Information and Computation 209 (2011) 248–279

Contents lists available at ScienceDirect

Information and Computation

journal homepage: www.elsevier.com/locate/ic

# A classification of the expressive power of well-structured transition systems

Parosh Aziz Abdulla [a,*], Giorgio Delzanno [b], Laurent Van Begin [c]

[a] Uppsla University, Dept. Information Technology, Box 337, 751 05 Uppsala, Sweden
[b] Dipartimento di Informatica e Scienze dell'Informazione, Universita' di Genova, via Dodecaneso 35, 16146 Genova, Italy
[c] Département d'Informatique, Université Libre de Bruxelles, Boulevard du Triomphe - C.P. 212, 1050 Bruxelles, Belgium

ARTICLE INFO

*Article history:*
Available online 18 November 2010

*Keywords:*
Expressiveness
Well-structured transition systems
Language theory

ABSTRACT

We compare the expressive power of a class of well-structured transition systems that includes relational automata (extensions of), Petri nets, lossy channel systems, constrained multiset rewriting systems, and data nets. For each one of these models we study the class of languages generated by labeled transition systems describing their semantics. We consider here two types of accepting conditions: coverability and reachability of a fixed a priori configuration. In both cases we obtain a strict hierarchy in which constrained multiset rewriting systems is the most expressive model.



## 1. Introduction

The theory of well-structured transition systems [1,14] is a powerful tool for studying the decidability of verification problems of infinite-state systems. A system is well-structured when its transition relation is monotonic with respect to a well-quasi ordering defined over configurations. A well-known example of well-structured system is that of *Petri nets* [22] equipped with marking inclusion [1,14]. For a well-structured transition system, the *coverability problem* can be decided by the symbolic backward reachability algorithm scheme proposed in [1]. Since checking safety properties can be translated into instances of the coverability problem, an algorithm for coverability as proposed in [1] can be used for automatic verification of an infinite-state system. This connection has been exploited in order to develop automatic verification procedures for several infinite-state models:

- *relational automata* (RA) [8], an abstract models of imperative programs with integer valued variables;
- *reset/transfer nets* [11,12], i.e., Petri nets extended with *whole-place operations* that atomically operate on the whole set of tokens in a place;
- *lossy (FIFO) channel systems* (LCSs) [4,7], an abstract models of unreliable communication systems;
- *constrained multiset rewriting systems* (CMRS) [2], an extension of Petri nets in which tokens are colored with natural numbers and in which transitions are guarded by conditions on colors;
- *affine well-structured nets* (AWSNs) [19] a generalization of reset/transfer nets in which the firing of a transition is split into three steps: subtraction, multiplication, and addition of black tokens. Multiplication is a whole-place operation that generalizes transfer and reset arcs;
- *Data nets* [20], a generalization of AWSNs in which subtraction, multiplication and addition are defined on tokens that carry data taken from an infinite, ordered domain. Conditions on data values can be used here to restrict the type of

* Corresponding author.
*E-mail addresses:* parosh@it.uu.se (P.A. Abdulla), giorgio@disi.unige.it (G. Delzanno), lvbegin@ulb.ac.be (L. Van Begin).

0890-5401/$ - see front matter © 2010 Published by Elsevier Inc.
doi:10.1016/j.ic.2010.11.003

tokens on which apply whole-place operations. Data nets are a natural extension of CMRS with whole-place operations on colored tokens.

Although several efforts have been spent on studying the expressive power of extensions of Petri nets like reset and transfer nets [12,13,15], a comparison of the relative expressiveness of the class of well-structured transition systems is still missing. Such a comparison is a challenging research problem with a possible practical impact. Indeed, it can be useful to extend the applicability of a verification method (e.g., a particular instance of the scheme of [1]) to an entire class of models.

In this paper, we apply tools of language theory to formally compare the expressive power of a large class of well-structured infinite-state systems that includes extensions of Petri nets, constrained multiset rewriting systems, lossy channel systems, relational automata, and data nets. To achieve the goal, for each one of these models we study the class of languages generated by labeled transition systems describing their semantics. We consider here two types of accepting conditions: coverability (with respect to a fixed ordering) and reachability of a given configuration. Two models are considered to be equivalent if they generate the same class of languages.

For coverability accepting conditions, we obtain the following classification:

- We show that, differently from nets with indistinguishable tokens, whole-place operations do not augment the expressive power of models in which tokens carry data taken from an ordered domain. The proof is based on a weak, effectively constructible encoding of data nets into CMRS that can be used to reduce the coverability problem from one model to the other. As a corollary, we have that the symbolic backward reachability algorithm for solving the coverability problem in CMRS described in [2] can also be applied to data nets.
  As a second application of our CMRS encoding is the extension of decidability results on data nets. By slightly extending the CMRS encoding, we prove that the coverability problem remains decidable for different extensions of data nets. In particular we consider data net transitions that select data that *must* be fresh (in [20] a transition selects values that *may* be fresh).
- We prove that lossy channel systems are equivalent to a syntactic fragment of constrained multiset rewriting, we named $\Gamma_0$. The fragment $\Gamma_0$ is obtained by restricting conditions of a rule in such a way that equalities cannot be used as guards. Furthermore, we prove that lossy channel systems are strictly less expressive than the full model of constrained multiset rewriting systems. We then show that Petri nets are equivalent to a syntactic fragment of constrained multiset rewriting systems, we named $\Gamma_1$, obtained by considering nullary predicates only.
- We prove that AWSNs are strictly more expressive than Petri nets and strictly less expressive than LCSs, thus separating Petri nets from LCSs with respect to their relative expressive power. Furthermore, we prove that AWSNs are as expressive as transfer/reset nets. This result show that the inclusion between the coverability languages of transfer/reset nets and LCS proved in [3] is strict.
- We prove that relational automata are equivalent to a syntactic fragment of constrained multiset rewriting, we named $\Gamma_2$, obtained by imposing an upper bound on the size (number of predicates) of reachable configurations.
- Finally, we prove that $\Gamma_2$ generates the class of regular languages. This implies that relational automata are strictly less expressive than Petri nets.

For reachability accepting conditions, we obtain a slightly different classification. First, we prove that $\Gamma_0$ is equivalent to constrained multiset rewriting systems and *two counter machines*. Thus, with reachability acceptance, $\Gamma_0$ and constrained multiset rewriting systems turn out to be strictly more expressive than lossy channel systems. On the contrary, $\Gamma_1$ is still equivalent to Petri nets and strictly less expressive than $\Gamma_0$ and $\Gamma_2$ is still equivalent to relational automata and to finite automata. Finally, we show that lossy channel systems and Petri nets define incomparable classes of languages.

### *1.1. Related work*

The relative expressiveness of well-structured systems has been investigated for a limited number of extensions of Petri nets with reset, transfer, and non-blocking arcs in [13,15]. Classical results on finite and infinite languages generated by Petri nets can be found, e.g., in [16]. A classification of infinite-state systems in terms of structural properties and decidable verification problems is presented in [17]. The classification is extended to well-structured systems in [6]. A classification of the complexity of the decision procedures for coverability is studied in [20]. In contrast with the aforementioned work, we provide here a strict classification of the expressive power of several well-structured transition systems built with the help of tools of language theory.

### *1.2. Outline*

In Section 2, we give some preliminary notions on well-structured transition systems. In Section 3, we introduce constrained multiset rewriting systems. In Section 4, we give some first results on the class of languages accepted by CMRS. In Section 5, we recall data nets and compare the class of languages accepted by CMRS and data nets. In Section 6, 7, and 8, we compare the class of languages recognized by constrained multiset rewriting systems and, respectively, lossy channel systems (extensions of), Petri nets, and relational automata. Finally, in Section 9 we discuss some final remarks.

## 2. Wsts and languages with coverability acceptance

In this section, we recall some definitions taken from [1]. A *transition system* is a tuple $T = (S, R)$ where $S$ is a (possibly infinite) set of configurations, $R$ is a finite set of transitions where each $\xrightarrow{\sigma} \in R$ is a binary relation over $S$, i.e., $\xrightarrow{\sigma} \subseteq S \times S$. We use $\gamma \xrightarrow{\sigma} \gamma'$ to denote $(\gamma, \gamma') \in \xrightarrow{\sigma}$, and $\gamma \xrightarrow{\rho_1 \ldots \rho_k} \gamma'$ to denote that there exist $\gamma_1, \ldots, \gamma_{k-1}$ such that $\gamma \xrightarrow{\rho_1} \gamma_1 \cdots \xrightarrow{\rho_{k-1}} \gamma_{k-1} \xrightarrow{\rho_k} \gamma'$. Sometimes we will also use $\gamma \longrightarrow \gamma'$ to denote that there exists $\sigma \in R$ such that $\gamma \xrightarrow{\sigma} \gamma'$. A quasi ordering $(S, \preceq)$ is a *well-quasi ordering* if for any infinite sequence $s_1 s_2 \ldots s_i \ldots$ there exist indexes $i < j$ such that $s_i \preceq s_j$. A transition system $T = (S, R)$ is *well-structured* with respect to a quasi order $\preceq$ on $S$ iff:

(i) $\preceq$ is a well-quasi ordering;
(ii) for any $\xrightarrow{\sigma} \in R$ and $\gamma_1, \gamma_1', \gamma_2$ s.t. $\gamma_1 \preceq \gamma_1'$ and $\gamma_1 \xrightarrow{\sigma} \gamma_2$, there exists $\gamma_2'$ s.t. $\gamma_1' \xrightarrow{\sigma} \gamma_2'$ and $\gamma_2 \preceq \gamma_2'$, i.e., $T$ is *monotonic*.

We use $T = (S, R, \preceq)$ to indicate a *well-structured transition system* (wsts for short).

To formalize the comparison between models, a wsts $T = (S, R, \preceq)$ can be viewed as a language acceptor. For this purpose, we assume a finite alphabet $\Sigma$ and a labeling function $\lambda : R \mapsto \Sigma$ that associates to each transition of $R$ a symbol of $\Sigma \cup \{\epsilon\}$, where $\epsilon$ denotes the empty sequence ($w \cdot \epsilon = \epsilon \cdot w = w$ for any $w \in \Sigma^*$). In the following, we use $\gamma_1 \xrightarrow{w} \gamma_2$ with $w \in \Sigma^*$ to denote that $\gamma_1 \xrightarrow{\rho_1 \cdots \rho_k} \gamma_2$ and $\lambda(\xrightarrow{\rho_1}) \cdots \lambda(\xrightarrow{\rho_k}) = w$. Furthermore, we associate to $T$ an *initial* configuration $\gamma_{init} \in S$ and a *final* configuration $\gamma_{acc} \in S$ and assume an accepting relation $\bowtie: S \times S$. For a fixed accepting relation $\bowtie$, we define the language accepted (generated) by $T = (S, R, \preceq, \gamma_{init}, \gamma_{acc})$ as:

$$L(T) = \{w \in \Sigma^* | \gamma_{init} \xrightarrow{w} \gamma \text{ and } \gamma_{acc} \bowtie \gamma\}$$

In this paper, we consider two types of accepting relations:

- *Coverability*: the accepting relation $\bowtie_c$ is defined as $\preceq$.
- *Reachability*: the accepting relation $\bowtie_r$ is defined as $=$.

Let $\mathcal{M}$ be a wsts model (e.g., Petri nets) and let $T$ be one of its instances (i.e., a particular net). We define $L_c(T)$, resp. $L_r(T)$, as the language accepted by $T$ with accepting relation $\bowtie_c$, resp. $\bowtie_r$. We say that $L$ is a $c$-language, resp. $r$-language, of $\mathcal{M}$ if there is an instance $T$ of $\mathcal{M}$ such that $L = L_c(T)$, resp. $L = L_r(T)$. We use $L_c(\mathcal{M})$, resp. $L_r(\mathcal{M})$, to denote the class of $c$-languages, resp. $r$-languages, of $\mathcal{M}$. Finally, given two classes of languages $\mathcal{L}_1$ and $\mathcal{L}_2$, we use $\mathcal{L}_1 \not\sim \mathcal{L}_2$ to denote that $\mathcal{L}_1$ and $\mathcal{L}_2$ are incomparable classes.

Given a wsts $T = (S, R, \preceq)$ with labels in $\Sigma \cup \{\epsilon\}$, a *lossy* version of $T$ is a wsts $T' = (S, R', \preceq)$ for which there exists a bijection $h : R \mapsto R'$ such that $\xrightarrow{\rho} \in R$ and $\xrightarrow{h(\rho)}$ have the same label, $\xrightarrow{\rho} \subseteq \xrightarrow{h(\rho)}$ and if $\gamma \xrightarrow{h(\rho)} \gamma'$, then $\gamma \xrightarrow{\rho} \gamma''$ with $\gamma' \preceq \gamma''$. In other words, in a lossy version of a wsts the set of reachable configurations contains configurations that are smaller than those of the original model. The next lemma states an important property used in the remainder of the paper.

**Lemma 1.** *For any lossy version $T'$ of a wsts $T$, we have that $L_c(T) = L_c(T')$.*

## 3. Constrained multiset rewriting systems (CMRSs)

In this section, we recall the main definitions and prove the first results for *constrained multiset rewriting systems* [2]. Let us first give some preliminary definitions. We use $\mathbb{N}$ to denote the set of natural numbers (including 0) and $\overline{n}$ to denote the interval $[0, \ldots, n]$ for any $n \in \mathbb{N}$. We assume a set $\mathbb{V}$ of variables which range over $\mathbb{N}$, and a set $\mathbb{P}$ of unary predicate symbols. For a set $A$, we use $A^*$ and $A^{\otimes}$ to denote the sets of (finite) words and (finite) multisets over $A$, respectively. Sometimes, we write multisets as lists built using an associative–commutative constructor, so $[1, 5, 5, 1, 1]$ (equivalent to any of its permutations) represents a multiset with three occurrences of 1 and two occurrences of 5; $[\,]$ represents the empty multiset. We use the usual relations and operations such as $\leq$ (inclusion), $+$ (union), and $-$ (difference) on multisets. Given a finite set or a finite multiset $A$, we use $|A|$ to denote the cardinality of $A$. For a set $V \subseteq \mathbb{V}$, a *valuation Val* of $V$ is a mapping from $V$ to $\mathbb{N}$. A *condition* is a finite conjunction of *gap order* formulas of the forms: $x <_c y, x \leq y, x = y, x < c, x > c, x = c$, where $x, y \in \mathbb{V}$ and $c \in \mathbb{N}$. Here $x <_c y$ stands for $x + c < y$. We often use $x < y$ instead of $x <_0 y$. Sometimes, we treat a condition $\psi$ as a set, and write e.g., $(x <_c y) \in \psi$ to indicate that $x <_c y$ is one of the conjuncts in $\psi$. We use *true* to indicate an empty set of conditions. A *term* is of the form $p(x)$ where $p \in \mathbb{P}$ and $x \in \mathbb{V}$. A *ground term* is of the form $p(c)$ where $p \in \mathbb{P}$ and $c \in \mathbb{N}$. We sometimes say that a predicate symbol is *nullary* to mean that its parameter is not relevant (hence may be omitted).

A *constrained multiset rewriting system (CMRS)* $\mathcal{S}$ consists of a finite set of *rules* each of the form $L \rightsquigarrow R : \psi$, where $L$ and $R$ are multisets of terms, and $\psi$ is a condition. We assume that $\psi$ is consistent (otherwise, the rule is never enabled). For a valuation *Val*, we use $Val(\psi)$ to denote the result of substituting each variable $x$ in $\psi$ by $Val(x)$. We use $Val \models \psi$ to denote that $Val(\psi)$ evaluates to *true*. For a multiset $T$ of terms we define $Val(T)$ as the multiset of ground terms obtained from $T$ by replacing each variable $x$ by $Val(x)$. A *configuration* is a multiset of ground terms. Each rule $\rho = L \rightsquigarrow R : \psi \in \mathcal{S}$ defines a relation between configurations. More precisely, $\gamma \xrightarrow{\rho} \gamma'$ if and only if there is a valuation $Val$ s.t. the following conditions are satisfied: (*i*) $Val \models \psi$, (*ii*) $\gamma \geq Val(L)$, and (*iii*) $\gamma' = \gamma - Val(L) + Val(R)$.

**Example.** Consider the CMRS rule:

$$\rho = [p(x), q(y)] \rightsquigarrow [q(z), r(x), r(w)] \quad : \quad \{x + 2 < y\ ,\ x + 4 < z\ ,\ z < w\}$$

A valuation which satisfies the condition is $Val(x) = 1$, $Val(y) = 4$, $Val(z) = 8$, and $Val(w) = 10$.

A CMRS configuration is a multiset of ground terms, e.g., $[p(1), p(3), q(4)]$. Therefore, we have that $[p(1), p(3), q(4)] \xrightarrow{\rho} [p(3), q(8), r(1), r(10)]$.

Let us fix a CMRS $\mathcal{S}$ operating on a set of predicate symbols $\mathbb{P}$. Let *cmax* be the maximal constant which appears in the rules of $\mathcal{S}$; *cmax* is equal to 0 if there is no constant in $\mathcal{S}$. We now define an ordering $\preceq_c$ on configurations extracted from the ordering defined in [2] to solve the coverability problem.

**Definition.** Given a configuration $\gamma$, we define the *index* of $\gamma$, $index(\gamma)$, to be a word of the form $D_0 \cdots D_{cmax}\ d_0\ B_0\ d_1\ B_1\ d_2 \cdots d_n\ B_n$ where

- $D_0, \ldots, D_{cmax}, B_0, \ldots, B_n \in \mathbb{P}^{\otimes}$ and $d_0, d_1, \ldots, d_n \in \mathbb{N}\backslash\{0\}$;
- $B_i$ must not be empty for $0 \leq i \leq n$;
- for each $p \in \mathbb{P}$, $D_i$ contains $k$ occurrences of predicate $p$ iff $p(i)$ occurs $k$ times in $\gamma$ for $0 \leq i \leq cmax$;
- given $v_0 = cmax + d_0$, for each $p \in \mathbb{P}$, $B_0$ contains $k$ occurrences of predicate $p$ iff $p(v_0)$ occurs $k$ times in $\gamma$;
- given $v_{i+1} = v_i + d_{i+1}$, for each $p \in \mathbb{P}$, $B_{i+1}$ contains $k$ occurrences of predicate $p$ iff $p(v_{i+1})$ occurs $k$ times in $\gamma$ for all $0 \leq i < n$;
- for all $p(v) \in \gamma$ with $v > cmax$, there exists $i : 0 \leq i \leq n$ such that $v = cmax + d_0 + d_1 + \cdots + d_i$.

The ordering $\preceq_c$ is obtained by composing string embedding and multiset inclusion. The ordering $\preceq_c$ is defined as follows.

**Definition.** Let $D_0\ D_1 \cdots D_{cmax}\ d_0\ B_0\ d_1\ B_1\ d_2 \cdots d_n\ B_n$ be the index of a configuration $\gamma_1$ and $D'_0\ D'_1 \cdots D'_{cmax}\ d'_0\ B'_0\ d'_1\ B'_1\ d'_2 \cdots d'_m\ B'_m$ be the index of a configuration $\gamma_2$. Then, $\gamma_1 \preceq_c \gamma_2$ iff $D_i \leq D'_i$ for $0 \leq i \leq cmax$ and there exists a monotone injection $h : \overline{n} \mapsto \overline{m}$ such that $B_0 \leq B'_{h(0)}$, $B_i \leq B'_{h(i)}$, $d_0 \leq \sum_{k=0}^{h(0)} d'_k$, and $d_i \leq \sum_{k=h(i-1)+1}^{h(i)} d'_k$ for $1 \leq i \leq n$.

From standard properties of orderings, it follows that $\preceq_c$ is a well-quasi ordering. Furthermore, a CMRS is monotonic with respect to corresponding ordering $\preceq_c$. The following property then holds.

**Proposition 1** [2]. *A CMRS $\mathcal{S}$ equipped with $\preceq_c$ is well-structured.*

Finally, to simplify the presentation, we assume in the rest of the paper that the values appearing in the initial configuration $\gamma_{init}$ and in the accepting configuration $\gamma_{acc}$ are smaller or equal than *cmax* (to satisfy this condition we can add a rule that is never fireable and in which there is a constant greater than all values in $\gamma_{init} + \gamma_{acc}$). We also assume that the final configuration $\gamma_{fin} = [p_{fin}]$ contains only one nullary term $p_{fin}$.

### *3.1. A symbolic algorithm for testing coverability*

In this section, we give an overview of the algorithm for solving the coverability problem based on the generic backward analysis algorithm presented in [1]. The difficult challenge in applying this methodology is to invent a symbolic representation (called *constraints*) which allows effective implementation of each step, and which guarantees termination of the algorithm.

The algorithm operates on *constraints*, where each constraint $\phi$ characterizes an infinite set $[\![\phi]\!]$ of configurations. A *constraint* $\phi$ is of the form $T : \psi$ where $T$ is a multiset of terms and $\psi$ is a condition. The constraint characterizes the (upward closed) set $[\![\phi]\!] = \{\gamma \mid \exists Val.\ (Val \models \psi) \wedge (Val(T) \preceq_c \gamma)\}$ of configurations. Notice that if $\psi$ is inconsistent, then $[\![\phi]\!]$ is empty. Such a constraint can be safely discarded in the reachability algorithm presented below. Therefore, we assume in the sequel that all conditions in constraints are consistent. We define $Var(\phi) = Var(T) \cup Var(\psi)$. Observe that the coverability problem can be reduced to constraint reachability. More precisely, $\gamma_{init} \xrightarrow{*} [p_{fin}]$ is equivalent to $\gamma_{init} \xrightarrow{*} \gamma$ for some $\gamma \in [\![\phi_{fin}]\!]$ where $\phi_{fin}$ is the constraint $[p_{fin}(x)] : true$.

For constraints $\phi_1, \phi_2$, we use $\phi_1 \sqsubseteq \phi_2$ to denote that $\phi_1$ is *entailed* by $\phi_2$, i.e., $[\![\phi_1]\!] \supseteq [\![\phi_2]\!]$. For a constraint $\phi$, we define $Pre(\phi)$ to be a finite set of constraints which characterize the configurations from which we can reach a configuration in $\phi$ through the application of a single rule. In other words, $[\![Pre(\phi)]\!] = \{\gamma \mid \exists \gamma' \in [\![\phi]\!].\ \gamma \longrightarrow \gamma'\}$.

