Rewrite Systems with Constraints ⋆

Jan Strejček

Faculty of Informatics MU,
Botanická 68a, 60200 Brno, Czech Republic
`xstrejc@fi.muni.cz`

Abstract. We extend a widely used concept of rewrite systems with a unit holding a kind of global information which can influence and can be influenced by rewriting. The unit is similar to the *store* used in concurrent constraint programming, and can be also seen as a special (weak) state unit. We present how this extension changes the expressive power of rewrite systems classes which are included in Mayr's PRS hierarchy [8]. The new classes (fcBPA, fcBPP, fcPA, fcPAD, fcPAN, fcPRS) are described and inserted into the hierarchy.

1 Introduction

The cornerstone of concurrency theory is the notion of labelled transition system. Caucal [4] presents a classification of transition systems using families of sequential rewrite systems related to the Chomsky hierarchy. Caucal's classification has been generalised by Moller [11] to both parallel and sequential rewrite systems. Moller's approach was further generalised by Mayr [8], who defines the dynamics for rewrite systems using sequential and parallel composition together. The resulting model is called *process rewrite systems (PRS)*.

Concurrent constraint programming (CCP) [14] is one of the most successful applications of the ideas of concurrency and computing with partial information. In CCP processes work concurrently with a shared *store*, which is seen as a constraint on the values that variables can represent. In any state of the computation, the store is given by the constraint established until that moment. CCP provides two operations to deal with the store, *tell* and *ask*. The *tell* monotonically updates the store by adding a constraint (provided the store remains *consistent*). The *ask* is a test on the store – it can be executed only if the current store is strong enough to *entail* a specified constraint. If this is not the case, then the process suspends (waiting for the store to accumulate more information by contributions of the other processes).

We transfer some principles of CCP to process rewrite systems. Previously, we have introduced an analogous modification of purely sequential and purely parallel rewrite systems in [15]. In both cases, the aim is to characterise the changes of expressive power of these systems. The mechanism of PRS is extended with the store, which can contain some global (monotonically evolving)

⋆ This work has been partially supported by the Grant Agency of Czech Republic, grant No. 201/00/0400.

information. We add two constraints to every rewrite rule. A rule can be applied only if the actual store is strong enough to entail the first constraint; the second constraint is added to the store when the extended rule is used (the rule is applicable if the store is kept consistent). Extended process rewrite systems are called *process rewrite systems with finite constraint systems (fcPRS)*.[1]

We obtain some interesting results by studying which labelled transition systems (up to bisimulation) can be denoted by specific classes of PRS systems (accordant with well-known formalisms like finite state systems, basic process algebra (BPA), basic parallel processes (BPP), process algebra (PA), pushdown processes, Petri nets, etc.) and by corresponding fcPRS classes. The expressive power of finite state systems, pushdown processes, and Petri nets keeps unchanged by adding the store. This does not hold in the case of BPA, BPP, PA, PAD, and PAN class when the expressive power strictly increases, thus some new classes are obtained in this way. Hence this new framework can be used to solve some interesting open problems, e.g. to examine the decidability border within the process hierarchies already maintained (in case of bisimilarity it is known that the border line goes between BPP and its "state-extended" version MSA continuing between (normed) PA and its "state-extended" version etc.); our new process classes are situated in this grey area.

2 Basic definitions

In this section we recall the notions of labelled transitions systems, language generated by such system, and bisimulation equivalence.

Definition 1. *A* labelled transition system (LTS) $\mathcal{L}$ *is a tuple* $(S, Act, \longrightarrow, \alpha_0)$*, where* S *is a set of* states *or* processes*,* Act *is a set of* atomic actions *or* labels*,* $\longrightarrow \subseteq S \times Act \times S$ *is a* transition relation *(written* $\alpha \xrightarrow{a} \beta$ *instead of* $(\alpha, a, \beta) \in \longrightarrow$*),* $\alpha_0 \in S$ *is a distinguished* initial state*. A state* $\alpha \in S$ *is* terminal *(or* deadlocked*, written* $\alpha \not\longrightarrow$*) if there is no* $a \in Act$ *and* $\beta \in S$ *such that* $\alpha \xrightarrow{a} \beta$*.*

The transition relation $\longrightarrow$ can be homomorphically extended to finite sequences of actions $\sigma \in Act^*$ so as to write $\alpha \xrightarrow{\varepsilon} \alpha$ and $\alpha \xrightarrow{a\sigma} \beta$ whenever $\alpha \xrightarrow{a} \gamma \xrightarrow{\sigma} \beta$ for some state γ. The set of states α such that $\alpha_0 \xrightarrow{\sigma} \alpha$ for the initial state α_0 and some $\sigma \in Act^*$ is called the set of *reachable* states.

Definition 2. *The* language *generated by the labelled transition system* $\mathcal{L}$ *is the set* $L(\mathcal{L}) = L(\alpha_0)$*, where* $L(\alpha) = \{w \in Act^* \mid \exists \beta : \alpha \xrightarrow{w} \beta \not\longrightarrow\}$*. States* α *and* β *of the system* $\mathcal{L}$ *are* language equivalent*, written* $\alpha \sim_L \beta$*, iff they generate the same language, i.e.* $L(\alpha) = L(\beta)$*.*

[1] Note that rules of fcPRS systems can be also seen as a new special "format" of SOS rules (in the sense of [6]) with side conditions referring to a global (monotonic) store. However this viewpoint is not examined in this paper.

Language equivalence is generally taken to be too coarse in the framework of concurrency theory. The second presented equivalence, *bisimulation equivalence*, is perhaps the finest behavioural equivalence studied. Bisimulation equivalence was defined by Park [13] and used by Milner [9, 10] in his work on CCS.

Definition 3. *A binary relation $\mathcal{R}$ on states of labelled transition system is a* bisimulation *iff whenever $(\alpha, \beta) \in \mathcal{R}$ we have that*

- *if $\alpha \xrightarrow{a} \alpha'$ then $\beta \xrightarrow{a} \beta'$ for some β' with $(\alpha', \beta') \in \mathcal{R}$,*
- *if $\beta \xrightarrow{a} \beta'$ then $\alpha \xrightarrow{a} \alpha'$ for some α' with $(\alpha', \beta') \in \mathcal{R}$.*

α and β are bisimulation equivalent *or* bisimilar*, written $\alpha \sim \beta$, iff $(\alpha, \beta) \in \mathcal{R}$ for some bisimulation $\mathcal{R}$.*

3 Process rewrite systems (PRS)

This section summarise the first part of Mayr's paper titled "Process Rewrite Systems" [8].

The *process rewrite systems (PRS)* developed by Mayr represent a very general term rewriting formalism offering a way for finite description of possibly infinite transition systems. The formalism covers many widely known models like finite-state processes (FS), basic parallel processes (BPP), context-free processes (BPA), pushdown processes (PDA), process algebras (PA), Petri nets (PN), and provides a unified view of these models. The definition of PRS is more general than the definitions of rewrite system given by Caucal [4] (only with sequential composition) and by Moller [11] (only purely sequential and purely parallel rewrite systems).

