Centrum voor Wiskunde en Informatica
REPORT RAPPORT

Linearization in Parallel pCRL

J.F. Groote, A. Ponse, Y.S. Usenko

Software Engineering (SEN)

SEN-R0019 July 31, 2000

Report SEN-R0019
ISSN 1386-369X

CWI
P.O. Box 94079
1090 GB Amsterdam
The Netherlands

Linearization in Parallel pCRL

Jan Friso Groote[1,2] Alban Ponse[1,3] Yaroslav S. Usenko[1]
JanFriso.Groote@cwi.nl Alban.Ponse@cwi.nl Yaroslav.Usenko@cwi.nl

[1] *CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands*

[2] *Computing Science Department, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands*

[3] *Programming Research Group, University of Amsterdam, P.O. Box 41882, 1009 DB Amsterdam, The Netherlands*

ABSTRACT
We describe a linearization algorithm for parallel pCRL processes similar to the one implemented in the linearizer of the μCRL Toolset. This algorithm finds its roots in formal language theory: the 'grammar' defining a process is transformed into a variant of Greibach Normal Form. Next, any such form is further reduced to *linear form*, i.e., to an equation that resembles a right-linear, data-parametric grammar. We aim at proving the correctness of this linearization algorithm. To this end we define an equivalence relation on recursive specifications in μCRL that is model independent and does not involve an explicit notion of solution.

2000 Mathematics Subject Classification: 68Q10; 68Q42; 68Q65; 68Q85
1998 ACM Computing Classification System: D.2.1; D.2.4; D.3.1; D.3.3; F.3.2; I.1.1
Keywords and Phrases: μCRL, Process Algebra, Linearization of Recursive Specifications

1. Introduction

In this paper we address the issue of linearization of recursive specifications in the specification language μCRL (micro Common Representation Language, [17, 13]). The language μCRL has been developed under the assumption that an extensive and mathematically precise study of the basic constructs of specification languages is fundamental to an analytical approach of much richer (and more complicated) specification languages such as SDL [29], LOTOS [21], PSF [23, 24] and CRL [28]. Moreover, it is assumed that μCRL and its proof theory provide a solid basis for the design and construction of tools for analysis and manipulation of distributed systems.

The language μCRL offers a uniform framework for the specification of data and processes. Data are specified by equational specifications: one can declare sorts and functions working upon these sorts, and describe the meaning of these functions by equational axioms. Processes are described in process algebraic style, where the particular process syntax stems from ACP [3, 2, 11], extended with data-parametric ingredients: there are constructs for conditional composition, and for data-parametric choice and communication. As is common in process algebra, infinite processes are specified by means of (finite systems of) recursive equations. In μCRL such equations can also be data-parametric. As an example, for action a and adopting standard semantics for μCRL, each solution for the equation $\mathsf{X} = \mathsf{a} \cdot \mathsf{X}$ specifies (or "identifies") the process that can only repeatedly execute a, and so does each solution for $\mathsf{Y}(17)$ where $\mathsf{Y}(n)$ is defined by the data-parametric equation $\mathsf{Y}(n) = \mathsf{a} \cdot \mathsf{Y}(n+1)$ with $n \in Nat$. An interesting subclass of systems of recursive equations consists of those that contain only one *linear* equation. Such a system is called an LPE (Linear Process Equation). Here, linearity refers both to the form of recursion allowed, and to a restriction on the process syntax allowed. The above examples $\mathsf{X} = \mathsf{a} \cdot \mathsf{X}$ and $\mathsf{Y}(n) = \mathsf{a} \cdot \mathsf{Y}(n+1)$ are both LPEs. The restriction to LPE format still yields an expressive setting (for example, it is not hard to show that each computable process over a finite set of actions can be simply defined using an LPE containing only computable functions over the natural numbers, cf. [27]). Moreover, in the design and construction of tools for μCRL, LPEs establish a basic and convenient representation format. This applies, for example, to tools for generation of labeled transition systems, or tools for optimization, deadlock checking, or simulation. The LPE format stems from [6], in which the notion of a *process operator* is distinguished, and a proof technique for dealing with convergent LPEs is defined. Furthermore, there is a strong resemblance between LPEs and specifications in UNITY [10, 8]. The restriction to linear systems has a long tradition in process algebra. For instance, restricting to so-called linear *specifications*, i.e., linear systems that in some distinguished model have a unique solution per variable, various completeness results were proved in a simple fashion (cf. [25, 4]). However, without data-parametric constructs for process specification, the expressiveness is limited: only regular processes can be defined.