For instance, given $\phi_1 = [p(x_1), q(x_2), q(x_3)] : \{x_1 <_2 x_2, x_2 <_1 x_3\}$, and the configurations $\gamma_1 = [p(2), q(8), q(5), p(1)]$ and $\gamma_2 = [p(2), q(2), q(5), p(1)]$. Then $\gamma_1 \in [\![\phi_1]\!]$ and $\gamma_2 \notin [\![\phi_1]\!]$. Consider now $\phi_2 = [p(y_1), q(y_2)] : \{y_1 < y_2\}$ and $\phi_3 = [p(y_1), q(y_2)] : \{y_1 <_4 y_2\}$. Then $\phi_2 \sqsubseteq \phi_1$ and $\phi_3 \not\sqsubseteq \phi_1$.

Given an instance of the coverability problem, defined by $\gamma_{init}$ and the constraint $\phi_{fin}$ corresponding to $p_{fin}$, the symbolic algorithm performs a fixpoint iteration starting from $\phi_{fin}$ and repeatedly applying *Pre* on the generated constraints. The iteration stops if either (i) we generate a constraint $\phi$ with $\gamma_{init} \in [\![\phi]\!]$; or (ii) we reach a point where, for each newly generated constraint $\phi$, there is a constraint $\phi'$ generated in a previous iteration with $\phi' \sqsubseteq \phi$. We give a positive answer to the coverability problem in the first case, while we give a negative answer in the second case.

In [2] we show computability of membership, entailment, and define an effective predecessor operator for constraints. To give an idea of these definition, let $\mathcal{S}$ be a CMRS and $\phi_2$ be a constraint. We define $Pre(\phi_2) = \bigcup_{\rho\in\mathcal{S}} Pre_\rho(\phi_2)$, where $Pre_\rho(\phi_2)$ describes the effect of running the rule $\rho$ backwards from the configurations in $\phi_2$. Let $\rho = (L \rightsquigarrow R : \psi)$ and $\phi_2 = (T_2 : \psi_2)$. Let $W$ be any set of variables such that $|W| = |Var(\phi_2) \cup Var(\rho)|$. We define $Pre_\rho(\phi_2)$ to be the set of constraints of the form $T_1 : \psi_1$, such that there are renamings $Ren$, $Ren_2$ of $Var(\rho)$ and $Var(\phi_2)$, respectively, to $W$, and

- $T_1 = Ren_2(T_2) - Ren(R) + Ren(L)$
- $\psi_1 = Ren(\psi) \wedge Ren_2(\psi_2)$

**Example.** For instance, consider the constraint $\phi = [q(x_1), s(x_2), r(x_2)] : \{x_1 < x_2\}$ and the rule $\rho = [p(y_1), p(y_3)] \rightsquigarrow [q(y_2), r(y_3)] : \{y_3 < y_2\}$.
Fix $W = \{w_1, w_2, w_3, w_4, w_5\}$, and define $Ren_2 = (x_1 \mapsto w_1, x_2 \mapsto w_2)$, and $Ren = (y_1 \mapsto w_3, y_2 \mapsto w_1, y_3 \mapsto w_4)$.
Then one member of $Pre_\rho$ is given by
$[s(w_2), r(w_2), p(w_3), p(w_4)] : \{w_1 < w_2, w_4 < w_1\}$.

The termination of the algorithm is obtained by a non-trivial application of a methodology based on the theory of well- and better-quasi orderings described in [2].

### 3.2. Three interesting fragments of CMRS: $\Gamma_0$, $\Gamma_1$ and $\Gamma_2$

In this section, we defined three fragments of CMRS that we use as a technical tool for comparisons with other wsts.

**The fragment** $\Gamma_0$ In the fragment $\Gamma_0$ of CMRS every rule $L \rightsquigarrow R : \psi$ satisfies the following conditions: every variable $x$ occurs at most once in $L$ and at most once in $R$, and $\psi$ does not contain equality constraints. As an example, $[p(x), r(y)] \rightsquigarrow [q(x), r(z)] : x < y, y < z$ is a rule in $\Gamma_0$, whereas $[p(x), q(x)] \rightsquigarrow [q(y)] : true$ and $[p(x)] \rightsquigarrow [q(y), r(y)] : true$ are not in $\Gamma_0$.

**The fragment** $\Gamma_1$ The fragment $\Gamma_1$ is obtained by restricting CMRS to nullary predicates only (i.e., predicates with no parameters).

**The fragment** $\Gamma_2$ The fragment $\Gamma_2$ is the fragment of CMRS in which each rule $L \rightsquigarrow R : \psi$ satisfies the condition $|R| \le |L|$. In other words, in $\Gamma_2$ the cardinality of a reachable configuration is always bounded by the cardinality of the initial configuration.

In the rest of the paper we show that these three fragments have the same expressive power resp. as lossy FIFO channel systems, Petri nets, and Integral Relational Automata. To prove this statement, it is useful to isolate properties of CMRS and of these fragments with respect to coverability acceptance.

### 3.3. Properties of CMRS

In this section, we prove some properties of CMRS needed in the rest of the paper.

We first introduce some new terminology. We say that a configuration $\gamma$ with $index(\gamma) = D_0 \dots D_{cmax} B_0 d_0 \dots d_n B_n$ is *linear* if $B_i$ is a singleton multiset for $i : 0 \le i \le n$. We also say that an execution $\gamma_0 \xrightarrow{\rho_1} \cdots \xrightarrow{\rho_k} \gamma_k$ is linear whenever $\gamma_i$ is linear for any $i : 0 \le i \le k$.

Furthermore, we say that $\gamma$ is *cmax*-bounded if $index(\gamma) = D_0 \dots D_{cmax}$, i.e., all the natural numbers in $\gamma$ are between 0 and *cmax*.

An important property of CMRS is related to the possibility of lifting an execution from an initial *cmax*-bounded configuration $\gamma_{init}$ to a configuration $\gamma$ to a new execution leading from $\gamma_{init}$ to a configuration with larger "gaps" (for values greater than *cmax*) than those in $\gamma$.

We first define a restriction $\prec$ of the relation $\preceq_c$ in which we require that the distribution of predicates in two configurations has the same structure but larger gaps.
Formally, $\gamma_1 \prec \gamma_2$ holds iff the following conditions are satisfied:

- $index(\gamma_1) = D_0 \dots D_{cmax} d_0 B_0 d_1 \dots d_n B_n$,
- $index(\gamma_2) = D_0 \dots D_{cmax} d'_0 B_0 d'_1 \dots d'_n B_n$,
- $d'_i \ge d_i$ for any $i : 0 \le i \le n$.

We say that an execution $\gamma_0 \xrightarrow{\rho_1} \gamma_1 \cdots \xrightarrow{\rho_k} \gamma_k$ subsumes an execution $\gamma'_0 \xrightarrow{\rho'_1} \gamma'_1 \cdots \xrightarrow{\rho'_k} \gamma'_k$ if for all $i : 0 \leq i \leq k$, $\gamma'_i \prec \gamma_i$ and for all $i : 1 \leq i \leq k$, $\rho_i = \rho'_i$.

The following property then holds. The proof is in Appendix Appendix A.

**Proposition 2.** *Let a CMRS with initial cmax-bounded configuration $\gamma_{init}$. For any execution $e = \gamma_{init} \xrightarrow{\rho_1 \ldots \rho_k} \gamma$, for any configuration $\gamma'$ such that $\gamma \prec \gamma'$, there exists an execution $\gamma_{init} \xrightarrow{\rho_1 \ldots \rho_k} \gamma'$ that subsumes e.*

Now, we introduce the notion of linearization of a configuration. Linearization is used later in the paper to characterize the class of CMRS languages.

Given a configuration $\gamma$ with

$$index(\gamma) = D_0 \ldots D_{cmax} d_0 B_0 \ldots d_i B_i + [p] d_{i+1} B_{i+1} \ldots d_n B_n$$

where $B_i$ is not empty, we say that $\gamma'$ is a linearization of $\gamma$ if

$$index(\gamma') = D_0 \ldots D_{cmax} d'_0 B_0 \ldots d'_i B_i d[p] d'_{i+1} B_{i+1} \ldots d'_n B_n$$

such that

- $\forall 0 \leq j \leq n : d'_j \geq d_j$;
- $d \geq 1$.

The following lemmas then hold. The proof is in Appendix Appendix A.

**Lemma 2.** *Let $\mathcal{S}$ be a $\Gamma_0$ model with initial cmax-bounded configuration $\gamma_{init}$. Suppose there exists a linear execution $\gamma_{init} \xrightarrow{\rho_1 \ldots \rho_k} \gamma_k$, $\gamma_k \xrightarrow{\rho} \gamma$ and $\gamma$ is not linear. Then, there exists a (possibly different) linear execution $\gamma_{init} \xrightarrow{\rho_1 \ldots \rho_k} \gamma'_k$ such that $\gamma'_k \xrightarrow{\rho} \gamma'$ and $\gamma'$ is a linearization of $\gamma$.*

**Lemma 3.** *For a $\Gamma_0$ model $\mathcal{S}$, let $\gamma_1$ and $\gamma_2$ be two configurations such that $\gamma_2$ is a linearization of $\gamma_1$, $\gamma_1 \xrightarrow{\rho_1 \ldots \rho_k} \gamma_3$ implies there exists $\gamma_4$ such that $\gamma_2 \xrightarrow{\rho_1 \ldots \rho_k} \gamma_4$.*

Given a CMRS $\mathcal{S}$ of $\Gamma_0$ with initial *cmax*-bounded linear configuration $\gamma_{init}$ and accepting *cmax*-bounded linear configuration $\gamma_{acc}$, we define $L^{lin}_c(\mathcal{S})$ as the set

$$\{w| \textit{ there is a linear execution } \gamma_{init} \xrightarrow{\rho_1 \ldots \rho_k} \gamma_k \; s.t. \gamma_{acc} \preceq_c \gamma_k, \lambda(\rho_1) \cdots \lambda(\rho_k) = w\}$$

From Lemma 2 and Lemma 3, we obtain the following proposition.

**Proposition 3.** *For all CMRS $\mathcal{S}$ of $\Gamma_0$ with an initial cmax-bounded linear configuration $\gamma_{init}$ and cmax-bounded linear accepting configuration $\gamma_{acc}$, we have $L_c(\mathcal{S}) = L^{lin}_c(\mathcal{S})$.*

**Proof**

$\supseteq$: Immediate.

$\subseteq$: To simplify the presentation, let us assume that $\forall L \rightsquigarrow R : \psi \in \mathcal{S}$ : for each variable $x$ that appears in $L + R$ : either $(x = c) \in \psi$ $(0 \leq c \leq cmax)$ or $(x > cmax) \in \psi$. This assumption implies that the effect of a rule $\rho$ is constant if we only consider ground terms $p(x)$ with $x : 0 \leq x \leq cmax$.

Suppose that $\gamma_{init} \xrightarrow{\rho_1} \gamma_1 \xrightarrow{\rho_2} \cdots \xrightarrow{\rho_k} \gamma_k$ with $\gamma_{acc} \preceq_c \gamma_k$.

Now suppose that $\gamma_1$ is not linear. Applying Lemma 2, we have a linearization $\gamma'_1$ of $\gamma_1$ such that $\gamma_{init} \xrightarrow{\rho_1} \gamma'_1$. Furthermore, following Lemma 3, we have $\gamma'_1 \xrightarrow{\rho_2 \ldots \rho_k} \gamma'_k$. Iterating the reasoning, we obtain a linear configuration $\gamma''_1$ such that $\gamma_{init} \xrightarrow{\rho_1} \gamma''_1$ and $\gamma''_1 \xrightarrow{\rho_2 \ldots \rho_k} \gamma''_k$.

Repeating the reasoning for the other intermediate configurations, we conclude that there exists a linear execution $\gamma_{init} \xrightarrow{\rho_1 \ldots \rho_k} \gamma'''_k$.

From our hypothesis, $\rho_1 \ldots \rho_k$ has constant effect if we only consider ground terms $p(x)$ with $x : 0 \leq x \leq cmax$. Hence, we have that

$$\sum_{p(n) \in \gamma_k, 0 \leq n \leq cmax} [p(n)] = \sum_{p(n) \in \gamma'''_k, 0 \leq n \leq cmax} [p(n)]$$

and $\gamma_{acc} \preceq_c \gamma_k$ implies that $\gamma_{acc} \preceq_c \gamma_k'''$ since $\gamma_{acc}$ is *cmax*-bounded. We conclude that $\lambda(\rho_1)\cdots\lambda(\rho_k) \in L_c(\mathcal{S})$ implies $\lambda(\rho_1)\cdots\lambda(\rho_k) \in L_c^{lin}(\mathcal{S})$. $\square$

## 4. Expressive power of CMRS

We are now ready to give a first characterization for the expressive power of CMRS. In [15, Proposition 4], the authors show that there exists a recursively enumerable (RE) language that cannot be recognized by any wsts with coverability acceptance. Hence, the following proposition holds.

**Theorem 1.** $L_c(CMRS) \subset RE.$

With reachability as accepting condition, CMRS recognize instead the class of recursively enumerable languages (RE).

**Theorem 2.** $L_r(CMRS) = RE.$

**Proof.** We prove that CMRS can weakly simulate 2-counter machines. In the proof we also show that repeated reachability is undecidable for CMRS. We recall the model of a *2-counter machine* (*CM*) which is pair $(Q, \delta)$, where $Q$ is a finite set of states, and $\delta$ is the transition function. A transition is of the form $(q_1, op, q_2)$, where $q_1, q_2 \in Q$, and *op* is either an *increment* (of the form $cnt_1 ++$ or $cnt_2 ++$); a *decrement* (of the form $cnt_1 --$ or $cnt_2 --$); or a *zero-testing* (of the form $cnt_1 = 0?$ or $cnt_2 = 0?$). Operations and tests on counters have their usual semantics, assuming that the values of counters are natural values. In particular, decrement on a counter equal to zero is blocking. A 2-counter machine accepts an execution if it ends into the state $q_{fin}$. A *lossy 2-counter machine* (*LCM*) is of the same form as a counter machine. The difference in semantics is in the zero-testing operation. More precisely, the zero-testing of $cnt_1$ is simulated by resetting the value $cnt_1$ to zero, and decreasing the value of $cnt_2$ by an arbitrary natural number (possibly 0). The zero-testing of $cnt_2$ is performed in a similar manner.

Assume an LCM $\mathcal{M} = (Q, \delta)$. We shall construct a CMRS $\mathcal{S}$ which simulates $\mathcal{M}$. The simulation of $\mathcal{M}$ occurs in a sequence of *phases*. During each phase, $\mathcal{S}$ simulates increment and decrement transitions of $\mathcal{M}$. Each phase is indexed by a natural number which is incremented at the end of the phase. As soon as $\mathcal{M}$ performs a zero-testing of a counter, $\mathcal{S}$ enters an intermediate stage. After conclusion of the intermediate stage, a new phase is started and the index phase is increased.

The set of predicates symbols in $\mathcal{S}$ is divided into three groups:

- Two nullary predicate symbols $q$ and $q'$ for each $q \in Q$. We use $q'$ during the intermediate stages of the simulation.
- Two predicate symbols $cnt_1$ and $cnt_2$, which encode the values of $cnt_1$ and $cnt_2$, respectively.
- A predicate *phase* whose argument carries the index of the current phase. Furthermore, we use a predicate symbol *phase*$'$ to store the index of the previous phase during the intermediate stages of the simulation.

A configuration of $\mathcal{S}$ contains, during a given phase of the simulation, the following ground terms:

- A term of the form $q$ which encodes the current state of $\mathcal{M}$.
- A term of the form $phase(c)$ where $c$ is the index of the current phase.
- Terms of the form $cnt_1(c)$ where $c$ is the index of the current phase. The number of such terms encodes the current value of $cnt_1$. There are also a number of terms of the form $cnt_1(d)$ where $d$ is strictly lesser than the index of the current phase. Such terms are redundant and do not affect the encoding. Similar terms exist to encode $cnt_2$.

W.l.o.g., assume that the initial configuration $I_0$ of the 2-counter machine has control state $q_0$ and both counters equal to zero. The $\mathcal{S}$ configuration that encodes it is defined then as

$$\gamma_{init} = \left[q_0, phase(0), phase'(0)\right]$$

where *phase*$'$ is an auxiliary predicate needed to simulate a reset. If $cnt_i$ is initially equal to $k_i$, then we simply add to $\gamma_0$ $k_i$ occurrences of term $cnt_i(0)$ for $i : 1, 2$. For instance, if in $I_0$ $cnt_1 = 1$ and $cnt_2 = 2$, then

$$\gamma_{init} = [q_0, phase(0), cnt_1(0), cnt_2(0), cnt_2(0)]$$

An increment transition $(q, cnt_1 ++, q_2) \in \delta$ labeled with $a$ is simulated by a rule labeled with $a$ of the form

$$[q_1, phase(x)] \;\rightsquigarrow\; [q_2, phase(x), cnt_1(x)] \quad : \quad true$$

We increase the value counter $cnt_1$ by adding one more term whose predicate symbol is $cnt_1$ and whose argument is equal to the index of the current phase.

A decrement transition $(q, cnt_1 --, q_2) \in \delta$ labeled with $a$ is simulated by a rule labeled with $a$ of the form

$$[q_1, phase(x), cnt_1(x)] \rightsquigarrow [q_2, phase(x)] \quad : \quad true$$

We decrease the value counter $cnt_1$ by removing one of the corresponding terms from the configuration. Observe that terms whose arguments are less than the index of the current phase are not used, and hence they do not affect the encoding.

A transition $(q, cnt_1 = 0?, q_2) \in \delta$ labeled by $a$ is simulated by the following three rules (the two first are labeled with $\epsilon$ and the last one with $a$):

$$\begin{aligned}
&[q_1, phase(x), phase'(x)] \rightsquigarrow [q_1', phase(y), phase'(x)] \quad : \quad x < y\\
&[q_1', cnt_2(x), phase(y), phase'(x)]\\
&\qquad \rightsquigarrow [q_1', cnt_2(y), phase(y), phase'(x)] \quad : \quad true\\
&[q_1', phase(y), phase'(x)] \rightsquigarrow [q_2, phase(y), phase'(y)] \quad : \quad true
\end{aligned}$$

We enter the intermediate phase by changing from $q_1$ to $q_1'$. We store the current index using $phase'$, and generate a new index which is strictly larger than the current one. This resets counter $cnt_1$ since all terms in its encoding now have too small arguments. Finally, we change the arguments of (some of) the terms encoding $cnt_2$ to the new phase. Here, not all such terms may receive new arguments, and hence the value $cnt_2$ may "unintentionally" be reduced. We use *redundant terms* to refer to terms which have either $cnt_1$ or $cnt_2$ as predicate symbol, and whose arguments are smaller than the current index.

Mayr shows in [21] undecidability of the *repeated state reachability* problem for LCM, a decision problem defined as follows: Given a lossy counter machine and two states $q_{init}$ and $q_{fin}$, check whether there is a computation starting from $q_{init}$ (with both counter values being equal to zero) that visits $q_{fin}$ infinitely often.

We can extend the proof to show Theorem 2 as follows. The key observation here is that redundant terms are not removed during the simulation procedure described above. As a consequence, any reachable configuration which does not contain redundant terms corresponds to a state in a computation of a perfect (i.e., non-lossy) counter machine. We add the nullary predicate $p_{fin}$ and the following rules labeled by $\epsilon$ to our CMRS:

$$\begin{aligned}
[q_{fin}] &\rightsquigarrow [p_{fin}] &&: \quad true\\
[p_{fin}, phase(x), cnt_1(x)] &\rightsquigarrow [p_{fin}, phase(x)] &&: \quad true\\
[p_{fin}, phase(x), cnt_2(x)] &\rightsquigarrow [p_{fin}, phase(x)] &&: \quad true\\
[p_{fin}, phase(x)] &\rightsquigarrow [p_{fin}] &&: \quad true
\end{aligned}$$

In other words, if we reach a configuration where $\mathcal{M}$ is in $q_{fin}$, we first move to $p_{fin}$. Then, we start erasing ground terms that encode the value of the counters such that their argument correspond to that of the current *phase* predicate. This way, redundant ground terms (i.e., with arguments corresponding to previous phases) are not erased. This implies that there exists an execution where $\mathcal{S}$ recognizes a word $w$ that reaches $[p_{fin}]$ (i.e., with no redundant terms) iff there exists an execution where a non-lossy 2-counter machine recognizes the word $w$ that reaches $q_{fin}$. $\square$

## 5. Data nets

*Data nets* [20] are an extension of Petri nets in which tokens are colored with data taken from an infinite domain $D$ equipped with a linear and dense ordering $\prec$. Due to lack of space, we present here only the key concepts needed in the rest of the paper (see [20] for formal definitions). A data net consists of a finite set of places $P$ and of a finite set of transitions. A data net marking $s$ is a multiset of tokens that carry data in $D$. Formally, a marking $s$ is a finite sequence of vectors in $\mathbb{N}^P \backslash \{\vec{0}\}$, where $\vec{0}$ is the vector that contains only 0's. Each index $i$ in the sequence $s$ corresponds to some $d_i \in D$ such that $i \leq j$ if and only if $d_i \prec d_j$. For each $p \in P$, $s(i)(p)$ is the number of tokens with data $d_i$ in place $p$.

First of all, a data net transition $t$ has an associated arity $\alpha_t$ (a natural number greater than zero). The arity $\alpha_t = k$ is used to non-deterministically select $k$ distinct data $d_1 \prec \ldots \prec d_k$ from the current configuration $s$. Some of the selected data may not occur in $s$ (they are fresh). This choice induces a finite and ordered partitioning of the data in $s$, namely $R(\alpha_t) = (R_0, S_1, R_1, \ldots, S_k, R_k)$, where $R_0$ contains all data $d : d \prec d_1$ in $s$, $S_i = \{d_i\}$ for $i : 1, \ldots, k$, $R_i$ contains all $d : d_i \prec d \prec d_{i+1}$ in $s$ for $i : 1, \ldots, k-1$, and $R_k$ contains all $d : d_k \prec d$ in $s$. Clearly, $R(\alpha_t)$ also induces a natural partitioning of the multiset of tokens in $s$ based on the attached data.