Let $Const = \{X, Y, Z, \cdots\}$ be a countably infinite set of *process constants.* The set $\mathcal{T}$ of *process terms* is defined by the abstract syntax

$$t = \varepsilon \mid X \mid t_1.t_2 \mid t_1 \| t_2,$$

where ε is the empty term, $X \in Const$ is a process constant (used as an atomic process), "$\|$" means parallel and "." means sequential compositions respectively.

We always work with equivalence classes of terms modulo commutativity and associativity of parallel composition and modulo associativity of sequential composition. Also we define $\varepsilon.t = t = t.\varepsilon$ and $t \| \varepsilon = t$.

The set $Const(t)$ is the set of all constants occurring in a process term t.

We distinguish four *classes of process terms.*

"1" Terms consisting of a single process constant like X.
"S" *Sequential* terms - without parallel composition. For example $X.Y.Z$.
"P" *Parallel* terms - without sequential composition. For example $X\|Y\|Z$.
"G" *General* terms with arbitrarily nested sequential and parallel compositions like $(X.(Y\|Z))\|W$.

We also let $\varepsilon \in S, P, G$, but $\varepsilon \notin 1$.

Definition 4. *Let* $Act = \{a, b, \cdots\}$ *be a countably infinite set of* atomic actions*,* $\alpha, \beta \in \{1, S, P, G\}$ *such that* $\alpha \subseteq \beta$*. An* (α, β)-PRS (process rewrite system) Δ *is a pair* (R, t_0)*, where*

- R *is a finite set of* rewrite rules *of the form* $t_1 \xrightarrow{a} t_2$*, where* $t_1 \in \alpha$*,* $t_1 \neq \varepsilon$*,* $t_2 \in \beta$ *are process terms and* $a \in Act$ *is an atomic action,*
- $t_0 \in \beta$ *is an* initial state.

A (G, G)*-PRS is simply called PRS.*

We write $(t_1 \xrightarrow{a} t_2) \in \Delta$ instead of $(t_1 \xrightarrow{a} t_2) \in R$, where $\Delta = (R, t_0)$.

For a given Δ we define $Const(\Delta)$ as the set of all constants that occur in rewrite rules or initial state, and $Act(\Delta)$ as the set of all actions that occur in rewrite rules of Δ. The sets $Const(\Delta)$ and $Act(\Delta)$ are both finite.

Each process rewrite system denotes a labelled transition system (LTS) that represents its dynamics. Let $\Delta = (R, t_0)$ be an (α, β)-PRS. The LTS $\mathcal{L}$ denoted by Δ is a tuple $(S, Act(\Delta), \longrightarrow, t_0)$, where $S = \{t \in \beta \mid Const(t) \subseteq Const(\Delta)\}$ is the set of states, t_0 is the initial state and transition relation $\longrightarrow$ is the least relation that satisfies the inference rules[2]

$$\frac{(t_1 \xrightarrow{a} t_2) \in \Delta}{t_1 \xrightarrow{a} t_2}, \qquad \frac{t_1 \xrightarrow{a} t_1'}{t_1 \| t_2 \xrightarrow{a} t_1' \| t_2}, \qquad \frac{t_1 \xrightarrow{a} t_1'}{t_1.t_2 \xrightarrow{a} t_1'.t_2},$$

where $t_1, t_2, t_1' \in \mathcal{T}$.

We speak about "process rewrite system" meaning "labelled transition system generated by process rewrite system".

Obviously, it can be assumed (w.l.o.g.) the initial state t_0 of a (α, β)-PRS is a single constant as there are only finitely many terms t_i such that $t_0 \xrightarrow{a_i} t_i$.

Figure 1 shows a graphical description of the hierarchy of (α, β)-PRS, simply called *PRS-hierarchy.* Some classes included in the hierarchy correspond to widely known models:

- $(1, 1)$-PRS are equivalent to finite-state systems (FS). Every process constant corresponds to a state and the state space is bounded by $|Const(\Delta)|$. Every finite-state system can be encoded as a $(1, 1)$-PRS.
- $(1, S)$-PRS are equivalent to Basic Process Algebra processes (BPA) defined in [1], which are the transition systems associated with Greibach normal form (GNF) context-free grammars in which only left-most derivations are allowed.
- $(1, P)$-PRS are equivalent to communication-free nets, the subclass of Petri nets where every transition has exactly one place in its preset [3]. This class of Petri nets is equivalent to Basic Parallel Processes (BPP) [5].
- $(1, G)$-PRS are equivalent to PA-processes, Process Algebras with sequential and parallel composition, but no communication (see [1] for details).

[2] Note that parallel composition is commutative and, thus, the inference rule for parallel composition also holds with t_1 and t_2 exchanged.

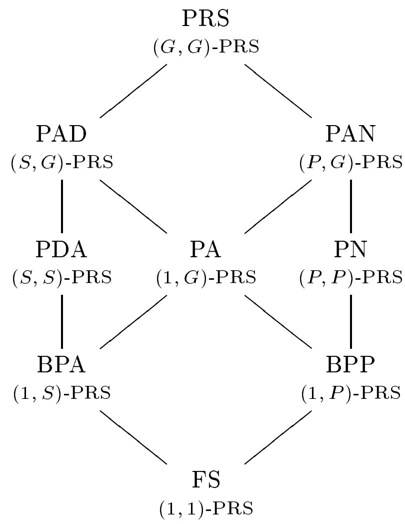


Fig. 1. The PRS-hierarchy

- It is easy to see that pushdown automata can be encoded as a subclass of (S,S)-PRS (with at most two constants on the left-hand side of rules). Caucal [4] showed that any unrestricted (S,S)-PRS can be presented as a pushdown automaton (PDA), in the sense that the transition systems are isomorphic up to the labelling of states. Thus (S,S)-PRS are equivalent to pushdown processes (which are the processes described by pushdown automata).
- (P,P)-PRS are equivalent to Petri nets (PN). Every constant corresponds to a place in the net and the number of occurrences of a constant in a term corresponds to the number of tokens in this place. This is because we work with classes of terms modulo commutativity of parallel composition. Every rule in Δ corresponds to a transition in the net.
- (S,G)-PRS is the smallest common generalisation of pushdown processes and PA-processes. They are called *PAD* (PA + PDA) in [8].
- (P,G)-PRS are called *PAN-processes* in [7]. It is the smallest common generalisation of Petri nets and PA-processes and it strictly subsumes both of them (e.g., PAN can describe all Chomsky-2 languages while Petri nets cannot).
- The most general case is (G,G)-PRS (here simply called PRS). PRS have been introduced in [8]. They subsume all of the previously mentioned classes.

The hierarchy is not strict w.r.t. language equivalence. For example, both BPA and PDA define exactly the (ε-free) context-free languages. The strictness of the hierarchy w.r.t. bisimulation equivalence follows from previous results [2, 11] and the proof, that there is a PDA system (described in Example 1) which is not bisimilar to any PAN system and a Petri net (described in Example 2) which is not bisimilar to any PAD process.

