# Partially-commutative context-free processes: expressibility and tractability[☆]

Wojciech Czerwiński

*Institute of Informatics, University of Warsaw*

Sibylle Fröschle

*University of Oldenburg*

Sławomir Lasota

*Institute of Informatics, University of Warsaw*

**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 algorithmic 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. Expressibility issue of the new class, in comparison with the normed PA class, is also tackled in the paper.

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) [2].

The bisimulation equivalence is decidable both in BPA and BPP [4, 11]. Under the assumption of *normedness* the polynomial-time algorithms exist [9, 10]. 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. [2] (Chapt. 9, p. 573):

"These algorithms are both based on an exploitation of the decomposition

[☆]The first and the last author acknowledge a partial support by Polish government grants no. N206 008 32/0810 and N N206 356036, respectively. The second author acknowledges a partial support by DAAD PPP Polen DeRCC, Projekt-ID 50725248.

properties enjoyed by normed transition systems; however, despite the apparent similarity of the two problems, different methods appear to be required."

In [8] a decision procedure was given for normed PA, a superclass of both BPA and BPP. It is however very complicated and has doubly exponential nondeterministic time complexity. In [12] 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 (Basic Partially-Commutative Algebra), build on the underlying concept of *(in)dependence* of elementary processes (non-terminal symbols). 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.

A natural question arises about a relationship between BPC and PA. Inituitively, the two classes differ in the way parallelism is introduced. In PA, the parallel composition (interleaving) is specified by a rewrite rule (production). In BPC, on the other hand, the parallel composition is a characteristic of a pair of elementary processes. Thus, in PA parallelism occurs whenever a production is used and it has a 'static' character, i.e., the scope of a parallel composition is static and does not change while a process evolves. Contrarily, in BPC parallelism occurs whenever a pair of 'independent' variables occur simultaneously in a process and it has a 'dynamic' character, i.e., the scope of parallel composition changes while a process evolves.

Section 2 is devoted to comparison of expressive power of BPC and PA. We show, in particular, that normed BPC is not a subclass of normed PA.

Our first main result is the proof of the unique decomposition property for normed BPC with a transitive dependence relation (Thm 3 in Section 3). Then in Sections 4–6 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. In Sect. 7 we provide a simple sufficient condition for feasibility. The rest of this section is devoted to sketching of the main technical points crucial for the algorithm.

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 [10] 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 [7] that this latter step yields *the greatest norm-reducing bisimulation* contained in $R \cap \exp(R)$, or equivalently, the norm-reducing bisimulation equivalence *relativized* to $R \cap \exp(R)$.

The feasibility condition, obviously satisfied by normed BPA and BPP, requires the bisimulation refinement step 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 [10] with the BPA procedure based on compressed string algorithms [14].

A preliminary version of this paper appeared as [5].

## 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 symmetric 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 $\mathtt{V}^{\diamondsuit}_{\mathtt{I}} = \mathtt{V}^*/{\sim_{\mathtt{I}}}$ from now on; we call $\mathtt{V}^{\diamondsuit}_{\mathtt{I}}$ 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 $\alpha$, $\beta$, .... Composition of $\alpha$ and $\beta$ in $\mathtt{V}^{\diamondsuit}$ is written $\alpha\beta$. Empty process (identity in $\mathtt{V}^{\diamondsuit}$) will be written as $\epsilon$. Our development is based on the decision to interpret right-hand sides $\alpha$ 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 process algebra) in short.

Formally, a *BPC process definition* $\Delta$ 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:

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

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:*

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

*there are, among the others, the following transitions:*

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

We would like to stress that the independence is defined on variables, and not on alphabet letters, as in trace theory [6].

In the sequel we will pay special attention to the case when $\mathtt{D}$ is transitive (and thus is an equivalence). We call this class *transitive BPC*, or *tBPC* in short. The process definition in Example 1 is not transitive: $(B, W)$ and $(W, C)$ belong do $\mathtt{D}$ but $(B, C)$ does not. Both BPA and BPP are subclasses of tBPC.

Equivalence classes of $\mathtt{D}$ we will call *threads* in the sequel. Intuitively, a tBPC process (i.e., an equivalence class of $\sim_{\mathtt{I}}$) may be seen as a collection of strings over non-terminal symbols, one string for each thread. The strings themselves we will call threads as well.

**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 $(\alpha, \beta)$ of processes such that for all $a \in \mathcal{A}$:

1. whenever $\alpha \stackrel{a}{\longrightarrow} \alpha'$, there is $\beta'$ with $\beta \stackrel{a}{\longrightarrow} \beta'$ 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* $B \sim B'$*:*

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

In the sequel we will always assume that $\Delta$ 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 $\epsilon$. The length of the shortest such sequence is *the norm* of $X$, written $|X|$. Norm extends additively to all processes.

## 2. Partially commutative context-free languages

Partially-commutative processes exhibit both sequential behaviour as well as parallel (interleaving) one. In this respect, our framework is similar to so called *Process Algebra*, *PA* in short [1, 15]. This section is devoted to relating expressibility of the two classes.

### *2.1. BPC languages*

For expressibility considerations in this section we choose to work on the level of languages generated, instead of (bisimulation classes of) transition systems (processes). For a chosen initial non-terminal $X$, the language generated by a process definition $\Delta$ contains precisely those words $a_1 \ldots a_k$ with $X \xrightarrow{a_1} \alpha_1 \ldots \xrightarrow{a_k} \alpha_k = \epsilon$, i.e., words labeling some path from $X$ to $\epsilon$ in the transition system induced by $\Delta$. Any such path we call a *derivation* of the word $a_1 \ldots a_k$ (note that derivations are 'left-most' only). For instance, if in Example 1 the initial process is $P$, the language generated is

$$L_1 = \bigcup_{n \geq 1} a^n s\, L^{(n)}, \qquad \text{where } L^{(n)} = b^n t \mid u c^n, \tag{2}$$

i.e., $L^{(n)}$ contains all the interleavings of the two words $b^n t$ and $uc^n$. Equivalently, BPC languages are those generated by Greibach context-free grammars, extended with additional productions: $XY \longrightarrow YX$, for each pair $(X, Y)$ of independent non-terminals.

Our choice is motivated by the fact that the incomparability results are stronger when stated on the level of languages: e.g., if one shows a language definable in BPC but not in PA, one immediately obtains that the corresponding normed BPC process is not bisimilar to any normed PA process.

### *2.2. PA languages*

In accordance with our choice, we view PA merely as a class of languages, generated by context-free grammars extended with interleaving (binary shuffle). A *PA grammar* is a set of non-terminal symbols together with a finite set of productions of one of the following types:

$$X \longrightarrow a \qquad X \longrightarrow Y; Z \qquad X \longrightarrow Y \mid Z, \tag{3}$$

where $X, Y$ and $Z$ are non-terminals, $a$ is an alphabet letter, and ';' and '|' stand for the sequential composition (concatenation) and parallel composition (interleaving), respectively. The last two types of productions we call sequential and parallel, respectively. The meaning of the two compositions will be as their names suggest: in particular, if '|' does not occur in the grammar, we obtain context-free grammars (in Chomsky normal form); and if ';' does not occur, we obtain multiset languages (essentially semi-linear sets). Note that in the latter case one would obtain the BPP-languages (cf. [3]) if the grammar was assumed in the Greibach normal form.