The language μCRL is considered to be a specification language because it contains ingredients that facilitate in a straightforward, natural way the modeling of distributed, communicating processes. In particular, it contains constructs for *parallelism*, *encapsulation* and *abstraction*. On the other hand, as sketched above, LPEs constitute a basic fragment of μCRL in terms of expressiveness and tool support. This explains our interest in transforming any system of μCRL equations into an equivalent LPE, i.e., our interest to *linearize* μCRL process definitions. In this paper we do not consider full μCRL as the source language for linearization, and allow only a restricted use of the above-mentioned constructs. In [6], pCRL (pico CRL) was defined as a fragment of μCRL. Essentially, pCRL restricts μCRL to the basic operations of process algebra, with data parametric choice, sequential composition and conditionals. Typically, in an LPE only pCRL syntax occurs. Now, as the source language for linearization we take *parallel* pCRL, an extension of pCRL in which a restricted use of more involved operations, such as $\|$ (parallel composition), is allowed. For example, in parallel pCRL the $\|$ may not occur in the scope of a recursion. Almost all real life, distributed processes have a straightforward definition in parallel pCRL. In [7], a linearization procedure was sketched for a fragment of μCRL which is similar to parallel pCRL.

We define the linearization algorithm on an abstract level, but in a very detailed manner. We do not concern ourselves with the question if and in what way systems of recursive equations over parallel pCRL define processes as their unique solutions (per variable). Instead, we argue that the transformation is correct in a more general sense: we show that linearization "preserves all solutions". This means that if a particular parallel pCRL system of recursive equations defines a series of solutions for its variables in some model, then the LPE resulting from linearization has (at least) the same solutions for the associated process terms. Consequently, if the resulting LPE is such that one can infer that these solutions are *unique* in some particular (process) model, then both systems define the same processes in that model. In our algorithm, most transformation steps satisfy a stronger property: the set of solutions is the same before and after the transformation. Both the detailed description of the linearization algorithm itself, and the preservation of solutions, which technically speaking is a notion of *implication* between process terms over different μCRL systems, can be considered the contribution of this paper. To the best of our knowledge, a first description of a transformation of (non-parallel) pCRL into an LPE like format was given in [5]. Transformation procedures from BPA to Greibach Normal Forms were outlined in [1] and presented in [20].

Structure of the paper. In Section 2 we discuss parallel pCRL. Furthermore, we define implication and equivalence between pCRL process terms defined over different pCRL specifications. Sections 3, 4 and 5 fully describe the linearization procedure. In Section 3 we describe in detail the first part of this transformation, which yields process definitions in so-called extended Greibach normal form. In Section 4 we define the LPE format, and describe the transformation from extended Greibach normal form into this format. Then, in Section 5 we consider the effect of the typical parallel pCRL operations on LPEs. The paper is ended with some conclusions in Section 6. In particular, we provide some comments on our transformation, and relate our approach to other work.

Acknowledgments. Thanks go to Jan Bergstra, Wan Fokkink, Bas Luttik, Faron Moller, Vincent van Oostrom, Jaco van de Pol, and Mark van der Zwaag for helpful discussions and comments.

2. Description of μCRL and Parallel pCRL

In this section we first recall some general information about μCRL. Then we consider (recursive) process definitions in detail, and define various notions of equivalence, among which equivalence between process terms defined over different μCRL specifications. Next, we shortly discuss guardedness and dependency in process definitions. Finally, we introduce pCRL and *parallel* pCRL as fragments of μCRL.