Example 1. Let us consider the following PDA system with initial state $U.X$.

$$\begin{array}{lll}
U.X \xrightarrow{a} U.A.X & U.A \xrightarrow{a} U.A.A & U.B \xrightarrow{a} U.A.B \\
U.X \xrightarrow{b} U.B.X & U.A \xrightarrow{b} U.B.A & U.B \xrightarrow{b} U.B.B \\
U.X \xrightarrow{c} V.X & U.A \xrightarrow{c} V.A & U.B \xrightarrow{c} V.B \\
U.X \xrightarrow{d} W.X & U.A \xrightarrow{d} W.A & U.B \xrightarrow{d} W.B \\
V.X \xrightarrow{e} V & V.A \xrightarrow{a} V & V.B \xrightarrow{b} V \\
W.X \xrightarrow{f} W & W.A \xrightarrow{a} W & W.B \xrightarrow{b} W
\end{array}$$

Example 2. Consider following Petri net given as (P, P)-PRS with initial state $X\|A\|B$.

$$\begin{array}{ll}
X \xrightarrow{g} X\|A\|B & Y\|A \xrightarrow{a} Y \\
X \xrightarrow{c} Y & Y\|B \xrightarrow{b} Y \\
X\|A \xrightarrow{d} Z & Y\|A \xrightarrow{d} Z \\
X\|B \xrightarrow{d} Z & Y\|B \xrightarrow{d} Z
\end{array}$$

4 PRS with finite constraint systems (fcPRS)

In this section we extend the PRS formalism with a unit (called *store*) able to keep a sort of global information which is accessible to all parallel threads of the term. It is quite surprising that this unit (which is not as powerful as a general finite-state control unit which gives Turing power even to the PA class) increases the expressive power of classes like PAN and PAD.

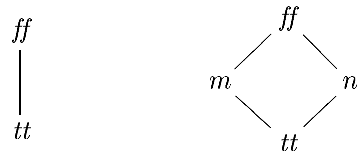
The state space and possible evolution of the store used by PRS with finite constraint system are described by a constraint system, i.e. a set of constraints with a structure of an algebraic lattice.

Definition 5. *A* constraint system *is a bounded lattice* $(C, \vdash, \wedge, tt, ff)$*, where* C *is the set of* constraints*,* $\vdash$ *(called* entailment*) is an ordering on this set,* $\wedge$ *is the lub operation, and* tt *(*true*),* ff *(*false*) are the least and the greatest elements of* C *respectively (*$ff \vdash tt$ *and* $tt \neq ff$*).*

In algebra, the symbol $\wedge$ usually denotes the *glb* operation, while *lub* operation is rather marked with symbol $\vee$. Our notation of lub operation corresponds to logical conjunction (as in CCP).

We say that a constraint m is *consistent* with a constraint n iff $m \wedge n \neq ff$. The state of the store cannot be ff as we require the consistency of the store initialised to tt. We use $C^\diamond$ to denote $C \smallsetminus \{ff\}$.

Example 3. Let $\mathcal{C}_\varepsilon$ be the trivial constraint system $(\{tt, ff\}, \vdash, \wedge, tt, ff)$, where $\vdash = \{(ff, tt), (tt, tt), (ff, ff)\}$, and $\mathcal{C}_{mn}$ the constraint system $\mathcal{C}_{mn} = (\{tt, m, n, ff\}, \vdash, \wedge, tt, ff)$, where $\vdash = \{(ff, tt), (m, tt), (n, tt), (ff, m), (ff, n)\} \cup \{(o, o) \mid o \in C\}$. These constraint systems are depicted below.



Definition 6. *Let $\alpha, \beta \in \{1, S, P, G\}$ such that $\alpha \subseteq \beta$. An (α,β)-*fcPRS (PRS with finite constraint system) *Δ is a tuple $(\mathcal{C}, R, t_0)$, where*

- *$\mathcal{C} = (C, \vdash, \wedge, tt, ff)$ is a finite constraint system describing the* store*; the elements of C represent the* states of the store*,*
- *R is a finite set of* rewrite rules *of the form $(t_1 \xrightarrow{a} t_2, m, n)$, where $t_1 \in \alpha$, $t_1 \neq \varepsilon$, $t_2 \in \beta$ are process terms, $a \in Act$ is an atomic action, and $m, n \in C^\diamond$ are constraints,*
- *$t_0 \in \beta$ is a distinguished* initial process term*.*

A (G,G)-fcPRS is simply called fcPRS.

We use human-readable abbreviations fcFS, fcBPA, fcBPP, fcPA, fcPDA, fcPN, fcPAD, fcPAN, and fcPRS for classes $(1,1)$-fcPRS, $(1,S)$-fcPRS, $(1,P)$-fcPRS, $(1,G)$-fcPRS, (S,S)-fcPRS, (P,P)-fcPRS, (S,G)-fcPRS, (P,G)-fcPRS, and (G,G)-fcPRS respectively.

Again, instead of $(t_1 \xrightarrow{a} t_2, m, n) \in R$ where $\Delta = (\mathcal{C}, R, t_0)$, we usually write $(t_1 \xrightarrow{a} t_2, m, n) \in \Delta$. The meaning of sets $Const(\Delta)$ (process constants used in rewrite rules) and $Act(\Delta)$ (actions occurring in rewrite rules) for a given fcPRS Δ is the same as in PRS case. Again, it can be assumed the initial term t_0 of an (α,β)-fcPRS is a single constant.

Every PRS with finite constraint system denotes a labelled transition system. Let $\Delta = (\mathcal{C}, R, t_0)$ be an (α,β)-fcPRS. The LTS $\mathcal{L}$ denoted by Δ has the form $(S, Act(\Delta), \longrightarrow, (t_0, tt))$, where $S = \{t \in \beta \mid Const(t) \subseteq Const(\Delta)\} \times C^\diamond$ is the set of states, (t_0, tt) is the initial state and transition relation $\longrightarrow$ is defined as the least relation that satisfies the inference rules

$$\frac{(t_1 \xrightarrow{a} t_2, m, n) \in \Delta}{(t_1, o) \xrightarrow{a} (t_2, o \wedge n)} \quad \text{if } o \vdash m \text{ and } o \wedge n \neq ff,$$

$$\frac{(t_1, o) \xrightarrow{a} (t_1', p)}{(t_1 \| t_2, o) \xrightarrow{a} (t_1' \| t_2, p)}, \qquad \frac{(t_1, o) \xrightarrow{a} (t_1', p)}{(t_1.t_2, o) \xrightarrow{a} (t_1'.t_2, p)},$$

where $t_1, t_2, t_1' \in \mathcal{T}$ and $m, n, o, p \in C^\diamond$.

The two side conditions in the first inference rule are very close to principles used in CCP. The first one ($o \vdash m$) ensures the rule $(t_1 \xrightarrow{a} t_2, m, n) \in \Delta$ can be used only if the current state of the store o *entails* m (it is similar to *ask(m)* in CCP). The second condition ($o \wedge n \neq ff$) guarantees that the store stays *consistent* after application of the rule (analogous to the consistency requirement when processing *tell(n)* in CCP).

An important observation is that the state of the store (starting at tt) can move in a lattice $\mathcal{C}$ only in one direction, from tt upwards. This can be easily seen from the fact that the actual state of the store o can be changed only by applying some rewrite rule $(t_1 \xrightarrow{a} t_2, m, n) \in \Delta$ and after this application the new state of the store $o \wedge n$ always entails o. Intuitively, the partial information can only be added to the store, not retracted. We say the store is *monotonic*.