For any $k \geq 1$, let $\overline{k}_0 = \{1, \ldots, k\}$. A transition $t$ operates on the regions in the partitioning $R(\alpha_t)$ in three steps defined resp. by three matrices $F_t, H_t \in \mathbb{N}^{R(\alpha_t) \times P}$, and $G_t \in \mathbb{N}^{R(\alpha_t) \times P \times R(\alpha_t) \times P}$.

$$F_t = \left( \begin{array}{cc|cc|cc} \multicolumn{2}{c}{R_0} & \multicolumn{2}{c}{S_1} & \multicolumn{2}{c}{R_1} \\ p & q & p & q & p & q \\ 0 & 0 & 1 & 0 & 0 & 0 \end{array} \right)$$

$$H_t = \left( \begin{array}{cc|cc|cc} \multicolumn{2}{c}{R_0} & \multicolumn{2}{c}{S_1} & \multicolumn{2}{c}{R_1} \\ p & q & p & q & p & q \\ 0 & 0 & 0 & 1 & 0 & 0 \end{array} \right)$$

$$G_t = \left( \begin{array}{cc|cc|cc|cc} & & \multicolumn{2}{c}{R_0} & \multicolumn{2}{c}{S_1} & \multicolumn{2}{c}{R_1} \\ & & p & q & p & q & p & q \\ \hline R_0 & p & 1 & 0 & 3 & 0 & 0 & 0 \\ & q & 3 & 1 & 0 & 0 & 0 & 0 \\ \hline S_1 & p & 0 & 0 & 1 & 0 & 0 & 0 \\ & q & 2 & 0 & 0 & 0 & 0 & 0 \\ \hline R_1 & p & 0 & 0 & 0 & 0 & 1 & 0 \\ & q & 0 & 0 & 0 & 0 & 0 & 1 \end{array} \right)$$

**Fig. 1**. A data net transition with arity $\alpha_t = 1$.

(1) *Subtraction:* $F_t$ specifies the number of tokens with data $d_1, \ldots, d_k$ that has to be removed from $s$. By definition, $F_t(R_i, p) = 0$ for $i \in \overline{k}$ and $p \in P$. The transition $t$ is enabled if the subtraction is possible on each place $P$. This step yields an intermediate configuration $s_1$ defined as follows:
For each $i \in \overline{k}_0$ and $p \in P$, $s_1(d_i)(p) = s(d_i)(p) - F_t(S_i, p)$.
For each $j \in \overline{k}$, $d \in R_j$, $p \in P$, $s_1(d)(p) = s(d)(p)$.

(2) *Multiplication:* $G_t$ specifies how many tokens are transferred from one place to another with possible multiplication of their occurrences and modification of their data ($G_t(\pi, p, \pi', p') \geq 0$ and, by definition, $G_t(R_i, p, R_j, q) = 0$ for any $i \neq j \in \overline{k}$ and any $p, q \in P$). This step yields an intermediate configuration $s_2$ defined as follows:
For each $i \in \overline{k}_0$ and $p \in P$:

$$s_2(d_i)(p) = \sum_{j \in \overline{k}_0} \sum_{q \in P} s_1(d_j)(q) \cdot G_t(S_j, q, S_i, p) + \sum_{j \in \overline{k}} \sum_{d \in R_j} \sum_{q \in P} s_1(d)(q) \cdot G_t(R_j, q, S_i, p)$$

For each $i \in \overline{k}$, $d \in R_i$, and $p \in P$:

$$s_2(d)(p) = \sum_{j \in \overline{k}_0, q \in P} s_1(d_j)(q) \cdot G_t(S_j, q, R_i, p) + \sum_{q \in P} s_1(d)(q) \cdot G_t(R_i, q, R_i, p)$$

Notice that transfers of tokens from region $R_i$ to region $R_j$ with $i \neq j$ are forbidden.

(3) *Addition:* Finally, $H_t$ specifies the number of tokens that are added to each place in $P$. Its application yields the successor configuration $s'$ such that:
For each $i \in \overline{k}_0$ and $p \in P$, $s'(d_i)(p) = s_2(d_i)(p) + H_t(S_i, p)$.
For each $j \in \overline{k}$, $d \in R_j$, $p \in P$, $s'(d)(p) = s(d)(p) + H_t(R_j, p)$.

As proved in [20], data nets are well-structured with respect to the well-quasi ordering $\preceq_d$ defined on markings as follows. Let $Data(s)$ be the set of data values that occur in a marking $s$. Then, $s_1 \preceq_d s_2$ iff there exists an injective function $h : Data(s_1) \mapsto Data(s_2)$ such that $(i)$ $h$ is monotonic and $(ii)$ $s_1(d)(p) \leq s_2(h(d))(p)$ for each $d \in Data(s_1)$ and $p \in P$.

**Example.** Consider a data net with $P = \{p, q\}$ and the transition in Fig. 1.
For a generic configuration $s$, the new configuration $s'$ is such that:

- $s'(d_1)(p) = s(d_1)(p) - 1 + \Sigma_{d \in R_0} 3 * s(d)(p)$ and $s'(d_1)(q) = 1$.
- For each $d \prec d_1$, $s'(d)(p) = s(d)(p) + 3 * s(d)(q) + 2 * s(d_1)(q)$ and $s'(d)(q) = s(d)(q)$.
- For each $d \succ d_1$, $s'(d)(p) = s(d)(p)$ and $s'(d)(q) = s(d)(q)$.

For instance, let $e_1 \prec e_2 \prec e_3 \prec e_4 \in D$ and assume that the transition selects $e_3$ as index in $S_1$, then:

$$\left(\begin{array}{cc|cc|cc|cc} \multicolumn{2}{c}{e_1} & \multicolumn{2}{c}{e_2} & \multicolumn{2}{c}{\boxed{e_3}} & \multicolumn{2}{c}{e_4} \\ p & q & p & q & p & q & p & q \\ 3 & 2 & 5 & 1 & 2 & 10 & 2 & 2 \end{array}\right) \to$$

$$\to \left(\begin{array}{cc|cc|cc|cc} \multicolumn{2}{c}{e_1} & \multicolumn{2}{c}{e_2} & \multicolumn{2}{c}{\boxed{e_3}} & \multicolumn{2}{c}{e_4} \\ p & q & p & q & p & q & p & q \\ 29 & 2 & 28 & 1 & 25 & 1 & 2 & 2 \end{array}\right)$$

### *5.1. Data nets vs CMRS*

In [20] the authors mention that it is possible to define an encoding of CMRS in the fragment of data net without whole place operations (*Petri data nets*) that preserves coverability. From this observation, it follows that $L_c(\mathit{CMRS}) \subseteq L_c(\mathit{data\ nets})$.

In this section, we tighten this relation and show that for each data net $\mathcal{D}$ we can effectively build a CMRS $\mathcal{S}$ such that $L_c(\mathcal{S}) = L_c(\mathcal{D})$. In the following, given a multiset $M$ with symbols in $P$ and a value or variable $x$, we use $M^x$ to denote the multi set of $P$-terms such that $M^x(p(x)) = M(p)$ (= number of occurrences of $p$ in $M$) for each $p \in P$, and $M^x(p(y)) = 0$ for any $y \neq x$ and $p \in P$.

#### *5.1.1. Configurations*

Assume an initial data net marking $s_0$ with data $d_1 \prec \ldots \prec d_n$. We build a CMRS representation of $s_0$ by non-deterministically selecting $n$ natural numbers $v_1 < \ldots < v_n$ strictly included in some interval $[f, l]$. $P$-terms with parameter $v_i$ represent tokens with data $d_i$ in place $p$. Formally, we generate the representation of $s_0$ by adding to $\mathcal{S}$ a rule labeled with $\epsilon$ that rewrites an initial nullary term *init* as follows[1]:

$$[\mathit{init}] \rightsquigarrow [\mathit{first}(f), \mathit{last}(l)] + \textstyle\sum_{i:1,\ldots,n} M_i^{x_i} \ : \ f < x_1 < \ldots < x_n < l \qquad (\mathit{init})$$

where $M_i$ is the multiset $s_0(d_i)$ for each $i \in \overline{n}_0$. The non-determinism in the choice of $f, l, x_1, \ldots, x_n$ make the CMRS representation of $s_0$ independent from specific parameters assumed by terms.

Transitions are encoded by CMRS rules that operate on the values in $[f, l]$ used in the representation of a marking. Most of the CMRS rule are based on left-to-right traversals of $P$-terms with parameters in $[f, l]$.

#### *5.1.2. Subtraction*

Consider a transition $t$ with $\alpha_t = k$. We first define a (silent) CMRS-rule that implements the subtraction step of $t$:

$$\begin{array}{ll} [\mathit{first}(f), \mathit{last}(l)] + F_t(S_1)^{x_1} + \cdots + F_t(S_k)^{x_k} \ \rightsquigarrow & \qquad (\mathit{subtract}) \\ \quad [\iota_0(f), \iota_1(x_1), \ldots., \iota_k(x_k), \iota_{k+1}(l), \mathit{new}_t] \ : \ f < x_1 < \ldots < x_k < l & \end{array}$$

In the *subtract* rule we non-deterministically associate a value $x_i$ to region $S_i$. The selection is performed by removing (from the current configuration) the multiset $F_t(S_i)^{x_i}$ that contains $F_t(S_i, p)$ occurrences of $p(x_i)$ for each $p \in P$. The association between value $x_i$ and region $S_i$ is maintained by storing $x_i$ in a $\iota_i$-term (introduced in the right-hand side of the rule). If $F_t(S_i, p) = 0$ for any $p \in P$, then $x_i$ may be associated to a data $d_i$ not occurring in the current marking (i.e., selection of fresh data is a special case). Furthermore, by removing both the *first*- and the *last*-term, we disable the firing of rules that encode other data net transitions. Fig. B.7 in Appendix shows an example of application of the *subtract* rule.

The values $x_1, \ldots, x_k$ stored in $\iota_1$-,...,$\iota_k$-terms play the role of pointers to the regions $S_1, \ldots, S_k$. We refer to them as to the set of $\alpha_t$-*indexes*. The parameters of terms in $[f, l]$ associated to the other regions $R_0, \ldots, R_k$ are called *region-indexes*.

#### *5.1.3. Multiplication*

To simulate the multiplication step we proceed as follows. We first make a copy of the multiset of $P$-terms with parameters $v_1, \ldots, v_n$ in $[f, l]$ by copying each $p$-term with parameter $v_i$ in a $\overline{p}$-term with parameter $w_i$ such that $f' < w_1 < \ldots < w_n < l'$ and $[f', l']$ is an interval to the right of $[f, l]$, i.e., $l < f'$. The $\mathit{new}_t$-term in the *subtract* rule is used to enable the set of (silent) CMRS rules in Fig. B.1 in Appendix that create the copy-configuration. During the copy we add a $\checkmark$-term for any *visited* region index. These terms are used to remember region indexes whose corresponding $\overline{P}$-terms are all removed in the multiplication step (e.g., when all tokens with data $d \in R_i$ are removed).

For instance, $[p(v_1), p(v_2), p(v_2), q(v_3)]$ with $f < v_1 < v_2 < v_3 < l$ is copied as $[\overline{p}(w_1), \checkmark(w_1), \overline{p}(w_2), \overline{p}(w_2), \checkmark(w_2), \overline{q}(w_3)\checkmark(w_3)]$ for some $w_1, w_2, w_3$ such that $f < l < f' < w_1 < w_2 < w_3 < l'$. The CMRS rules of Fig. B.1 use

1 We recall that $[t_1, \ldots, t_n]$ denotes a multisets of terms. Furthermore, $\sum_{i:1,\ldots,k} M_i = M_1 + \cdots + M_k$, where $+$ is multiset union.

a special term $\uparrow$ as a pointer to scan the indexes in $[f, l]$ from left to right and create new $\overline{P}$-terms with parameters in the interval $[f', l']$. The pointer is non-deterministically moved to the right. Thus during the traversal we may forget to copy some token. This is the first type of loss we find in our encoding. Notice that lost tokens have parameters strictly smaller that $f'$.

The simulation of the multiplication step operates on the copy-configuration only (that with $\overline{P}$-terms). The (silent) CMRS rules that implement this step are shown in Fig. B.2 in Appendix. The intuition behind their definition is as follows.

We first consider all $\alpha_t$-indexes of $\overline{P}$-terms from left to right. For each $\alpha_t$-index $v_i$, we proceed as follows. We first select and remove a term $\overline{p}(v_i)$ (encoding a given token). We compute then the effect of the whole-place operation on the entire set of $\alpha_t$-indexes (including $v_i$ itself). More specifically, for an $\alpha_t$-index $v_j$ we add $G_t(S_i, p, S_j, q)$ occurrences of the term $q(v_j)$ to the current CMRS configuration. The use of $P$- and $\overline{P}$-terms with parameters in the same interval allows us to keep track of tokens still to transfer ($\overline{P}$-terms) and tokens already transferred ($P$-terms). We then consider all remaining indexes by means of a left-to-right traversal of region-indexes in the current configuration. During the traversal, we add new $P$-terms with region-indexes as parameters as specified by $G_t$. During this step, we may forget to transfer some $\overline{P}$-term. This is the second type of loss we find in the encoding. After this step we either consider the next token with $\alpha_t$-index $v_i$ or we move to the next $\alpha_t$-index. Fig. B.8(a) in Appendix illustrates the simulation of this kind of transfers (i.e., from $S_i$ to $S_j/R_j$).

After the termination of the whole-place operations for terms with $\alpha_t$-indexes, we have to simulate the transfer of $\overline{P}$-terms with region-indexes. For each such an index, we transfer tokens within the same region-index or to an $\alpha_t$-index. To simulate these operations we scan region-indexes from left-to-right to apply the matrix $G_t$. The (silent) CMRS rules that implement this step (enabled by the by term $trR_t$) are shown in Fig. B.3. Fig. B.8(b) in Appendix illustrates the simulation of this type of whole-place operation.

*5.1.4. Addition*

As a last step we add tokens to $\alpha_t$-indexes and visited region indexes as specified by $H_t$. For $\alpha_t$-indexes, we need a single rule that applies the matrix $H_t$. For region-indexes, we traverse from left-to-right the current configuration and apply $H_t$ to each marked (with a $\checkmark$-term) region-index $w$. As mentioned before, the $\checkmark$-term allows us to apply $H_t$ to regions emptied by the multiplication step. The rules for this step (associated to terms $add_t$ and $addR_t$) are shown in Fig. B.4. All the rules are silent except the last one whose label is the same as that of $t$. Fig. B.8(c) in Appendix shows an example of their application.

During the traversal, we may ignore some (marked) region-index. This is the last type of loss in our encoding. The new configuration is the final result of the simulation of the transition. Due to the possible losses in the different simulation steps, we may get a representation of a data net configuration smaller than the real successor configuration.

To formalize the relation between a data net $\mathcal{D}$ and its CMRS encoding $\mathcal{E}(\mathcal{D})$, for a configuration $s$ with data $d_1 \prec \ldots \prec d_k$ we use $s^{\vec{v}}$ to denote the CMRS representation with indexes $\vec{v} = (v_1, \ldots, v_k)$.

**Proposition 4.** *For configurations $s_0, s_1, s, s'$, the following properties hold:*

(i) *If $s_0 \xrightarrow{w} s_1$ in $\mathcal{D}$, then there exists $\vec{v}$ such that $[init] \xrightarrow{w} {s_1}^{\vec{v}}$ in $\mathcal{E}(\mathcal{D})$.*
(ii) *Furthermore, if $[init] \xrightarrow{w} c$ in $\mathcal{E}(\mathcal{D})$ and $s^{\vec{v}} \preceq_c c$ for some $\vec{v}$, then there exists $s_1$ such that $s_0 \xrightarrow{w} s_1$ in $\mathcal{D}$ with $s \preceq_d s_1$.*

Finally, suppose that the accepting data net marking is a sequence $M_1 \ldots M_k$ of $k$ vectors (multisets) over $\mathbb{N}^P$. Then, we add a silent CMRS rule

$$[first(f), last(l)] + \sum_{i \in \{1,\ldots,k\}} M_i^{x_i} \rightsquigarrow [p_{fin}] \;:\; f < x_1 < x_2 < \ldots < x_k < l, x = 0$$

where $p_{fin}$ is a fresh (with arity zero) predicate. By adding this rule, the accepting CMRS configuration can be defined as the singleton $[p_{fin}]$. From Lemma 1 and Proposition 4, we have the following result.

**Theorem 3.** $L_c(\textit{data nets}) = L_c(\textit{CMRS})$.

*5.2. Extensions of data nets*

In this section, we show how to modify the encoding of data nets defined in the previous section to encode some extensions of data nets. This allows us to show that the proposed extensions have the same expressiveness as CMRS and, hence, as data nets. Since the encoding we propose is effective (i.e., it can be computed automatically), from the algorithm for coverability in CMRS we obtain for free verification algorithm for the proposed extensions of data nets.

As a first extension, we consider freshness of data values. Let us consider a data net transition $t$ with $\alpha_t = k$. In the semantics of data nets, some of the $k$ data values selected by $t$ *may* be fresh, i.e., they do not have to occur in the current configuration. This definition can be extended by introducing the constraint that some of the selected data *must* be fresh for the transition to be fired. In the Petri net setting a similar operator has been considered in the $\nu$-nets of [18] to create new, unused identifier.

For simplicity, we consider here the extension of data net transitions in which we require that only one of the $\alpha_t$ data value must be fresh. This kind of transition can be modeled by extending the CMRS encoding of the subtraction step as follows. Before selecting the $\alpha_t$ data, we make a copy (in a new interval) of the current configuration. In the new configuration we non-deterministically mark using predicate *new* a value $x$ distinct from the values used to represent tokens. After this preliminary step, we apply the subtraction phase by requiring that the value $x$ is one of the selected ones (i.e., we need $\alpha_t$ rules for this last step). Formally, we use the rules in Fig. B.5 in Appendix. This extension provides a direct way to model freshness without need of ordering identifiers and of maintaining in a special place the last used one (the natural way of modeling $\nu$-nets in ordinary data nets).

Another possible extension concerns the relaxation of some of the restrictions in the definition of data nets in [20]. Assume we allow transfers between regions $R_i$ and $R_j$ with $i \neq j$. The semantics of a transfer with $G_t(R_i, p, R_j, p') = m > 0$ is the following. For each $d \in R_i$, place $p \in P$, and each token with $d$ in $p$, we add $m$ tokens with data $d'$ to $p'$ for each $d' \in R_j$. Furthermore, we can also consider a new type of whole-place operation within the same region $R_i$ in which we can multiply the tokens with data $d$ for each data with value $d' \in R_i$ with $d \neq d'$. More formally, assume we add a new matrix $M_t(R_i, p, R_i, p')$ to specify, for each token in $p$ with data $d \in R_i$, how many tokens to add to place $p'$ with data $d'$ for each $d' \in R_i, d' \neq d$. These extensions of data net transitions are still monotonic w.r.t. $\preceq_d$. Furthermore, they can be weakly simulated in CMRS as shown in Fig.B.6 in Appendix. Thus, we have that coverability remains decidable and, from Lemma 1, we have that

$$L_c(\textit{extended data nets}) = L_c(\textit{CMRS}).$$

## 6. Lossy FIFO channel systems

In this section, we study the relationship between the fragment $\Gamma_0$ of CMRS defined in Section 3.2 and *lossy (FIFO) channel systems* (LCSs) [4].

A *lossy FIFO channel system* (LCS) consists of an asynchronous parallel composition of finite-state machines that communicate by sending and receiving messages via a finite set of unbounded lossy FIFO channels (in the sense that they can non-deterministically lose messages). Formally, an LCS $\mathcal{F}$ is a tuple $(Q, C, M, \delta)$ where $Q$ is a finite set of control states (the Cartesian product of those of each finite-state machine), $C$ is a finite set of channels, $M$ is a finite set of messages, $\delta$ is a finite set of transitions, each of which is of the form $(q_1, Op, q_2)$ where $q_1, q_2 \in Q$, and $Op$ is a mapping from channels to channel operations. For any $c \in C$ and $a \in M$, an operation $Op(c)$ is either a *send* operation $!a$, a *receive* operation $?a$, the *empty* test $\epsilon?$, or the *null* operation *nop*. A configuration $\gamma$ is a pair $(q, w)$ where $q \in Q$, and $w$ is a mapping from $C$ to $M^*$ giving the content of each channel. The initial configuration $\gamma_{init}$ of $\mathcal{F}$ is the pair $(q_0, \varepsilon)$ where $q_0 \in Q$, and $\varepsilon$ denotes the mapping that assigns the empty sequence $\epsilon$ to each channel. To simplify the presentation, w.l.o.g. we fix usually the accepting configuration $\gamma_{fin} = (q_{fin}, \varepsilon)$ for some $q_{fin} \in Q$. The (strong) transition relation (that defines the semantics of machines with *perfect* FIFO channels) is defined as follows: $(q_1, w_1) \xrightarrow{\sigma} (q_2, w_2)$ if and only if $\sigma = (q_1, Op, q_2) \in \delta$ such that if $Op(c) = !a$, then $w_2(c) = w_1(c) \cdot a$; if $Op(c) = ?a$, then $w_1(c) = a \cdot w_2(c)$; if $Op(c) = \epsilon?$ then $w_1(c) = \epsilon$ and $w_2(c) = \epsilon$; if $Op(c) = nop$, then $w_2(c) = w_1(c)$. Now let $\preceq_l$ be the quasi ordering on LCS configurations such that $(q_1, w_1) \preceq_l (q_2, w_2)$ iff $q_1 = q_2$ and $\forall c \in C : w_1(c) \preceq_w w_2(c)$ where $\preceq_w$ indicates the subword relation. By Higman's theorem, we know that $\preceq_l$ is a well-quasi ordering. We introduce then the weak transition relation $\xRightarrow{\sigma}$ that defines the semantics of LCS: we have $\gamma_1 \xRightarrow{\sigma} \gamma_2$ iff there exists $\gamma_1'$ and $\gamma_2'$ s.t. $\gamma_1' \preceq_l \gamma_1$, $\gamma_1' \xrightarrow{\sigma} \gamma_2'$, and $\gamma_2 \preceq_l \gamma_2'$. Thus, $\gamma_1 \xRightarrow{\sigma} \gamma_2$ means that $\gamma_2$ is reachable from $\gamma_1$ by first losing messages from the channels and reaching $\gamma_1'$, then performing a transition, and, thereafter losing again messages from channels. As shown in [4], an LCS is well-structured with respect to $\preceq_l$. Furthermore, notice that for any model with lossy semantics like LCS, e.g., lossy vector addition systems [21], the class of $c$-languages coincide with the class of $r$-languages, i.e., $L_r(LCS) = L_c(LCS)$.