Our definition of PA grammars resembles Chomsky normal form. We have chosen this variant as it gives the simplest definition and the largest class of languages; e.g., the Greibach PA grammars, with productions of the form:

$$X \longrightarrow a \qquad X \longrightarrow a;(Y;Z) \qquad X \longrightarrow a;(Y \mid Z),$$

may be easily transformed to the Chomsky form used by us.

Similarly as for the pure context-free grammars, one naturally defines a notion of *derivation tree*: a binary tree, with each leaf labeled by an alphabet letter, and each inner node labeled by a production (its left-hand side we call a symbol of a node). We assume that the right-hand side of the production labeling an inner node is consistent with the symbols of children of that node. According to a label, we call inner nodes either *sequential* or *parallel*. The language generated by the grammar, for a fixed initial non-terminal $X_0$, contains all the words that are *induced* by a derivation tree whose root's symbol is $X_0$. Inducing a word is defined recursively: a sequential (resp. parallel) node induces the concatenation (resp. any interleaving) of any pair of words induced by the left and the right child.

### *2.3. BPC vs PA*

For showing that BPC languages are not included in PA languages, it is sufficient to consider language $L_1$ specified in (2). However, in the sequel we will pay special attention to the case when D is transitive; and hence we aim at showing that tBPC languages are not included in PA either. Language $L_1$ is not a good witness as it is not in tBPC (this follows from Lemma 2 below, the pumping lemma for tBPC).

For showing that tBPC is not included in PA, we consider the language $L_2$ generated by the following grammar with the initial non-terminal $S$:

$$\begin{array}{llll} S \xrightarrow{s} \epsilon & S \xrightarrow{a} SA & A \xrightarrow{c} A' & A' \xrightarrow{a} \epsilon \\ & S \xrightarrow{b} SB & B \xrightarrow{c} B' & B' \xrightarrow{b} \epsilon \end{array}$$

The dependency D partitions the non-terminals into equivalence classes $\{S, A, B\}$, $\{A'\}$ and $\{B'\}$. If we remove $c$ from all the words in $L_2$ we obtain

$$L_3 = \{wsv : w, v \in \{a, b\}^*, \#w(a) = \#v(a), \#w(b) = \#v(b)\}, \qquad (4)$$

where by $\#w(a)$ we mean the number of occurences of $a$ in $w$. If $L_2$ was in PA, $L_3$ would be in PA as well, as PA is clearly closed under images of morphisms. Thus knowing that $L_3$ is not in PA, which we prove in Lemma 1 below, we deduce the following:

**Theorem 1.** *tBPC is not included in PA.*

**Lemma 1.** *$L_3$ is not generated by a PA grammar.*

Proof. For the sake of contradiction, assume $L_3$ is generated by a PA grammar $G$. Partition the non-terminals into symbols that generate some word containing $s$, and symbols that do not; and call them $s$-symbols and non-$s$-symbols, respectively. In the sequel we will rely on the fact that $s$ appears precisely once in any word from $L_3$, and that in a derivation tree one may substitute a subtree with another one with the same root symbol. Thus in particular, each word generated by an $s$-symbol contains necessarily $s$.

Consider a derivation tree $t$ for any word $wsv \in L_3$. The unique path leading from the root to the leaf labeled by $s$ let us call *the $s$-path*. Observe that an $s$-symbol may only appear on the $s$-path and a non-$s$-symbol may only appear outside the $s$-path. Knowing that the number of occurences of $a$ and $b$ on both sides of the $s$-path is the same, cf. (4), we deduce that each production labeling a node of the $s$-path is necessarily sequential $(*)$. Indeed, assume a parallel production $X \longrightarrow Y \mid Z$ labels a node of the $s$-path. Wlog. let $X$ and $Y$ be $s$-symbols and $Z$ be a non-$s$-symbol. Let $u$, $u'$ be words induced by $Y$-node and $Z$-node, respectively, contributing to derivation of the word $wsv \in L_3$. Clearly there are at least two different interleavings of $u$ and $u'$, both induced by the $X$-node. A crucial observation is that the two interleavings may be chosen such that the $s$ symbol, appearing in $u$, is placed in the interleaving in two different positions in the word $u'$. Thus at least one of the interleavings must lead to violation of the condition (4) in a word induced by the tree $t$, thus belonging to $L_3$. Condition $(*)$ is proved.

Now consider a non-$s$-symbol $X$ appearing in $t$. The number of occurences $\#a(u)$ of $a$ in all words $u$ generated by $X$ is necessarily the same, and the same applies to $\#b(u)$. Indeed, otherwise one gets a similar contradiction as above by considering two words induced by the $X$ node, differing in the number of occurences of $a$ or $b$.

As a consequence $X$ generates a finite language which may clearly be defined by a context-free grammar, say $G_X$.

If we apply the last observation to the very first non-$s$-symbol $X$ on every path in $t$ (except the $s$-path), and extend $G$ by the productions from $G_X$, we obtain a tree without parallel nodes. As $G_X$ does not depend on the particular derivation tree $t$ chosen, and the word $wsv \in L_3$ was chosen arbitrary, we conclude that $L_3$ is generated by a context-free grammar (the parallel productions may be safely removed from $G$). As $L_3$ is clearly not context-free, we obtain a contradiction and thus complete the proof. □

*2.4. BPC vs tBPC*

We derive now the pumping lemma for tBPC. It completes the pumping lemmas for regular, context-free and BPP-languages (for the latter cf. [3]). We use it to prove that tBPC is strictly included in BPC.

**Lemma 2 (Pumping lemma for tBPC).** *For any tBPC language $L$ there is some $n \geq 0$ such that any $w \in L$ of length at least $n$ may be split into words $w = xyz$, so that there is a nonempty word $t$ and (possibly empty) word $u$ such that for each $m \geq 0$, $xt^m yu^m z$ is in $L$.*

Proof. Let us consider a fixed tBPC grammar. Recall that equivalence classes of `D` we call threads and that a process may be seen as a collection of strings, one string for each thread. Each of the strings we called thread as well.

A non-terminal we call *recurrent* if

$$X \xrightarrow{t} X\beta \tag{5}$$

for some nonempty word $t$ and some process $\beta$. As a direct conclusion one obtains:

$$X \xrightarrow{t^m} X\beta^m \tag{6}$$

for every $m$. Note that wlog. we may assume that non-terminals appearing in $\beta$ in (5) are all dependent with $X$ – otherwise, those among them that are not, can be swapped with $X$ and made to contribute to generating the word $t$.