2.1 Theory of μCRL

First we define the signature and axioms for booleans which are quite standard and can be found for instance in [9] (page 116). We use equational logic to prove boolean identities. Booleans are obligatory in any μCRL specification.

Definition 2.1. The signature of *Bool* consists of constants $\mathbf{t}, \mathbf{f}$, unary operation *not* and binary operations *and*, *or*, *eq*.

Note (Booleans). We use infix notation $\neg, \wedge, \vee, \leftrightarrow$ for *not*, *and*, *or*, *eq* respectively.

Definition 2.2. The axioms of *Bool* are the ones presented in Table 1.

$$
\begin{array}{rcl@{\qquad\qquad}rcl}
x \wedge y &=& y \wedge x & x \vee y &=& y \vee x \\
(x \wedge y) \wedge z &=& x \wedge (y \wedge z) & (x \vee y) \vee z &=& x \vee (y \vee z) \\
x \wedge x &=& x & x \vee x &=& x \\
x \wedge (x \vee y) &=& x & x \vee (x \wedge y) &=& x \\
(x \wedge y) \vee (x \wedge z) &=& x \wedge (y \vee z) & (x \vee y) \wedge (x \vee z) &=& x \vee (y \wedge z) \\
x \wedge \mathbf{f} &=& \mathbf{f} & x \vee \mathbf{t} &=& \mathbf{t} \\
x \wedge \neg x &=& \mathbf{f} & x \vee \neg x &=& \mathbf{t} \\
x \leftrightarrow y &=& (x \wedge y) \vee (\neg x \wedge \neg y) & & &
\end{array}
$$

Table 1: Axioms of *Bool*.

Next we define the generalized equational theory of μCRL by defining its signature and the axioms. The axioms are taken from, or inspired by [15, 16].

Note (Vector Notation). Tuples occur a lot in the language, so we use a vector notation for them. Expression $\overrightarrow{d}$ is an abbreviation for $d^1, \ldots, d^n$, where d^k are data variables. Similarly, if type information is given, $\overrightarrow{d{:}D}$ is an abbreviation for $d^1{:}D^1, \ldots, d^n{:}D^n$ for some natural number n. In case $n = 0$ the whole vector vanishes as well as brackets surrounding it. For instance $\mathsf{a}(\overrightarrow{d})$ is an abbreviation for a in this case (here a is an *action*, this notion is introduced below). For all vectors $\overrightarrow{d}$ and $\overrightarrow{e}$ we have $\overrightarrow{d}, \overrightarrow{e} = \overrightarrow{d, e}$. Thus $\overrightarrow{d, e}$ is an abbreviation for $d^1, \ldots, d^n, e^1, \ldots, e^{n'}$. We also write $\overrightarrow{d{:}D} \,\&\, e{:}E$ for $d^1{:}D^1, \ldots, d^n{:}D^n, e{:}E$.

For any vector of variables $\overrightarrow{d}$, $\overrightarrow{f}(\overrightarrow{d})$ is an abbreviation for $f^1(\overrightarrow{d}), \ldots, f^m(\overrightarrow{d})$ for some $m \in \mathit{Nat}$, where each $f^k(\overrightarrow{d})$ is a data term that may contain elements of $\overrightarrow{d}$ as free variables. As with vectors of variables, in case $m = 0$ the vector of data terms vanishes. We often use $\overrightarrow{t}$ to express a data term vector without explicitly denoting its variables.