Note that when the system (with o on the store) executes a transition generated by a rule $(t_1 \xrightarrow{a} t_2, m, n) \in \Delta$ then for every subsequent state of the store p conditions $p \vdash m$ and $p \wedge n \neq \mathit{ff}$ are satisfied. The first condition $p \vdash m$ comes from the monotonic behaviour of the store. The second condition comes from the facts that the constraint n in the rule can change the store only in the first application of the rule and that for each subsequent state p of the store $p \wedge n = p$ holds.

On the other hand, the fact that some rule is applicable (hence entailment and consistency are satisfiable) does not imply that this rule is applicable forever. The insidious point is the consistency requirement. The store can evolve to a state inconsistent with the second constraint from the rule.

The first information about the relationship between fcPRS and PRS is provided by the following lemma.

Lemma 1. *Let $\alpha, \beta \in \{1, P, S, G\}$. The systems (α, β)-PRS $\Delta' = (R', t_0)$ and (α, β)-fcPRS $\Delta = (\mathcal{C}_\varepsilon, R, t_0)$ are isomorphic on the assumption that $R' = \{t_1 \xrightarrow{a} t_2 \mid (t_1 \xrightarrow{a} t_2, \mathit{tt}, \mathit{tt}) \in R\}$.*

Proof. It is easy to check that if we remove tt from the states of LTS generated by fcPRS Δ, we get an isomorphic system which corresponds to the PRS Δ'. ■

The lemma above says that PRS classes can be seen as fcPRS classes with a trivial constraint system. The lemma can be used in both directions, to show that any fcPRS of the specified form has an equivalent PRS as well as for constructing an fcPRS equivalent to a given PRS.

5 The fcPRS-hierarchy

Figure 2 shows the hierarchy of PRS and fcPRS classes, simply called *fcPRS-hierarchy*. The relations depicted in the hierarchy partly result from the definition of classes and Lemma 1. The rest of the paper is dedicated to three equalities (fcFS = FS, fcPDA = PDA, and fcPN = PN) and the strictness of the hierarchy.

Theorem 1. *(i) Let Δ be an fcFS. There exists FS Δ' denoting a labelled transition system isomorphic to the one given by Δ.*

(ii) Let Δ be an fcPDA. There exists PDA Δ' denoting a labelled transition system isomorphic to the one given by Δ.

(iii) Let Δ be an fcPN. There exists PN Δ' denoting a labelled transition system isomorphic to the one given by Δ.

Proof. (i) The construction is obvious, every state (X, m) of Δ is transformed into state $X^{(m)}$ of Δ'.

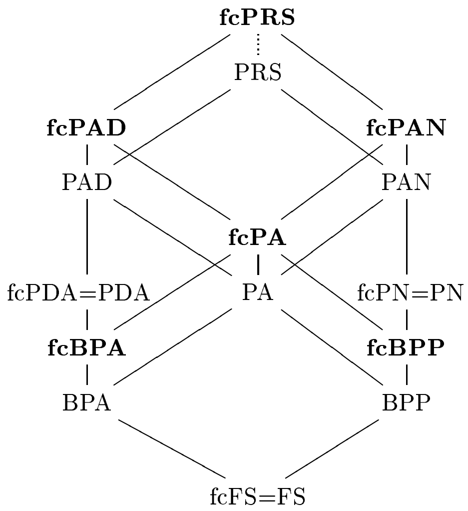


Fig. 2. The fcPRS-hierarchy

(ii) The idea of the proof is based on the fact that we can add special process constants corresponding to the actual states of the store, one to each state of fcPDA. Then the content of the store will be represented by such special constants.

Let $\Delta = (\mathcal{C}, R, t_0)$, where $\mathcal{C} = (C, \vdash, \wedge, tt, ff)$. Let $S = \{S^{(m)} \mid m \in C^\diamond\}$ be the set of special process constants. A PDA Δ' is constructed as $(R', S^{(tt)}.t_0)$, where $S^{(tt)}.t_0$ is the initial term with the special constant holding the initial state of the store. We replace every rule

$$(t_1 \xrightarrow{a} t_2, m, n) \in R$$

by the set of rules

$$(S^{(o)}.t_1 \xrightarrow{a} S^{(o \wedge n)}.t_2) \in R'$$

for every $o \in C^\diamond$ which satisfies the entailment condition $o \vdash m$ and the consistency condition $o \wedge n \neq ff$. The new rules are constructed to abide by the entailment and consistency conditions connected with the original rules. The isomorphism of Δ and Δ' is obvious as every state $S^{(m)}.t$ of Δ' corresponds exactly to the state (t, m) of the system Δ.

(iii) The proof is the same as for (ii) if we replace every sequential composition by the parallel composition. ∎

As the PRS-hierarchy is not strict w.r.t. the language equivalence, the fcPRS-hierarchy cannot also be strict on the language expressibility level. However, the fcPRS-hierarchy is strict w.r.t. the bisimulation equivalence with possibly one exception: the relation between PRS and fcPRS (this case will be discussed later). To prove that each of the classes fcBPA, fcBPP, fcPA, fcPAD, and fcPAN differs from the corresponding standard class, we present two fcPRS systems. The first one is an fcBPA system which is not bisimilar to any PAN system. The second system will be an fcBPP which is not bisimilar to any PAD system.

Example 4. Let us consider an fcBPA system with the constraint system $\mathcal{C}_{mn}$ from Example 3 and the initial process term $U.X$.

$$
\begin{array}{ll}
(U \xrightarrow{a} U.A, tt, tt) & (A \xrightarrow{a} \varepsilon, tt, tt) \\
(U \xrightarrow{b} U.B, tt, tt) & (B \xrightarrow{b} \varepsilon, tt, tt) \\
(U \xrightarrow{c} \varepsilon, tt, m) & (X \xrightarrow{e} \varepsilon, m, tt) \\
(U \xrightarrow{d} \varepsilon, tt, n) & (X \xrightarrow{f} \varepsilon, n, tt)
\end{array}
$$

The fcBPA above is bisimilar to the PDA system described in 1 which is not bisimilar to any PAN system and thus also the considered fcBPA process is not bisimilar to any PAN system. Hence we obtain a following corollary, where $X \subsetneq Y$ means that X is a strict subclass of Y and $X \not\subseteq Y$ means that X is not a subclass of Y.