Consider now a derivation of a word $w$ from the starting non-terminal, say $S$. Assuming $w$ is sufficiently long, some recurrent non-terminal $X$ must be used in the derivation:

$$S \xrightarrow{x} X\alpha \xrightarrow{v} \epsilon, \tag{7}$$

where $w = xv$. It is sufficient to know that the length of $w$ is at least exponential wrt. the size of the grammar. Note that $X$ appears *active* in $X\alpha$, i.e., capable to contribute to the next step of $X\alpha$. Combining (7) with (6) we get:

$$S \xrightarrow{x} X\alpha \xrightarrow{t^m} X\beta^m\alpha. \tag{8}$$

Let us analyze in more detail the derivation of the word $v$ according to (7). We will pay special attention to the thread of $X$. Intuitively, imagine $X$ is colored red and all non-terminals appearing in $\alpha$ are colored green. Thus, initially the active non-terminal from the $X$'s thread is red and all other non-terminals from that thread are green. Assume also that color of a non-terminal is inherited through an application of a production; that is, non-terminals derived from $X$ will be all colored red, and non-terminals derived from $\alpha$ will be all colored green. In the course of the derivation of $v$ it will finally happen that all the non-terminals from the $X$'s thread are green (in particular, when the thread finally gets empty). Let $\gamma$ be the first such configuration. We split the derivation into:

$$X\alpha \xrightarrow{y} \gamma \xrightarrow{z} \epsilon, \tag{9}$$

where $v = yz$. Our aim is now to appropriately merge derivations (8) and (9). A crucial observation is that the derivation of $y$ according to (9) is also possible from $X\beta^m\alpha$,

$$X\beta^m\alpha \xrightarrow{y} \beta^m\gamma. \tag{10}$$

To see this, recall that we may assume that all of non-terminals appearing in $\beta$ are dependent with $X$. As the derivation (9) of $y$ only involved $X$ and those of non-terminals in $\alpha$ that are independent with $X$ (and thus does not involve those variables appearing in $\alpha$ that are dependent with $X$) it may be repeated in presence of $\beta^m$ (note that transitivity of D is essentially used here).

Finally, let $u$ be any word generated by $\beta$. Thus we have:

$$\beta^m \xrightarrow{u^m} \epsilon. \tag{11}$$

Our coloring argument yields that $\beta$ becomes active after generating $y$ in (10). By merging (8), (10) and (11) we get

$$S \xrightarrow{xt^m} X\beta^m\alpha \xrightarrow{y} \beta^m\gamma \xrightarrow{u^m} \gamma \xrightarrow{z} \epsilon, \tag{12}$$

so $xt^myu^mz$ is generated, for every $m > 0$. □

As a corollary of the pumping lemma we get:

**Theorem 2.** *tBPC is a strict subclass of BPC.*

Proof. Using Lemma 2 we will demonstrate that $L_1$, cf. (2), being in BPC, is not in tBPC.

Consider words $w_n = a^nsb^ntuc^n \in L_1$, $n > 0$. We need to show the following: for sufficiently large $n$, whatever split $w_n = xyz$ is considered, and whatever two words $v, v'$ are chosen, $v$ nonempty, the word $xv^myv'^mz$ is not in $L_3$ for some $m > 0$.

Let a split $w_n = xyz$ be induced by two 'cutting points' inside $w_n$. Recall that $L_1 = \bigcup_{n\geq 1} a^n s\, L^{(n)}$, where $L^{(n)} = b^nt \mid uc^n$. The intuition is as follows: knowing that $t$ precedes $u$ in a word from $L_1$, we are sure that all occurrences of $b$ preced all occurrences of $c$ in that word. In $w_n$ there are thus three segments, the $a$-, $b$- and $c$-segment, of equal lengths. As long as $t$ precedes $u$, which is always the case in any word of the form $xv^myv'^mz$ (there may be not more than one $t$ or $u$), we are sure that all the three segments are separated. Thus we can not manage keeping their equal lengths with two cutting points only. □

*2.5. BPC vs trace context-free languages*

For completeness we remark on relationship with another approach, that of trace theory [6], where independence is imposed on alphabet letters instead of non-terminals. It is easy to find a tBPC language which is not a trace closure of any context-free language, for example the language $abcL$, where $L$ contains all the words with the same number of occurences of $a$, $b$ and $c$.

In the opposite direction, $L_1$ is clearly a trace closure of a context-free language not definable in tBPC.

Further comparison of expressive power deserves a study. It seems that all the classes, namely tBPC, PA and trace context-free languages are imcomparable; likewise when BPC is considered instead of tBPC.

## 3. The unique decomposition for normed 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 $\Delta$ 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 $\alpha$ and $\beta$ are related then $|\alpha| = |\beta|$.

Proof. Assume the contrary, i.e., $|\alpha| < |\beta|$. Consider the shortest path from $\alpha$ to the empty process. Even if it may be matched by a sequence of moves starting in $\beta$, and leading to a process $\beta'$, say, then a move of $\beta'$ may not be matched. □

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 $\Delta$, 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 $\alpha$ there is some $\gamma \in \mathtt{P}^{\diamondsuit}$ with $\alpha \equiv \gamma$; in such case $\gamma$ is called a *prime decomposition* of $\alpha$. 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; such a congruence we also call *unique decomposition congruence*. 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 3 below) assumed that $\mathtt{D}$ is transitive (hence $\mathtt{D}$ is an equivalence). Recall that equivalence classes of $\mathtt{D}$ are called threads and that a process may be seen as a collection of strings over $\mathtt{V}$, called threads as well.

This concrete representation will be extensively exploited in the sequel. In the rest of this section we assume $\mathtt{D}$ to be transitive.

For $\alpha, \gamma \in \mathtt{V}^\diamond$ we say that $\alpha$ *masks* $\gamma$ if any thread nonempty in $\gamma$ is also nonempty in $\alpha$. The notion will be useful when one needs to know that a transition performed by $\alpha\gamma$ is for sure performed by $\alpha$. 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 $\alpha$ and $\beta$ mask $\gamma$ then $\alpha \approx \beta$.

**Proposition 2.** A unique decomposition congruence is strongly right-cancellative.

Proof. Let $\equiv$ be a congruence. Denote the unique prime decomposition of $\alpha$ wrt. $\equiv$ by $d_\equiv(\alpha)$. Assume $\alpha\gamma \equiv \beta\gamma$. We have

$$d_\equiv(\alpha)d_\equiv(\gamma) = d_\equiv(\alpha\gamma) = d_\equiv(\beta\gamma) = d_\equiv(\beta)d_\equiv(\gamma).$$

Considering this equality thread-wise, one gets $d_\equiv(\alpha) = d_\equiv(\beta)$, hence $\alpha \equiv \beta$. □

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

Proof. Assume $(\alpha\gamma, \beta\gamma) \in \exp(\approx)$. Assuming $\gamma$ to be masked, each transition of $\alpha$ ($\beta$, resp.) is matched by a transition of $\beta$ ($\alpha$, resp.): $(\alpha, \beta) \in \exp(\{(\alpha', \beta') : \alpha'\gamma \approx \beta'\gamma\})$. As $\approx$ is strongly right-cancellative, we deduce $(\alpha, \beta) \in \exp(\approx)$. The proof for n-r-exp($\approx$) is identical. □

Note that we did not require that exp($\approx$) or n-r-exp($\approx$) is a congruence.

A counterexample to the unique decomposition property of $\equiv$, if any, is a pair $(\alpha, \beta)$ of processes from $\mathtt{P}^\diamond$ 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 $(\alpha\gamma, \beta\gamma)$ is a minimal $\equiv$-counterexample and both $\alpha$ and $\beta$ mask $\gamma$ then $\alpha \equiv \beta$.