Definition 2.3. The signature of μCRL consists of data sorts (or 'data types') including *Bool* as defined above, and a distinct sort *Proc* of processes. Each data sort D is assumed to be equipped with a binary function $eq : D \times D \rightarrow Bool$. (This requirement can be weakened by demanding such functions only for data sorts that are parameters of communicating actions). The operational signature of μCRL is parameterized by the set of action labels *ActLab* and a partial commutative and associative function $\gamma : ActLab \times ActLab \rightarrow ActLab$ such that $\gamma(\mathsf{a}_1, \mathsf{a}_2) \in ActLab$ implies that $\mathsf{a}_1, \mathsf{a}_2$ and $\gamma(\mathsf{a}_1, \mathsf{a}_2)$ have parameters of the same sorts. The process operations are the ones listed below:

- actions $\mathsf{a}(\overrightarrow{t})$ parameterized by data terms $\overrightarrow{t}$, where $\mathsf{a} \in ActLab$ is an action label. More precisely, a is an operation $\mathsf{a} : \overrightarrow{D} \rightarrow Proc$.
- constants δ and τ of sort *Proc*.
- binary operations $+, \cdot, \|, ⌊⌊, |$ defined on *Proc*, where $|$ is defined using γ.
- unary *Proc* operations $\partial_H, \tau_I, \rho_R$ for each set of action labels $H, I \subseteq ActLab$ and action label renaming function $R : ActLab \rightarrow ActLab$ such that a and $R(\mathsf{a})$ have parameters of the same sorts.
- ternary operation $\_ \lhd \_ \rhd \_ : Proc \times Bool \times Proc \rightarrow Proc$.
- binders $\sum_{d:D}$ defined on *Proc*, for each data variable d of sort D.

The partial function γ is called a *communication function.* If $\gamma(\mathsf{a}, \mathsf{b}) = \mathsf{c}$ this indicates that actions with labels a and b can synchronize, becoming action c, provided that the data parameters of these actions are equal. The constant δ represents a deadlocked process and the constant τ represents some internal or hidden activity. The *choice* operator $+$ and the *sequential composition* operator $\cdot$ are well known. The *merge* operator $\|$ represents *parallel composition.* The ⌊⌊ (*left merge*) and $|$ (*communication merge*) are auxiliary operations used to equationally define $\|$. The *encapsulation* operator $\partial_H(q)$ blocks actions in q with action labels in the set H, which is especially used to enforce actions to communicate. The *hiding* operator $\tau_I(q)$ with a set of action labels $I = \{\mathsf{a}, \mathsf{b} \ldots\}$ hides actions with these labels in q by renaming them to τ. The *renaming* operator $\rho_R(q)$ where R is a function from action labels to action labels renames each action with label a in q to an action with label $R(\mathsf{a})$. The operator $p_1 \lhd c \rhd p_2$ is the *if* c *then* p_1 *else* p_2 operator, where c is an expression of type *Bool*. The *sum operator* $\sum_{d:D} p$ expresses a (potentially infinite) summation $p[d := d_0] + p[d := d_1] + \ldots$ if data domain $D = \{d_0, d_1, \ldots\}$.

Definition 2.4. Axioms of μCRL are the ones presented in Tables 2,3,4,5,6 and 7. We assume that

- $+$ binds weaker, and $\cdot$ binds stronger than other operations.
- x, y, z are variables of sort *Proc*.
- c, c_1, c_2 are variables of sort *Bool*.
- $d, d^1, d^n, d', \ldots$ are data variables (but not in $\sum_{d:D}$, where d is part of the operation).
- b stands for either $\mathsf{a}(\overrightarrow{d})$, or τ, or δ.
- $\overrightarrow{d} = \overrightarrow{d'}$ is an abbreviation for $eq(d^1, d^{1\prime}) \wedge \ldots \wedge eq(d^n, d^{n\prime})$, where $\overrightarrow{d} = d^1 \ldots d^n$ and $\overrightarrow{d'} = d^{1\prime} \ldots d^{n\prime}$.

- the axioms where p and q occur are schemas ranging over all terms p and q of sort *Proc*, *including* those in which d occurs freely.
- the axiom (SUM2) is a scheme ranging over all terms r of sort *Proc* in which d *does not* occur freely.

The axioms in Table 7 (actually only (SC3)) are used only for the parallel composition elimination (Section 5). Note that due to (SC3), the axioms (CM6), (CM9), (CT2), (CD2), (Cond9′) and (SUM7′) become derivable. The axioms (B1) and (B2) are not used in the transformations described in this paper, so they are also valid in models where these two axioms do not hold.

