FI MU

Faculty of Informatics
Masaryk University

Constrained Rewrite Transition Systems

by

Jan Strejček

FI MU Report Series FIMU-RS-2000-12

 December 2000

Constrained Rewrite Transition Systems*

Jan Strejček

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

Abstract

We extend broadly studied rewrite transition systems with a mechanism for computing with partial information in the form similar to that one used in Concurrent Constraint Programming (CCP). Two new classes of transition systems (fcBPA and fcBPP) are introduced as this extension changes expressibility power of rewrite transition systems corresponding to BPA and BPP. The power of rewrite systems corresponding to other classes (FSA, PDA, PPDA, and PN) remains unchanged. The new classes are inserted to the hierarchy of standard process classes presented by Moller [Mol96].

1 Introduction

For many years, computing with partial information is one of deeply studied domains of theoretical computer science. This conception becomes even more interesting in conjunction with the idea of concurrency as this combination corresponds to situations occurring in real world and thus can be used for modeling such situations.

One of the most successful applications of the ideas of concurrency and computing with partial information has led to Concurrent Constraint Programming (CCP) presented by Saraswat [Sar89] and consequently studied

*This work has been partially supported by the Grant Agency of Czech Republic, grants No. 201/00/0400 and No. 201/00/1023.

also by Rinard, Panangaden, de Boer, Palamidessi and others (see References for more details). In CCP processes work concurrently with a shared *store*, which is seen as a constraint on the values that variables can assume. At any stage of the computation, the store is given by the constraint established until that moment. CCP provides two primitive actions for the manipulation with the store, *tell* and *ask*. The execution of a *tell* action modifies the current store by adding a constraint (*tell* can be executed under the condition that the store keeps *consistent*, i.e. there is some valuation of variables which satisfies constraint in the store). An *ask* action 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 the contributions of the other processes). The execution of an *ask* itself leaves the store unchanged. Hence both *tell* and *ask* actions are monotonic in the sense that after their execution the store contains the same or more information. Therefore the store evolves monotonically during the computation, i.e. the set of possible values for the variables shrinks.

Operational semantics of concurrent systems is traditionally modeled by labelled transition systems. For CCP such an operational semantics was given by Saraswat in [Sar89]. Caucal [Cau92] presents an elegant classification of transition systems using families of rewrite systems defined by restrictions on rewrite rules related with Chomsky hierarchy. Caucal's classification has been generalized by Moller [Mol96] to both, parallel and sequential rewrite transition systems.

We include some principles of CCP to rewrite systems with the aim to observe changes of expressibility power of these systems. The mechanism of rewrite systems is extended by the store which can contain a partial information. We talk about constraints (as the theory around CCP does) although we do not specify the shape of partial information as sharply as CCP does. We add two constraints to every standard rewrite rule. The 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 if the extended rule is used (the rule is applicable under condition the store keeps consistent). After application of the rule the store contains the same or more information, thus we say that the store is monotonic. Extended systems are called *Constrained Labelled Rewrite Transition Systems*.

The comparison of standard and constrained rewrite transition systems gives some interesting results. At first, rewrite systems corresponding to transition systems of FSA, PDA, PPDA, and PN classes do not change their expressibility power by adding the store. More interesting result is that the expressibility power of rewrite systems corresponding to transition systems of BPA and BPP classes strictly increases, hence two new classes of transition systems are introduced and Moller hierarchy is refined.

1.1 Outline of the paper

The rest of the paper is structured as follows. In Section 2 we summarize Moller's results, especially the classification and the hierarchy of standard classes of transition systems. Section 3 defines the notion of constrained rewrite transition system and presents the classification and the hierarchy of classes of transition systems which can be defined by such constrained rewrite transition system. This section also introduces two new classes, fcBPP and fcBPA, which are studied in detail in Section 4 (fcBPP) and Section 5 (fcBPA). The paper closes with a section that summarizes our results and points out some directions for future research.

2 Rewrite Transition Systems

In this section we summarize (and slightly modify) the first part of Moller's paper titled "Infinite Results" [Mol96].

2.1 Definitions

Concurrent systems are modeled semantically as edge-labelled directed graphs, whose nodes represent the states in which a system may exist, and whose transitions represent the possible behavior of the system originating in the state represented by the node from which the transition emanates. A label assigned to a transition represents an event (or action) corresponding to the execution of the transition, which will typically represent an interaction with the environment. The starting point for our study will be such graphs (representing processes).

Definition 2.1. *A* labelled transition system $\mathcal{T}$ *is a tuple* $(S, \Sigma, \Longrightarrow, \alpha_0, F)$ *where*

- S *is a set of* states,
- Σ *is a finite set of* labels,
- $\Longrightarrow \subseteq S \times \Sigma \times S$ *is a* transition relation, *written* $\alpha \stackrel{a}{\Longrightarrow} \beta$ *for* $(\alpha, a, \beta) \in \Longrightarrow$,

- α_0 *is a distinguished* start state,
- $F \subseteq S$ *is a finite set of* final states *which are* terminal, *i.e. for each* $\alpha \in F$ *there is no* $a \in \Sigma$ *and* $\beta \in S$ *such that* $\alpha \overset{a}{\Longrightarrow} \beta$.

This notion of a labelled transition system differs from the standard definition of a (nondeterministic) finite-state automaton (as for example given in [HU79]) in possible infiniteness of the set of states and in that final states must not have any outgoing transitions. This last restriction is mild and justified by the fact that a final state refers to the successful termination of a concurrent system. This contrasts with unsuccessful termination (i.e. deadlock) which is represented by all non-final terminal states.

We follow the example set by Caucal [Cau92] and consider the families of labelled transition systems defined by various rewrite systems. Such an approach provides us with a clear link between well-studied classes of formal languages and transition system generators, a link which is of particular interest when it comes to exploiting techniques from process theory in solving problems in classic formal language theory.

Definition 2.2. *A* sequential labelled rewrite transition system $\mathcal{V}$ *is a tuple* $(V, \Sigma, P, \alpha_0, F)$ *where*

- V *is a finite set of* variables, *the elements of* V^* *are referred to as* states,
- Σ *is a finite set of* labels,
- $P \subseteq V^* \times \Sigma \times V^*$ *is a finite set of* rewrite rules, *written* $\alpha \overset{a}{\longrightarrow} \beta$ *for* $(\alpha, a, \beta) \in P$. *A* transition relation $\Longrightarrow$ *is derived from* P *by* prefix rewriting rule, *i.e. if* $\alpha \overset{a}{\longrightarrow} \beta$ *then* $\alpha\gamma \overset{a}{\Longrightarrow} \beta\gamma$ *for each* $\gamma \in V^*$,
- $\alpha_0 \in V^*$ *is a distinguished* start state,
- $F \subseteq V^*$ *is a finite set of* final states *which are* terminal.

A parallel labelled rewrite transition system *is defined as above, except that the elements of* V^* *are read modulo commutativity of catenation, which is thus interpreted as parallel composition rather than sequential composition (for example* $XBB = BXB = BBX$*).*

We shall freely extend the transition relation $\Longrightarrow$ homomorphically to finite sequences of actions $w \in \Sigma^*$ so as to write $\alpha \overset{\varepsilon}{\Longrightarrow} \alpha$ and $\alpha \overset{aw}{\Longrightarrow} \beta$ whenever $\alpha \overset{a}{\Longrightarrow} \gamma \overset{w}{\Longrightarrow} \beta$ for some state γ. The set of states α such that

$\alpha_0 \stackrel{w}{\Longrightarrow} \alpha$ for the initial state α_0 and some $w \in \Sigma^*$ is referred as the set of *reachable* states. Although we do not insist that all states are reachable, we shall assume that all variables in V are accessible from the initial state, i.e. for each $X \in V$ there is some $w \in \Sigma^*$ and $\alpha, \beta \in V^*$ such that $\alpha_0 \stackrel{w}{\Longrightarrow} \alpha X \beta$.

This definition is slightly more general than that given by Caucal which does not take into account final states nor the possibility of parallel rewriting as an alternative to sequential rewriting. By doing this, we expand the study of the classes of transition systems which are defined and extend some of the results given by Caucal.

2.2 Languages and bisimilarity

An important question in the realm of concurrency theory is to determine when two transition systems are considered as "the same". It turns out that the isomorphism of transition systems is too strong equivalence. The plethora of other equivalences was defined in eighties by many people, the catalog of these equivalences was compiled by van Glabbeek [vG90]. We define just two of them, namely language equivalence and bisimulation equivalence.

Given a labelled transition system T with initial state α_0, we can define its *language* $L(T)$ to be the language generated by its initial state α_0, where the language generated by a state is defined in usual fashion as sequences of labels associated with transitions leading from the given state to a final state.