**Theorem 3.** *Assume* $\mathtt{D}$ *to be transitive. Then each weakly right-cancellative congruence that is a n-r-bisimulation has the unique decomposition property.*

**Remark 1.** Theorem 3 a generalization of the unique decomposition in BPA and BPP (cf. [9] and [10], 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 $\sim$. Even if it is not clear at this point if the unique decomposition property holds for the bisimulation equivalence, it is indeed the case, as stated in Prop. 10 in Section 4.3. Interestingly, we do not need to prove it prior to the design of the algorithm! ( For the correctness of the algorithm one only needs the unique decomposition property of bisimulation approximants, to be defined in Section 4. ) For the sake of clarity we state here explicitely the following corollary, keeping in mind that it only follows from Prop. 10:

**Corollary 1.** *Assume* $\mathtt{D}$ *to be transitive. Then the bisimulation equivalence has the unique decomposition property.*

The rest of this section is devoted to the proof of the theorem.

**Proof of Theorem 3.** 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 $(\alpha, \beta)$.

We say that a transition of one of $\alpha$, $\beta$ 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 $\alpha, \beta$ may be matched with a transition of the other. Due to the minimality of the counterexample $(\alpha, \beta)$, *any* prime decompositions of the resulting processes, say $\alpha'$ and $\beta'$, are necessarily identical. For convenience assume that each right-hand side of $\Delta$ was replaced by a prime decomposition wrt. $\equiv$. Thus $\alpha', \beta'$ 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 $\alpha$ restricted to the $i$th thread we denote by $\alpha_i \in \mathtt{V}_i{}^*$. Hence $\alpha = \alpha_1 \ldots \alpha_t$ and the order of composing the processes $\alpha_i$ is irrelevant.

A (n-r-)transition of $\alpha$, or $\beta$, is always a transition of the first variable in some $\alpha_i$, or $\beta_i$; 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 $(\alpha, \beta)$.

**Claim 1.** *For each* $i \leq t$*, one of* $\alpha_i$*,* $\beta_i$ *is a suffix of the other.*

Proof. Suppose that some thread $i$ does not satisfy the requirement, and consider the longest common suffix $\gamma$ of $\alpha_i$ and $\beta_i$. Thus $\gamma$ is masked in $\alpha$ and

$\beta$. As $\equiv$ is weakly right-cancellative, $\gamma$ must be necessarily empty – otherwise we would obtain a smaller counterexample. Knowing that the last letters of $\alpha_i$ and $\beta_i$, say $P_\alpha$, $P_\beta$, 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 $P_\alpha$, $P_\beta$ will still appear in the resulting processes $\alpha'$, $\beta'$, thus necessarily $\alpha' \neq \beta'$ – a contradiction to the minimality of $(\alpha, \beta)$.

CASE 2: $\|\alpha_i\| = 1$, $\|\beta_i\| \geq 2$ (or a symmetric one). Thus $\alpha_i = P_\alpha$. As $P_\alpha$ is prime, some other thread is necessarily nonempty in $\alpha$. Perform any n-r-transition from that other thread. Irrespective of a matching move in $\beta$, the last letters $P_\alpha$ and $P_\beta$ 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 $\alpha$ and $\beta$. Asssume wlog. $|P_\alpha| \geq |P_\beta|$. Perform any n-r-transition in $\alpha$ from a thread different than $i$. Irrespective of a matching move in $\beta$, in the resulting processes $\alpha'$, $\beta'$ the last letter $P_\alpha$ 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 $(\alpha, \beta)$. If $\alpha_i$, say, is a proper suffix of $\beta_i$, then a n-r-transition of $\alpha_i$ may not be matched in $\beta$. □

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

**Claim 3.** *A n-r-transition of one of $\alpha$, $\beta$ 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 $\alpha_i$ decreases $|\alpha_i|$. By minimality of $(\alpha, \beta)$, $|\beta_i|$ 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 $\alpha$, may not be matched in $\beta$, due to Claim 3, so that the norms of $\alpha_j$ and $\beta_j$ become equal as implied by minimality of $(\alpha, \beta)$. □

**Claim 5.** *One of $\alpha$, $\beta$, say $\alpha$, has only one nonempty thread.*

PROOF. Consider the greatest (wrt. $\leq$) active variable and assume wlog. that it appears in $\alpha$, hence it does not occur in $\beta$. We claim $\alpha$ has only one nonempty thread. Indeed, if some other thread is nonempty, a n-r-transition of this thread can not be matched in $\beta$ as required by minimality of $(\alpha, \beta)$. □

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

**Claim 6.** *$|P_i|$ is greater than norm of any variable appearing in $\gamma_i$.*

PROOF. Consider any thread $\beta_j = P_j\gamma_j$ nonempty in $\beta$. We know that $|P_i| \geq |P_j|$ as any n-r-transition from $P_i$ must produce $P_j\gamma_j$. As the thread $i$ is empty in $\beta$, the norm of $P_j$ must be sufficiently large to "produce" all of $\gamma_i$ 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 $\alpha$, 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 $\beta'$ be the resulting process, and let $\alpha'$ 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 $\alpha'$ (as both $P_i$'s at the beginning of $P_iP_i\gamma_i$ were necessarily involved in the matching transitions, and thus all variables that appear in $\alpha'$, including those in $\gamma_i$, are of smaller norm) while $P_i$ clearly appears in $\beta'$. Thus $\alpha' \equiv \beta'$, $\alpha' \neq \beta'$ and $|\alpha'| = |\beta'|$ is smaller than $|\alpha| = |\beta|$ – a contradiction to the minimality of the counterexample $(\alpha, \beta)$. This completes the proof of the theorem.

## 4. The algorithm

From now on we only consider normed BPC process definitions $\Delta$ with a transitive dependence relation D.

We prefer to separate description of the algorithm from the implementation details. In this section we specify a *feasiblity* condition that must be satisfied by $\Delta$ for the algorithm to work, and provide an outline of the algorithm only. In Section 5 we explain how each step of the algorithm can be implemented. Without further refinement, this would give an exponential-time procedure. Finally, in Section 6 we provide the polynomial-time implementation of crucial subroutines. Altogether, Sections 4–6 contain the proof of our main result:

**Theorem 4.** *The bisimulation equivalence is decidable in polynomial time for feasible BPC process definitions.*

At first reading, Section 5 and 6 may be skipped.

### *4.1. Feasibility*

The algorithm, to be outlined in this section, will compute a finite representation of the bisimulation equivalence $\sim$, relying on the unique decomposition property. For the application of Theorem 3 to be possible, we at least need to know that the bisimulation equivalence is a congruence. Unfortunately, this is not always the case, as shown in Example 2. This motivates introducing the feasibility condition below.

For a norm-preserving equivalence $\equiv$ over processes, let $\sim^{\equiv}_{\text{n-r}}$ denote the union of all n-r-bisimulations contained in $\equiv$. It witnesses most of the typical properties of bisimulation equivalence. Being the union of n-r-bisimulations, $\sim^{\equiv}_{\text{n-r}}$ is a n-r-bisimulation itself, in fact the greatest n-r-bisimulation that is contained in $\equiv$. It admits the following fix-point characterization:

**Proposition 4.** $(\alpha,\beta) \in \sim^{\equiv}_{\text{n-r}}$ iff $\alpha \equiv \beta$ and $(\alpha,\beta) \in \text{n-r-exp}(\sim^{\equiv}_{\text{n-r}})$.

PROOF. The *only-if* implication is obvious. For the opposite implication, we argue that the relation $\{(\alpha,\beta) : \alpha \equiv \beta \text{ and } (\alpha,\beta) \in \text{n-r-exp}(\sim^{\equiv}_{\text{n-r}})\}$ is a n-r-bisimulation contained in $\equiv$, using the only-if implication. □

Moreover $\sim^{\equiv}_{\text{n-r}}$ is clearly an equivalence as $\equiv$ is assumed to be so. The relation $\sim^{\equiv}_{\text{n-r}}$ may be thus seen as the bisimulation equivalence relativized to pairs of processes related by $\equiv$ and to norm-reducing moves only.