Corollary 1. *BPA $\subsetneq$ fcBPA, PA $\subsetneq$ fcPA, PAN $\subsetneq$ fcPAN and fcBPA $\not\subseteq$ PA, fcBPA $\not\subseteq$ PAN, fcPA $\not\subseteq$ PAN.*

Proof. Directly from the definition of BPA and PAN classes and from it follows that the BPA class is a subclass of the PAN class. Lemma 1 implies that the BPA class is a subclass of the fcBPA class. We know that there exists an fcBPA system which is not bisimilar to any PAN system and thus also to any BPA system. Hence we know that BPA is strict subclass of fcBPA. The proofs of the other relations are similar. ∎

Example 5. Let us consider an fcBPP system with the constraint system depicted below and the initial state (X, tt).

$$
\begin{array}{c}
ff \\
| \\
o \\
| \\
tt
\end{array}
\qquad
\begin{array}{l}
(X \xrightarrow{a} X \| A, tt, tt) \\
(X \xrightarrow{b} X \| B, tt, tt) \\
(X \xrightarrow{e} \varepsilon, tt, o) \\
(A \xrightarrow{c} \varepsilon, o, tt) \\
(B \xrightarrow{d} \varepsilon, o, tt)
\end{array}
$$

Lemma 2. *If there is a PAD system bisimilar to the fcBPP system from Example 5, then there is also a PDA system bisimilar to this fcBPP.*

Proof. Let Δ be a PAD with the initial state Q (w.l.o.g.) such that Q is bisimilar to the initial state (X, tt) of considered fcBPP system. As on the left-hand side of rewrite rules Δ only sequential composition can occur, some part of parallel composition $t_1 \| t_2$ can influence the behaviour of such system only if there is a reachable state of the form $(t_1 \| t_2).t_3$ where t_3 can be ε. If there is no such a state, we can remove all parallel compositions from the rules and we get a PDA system bisimilar to Δ and thus also bisimilar to the considered fcBPP process.

Another situation arises if there is a reachable state of Δ of the form $(t_1\|t_2).t_3$, where t_3 can be ε. Let us assume that during the derivation of the state $(t_1\|t_2).t_3$ from Q there is no other state of the form $(t'_1\|t'_2).t'_3$ (t_3 can be ε). As Q is a single process constant and any parallel composition $s_1\|s_2$ in a term $p.(s_1\|s_2).p'$ cannot be changed by any rewriting until p is ε, there must be some rewrite rule $(t \xrightarrow{x} l.(t_1\|t_2).r) \in \Delta$ (l, r can be ε, $x \in \{a, b, c, d, e\}$) such that $t_1\|t_2$ is the mentioned parallel composition. There are two cases.

1. The state $(t_1\|t_2).t_3$ was derived from Q under a word $w \in \{a, b\}^*$. We show that t_1 or t_2 is then deadlocked. With respect to the definition of PAD, which does not provide any form of communication or synchronisation between processes in a parallel composition, just one component of $t_1\|t_2$ can enable the action e, let us assume that it is t_2. Then t_1 is deadlocked – it cannot do neither the actions a or b (as these actions are disabled after the action e) nor the actions c or d (as these actions are disabled before e). Nevertheless, the term $t_1.t'$ is not necessarily deadlocked for some term t'. Hence, the parallel composition $t_1\|t_2$ in the rule $(t \xrightarrow{x} l.(t_1\|t_2).r) \in \Delta$ can be changed to the sequential composition $t_2.t_1$. We should insert some separator between t_2 and t_1 (resp. l and t_2) to keep the impossibility of communication between parts of parallel composition (resp. between l and part of the following parallel composition). Thus we replace the rule $(t \xrightarrow{x} l.(t_1\|t_2).r) \in \Delta$ by the rule $t \xrightarrow{x} l.X.t_2.X.t_1.r$ (resp. $t \xrightarrow{x} t_2.X.t_1.r$ if $l = \varepsilon$), where $X \notin Const(\Delta)$ is a new constant, and we add new rewrite rule $X.s \xrightarrow{x} s'$ to Δ for every rewrite rule $s \xrightarrow{x} s' \in \Delta$ (if we already have the rules of the form $X.s \xrightarrow{x} s'$ in modified Δ, we do not need to add them again in the future). These changes do not affect the behaviour of Δ.
2. The action e occurs during the derivation of the state $(t_1\|t_2).t_3$ from Q. The state $(t_1\|t_2).t_3$ is then bisimilar to a state $(A^n\|B^m, o)$[3] of considered fcBPP and thus every possible sequence of actions performed by the process $(t_1\|t_2).t_3$ is finite, as well as every possible sequence performed by the term $t_1\|t_2$. We construct a finite labelled (acyclic) transition graph where the vertices are processes reachable from the parallel composition $t_1\|t_2$ (which is the root of the graph) and edges naturally correspond to actions (resp. applications of rewrite rules). Now we assign a fresh process constant to each vertex of the graph which has some parallel composition inside (the vertices without any parallel composition keep unchanged). We replace the rule $(t \xrightarrow{x} l.(t_1\|t_2).r) \in \Delta$ by the rule $t \xrightarrow{x} l.Z.r$, where $Z \notin Const(\Delta)$ is a process constant assigned to $t_1\|t_2$. For every edge of the graph from the vertex A (where A is a fresh constant) to the vertex v we add a rule $A \xrightarrow{x} v$ (where x is the label of the edge) to Δ. The behaviour of Δ is still unchanged thanks to the fact that if $(t_1\|t_2).t_3 \xrightarrow{*} t'.t_3$ then the term t_3 can be changed

[3] The expression A^n is an abbreviation for n copies of process constant A in parallel composition. The abbreviation B^m has an analogous meaning.

by the following transition only if there is no parallel composition in t', and the fact that the vertices without any parallel composition are unchanged.

In both cases, the number of parallel compositions in rewrite rules has decreased (with one exception – when we add rules of the form $X.s \xrightarrow{x} s'$, then the number of parallel compositions can be doubled, but it does not matter as we make it only once). If there is still a reachable state of the form $(t_1\|t_2).t_3$ in modified Δ, we can use the same method again. As the number of parallel compositions in rewrite rules is finite, after finite number of steps we get a PAD system without any reachable state of the form $(t_1\|t_2).t_3$, which is the situation discussed at the beginning of this proof. ■

The class of context-free languages (i.e. the class of languages generated by PDA processes) is closed under intersection with regular languages. The language L generated by the fcBPP system from Example 5 is not context-free, as $L \cap a^*b^*ec^*d^* = \{a^nb^mec^nd^m \mid m,n \geq 0\}$ which is not context-free. Thus there is no PDA process bisimilar to fcBPP from Example 5 and from Lemma 2 it follows that there is no PAD process bisimilar to the fcBPP presented above. Hence we get:

Corollary 2. *BPP $\subsetneq$ fcBPP, PAD $\subsetneq$ fcPAD and fcBPP $\not\subseteq$ PA, fcPA $\not\subseteq$ PAD, fcBPP $\not\subseteq$ PAD.*

The fcBPP class differs from PN even w.r.t. language equivalence. The language $L = \{a^nbc^nde^nf \mid n \geq 0\}$ generated by PN from Example 6 is an instance of a language generated by PN, which cannot be described by any fcBPP due to the following Pumping Lemma.

Example 6. Let $\Delta = (R, W)$ be a Petri net with rewrite rules as below.

$$\begin{array}{ll} W \xrightarrow{a} W\|A\|B & Y\|B \xrightarrow{e} Y \\ W \xrightarrow{b} X & Y \xrightarrow{f} Z \\ X\|A \xrightarrow{c} X & Z\|A \xrightarrow{z} Z\|A \\ X \xrightarrow{d} Y & Z\|B \xrightarrow{z} Z\|B \end{array}$$

Lemma 3 (Pumping Lemma for fcBPP). *Let L be a language of an fcBPP system Δ. There exists a constant h such that if $u \in L$ and $|u| > h$ then there exist $x, y, z, w \in Act^*$ such that $u = xz$, $|y| > 1$, and $\forall i \geq 0$ it holds that $xy^izw^i \in L$.*[4]

Proof. The proof can be found in Appendix A. ■

To prove the strictness of the fcPRS-hierarchy completely we introduce a PDA process which is not bisimilar to any fcPAN process and a PAN process which is not bisimilar to any fcPAD process.

