = h^*(z')$?*
- *is $h^*(z)$ a prefix of $h^*(z')$?*

A relevant parameter of a symbol z, wrt. an acyclic morphism h, is its *depth*, written $\mathtt{depth}_h(z)$, and defined as the longest path in the derivation tree of z. A depth of h, $\mathtt{depth}(h)$, is the greatest depth of a symbol.

An acyclic morphism h is *binary* if $\|h(z_i)\| \leq 2$, for all $z_i \in \mathtt{S}$. Any acyclic morphism h may be transformed to the equivalent binary one: replace each $h(z_i)$ of length greater than 2 with a balanced binary tree, using $\|h(z_i)\| - 2$ auxiliary symbols. Note that the depth d of h may increase to $d \log d$.

Lemma 2. *Given a binary acyclic morphism h, a symbol $z \in \mathtt{S}$, and $k < \|h^*(z)\|$, one may compute in polynomial-time an acyclic morphism h' extending h, such that one of new symbols of h' represents the suffix of $h^*(z)$ of length k, and $\mathtt{size}(h') \leq \mathtt{size}(h) + \mathcal{O}(\mathtt{depth}_h(z))$ and $\mathtt{depth}(h') \leq \mathtt{depth}(h)$.*

Proof. By inspecting the path in the derivation tree of z leading to the "cutting point", that is, to the first letter of the suffix of length k. For each symbol y appearing on this path, we will add its copy $\widetilde{y}$ to $\mathtt{S}$. Our intention is that $\widetilde{y}$ represents a suitable suffix of $h^*(y)$. This is achieved as follows. Assume that $h(y) = y_1 y_2$. If the path to the cutting point traverses y and y_1, we define h for $\widetilde{y}$ as: $h(\widetilde{y}) = \widetilde{y_1} y_2$. Otherwise, if the path traverses y_2, we put $h(\widetilde{y}) = \widetilde{y_2}$.

The total overhead in increase of size of h is constant. Hence by repeating this procedure along the whole path from the root, labelled by z, to the cutting point, the size of h will increase by $\mathcal{O}(\mathtt{depth}_h(z))$. Clearly, $\widetilde{z}$ represents the required suffix of $h^*(z)$. The new acyclic morphism is an extension of h, in the sense that value of h^* is preserved for all symbols that were previously in $\mathtt{S}$. $\square$

5.2 Representation of a base by an acyclic morphism

We focus on the case of BPA first, $\mathtt{V}^{\diamondsuit} = \mathtt{V}^*$. Extension to BPC, being straightforward, is discussed at the end of this section. The complexity considerations are wrt. the size N of $\varDelta$, i.e., the sum of lengths of all rules. The subroutines (i) and (ii) may be implemented in polynomial time due to Lemma 1 and 2.

In each iteration of the algorithm, a base $\mathtt{B} = (\mathtt{P}, \mathtt{E})$ will be represented succinctly by a binary acyclic morphism h. For each $X_i \notin \mathtt{P}$, the right-hand side of its decomposition $(X_i = \alpha_i) \in \mathtt{E}$ will be represented by a designated symbol $x_i \in \mathtt{S}$. Thus $d_{\mathtt{B}}(X_i) = \alpha_i = h^*(x_i)$. The set $\mathtt{P}$ of primes will be the alphabet.

In the initial step (1), cf. Section 4, it is easy to construct such h of size $\mathcal{O}(N)$ and depth $\mathcal{O}(N \log N)$. Implementation of step (2) will be similar to checking the conditions (a) and (b) in step (3) (cf. Sect. 4), to be described now.

Given a 'compressed' representation h of $\mathtt{B}$, we now show how to construct a representation h' of $\mathtt{B}'$ in each execution of step (3) of the algorithm; h' will be of size $\mathcal{O}(N^2 \log N)$ and depth $\mathcal{O}(N \log N)$. The alphabet $\mathtt{A}$ will be $\mathtt{P}'$.

We construct h' by consecutive extensions, according to step (3) of the algorithm. Initially, h' is empty and $\mathtt{A} = \{X_1\}$. For the extension step, suppose that each non-prime $X_j \notin \mathtt{P}'$, $j < i$, has already a designated symbol x_j in h' that represents α_j, where $(X_j = \alpha_j) \in \mathtt{E}'$. The most delicate point is the size of the representation of a 'candidate' α in step (3), as the decomposition α_i of X_i, if any, is finally found among the candidates.