Our first result is that $\Gamma_0$ and LCS define the same class of $c$-languages.

**Theorem 4.** $L_c(\Gamma_0) = L_c(LCS)$.

To prove the previous result, we give separate proofs of the two inclusions.

**Proposition 5.** $L_c(LCS) \subseteq L_c(\Gamma_0)$.

**Proof.** Assume an LCS $\mathcal{F}$. We build a $\Gamma_0$ $\mathcal{S}$ that simulates $\mathcal{F}$. The set of predicate symbols in $\mathcal{S}$ consists of the following: For each $q \in Q$, there is a nullary predicate symbol $q$ in $\mathcal{S}$. For each channel $c_i$ we use the function symbols $head^i$ and $tail^i$ as pointers to the head and tail of the queue $c_i$. For each channel $c_i$ and each message $a \in M$ we have the predicate symbol $a^i$ in $\mathcal{S}$. If $C = \{c_1, \ldots, c_n\}$, then the initial configuration $(s_0, \epsilon)$ is represented as

$$M_0 = \left[s_0, head^1(v_0), tail^1(v_0+1), \ldots, head^n(v_0), tail^n(v_0+1)\right]$$

for some $v_0 \in \mathbb{N}$.

In order to represent the queue $c_i$ containing the word $a_1a_2 \ldots a_n$, we will use the multiset

$$\Big[head^i(v_0),\, a_1^i(v_1),\, \ldots,\, a_n^i(v_n),\, tail^i(v_{n+1})\Big]$$

for some positive integers $v_0 < v_1 < \ldots < v_{n+1}$.
Since an LCS transition $(q_1, Op, q_2)$ operates simultaneously on all the queues, the corresponding CMRS rule (with the same label) has the following form:

$$[\, q_1 \,] \;+\; B_1 \;+\; \cdots \;+\; B_n \rightsquigarrow [\, q_2 \,] \;+\; B_1' \;+\; \cdots \;+\; B_n' \;:\; C_1 \cup \ldots \cup C_n$$

where $B_i, B_i'$ and $C_i$ define the encoding of $Op(c_i)$ for $i : 1, \ldots, n$. The encoding of the operation is defined by atomic formulas defined on a distinct variables $x_1, y_1, \ldots, x_n, y_n$ as follows.
For $Op(c_i) = !a$,

$$B_i = \Big[tail^i(x_i)\Big] \quad B_i' = \Big[a^i(x_i), tail^i(y_i)\Big] \quad C_i = \{x_i < y_i\}$$

For $Op(c_i) = ?a$,

$$B_i = \Big[head^i(x_i), a^i(y_i)\Big] \quad B_i' = \Big[head^i(y_i)\Big] \quad C_i = \{x_i < y_i\}$$

For $Op(c_i) = empty?$,

$$B_i = \Big[head^i(x_i), tail^i(y_i)\Big] \quad B_i' = \Big[head^i(x_i'), tail^i(y_i')\Big] \quad C_i = \{y_i < x_i' < y_i'\}$$

For $Op(c_i) = nop$,

$$B_i = B_i' = [\,] \quad C_i = true$$

The accepting CMRS configuration is $[q_{fin}]$. Let us consider a LCS with one channel. Note that, as shown in [5], $n$ channels can be encoded into one channel in presence of transitions labeled with $\epsilon$. Hence, considering a unique channel is not restrictive. The following properties then hold. Given an LCS configuration $\gamma = (s, w)$, let $\gamma^\bullet$ be the corresponding CMRS encoding. Moreover, given $\gamma^\bullet$ containing $head^1(c)$, let $\mathsf{G}(\gamma^\bullet)$ be the set of CMRS configurations built from $\gamma^\bullet$ by adding some ground terms $a^1(c')$ where $a \in M$ and $c' < c$, i.e., by adding useless ground terms corresponding to lost messages.
It is easy to check that (1) if $\gamma_1^\bullet \xrightarrow{\rho} \eta$ with $\eta \in \mathsf{G}(\gamma_2^\bullet)$ in $\mathcal{S}$, then $\gamma_1 \xRightarrow{\sigma_\rho} \gamma_2$ in $\mathcal{F}$ where $\rho$ is the CMRS rule corresponding to the LCS transition $\sigma_\rho$. Indeed, notice that in the CMRS implementation of the *dequeue* operation we move the *head* pointer to an arbitrary position within the queue and thus we perform a lossy step followed by a dequeue step. Similarly, the emptiness test is simulated by means of a lossy step in which all elements are removed from the queue (with the weak reduction of LCS the emptiness test is always executable and it has the effect of emptying the queue). Finally, the enqueue operation is simulated in an exact way. We can also easily see that for all $\eta \in \mathsf{G}(\gamma_1^\bullet)$: (2) if $\eta \xrightarrow{\rho} \eta'$ in $\mathcal{S}$, then $\gamma_1^\bullet \xrightarrow{\rho} \gamma_2^\bullet$ with $\eta' \in \mathsf{G}(\gamma_2^\bullet)$.

Hence, if $\gamma_0^\bullet \xrightarrow{\rho_1} \eta_1 \cdots \xrightarrow{\rho_n} \eta_n$ with $[q_{fin}] \preceq_c \eta_n$, then we deduce from (2) that for all $i : 1 \leq i \leq n$, there exists $\gamma_i^\bullet$ such that $\eta_i \in \mathsf{G}(\gamma_i^\bullet)$. Moreover, for all $i : 0 \leq i < n$, $\gamma_i^\bullet \xrightarrow{\rho_i} \eta_{i+1}'$ with $\eta_{i+1}' \in \mathsf{G}(\gamma_{i+1}^\bullet)$. Since $[q_{fin}] \preceq_c \eta_n$ and $\eta_n \in \mathsf{G}(\gamma_n^\bullet)$ we also have $[q_{fin}] \preceq_c \gamma_n^\bullet$. Following (1), we deduce $\gamma_0 \xRightarrow{\sigma_{\rho_1}} \gamma_1 \ldots \xRightarrow{\sigma_{\rho_n}} \gamma_n$ with $\gamma_{fin} \preceq_l \gamma_n$.
Vice versa, suppose that $\gamma_1 \xRightarrow{\sigma} \gamma_2$ in $\mathcal{F}$. Then, we have that there exists $\eta$ such that $\gamma_1^\bullet \xrightarrow{\rho_\sigma} \eta^\bullet$ and $\gamma_2^\bullet \preceq_c \eta^\bullet$ where $\rho_\sigma$ is the CMRS rule corresponding to $\sigma$. This is immediately verified for the enqueue operation and for the empty test (their simulation is exact, and thus returns a more precise representation of the queues). The same holds for the dequeue operation since we cannot forget elements to the right of the new position of the header.
Now let $\gamma_1 \xRightarrow{\rho_0} \gamma_2 \ldots \xRightarrow{\rho_n} \gamma_n$ with $\gamma_{fin} \preceq_l \gamma_n$. Then, we know that there exist $\eta_2', \ldots, \eta_n'$ such that $\gamma_i^\bullet \xrightarrow{\sigma_{\rho_{i-1}}} \eta_{i+1}^\bullet$ and $\gamma_{i+1}^\bullet \preceq_c \eta_{i+1}^\bullet$ for $i : 1, \ldots, n-1$. By the monotonicity of CMRS, we have that $\gamma_1^\bullet \xrightarrow{\sigma_{\rho_0} \ldots \sigma_{\rho_n}} \eta_n^\bullet$, and $[q_{fin}] \preceq_c \gamma_{fin}^\bullet \preceq_c \eta_n^\bullet$. $\square$

**Proposition 6.** $L_c(\Gamma_0) \subseteq L_c(LCS)$.

**Proof.** Consider a $\Gamma_0$ $\mathcal{S}$ over the finite set of predicate symbols $\mathbb{P}$, an initial configuration $\gamma_{init}$ and an accepting configuration $\gamma_{fin}$. Remember that we assume that for each $p(v) \in \gamma_{init} + \gamma_{fin} : 0 \leq v \leq cmax$.

The proof follows three steps: first, we show how to encode a configuration as words (i.e., contents of LCS queues). Second, we show how a rule $L \rightsquigarrow R : \psi$ can be applied to the word representation of configurations, and finally we show how to simulate such an application using an LCS. $\square$

### 6.1. $\Gamma_0$ configurations as words

$\Gamma_0$ configurations consisting of terms with strictly increasing parameters can be naturally viewed as words defined over the corresponding predicate symbols. The execution of a $\Gamma_0$ on such a configuration, however, may lead to a new configuration with two terms with the same value.

As an example, consider the rule $\rho$ defined as $[p(x), q(y)] \rightsquigarrow [p(x), r(z), q(y)] : x < z < y$ and a configuration $\gamma = [p(0), t(3), q(6)]$. We notice here that the application of $\rho$ to $\gamma$ may lead to different results depending on the valuation of $z$. One of the possible successors is $\gamma' = [p(0), r(3), t(3), q(6)]$. $\gamma'$ is obtained by applying the valuation $x \mapsto 0, z \mapsto 3, y \mapsto 6$. The question now is if we gain something in assigning to $z$ the same value of a parameter in another term. The answer is no. Indeed, since in $\Gamma_0$ we cannot test for $=$ in a rule, the effect of mapping $z$ to 3 only restricts the set of rules that can be fired at $\gamma'$, i.e., this choice can lead to dead ends.

This intuition is made formal in Proposition 3. This lemma tells us that all strings in $L_c(\mathcal{S})$ can be recognized by an execution that passes through configurations where all the terms with a value greater than *cmax* are totally ordered on the values of their parameters w.r.t. $<$ (i.e., they can be viewed as words). Notice that this reasoning can be applied only to terms with values greater than *cmax*. Indeed, for this kind of terms Proposition 2 tells us that if we fire a sequence of transitions and reach a configurations $\gamma$ from $\gamma_{init}$ then we can fire the same sequence of transitions from $\gamma_{init}$ and reach a configuration with larger gap than in $\gamma$. Proposition 2 also implies that we do not have to retain gap between parameters greater than *cmax* since it is always possible to increase them. Terms with values smaller than *cmax* must be treated in a special way.

More precisely, a configuration $\gamma$ is encoded as a word $w_1 \cdot w_2$ where $w_1$ and $w_2$ are built as follows:

- Each ground term $p(c) \in \gamma$ with $0 \le c \le cmax$ is encoded as a (message) symbol $(p, c)$. Thus, from $\gamma$ we first extract the word

  $$w_1 = w_1^0 \cdots w_1^{cmax}$$

  where $w_1^i$ has many occurrences of $(p, i)$ as those of $p(i)$ for any predicate $p \in \mathbb{P}$ and $0 \le i \le cmax$ (multiple occurrence of the same term produce different symbols, to disambiguate the encoding we assume a total order on symbols in $\mathbb{P}$).
- Each ground term $p(c) \in \gamma$ with $c > cmax$ is encoded as a symbol $p$. Thus, from $\gamma$ we also extract the word

  $$w_2 = p_1 \cdots p_k$$

  where $p_i(c_i) \in \gamma, c_i > cmax$ and $c_i < c_j$ for $1 \le i < j \le k$. Here we assume that there cannot be two terms with the same value for parameters greater than *cmax*.

### 6.2. Applying rewriting rules to words

W.l.o.g. we assume that each rule $L \rightsquigarrow R : \psi$ in $\mathcal{S}$ with set of variables $V$ satisfies the following conditions:

- For each $x \in V$, either $(x = c) \in \psi$ and $0 \le c \le cmax$ or $(x > cmax) \in \psi$.
- Furthermore, we assume that for all pair of variables $x, y$ in $L + R$ such that $x > cmax \in \psi$ and $y > cmax \in \psi$ we have that $x \bullet y \in \psi$ with $\bullet \in \{=, >, <\}$.

Given a $\Gamma_0$ rule $\rho$, we can compile $\rho$ in a finite set of $\Gamma_0$ rules that satisfy the above mentioned conditions and that model the possible effects of applying $\rho$. The rules are obtained by completing the order in $\rho$ with all possible missing relations between variables. By Proposition 3, we can safely introduce new equality constraints only when the resulting rule respecting the restrictions of $\Gamma_0$ (i.e., we do not need to introduce equality constraints involving more than two variables). As an example, the effect of the rule $[p(x), q(y)] \rightsquigarrow [r(z)] : x < y$ on a configuration in $\Gamma_0$ is modeled by the rules $[p(x), q(y)] \rightsquigarrow [p(z)] : x < y < z$, $[p(x), q(y)] \rightsquigarrow [p(z)] : x < z < y$, $[p(x), q(y)] \rightsquigarrow [p(z)] : z < x < y$, $[p(x), q(y)] \rightsquigarrow [p(x)] : x < y$, and $[p(x), q(y)] \rightsquigarrow [p(y)] : x < y$. Notice that in the last two rules we introduce an implicit equality between a variable in the rhs and a variable in the lhs.

Under these assumptions, a rule $\rho = L \rightsquigarrow R : \psi$ in $\mathcal{S}$ defined over the variables $V = \{x_1, \ldots, x_{m+r}\}$ can be represented by the word

$$w^\rho = w_1^\rho \cdot w_2^\rho$$

where $w_1^\rho$ describes the effect of $\rho$ on $w_1$, i.e., on ground terms with parameter smaller than *cmax*, and $w_2^\rho$ describes the effect of $\rho$ on $w_2$, i.e., on ground terms with parameter greater than *cmax*. More precisely,

$$w_2^\rho = \begin{bmatrix} \alpha_1 \\ \beta_1 \end{bmatrix} \cdots \begin{bmatrix} \alpha_r \\ \beta_r \end{bmatrix}$$

is the maximal sequence that satisfies the following conditions:

- For $1 \leq k \leq r$,
  - $\alpha_k = p \in \mathbb{P}$, if $p(x_{j_k}) \in L$ and $(x_{j_k} > cmax) \in \psi$,
  - $\beta_k = q \in \mathbb{P}$, if $q(x_{j_k}) \in R$ and $(x_{j_k} > cmax) \in \psi$,
  - $\alpha_k$ and $\beta_k$ are equal to $\epsilon$ in all other cases.
- For $1 \leq k \leq r-1$, $x_{j_k} < x_{j_{k+1}}$ follows from $\psi$;

Pairs of the form $\begin{bmatrix} \epsilon \\ \epsilon \end{bmatrix}$ are not included in $w_2^\rho$. The word $w_2^\rho$ specifies the order of terms in $\rho$ and how a single term of a configuration (element in a word) is modified (using the pair $\begin{bmatrix} \alpha \\ \beta \end{bmatrix}$) by the rule. Notice that the syntactic restrictions of $\Gamma_0$ ensure that there cannot be elements $\begin{bmatrix} \alpha \\ \beta \end{bmatrix}$ with more than one predicate in $\alpha$ or $\beta$. Furthermore, if $\alpha = \epsilon$ then $\rho$ adds a new occurrence of $\beta$, if $\beta = \epsilon$ then $\rho$ removes an occurrence of $\alpha$.

As an example, for $cmax = 2$, the rule $\rho$ defined as $[p(x), q(y)] \rightsquigarrow [q(x), r(z)] : 2 < x < z < y$ is represented by the word

$$w_2^\rho = \begin{bmatrix} p \\ q \end{bmatrix} \cdot \begin{bmatrix} \epsilon \\ r \end{bmatrix} \cdot \begin{bmatrix} q \\ \epsilon \end{bmatrix}$$

The word $w_1^\rho = w_{1,0}^\rho \cdots w_{1,cmax}^\rho$ is such that for all $i : 0 \leq i \leq cmax$, $w_{1,i}^\rho = \varepsilon$ if there is no variable $x$ such that $x = i \in \psi$, otherwise $w_{1,i}^\rho$ is a sequence of elements of the form $\begin{bmatrix} (p, i) \\ \epsilon \end{bmatrix}$ or $\begin{bmatrix} \epsilon \\ (p, i) \end{bmatrix}$ that describe the modifications that concerns ground terms with parameter $x = i$ as mentioned above.

Given a word $w_1 \cdot w_2$ associated to a configuration $\gamma$ and a word $w_1^\rho \cdot w_2^\rho$ associated to a rule $\rho$, it should be clear now that the application of $\rho$ to $\gamma$ can be simulated by rewriting $w_1$ according to the ordered pairs in $w_1^\rho$ and $w_2$ according to the ordered pairs in $w_2^\rho$. Clearly, the application of $w_2^\rho$ to $w_2$ has some non-determinism, since we only have to ensure that in the resulting string $w_2'$ the order in the two strings is preserved.

Going back to our example, the application of $w_2^\rho$ to the word $w_2 = q \cdot p \cdot s \cdot q \cdot t$ produces the strings $q \cdot q \cdot r \cdot s \cdot t$ and $q \cdot q \cdot s \cdot r \cdot t$. (we recall that Proposition 3 tells us that we can safely ignore configurations in which $r$ gets the same value as $s$).

### 6.3. Simulation in LCS

We are ready now to define the encoding of a $\Gamma_0$ $\mathcal{S}$ into an LCS $\mathcal{F}$. The LCS $\mathcal{F}$ has one channel $c$ that contains the word encodings of configurations and one channel $c'$ used as auxiliary memory. The control states of $\mathcal{F}$ are used for encoding different steps of simulation of a rule $\rho = L \rightsquigarrow R : \psi$ where we assume that all pair of variables $x, y$ in $\rho$ are in the relation $< \cup =$ induced by $\psi$. In particular, we will assume to have one distinct control state for each pair in $w^\rho$.

First, we simulate the effect on the terms with parameters less or equal than $cmax$. For a fixed $i$ $1 \leq i \leq cmax$, the simulation consists in dequeuing symbols of the form $(p, i)$ from $c$, and by copying them into $c'$ after applying the transformations defined in $w_1^\rho$. Notice that the information on the structure of $w_1^\rho$ can be stored in the control states of $\mathcal{F}$. When there are no more symbols of the form $(p, i)$ in $c$, we moves to the value $i + 1$.

Second, we simulate the effect of $w_2^\rho$ for ground terms $p(c)$ with $c > cmax$. Suppose that $w_2^\rho$ has $r$ pairs $\begin{bmatrix} \alpha \\ \beta \end{bmatrix}$. Starting from the first pair in $w_2^\rho$, we define control states in which we either copy symbols from $c$ to $c'$ or, non-deterministically, decide to apply the current pair $\begin{bmatrix} \alpha \\ \beta \end{bmatrix}$ to the head $p$ of the queue:

- If $\alpha = p$ and $\beta = q$, then we remove $p$ from $c$, add $q$ to $c'$, and move to the next pair in $w_2^\rho$.
- If $\alpha = p$ and $\beta = \epsilon$, then remove $p$ from $c$, and move to the next pair in $w_2^\rho$.
- If $\alpha = \epsilon$ and $\beta = q$, we add $q$ to $c'$, and move to the next pair in $w_2^\rho$.

Note that since we non-deterministically choose the positions where modifications must be applied, the LCS $\mathcal{F}$ may get into a deadlock. Deadlocked computations do not influence the language $L_c(\mathcal{F})$.

Once the new word has been written into $c'$ (and $c$ is empty), we copy the content of $c'$ into $c$ and get ready for simulating the execution of another rule. In this last step we also recognize the symbols $\lambda(\rho)$ that labels $\rho$ (all the other transitions used to simulate $\rho$ are labeled by $\varepsilon$).

Finally, note that channels may lose messages. As a consequence, we encode *lossy* $\Gamma_0$ into LCS where ground terms may non-deterministically disappear during executions. However, following Lemma 1, the languages accepted by $\Gamma_0$ and lossy $\Gamma_0$ are the same (for coverability acceptance). $\square$

We show next that CMRS are strictly more expressive than LCS and $\Gamma_0$.

**Theorem 5.** $L_c(LCS) \subset L_c(CMRS)$.

**Proof.** We define a language $L_{ent}$ which is accepted by a CMRS and that cannot be accepted by any LCS. Assume a finite alphabet $\Sigma$ such that $\{\$, \#\} \not\subseteq \Sigma$. For each $w = a_1 \cdots a_k \in \Sigma^*$, we interpret $w$ in the following as the multiset $[a_1, \ldots, a_k]$. Hence, we do not distinguish words in $\Sigma^*$ from the multiset they represent, and vice versa. In particular, we will use the notation $a_1 \cdots a_k \le a'_1 \cdots a'_l$ to denote that $[a_1, \ldots, a_k] \le [a'_1, \ldots, a'_l]$. Define $V$ to be the set of words of the form $w_1\#w_2\# \cdots \#w_n$ where $w_i \in \Sigma^*$ for each $i : 1 \le i \le n$. Consider $v = w_1\#w_2\# \cdots \#w_m \in V$ and $v' = w'_1\#w'_2\# \cdots \#w'_n \in V$. We write $v \sqsubseteq v'$ to denote that there is an injection $h : \{1, \ldots, m\} \mapsto \{1, \ldots, n\}$ such that

1. $1 \le i < j \le m$ implies $h(i) < h(j)$ ($h$ is monotonic) and
2. $w_i \le w'_{h(i)}$ ($\le$ is multiset inclusion) for each $i : 1 \le i \le m$.

We now define the language $L_{ent} = \{v\$v' | v' \sqsubseteq v\} \subseteq (\Sigma \cup \{\#, \$\})^*$. As an example, given $\Sigma = \{a, b\}$, we have that $[a, b, b]\#[a, b, b]\#[a, a]\$[b, a]\#[a, a]$ is in $L_{ent}$, whereas $[a, b, b]\#[b, a, b]\#[a, a]\$[a, a]\#[a, b]$ is not in $L_{ent}$.