Definition 2.3. *The* language *generated by a labelled transition system T (with initial state α_0 and a set of final states F) is the set $L(T) = L(\alpha_0)$, where*

$$L(\alpha) = \{w \in \Sigma^* \mid \alpha \stackrel{w}{\Longrightarrow} \beta \text{ for some } \beta \in F\}.$$

States α and β of the system T are language equivalent, *written $\alpha \sim_L \beta$, iff they generate the same language, i.e. $L(\alpha) = L(\beta)$.*

Language equivalence is generally taken to be too coarse. The second presented equivalence, *bisimulation equivalence*, is perhaps the finest behavioral equivalence studied. Bisimulation equivalence was defined by Park [Par81] and used to great effect by Milner [Mil80, Mil89]. Its definition, in presence of final states, is as follows.

Definition 2.4. *A binary relation $\mathcal{R}$ on states of transition system is a* bisimulation *iff whenever $(\alpha, \beta) \in \mathcal{R}$ we have that*

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

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

Bisimulation equivalence has an elegant characterization in terms of certain two-player games presented by Stirling [Sti95].

The last definition in this subsection is the definition of *norm* which is used mainly in Section 5.

Definition 2.5. *We define the* norm *of the state* α *of a labelled transition system, written* $norm(\alpha)$*, to be the length of the shortest rewrite transition sequence which takes* α *to a final state, that is, the length of a shortest word in* $L(\alpha)$*. By convention, we define* $norm(\alpha) = \infty$ *if there is no sequence of transitions from* α *to a final state, that is,* $L(\alpha) = \emptyset$*. The transition system is* normed *iff every reachable state* α *has a finite norm.*

2.3 Classification

The families of transition systems which can be defined by restricted rewrite systems can be easily classified using a form of Chomsky hierarchy. (Type 1, context-sensitive rewrite systems do not feature in this hierarchy since the rewrite rules by definition are only applied to the prefix of a composition.) This hierarchy provides an elegant classification of several important classes of transition systems which have been defined and studied independently of their appearance as particular rewrite systems. This classification is presented as follows.

	Restriction on the rules $\alpha \stackrel{a}{\longrightarrow} \beta$ of P	Restriction on F	Sequential composition	Parallel composition
Type 0	none	none	PDA	PN
Type $1\frac{1}{2}$	$\alpha \in Q\Gamma$ and $\beta \in Q\Gamma^*$ where $V = Q \uplus \Gamma$ [1]	$F = Q$	PDA	PPDA
Type 2	$\alpha \in V$	$F = \{\varepsilon\}$	BPA	BPP
Type 3	$\alpha \in V, \beta \in V \cup \{\varepsilon\}$ [2]	$F = \{\varepsilon\}$	FSA	FSA

In the reminder of this subsection, we explain the classes of transition systems which are represented in this table.

FSA represents the class of *finite-state automata*. Clearly if the rules are restricted to be of the form $A \xrightarrow{a} B$ or $A \xrightarrow{a} \varepsilon$ with $A, B \in V$, then the reachable states of both the sequential and parallel transition systems will be a subset of the finite set of variables V.

BPA represents the class of *Basic Process Algebra* processes of Bergstra and Klop [BK85], which are the transition systems associated with Greibach normal form (GNF) context-free grammars in which only left-most derivations are allowed.

BPP represents the class of *Basic Parallel Processes* introduced by Christensen [Chr93] as a parallel analogy to BPA, and are defined by the transition system associated with GNF context-free grammars in which arbitrary grammar derivations are permitted.

PDA represents the class of *push-down automata* which accept on empty stack. To define PDA as a restricted form of rewrite system, we assume that the variable set V is partitioned into disjoint sets Q (finite control states) and Γ (stack symbols). The rewrite rules are of the form $pA \xrightarrow{a} q\beta$ with $p, q \in Q$, $A \in \Gamma$ and $\beta \in \Gamma^*$, which represents the usual PDA transition saying that while in control state p with the symbol A at the top of the stack, PDA may read the input symbol a, move into control state q, and replace the stack element A with the sequence β. Finally, the set of final states is given by Q, which represent the PDA configurations in which the stack is empty. PDA can be seen as a state-extended BPA.

Caucal [Cau92] demonstrates that, disregarding final states, any unrestricted (type 0) sequential rewrite transition system can be presented as a PDA, in the sense that the transition systems are isomorphic up to the labelling of states. The stronger result, in which final states are taken into consideration, holds as well.

PPDA represents the class of *"parallel" push-down automata*, which are defined as PDA except that they have random access capability to the stack. Thus a PPDA transition rule $pA \xrightarrow{a} q\beta$ with $p, q \in Q$, $A \in \Gamma$ and $\beta \in \Gamma^*$ says that while in control state p with the symbol A anywhere in the stack, PPDA may read the input symbol a, move into control state q, and replace the stack element A with the sequence β. As the order of symbols in the

[1] We assume that $\alpha_0 \in Q\Gamma^*$.

[2] We also assume that the initial state α_0 is a member of V.

stack is ignored, the stack can be seen as a multiset. This is the reason why PPDA are also called MSA (*multiset automata*). PPDA can be also presented as state-extended BPP.

PN represents the class of (finite, labelled, weighted place/transition) *Petri nets*, as is justified by the following interpretation of unrestricted parallel rewrite systems. The variable set V represents the set of places of the Petri net, and each rewrite rule $\alpha \xrightarrow{a} \beta$ represents a Petri net transition labelled by a with the input and output places represented by α and β respectively, with the weights on the input and output arcs given by the relevant multiplicities in α and β.

2.4 Hierarchy

The classification of standard classes of transition systems presented in previous subsection gives some relations between these classes directly from the form of restrictions on corresponding rewrite systems. It is easy to see that FSA is a subclass of BPA and BPP, BPA is a subclass of PDA, and that BPP and PPDA are subclasses of PN. Still not hard to see is the fact that BPP is a subclass of PPDA. The whole hierarchy is depicted in Figure 1.

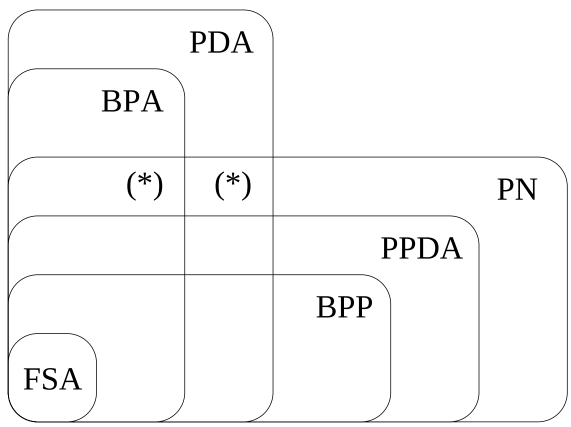

Figure 1: Hierarchy of classes of transition systems

The strictness of this hierarchy (except for the relation between PN and PPDA) with respect to bisimulation equivalence was compactly demonstrated by Moller in [Mol96], where he complements the results presented by Burkart, Caucal and Steffen [BCS96]. The strictness with respect to language equivalence doesn't hold in general. For example, both BPA and PDA express exactly the (ε-free) context-free languages.

The question about relation between PN and PPDA was partially solved by Moller in [Mol98], where he presents an example of PN which is not bisimilar neither to any PPDA nor to any PDA. Uncertainty around existence of the gaps marked in Figure 1 with (*) remains open.

Recently, Hirshfeld proved in so far unpublished manuscript that PN and PPDA are language equivalent.

3 Constrained Rewrite Transition Systems

This section presents the new way how some (possibly infinite) transition systems can be finitely represented as a constrained rewrite transition system. The notion of constrained rewrite transition systems has grown up from rewrite transition systems presented in previous section and from the idea of the common store used in Concurrent Constraint Programming.

3.1 Constraint systems

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

Definition 3.1. *A* constraint system *is a complete lattice* $(C, \leq, \wedge, tt, ff)$*, where* C *is the set of* constraints, $\leq$ *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 ($tt \neq ff$)*.*

Following the standard terminology and notation, instead of $\leq$ we will refer to its inverse relation, denoted by $\vdash$ and called *entailment*. Formally

$$\forall m, n \in C : m \vdash n \iff n \leq m.$$

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

Two following examples show constraint systems employed in the rest of this thesis.