We use many sorted equational logic for processes and booleans, while other data types can have slightly different proof rules, which may include induction principles, quantifier introduction principles, etc. The proof theory of μCRL consists of proof rules for the data sorts, the rules of equational logic for the booleans, and the rules of generalized equational logic [15] for the processes. Note that the rules of generalized equational logic do not allow to substitute terms containing free variables if they become bound. For example, in axiom (SUM1) we cannot substitute $\mathsf{a}(d)$ for x.

Definition 2.5. Two process terms p_1 and p_2 are *(unconditionally) equivalent* (notation $p_1 = p_2$) if $p_1 = p_2$ is derivable from the axioms of μCRL and boolean identities by using many sorted generalized equational logic ($\{\mu\text{CRL}, BOOL\} \vdash p_1 = p_2$). Here $BOOL$ is used to refer to the specification of the booleans, and the use of equational logic for deriving boolean identities.

Two process terms p_1 and p_2 are *conditionally equivalent* if $\{\mu\text{CRL}, BOOL, DATA\} \vdash p_1 = p_2$. Here $DATA$ is used to refer to the specification of all data sorts involved, and all proof rules that may be applied.

$$x + y = y + x \tag{A1}$$
$$x + (y + z) = (x + y) + z \tag{A2}$$
$$x + x = x \tag{A3}$$
$$(x + y) \cdot z = x \cdot z + y \cdot z \tag{A4}$$
$$(x \cdot y) \cdot z = x \cdot (y \cdot z) \tag{A5}$$
$$x + \delta = x \tag{A6}$$
$$\delta \cdot x = \delta \tag{A7}$$
$$x \cdot \tau = x \tag{B1}$$
$$z \cdot (\tau \cdot (x + y) + x) = z \cdot (x + y) \tag{B2}$$

Table 2: Basic axioms of μCRL.

2.2 Systems of Recursion Equations

We assume a fixed and infinite set $\mathsf{Procnames} = \{\mathsf{X}, \mathsf{Y}, \mathsf{Z}, \ldots\}$ of *process names* with type information associated to them. We extend the sort *Proc* of processes by allowing the process names in $P \subseteq \mathsf{Procnames}$ as variables of type $\overrightarrow{D} \to Proc$. These terms are further called *(μCRL) process terms* and the set of all of them is denoted by $Terms(P)$. The free data variables in a process term are those not bound by $\sum_{d:D}$ occurrences. We write $DVar$ for the set of all free and bound data variables that can occur in a term.

$$x \parallel y = (x \mathbin{\lfloor\!\lfloor} y + y \mathbin{\lfloor\!\lfloor} x) + x \mid y \quad \text{(CM1)}$$
$$b \mathbin{\lfloor\!\lfloor} x = b \cdot x \quad \text{(CM2)}$$
$$(b \cdot x) \mathbin{\lfloor\!\lfloor} y = b \cdot (x \parallel y) \quad \text{(CM3)}$$
$$(x + y) \mathbin{\lfloor\!\lfloor} z = x \mathbin{\lfloor\!\lfloor} z + y \mathbin{\lfloor\!\lfloor} z \quad \text{(CM4)}$$
$$(b \cdot x) \mid b' = (b \mid b') \cdot x \quad \text{(CM5)}$$
$$b \mid (b' \cdot x) = (b \mid b') \cdot x \quad \text{(CM6)}$$
$$(b \cdot x) \mid (b' \cdot y) = (b \mid b') \cdot (x \parallel y) \quad \text{(CM7)}$$
$$(x + y) \mid z = x \mid z + y \mid z \quad \text{(CM8)}$$
$$x \mid (y + z) = x \mid y + x \mid z \quad \text{(CM9)}$$
$$\mathsf{a}(\overrightarrow{d}) \mid \mathsf{a}'(\overrightarrow{d'}) = \gamma(\mathsf{a}, \mathsf{a}')(\overrightarrow{d}) \lhd \overrightarrow{d} = \overrightarrow{d'} \rhd \delta \quad \text{if } \gamma(\mathsf{a}, \mathsf{a}') \text{ is defined} \quad \text{(CF1)}$$
$$\mathsf{a}(\overrightarrow{d}) \mid \mathsf{a}'(\overrightarrow{d'}) = \delta \quad \text{otherwise} \quad \text{(CF2)}$$
$$\tau \mid b = \delta \quad \text{(CT1)}$$
$$b \mid \tau = \delta \quad \text{(CT2)}$$
$$\delta \mid b = \delta \quad \text{(CD1)}$$
$$b \mid \delta = \delta \quad \text{(CD2)}$$