[4] $|u|$ denotes the length of the word u.

Example 7. Let us consider a PDA system described in Example 1 with initial state $U.X.Y$ and with the following additional rewrite rules.

$$\begin{array}{ll} V.Y \xrightarrow{x} U.X.Y & W.Y \xrightarrow{x} U.X.Y \\ V.Y \xrightarrow{z} Z & W.Y \xrightarrow{z} Z \end{array}$$

This system behaves like that defined in Example 1, but when the original system terminates, the enhanced system can choose between termination under the action z and restart under the action x.

Lemma 4. *There is no fcPAN system bisimilar to the PDA process given in Example 7.*

Proof. We assume the contrary and derive a contradiction. Let Δ be an fcPAN bisimilar to the PDA process defined in Example 7. From the finiteness of the constraint system used in Δ follows that there exists a non-terminal reachable state (t, o) of Δ such that every non-terminal state reachable from (t, o) has also o on the store (the contrary implies the infiniteness of the constraint system). As (t, o) is non-terminal, there exist a word $w \in \{a, b, c, d, e, f\}^*$ such that $(t, o) \xrightarrow{w.x} (s, o)$, where (s, o) is bisimilar to the state $U.X.Y$ of the PDA process from Example 7. If the rules labelled by actions x, z are removed from Δ and (s, o) is taken as an initial state, we obtain the system whose reachable states all have o as their store, bisimilar to the pushdown process from Example 1.

Now, let Δ' be a PAN system with the initial state s and with the set of rewrite rules consisting of rules $l \xrightarrow{v} r$, where $(l \xrightarrow{v} r, m, n) \in \Delta$, $o \vdash m$, $o \wedge n = o$ and $v \in \{a, b, c, d, e, f\}$. It is obvious that this PAN system Δ' is bisimilar to the PDA system defined in Example 1. This is a contradiction. ■

Corollary 3. *fcBPA $\subsetneq$ PDA, fcPA $\subsetneq$ fcPAD and fcPAD $\not\subseteq$ fcPAN.*

Example 8. Let Δ be a PAN process with the initial state $(X\|A\|B).W$ and the following rewrite rules.

$$\begin{array}{llll} X \xrightarrow{g} X\|A\|B & Y\|A \xrightarrow{a} Y & X \xrightarrow{y} \varepsilon & W \xrightarrow{x} (X\|A\|B).W \\ X \xrightarrow{c} Y & Y\|B \xrightarrow{b} Y & Y \xrightarrow{y} \varepsilon & W \xrightarrow{z} D \\ X\|A \xrightarrow{d} Z & Y\|A \xrightarrow{d} Z & Z \xrightarrow{y} \varepsilon & \\ X\|B \xrightarrow{d} Z & Y\|B \xrightarrow{d} Z & A \xrightarrow{y} \varepsilon & \\ & & B \xrightarrow{y} \varepsilon & \end{array}$$

The first two columns of rewrite rules include the same rules as Petri net given by Example 2. This PAN system can behave as mentioned Petri net (it can deviate from the behaviour of PN only under action y). States of PAN corresponding to terminal states of considered PN can perform a sequence of actions y^* to reach the state W and then terminate under action z or restart the system under action x.

Lemma 5. *There is no fcPAD system bisimilar to the PAN process from Example 8.*

Proof. The proof is similar to the proof of the previous lemma, instead of PDA from Example 1 it uses Petri net from Example 2. ∎

Corollary 4. *fcPA $\subsetneq$ fcPAN and fcPAN $\not\subseteq$ fcPAD.*

The incomparability of fcPAD and fcPAN implies that these classes are strict subclasses of fcPRS.

The edge between PRS and fcPRS classes in the fcPRS-hierarchy is dotted as we have no proof that the fcPRS class is strictly more expressive (w.r.t. bisimilarity) than the PRS class. It is obvious from the definitions that PRS $\subseteq$ fcPRS, but we can provide only intuition for PRS $\subsetneq$ fcPRS. The conjectured witness of the inequality can be found in the fcPA below.

Example 9. Let Δ be an fcPA system with the initial process term $X\|Y$ and the following constraint system and rewrite rules.

$$\begin{array}{c} ff \\ | \\ o \\ | \\ p \\ | \\ tt \end{array} \qquad \begin{array}{ll} (X \xrightarrow{a} X.A, tt, tt) & (A \xrightarrow{a'} \varepsilon, o, tt) \\ (X \xrightarrow{b} X.B, tt, tt) & (B \xrightarrow{b'} \varepsilon, o, tt) \\ (Y \xrightarrow{c} Y\|C, tt, tt) & (C \xrightarrow{c'} \varepsilon, o, tt) \\ (X \xrightarrow{x} \varepsilon, tt, p) & \\ (Y \xrightarrow{y} \varepsilon, p, o) & \end{array}$$

We can prove that this fcPA system is not bisimilar to any PAD process and to any Petri net either.

Now we try to explain why we conjecture that there is no PRS process bisimilar to the considered fcPA. The weak point of PRS (or rewrite system in general) is the "local potency" of rewriting. Having a parallel composition with at least one sequential component larger than the left side of any rewrite rule, the rule cannot influence both this large component and the rest of the parallel composition at once. Roughly speaking, communication between large component and other component(s) of parallel composition is not possible in general. Any PRS process bisimilar to the fcPA system under consideration should have such a parallel composition with one component which has a sequential character (as it is necessary to keep the information about the order in which the actions a and b are performed) and it can be arbitrary large. And we need to announce to the term that action y has just been done.

6 Conclusion

We have enriched process rewrite systems with the mechanism related to computing with partial information in the form used in widely studied concurrent constraint programming. In the case of process rewrite systems, this mechanism can be effectively used to provide some information to every part of the process term, thus it can be seen as a unit holding a special kind of global information.

It has been proven that enriching the classes of finite systems, pushdown processes, and Petri nets with a finite constraint system does not change their expressibility even w.r.t. isomorphism of the generated labelled transition systems. On the contrary, the process rewrite systems classes BPA, BPP, PA, PAD, and PAN extended with finite constraint systems establish corresponding new classes fcBPA, fcBPP, fcPA, fcPAD, and fcPAN as the expressive power of such systems increases. This may seem quite surprising in the cases of PAD and PAN classes as the formalism of these classes subsumes the formalism of PDA or PN respectively. However PDA and PN do not increase their expressive power if enriched with a finite constraint system.

The hierarchy of fcPRS classes has been introduced and its strictness w.r.t. the bisimulation equivalence (with the exception in the relation between PRS and fcPRS classes) has been proven.