Example 3.2. *Let* $C = \{tt, ff\}$*,* $\leq = \{(tt, ff), (tt, tt), (ff, ff)\}$*. Then* $\mathcal{C}_\varepsilon$ *is the trivial constraint system* $(C, \leq, \wedge, tt, ff)$ *depicted in Figure 2.*

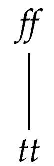

Figure 2: Constraint system $\mathcal{C}_\varepsilon$

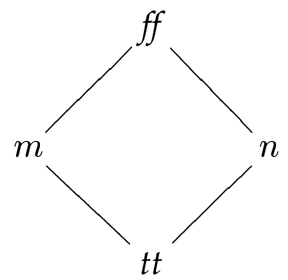

Figure 3: Constraint system $\mathcal{C}_{mn}$

Example 3.3. *Let* $C = \{tt, m, n, ff\}$*,* $\leq = \{(tt, ff), (tt, m), (tt, n), (m, ff), (n, ff)\} \cup \{(o, o) \mid o \in C\}$*. Then* $\mathcal{C}_{mn} = (C, \leq, \wedge, tt, ff)$ *is the constraint system depicted in Figure 3.*

We add one more example which can provide better illustration of the relation between partial information, constraint system and the evolution of the store.

Example 3.4. *The Herbrand constraint system on* $\{a, b\}$ *with variables* x, y *is diagrammatically represented in Figure 4.*

3.2 Definitions

Please observe the similarity between following definition of constrained rewrite transition system (CRTS) and the definition of labelled rewrite transition system (Definition 2.2). A transition relation is defined separately as its definition is essential and deserves some explanation.

Definition 3.5. *A* sequential constrained labelled rewrite transition system $\mathcal{V}$ *is a tuple* $(\mathcal{C}, V, \Sigma, P, \alpha_0, F)$ *where*

- $\mathcal{C} = (C, \leq, \wedge, tt, ff)$ *is a finite constraint system describing the* store*; the elements of* C *represent the* states of the store*,*

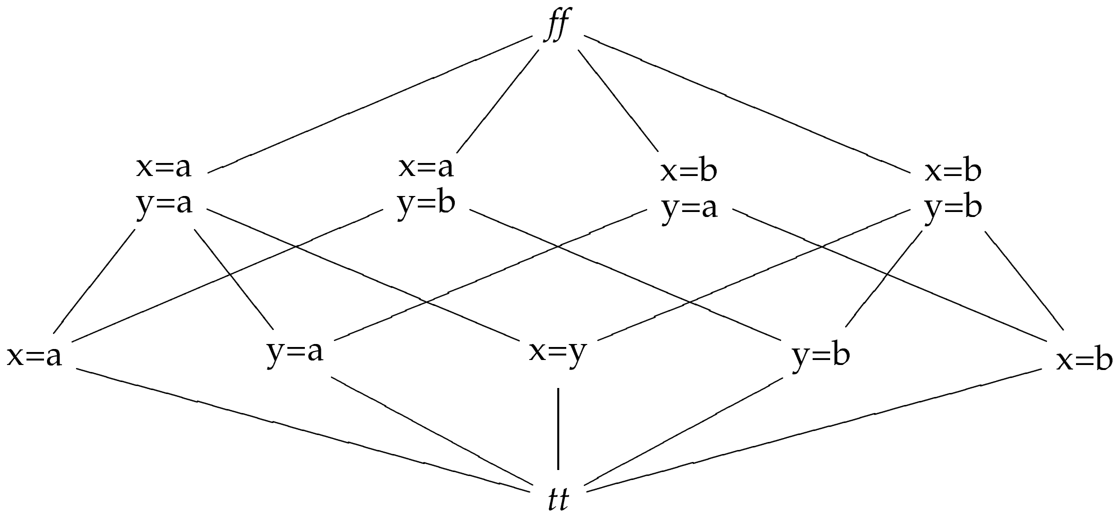

Figure 4: Herbrand constraint system on $\{a, b\}$ with variables x, y

- V *is a finite set of* variables; *the elements of* $V^* \times C^\diamond$ *are referred to as* states *of the transition system,*
- Σ *is a finite set of* labels *or* actions,
- $P \subseteq V^* \times \Sigma \times V^* \times C^\diamond \times C^\diamond$ *is a finite set of* rewrite rules, *written* $(\alpha \xrightarrow{a} \beta, m, n)$ *for* $(\alpha, a, \beta, m, n) \in P$,
- $(\alpha_0, tt) \in V^* \times C^\diamond$ *is a distinguished* start state,
- $F \subseteq V^* \times C^\diamond$ *is a finite set of* final states, *which are* terminal.

A parallel constrained labelled rewrite transition system *is defined as above, except that the elements of* V^* *are read modulo commutativity of catenation, which is interpreted as parallel composition now. Thus for example,* $XBB = BXB = BBX$.

The previous definition specifies the syntax of constrained rewrite transition systems only. A transition relation $\Longrightarrow$ should be introduced to assign the semantics for the syntax. The following definition stands for both sequential and parallel case.

Definition 3.6. *Let* $\mathcal{V} = (\mathcal{C}, V, \Sigma, P, \alpha_0, F)$ *be a constrained labelled rewrite transition system. For* $x, y \in V^*$, $a \in \Sigma$, $m, n, o \in C^\diamond$ *we write* $(x, o) \stackrel{a}{\Longrightarrow} (y, o \wedge n)$ *iff there exists some rewrite rule* $(\alpha \xrightarrow{a} \beta, m, n) \in P$ *such that* $x = \alpha\gamma$, $y = \beta\gamma$, $\gamma \in V^*$, $o \vdash m$, *and* $o \wedge n \neq ff$.

The last two conditions are very close to principles used in Concurrent Constraint Programming (CCP). The first one ($o \vdash m$) ensures that the appropriate rule will be used only if the actual state of the store *entails* constraint m from the rule (similar to *ask(m)* in CCP). The second condition ($o \wedge n \neq \mathit{ff}$) guarantees that the store keeps *consistent* after application of the rule (analogous to consistency requirement when processing *tell(n)* action in CCP). If these two conditions are satisfied then a rewriting under the rule $(\alpha \xrightarrow{a} \beta, m, n) \in P$ corresponds to a prefix rewriting under the rule $\alpha \xrightarrow{a} \beta$ in standard rewrite transition systems.

We shall freely extend a transition relation $\Longrightarrow$ homomorphically to finite sequences of actions $w \in \Sigma^*$ so as to write $(x, m) \stackrel{\varepsilon}{\Longrightarrow} (x, m)$ and $(x, m) \stackrel{aw}{\Longrightarrow} (y, n)$ whenever $(x, m) \stackrel{a}{\Longrightarrow} (z, o) \stackrel{w}{\Longrightarrow} (y, n)$ for some $(z, o) \in V^* \times C^{\diamond}$. The set of states (α, m) such that $(\alpha_0, \mathit{tt}) \stackrel{w}{\Longrightarrow} (\alpha, m)$ for the initial state (α_0, tt) and some $w \in \Sigma^*$ is referred as the set of *reachable* states.

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 application of some rewrite rule $(\alpha \xrightarrow{a} \beta, m, n) \in P$ and after this application the new state $o \wedge n$ always entails the old state o. Intuitively, the partial information can only be added to the store, not retracted. We say the store has *monotonic* behavior, or simply that the store is monotonic.

Note that when a rule is used for some transition then this rule can always be used again because the actual store already entails the first constraint of the rule (due to monotonic behavior) and consistency requirement is always satisfied. The last statement comes from the fact that the second constraint of the rule (denoted as n) changes the store only in the first application of the rule provided the content of the store before this application (denoted as o) does not entail n ($o \wedge n \neq o$). All subsequent applications of this rule do not change the store (again thanks to monotonic behavior), i.e. for each sequent state p of the store $p \wedge n = p$ holds.

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

The first information about the relation between constrained and standard rewrite transition systems is provided by following lemma formulated for sequential case (it holds for parallel case too).

Lemma 3.7. *Transition systems defined by sequential rewrite transition system $\mathcal{V}' = (V, \Sigma, P', \alpha_0, F')$ and by sequential CRTS $\mathcal{V} = (\mathcal{C}_\varepsilon, V, \Sigma, P, \alpha_0, F)$ are isomorphic on the assumption that $P' = \{\alpha \xrightarrow{a} \beta \mid (\alpha \xrightarrow{a} \beta, tt, tt) \in P\}$ and $F' = \{\alpha \mid (\alpha, tt) \in F\}$.*

Proof. Let $\mathcal{T}$ be the transition system corresponding to the sequential CRTS $\mathcal{V}$. The state of the store is *tt* in every state of $\mathcal{T}$ due to the shape of trivial constraint system $\mathcal{C}_\varepsilon$ (defined in Example 3.2).

Further, rewrite rules of the form $(\alpha \xrightarrow{a} \beta, tt, tt)$ are always applicable as two necessary conditions are always satisfied ($tt \vdash tt$ and $tt \wedge tt = tt \neq ff$).

Now we know that the transition system $\mathcal{T}$ is of the form $(S, \Sigma, \Longrightarrow, (\alpha_0, tt), F)$, where $S = \{(\alpha, tt) \mid \alpha \in V^*\}$ and the transition relation $\Longrightarrow$ can be alternatively defined as $(\alpha\gamma, tt) \stackrel{a}{\Longrightarrow} (\beta\gamma, tt)$ for each rewrite rule $(\alpha \xrightarrow{a} \beta, tt, tt) \in P$ and each $\gamma \in V^*$.