We now exhibit a CMRS $\mathcal{S}$ with $L_c(\mathcal{S}) = L_{ent}$. The set of predicate symbols which appear in $\mathcal{S}$ consists of (i) a predicate symbol $a$ for each $a \in \Sigma$, and (ii) the symbols *guess*, *check*, $sep_\#$ and the nullary predicate $p_{fin}$. The initial configuration $\gamma_{init}$ is defined as $[guess(0)]$. Furthermore, we have the following rules:

(1) For each $a \in \Sigma$, we have a rule labeled with $a$ and which is of the form

$$[guess(x)] \rightsquigarrow [guess(x), a(x)] \quad : \quad true$$

Rules of this form are used to guess the letters in $w_i$ in the first part of a word in $L_{ent}$. We keep track of the symbols inside $w_i$ through their argument. These arguments are all the same by definition of the rule.

(2) A rule labeled with # of the form:

$$[guess(x)] \rightsquigarrow [sep_\#(x)\ ,\ guess(y)] \quad : \quad \{x < y\}$$

This rule is used to switch from the guessing of the part $w_i$ to the guessing of the next part $w_{i+1}$. $sep_\#(x)$ remembers the parameter on which the switch has been executed.

(3) A rule labeled with \$ of the form:

$$[guess(x)\ ] \rightsquigarrow [check(y)\ ,\ sep_\#(x)] \quad : \quad \{y = 0\}$$

This rule is used to switch from the guessing of the part $w_1\# \ldots \#w_n$ to the selection of the second part of the word. The parameter of *check* is equal to the initial value of *guess*, i.e., to 0. This way, we can scan the word stored in the first phase from left-to-right, i.e., working on the argument order we define a monotonic injective mapping $h$.

(4) For each $a \in \Sigma$, we have a rule labeled with $a$ which is of the form

$$[check(y), a(y)] \rightsquigarrow [check(y)] \quad : \quad true$$

This rule is used to read a word (multiset) $u_i$ contained in $w_{h(i)}$.

(5) A rule labeled with # of the form:

$$[check(x)\ ,\ sep_\#(x)\ ,\ sep_\#(y)\ ] \rightsquigarrow [check(y)\ ,\ sep_\#(y)] \quad : \quad \{x < y\}$$

This rule is used to pass from $u_i$ to $u_{i+1}$ for $i \ge 1$.

(6) A rule labeled with $\epsilon$ of the form:

$$[check(x)\ ] \rightsquigarrow [p_{fin}] \quad : \quad true$$

This rule is used to non-deterministically terminate the checking phase. The accepting configuration $\gamma_{fin}$ is defined as $[p_{fin}]$.

Assuming that $\Sigma = \{a, b\}$, we now show that $L_{ent}$ is not an LCS language. Suppose that $L_c(\mathcal{F}) = L_{ent}$ for some LCS $\mathcal{F} = (Q, \{c\}, M, \delta)$. We show that this leads to a contradiction. Let $\gamma_{init}$ be the initial global state in $\mathcal{F}$ and $\gamma_{fin}$ be the accepting global state. We use a binary encoding $enc : Q \cup M \mapsto \Sigma^*$ such that $enc(m) \not\le enc(m')$ if $m \ne m'$. We will also use a special word $v_{init} \in \Sigma^*$ such that $v_{init} \not\le enc(m)$ for each $m \in Q \cup M$. It is clear that such $enc$ function and $v_{init}$ exist. As an example, if $|Q \cup M| = n$ then we define $enc$ as an injective map from $Q \cup M$ to multisets of $n + 1$ elements with $i + 1$ occurrences of $a$ and $n - i$ occurrences of $b$ for $0 \le i \le n$, and we use the multiset with $n + 1$ occurrences of $b$ for $v_{init}$. For instance, for $n = 2$ we use $[a, a, a]$, $[a, a, b]$, $[a, b, b]$ for control states and messages and $[b, b, b]$ for $v_{init}$. We extend $enc$ to global states such that if $\gamma = (q, m_1 m_2 \cdots m_n)$ then

$$enc(\gamma) = enc(q)\#enc(m_1)\#enc(m_2)\# \cdots \#enc(m_n)$$

Observe that (i) $enc(\gamma) \in V$; (ii) for global states $\gamma_1$ and $\gamma_2$, it is the case that $\gamma_1 \preceq_l \gamma_2$ iff $enc(\gamma_1) \sqsubseteq enc(\gamma_2)$; and (iii) $v_{init} \not\sqsubseteq enc(\gamma)$ for each global state $\gamma$.

Since $L_{ent} = L_c(\mathcal{F})$ and $v\$v \in L_{ent}$ for each $v \in V$, it follows that for each $v \in V$, there is a global state $\gamma$ such that $\gamma_{init} \xrightarrow{v} \gamma \xrightarrow{\$v} \gamma'$ with $\gamma_{fin} \preceq_l \gamma'$. We use $reach(v)$ to denote $\gamma$. We define two sequences $\gamma_0, \gamma_1, \gamma_2, \ldots$ of global states, and $v_0, v_1, v_2, \ldots$ of words in $V$ such that $v_0 = v_{init}$, $\gamma_i = reach(v_i)$, and $v_{i+1} = enc(\gamma_i)$ for each $i \geq 0$. By Higman's theorem we know that there is a $j$ such that $\gamma_i \preceq_l \gamma_j$ for some $i < j$. Let $j$ be the smallest natural number satisfying this property. First, we show that $v_i \not\sqsubseteq v_j$. There are two cases: if $i = 0$ then $v_i \not\sqsubseteq v_j$ by (*iii*); if $i > 0$ then we know that $\gamma_{i-1} \not\preceq_l \gamma_{j-1}$ and hence, following (*ii*), $v_i = enc(\gamma_{i-1}) \not\sqsubseteq enc(\gamma_{j-1}) = v_j$. Since $\gamma_j = reach(v_j)$, we know that $\gamma_{init} \xrightarrow{v_j} \gamma_j$. By monotonicity, $\gamma_i \xrightarrow{\$v_i} \gamma_i'$, $\gamma_{fin} \preceq_l \gamma_i'$, $\gamma_i \preceq_l \gamma_j$ implies $\gamma_j \xrightarrow{\$v_i} \gamma_j'$ with $\gamma_{fin} \preceq_l \gamma_i' \preceq_l \gamma_j'$. We conclude that $\gamma_{init} \xrightarrow{v_j} \gamma_j \xrightarrow{\$v_i} \gamma_j'$ with $\gamma_{fin} \preceq_l \gamma_j'$. Hence, $v_j\$v_i \in L_c(\mathcal{F}) = L_{ent}$ which is a contradiction since $v_i \not\sqsubseteq v_j$. $\square$

Let us now consider $r$-languages. As mentioned at the beginning of the section, the expressive power of LCS remains the same as for coverability accepting conditions, However, this property does not hold anymore for $\Gamma_0$.

**Proposition 7.** $L_c(\Gamma_0) \subset L_r(\Gamma_0) = L_r(CMRS) = RE$.

**Proof.** It is well known that *perfect* FIFO channel systems with reachability accepting condition recognize the class RE. We prove that perfect channel systems accept the same languages as $\Gamma_0$ with reachability accepting condition. Given an LCS $\mathcal{F}$, let $\mathcal{S}$ be the $\Gamma_0$ used to encode an LCS in the proof of Theorem 4. In each step of a run $\sigma$ in $\mathcal{S}$ the head and tail delimiters are moved to the right of their current positions. Thus, a "lost" ground term to the left of the head delimiter corresponding to its queue $c_i$, i.e., with parameter smaller than that of $head^i$, can never be removed in successive steps of $\sigma$. This implies that an accepting configuration in which all ground terms have parameters strictly greater than the parameter of the head delimiter characterize reachable configurations of a perfect FIFO channel system. $\square$

Hence, we have the following property.

**Corollary 1.** $L_r(LCS) \subset L_r(CMRS)$.

## 7. Petri nets extensions

Petri nets (PNs), a well-known model of concurrent computation [22], can naturally be reformulated in a multiset rewriting system operating on nullary predicates only (i.e., predicates with no parameters). This class of rewriting rules corresponds to those in the fragment $\Gamma_1$ of CMRS defined in Section 3.2. To fix the notations, a PN configuration, called *marking*, is a multiset of symbols taken from the set of places $P$ of the PN. A marking $M$ containing $k$ symbols $p$ means that the place $p$ contains $k$ tokens. A PN transition $t$ is a pair of multiset $(I_t, O_t)$ where $I_t$, resp. $O_t$, defines the tokens removed, resp. added, when applying $t$; i.e., firing $t$ from a marking $M$ leads to the marking $M' = M - I_t + O_t$. Notice that the firing of $t$ from $M$ can occur only if $I_t \leq M$. It is easy to see that, if we associate a predicate symbol to each place of a net, configurations and rules of a $\Gamma_1$ model are just alternative representations of markings and transitions of a Petri net. As an immediate consequence of this connection, we have that $L_c(\Gamma_1) = L_c(PN)$ and $L_r(\Gamma_1) = L_r(PN)$. To formally compare $\Gamma_1$ with the other models, we use the following extensions of Petri nets:

*Lossy Petri net with inhibitor arcs* (LN) are Petri nets in which it is possible to test if some places have no tokens and in which tokens may get lost before and after executing a transition. To achieve this, each transition $t$ is equipped with a (possibly empty) set of place $Z_t$, often called *inhibitor arc*, with $Z_t \cap I_t = \emptyset$. A transition $t = (I_t, O_t)$ is fireable from a marking $M$ as usual. If it does, the firing of $t$ leads to any marking $M'$ such that there exists three markings $M_1, M_2, M_3$: $I_t \leq M_1 \leq M$ and $M_1$ contains no $p \in Z_t$, $M_2 = M_1 - I_t + O_t$, and $M_3 \leq M_2$.

*Transfer nets* (TNs) are Petri nets extended with *transfer arcs*. A transfer arc is a pair $S \hookrightarrow q$ where $S$ is a set of places of the net and $q \notin S$ is a place. Given a set of places $P$, let us consider a transition $t = (I_t, O_t)$ with transfer $S \hookrightarrow q$ such that $S \cap I_t = \emptyset$. Given a marking $M$, $t$ is fireable if $I_t \leq M$. Its firing leads to the new marking $M'$ computed in three steps: we first compute $M_1 = M - I_t$, then we move all tokens in the places in $S$ to the place $q$ obtaining $M_2$; finally, we compute $M'$ as $M_2 + O_t$.

*Reset nets* (RNs) are Petri nets extended with *reset arcs*, i.e., with a transfer arc $S \hookrightarrow \perp$ where $\perp$ is a special place used only to reset places.

As an example, let $P = \{p, q, r, s\}$ and consider a transition $t$ with $I = \{p, q\}$, $O = \{p, s\}$, and transfer arc $\{p, q\} \hookrightarrow r$. Now, consider the marking $M = [p, p, q, q, q]$. Then, the execution of $t$ leads to the marking $M' = [p, s, r, r, r]$ (we first compute $M - I = [p, q, q]$, then execute the transfer obtaining $[r, r, r]$, and, finally, add $O$). If the transfer arc is instead $\{p, q\} \hookrightarrow \perp$, the execution of $t$ leads to the marking $M' = [p, s]$ (markings do not refer to tokens in the special place $\perp$).

We first notice that the lossy version of RN(TN) (i.e., where tokens can be lost before and after applying the effect of transitions)define the same $c$-languages as RN(TN). We now prove that Lossy TN, Lossy RN and LN recognize the same class of $c$-languages.

- *Lossy TN as Lossy RN*: Let $P$ and $T$ be the set of places and transitions of a Lossy TN $\mathcal{N}$. We build a Lossy RN with places $P$ augmented by place $n$ and $s_t$ for each transition $t \in T$. The new places are used to distinguish normal transitions from simulations of the transfer of transition $t$. Consider now a transition $t$ with label $\ell$, $I_t = \{p_1, \ldots, p_m\}$, $O_t = \{q_1, \ldots, q_n\}$ and transfer $S \hookrightarrow q$ with $S = \{r_1, \ldots, r_k\}$. Transition $t$ is simulated via the following set of transitions:
  - A transition $t_0$ labeled with $\epsilon$ such that $I_{t_1} = I \cup \{n\}$ and $O_{t_1} = \{s_t\}$. This transition checks if $t$ is fireable and then activates the simulation of its transfer by adding a token to $s_t$.
  - A set of transitions $t_1, \ldots, t_k$ labeled with $\epsilon$ such that $I_{t_i} = \{s_t, r_i\}$ and $O_{t_i} = \{s_t, q\}$ for $i : 1, \ldots, k$. Each such transition moves a single token from a places in $S$ to $q$.
  - A transition $t'$ labeled with $\ell$ such that $I_{t'} = \{s_t\}$, $O_{t'} = O_t \cup \{n\}$, and with the reset arc $S \hookrightarrow \perp$. This transition non-deterministically terminates the simulation of the transfer and the tokens that remained in the places of $S$ are lost.
- *Lossy RN as LN*: Given a Lossy RN $\mathcal{N}$, we can build a LN $\mathcal{N}'$ that accepts the same $c$-language simply by replacing each reset arc $S \hookrightarrow \perp$ of a transitions $t$ with an inhibitor arc $Z_t = S$. Indeed, notice that the firing of a transition $t$ with reset arc $S \hookrightarrow \perp$ in $\mathcal{N}$ at a marking $M$ has the effect of forcing all places in $S$ to be empty in the successor marking of $M$. Now, the corresponding transition $t'$ in $\mathcal{N}'$ can be fired at $M$ only if each place in $S$ is empty in $M$. However, since $\mathcal{N}'$ is lossy this condition can always be verified (all tokens in places in $S$ may get lost) and it has the same effect on $M$ as $t$. Vice versa, if all tokens in places in $S$ get lost, then we can fire $t'$ and its firing has the same effect of $t$.
- *LN as Lossy TN*: Given a LN $\mathcal{N}$, we build a Lossy TN $\mathcal{N}'$ that accepts the same $c$-language simply by replacing each inhibitor arc $Z_t = S$ of a transitions $t$ with a transition $t'$ with a transfer arc $S \hookrightarrow p_t$ where $p_t$ is a new place. We assume that tokens in $p_t$ can never be re-used (i.e., $p_t$ cannot occur in the preset of a transition in $\mathcal{N}'$). Indeed, notice that, since $\mathcal{N}$ is lossy, the inhibitor arcs $Z_t = S$ in $\mathcal{N}$ are enabled if we first lose all tokens in places in $S$. Thus, the inhibitor arcs have the same effect of a transfer to the new place $p_t$ from which tokens can never be re-used. Vice versa, a transition $t$ with transfer of all tokens of places in $S$ to place $p_t$ can be simulated by its corresponding transition with inhibitor arc $t'$. Indeed, in a lossy step all tokens in places in $S$ may get lost thus enabling the inhibitor arc $Z_t$.
- *LN as LCS*: Given a LN $\mathcal{N}$ with places $P$ and transitions in $T$, we build an LCS $\mathcal{F}$ that accepts the same $c$-language as follows. The LCS $\mathcal{F}$ has messages defined over the singleton set of symbols $\{\bullet\}$. Furthermore, it uses a distinguished channel $c_p$ to model each place $p \in P$. Thus, we use a queue $c_p$ with $k$ occurrences of $\bullet$ to simulate a place $p$ with $k$ tokens. Notice that we do not need to exploit the FIFO ordering of channels. Based on this idea, the simulation of a transition becomes straightforward. The consumption of a token from place $p$ is simulated by a *dequeue* operation of message $\bullet$ executed on channel $c_p$, the production of a token in place $p$ is simulated by an *enqueue* operation on channel $c_p$, and an inhibitor arc on place $p$ is modeled by the *empty* test on channel $c_p$.

Thus, we have that $L_c(LN) = L_c(RN) = L_c(TN)$, $L_c(LN) \subseteq L_c(LCS)$, and, as for LCS, $L_r(LN) = L_c(LN)$.

Furthermore, in [15] the authors proved that $L_c(PN) \subset L_c(TN)$. From all these properties, we obtain the following result.

**Theorem 6.** $L_c(\Gamma_1) \subset L_c(\Gamma_0)$.

For $r$-languages, the classification changes as follows.

**Theorem 7.** *$L_r(\Gamma_1) \not\sim L_r(LCS)$, $L_r(\Gamma_1) \not\sim L_r(LN)$, and $L_r(\Gamma_1) \subset L_r(\Gamma_0)$.*

**Proof.** We first prove that $L_r(\Gamma_1) = L_r(PN) \not\subseteq L_c(LCS) = L_r(LCS)$, hence $L_r(\Gamma_1) \not\subseteq L_c(LN) = L_r(LN)$ since $L_c(LN) \subseteq L_c(LCS) = L_r(LCS)$. Consider the language $L = \{a^n b^n | n \geq 0\}$. It is easy to verify that there exists a Petri net $\mathcal{N}$ such that $L_r(\mathcal{N}) = L$. We now prove that $L \notin L_r(LCS)$. Per absurdum, suppose there exists an LCS $\mathcal{F}$ such that $L_c(\mathcal{F}) = L$. For any $k \geq 1$, let $\gamma_k$ and $\gamma'_k$ be two global states s.t. $\gamma_{init}$ leads to $\gamma_k$ by accepting the word $a^k$, $\gamma_k$ leads to $\gamma'_k$ by accepting the word $b^k$, and $\gamma_{acc} \preceq_l \gamma'_k$. Since $\preceq_l$ is a well-quasi ordering, there exists $i < j$ such that $\gamma_i \preceq_l \gamma_j$. By monotonicity of $\mathcal{F}$, we have $\gamma_j$ leads to $\gamma''$ by accepting the word $b^i$ and $\gamma_{acc} \preceq_l \gamma'_i \preceq_l \gamma''$. We conclude that $a^j b^i \in L_c(\mathcal{F})$ with $i < j$, which gives us a contradiction.

We now prove that $L_c(LN) \not\subseteq L_r(\Gamma_1)$, hence $L_c(LCS) \not\subseteq L_r(\Gamma_1)$. Let $\Sigma = \{a, b\}$ and let $L_{par}$ be the language over the alphabet $\Sigma \cup \{\#\}$ that contains all the words $w_1\# \ldots \#w_n$ with $n \geq 0$ such that $w_i \in \Sigma^*$ and there is no prefix of $w_i$ that contains more occurrences of symbol $b$ than those of symbol $a$, for $i : 1 \leq i \leq n$. Notice that the number of occurrences of symbols $a$ and $b$ in $w_i$ may be different. The language can be accepted by a *LN* defined as follows. When we accept the

symbol $a$ we add one token in a special place $p_a$. To accept the symbol $b$, we remove one token from $p_a$. To pass from $w_i$ to $w_{i+1}$, we accept symbol # whenever $p_a$ is empty (in LN the empty test is just a reset).
We now show that $L_{par}$ cannot be recognized by a Petri net with reachability accepting condition. Suppose that there exists a Petri net $\mathcal{N}$ such that $L_r(\mathcal{N}) = L_{par}$. Starting from $\mathcal{N}$, we build a net $\mathcal{N}_1$ by adding a new place $d$ that keeps track of the difference between the number of occurrences of symbols $a$ and $b$ in the prefix of the word that is being processed in $\mathcal{N}$. Furthermore, we add the condition that $d$ is empty to the accepting marking of $\mathcal{N}$. It is easy to verify that $\mathcal{N}_1$ accepts the language $L_{bal}$ consisting of words of the form $w = w_1\# \cdots \# w_n$ where $w_i$ belongs the language of *balanced parentheses* on the alphabet $\Sigma$ for $i : 1 \leq i \leq n$. We exploit now [16, Lemma 9.8] that states that $L_{bal}$ cannot be recognized by a Petri net with reachability accepting condition, which gives us a contradiction.

Finally, the property $L_r(\Gamma_1) = L_r(PN) \subset L_r(\Gamma_0)$ follows from [16, Lemma 9.8] and Proposition 7, Indeed, we have that $L_{bal} \in L_r(\Gamma_0) = RE$ and $L_{bal} \notin L_r(\Gamma_1)$. $\square$

Finally, we observe that we can use an argument similar to that used in the proof of Theorem 7 (part $L_r(\Gamma_1) \not\prec L_r(LCS)$) to show that $L_r(PN) \not\prec L_c(CMRS)$.

### 7.1. Affine well-structured nets

*Affine well-structured nets* (AWSNs) [19] are a generalization of Petri nets with black tokens and *whole-place* operations like reset and transfer arcs [12]. They can also be viewed as a subclass of data nets in which a configuration $s$ is such that $s(d)(p) > 0$ only for a specific data $d$ chosen a priori from $D$. Furthermore, all transitions have arity 1 and we can remove from $F_t$, $H_t$ and $G_t$ all the components in regions different from $S_1$, i.e., $F_t$ and $H_t$ are vectors in $\mathbb{N}^P$ where $P$ is the set of places, and $G_t$ is a matrix in $\mathbb{N}^P \times \mathbb{N}^P$. In the remainder of this section, we see markings $M$ as vectors in $\mathbb{N}^P$. For any place $p$, $M(p)$ gives the number of occurrences of $p$ in $M$. In that case, the order $\leq$ is defined as follows: $M_1 \leq M_2$ iff $M_1(p) \leq M_2(p)$ for all $p \in P$.

An AWSN-transition $t$ is enabled at marking $M$ if $F_t \leq M$. The firing of $t$ at $M$ produces a new marking $M' = (M - F_t)G_t + H_t$. AWSN are well-structured with respect to the order $\leq$.

**Example.** The projection of $F_t$, $H_t$ and $G_t$ in Fig. 1 on $S_1$ (i.e., restricted to the single data $d_1$) gives us the AWSN-transition $t$ with $\alpha_t = 1$ defined as

$$F_t = (1\ 0)\ \ H_t = (0\ 1)\ \ G_t = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$$

This transition removes a token from $p$ and resets the number of tokens in $q$ to 1, i.e., for $M = (m_1, m_2)$ with $m_1 \geq 1$, it yields $M' = (m_1 - 1, 1)$.

We compare now AWSNs and LCSs.