Let $X_i \xrightarrow{a} \beta$ be a chosen norm-reducing rule. Replace occurences of non-prime X_j's in β by the corresponding x_j's. Extend h' by $h'(y_i) = \beta$, for a fresh auxiliary symbol y_i, and transform h' to the binary form (we say that we *encode* the rule in h'). This increases the size of h' by $\mathcal{O}(N)$ and its depth by at most $\log N$. To compute a representation of a candidate α, we extend h' similarly as above, to encode a rule $X_j \xrightarrow{a} \gamma$, by $h'(y_j) := \gamma$. Then we apply Lemma 1 to check whether $h^*(y_j)$ is a prefix of $h^*(y_i)$, and if so, apply Lemma 2 to y_i, so that the newly added symbol $\widetilde{y_i}$ represents the required suffix of $h^*(x_i)$. This increases the size of h' by $\mathcal{O}(\mathtt{depth}(h'))$ and keeps its previous depth. Finally, we compose X_j, represented by x_j, with $\widetilde{y_i}$: $h(x_i) = x_j \widetilde{y_i}$, according to step (3) of the algorithm. The cumulative increase of size and depth, after at most N repetitions of the above procedure, fits the required bounds, $\mathcal{O}(N^2 \log N)$ and $\mathcal{O}(N \log N)$, on the size and depth of h', respectively.

To test the conditions (a) and (b) we invoke the subroutine (i) for the successors of X_i and the candidate α. This involves encoding the rules of X_i and X_j in h', in the similar way as above. The condition (b) refers to $\mathtt{B}$, so we need to merge h' with h. As the total number of candidates is polynomial, it follows that the whole algorithm runs in polynomial time.

Remark 1. The input process definition may be well given in a compressed form, i.e., by an acyclic morphism representing the right-hand sides of all rules.

Implementation for BPC. The only difference is that there may be more threads than one. Hence instead of single symbol x_i, representing decomposition of X_i, we need to have a tuple of symbols, $x_i^1, \ldots, x_i^t$, where t is the number of threads, to represent the content of each thread separately. The overall idea is that the algorithm should work thread-wise, i.e., process separately the strings appearing in each thread. E.g., the subroutines (i) and (ii) may be implemented analogously as for BPA, by referring to Lemma 1 and 2 for each thread.

6 Conclusions

We have provided an evidence that the bisimulation equivalence in both normed BPA and BPP can be solved by essentially the same polynomial-time algorithm.

The algorithm works correctly in a feasible fragment of normed BPC. An interesting open question remains whether the procedure may be extended to work for all of BPC with transitive `D`. This would probably require a quite different method, as the core ingredient of the approach of this paper is that the bisimulation equivalence is a congruence, which is not the case in general. Another open question is whether our setting can solve the normed BPA vs. BPP problem (disjoint union of normed BPA and BPP needs not be feasible, cf. Example 2).

It also remains to investigate possible applications of the BPC framework as an abstraction of programs, e.g., of multi-core computations.

Concerning the expressibility, normed BPC class and normed PA seem to be incomparable, even with respect to the trace equivalence (e.g., the process in Example 1 is not expressible in normed PA).

References

1. O. Burkart, D. Caucal, F Moller, and B. Steffen. Verification of infinite structures. In *Handbook of Process Algebra*, pages 545–623. Elevier, 2001.
2. S. Christensen, H. Hüttel, and C. Stirling. Bisimulation equivalence is decidable for all context-free processes. *Inf. Comput.*, 121(2):143–148, 1995.
3. S. Fröschle and S. Lasota. Normed processes, unique decomposition, and complexity of bisimulation equivalences. In *Proc. INFINITY'06*, ENTCS. To appear.
4. Y. Hirshfeld and M. Jerrum. Bisimulation equivalence is decidable for normed process algebra. In *ICALP'99*, volume 1644 of *LNCS*, pages 412–421, 1999.
5. Y. Hirshfeld, M. Jerrum, and F. Moller. A polynomial algorithm for deciding bisimilarity of normed context-free processes. *Theor. Comput. Sci*, 158(1-2):143–159, 1996.
6. 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.
7. P. Jančar. Bisimilarity of Basic Parallel Processes is PSPACE-complete. In *Proc. LICS'03*, pages 218–227, 2003.
8. P. Jančar, M. Kot, and Z. Sawa. Normed BPA vs. normed BPP revisited. In *Proc. CONCUR'08*, volume 5201 of *LNCS*, pages 434–446. Springer-Verlag, 2008.
9. M. Karpiński, W. Rytter, and A. Shinohara. An efficient pattern-matching algorithm for strings with short descriptions. *Nord. J. Comput.*, 4(2):172–186, 1997.
10. S. Lasota and W. Rytter. Faster algorithm for bisimulation equivalence of normed context-free processes. In *Proc. MFCS'06*, volume 4162 of *LNCS*, pages 646–657. Springer-Verlag, 2006.