If we remove *tt* from the states of transition system $\mathcal{T}$, we get an isomorphic system $\mathcal{T}'$ which corresponds to the rewrite transition system $\mathcal{V}'$. $\square$

Roughly speaking, the lemma says that the trivial constraint system cannot hold any significant information and thus such a constrained rewrite transition system is isomorphic to the corresponding standard rewrite transition system. The lemma can be used in both directions, for proving that CRTS of the specified form has equivalent rewrite transition system as well as for constructing CRTS equivalent to the given rewrite transition system.

The following lemma describes another case when added constraint system cannot increase expressibility power of rewrite transition systems.

Lemma 3.8. *For every sequential CRTS $\mathcal{V} = (\mathcal{C}, V, \Sigma, P, \alpha_0, F)$ with the rewrite rules of the form $(\alpha \xrightarrow{a} \beta, tt, tt)$, there is (effectively constructible) sequential rewrite transition system $\mathcal{V}'$ with the transition system isomorphic to the transition system of $\mathcal{V}$.*

Proof. We may assume the constraint system $\mathcal{C}$ of $\mathcal{V}$ is not the trivial $\mathcal{C}_\varepsilon$ (if $\mathcal{C} = \mathcal{C}_\varepsilon$ then the lemma is a direct corollary of Lemma 3.7).

The fundamental step is to observe that the state n of the store cannot be changed by any application of the rewrite rule of the form $(\alpha \xrightarrow{a} \beta, tt, tt)$ as $n \wedge tt = n$. This means that there is no transition between states (α, n) and (β, m) for any $\alpha, \beta \in V^*$ and $m, n \in C^\diamond$, $m \neq n$.

Next important observation says that applicability of rewrite rules of the specified form does not depend on the current state of the store as necessary conditions are always satisfied because every $m \in C$ entails *tt* and every $n \in C^\diamond$ is consistent with tt ($n \wedge tt = n \neq ff$). Thus if there is a transition $(\alpha, tt) \stackrel{a}{\Longrightarrow} (\beta, tt)$ then there is also transition $(\alpha, m) \stackrel{a}{\Longrightarrow} (\beta, m)$ for every $m \in C^\diamond$.

The conclusion is that the transition system defined by $\mathcal{V}$ can be separated into $|C^\diamond|$[3] isolated isomorphic parts. Let $\mathcal{T}_m$ denote the part with m on the store for every $m \in C^\diamond$. It is easy to see that if we change the constraint system in $\mathcal{V}$ to $\mathcal{C}_\varepsilon$, the modified CRTS describes exactly $\mathcal{T}_{tt}$. From the Lemma 3.7 it follows that we can construct a standard rewrite transition system $\mathcal{V}'_{tt}$ with a transition graph isomorphic to $\mathcal{T}_{tt}$ (i.e. isomorphic to every $\mathcal{T}_m$). The desired rewrite transition system $\mathcal{V}'$ consists of $|C^\diamond|$ copies of $\mathcal{V}'_{tt}$. □

Again, the lemma given above holds for parallel case as well. Intuitively, the lemma says that if the power of the store is not employed by the rules, then (without respect to the current constraint system) the constrained rewrite transition system is isomorphic to some standard rewrite transition system. The proof also says that the reachable part of transition system defined by such CRTS consists of states with *tt* on the store.

Although adding finite constraint system looks like quite powerless expansion, in what follows we will demonstrate this mechanism can change expressibility of standard families of rewrite transition systems in some cases.

3.3 Languages and bisimilarity

As the syntax of states of transition systems defined by constrained rewrite transition systems is a bit different from the standard syntax, we enunciate modified definitions of language and bisimulation equivalences. The meaning of these definitions keeps unchanged.

Definition 3.9. *The* language *generated by a labelled transition system T defined by some CRTS $\mathcal{V} = (\mathcal{C}, V, \Sigma, P, \alpha_0, F)$ is the set $L(T) = L((\alpha_0, tt))$, where*

$$L((\alpha, m)) = \{w \in \Sigma^* \mid (\alpha, m) \stackrel{w}{\Longrightarrow} (\beta, n) \text{ for some } (\beta, n) \in F\}.$$

States (α, m) and (β, n) of T are language equivalent, *written $(\alpha, m) \sim_L (\beta, n)$, iff they generate the same language, i.e. $L((\alpha, m)) = L((\beta, n))$.*

Definition 3.10. *A binary relation $\mathcal{R}$ on states of transition system defined by some CRTS is a* bisimulation *iff whenever $((\alpha, m), (\beta, n)) \in \mathcal{R}$ we have that*

[3] Syntax $|M|$ is used to denote cardinality of the set M.

- *if* $(\alpha, m) \stackrel{a}{\Longrightarrow} (\alpha', m')$ *then* $(\beta, n) \stackrel{a}{\Longrightarrow} (\beta', n')$ *for some* (β', n') *with* $((\alpha', m'), (\beta', n')) \in \mathcal{R}$,
- *if* $(\beta, n) \stackrel{a}{\Longrightarrow} (\beta', n')$ *then* $(\alpha, m) \stackrel{a}{\Longrightarrow} (\alpha', m')$ *for some* (α', m') *with* $((\alpha', m'), (\beta', n')) \in \mathcal{R}$,
- $(\alpha, m) \in F$ *iff* $(\beta, n) \in F$.

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

3.4 Classification

The classification of families of transition systems, which can be defined by constrained rewrite transition systems, can be obtained in the same way as in the case of standard rewrite transition systems, i.e. by putting different restrictions derived from Chomsky hierarchy on rewrite rules.

	Restriction on the rules $(\alpha \stackrel{a}{\longrightarrow} \beta, m, n) \in P$	Restriction on F	Sequential composition	Parallel composition
Type 0	none	none	PDA	PN
Type $1\frac{1}{2}$	$\alpha \in Q\Gamma$ and $\beta \in Q\Gamma^*$ where $V = Q \uplus \Gamma$ [4]	$F = Q \times C^{\diamond}$	PDA	PPDA
Type 2	$\alpha \in V$	$F = \{\varepsilon\} \times C^{\diamond}$	**fcBPA**	**fcBPP**
Type 3	$\alpha \in V, \beta \in V \cup \{\varepsilon\}$ [5]	$F = \{\varepsilon\} \times C^{\diamond}$	FSA	FSA

The contiguous table is very similar to the one presented in Subsection 2.3. The only two essential differences can be found on the line marked with "Type 2". The sameness in other raws states that every transition system defined by CRTS of an arbitrary type except type 2 is identical to the transition system described by an appropriate rewrite system of the same type with respect to bisimulation equivalence. In fact, the sameness holds even with respect to isomorphism of transition graphs with one exception for CRTS of type 3. These statements are corollary of the following theorems and Lemma 3.7.

[4] We assume that $\alpha_0 \in Q\Gamma^*$.

[5] We also assume that the initial state α_0 is a member of V.

Theorem 3.11. *Let $\mathcal{V} = (\mathcal{C}, V, \Sigma, P, \alpha_0, F)$ be sequential (resp. parallel) CRTS of type $1\frac{1}{2}$ or type 0. There exists sequential (resp. parallel) CRTS $\mathcal{V}'$ of the same type with trivial constraint system $\mathcal{C}_\varepsilon$, isomorphic to $\mathcal{V}$.*

Proof. The idea of the proof for CRTS of type $1\frac{1}{2}$ is based on the fact that every state of such a system has just one distinguished variable in the set Q corresponding to the state of (parallel) push-down automata. Thus the actual state of the store can be held as a part of such a variable.

Let $\mathcal{V}$ be the sequential (resp. parallel) CRTS of type $1\frac{1}{2}$ defined as $\mathcal{V} = (\mathcal{C}, Q \cup \Gamma, \Sigma, P, q_0\gamma_0, Q \times C^\diamond)$ where constraint system $\mathcal{C}$ is of the form $(C, \leq, \wedge, tt, ff)$, Q and Γ are disjoint sets, q_0 is a member of Q, and $\gamma \in \Gamma^*$. The new sequential (resp. parallel) CRTS $\mathcal{V}'$ of type $1\frac{1}{2}$, isomorphic to $\mathcal{V}$ is constructed as $\mathcal{V}' = (\mathcal{C}_\varepsilon, Q' \cup \Gamma, \Sigma, P', q_0^{(tt)}\gamma_0, Q' \times \{tt\})$, where $Q' = \{q^{(m)} \mid q \in Q,\ m \in C^\diamond\}$ is the set of variables which include the state of the store. In P' we replace every rewrite rule

$$(pA \xrightarrow{a} q\gamma, m, n) \in P$$

by the set of rules

$$(p^{(o)}A \xrightarrow{a} q^{(o\wedge n)}\gamma, tt, tt) \in P'$$

for every $o \in C^\diamond$ which satisfies the entailment condition $o \vdash m$ and the consistency condition $o \wedge n \neq ff$. In other words, entailment and consistency conditions are always satisfied in $\mathcal{V}'$, but the power of checking for these conditions is not lost, just moved to the new set of rules. The isomorphism of $\mathcal{V}$ and $\mathcal{V}'$ is obvious as every state $(q^{(m)}\gamma, tt)$ of $\mathcal{V}'$ corresponds exactly to the state $(q\gamma, m)$ of the system $\mathcal{V}$.