The area of process rewrite systems with finite constraint systems still offers some interesting topics for further research. One interesting challenge is to specify the boundary of decidability of the bisimulation equivalence and the weak bisimulation equivalence with finite-state processes in the area of fcBPP class (as both problems are decidable for BPP and undecidable in the case of MSA[5]). Another possible topic for further research is to replace the constraint system with a (finite) state unit, where the evolution of the actual state is determined by a given ordering. A totally different mission is to employ an infinite constraint system.

Acknowledgements: I would like to thank Mojmír Křetínský for valuable discussions, reading the draft, and constant support. My thanks go also to Antonín Kučera for encouragement and inspiring comments. I thank three anonymnous referees for helpful remarks.

References

1. Bergstra, J. A., and J. W. Klop, *Algebra of communicating processes with abstraction*, Theoretical Computer Science **37** (1985), 77–121.
2. Burkart, O., D. Caucal, and B. Steffen, *Bisimulation collapse and the process taxonomy*, Proceedings of CONCUR '96, Lecture Notes in Computer Science, vol. 1119, Springer-Verlag, 1996, 247–262.
3. Burkart, O., and J. Esparza, *More infinite results*, Bulletin of the European Association for Theoretical Computer Science **62** (1997), 138–159.
4. Caucal, D., *On the regular structure of prefix rewriting*, Theoretical Computer Science **106** (1992), 61–86.
5. Christensen, S., "Decidability and Decomposition in Process Algebras," Ph.D. thesis, Department of Computer Science, University of Edinburgh, 1993.

[5] MSA are in [11] called PPDA. In [12] it was also demonstrated that the class of MSA is a strict subclass of Petri nets. It was proven in [15] that fcBPP is a strict subclass of MSA and that the expressibility of MSA systems is not changed by enriching them with finite constraint systems.

6. Groote, J. F., and F. Vaandrager, *Structured operational semantics and bisimulation as a congruence*, Information and Computation **100(2)** (1992), 202–260.
7. Mayr, R., *Combining Petri nets and PA-processes*, Proceedings of TACS '97, Lecture Notes in Computer Science, vol. 1281, Springer-Verlag, 1997, 547–561.
8. Mayr, R., *Process rewrite systems*, Electronic Notes in Theoretical Computer Science **7** (1997).
9. Milner, R., "A calculus on communicating systems," Lecture Notes in Computer Science, vol. 92, Springer-Verlag, 1980.
10. Milner, R., "Communication and Concurrency," Prentice Hall, 1989.
11. Moller, F., *Infinite results*, Proceedings of CONCUR '96, Lecture Notes in Computer Science, vol. 1119, Springer-Verlag, 1996, 195–216.
12. Moller, F., *A taxonomy of infinite state processes*, Electronic Notes in Theoretical Computer Science **18** (1998).
13. Park, D. M. R., *Concurrency and automata on infinite sequences*, Theoretical Computer Science: 5th GI-Conference, Lecture Notes in Computer Science, vol. 104, Springer-Verlag, 1981, 167–183.
14. Saraswat, V. A., "Concurrent Constraint Programming Languages," Ph.D. thesis, Computer Science Department, Carnegie Mellon University, 1989.
15. Strejček, J., *Constrained rewrite transition systems*, Technical Report FIMU-RS-2000-12, Faculty of Informatics, Masaryk University Brno, 2000.

A Appendix - Pumping lemma for fcBPP (Lemma 3)

The pumping lemma for fcBPP is formulated and proved in this appendix. The proof is similar to the one presented by Christensen for BPP case [5] thanks to the fact that every possible sequence of actions contains a finite number of transitions which change the state of the store due to finiteness of a constraint system.

Let $\Delta = (\mathcal{C}, R, t_0)$ be an fcBPP. For every process constant $X \in Const(\Delta)$ and every constraint $m \in C^\diamond$, let $S_m(X)$ denote the set

$$S_m(X) = \{Y \in Const(\Delta) \mid \exists t \in P : (X, m) \longrightarrow^+ (Y \| t, m)\},$$

i.e. the set of process constants Y which can be derived[6] from (X, m) without changes on the store. We extend this definition to parallel terms in obvious manner:

$$S_m(A_1 \| A_2 \| \dots \| A_j) = \bigcup_{i \in \{1,2,\dots,j\}} S_m(A_i)$$

Lemma 6. *Let $\Delta = (\mathcal{C}, R, t_0)$ be an fcBPP. If there exists some derivation of a word $u = u_1 u_2 \dots u_k \in L(\Delta)$ of the form*

$$(t_0, tt) = (t_0, m_0) \xrightarrow{u_1} (t_1, m_1) \xrightarrow{u_2} \dots \xrightarrow{u_k} (t_k, m_k) \not\longrightarrow$$

such that $\forall i \in \{0, 1, 2, \dots, k\}, \forall X \in t_i$ it holds $X \notin S_{m_i}(X)$, then $|u| \leq h$, where h is a constant depending only on Δ.

[6] The relation $\longrightarrow^+$ (resp. $\longrightarrow^*$) is apprehended as usual, i.e. $(t_1, m) \longrightarrow^+ (t_2, n)$ (resp. $(t_1, m) \longrightarrow^* (t_2, n)$) iff there exists $w \in Act^+$ (resp. $w \in Act^*$) such that $(t_1, m) \xrightarrow{w} (t_2, n)$.

Proof. At first we focus on maximum "flat" parts of the above derivation, which are of the form

$$(t_i, m_i) \xrightarrow{u_{i+1}} (t_{i+1}, m_{i+1}) \xrightarrow{u_{i+2}} \dots \xrightarrow{u_{i+j}} (t_{i+j}, m_{i+j}),$$

where the state of the store (in following marked as m) keeps unchanged ($m = m_i = m_{i+1} = \dots = m_{i+j}$), $i = 0$ or $m_{i-1} \neq m$, and $i+j = k$ or $m \neq m_{i+j+1}$. We denote $u' = u_{i+1}u_{i+2}\dots u_{i+j}$. From this flat part we deduce another derivation sequence

$$(r_0\|s_0, m) \xrightarrow{v_1} (r_1\|s_1, m) \xrightarrow{v_2} \dots \xrightarrow{v_p} (r_p\|s_p, m),$$

where $v_1, v_2, \dots, v_p \in Act^+$, $r_0\|s_0 = t_i$, in r_0 there are all constants from t_i which are rewritten in the derivation sequence $(t_i, m) \xrightarrow{u'} (t_{i+j}, m)$, and in s_0 there are constants which do not actively participate in this derivation sequence. Now $r_l\|s_l$ ($l = 1, 2, \dots, p$) rises from $r_{l-1}\|s_{l-1}$ by one rewriting of each constant from r_{l-1} in the same way as a constant has been rewritten in the original flat derivation sequence (thus $|v_l| = |r_{l-1}|$) and still it holds that r_l contains constants, which are rewritten in the original flat derivation sequence, while s_l contains the other constants (thus $s_{l-1} \subseteq s_l$). We finish rewriting when r_l is empty (thus $r_p = \varepsilon$ and $s_p = t_{i+j}$). It is clear that $v = v_1v_2\dots v_p$ is a permutation of u', especially $|v| = |u'|$. By replacing $(t_i, m) \xrightarrow{u'} (t_{i+j}, m)$ with $(r_0\|s_0, m) \xrightarrow{v} (r_p\|s_p, m)$ in the original derivation we get a correct derivation of the word $u_1\dots u_i v u_{i+j+1}\dots u_n$ of the length k. Further, for each X in r_l ($l = 0, 1, 2, \dots, p$) there exists t_z ($i \leq z \leq i+j$) such that $X \in t_z$.