Table 3: Axioms for parallel composition in μCRL.

Definition 2.6. A *process equation* is an equation of the form $\mathsf{X}(\overrightarrow{d_{\mathsf{X}}{:}D_{\mathsf{X}}}) = q_{\mathsf{X}}$, where X is a process name with a list of data parameters $\overrightarrow{d_{\mathsf{X}}{:}D_{\mathsf{X}}}$, and q_{X} is a process term, in which only the data variables from $\overrightarrow{d_{\mathsf{X}}}$ may occur freely. We write $\mathit{rhs}(\mathsf{X})$ for q_{X}, $\mathit{pars}(\mathsf{X})$ for $\overrightarrow{d_{\mathsf{X}}}$, and $\mathit{type}(\mathsf{X})$ for $\overrightarrow{D_{\mathsf{X}}}$.

Definition 2.7. Let $P \subseteq \mathsf{Procnames}$ be a finite set of process names such that each process name is uniquely typed. A (finite) non-empty set G of process equations over $\mathit{Terms}(P)$ is called a *(finite) system of process equations* if each process name in P occurs exactly once at the left. The set of process names (with types) that appear within G is denoted as $|G|$ (so, $|G| = P$). We use $\mathit{rhs}(\mathsf{X}, G)$, $\mathit{pars}(\mathsf{X}, G)$ and $\mathit{type}(\mathsf{X}, G)$ to refer to the corresponding parts of the equation for X in G.

Although the original definition of a μCRL specification allows to have the same process names with different types, we do not treat this possibility here as it would make the explanation only more long-winded.

Definition 2.8. Let G be a finite system of process equations, X be a process name in it, and $\overrightarrow{t}$ be a data term vector of type $\mathit{type}(\mathsf{X}, G)$. Then the pair $(\mathsf{X}(\overrightarrow{t}), G)$ is called a *process definition*. We use the abbreviation (X, G) for $(\mathsf{X}(\mathit{pars}(\mathsf{X}, G)), G)$.

Example 2.9. Both $G_1 = \{\mathsf{X} = \mathsf{a}{\cdot}\mathsf{Y}, \mathsf{Y} = \mathsf{b}{\cdot}\mathsf{X}, \mathsf{Z} = \mathsf{X} \| \mathsf{Y}\}$ and $G_2 = \{\mathsf{T}(n{:}\mathit{Nat}) = \mathsf{a}(\mathit{even}(n)){\cdot}\mathsf{T}(S(n))\}$ with $\mathit{even} : \mathit{Nat} \to \mathit{Bool}$ as expected and $S : \mathit{Nat} \to \mathit{Nat}$ the successor function, are examples of systems of process equations. All of $(\mathsf{X}, G_1), (\mathsf{T}, G_2), (\mathsf{T}(m), G_2)$ are process definitions.

Definition 2.10. Process term q *directly depends* on process name X if this name occurs in q. Process name X *directly depends* on process name Y in a system of process equations G if $\mathit{rhs}(\mathsf{X}, G)$ directly depends on Y. Process term q *depends* on X in G if it either directly depends on it, or there is a sequence of process names $\mathsf{Y}_1, \ldots, \mathsf{Y}_n = \mathsf{X}$ such that q directly depends on Y_1 and for each $i < n$, Y_i directly depends on Y_{i+1}. Process name X *depends* on Y in G if $\mathit{rhs}(\mathsf{X}, G)$ depends on it.