A bit different idea is used to prove the theorem for CRTS of type 0. There is no suitable variable to hold the state of the store, because in general in each state there is no variable which participates in every rewrite rule applicable to this state. But we can add such a variable artificially as CRTS of type 0 has no restriction on the rules and F (this is impossible for constrained rewrite transition systems of type 2 due to restrictions on the rules and on the set of final states).

Let $\mathcal{V} = (\mathcal{C}, V, \Sigma, P, \alpha_0, F)$ be a sequential (resp. parallel) CRTS of type 0 with the constraint system $\mathcal{C} = (C, \leq, \wedge, tt, ff)$. The new sequential (resp. parallel) CRTS $\mathcal{V}'$ of the same type and isomorphic to $\mathcal{V}$ is constructed as $\mathcal{V}' = (\mathcal{C}_\varepsilon, V', \Sigma, P', S^{(tt)}\alpha_0, F')$, where $V' = V \cup \{S^{(m)} \mid m \in C^\diamond\}$ is the set of variables (assuming $S \notin V$), $S^{(tt)}$ is the variable representing the initial state of the store, $F' = \{(S^{(m)}\beta, tt) \mid (\beta, m) \in F\}$. In P' we replace every rewrite rule

$$(\alpha \xrightarrow{a} \beta, m, n) \in P$$

by the set of rules

$$(S^{(o)}\alpha \xrightarrow{a} S^{(o\wedge n)}\beta, tt, tt) \in P'$$

for every $o \in C^{\diamond}$ which satisfies the entailment condition $o \vdash m$ and the consistency condition $o \wedge n \neq ff$. Again, entailment and consistency conditions are always satisfied in $\mathcal{V}'$ and the power of checking for these conditions is moved to the new set of rules. The isomorphism comes from the fact that every state $(S^{(m)}\alpha, tt)$ of the system $\mathcal{V}'$ corresponds exactly to the state (α, m) of $\mathcal{V}$. □

In general, it is impossible to find rewrite transition system of type 3 which defines transition system isomorphic to the one defined by CRTS of type 3. The reason is that rewrite transition systems of type 3 have always just one final state while CRTS of type 3 can have more then one final state. We decided not to define new class of finite-state automata with more final states as the difference is inconsiderable due to the fact that every final state is terminal and thus indistinguishable from other final states with respect to bisimulation equivalence.

Theorem 3.12. *Let $\mathcal{V} = (\mathcal{C}, V, \Sigma, P, \alpha_0, F)$ be CRTS of type 3. There exists CRTS $\mathcal{V}'$ of type 3 with trivial constraint system $\mathcal{C}_\varepsilon$, bisimilar to $\mathcal{V}$.*

Proof. Except the final states which are of the form (ε, m), each state of CRTS of type 3 consists of exactly one variable and one constraint. Thus the actual state of the store can be held as a part of such a variable.

Let $\mathcal{V} = (\mathcal{C}, V, \Sigma, P, \alpha_0, \{\varepsilon\} \times C^{\diamond})$ be a sequential (resp. parallel) CRTS of type 3 with the constraint system $\mathcal{C} = (C, \leq, \wedge, tt, ff)$. A new sequential (resp. parallel) CRTS $\mathcal{V}'$ of the same type bisimilar to $\mathcal{V}$ is constructed as $\mathcal{V}' = (\mathcal{C}_\varepsilon, V', \Sigma, P', \alpha_0^{(tt)}, \{(\varepsilon, tt)\})$, where $V' = \{A^{(m)} \mid A \in V,\ m \in C^{\diamond}\}$ is the set of variables which include the state of the store and $\alpha_0^{(tt)}$ is the initial variable holding the initial state of the store. In P' we replace every rewrite rule

$$(A \xrightarrow{a} B, m, n) \in P \ \text{ or } \ (A \xrightarrow{a} \varepsilon, m, n) \in P$$

by the set of rules

$$(A^{(o)} \xrightarrow{a} B^{(o\wedge n)}, tt, tt) \in P' \ \text{ or } \ (A^{(o)} \xrightarrow{a} \varepsilon, tt, tt) \in P'$$

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 original rules. The system $\mathcal{V}'$ is "almost isomorphic" to $\mathcal{V}$ in the sense, that each state $(A^{(m)}, tt)$ of $\mathcal{V}'$ corresponds to the state (A, m) of $\mathcal{V}$. But the isomorphism is corrupt due to the final state (ε, tt) of $\mathcal{V}'$ corresponding to all reachable final states (ε, m) of $\mathcal{V}$. As final states are terminal, systems $\mathcal{V}$ and $\mathcal{V}'$ are bisimilar. $\square$

Notice that last two proofs provide an idea of algorithms for converting CRTS of mentioned types to bisimilar rewrite transition systems of the same type.

Rewrite transition system of type 2 (corresponding to BPA resp. BPP) can change its expressibility with respect to bisimulation equivalence, when a finite constraint system is added. The new classes of transition systems defined by constrained rewrite transition systems of type 2, fcBPA and fcBPP, come in sight here and will be studied in more details in next sections.

3.5 Hierarchy

This subsection presents the hierarchy of classes of transition systems illustrated in Figure 1, extended by two new classes, fcBPA and fcBPP. The position of these classes in the hierarchy follows directly from the classification demonstrated in Subsection 3.4, except for the relation between BPA and fcBPA and between BPP and fcBPP, which follows from Lemma 3.7. The extended hierarchy is depicted in Figure 5.

The extended hierarchy keeps strict with respect to bisimulation equivalence. The proof can be found in the following sections dedicated to the new classes. It also will be proved that BPP is a strict subclass of fcBPP and fcBPP is a strict subclass of PPDA with respect to language equivalence. On the other hand, from the fact that $L(BPA) = L(PDA)$ it follows that the language expressibility of fcBPA is identical to the language expressibility of BPA and PDA.

4 The fcBPP Class

This section presents some basic facts around the class of the transition systems which can be defined by a parallel constrained rewrite transition system of type 2. The abbreviation *fcBPP* corresponds to BPP with finite constraint system.

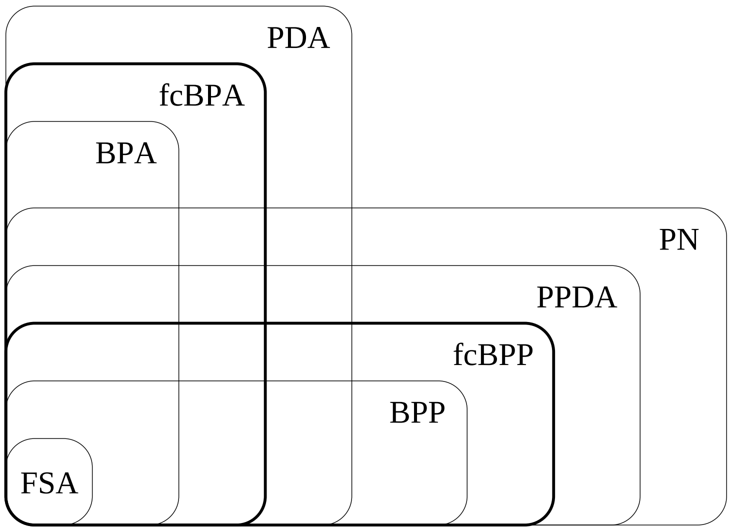

Figure 5: Hierarchy extended with the new classes fcBPA and fcBPP

4.1 BPP is a strict subclass of fcBPP

The class of fcBPP amplifies the power of BPP with poor mechanism of asynchronous communication via shared store. Although the store is monotonic and has only finite number of possible states, BPP is a strict subclass of fcBPP. We have already demonstrated that BPP is a subclass of fcBPP, the strictness follows from the example given below which offers a fcBPP transition system which is not in BPP class.