The relativized bisimulation equivalence will play a crucial role in the algorithm, that will work by consecutive refinements of a current congruence until it finally stabilizes. However, instead of the classical refinement step $\equiv \mapsto \equiv \cap \exp(\equiv)$, we prefer to use the following one:

$$\equiv \;\mapsto\; \sim^{\equiv \cap \exp(\equiv)}_{\text{n-r}} .$$

This function will be refered to as *the refinement step*, and $\sim^{\equiv \cap \exp(\equiv)}_{\text{n-r}}$ will be called *the refinement of* $\equiv$.

As the algorithm will work by iterative improvement of the approximating congruence, an important issue is the choice of the initial approximation $\equiv_0$. In this section, the Reader may think of $\equiv_0$ to be the norm equality:

$$\alpha \equiv_0 \beta \iff |\alpha| = |\beta|.$$

However, we prefer to make our setting parametric with respect to the initial approximation. In principle, any congruence between bisimulation equivalence and the norm equality may be taken as $\equiv_0$. (We thus assume that $\equiv_0$ is norm-preserving.)

A congruence that is included in $\equiv_0$ we call *correct*. A BPC process definition $\Delta$ is *feasible* if it satifies the following:

**Assumption 1 (Feasibility).** *The refinement step preserves congruences: if $\equiv$ is a correct congruence then the refinement of $\equiv$ is a congruence too.*

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

**Proposition 5.** Every normed BPA or BPP process definition is feasible.

PROOF. Assume a norm-preserving congruence $\equiv$. Both in BPA and BPP, whenever $(\alpha,\alpha'),(\beta,\beta') \in \equiv \cap \exp(\equiv)$, then a transition of $\alpha\beta$ may be always matched by a transition of $\alpha'\beta'$ so that the resulting processes are related by $\equiv$. Hence $(\alpha\beta,\alpha'\beta') \in \equiv \cap \exp(\equiv)$ and thus $\equiv \cap \exp(\equiv)$ is a congruence.

It remains to demonstrate that for a congruence $\equiv$, the relation $\sim^{\equiv}_{\text{n-r}}$ is a congruence too. Observe that $\sim^{\equiv}_{\text{n-r}}$ is the intersection of the descending chain of relations $\equiv_1 := \equiv \cap \text{n-r-exp}(\equiv)$, $\equiv_2 := \equiv_1 \cap \text{n-r-exp}(\equiv_1), \ldots$. Similarly as above one argues that both in BPA and BPP, each of the approximants $\equiv_i$ is a congruence. Thus the intersection of the descending chain of congruences is a congruence too. □

As in the sequel we will only apply the assumption to $\equiv$ with the unique decomposition property, we could equally well assume a weaker property:

**Assumption 2 (Weak feasibility).** *If $\equiv$ is a correct unique decomposition congruence then the refinement of $\equiv$ is a congruence.*

We would like to stress that both feasibility and weak feasibility are parametric wrt. a chosen initial approximation $\equiv_0$, as they only apply to correct congruences.

A sufficient condition for weak feasibility, subsuming both BPA and BPP, will be given in section 7. Interestingly, the initial approximation will have to be different than norm equality.

*4.2. Bases*

From now on let $\Delta$ be a fixed feasible process definition with variables $\mathtt{V}$ and dependence $\mathtt{D}$; we also fix an ordering of variables $\mathtt{V} = \{X_1, \ldots, X_n\}$.

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 $X_i \notin \mathtt{P}$, $\alpha \in (\mathtt{P} \cap \{X_1, \ldots, X_{i-1}\})^{\Diamond}$ 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 $\mathtt{P}^{\Diamond}$. 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 $\mathtt{V}^{\Diamond}$ to $\mathtt{P}^{\Diamond}$. 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 $\mathtt{P}^{\Diamond}$.

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 $\mathtt{V}^{\Diamond}$: $\alpha =_{\mathtt{B}} \beta$ 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 6.** A norm-preserving congruence in $\mathtt{V}^{\Diamond}$ 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*.

*4.3. Outline of the algorithm*

Here is the overall idea. We start with the initial congruence $=_{\mathtt{B}}$ given by the initial approximation $\equiv_0$ (in this section simply the norm equality), 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: $=_{\mathtt{B}} \mapsto \sim_{\text{n-r}}^{=_{\mathtt{B}} \cap \exp(=_{\mathtt{B}})}$. We will need the following fact:

**Proposition 7.** $\sim_{\text{n-r}}^{=_{\mathtt{B}} \cap \exp(=_{\mathtt{B}})}$ is weakly right-cancellative.

Proof. For convenience, we write $\equiv$ for $=_{\mathtt{B}} \cap \exp(=_{\mathtt{B}})$. Note that $\equiv$ is an equivalence but not necessarily a congruence; by Prop. 2 and 3 applied to $=_{\mathtt{B}}$, $\exp(=_{\mathtt{B}})$ is right-cancellative, hence $\equiv$ also is. Recall that by Assumption 2 we know that $\sim_{\text{n-r}}^{\equiv}$ is a congruence.

Consider a minimal $\sim_{\text{n-r}}^{\equiv}$-counterexample $(\alpha\gamma, \beta\gamma)$ such that $\gamma$ is masked both by $\alpha$ and $\beta$. The pair $(\alpha\gamma, \beta\gamma)$, being in $\sim_{\text{n-r}}^{\equiv}$, satisfies the norm reducing expansion wrt. $\sim_{\text{n-r}}^{\equiv}$, $(\alpha\gamma, \beta\gamma) \in \text{n-r-exp}(\sim_{\text{n-r}}^{\equiv})$, by the *only if* implication of Prop. 4. Hence each n-r-transition of $\alpha\gamma$ ($\beta\gamma$, resp.) is matched by a transition of $\beta\gamma$ ($\alpha\gamma$, resp.). As $\gamma$ is always masked, these are always transitions of $\alpha$ or $\beta$, respectively, and are of the form $\alpha\gamma \xrightarrow{a} \alpha'\gamma$ (or $\beta\gamma \xrightarrow{a} \beta'\gamma$, resp.). Furthermore, the resulting processes $\alpha'\gamma$ and $\beta'\gamma$ have always the same prime decompositions, due to minimality of $(\alpha\gamma, \beta\gamma)$.

It follows that the processes $\alpha'$, $\beta'$ have always the same prime decompositions too, and thus are related by $\sim_{\text{n-r}}^{\equiv}$. This proves that $(\alpha, \beta) \in \text{n-r-exp}(\sim_{\text{n-r}}^{\equiv})$. Due to right-cancellativity of $\equiv$ we have also $\alpha \equiv \beta$. Now by the *if* implication of Prop. 4 we deduce $(\alpha, \beta) \in \sim_{\text{n-r}}^{\equiv}$. □

Prop. 7 together with Assumption 2 assure that Thm. 3 applies to $\sim_{\text{n-r}}^{=_{\mathtt{B}} \cap \exp(=_{\mathtt{B}})}$. Thus we obtain:

**Proposition 8.** The refinement step preserves unique decomposition congruences: whenever $\equiv$ is a unique decomposition congruence then $\sim_{\text{n-r}}^{\equiv \cap \exp(\equiv)}$ is a unique decomposition congruence as well.