**Theorem 8.** $L_c(AWSN) \subset L_c(LCS)$.

**Proof.** (1) We first prove the inclusion $L_c(AWSN) \subseteq L_c(LCS)$. Assume an AWSN $W$ with the set of places $P = \{p_1, \ldots, p_n\}$. We build an LCS $\mathcal{F} = (Q, C, N, \delta)$ such that $L_c(W) = L_c(\mathcal{F})$. The set of channels is defined as $C = P \cup P'$ where $P'$ (auxiliary channels) contains a primed copy of each element in $P$. The set of messages $N$ contains the symbol $\bullet$ (a representation of a black token).
Assume that $q_0 \in Q$ is the initial state of $\mathcal{F}$. Then, a marking $M = (m_1, \ldots, m_n)$ is encoded as an LCS configuration $enc(M)$ with state $q_0$ and in which channel $p_i \in P$ contains a word $\bullet^{m_i}$ containing $m_i$ occurrences of symbol $\bullet$ for $i \in \overline{n}_0$.
For each transition $t$ with label $\ell$, we need to simulate the three steps (subtraction, multiplication, and addition) that correspond to $F_t$, $G_t$ and $H_t$. Subtraction and addition can be simulated in a straightforward way by removing/adding the necessary number of tokens from/to each channel. The multiplication step is simulated as follows. For each $i \in \overline{n}_0$, we first make a copy of the content of channel $p_i$ in the auxiliary channel $p_i'$. Each copy is defined by repeatedly moving a symbol from $p_i$ to $p_i'$ and terminates when $p_i$ becomes empty. After the copy is terminated for all channels, we start the multiplication step. For each $i \in \overline{n}_0$, we remove a symbol from $p_i'$ and add as many symbol to channel $p_j$ as specified by $G_t(p_i, p_j)$ for $j \in \overline{n}_0$. The analysis terminates when the channels $p_1', \ldots, p_n'$ are all empty. The following properties then hold:

(i) We first notice that $M \leq M'$ iff $enc(M) \preceq_l enc(M')$.
(ii) Furthermore, if $M_0 \overset{w}{\Longrightarrow} M_1$ in $W$, then $enc(M_0) \overset{w}{\Longrightarrow} enc(M_1)$ in $\mathcal{F}$.
(iii) Finally, since $\bullet$ symbols may get lost in $\mathcal{F}$, if $enc(M_0) \overset{w}{\Longrightarrow} enc(M_1)$ then there exists $M_2$ such that $M_0 \overset{w}{\Longrightarrow} M_2$ and $M_1 \leq M_2$.

If the accepting marking is $M_f = (m_1, \ldots, m_k)$ then the accepting LCS configuration contains the control state $q_0$, the channel $p_i \in P$ contains $m_i$ symbols $\bullet$, and the channels $p' \in P'$ are empty. Since we consider languages with coverability acceptance, $L_c(W) = L_c(\mathcal{F})$ immediately follows from properties *(i)*,*(ii)*, *(iii)* and Lemma 1.

(2) We prove now that $L_c(LCS) \not\subseteq L_c(AWSN)$. For this purpose, we exhibit a language in $L_c(LCS)$ and prove that it cannot be recognized by any AWSN.

Fix a finite alphabet $\Sigma = \{a, b, \sharp\}$ and let $\mathcal{L} = \{w\sharp w' |\ w \in \{a, b\}^* \text{ and } w' \preceq_w w\}$. It is easy to define a LCS that accepts the language $L$: we first put $w$ in a lossy channel and then remove one-by-one all of its messages. Thus, we have that $\mathcal{L} \in L_c(LCS)$. We now prove that there is no AWSN that accepts $\mathcal{L}$. Suppose it is not the case and there exists a AWSN $N$, with (say) $n$ places, that recognizes $\mathcal{L}$ with initial marking $M_{init}$ and accepting marking $M_f$.

For each $w \in \{a, b\}^*$, there is a marking $M_w$ such that $M_{init} \stackrel{w\sharp}{\Longrightarrow} M_w \stackrel{w}{\Longrightarrow} M$ and $M_f \leq M$ (otherwise $w\sharp w$ would not be in $L_c(N)$). Consider the sequences $w_0, w_1, w_2, \ldots$ and $M_{w_0}, M_{w_1}, M_{w_2}, \ldots$ of words and markings defined as follows:

- $w_0 := b^n$.
- If $M_{w_i} = (m_1, \ldots, m_n)$ then $w_{i+1} := a^{m_1}\, b\, a^{m_2}\, b\, \cdots\, b\, a^{m_n}$, for $i = 0, 2, \ldots$

We observe that $(a)$ $w_0 \not\preceq_w w_i$ for all $i > 0$, since $w_0$ contains $n$ occurrences of $b$, while $w_i$ contains only $n - 1$ occurrences of $b$; and $(b)$ for any $i < j$, $M_{w_i} \leq M_{w_j}$ iff $w_{i+1} \preceq_w w_{j+1}$. By Dickson's lemma [10], there are $i < j$ such that $M_{w_i} \leq M_{w_j}$. Without loss of generality, we can assume that $j$ is the smallest natural number satisfying this property. Remark that we have that $w_i \not\preceq_w w_j$. Indeed, $w_0 \not\preceq_w w_j$ for any $j > 0$ by $(a)$, and in the case of $i > 0$ we have by $(b)$ that $w_i \not\preceq_w w_j$ since $M_{w_{i-1}} \not\leq M_{w_{j-1}}$. Since $M_{w_i} \leq M_{w_j}$, by monotonicity of AWSNs, we have that $M_{w_i} \stackrel{w_i}{\Longrightarrow} M$ with $M_f \leq M$ implies that $M_{w_j} \stackrel{w_i}{\Longrightarrow} M'$ with $M_f \leq M \leq M'$. Hence, $M_{init} \stackrel{w_j\sharp w_i}{\Longrightarrow} M'$ and $w_j\sharp w_i \in L_c(N) = \mathcal{L}$, which is a contradiction. $\square$

It is interesting to notice that AWSNs can also be simulated by reset nets by using an encoding similar to the one based on LCSs. Indeed, in that encoding the channels are used as counters. The emptiness test on a channel is replaced by a reset on the corresponding place. From this observation and from the results in [3], we have the following classification.

**Proposition 8.** $L_c(PN) \subset L_c(TN) = L_c(RN) = L_c(LN) = L_c(AWSN) \subset L_c(LCS)$.

This result shows that $c$-language recognized by reset/transfer nets are strictly included in those recognized by LCSs.

Finally, we finish the section by reminding that $L_r(TN) = L_r(RN)$ is the class of recursively enumerable languages [9]. Hence, since transfer/reset nets are subclasses of AWSNs, we directly conclude that

$$L_r(AWSNs) = RE.$$

## 8. (Integral) Relational automata

In this section, we compare the class of languages accepted by the fragment $\Gamma_2$ of CMRS defined in Section 3.2 with those accepted by *relational automata* [8].

An *(integral) relational automaton* (RA) operates on a finite set $X$ of positive integer variables, and is of the form $(Q, \delta)$ where $Q$ and $\delta$ are finite sets of control states and transitions, respectively. A transition is a triple $(q_1, op, q_2)$ where $q_1, q_2 \in Q$ and $op$ is of one of the following three operations: (i) *reading*: $read(x)$ reads a new value of variable $x$ (i.e., assigns a non-deterministically chosen value to $x$), (ii) *assignment*: $x := y$ assigns the value of variable $y$ to $x$; (ii) *testing*: $x < y$, $x = y$, $x < c$, $x = c$, and $x > c$ are guards which compare the values of variables $x, y$ and the natural constant $c$. Assume a RA $\mathcal{A} = (Q, \delta)$. A *valuation* $v$ is a mapping form $X$ to $\mathbb{N}$. A *configuration* is of the form $(q, v)$, where $q \in Q$ and $v$ is a valuation. We define $\gamma_{init}$ to be $(q_{init}, v_{init})$ where $q_{init} \in Q$ and $v_{init}(x) = 0$ for all $x \in X$. For a transition $\rho \in \delta$ of the form $(q_1, op, q_2)$, we let $\gamma_1 \stackrel{\rho}{\longrightarrow} \gamma_2$ if and only if $\gamma_1 = (q_1, v_1)$, $\gamma_2 = (q_2, v_2)$, and one of the following holds: $op = read(x)$ and $v_2(y) = v_1(y)$ for each $y \in X - \{x\}$; $op = (y := x)$, $v_2(z) = v_1(z)$ for each $z \in X - \{y\}$, and $v_2(y) = v_1(x)$; $op = (x < y)$, $v_2 = v_1$, and $v_1(x) < v_1(y)$. Other testing operations are defined in a similar manner.

In [8] Čerāns has shown that RA equipped with the *sparser-than* order of tuples of natural numbers are well-structured. The sparser-than order is defined as follows. Let $c_{min}$ (resp. $c_{max}$) be the smallest (resp. largest) constant in the RA $\mathcal{A}$. Let $C$ be the set of integers in the interval $[c_{min}, c_{max}]$. Given two RA configurations $\gamma_1$ and $\gamma_2$, $\gamma_2 = (q_2, v_2)$ is sparser than $\gamma_1 = (q_1, v_1)$, written $\gamma_1 \lhd \gamma_2$, if the following conditions hold:

- $q_1 = q_2$.
- For every $x, y \in X \cup C$,
  - $v_1(x) \leq v_1(y)$ iff $v_2(x) \leq v_2(y)$ for every $x, y \in X \cup C$;
  - $v_1(x) < v_1(y)$ implies $v_1(y) - v_1(x) \leq v_2(y) - v_2(x)$.

For instance, assume that $X = \{x_1, \ldots, x_5\}$, i.e., valuation are 5-tuples, $C = \{0, 1, 2\}$ and $Q$ is a singleton. Then, the valuation $(2, 10, 12, 1994)$ is sparser than $(2, 4, 6, 1000)$, but not sparser than $(1, 10, 12, 1994)$ since the value of the first variable is no longer equal to 2, and not sparser than $(2, 4, 7, 17)$, since the gap between 7 and 4 is larger than the gap between 10 and 12, i.e., $7 - 4 > 12 - 10$.

For RA equipped with the sparser-than order, the coverability accepting condition is equivalent to the control state acceptance, i.e., a word is accepted if it is recognized by an execution ending in a particular control state $q_{fin} \in Q$.

As stated in the following propositions, RA and $\Gamma_2$ define the same class of $c$- and $r$-languages.

**Proposition 9.** $L_c(\Gamma_2) = L_c(RA)$.

**Proof.** Given an RA $\mathcal{A} = (Q, \delta)$ over the set of variables $X$, we can build the $\Gamma_2$ $\mathcal{S}$ defined below. The set of predicate symbols in $\mathcal{S}$ consists of the following: (i) for each $q \in Q$, there is a predicate symbol $q$ in $\mathcal{S}$; and (ii) for each variable $x$ in $X$, there is a predicate symbol $p_x$ in $\mathcal{S}$. Transitions in $\delta$ are encoded via the following CMRS rules (with the same labels)

$$\begin{array}{lll}
(q_1, read(x), q_2) & \Rightarrow & [q_1, p_x(z)] \leadsto [q_2, p_x(w)] : true \\
(q_1, x := y, q_2) & \Rightarrow & [q_1, p_x(z), p_y(w)] \leadsto [q_2, p_x(w), p_y(w)] : true \\
(q_1, x < y, q_2) & \Rightarrow & [q_1, p_x(z), p_y(w)] \leadsto [q_2, p_x(z), p_y(w)] : \{z < w\}
\end{array}$$

We observe now that the sparser-than order of [8] is just a special case of the CMRS ordering $\preceq_c$ in which, for each reachable configuration, the number of bags occurring in  is bounded by the number of variables in $X$ (the number of possible partitioning of the variables in $X$ w.r.t. their current value).

For $X = \{x_1, \ldots, x_n\}$, the initial configuration is $\gamma_{init} = [q_0, p_{x_1}(0), \ldots, p_{x_n}(0)]$. The accepting configuration $\gamma_{fin}$ is the multiset $[q_{fin}]$. It is important to remark that in general for CMRS we cannot determine a priori the number of bags occurring in the index  of every reachable configuration $\gamma$. Thus, the encoding of RA reachability and coverability accepting conditions in $\Gamma_2$ is straightforward.

For the other inclusion, by using Proposition 2, we assume w.l.o.g. that there is no gap order formula $x <_c y$ with $c > 0$ in $\mathcal{S}$. We also observe that we can assume that all configurations of $\mathcal{S}$ have the same size (the size of the initial configuration of the $\Gamma_2$ model). Thus, we associate a variable of $X$ to each ground term of the initial CMRS configuration and compose the predicate symbols in a CMRS configuration to form a single control state. CRMS rules can then be simulated in several steps by operations on variables and updates of control states.

Remember we assume that the accepting configuration of $\mathcal{S}$ is $\gamma_{fin} = [p_{fin}]$. Hence, to each control state containing $p_{fin}$, we add a transition labeled with $\epsilon$ to the accepting control state $q_{fin}$. Those transitions are labeled with either a reading or an assignment operation, hence they can always be followed. $\square$

We now prove that $L_c(\Gamma_2)$ is the class of regular languages. For this purpose, we first need some preliminary definitions. Given a configuration $\gamma$ with

$$index(\gamma) = D_0 \ldots D_{cmax} d_0 B_0 \ldots d_n B_n$$

we define

$$index'(\gamma) = D_0 \ldots D_{cmax} \sharp B_0 \ldots B_n$$

Let us now consider a $\Gamma_2$ specification $\mathcal{S}$ with an initial ($cmax$-bounded) configuration $\gamma_{init}$ and a final ($cmax$-bounded) configuration $\gamma_{fin} = [p_{fin}]$. The symbolic graph $\mathcal{G}_\mathcal{S}$ associated to $\mathcal{S}$ is an automaton $(V, \rightarrow_{\mathcal{G}_\mathcal{S}}, c_0, F)$ where

- $V = \{index'(\gamma) \mid |\gamma| \le |\gamma_{init}|\}$;
- $\rightarrow_{\mathcal{G}_\mathcal{S}} \subseteq V \times \mathcal{S} \times V$ such that $\forall c_1, c_2 \in V : (c_1, \rho, c_2) \in \rightarrow_{\mathcal{G}_\mathcal{S}}$ iff there exist two configurations $\gamma_1$ and $\gamma_2$ such that $index'(\gamma_1) = c_1$, $index'(\gamma_2) = c_2$ and $\gamma_1 \xrightarrow{\rho} \gamma_2$;
- $c_0 = index'(\gamma_{init})$;
- $F = \{index'(\gamma) \in V \mid \gamma_{fin} \preceq_c \gamma\}$.

We easily see that $\mathcal{G}_\mathcal{S}$ is a finite automata since the number of predicate symbols that appears in states is bounded by the size of $\gamma_{init}$.

In the following, we use $c \xrightarrow{\rho}_{\mathcal{G}_\mathcal{S}} c'$ to denote that $(c, \rho, c') \in \rightarrow_{\mathcal{G}_\mathcal{S}}$.

Moreover, given a sequence of rules $w = \rho_1 \ldots \rho_l$, $c \xrightarrow{w}_{\mathcal{G}_\mathcal{S}} c'$ denotes that there exists $c_1, \ldots, c_{l-1}$ in $V$ such that $c \xrightarrow{\rho_1}_{\mathcal{G}_\mathcal{S}} c_1 \ldots \xrightarrow{\rho_l}_{\mathcal{G}_\mathcal{S}} c_{l-1} \xrightarrow{\rho_l}_{\mathcal{G}_\mathcal{S}} c'$ and $w = \rho_1 \cdots \rho_{l+1}$.

The next lemma states the main property of $\mathcal{G}_\mathcal{S}$: all the executions of $\mathcal{G}_\mathcal{S}$ corresponds to an execution in $\mathcal{S}$ starting from $\gamma_{init}$.

**Lemma 4.** *If* $index'(\gamma_{init}) \xrightarrow{w}_{\mathcal{G}_\mathcal{S}} c$*, then* $\gamma_{init} \xrightarrow{w} \gamma$ *such that* $index'(\gamma) = c$.

**Proof.** The proof is by induction on the number of transitions to reach $c$.
$n = 0$: Immediate.
$n > 0$: Let

$$index'(\gamma_{init}) \xrightarrow{w}_{\mathcal{G}_\mathcal{S}} c \xrightarrow{\rho}_{\mathcal{G}_\mathcal{S}} c'$$

and suppose that $c = D_0 \ldots D_{cmax} \sharp B_0 \ldots B_n$ and $c' = D'_0 \ldots D'_{cmax} \sharp B'_0 \ldots B'_{n'}$. By ind. hypothesis, we have $\gamma_{init} \xrightarrow{w} \gamma$ with $index'(\gamma) = c$.
Since $c \xrightarrow{\rho}_{\mathcal{G}_\mathcal{S}} c'$, we know from definition of symbolic graph that there exist two configurations $\gamma_1, \gamma_2$ such that $index'(\gamma_1) = c$, $index'(\gamma_2) = c'$ and $\gamma_1 \xrightarrow{\rho} \gamma_2$.
Since

$$index'(\gamma) = index'(\gamma_1) = D_0 \ldots D_{cmax} \sharp B_0 \ldots B_n$$

we have

$$index(\gamma) = D_0 \ldots D_{cmax} d_0 B_0 \ldots d_n B_n$$

and

$$index(\gamma_1) = D_0 \ldots D_{cmax} d'_0 B_0 \ldots d'_n B_n$$

Moreover, suppose that

$$index(\gamma_2) = D'_0 \ldots D'_{cmax} b_0 B'_0 \ldots b_{n'} B_{n'}$$

Following Proposition 2, we have $\gamma_{init} \xrightarrow{w} \gamma'$ such that

$$index(\gamma') = D_0 \ldots D_{cmax} d''_0 B_0 \ldots d''_n B_n$$

and for any $i : 0 \leq i \leq n\; d''_i \geq d_i + d'_i$.
Since CMRS are monotonic (so do $\Gamma_0$), $\gamma_1 \xrightarrow{\rho} \gamma_2$ and $\gamma_1 \preceq_c \gamma'$, we have that $\gamma' \xrightarrow{\rho} \gamma''$ with $\gamma_2 \preceq_c \gamma''$. Since the number of ground terms in configurations is bounded, $\gamma_2 \preceq_c \gamma''$ implies $\gamma_2 \prec \gamma''$. Thus, we have that $index'(\gamma'') = index'(\gamma_2) = c'$. □

**Theorem 9.** $L_c(\Gamma_2) =$ *Regular Languages.*

**Proof.** We first show how to encode a finite automata in $\Gamma_2$. The encoding of a finite automaton is direct: each state corresponds to a nullary predicate and CMRS rules mimic the transition relation. Acceptance of words is simulated as follows: for any final state $c$ we have a rule $\{c\} \rightsquigarrow \{p_{fin}\} : true$ labeled with $\epsilon$ and the final configuration is $\{p_{fin}\}$. Finally, the initial configuration is $\{c_0\}$ where $c_0$ is the initial state of the automaton.

We now show that all the $c$-languages accepted by a $\Gamma_2$ are regular. Consider a $\Gamma_2$ $\mathcal{S}$ with an initial ($cmax$-bounded) configuration $\gamma_{init}$ and the final ($cmax$-bounded) final configuration $\gamma_{fin} = [p_{fin}]$. From Lemma 4 we have that a word accepted by the symbolic graph $\mathcal{G}_\mathcal{S}$ corresponds to a sequence of rules corresponding to a word accepted by $\mathcal{S}$ (following definition of $\mathcal{G}_\mathcal{S}$, $\gamma_{fin} \preceq_c \gamma$ iff $index'(\gamma) \in F$). Moreover, from the definition of $\mathcal{G}_\mathcal{S}$ we have

$$\gamma_{init} \xrightarrow{\rho_1} \gamma_1 \xrightarrow{\rho_2} \ldots \xrightarrow{\rho_{l-1}} \gamma_l$$

implies that

$$index'(\gamma_{init}) \xrightarrow{\rho_1}_{\mathcal{G}_\mathcal{S}} index'(\gamma_1) \xrightarrow{\rho_2}_{\mathcal{G}_\mathcal{S}} \ldots \xrightarrow{\rho_{l-1}}_{\mathcal{G}_\mathcal{S}} index'(\gamma_l)$$

by definition of $\mathcal{G}_\mathcal{S}$. Furthermore, from definition of accepting states $F$, $\gamma_{fin} \preceq_c \gamma_l$ if and only if $index'(\gamma_l) \in F$. Hence, if we replace symbols $\rho$ in $\mathcal{G}_\mathcal{S}$ by $\lambda(\rho)$ we conclude that a words $w$ is accepted by $\mathcal{S}$ if and only if $w$ is accepted by $\mathcal{G}_\mathcal{S}$. □

We are ready now to compare $\Gamma_2$ (hence RA) with the other models studied in this paper. For this purpose, we first observe that Petri nets can accept regular languages (finite automata can be encoded as Petri nets). Furthermore, it is straightforward to build a Petri net that accepts a non-regular language like $L = \{a^n \# b^m | n \geq m\}$. As a consequence of this observation and of Theorem 9, we have the following result.

**Corollary 2.** $L_c(\Gamma_2) \subset L_c(\Gamma_1)$.

Let us now consider the reachability accepting condition. We first notice that $L_c(\Gamma_2) = L_r(\Gamma_2) = L_c(RA) = L_r(RA)$. Indeed, in both cases of $\Gamma_2$ and *RA* we can encode the reachability acceptance into the coverability acceptance by adding transitions (labeled with $\epsilon$) that can be fired only from the accepting configuration and leads to a configuration with control state $q_{fin}$ in the case of *RA* and a configuration containing a special accepting predicate symbol $p_{fin}$ in the case of $\Gamma_2$. Furthermore, reduce the coverability acceptance to reachability acceptance is straightforward. Indeed, for RA it suffices to add a mechanism that sets all the counters to 0 once an accepting configuration (for coverability) is reached. In the case of $\Gamma_2$, it suffices to add a mechanism to remove all the terms but $p_{fin}$ once an accepting configuration is reached. Thus, we have the following property.