Example 4.1. *Let $\mathcal{V}$ be a parallel CRTS $(\mathcal{C}_{mn}, \{A, X\}, \{a, b, c\}, P, A, F)$ of type 2, where $\mathcal{C}_{mn}$ is the constraint system from Example 3.3, the set of final states $F = \{\varepsilon\} \times \{tt, m, n\}$ and P contains the following rules:*

$$\begin{array}{ll} (A \xrightarrow{a} AX, tt, tt) & (X \xrightarrow{b} \varepsilon, m, tt) \\ (A \xrightarrow{b} \varepsilon, tt, m) & (X \xrightarrow{c} \varepsilon, n, tt) \\ (A \xrightarrow{c} \varepsilon, tt, n) & \end{array}$$

Behavior of this system is represented in Figure 6.

Language generated by this system $L(\mathcal{V}) = \{a^{n-1}b^n, a^{n-1}c^n \mid n \geq 1\}$ cannot be generated by any BPP due to the Pumping Lemma presented by Christensen [Chr93] in the following form.

Lemma 4.2 (Pumping Lemma for BPP). *Let L be any language of $L(BPP)$. Then there exists a constant m such that if u is a word of L and $|u| > m$ then there exist $x, y, z \in \Sigma^*$ such that*

$$
\begin{tikzcd}
 & (\varepsilon, m) & (X, m) \arrow[l, "b"'] & (XX, m) \arrow[l, "b"'] & \cdots \arrow[l, "b"'] \\
 {} \arrow[r] & (A, tt) \arrow[u, "b"'] \arrow[d, "c"] \arrow[r, "a"] & (AX, tt) \arrow[u, "b"'] \arrow[d, "c"] \arrow[r, "a"] & (AXX, tt) \arrow[u, "b"'] \arrow[d, "c"] \arrow[r, "a"] & \cdots \\
 & (\varepsilon, n) & (X, n) \arrow[l, "c"'] & (XX, n) \arrow[l, "c"'] & \cdots \arrow[l, "c"']
\end{tikzcd}
$$

Figure 6: Transition system described in Example 4.1

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

In conclusion, we demonstrated that fcBPP has bigger expressibility power than usual BPP even with respect to language equivalence.

4.2 Pumping Lemma

The pumping lemma for fcBPP is formulated and proved in this subsection. The proof is similar to the one presented by Christensen for BPP case ([Chr93]) thanks to the fact that number of derivation steps which change the state of the store is bounded due to finiteness of constraint system.

Let $\mathcal{V} = (\mathcal{C}, V, \Sigma, P, \alpha_0, F)$ be a constrained rewrite transition system of type 2. For $X \in V$, let $R_{m,n}(X)$ denote the set

$$\{Y \in V \mid \exists \beta \in V^* : (X, m) \stackrel{+}{\Longrightarrow} (Y\beta, m) \text{ and } (\beta, n) \stackrel{*}{\Longrightarrow} (\varepsilon, n)\}^6,$$

i.e. the set of variables Y which can be derived from (X, m) without changes on the store and where string of variables β derived from (X, m) during the derivation of Y can be rewritten to ε without changing the store set to the constraint n. We extend this definition to multisets of variables in obvious manner, thus $R_{m,n}(A_1 A_2 \ldots A_j) = \bigcup_{i \in \{1,2,\ldots,j\}} R_{m,n}(A_i)$.

[6] The relation $\stackrel{*}{\Longrightarrow}$ (resp. $\stackrel{+}{\Longrightarrow}$) is apprehended as usual, i.e. $(\alpha, m) \stackrel{*}{\Longrightarrow} (\beta, n)$ (resp. $(\alpha, m) \stackrel{+}{\Longrightarrow} (\beta, n)$) iff there exists $w \in \Sigma^*$ (resp. $w \in \Sigma^+$) such that $(\alpha, m) \stackrel{w}{\Longrightarrow} (\beta, n)$.

Lemma 4.3. *Let $\mathcal{V} = (\mathcal{C}, V, \Sigma, P, \alpha_0, F)$ be parallel CRTS of type 2. If there exists some derivation of a word $u = u_1 u_2 \dots u_k \in L(\mathcal{V})$ of the form*

$$(\alpha_0, tt) = (\alpha_0, m_0) \stackrel{u_1}{\Longrightarrow} (\alpha_1, m_1) \stackrel{u_2}{\Longrightarrow} \dots \stackrel{u_k}{\Longrightarrow} (\alpha_k, m_k) = (\varepsilon, m_k)$$

such that $\forall i \in \{1, 2, \dots, k\}, \forall X \in \alpha_i$ it holds $X \notin R_{m_i, m_k}(X)$, then $|u| \leq h$, where h is a constant depending only on $\mathcal{V}$.

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

$$(\alpha_i, m_i) \stackrel{u_{i+1}}{\Longrightarrow} (\alpha_{i+1}, m_{i+1}) \stackrel{u_{i+2}}{\Longrightarrow} \dots \stackrel{u_{i+j}}{\Longrightarrow} (\alpha_{i+j}, m_{i+j}),$$

where the state of the store (in following marked as m) keeps unchanged ($m = m_i = m_{i+1} = \dots = m_{i+j}$) 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

$$(\beta_0 \gamma_0, m) \stackrel{v_1}{\Longrightarrow} (\beta_1 \gamma_1, m) \stackrel{v_2}{\Longrightarrow} \dots \stackrel{v_p}{\Longrightarrow} (\beta_p \gamma_p, m),$$

where $v_1, v_2, \dots, v_p \in \Sigma^+$, $\beta_0 \gamma_0 = \alpha_i$, in β_0 there are all variables from α_0 which are rewritten in derivation sequence $(\alpha_i, m) \stackrel{u'}{\Longrightarrow} (\alpha_{i+j}, m)$, and in γ_0 there are variables which do not actively participate in this derivation sequence. Now $\beta_l \gamma_l$ ($l = 1, 2, \dots, p$) rises from $\beta_{l-1} \gamma_{l-1}$ by one rewriting of each variable from β_{l-1} in the same way as this variable was rewritten in original flat derivation sequence (thus $|v_l| = |\beta_{l-1}|$) and still it holds that β_l contains variables, which are rewritten in the original flat derivation sequence, while γ_l contains the other variables (thus $\gamma_{l-1} \subseteq \gamma_l$). We finish rewriting when β_l is empty (thus $\beta_p = \varepsilon$ and $\gamma_p = \alpha_{i+j}$). It is clear that $v = v_1 v_2 \dots v_p$ is a permutation of u', especially $|v| = |u'|$. By replacing $(\alpha_i, m) \stackrel{u'}{\Longrightarrow} (\alpha_{i+j}, m)$ with $(\beta_0 \gamma_0, m) \stackrel{v}{\Longrightarrow} (\beta_p \gamma_p, m)$ in the original derivation we get correct derivation of the word $u_1 \dots u_i v u_{i+j+1} \dots u_n$ of length k. Further, for each X in β_l ($l = 1, 2, \dots, p$) there exists α_t ($i \leq t \leq i + j$) such that $X \in \alpha_t$.

Now we show that $R_{m,m_k}(\beta_{l-1}) \supsetneq R_{m,m_k}(\beta_l)$ for each $1 \leq l < p$.

"$\supseteq$" It comes directly from the fact that each variable from β_l has an ancestor in β_{l-1}.

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

Hence we have

$$|V| \geq |R_{m,m_k}(\beta_0)| > |R_{m,m_k}(\beta_1)| > \ldots > |R_{m,m_k}(\beta_{p-1})| > 0.$$

This implies $p \leq |V|$. Further, for each $1 \leq l \leq p$ it holds that

$$|v_l| = |\beta_{l-1}| \leq |\beta_0|s^{l-1} \leq |\beta_0|s^p \leq |\beta_0|s^{|V|},$$

where s is a maximum length of right sides of rewrite rules in P. Now we restrict length of u'

$$|u'| = |v| = \sum_{l=1}^{p} |v_l| \leq \sum_{l=1}^{p} |\beta_0|s^{|V|} = p|\beta_0|s^{|V|} \leq |V||\alpha_i|s^{|V|}.$$

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

$$|u'| \leq |\alpha_i|r,$$

where $r = |V|s^{|V|}$.

In general it holds that each sequence of derivation steps consists of non-flat steps and flat derivation sequences. Number of "unflat" steps $(\alpha_i, m_i) \overset{u_{i+1}}{\Longrightarrow} (\alpha_{i+1}, m_{i+1})$, 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

$$|u| \leq |C^\diamond| - 1 + \sum_{j=1}^{|C^\diamond|} |\alpha'_j|r,$$

where (α'_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 (α'_j, m'_j) is the first state in this derivation with the constraint m'_j in the store. Thus $(\alpha'_1, m'_1) = (\alpha_0, tt)$.