*Outline of the algorithm:*

(1) Compute the base $\mathtt{B}$ of the initial approximation $\equiv_0$.
(2) If $=_{\mathtt{B}}$ is a bisimulation then halt and return $\mathtt{B}$.
(3) Otherwise, compute the base of the congruence $\sim_{\text{n-r}}^{=_{\mathtt{B}} \cap \exp(=_{\mathtt{B}})}$.
(4) Assign this new base to $\mathtt{B}$ and go to step (2).

This scheme is a generalization of the BPP algorithm [10]. As our setting is more general, the implementation details, to be given in the next sections, will be necessarily more complex than in [10]. 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 $\mathtt{B}$ 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 $\sim_{\text{n-r}}^{=_{\mathtt{B}} \cap \exp(=_{\mathtt{B}})}$ and that these relations play a prominent role in the algorithm (cf. [7]).

## 5. Implementation

Step (1), for $\equiv_0$ instantiated as norm equality, is easy: recalling that $|X_1| = 1$, initialize $\mathtt{B}$ by $\mathtt{P} := \{X_1\}$, $\mathtt{E} := \{X_i = {X_1}^{|X_i|} : i = 2 \dots n\}$. 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 $(X_i = \alpha) \in \mathtt{E}$, $\alpha$ 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 $\alpha$, from a thread different than that of $X_i$. As $\mathtt{B}$ is $\mathtt{I}$-preserving, $X_j\alpha = \alpha X_j$. Hence $\alpha$, 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 $\alpha$ and $\beta$ contain primes from the same threads. That is, $\alpha$ contains a prime from a thread iff $\beta$ 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 $\alpha$ but empty in $\beta$: $\alpha_t \neq \epsilon$, $\beta_t = \epsilon$. 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* [2] 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 $\alpha, \beta$ 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 $\alpha$ and $\beta$. 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 $\alpha$ and $\gamma$, knowing that all decompositions in $\mathtt{E}$ are pure.

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

Now we consider the possible transitions of $\gamma$. Let a prime $X_j$ be active in $\gamma$. 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 $\alpha$, i.e., $\alpha = X_i\alpha'$, for some $\alpha'$. Similarly as above we have $\gamma = \delta\gamma'$. A transition of $X_j$ is necessarily a transition of $\delta$, 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 $\sim_{\text{n-r}}^{=_{\mathtt{B}} \cap \exp(=_{\mathtt{B}})}$, is pure by Prop. 11, 8 and 6, 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 $\sim^{\equiv}_{\text{n-r}}$ (cf. Prop. 4).

*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 $\alpha$ 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 $\sim^{\equiv}_{\text{n-r}}$.

Proof. We show by induction on $i$ that $\mathtt{B}'$ coincides with the base of $\sim^{\equiv}_{\text{n-r}}$ on the set $\mathtt{V}_i = \{X_1, \ldots, X_i\}$. Specialized to $i = n$, this implies that $\mathtt{B}'$ computed in step (3) coincides with the base of $\sim^{\equiv}_{\text{n-r}}$.

For $i = 1$ the claim is obvious as $X_1$ is prime wrt. any congruence. Assume that $\mathtt{B}'$ coincides with the base of $\sim^{\equiv}_{\text{n-r}}$ on the set $\mathtt{V}_{i-1}$. Thus $=_{\mathtt{B}'}$ coincides with $\sim^{\equiv}_{\text{n-r}}$ on ${\mathtt{V}_{i-1}}^{\diamondsuit}$ $(*)$. If $\alpha$ is found satisfying the conditions (a) and (b), and hence $(X_i = \alpha)$ is added to $\mathtt{E}'$, then by Prop. 4 and by $(*)$ the pair $(X_i, \alpha)$ belongs to $\sim^{\equiv}_{\text{n-r}}$. On the other hand, if no $\alpha$ is found, and hence $X_i$ is prime in $\mathtt{B}'$, then again by Prop. 4 and by $(*)$ we deduce that $X_i$ is prime in $\sim^{\equiv}_{\text{n-r}}$ too. Thus $\mathtt{B}'$ coincides with the base of $\sim^{\equiv}_{\text{n-r}}$ on $\mathtt{V}_i$. □

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 $(X_i, \alpha) \in \text{n-r-exp}(=_{\mathtt{B}'})$ then $\alpha$ is necessarily among the candidates.

*Computing candidates $\alpha$:*

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 $\beta$ 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 $\gamma''$, 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 $d_{\mathtt{B}'}(\beta)$.

## 6. 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 $\alpha$, $\beta \in \mathtt{V}^{\diamondsuit}$ and $\mathtt{B}$, check if $\alpha =_{\mathtt{B}} \beta$.
(ii) Given $\alpha$, $\beta \in \mathtt{V}^{\diamondsuit}$ and $\mathtt{B}$, check if $\alpha \leq_{\mathtt{B}} \beta$. If so, compute $\gamma$ 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. This applies even if right-hand sides of the rules of $\Delta$ are represented in binary.

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 6.1, in Section 6.2 we explain how to implement steps (1)–(3) in polynomial time.

### *6.1. 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 3 ([13]).** *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 of $h$ may increase by a logarithmic factor.

**Lemma 4.** *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 $\mathtt{S}$. □

*6.2. 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 $\Delta$, i.e., the sum of lengths of all rules. The subroutines (i) and (ii) may be implemented in polynomial time due to Lemma 3 and 4.

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 5, 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. 5), 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 $\alpha_j$, where $(X_j = \alpha_j) \in \mathtt{E}'$. The most delicate point is the size of the representation of a 'candidate' $\alpha$ in step (3), as the decomposition $\alpha_i$ 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 $\beta$ 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 $\alpha$, we extend $h'$ similarly as above, to encode a rule $X_j \xrightarrow{a} \gamma$, by $h'(y_j) := \gamma$. Then we apply Lemma 3 to check whether $h^*(y_j)$ is a prefix of $h^*(y_i)$, and if so, apply Lemma 4 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 $\alpha$. 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 2.** 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.

*6.2.1. 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 3 and 4 for each thread.

## 7. Feasibility

In this section we provide a condition on a process definition $\Delta$ that guarantees weak feasibility (cf. Assumption 2 in Sect. 4). We only consider transitive

D.

By the alphabet of a variable $X$ we mean the set $\{a : X \xrightarrow{a} \alpha, \text{ for some } \alpha\}$. The alphabet of a thread will be the union of alphabets of all its variables.

We will distinguish *singleton threads*, i.e., those containing precisely one variable, from *non-singleton* ones. We call a BPC process definition $\Delta$ *disjoint* if whenever alphabets of two threads intersect then both the threads are singleton ones. In other words, the alphabet of a non-singleton thread is disjoint from the alphabets of all other threads. Both BPA and BPP are special cases: there is only one thread, or there are only singleton threads, respectively.

We will use below a refined notion of norm. For a set of threads T, *$T$-norm* $|\alpha|_T$ of a process $\alpha$ is the length $k$ of the shortest sequence of *norm-reducing* transitions $\alpha = \alpha_0 \xrightarrow{a_1} \alpha_1 \ldots \xrightarrow{a_k} \alpha_k$ such that in $\alpha_k$ all threads from $T$ are empty. In other words, the $T$-norm is the length of the shortest path to emptying all the threads from $T$ using only norm-reducing transitions. If $T$ contains all threads we have the usual norm.