**Theorem 10.** $L_r(\Gamma_2) \subset L_r(\Gamma_1)$.

## 9. Conclusions

In this paper, we have compared wsts by using languages with coverability acceptance and reachability acceptance as a measure of their expressiveness. From our results we obtain the following classification for coverability acceptance:

$$\begin{pmatrix} L_c(FA) \\ = \\ L_c(RA) \\ = \\ \textit{regular} \\ \textit{languages} \end{pmatrix} \subset L_c(PN) \subset \begin{pmatrix} L_c(aWSN) \\ = \\ L_c(TN) \\ = \\ L_c(RN) \\ = \\ L_c(LN) \end{pmatrix} \subset L_c(LCS) \subset \begin{pmatrix} L_c(CMRS) \\ = \\ L_c(\textit{data nets}) \end{pmatrix} \subset RE$$

Furthermore, since CMRS and Petri data nets (data nets without whole-place operations) recognize the same class of *c*-languages (coverability in CMRS can be reduced to coverability in Petri data nets [20]) we have that data nets, Petri data nets, and transfer data nets (another subclass of data nets with restrictions on the type of transfers) all define the same class of *c*-languages as CMRS, i.e.,

$$L_c(\textit{data nets}) = L_c(\textit{Petri data nets}) = L_c(\textit{transfer data nets}) = L_c(CMRS)$$

When considering the reachability acceptance, the picture changes and becomes:

$$\begin{pmatrix} L_r(FA) \\ = \\ L_r(RA) \\ = \\ \textit{regular} \\ \textit{languages} \end{pmatrix} \subset \begin{array}{c} L_r(PN) \\ \not\subset \\ L_r(LN) \subset L_r(LCS) \end{array} \subset \begin{pmatrix} L_r(aWSN) \\ = \\ L_r(TN) \\ = \\ L_r(RN) \\ = \\ L_r(CMRS) \\ = \\ L_r(\textit{data nets}) \\ = \\ RE \end{pmatrix}$$

Finally, with the two previous pictures we can also compare classes of languages obtained with coverability acceptance and with reachability acceptance. Beside the results we summarized herebefore, we also obtained three results that make the picture of comparisons between classes of languages complete. First, some models recognize the same class of languages with the two accepting conditions we consider in this paper. More precisely, we have that $L_c(LN) = L_r(LN)$ and $L_c(LCS) = L_r(LCS)$. We also know that $L_c(PN) \subset L_r(PN)$. Finally, we obtained as result that the class $L_r(PN)$ is incomparable with all the classes of languages with coverability acceptance between $L_c(LN)$ and $L_c(CMRS)$.

## Appendix A. Technical lemmas and propositions

### *Appendix A.1. Proof of Proposition 2*

The proof is by induction on the size $k$ of the sequence of rules $\rho_1 \dots \rho_k$.

**Base case:** ($k = 0$) Since we assume that all the constants that appear in $\gamma_{init}$ are lesser or equal than *cmax*, we have that $index(\gamma_{init})$ is of the form $D_0 \dots D_{cmax}$. Hence, the lemma trivially holds.

**Induction step:** ($k > 0$) Suppose that we have $\gamma_{init} \xrightarrow{\rho_1 \dots \rho_{k-1}} \gamma_1 \xrightarrow{\rho_k} \gamma$ with

$$index(\gamma_1) = E_0 \dots E_{cmax} e_0 F_0 \dots e_m F_m$$

Suppose that $\gamma$ is built from $\gamma_1$ by applying the instance $\rho = L_1 \rightsquigarrow R_1$ of $\rho_k = L \rightsquigarrow R : \psi$. This means that there exists a multi-set of ground terms $\eta$ such that $\gamma_1 = L_1 + \eta$ and $\gamma = R_1 + \eta$.
Under this hypothesis, the multisets in $index(\gamma_1)$ satisfy the following conditions:

- For any $i : 0 \le i \le cmax$, we have that $E_i = G_i + E_i^L$ where
  - $E_i^L$ is the maximal (possibly empty) multiset of predicate symbols with parameter equal to $i$ that occur in $L_1$.
  - $G_i$ is the maximal multiset of predicates with parameter $i$ that are not consumed by $\rho$ (i.e., they also occur in $index(\gamma)$).
- For any $i : 0 \le i \le m$ and given $v_i = cmax + \Sigma_{0 \le j \le i} e_j$, we have that $F_i = H_i + F_i^L$ where
  - $F_i^L$ is the maximal (possibly empty) multiset of predicate symbols with parameter equal to $v_i$ that occur in $L_1$.
  - $H_i$ is the maximal multiset of predicates with parameter $v_i$ that are not consumed by $\rho$ (i.e., they also occur in $index(\gamma)$).

Let us now suppose that instead of removing $L_1$ from $\gamma_1$, we add $R_1$ to $\gamma_1$. The resulting configuration $\gamma_2 = \gamma_1 + R_1$ has index

$$index(\gamma_2) = E'_0 \dots E'_{cmax} \sigma_0 c_0 F'_0 \dots \sigma_m c_m F'_m \sigma_{m+1}$$

with

$$\forall 0 \le i \le m+1 : \sigma_i = c_0^i K_0^i \dots c_{n_i}^i K_{n_i}^i$$

where (assuming $\forall 0 \le k \le m : e'_k = c_k + \Sigma_{0 \le j \le n_k} c_j^k$).

- For any $i : 0 \le i \le cmax$, we have that

  $$E'_i = G_i + E_i^L + E_i^R$$

  where $E_i^R$ is the maximal (possibly empty) multiset of predicate symbols with parameter equal to $i$ that occur in $R_1$;
- for any $i : 0 \le i \le m+1$, for any $j : 0 \le j \le n_i$, $K_j^i$ is the maximal (possibly empty) multiset of predicate symbols that occur in $R_1$ with parameter equal to $cmax + \sum_{0 \le k < i} c_k + \sum_{0 \le k \le j} c_k^i$.
- Furthermore, for any $i : 0 \le i \le m$ :

  $$F'_i = G_i + F_i^L + F_i^R$$

  where $F_i^R$ is the maximal (possibly empty) multiset of predicate symbols with parameter equal to $cmax + \Sigma_{0 \le j \le i} e'_i$ that occur in $R_1$.

Intuitively, $\sigma_i$ represent the structure added to $index(\gamma_1)$ by $R_1$ for what concerned all predicate symbols with a parameter $v$ not directly represented in $index(\gamma_1)$, i.e., such that

$$cmax + \Sigma_{0 \le j \le i-1} e_j < v < cmax + \Sigma_{0 \le j \le i} e_j.$$

The sequence

$$\varsigma = \left(E_0^L + E_0^R\right) \dots \left(E_{cmax}^L + E_{cmax}^R\right) \sigma_0 c_0 \left(F_0^L + F_0^R\right) \dots \sigma_m c_m \left(F_m^L + F_m^R\right) \sigma_{m+1}$$

can be transformed into $index(L_1 + R_1)$ by removing all empty multisets and summing up constants in order to correctly maintain gaps between non-empty multisets of predicates.

To simplify the presentation, let us assume that $\varsigma$ coincides with $index(L_1 + R_1)$. The following can be easily extended to the general case. We now observe that $\rho$ corresponds to an instance of a specialization $\rho'$ of $\rho_k$ in which the variables in $\rho_k$ are totally ordered w.r.t. $< \cup =$. In other words, from $\varsigma$ we can reconstruct the constraint $\psi'$ of $\rho'$ as follows:

- To each non-empty multiset $M$ in $\varsigma$ we associate a distinct variable $x_M$, each predicate in $M$ takes $x_M$ as parameter in $\rho'$.
- For each non empty multiset $M = E_i^L + E_i^R$ with $0 \leq i \leq cmax$, we associate the condition $x_M = i$ in $\psi'$.
- If $M$ and $M'$ are two consecutive multisets in $\varsigma$ (and $M$ occurs before $M'$) with a constant $c$ between them then $\rho'$ contains the gap-order constraint $x_M <_c x_{M'}$.

Since the condition $\psi'$ of $\rho'$ corresponds to one of the possible linearizations of the condition of $\rho_k$, every instance of $\rho'$ is also an instance of $\rho_k$. Furthermore, $\psi'$ in $\rho'$ represents the minimal gap-order constraints extracted from $\varsigma$ which is compatible with $\psi$ (i.e., $\psi \wedge \psi' \equiv \psi'$). This implies that any other instance $L_2 \rightsquigarrow R_2$ of $\rho'$ can be represented by a sequence

$$\varsigma_1 = \left(E_0^L + E_0^R\right) \ldots \left(E_{cmax}^L + E_{cmax}^R\right) \sigma_0' f_0 \left(F_0^L + F_0^R\right) \ldots \sigma_m' f_m \left(F_m^L + F_m^R\right) \sigma_{m+1}'$$

where the following conditions are satisfied:

- for any $i : 0 \leq i \leq m$ we have $f_i \geq c_i$,
- for any $i : 0 \leq i \leq m+1$, $\sigma_i' = f_0^i K_0^i \ldots f_{n_i}^i K_{n_i}^i$ such that $f_j^i \geq c_j^i$ for all $j : 0 \leq j \leq n_i$.

Fixed a given instance $L_2 \rightsquigarrow R_2$ of $\rho'$ with associated sequence $\varsigma_1$, we define the new sequence

$$\varsigma_2 = S_0 \ldots S_{cmax} \sigma_0' f_0 T_0 \ldots \sigma_m' f_m T_m \sigma_{m+1}$$

with $S_i = E_i^L + E_i^R + G_i$ for $i : 0 \leq i \leq cmax$ and $T_i = F_i^L + F_i^R + H_i$ for $i : 0 \leq i \leq m$.
Following the definition of *index* and $\prec$, this sequence corresponds to the index of a configuration $\gamma_2'$ such that $\gamma + R_1 = \gamma_2 \prec \gamma_2'$. Now let us define the values

$$e_i'' = f_i + \Sigma_{0 \leq j \leq n_i} f_j^i$$

for $i : 0 \leq i \leq m$. Furthermore, let us define the sequence

$$\varsigma_3 = \left(E_0^L + G_0\right) \ldots \left(E_{cmax}^L + G_0\right) e_0'' \left(F_0^L + H_0\right) \ldots e_m'' \left(F_m^L + H_m\right)$$

Again, following definition of *index* and $\prec$, there exists $\gamma_1'$ such that $\varsigma_3 = index(\gamma_1')$ and $\gamma_1 \prec \gamma_1'$.

Now we note that $\gamma_2'$ corresponds to $\gamma_1' + R_2$. This implies that the instance $L_2 \rightsquigarrow R_2$ of $\rho$ can be applied at $\gamma_1'$.

If we now define $\gamma = \gamma_2 - L_1$ and $\gamma'' = \gamma_2' - L_2$, then we have $\gamma \prec \gamma''$. Indeed, $index(\gamma)$ and $index(\gamma'')$ are obtained by removing predicate symbols in multiset occurring in the same position in $index(\gamma_2)$ and $index(\gamma_2')$, respectively. Furthermore, we have that $\gamma_1' \xrightarrow{\rho'} \gamma''$. Finally, note that for any sequence $\varsigma_1$ there exists an instance $\rho''$ of $\rho'$ (the specialization of $\rho_k$ we consider). Hence, there exists an instance $\rho''$ such that $\gamma'' = \gamma'$.

By applying the inductive hypothesis, we have that there exists an execution $\gamma_{init} \xrightarrow{\rho_1 \ldots \rho_{k-1}} \gamma_1'$ that subsumes $\gamma_{init} \xrightarrow{\rho_1 \ldots \rho_{k-1}} \gamma_1$ such that $\gamma_1' \xrightarrow{\rho_k} \gamma'$. We conclude that there exists an execution $\gamma_{init} \xrightarrow{\rho_1 \ldots \rho_k} \gamma'$ that subsumes $\gamma_{init} \xrightarrow{\rho_1 \ldots \rho_k} \gamma$. $\square$

*Appendix A.2. Proof of Lemma 2*

If the length of the execution is one the thesis trivially holds because $\gamma_{init}$ is *cmax*-bounded. Now suppose that

$$\gamma_{init} \xrightarrow{\rho_1} \gamma_1 \cdots \xrightarrow{\rho_k} \gamma_k \xrightarrow{\rho} \gamma$$

and $\gamma$ is not linear. Suppose that $L_1 \rightsquigarrow R_1$ is the instance of $\rho$ applied to $\gamma_k$ to obtain $\gamma$. As in the proof of Proposition 2, we have that

$$index(\gamma_k) = E_0 \ldots E_{cmax} e_0 \left(F_0^L + G_0\right) \ldots e_m \left(F_m^L + G_m\right)$$

where $F_i^L$ are the predicate symbols of terms in $L_1$ with parameter $cmax + \Sigma_{0\leq j\leq i} e_j$ for $i : 0 \leq i \leq m$.
Now let us consider $\gamma_k + R_1$. Then we have that

$$index(\gamma_k + R_1) = E'_0 \ldots E'_{cmax}\sigma_0 c_0 T_0 \ldots \sigma_m c_m T_m \sigma_{m+1}$$

where

- $E'_i = E_i + E_i^R$ where $E_i^R$ are the predicate symbols of terms in $R_1$ with parameter $i$ for $i : 0 \leq i \leq cmax$;
- $T_i = F_i^L + F_i^R + G_i$ where $F_i^R$ are the predicate symbols of terms in $R_1$ with parameter $cmax + \Sigma_{0\leq j\leq i} e_j$ for $i : 0 \leq i \leq m$; and
- $\sigma_i$ is a sub-sequence

  $$c_0^i K_0^i \ldots c_{n_i}^i K_{n_i}^i$$

  that represents terms with new values added by $R_1$ for $i : 0 \leq i \leq m + 1$.

Since $\gamma$ is not linear, there is a multiset $F_r^R + G_r$ or a multiset $K_j^i$ that contains at least two predicates $p$ and $q$ with the same parameter say $v$. Let us suppose such a multiset $F_r^R + G_r$. The case where a multi-set $K_j^i$ contains at least two predicates is treated is a similar way. Since by hypothesis $F_r^L + G_r$ contains at most one symbol and by the syntactic restriction of $\Gamma_0$, we have that (at least) one between $p(v)$ and $q(v)$ is produced by a valuation to a variable in $\rho$ which is not involved in $=$ constraints.

Following Proposition 2, for any $\gamma'_k$ with $\gamma_k \prec \gamma'_k$ we know that there exists a linear execution from $\gamma_{init}$ to $\gamma'_k$ with the same rules $\rho_1, \ldots, \rho_k$ and passing through $\gamma'_1, \ldots, \gamma'_{k-1}$ such that $\gamma_i \prec \gamma'_i$ for $i : 1 \leq i \leq k$. This implies that we can choose $\gamma'_k$ and instance $L_2 \rightsquigarrow R_2$ of $\rho_k$ such that

$$index\left(\gamma'_k + R_2\right) = E'_0 \ldots E'_{cmax}\sigma_0 c_0 T_0 \ldots \sigma_r c_r T_r \sigma'_{r+1} c'_{r+1} T_{r+1} \ldots \sigma'_m c'_m T_m \sigma_{m'+1}$$

where for any $j > r$ $c'_j$ and the constants in $\sigma'_j$ are strictly greater than $c_j$ and the values in $\sigma_j$, respectively (i.e., we "shift to the right" all values greater than $v$).
Now notice that in a $\Gamma_0$ rule it is not possible to impose the equalities over more than two parameters. Furthermore, when imposing equality of two parameters of ground terms, one ground term is removed by the rule and the second one is added to configurations by the rule. Hence, there is no constraints that impose that the parameter of $p(v)$ and $q(v)$ must be equal. W.l.o.g. we assume that there is no constraint that impose that the parameter $v$ of $p(v)$ must be equal to another parameter. This means that $\rho$ remains applicable to $\gamma'_k$ whenever the evaluation for the argument of predicate $p$ is the value $v' = v + 1$. With this new instance $L_3 \rightsquigarrow R_3$ of $\rho$ we have that

$$index(\gamma'_k + R_3) = E'_0 \ldots E'_{cmax}\sigma_0 c_0 T_0 \ldots \sigma_r c_r T_r 1[p]\sigma''_{r+1} c'_{r+1} T_{r+1} \ldots \sigma'_m c'_m T_m \sigma_{m'+1}$$

where $\sigma''_{r+1}$ is obtained from $\sigma'_{r+1}$ by decrementing by 1 the first constant that appears.

We conclude by noticing that from $\gamma'_k + R_3$ we can compute $\gamma'$ by removing $L_3$. This operation maintains the same structure of the index of $\gamma'_k + R_3$ for what concerns predicate $[p]$. Hence, assuming that $d_1, \ldots, d_n$ are the constants of $index(\gamma)$ and $v = cmax + \sum_{j=1..i} d_i$ we have that

$$index(\gamma') = D_0 \ldots D_{cmax} d_0 B_0 \ldots d_i B_i 1[p] d'_{i+1} B_{i+1} \ldots d'_n B_n$$

such that $\forall i + 1 \leq j \leq n : d'_j \geq d_j$. □

*Appendix A.3. Proof of Lemma 3*

We prove by induction on the number of transitions that $\gamma_1 \xrightarrow{\rho_1 \ldots \rho_k} \gamma_3$ implies there exists $\gamma_4$ such that $\gamma_2 \xrightarrow{\rho_1 \ldots \rho_k} \gamma_4$ with either $\gamma_3 \prec \gamma_4$ or $\gamma_4$ is a linearisation of $\gamma_3$.

**Base case:** ($k = 1$) Suppose that the lemma does not hold. Let $L_1 \rightsquigarrow R_1$ be the instance of $\rho_1$ that allows to build $\gamma_3$ from $\gamma_1$, i.e., $\gamma_3 = \gamma_1 - L_1 + R_1$. Suppose that $\gamma_1 = \gamma'_1 + [p(v)]$, $\gamma_2 = \gamma'_2 + [p(v')]$ and $\gamma'_1 \prec \gamma'_2$. In other words, the predicate $p$ is "isolated" in the index of $\gamma_2$. We consider two cases: either $L_1 \leq \gamma'_1$ or not, i.e., the instance $L_1 \rightsquigarrow R_1$ does not remove $p(v)$ or it does.

In the case of $L_1 \leq \gamma'_1$, let

$$index(L_1 + R_1) = E_1 \ldots E_{cmax} e_0 H_0 \ldots e_r H_r$$

To any sequence,

$$\varsigma = E_1 \dots E_{cmax} e'_0 H_0 \dots e'_r H_r$$

such that $\forall 0 \leq i \leq r : e'_i \geq e_i$ correspond another instance $L_2 \leadsto R_2$ of $\rho_1$. Furthermore, since $\gamma_2$ is a linearization of $\gamma_1$, all the constants that appear in $index(\gamma_2)$ are greater than the corresponding ones in $index(\gamma_1)$. Hence, there exists an instance $L_3 \leadsto R_3$ of $\rho_1$ that has the same effect on the structure of $index(\gamma_2)$ than the instance $L_1 \leadsto R_1$ on $index(\gamma_1)$, i.e., predicates are removed from and added to the same multi-sets and the same sequences of multi-sets (interleaved with constants) are added at the same point into $index(\gamma_2)$. Hence, $\gamma_2 \xrightarrow{\rho_1} \gamma_4 = \gamma_2 - L_3 + R_3$ and $\gamma_4$ is a linearisation of $\gamma_3$.

In the second case, i.e., $L_1 \not\preceq \gamma'_1$ and $p(v)$ is removed from $\gamma_1$ when applying $L_1 \leadsto R_1$, let

$$index(L_1 + R_1) = E_1 \dots E_{cmax} e_0 H_0 \dots e_i H_i + [p] e_{i+1} H_{i+1} \dots e_r H_r$$

such that $v = cmax + \sum_{j=0..i} e_i$. Following the syntactic restriction of $\Gamma_0$, either ($i$) $\rho_1$ imposes no equality constraint between the parameter of $p(v)$ and the parameter of another ground term $q(v')$, or ($ii$) $\rho_1$ imposes such an equality constraint on the parameters of $p(v)$ and $q(v')$ which is added by $\rho_1$. In case ($i$), the sequence

$$\varsigma = E_1 \dots E_{cmax} e_0 H_0 \dots e_i H_i 1[p] e_{i+1} H_{i+1} \dots e_r H_r$$

corresponds to another instance $L_2 \leadsto R_2$ of $\rho_1$, i.e., $index(L_2 + R_2) = \varsigma$, since the gap-orders between predicates defined by $L_1 + R_1$ are not violated. Furthermore, if we increase the gap orders defined by $\varsigma$ we still obtain a sequence that corresponds to an instance of $\rho_1$. Hence, any sequence

$$\varsigma' = E_1 \dots E_{cmax} e'_0 H_0 \dots e'_i H_i e[p] e'_{i+1} H_{i+1} \dots e'_r H_r$$

such that $e \geq 1$ and $e'_i \geq e_i$ for any $i : 0 \leq i \leq r$ corresponds to an instance of $\rho_1$. Furthermore, since $\gamma_2$ is a linearisation of $\gamma_1$, all the constants that appear in $index(\gamma_2)$ are greater than the corresponding ones in $index(\gamma_1)$. Hence, there exists a instance $L_3 \leadsto R_3$ of $\rho_1$ that has the same effect on the structure of $index(\gamma_2)$ than $L_1 \leadsto R_1$ on $index(\gamma_1)$, except that $L_3 \leadsto R_3$ removes the multi-set $[p]$ that corresponds to the ground term $p(v)$. We conclude that $\gamma_2 \xrightarrow{\rho_1} \gamma_4 = \gamma_2 - L_3 + R_3$ and $\gamma_3 \prec \gamma_4$.

In case ($ii$), the sequence

$$\varsigma = E_1 \dots E_{cmax} e_0 H_0 \dots e_i H_i 1[p, q] e_{i+1} H_{i+1} \dots e_r H_r$$

corresponds to another instance $L_2 \leadsto R_2$ of $\rho_1$, i.e., $index(L_2 + R_2) = \varsigma$, since the gap-orders (and equality between parameters of $p$ and $q$) defined by $L_1 + R_1$ are not violated. Again, any sequence

$$\varsigma' = E_1 \dots E_{cmax} e'_0 H_0 \dots e'_i H_i e[p, q] e'_{i+1} H_{i+1} \dots e'_r H_r$$

such that $e \geq 1$ and $e'_i \geq e_i$ for any $i : 0 \leq i \leq r$ corresponds to an instance of $\rho_1$. Furthermore, since $\gamma_2$ is a linearisation of $\gamma_1$, all the constants that appear in $index(\gamma_2)$ are greater than the corresponding ones in $index(\gamma_1)$. Hence, there exists a instance $L_3 \leadsto R_3$ of $\rho_1$ that has the same effect on the structure of $index(\gamma_2)$ than $L_1 \leadsto R_1$ on $index(\gamma_1)$, except that $L_3 \leadsto R_3$ replaces the multi-set $[p]$ that corresponds to the ground terms $p(v')$ by the multi-set $[q]$. We conclude that $\gamma_2 \xrightarrow{\rho_1} \gamma_4 = \gamma_2 - L_3 + R_3$ and $\gamma_4$ is a linearisation of $\gamma_3$.