The last step is to restrict the length of α'_j for $j > 1$. We can deduce a restriction

$$|\alpha'_j| \leq |\alpha'_{j-1}| + (s-1)(|\alpha'_{j-1}|r + 1)$$

thanks to the facts that each application of rewrite rule cannot add more than $s-1$ variables to the string of variables 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.

$$\begin{array}{rcl} |\alpha'_j| & \leq & |\alpha'_{j-1}| + s(|\alpha'_{j-1}|r + 1) \\ |\alpha'_j| & \leq & |\alpha'_{j-1}|(1 + rs + s) \\ |\alpha'_j| & \leq & |\alpha'_1|(1 + rs + s)^{j-1} \\ |\alpha'_j| & \leq & |\alpha_0|(1 + rs + s)^{j-1} \end{array}$$

By summarization we get

$$|u| \leq |C^\diamond| - 1 + r|\alpha_0| \sum_{j=1}^{|C^\diamond|} (1 + rs + s)^{j-1},$$

where $r = |V|s^{|V|}$. The sum on the right side of previous inequality can be modified as it is an geometric progression. The final form of desired h is then

$$h = |C^\diamond| - 1 + r|\alpha_0| \frac{(1 + rs + s)^{|C^\diamond|} - 1}{rs + s},$$

where s is the maximum length of right sides of rewrite rules in P and $r = |V|s^{|V|}$. □

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

Lemma 4.4 (Pumping Lemma for fcBPP). *Let L be a language of L(fcBPP) Then there exists a constant h such that if u is a word of L and $|u| > h$ then there exist $x, y, z, w \in \Sigma^*$ such that*

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

Proof. Given L we have a parallel CRTS $\mathcal{V}$ of type 2 such that $L = L(\mathcal{V})$. It follows from Lemma 4.3 that each derivation

$$(\alpha_0, tt) = (\alpha_0, m_0) \xrightarrow{u_1} (\alpha_1, m_1) \xrightarrow{u_2} \ldots \xrightarrow{u_k} (\alpha_k, m_k) = (\varepsilon, m_k)$$

of the word $u = u_1 u_2 \ldots u_k \in L(\mathcal{V})$, $|u| > h$ contains some state $(\alpha_j, m_j) = (X\gamma, m_j)$, where $X \in R_{m_j, m_k}(X)$. The definition of $R_{m_j, m_k}(X)$ says that there exist $\beta \in V^*$ and $y, w \in \Sigma^*$ such that $|y| \geq 1$, $(X, m_j) \stackrel{y}{\Longrightarrow} (X\beta, m_j)$ and $(\beta, m_k) \stackrel{w}{\Longrightarrow} (\varepsilon, m_k)$. Now the derivation

$$(\alpha_0, tt) \stackrel{u_1 \ldots u_j}{\Longrightarrow} (\alpha_j, m_j) \stackrel{y^i}{\Longrightarrow} (\alpha_j \beta^i, m_j) \stackrel{u_{j+1} \ldots u_k}{\Longrightarrow} (\beta^i, m_k) \stackrel{w^i}{\Longrightarrow} (\varepsilon, m_k)$$

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

4.3 fcBPP is a strict subclass of PPDA

From the classification presented in Subsection 3.4 it follows that fcBPP is a subclass of PPDA.

The pumping lemma represents the powerful instrument which allows us to enunciate without any other arguments that the language $\{a^n bc^n de^n \mid n \geq 0\}$ generated by PPDA from Example 4.5 cannot be generated by any fcBPP.

Example 4.5. *Let $\mathcal{V} = (\{p, q, r, X, A, B\}, \{a, b, c\}, P, pX, \{p, q, r\})$ be a parallel rewrite system of type $1\frac{1}{2}$ (corresponding to PPDA), where the set P consists of the following rewrite rules:*

$$\begin{aligned} pX &\stackrel{a}{\longrightarrow} pXAB \\ pX &\stackrel{b}{\longrightarrow} qX \\ qA &\stackrel{c}{\longrightarrow} q \\ qX &\stackrel{d}{\longrightarrow} r \\ rB &\stackrel{e}{\longrightarrow} r \end{aligned}$$

Then $L(\mathcal{V}) = \{a^n bc^n de^n \mid n \geq 0\}$ is the language generated by system $\mathcal{V}$.

To sum up, we have demonstrated that fcBPP is a strict subclass of PPDA even with respect to language equivalence.

5 The fcBPA Class

This section is devoted to the class of the transition systems which can be defined by a sequential constrained rewrite transition system of type 2. The abbreviation *fcBPA* corresponds to BPA with finite constraint system.

5.1 BPA is a strict subclass of fcBPA

The fact that the BPA is a subclass of fcBPA follows from Lemma 3.7. The witness of the strictness can be found in Example 4.1. If we look at the CRTS described there as to a sequential CRTS, we get fcBPA with the same transition diagram as the one presented in Figure 6. This transition system is not bisimilar to any BPA as proven in [Mol96]. The Moller's argumentation is as follows.

Suppose that we have a BPA which represents this transition system up to bisimilarity, and let h be greater than the norm of any of its variables. Then the BPA state corresponding to (AX^h, tt) in Figure 6 must be of the form $B\beta$ where $B \in V$ and $\beta \in V^+$. But then any sequence of $norm(B)$ norm-reducing transition must lead to BPA state β, while the transition system on Figure 6 has two such non-bisimilar derived states, namely (X^k, m) and (X^k, n) where $k = norm(\beta)$.

Another example of fcBPA which is not bisimilar to any BPA can be found in [BCS96]. The transition system is described in following example.

Example 5.1. *Let $\mathcal{V}$ be a sequential CRTS $(\mathcal{C}_{mn}, \{A, X, Y\}, \{a, b, c, d\}, P, A, F)$ of type 2, where the constraint system $\mathcal{C}_{mn}$ is described in Example 3.3, the set of final states $F = \{\varepsilon\} \times \{tt, m, n\}$ and P contains the following rules:*

$$\begin{array}{ll} (A \xrightarrow{a} AX, tt, tt) & (X \xrightarrow{d} Y, m, tt) \\ (A \xrightarrow{c} \varepsilon, tt, m) & (Y \xrightarrow{d} \varepsilon, m, tt) \\ (A \xrightarrow{b} \varepsilon, tt, n) & (X \xrightarrow{d} \varepsilon, n, tt) \end{array}$$

Behavior of this system is represented in Figure 7.

The transition graph depicted in Figure 7 cannot be described by any BPA as its factor graph with respect to bisimulation equivalence is not regular ([CM90]) while the bisimulation collapse of each BPA graph is a regular graph ([BCS96]).

To conclude, we have demonstrated that fcBPA has greater expressibility power than classic BPA, but not on the language expressibility level as $L(BPA) = L(PDA)$ and fcBPA is a subclass of PDA. Thus we have $L(BPA) = L(fcBPA) = L(PDA)$.

5.2 fcBPA is a strict subclass of PDA

From the classification of transition systems presented in Subsection 3.4 follows that fcBPA is a subclass of PDA. A PDA transition system which

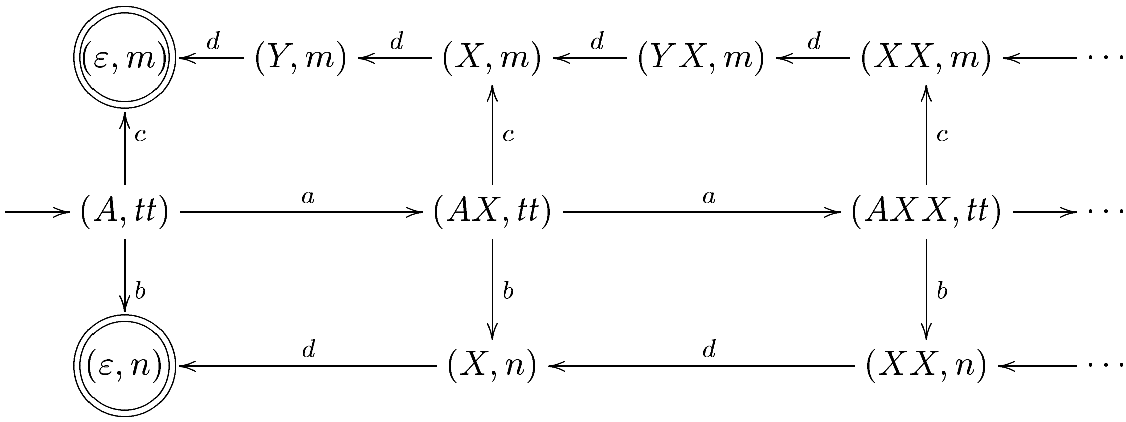

Figure 7: Transition system described in Example 5.1

cannot be described up to bisimilarity by any fcBPA is presented in Example 5.2.