Our intension is to consider all the singleton threads jointly, according to the definition of a disjoint process definition. Thus we will only consider sets $T$ that either contains all singleton threads, or no such thread; such sets $T$ we call *legal*.

In the sequel we will only investigate singleton sets $T = \{t\}$, for a non-singleton thread $t$; or $T$ containing exactly all singleton threads. $T$-norm induced by any such set we will call *local norm*. To avoid confusion, the standard norm $|\_|$ we call here *global norm*. Note that only the transitions that reduce global norm are taken into account when defining the local norms. Nevertheless, in special cases of BPA or BPP, there is no difference between global norm and (the unique) local norm.

**Proposition 16.** Assume $\Delta$ is disjoint. If a pair $(\alpha, \beta)$ of processes is related by a n-r-bisimulation then for all legal sets $T$ of threads, $|\alpha|_T = |\beta|_T$.

Proof. Similarly as the proof of Proposition 1 in Sect. 3. □

To deal with disjoint BPC process definitions we need to choose the initial approximation $\equiv_0$ different than just the norm equality (cf. Section 4.1). Let $\alpha \equiv_0 \beta$ if $\alpha$ and $\beta$ have the same local norms, as well as global norm. The local norms of variables, and hence also the base of $\equiv_0$, are computable in polynomial time, analogously as for global norm. The correct congruences (wrt. the initial approximation $\equiv_0$ chosen here) we call *local norm-preserving*. By Prop. 16 the bisimulation equivalence is local norm-preserving, thus the choice of $\equiv_0$ satisfies the requirement stated in Section 4.1: $\equiv_0$ lies between the bisimulation equivalence and the norm equality.

In the sequel we use notation $\alpha_T$ for the restriction of $\alpha$ to threads belonging to the set $T$.

**Lemma 5.** *Each local norm-preserving unique decomposition congruence $\equiv$ is* thread-wise*, i.e., if $\alpha \equiv \beta$ then $\alpha_T \equiv \beta_T$ for any legal set $T$ of threads.*

PROOF. Consider the base $\mathtt{B}$ of $\equiv$. By Propositions 6 and 11 (cf. Sect. 4 and 5, respectively) we know that $\mathtt{B}$ is pure: the prime decomposition $\gamma$ of any variable $X$ contains only variables from the thread of $X$, say $t$, and from (other) monic threads. As $\equiv$ preserves all local norms, $\gamma$ may only contain variables from $t$, in case when $t$ is non-singleton; and $\gamma$ may only contain variables from singleton (and hence necessarily monic) threads, in case when $t$ is a singleton thread. Thus $\alpha_t \equiv \beta_t$ for any non-singleton thread $t$; and $\alpha_T \equiv \beta_T$, for $T$ containing exactly all singleton threads. Hence $\equiv$ is thread-wise. □

**Theorem 5.** *Every disjoint BPC process definition is weakly feasible.*

PROOF. Assume an arbitrary local norm-preserving unique decomposition congruence $\equiv$. We will demonstrate that its refinement $\sim_{\text{n-r}}^{\equiv \cap \exp(\equiv)}$ is a congruence.

We will exploit a classical game-theoretical characterization of bisimulation equivalence, specialized to $\sim_{\text{n-r}}^{\equiv \cap \exp(\equiv)}$. Consider a two-player game between Spoiler and Duplicator. The arena consists of all the pairs of processes $(\alpha, \beta)$ such that $(\alpha, \beta) \in \equiv \cap \exp(\equiv)$. The play starts in a chosen initial position and proceeds in rounds. In each round, in position $(\alpha, \beta)$, Spoiler plays first by choosing one of $\alpha$ and $\beta$, say $\alpha$, and a n-r-transition $\alpha \xrightarrow{a} \alpha'$ from the chosen process. The Duplicator's response is by choosing a (necessarily norm-reducing) transition $\beta \xrightarrow{a} \beta'$ from the other process, with the same label $a$. Duplicator is obliged to choose $\beta'$ with $(\alpha', \beta') \in \equiv \cap \exp(\equiv)$. Then the next round of the play continues from $(\alpha', \beta')$.

If one of players is unable to choose a move, the other player wins the game. This will surely happen, at latest when both processes are finally empty. It is well known that $\alpha$ and $\beta$ are related by $\sim_{\text{n-r}}^{\equiv \cap \exp(\equiv)}$ if and only if Duplicator has a winning strategy in the game starting from $(\alpha, \beta)$. In that case a memoryless winning strategy always exists; it is represented by a bisimulation relation containing $(\alpha, \beta)$.

We will prove that $\sim_{\text{n-r}}^{\equiv \cap \exp(\equiv)}$ is a congruence. Similarly like for the bisimulation equivalence one shows that it is an equivalence; it remains to demonstrate compositionality, i.e., assumed that $(\alpha, \alpha')$ and $(\beta, \beta')$ are in $\sim_{\text{n-r}}^{\equiv \cap \exp(\equiv)}$, we must show that $(\alpha\beta, \alpha'\beta') \in \sim_{\text{n-r}}^{\equiv \cap \exp(\equiv)}$ as well. We will follow the standard lines: assumed two winning strategies $S_A, S_B$ for Duplicator in the games started in $(\alpha, \alpha')$ and $(\beta, \beta')$, respectively, we will show how to combine the strategies into one winning strategy in the game starting from $(\alpha\beta, \alpha'\beta')$.

As $\Delta$ is disjoint we know that Duplicator always responds in the same thread whenever the Spoiler's move is in a non-singleton thread – this simple observation will be crucial for combining the two strategies. Moreover, if Spoiler plays in a singleton thread, the Duplicator's response is in a (possibly different) singleton thread as well.

For simplicity, assume now that the alphabets of threads are pair-wise disjoint. In the sequel it will be apparent that non-disjoint alphabets of singleton threads pose no additional difficulties in the proof.

Now we consider a game starting in $(\alpha\beta, \alpha'\beta')$. We will show existence of a Duplicator's winning strategy. For convenience, we will use an intuitive

'coloring' argument. Assume that in the course of the play all variables are colored either color A or color B. The intuition is that all variables derived from variables in $\alpha, \alpha'$ will be colored A while those derived from variables in $\beta, \beta'$ will be colored B.

At each position, the Duplicator's strategy will exploit either strategy $S_A$ or $S_B$. To be sure that it is always doable, we will show that throughout the play the following invariants are preserved at every position $(\gamma, \gamma')$ of the play (by $\gamma \restriction_A$ we mean $\gamma$ after removing all variable occurrences that are not colored $A$, and similarly $\gamma \restriction_B$):

(1) $(\gamma \restriction_A, \gamma' \restriction_A)$ is winning in $S_A$ and $(\gamma \restriction_B, \gamma' \restriction_B)$ is winning in $S_B$,

(2) $\gamma \equiv \gamma'$,

(3) $(\gamma, \gamma') \in \exp(\equiv)$.