Now we show that $S_m(r_{l-1}) \supsetneq S_m(r_l)$ for each $1 \leq l < p$.

"$\supseteq$" It comes directly from the fact that each constant from r_l has an ancestor in r_{l-1}.

"$\neq$" Let us assume that for some $1 \leq l < p$ we have $S_m(r_{l-1}) = S_m(r_l)$. For each $X \in r_l$ ($r_l \neq \varepsilon$) it holds that $X \in S_m(r_{l-1})$ and thus $X \in S_m(r_l)$. From the premise $X \notin S_m(X)$ follows that there exists some $Y \in r_l$, $Y \neq X$ such that $X \in S_m(Y)$. Analogous reasoning as for X can be done for Y, i.e. from $Y \in r_l$ it follows that $Y \in S_m(r_{l-1}) = S_m(r_l)$ and $Y \notin S_m(Y)$, $Y \notin S_m(X)$. In conclusion we get $Y \in S_m(r_l)$ and $Y \notin S_m(X\|Y)$. Again, there exists $Z \in r_l$, $Z \notin \{X, Y\}$ such that $Y \in S_m(Z)$ and thus also $\{X, Y\} \subseteq S_m(Z)$. We know $Z \in r_l$ and $Z \notin S_m(Z)$, hence we get $Z \in S_m(r_l)$ and $Z \notin S_m(X\|Y\|Z)$. We can continue in this fashion to the point where we have the contradiction $W \in S_m(r_l)$ and $W \notin S_m(r_l)$.

Hence we have

$$|Const(\Delta)| \geq |S_m(r_0)| > |S_m(r_1)| > \dots > |S_m(r_{p-1})| \geq 0.$$

This implies $|Const(\Delta)| \geq p - 1$. Further, for each $1 \leq l \leq p$ it holds that

$$|v_l| = |r_{l-1}| \leq |r_0|a^{l-1} \leq |r_0|a^{p-1} \leq |r_0|a^{|Const(\Delta)|},$$

where a is a maximum number of constants in right sides of rewrite rules in Δ. Now we restrict the length of u'

$$|u'| = |v| = \sum_{l=1}^{p} |v_l| \leq \sum_{l=1}^{p} |r_0|a^{|Const(\Delta)|} = p|r_0|a^{|Const(\Delta)|},$$
$$|u'| \leq p|r_0|a^{|Const(\Delta)|} \leq (|Const(\Delta)| + 1)|t_i|a^{|Const(\Delta)|}.$$

In conclusion we get the restriction on the length of flat parts of the original derivation

$$|u'| \leq |t_i|b,$$

where $b = (|Const(\Delta)| + 1)a^{|Const(\Delta)|}$.

In general it holds that each sequence of derivation steps consists of non-flat steps and flat derivation sequences. The number of "unflat" steps $(t_i, m_i) \xrightarrow{u_{i+1}} (t_{i+1}, m_{i+1})$, where $m_i \neq m_{i+1}$, is limited by $|C^\diamond| - 1$. The cardinality of the set C also constrains the number of flat parts to $|C^\diamond|$. Therefore

$$|u| \leq |C^\diamond| - 1 + \sum_{j=1}^{|C^\diamond|} |t'_j|b,$$

where (t'_j, m'_j) is the first state of the j-th flat derivation sequence, i.e. m'_j is the j-th different state of the store used in the original derivation and (t'_j, m'_j) is the first state in this derivation with the constraint m'_j in the store. Hence $(t'_1, m'_1) = (t_0, tt)$.

The last step is to restrict the length of t'_j for $j > 1$. We can deduce a restriction

$$|t'_j| \leq |t'_{j-1}| + (a - 1)(|t'_{j-1}|b + 1)$$

thanks to the facts that each application of a rewrite rule cannot add more than $a - 1$ constants to the string of constants in the actual state and that the number of these applications is limited by the length of the previous flat string plus one (the unflat derivation step). The previous inequality can be modified in the following way.

$$|t'_j| \leq |t'_{j-1}| + a(|t'_{j-1}|b + 1)$$
$$|t'_j| \leq |t'_{j-1}|(1 + ab + a)$$
$$|t'_j| \leq |t'_1|(1 + ab + a)^{j-1}$$
$$|t'_j| \leq |t_0|(1 + ab + a)^{j-1}$$

By summarisation we get

$$|u| \leq |C^\diamond| - 1 + b|t_0| \sum_{j=1}^{|C^\diamond|} (1 + ab + a)^{j-1},$$

where $b = (|Const(\Delta)|+1)a^{|Const(\Delta)|}$. The sum on the right side of the previous inequality can be modified as it is an geometric progression. The final form of desired h is then

$$h = |C^{\diamond}| - 1 + b|t_0|\frac{(1+ab+a)^{|C^{\diamond}|} - 1}{ab+a},$$

where a is the maximum number of constants in right sides of rewrite rules in Δ and $b = (|Const(\Delta)| + 1)a^{|Const(\Delta)|}$. ■

The pumping lemma formulated below is a simple consequence of the previous lemma.

Lemma 7 (Pumping Lemma for fcBPP). *Let L be a language of an fcBPP system Δ. There exists a constant h such that if u is a word of L and $|u| > h$ then there exist $x, y, z, w \in Act^*$ such that*

- $u = xz$,
- $|y| > 1$,
- $\forall i \geq 0 : xy^i zw^i \in L$.

Proof. We have an fcBPP Δ such that $L = L(\Delta)$. It follows from Lemma 6 that each derivation

$$(t_0, tt) = (t_0, m_0) \xrightarrow{u_1} (t_1, m_1) \xrightarrow{u_2} \dots \xrightarrow{u_k} (t_k, m_k) \not\rightarrow$$

of the word $u = u_1 u_2 \dots u_k \in L(\Delta)$, $|u| > h$ contains some state $(t_j, m_j) = (X \| t'_j, m_j)$, where $X \in S_{m_j}(X)$. The definition of $S_{m_j}(X)$ says that there exist $t \in P$ and $y \in Act^+$ such that $(X, m_j) \xrightarrow{y} (X \| t, m_j)$. Further, let $w \in Act^*$ be a word in $L((t, m_k))$, i.e. there exists a terminal state (t', n) such that $(t, m_k) \xrightarrow{w} (t', n)$. Now the derivation

$$(t_0, tt) \xrightarrow{u_1 \dots u_j} (t_j, m_j) \xrightarrow{y^i} (t_j t^i, m_j) \xrightarrow{u_{j+1} \dots u_k} (t^i, m_k) \xrightarrow{w^i} (t'^i, n) \not\rightarrow$$

is the correct one for all $i \geq 0$. To make the proof complete we should add that $x = u_1 \dots u_j$ and $z = u_{j+1} \dots u_k$. ■