Example 5.2. *The PDA given by the following rewrite rules and the initial state qX describes transition system represented in Figure 8.*

$$\begin{array}{lll} qX \xrightarrow{a} qAX & qX \xrightarrow{c} r & rX \xrightarrow{b} r \\ qA \xrightarrow{a} qAA & qA \xrightarrow{c} r & rA \xrightarrow{b} r \\ qA \xrightarrow{b} q & & \end{array}$$

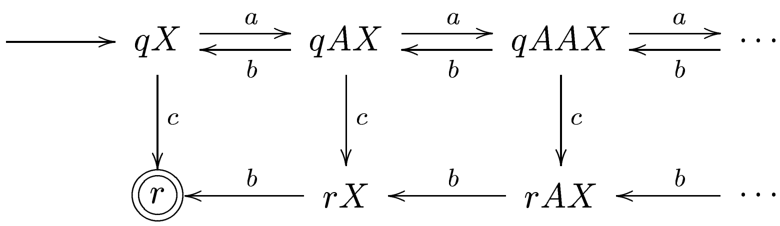

Figure 8: Transition system described in Example 5.2

To see that there is no fcBPA bisimilar to the transition system presented in Figure 8, suppose that we have such a fcBPA and let M be the set of all strings $\alpha \in V^*$ of its variables, such that α does not contain two occurrences of any variable. This condition implies that M is a finite set. Let s be greater than the norm of any reachable state (α, m), where $\alpha \in M$ and m is an arbitrary member of $C^\diamond$. Then the fcBPA state corresponding to qA^sX must be of the form (α, m) where $\alpha \notin M$. This means that there exists some $A \in$

V and $\beta, \gamma, \delta \in V^*$ such that $\alpha = \beta A \gamma A \delta$. Due to bisimilarity with qA^sX we can make transitions under b^* from $(\beta A \gamma A \delta, m)$ to the state $(A \gamma A \delta, n)$ and then we can make next transitions $(A \gamma A \delta, n) \stackrel{cb^*}{\Longrightarrow} (A\delta, o)$. The state $(A\delta, o)$ is bisimilar to the state rA^iX of PDA for appropriate i. The only one possible transition from the state rA^iX is the transition with label b. But in the set of possible transitions from the state $(A\delta, o)$ there is also the transition with label c corresponding to the rule used for the first transition of previous derivation sequence $(A \gamma A \delta, n) \stackrel{cb^*}{\Longrightarrow} (A\delta, o)$ as used rule is usable forever in CRTS.

To summarize, we have demonstrated that fcBPA is a strict subclass of PDA with respect to bisimulation equivalence.

6 Conclusion

We have enriched standard rewrite transition systems with the mechanism related to computing with partial information in the form used in widely studied Concurrent Constraint Programming. It has been proven that the enrichment of rewrite transition systems from classes FSA, PDA, PPDA, and PN with the finite constraint system does not change their expressibility with respect to bisimulation equivalence and except for FSA even with respect to graph isomorphism. It means that we open a more comfortable way how to describe transition systems of classes PDA, PPDA, and PN. On the contrary, the rewrite transition systems of classes BPA and BPP extended by finite constraint system establish two new classes, fcBPA and fcBPP, as the expressibility power of such systems increases. We have demonstrated that BPA is a strict subclass of fcBPA, BPP is a strict subclass of fcBPP and moreover that fcBPA is a strict subclass of PDA and fcBPP is a strict subclass of PPDA. We have also presented the Pumping Lemma for fcBPP.

6.1 Future research

Two topics for our future work are provided by the fcBPP class. The first one is an open question of decidability of bisimulation equivalence for fcBPP since the decidability of bisimulation equivalence for BPP has been already proven by Christensen, Hirshfeld and Moller [CHM93]. Using Jančar's method for proving undecidability of bisimulation equivalence for PN ([Jan95]) Moller [Mol96] has shown that bisimulation equivalence is undecidable for PPDA. The second interesting challenge is to specify the boundary of decidability of weak bisimulation equivalence with finite-state processes. Mayr has proved in [May96] that weak bisimulation equivalence with finite-state processes is decidable for BPP and Jančar, Kučera and Mayr [JKM98] have demonstrated undecidability of this problem for PPDA.

The next task is to extend similarly the transition systems of classes in Mayr's PRS-hierarchy [May97], very promising is the extension of the PA class. Totally different mission is to employ infinite constraint system.

Acknowledgements

First of all, I would like to thank Luboš Brim for his effective help and encouragement during my work on master thesis. My thanks go also to Mojmír Křetínský and Antonín Kučera for valuable discussions and their benign approach.

References

[BCS96] O. Burkart, D. Caucal, and B. Steffen. Bisimulation collapse and the process taxonomy. In Ugo Montanari and Vladimiro Sassone, editors, *CONCUR '96: 7th International Conference on Concurrency Theory*, volume 1119 of *Lecture Notes in Computer Science*, pages 247–262. Springer-Verlag, 1996.

[BK85] J. A. Bergstra and J. W. Klop. Algebra of communicating processes with abstraction. *Theoretical Computer Science*, 37:77–121, 1985.

[Cau92] D. Caucal. On the regular structure of prefix rewriting. *Theoretical Computer Science*, 106:61–86, 1992.

[CHM93] S. Christensen, Y. Hirshfeld, and F. Moller. Bisimulation equivalence is decidable for basic parallel processes. In Eike Best, editor, *CONCUR '93: 4th International Conference on Concurrency Theory*, volume 715 of *Lecture Notes in Computer Science*, pages 143–157. Springer-Verlag, 1993.

[Chr93] S. Christensen. *Decidability and Decomposition in Process Algebras*. PhD thesis, Department of Computer Science, University of Edinburgh, 1993.

[CM90] D. Caucal and R. Monfort. On the transition graphs of automata and grammars. In *WG '90: Graph-Theoretic Concepts in Computer Science*, volume 484 of *Lecture Notes in Computer Science*, pages 311–337. Springer-Verlag, 1990.

[dBP92] F. S. de Boer and C. Palamidessi. A process algebra of concurrent constraint programming. In Krzysztof Apt, editor, *JICSLP '92: Joint International Conference and Symposium on Logic Programming*, pages 463–477. MIT Press, 1992.

[HU79] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.

[Jan95] P. Jančar. Undecidability of bisimilarity for Petri nets and some related problems. *Theoretical Computer Science*, 148(2):281–301, 1995.

[JKM98] P. Jančar, A. Kučera, and R. Mayr. Deciding bisimulation-like equivalences with finite-state processes. *Lecture Notes in Computer Science*, 1443:200–211, 1998.

[May96] R. Mayr. Weak bisimulation and model checking for basic parallel processes. *Lecture Notes in Computer Science*, 1180:88–99, 1996.

[May97] R. Mayr. Process rewrite systems. *Electronic Notes in Theoretical Computer Science*, 7, 1997.

[Mil80] R. Milner. A calculus on communicating systems. *Lecture Notes in Computer Science*, 92, 1980.

[Mil89] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[Mol96] F. Moller. Infinite results. In Ugo Montanari and Vladimiro Sassone, editors, *CONCUR '96: 7th International Conference on Concurrency Theory*, volume 1119 of *Lecture Notes in Computer Science*, pages 195–216. Springer-Verlag, 1996.

[Mol98] F. Moller. A taxonomy of infinite state processes. *Electronic Notes in Theoretical Computer Science*, 18, 1998.

[Par81] D. M. R. Park. Concurrency and automata on infinite sequences. In Peter Deussen, editor, *Theoretical Computer Science: 5th GI-Conference*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183. Springer-Verlag, 1981.

[Sar89] V. A. Saraswat. *Concurrent Constraint Programming Languages*. PhD thesis, Computer Science Department, Carnegie Mellon University, 1989.

[SR90] V. A. Saraswat and M. Rinard. Concurrent constraint programming. In *Conference Record of the Seventeenth Annual ACM Symposium on Principles of Programming Languages*, pages 232–245. ACM Press, 1990.

[SRP91] V. A. Saraswat, M. Rinard, and P. Panangaden. Semantic foundations of concurrent constraint programming. In *Conference Record of the Eighteenth Annual ACM Symposium on Principles of Programming Languages*, pages 333–352. ACM Press, 1991.

[Sti95] C. Stirling. Local model checking games. In Insup Lee and Scott A. Smolka, editors, *CONCUR '95: 6th International Conference on Concurrency Theory*, volume 962 of *Lecture Notes in Computer Science*, pages 1–11. Springer-Verlag, 1995.

[vG90] R. J. van Glabbeek. The linear time-branching time spectrum. In J. C. M. Baeten and J. W. Klop, editors, *CONCUR '90: 1st International Conference on Concurrency Theory*, volume 458 of *Lecture Notes in Computer Science*, pages 278–297. Springer-Verlag, 1990.

Publications in the FI MU Report Series are in general accessible via WWW and anonymous FTP:

`http://www.fi.muni.cz/informatics/reports/`
`ftp  ftp.fi.muni.cz (cd pub/reports)`

Copies may be also obtained by contacting:

Faculty of Informatics
Masaryk University
Botanická 68a
602 00 Brno
Czech Republic