**Induction Step:** ($k > 1$) By induction Hypothesis, we have that $\gamma_1 \xrightarrow{\rho_1 \dots \rho_{k-1}} \gamma'_3$ implies that there exists $\gamma'_4$ such that $\gamma_2 \xrightarrow{\rho_1 \dots \rho_{k-1}} \gamma'_4$ with either $\gamma'_3 \prec \gamma'_4$ or $\gamma'_4$ is a linearisation of $\gamma'_3$.

In the case where $\gamma'_4$ is a linearisation of $\gamma'_3$, we apply the same reasoning than in the base base.

In the case where $\gamma'_3 \prec \gamma'_4$, let $\gamma'_3 \xrightarrow{\rho_k} \gamma_3$ and $L_1 \leadsto R_1$ be the instance of $\rho_k$ used to build $\gamma_3$ from $\gamma'_3$, i.e., $\gamma_3 = \gamma'_3 - L_1 + R_3$. Consider

$$index(L_1 + R_1) = E_1 \dots E_{cmax} e_0 H_0 \dots e_r E_r$$

To any sequence

$$\varsigma = E_1 \dots E_{cmax} e'_0 H_0 \dots e'_r E_r$$

with $\forall 0 \le i \le r : e_i \le e'_i$ corresponds to another instance of $\rho_k$ where the gap-orders between parameters of ground terms increase. Hence, there exists an instance $L_2 \rightsquigarrow R_2$ of $\rho_k$ such that $\gamma_4 = \gamma'_4 - L_2 + R_2$ and $\gamma_3 \prec \gamma_4$. $\square$

## Appendix B. From Data nets to CMRS

See Figs. B.1–B.8.

$$
\begin{array}{l}
\textit{For } k = \alpha_t,\ i \in \{0, \ldots, k\},\ \textit{and any } p \in P:\\
\textit{Copy of indexes in } \alpha_t\\
[\iota_0(x_0), \ldots, \iota_{k+1}(x_{k+1}), new_t] \rightsquigarrow\\
\quad [\iota_0(x_0), \ldots, \iota_{k+1}(x_{k+1}), \jmath_0(x'_0), \ldots, \jmath_{k+1}(x'_{k+1}), \uparrow(x_0), \Uparrow(x'_0)] :\\
\quad\quad x_{k+1} < x'_0 < \ldots < x'_{k+1}\\
\textit{Copy } p \textit{ to } \overline{p} \textit{ for } \alpha_t - \textit{indexes}\\
[\uparrow(x), \iota_i(x), \Uparrow(y), \jmath_i(y), p(x)] \rightsquigarrow [\uparrow(x), \iota_i(x), \Uparrow(y), \jmath_i(y), \overline{p}(y)] : true\\
\textit{Copy } p \textit{ to } \overline{p} \textit{ for region} - \textit{indexes}\\
[\iota_i(x), \uparrow(u), p(u), \iota_{i+1}(x'), \jmath_i(y), \Uparrow(v), \jmath_{i+1}(y')] \rightsquigarrow\\
\quad [\iota_i(x), \uparrow(u), \iota_{i+1}(x'), \jmath_i(y), \overline{p}(v), \Uparrow(v), \jmath_{i+1}(y')] : x < u < x', y < v < y'\\
\textit{Move pointers to the right}\\
[\uparrow(u), p(u'), \iota_{k+1}(x), \Uparrow(v), \jmath_{k+1}(y)] \rightsquigarrow\\
\quad [\uparrow(u'), \iota_{k+1}(x), \Uparrow(v'), p(v'), \checkmark(v'), \jmath_{k+1}(y)] : u < u' < x, v < v' < y\\
\textit{Terminate copy, replace current conf with new one}\\
[\iota_0(f), \iota_1(x_1), ...., \iota_k(x_k), \iota_{k+1}(l), \jmath_0(f'), \jmath_1(x'_1), ...., \jmath_k(x'_k), \jmath_{k+1}(l'), \uparrow(u), \Uparrow(v)] \rightsquigarrow\\
\quad [\iota_0(f'), \iota_1(x'_1), ...., \iota_k(x'_k), \iota_{k+1}(l'), tr_t]\ :\ true
\end{array}
$$

**Fig. B.1**. Silent CMRS rules for $new_t$: generation of a new configuration with $\overline{p}$-terms.

$$
\begin{array}{l}
\textit{For } k = \alpha_t,\ i \in \{1, \ldots, k\}, j \in \{0, \ldots, k\},\ \textit{and } p \in P:\\
\textit{Start from first index}\\
[tr_t] \rightsquigarrow [tr_{t,1}]\ :\ true\\
\textit{Select a token from an index in } \alpha_t,\ \textit{apply } G_t \textit{ to other indexes}:\\
[\iota_0(x_0), \iota_1(x_1), \ldots, \iota_i(x_i), \ldots, \iota_k(x_k), \iota_{k+1}(x_{k+1}), \overline{p}(x_i), tr_{t,i}] \rightsquigarrow\\
\quad [\iota_0(x_0), \iota_1(x_1), \ldots, \iota_k(x_k), \iota_{k+1}(x_{k+1}), apply_{t,i,p}(x)] + \Sigma_{j=1}^{k} G_t(S_i, p, S_j)^{x_j} : x_0 < x < x_{k+1}\\
\textit{Apply } G_t \textit{ to indexes inside regions, move to the right}\\
[\iota_j(v), apply_{t,i,p}(u), \overline{p}(u), \iota_{j+1}(v')] \rightsquigarrow\\
\quad [\iota_j(v), apply_{t,i,p}(u'), \overline{p}(u), \iota_{j+1}(v')] + G_t(S_i, p, R_j)^u : v < u < v', u < u'\\
\textit{Terminate visit continue with next token}\\
[apply_{t,i,p}(u)] \rightsquigarrow [tr_{t,i}]\ :\ true\\
\textit{Move to next index}\\
[tr_{t,j}] \rightsquigarrow [tr_{t,j+1}]\ :\ true\\
\textit{Terminate transfer of tokens for indexes in } \alpha_t,\ \textit{start transfer of tokens of regions}\\
[\iota_0(f), tr_{t,k}] \rightsquigarrow [trR_t(f)]\ :\ true
\end{array}
$$

**Fig. B.2**. Silent CMRS rules for simulation of transfer: $G_t(S_i, p, \pi)^x$ is the multiset that, for each $q \in P$, contains $G_t(S_i, p, \pi, q)$ occurrences of the term $q(x)$.

$$
\begin{array}{l}
\text{For } \mathrm{k} = \alpha_{\mathrm{t}},\ \mathrm{i} \in \{0, \ldots, \mathrm{k}\},\ \text{and any } \mathrm{p} \in \mathrm{P}: \\
\textit{Remove token and apply } G_t \textit{ to indexes inside regions} \\
[\iota_0(x_0), \iota_1(x_1), \ldots, \iota_i(x_i), \iota_{i+1}(x_{i+1}), \ldots, \iota_{k+1}(x_{k+1}), \overline{p}(u), trR_t(u)] \rightsquigarrow \\
\quad [\iota_0(x_0), \iota_1(x_1), \ldots, \iota_i(x_i), \iota_{i+1}(x_{i+1}), \ldots, \iota_{k+1}(x_{k+1}), trR_t(u)] \\
\quad\quad + G_t(R_i, p, R_i)^u + \Sigma_{j=1}^{k} G_t(R_i, p, S_j)^{x_j} : x_i < u < x_{i+1} \\
\textit{Move pointers to the right} \\
[trR_t(u), \iota_{k+1}(l)] \rightsquigarrow [trR_t(u'), \iota_{k+1}(l), ] : u < u' < l \\
\textit{Terminate visit, move to addition step} \\
[trR_t(u)] \rightsquigarrow [add_t] : true
\end{array}
$$

**Fig. B.3**. Silent CMRS rules for *trR*: transfer inside a region-index and from a region-index to $\alpha_t$-indexes: $G_t(R_i, p, \pi)^x$ is the multiset that, for each $q \in P$, contains $G_t(R_i, p, \pi, q)$ occurrences of the term $q(x)$.

$$
\begin{array}{l}
\textit{For } k = \alpha_t,\ i \in \{0, \ldots, k\},\ \textit{and any } p \in P: \\
\textit{Apply } H_t \textit{ to indexes in } \alpha_t \\
[\iota_0(x_0), \iota_1(x_1), \ldots, \iota_k(x_k), add_t] \rightsquigarrow \\
\quad [\iota_0(x_0), \iota_1(x_1), \ldots, \iota_k(x_k), addR_t(x_0)] + \Sigma_{j=1}^{k} H_t(S_j)^{x_j} : true \\
\textit{Apply } H_t \textit{ to an index inside a region and advance pointer} \\
[\iota_i(v), \iota_{i+1}(v'), addR_t(u), \checkmark(u)] \rightsquigarrow \\
\quad [\iota_i(v), \iota_{i+1}(v'), addR_t(u')] + H_t(R_i)^u : v < u < v', u < u' \\
\textit{Terminate simulation of transition } t \\
[\iota_0(x_0), \iota_1(x_1), \ldots, \iota_{k+1}(x_{k+1}), addR_t(u)] \overset{\lambda(t)}{\rightsquigarrow} [first(x_0), last(x_{k+1})] : true
\end{array}
$$

**Fig. B.4**. CMRS rules for $add_t$ and $addR_t$ (all silent except the last one): $H_t(\pi)^x$ is the multiset that, for each $p \in P$, contains $H_t(\pi, p)$ occurrences of the term $p(x)$ for any $\pi \in R(\alpha_t)$.

$$
\begin{array}{l}
\textit{Start copy}: \\
[first(f), last(l)] \rightsquigarrow [first_1(f), scan_t^0(f), last_1(l), copy_t^0(l')] : l < l' \\
\textit{Copy a term}: \\
\textit{For } i \in \{0, 1\} \\
[scan_t^i(x), p(x), copy_t^i(y)] \rightsquigarrow [scan_t^i(x), copy_t^i(y), p(y)] : true \\
\textit{Move to the right}: \\
\textit{For } i \in \{0, 1\} \\
[scan_t^i(x), last_1(l), copy_t^i(y)] \rightsquigarrow \\
\quad [scan_t^i(x'), last_1(l), copy_t^i(y')] : x < x' < l, y < y' \\
\textit{Move to the right and reserve a fresh value in the copy tape}: \\
[scan_t^0(x), last_1(l), copy_t^0(y)] \rightsquigarrow \\
\quad [scan_t^1(x'), last_1(l), fresh_t(y'), copy_t^1(y'')] : x < x' < l, y < y' < y'' \\
\textit{Terminate copy (copy tape becomes current configuration)}: \\
[first_1(f), scan_t^1(x), last_1(l), copy_t^1(y)] \rightsquigarrow [first_1(l), last_1(y)] : true \\
\textit{Subtraction}: \\
\textit{For } j \in \{1, \ldots, k\} \\
[first_1(f), fresh_t(x), last_1(l)] + \Sigma_{i=1}^{k} F_t(S_i)^{x_i} \rightsquigarrow \\
\quad [\iota_0(f), \iota_1(x_1), .., \iota_j(x), .., \iota_k(x_k), \iota_{k+1}(l), new_t] : \\
\quad\quad f < x_1 < .. < x_j < x < x_{j+1} < x_k < l
\end{array}
$$

**Fig. B.5**. Silent CMRS rules for the simulation of subtraction with selection of one fresh value.

For $k = \alpha_t$, $i \in \{0, \dots, k\}$, and any $p \in P$ :

*Remove token, apply $G_t$ to indexes inside region and to $\alpha_t$ − indexes*

$$[\iota_0(x_0), \iota_1(x_1), \dots, \iota_i(x_i), \iota_{i+1}(x_{i+1}), \dots, \iota_{k+1}(x_{k+1}), \overline{p}(u), trR_t(u)] \rightsquigarrow$$
$$[\iota_0(x_0), \iota_1(x_1), \dots, \iota_i(x_i), \iota_{i+1}(x_{i+1}), \dots, \iota_{k+1}(x_{k+1}), applyR_{i,p,t}(u)]$$
$$+ G_t(R_i, p, R_i)^u + \Sigma_{j=1}^{k} G_t(R_i, p, S_j)^{x_j} : x_i < u < x_{i+1}$$

*Apply $G_t$ to other region − indexes, move to the right* For $i \neq j$, $\pi \in \{\overline{p}, \checkmark\}$ :

$$[\iota_j(v), applyR_{t,i,p}(u), \pi(u), \iota_{j+1}(v')] \rightsquigarrow$$
$$[\iota_j(v), applyR_{t,i,p}(u'), \pi(u), \iota_{j+1}(v')] + G_t(R_i, p, R_j)^u : v < u < v', u < u'$$

*Apply $M_t$ to other index in the same region, move to the right*

$$[\iota_i(v), applyR_{t,i,p}(u), \pi(z), \iota_{i+1}(v')] \rightsquigarrow$$
$$[\iota_i(v), applyR_{t,i,p}(u'), \pi(z), \iota_{i+1}(v')] + M_t(R_i, p, R_j)^z : z \neq u, v < u < v', u < u'$$

*Terminate visit continue with next token*

$$[apply_{t,i,p}(u)] \rightsquigarrow [trR_t] \ : \ true$$

*Move $trR_t$ pointer to the right*

$$[trR_t(u), \iota_{k+1}(l)] \rightsquigarrow [trR_t(u'), \iota_{k+1}(l),] \ : \ u < u' < l$$

*Terminate visit, move to addition step*

$$[trR_t(u)] \rightsquigarrow [add_t] \ : \ true$$

**Fig. B.6**. Silent CMRS rules for the simulation of transfer between distinct regions, or distinct indexes inside the same region.

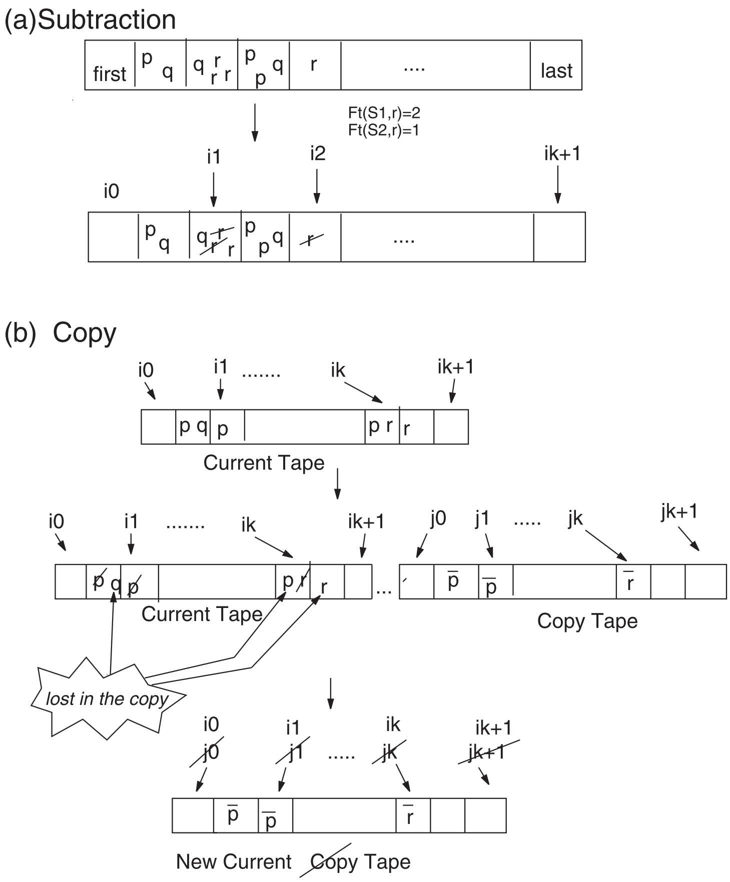


**Fig. B.7**. Simulation of: (a) Subtraction and selection of indexes; (b) Creation of a new configuration.

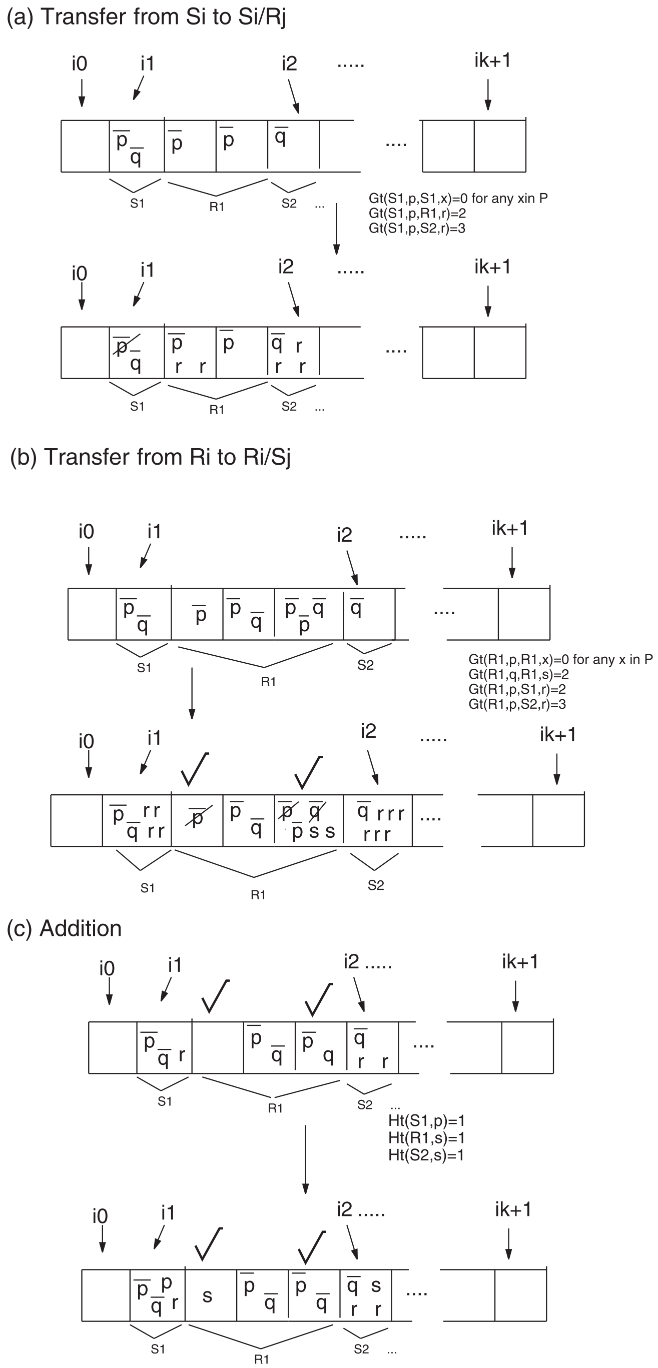


**Fig. B.8**. Simulation of: (a) transfer from $S_i$ to $R_j/S_j$; (b) transfer from $R_i$ to $R_i/S_j$; (c) addition.

## References


[1] P.A. Abdulla, K. Čerāns, B. Jonsson, T. Yih-Kuen, General decidability theorems for infinite-state systems, LICS (1996) 313–321.
[2] P.A. Abdulla, G. Delzanno, On the coverability problem for constrained multiset rewriting systems, AVIS 2006 (ETAPS 2006 workshop), Available at the URL http://www.disi.unige.it/person/DelzannoG/Papers/avis06.pdf.
[3] P.A. Abdulla, G. Delzanno, L. Van Begin, Comparing the expressive power of well-structured transition systems, CSL (2007) 99–114.
[4] P.A. Abdulla, B. Jonsson, Verifying programs with unreliable channels, Inf. Comput. 127 (2) (1996) 91–101.
[5] P.A. Abdulla, B. Jonsson, Undecidable verification problems for programs with unreliable channels, Inf. Comput. 130 (1) (1996) 71–90.
[6] N. Bertrand, Ph. Schnoebelen, A short visit to the STS hierarchy, ENTCS 154 (3) (2006) 59–69.
[7] G. Cécé, A. Finkel, S.P. Iyer, Unreliable channels are easier to verify than perfect channels, Inf. Comput. 124 (1) (1996) 20–31.
[8] K. Čerāns, Deciding properties of integral relational automata, ICALP (1994) 35–46.

[9] G. Ciardo, Petri nets with marking-dependent arc multiplicity: properties and analysis, ICATPN (1994) 179–198.
[10] L.E. Dickson, Finiteness of the odd perfect and primitive abundant numbers with r distinct prime factors, Am. J. Math. 35 (1913) 413–422.
[11] J. Esparza, A. Finkel, R. Mayr, On the verification of broadcast protocols, LICS (1999) 352–359.
[12] C. Dufourd, A. Finkel, Ph. Schnoebelen, Reset nets between decidability and undecidability, ICALP (1998) 103–115.
[13] A. Finkel, G. Geeraerts, J.-F. Raskin, L. Van Begin, On the $\omega$-language expressive power of extended Petri nets, TCS 356 (3) (2006) 374–386.
[14] A. Finkel, Ph. Schnoebelen, Well-structured transition systems everywhere, TCS 256 (1–2) (2001) 63–92.
[15] G. Geeraerts, J.-F. Raskin, L. Van Begin, Well-structured languages, Acta Inform. 44 (3–4) (2007) 249–288.
[16] M. Hack, Petri net languages, Technical Report TR-159, MIT, 1976.
[17] T.A. Henzinger, R. Majumdar, J.-F. Raskin, A classification of symbolic transition systems, ACM Trans. Comput. Log. 6 (1) (2005) 1–32.
[18] F. Rosa Velardo, D. de Frutos-Escrig, Name creation vs. replication in Petri net systems, ICATPN (2007) 402–422.
[19] A. Finkel, P. McKenzie, C. Picaronny, A well-structured framework for analysing petri net extensions, Inf. Comput. 195 (1–2) (2004) 1–29.
[20] R. Lazic, T.C. Newcomb, J. Ouaknine, A.W. Roscoe, J. Worrell, Nets with tokens which carry data, ICATPN (2007) 301–320.
[21] R. Mayr, Undecidable problems in unreliable computations, LATIN (2000) 377–386.
[22] C.A. Petri, Kommunikation mit Automaten, Ph.D. Thesis, University of Bonn, 1962.