At every position $(\gamma, \gamma')$ of the play, each thread $\gamma_t$ or ${\gamma'}_t$ is of the following form:

$$\gamma_t = \alpha_1^t \beta_1^t \dots \alpha_{n(t)}^t \beta_{n(t)}^t \qquad {\gamma'}_t = \alpha_1'^t \beta_1'^t \dots \alpha_{n'(t)}'^t \beta_{n'(t)}'^t \tag{13}$$

where all $\alpha$ segments are colored A and $\beta$ segments are colored B, and the first and the last segment may be empty but all others are nonempty. Thus instead of invariant (2) we prefer to show the following one, clearly implying (2):

(2') for each thread $t$, $n(t) = n'(t)$ and $\alpha_i^t \equiv \alpha_i'^t$, $\beta_i^t \equiv \beta_i'^t$ for all $i \leq n(t)$.

*Invariants (1) and (2').* The initial position of the game is $(\alpha\beta, \alpha'\beta')$. Color $\alpha$, $\alpha'$ with color $A$, and $\beta$, $\beta'$ with color $B$. The two invariants clearly hold.

Assume the play is at a position $(\gamma, \gamma')$ such that the invariants (1) and (2') hold. We will show that Duplicator can respond to any move of Spoiler in such a way that the invariants are preserved.

Say Spoiler chooses a norm-reducing transition of a variable $X$ and assume wlog that $X$ is colored $A$. Then Spoiler can also choose this move in the $(\alpha, \alpha')$-game at position $(\gamma \restriction_A, \gamma' \restriction_A)$. By (1) Duplicator's has an answer according to $S_A$. Note that this move is necessarily in the same thread. By (2'), and since $\equiv$ may not relate the empty process with a nonempty one, Duplicator can perform this transition at the current position of the combined game. Color all new variables $A$.

It remains to prove that the invariant holds at the new position $(\bar\gamma, \bar\gamma')$ of the game. (1) is clearly satisfied by the choice of the moves. To prove (2') consider any thread $t$ that changed during the current round of the game and the partition of $\bar\gamma_t$ and $\bar\gamma'_t$ into segments, cf. (13). Due to (1) we know that

$$\bar\gamma \restriction_A \equiv \bar\gamma' \restriction_A .$$

By Lemma 5 $\equiv$ is thread-wise, hence we obtain:

$$\alpha_1^t \dots \alpha_{n(t)}^t = (\bar\gamma \restriction_A)_t \equiv (\bar\gamma' \restriction_A)_t = \alpha_1'^t \dots \alpha_{n'(t)}'^t.$$

At most the first segments $\alpha_1^t$ and $\alpha_1'^t$ have possibly changed during the current round and thus by assumption (2') applied to the previous position we know

$$\alpha_2^t \dots \alpha_{n(t)}^t \equiv \alpha_2'^t \dots \alpha_{n'(t)}'^t.$$

As $\equiv$ is right-cancellative by Prop. 2, we deduce $\alpha_1^t \equiv \alpha_1'^t$ and thus (2') holds.

*Invariant (3).* Assume (1) and (2') hold at a position $(\gamma, \gamma')$. To show that (3) holds consider a (not necessarily norm-reducing) transition of a variable $X$, and assume wlog that $X$ is colored $A$. By invariant (1) there is a corresponding position $(\gamma \restriction_A, \gamma' \restriction_A) \in S_A$, and thus $(\gamma \restriction_A, \gamma' \restriction_A) \in \exp(\equiv)$. Then there is a response of Duplicator such that, if we denote by $\bar{\gamma}$ and $\bar{\gamma}'$ the result of executing these two transitions from $\gamma$ and $\gamma'$, respectively, it holds $\bar{\gamma} \restriction_A \equiv \bar{\gamma}' \restriction_A$. We need to show $\bar{\gamma} \equiv \bar{\gamma}'$. But we can use right-cancellativity exactly as above to obtain $\bar{\gamma}_t \equiv \bar{\gamma}'_t$, for every thread $t$, and hence $\bar{\gamma} \equiv \bar{\gamma}'$ as required.

*Singleton threads with non-disjoint alphabets.* For convenience, in the proof we have assumed that each two threads have disjoint alphabets, even if the alphabets of singleton threads need not to be disjoint. However, a careful examination of the proof reveals that the same pattern of proof of invariants (1), (2') and (3) applies to the case of unrestricted disjoint BPC definition: whenever Spoiler plays in a singleton thread, Duplicator can always match using either strategy $S_A$ or $S_B$, form a (possibly different) singleton thread. Intuitively, in case of singleton threads, coloring of a particular occurrence of a (unique) variable is irrelevant. We omit the details.

The proof of Thm 5 is now completed. □

## 8. 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. As one of contributions, we have introduced a framework of partially-commutative context-free processes, called BPC, that seems convenient for jointly considering sequential and parallel processes.

The algorithm works correctly in a *feasible* fragment of normed BPC. A condition sufficient for feasibility, subsuming both BPA and BPP, has been provided in the paper.

Concerning expressibility, normed BPC and normed PA seem to be incomparable. We have shown that even with respect to the trace equivalence BPC is not included in PA, even if we additionally restrict to only transitive dependency relations `D` (the fragment called tBPC in the paper). The detailed comparison of expressive power of BPC with respect to other approaches, including PA and trace context-free languages, is planned to be investigated in a separate paper. Moreover, it should be checked if the feasible fragment of tBPC is not included in PA.

An interesting open question remains whether the decision procedure for the bisimulation equivalence 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.

A further natural question is whether the bisimulation equivalence in *unnormed* BPA and BPP classes may be solved by a common algorithm. Similarly as in normed case, the existing procedures seem to be quite different.

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

*Acknowledgements.* We are grateful to the anonymous referees for valuable comments that allowed us to improve this paper.

## References

[1] J.A. Bergstra and J.W. Klop. Algebra of communicating processes with abstraction. *Theoret. Comput. Sci.*, 37:77–121, 1985.

[2] O. Burkart, D. Caucal, F Moller, and B. Steffen. Verification of infinite structures. In *Handbook of Process Algebra*, pages 545–623. Elevier, 2001.

[3] S. Christensen. *Decidability and Decomposition in process algebras.* PhD thesis, Dept. of Computer Science, University of Edinburgh, UK, 1993.

[4] S. Christensen, H. Hüttel, and C. Stirling. Bisimulation equivalence is decidable for all context-free processes. *Inf. Comput.*, 121(2):143–148, 1995.

[5] W. Czerwiński, S. Fröschle, and S. Lasota. Partially-commutative context-free processes. In *Proc. CONCUR'09*, volume 5710 of *LNCS*, pages 259–273. Springer-Verlag, 2009.

[6] V. Diekert and G. Rozenberg. *The book of traces.* World Scientific, 1995.

[7] S. Fröschle and S. Lasota. Normed processes, unique decomposition, and complexity of bisimulation equivalences. In *Proc. INFINITY'06*, volume 239 of *ENTCS*, pages 17–42. Elsevier, 2009.

[8] Y. Hirshfeld and M. Jerrum. Bisimulation equivalence is decidable for normed process algebra. In *ICALP'99*, volume 1644 of *LNCS*, pages 412–421, 1999.

[9] 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.

[10] 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.

[11] P. Jančar. Bisimilarity of Basic Parallel Processes is PSPACE-complete. In *Proc. LICS'03*, pages 218–227, 2003.

[12] 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.

[13] 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.

[14] 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.

[15] R. Mayr. Process rewrite systems. *Inf. Comput.*, 156(1-2):264–286, 2000.