We note that the combination of the given data specification with a system G of process equations determines a μCRL *specification* in the sense as defined in [17]. Such a specification depends on a

$$x \lhd \mathbf{t} \rhd y = x \tag{Cond1}$$
$$x \lhd \mathbf{f} \rhd y = y \tag{Cond2}$$
$$x \lhd c \rhd y = x \lhd c \rhd \delta + y \lhd \neg c \rhd \delta \tag{Cond3}$$
$$(x \lhd c_1 \rhd \delta) \lhd c_2 \rhd \delta = (x \lhd c_1 \wedge c_2 \rhd \delta) \tag{Cond4}$$
$$(x \lhd c_1 \rhd \delta) + (x \lhd c_2 \rhd \delta) = x \lhd c_1 \vee c_2 \rhd \delta \tag{Cond5}$$
$$(x \lhd c \rhd \delta) \cdot y = (x \cdot y) \lhd c \rhd \delta \tag{Cond6}$$
$$(x + y) \lhd c \rhd \delta = x \lhd c \rhd \delta + y \lhd c \rhd \delta \tag{Cond7}$$
$$(x \lhd c \rhd \delta) \mathbin{⌊\!⌊} y = (x \mathbin{⌊\!⌊} y) \lhd c \rhd \delta \tag{Cond8}$$
$$(x \lhd c \rhd \delta) \mid y = (x \mid y) \lhd c \rhd \delta \tag{Cond9}$$
$$x \mid (y \lhd c \rhd \delta) = (x \mid y) \lhd c \rhd \delta \tag{Cond9$'$}$$
$$(x \lhd c \rhd \delta) \cdot (y \lhd c \rhd \delta) = (x \cdot y) \lhd c \rhd \delta \tag{Sca}$$

Table 4: Axioms for conditions in μCRL.

finite subset **act** of *ActLab* and on **comm**, an enumeration of γ restricted to the labels in **act**. So a finite system G implicitly describes a finitary based language.

For a consistent (meaningful) specification, i.e., a *Statically Semantically Correct* specification, it is necessary that all objects are specified only once, that all typing is respected and that the communications in **comm** are specified in a functional way. Furthermore, the *eq* functions for the data sorts should have the following properties:

$$\{DATA, eq(d,e) = \mathbf{t}\} \vdash d = e \quad \text{and} \quad \{DATA, x = y\} \vdash eq(d,e) = \mathbf{t}$$

All data sorts that are introduced during the linearization must have *eq* functions satisfying these properties.

2.3 Equivalence of Process Definitions

We introduce *equivalence* over systems of process equations in a stepwise manner. Let G_1 and G_2 be systems of process equations, and assume that the common data sorts of G_1 and G_2 are equally defined. Then $DATA(G_1, G_2)$ represents all data specifications occurring in G_1 and G_2 and all proof rules adopted for these data. We first define (conditional) implication between process terms, and then the equivalence.

In the following definition, derivabilities of the form $\{\mu\text{CRL}, BOOL, DATA\} \cup G_1 \vdash \phi$ are required. In this case, the axioms from μCRL, *BOOL* and *DATA* may be used to derive ϕ, as well as the process equations in G_1. However, we restrict derivability by requiring that the (data-parametric) process names from G_1 are considered as (data-parametric) *constants*. For example, if $G_1 = \{\mathsf{X} = \mathsf{a} \cdot \mathsf{X}\}$, we may use $\mathsf{X} = \mathsf{a} \cdot \mathsf{X}$ as an axiom in $\{\mu\text{CRL}, BOOL, DATA\} \cup \{\mathsf{X} = \mathsf{a} \cdot \mathsf{X}\} \vdash \phi$, but X may *not* be used as a variable that can be instantiated (e.g., $\{\mu\text{CRL}, BOOL, DATA\} \cup \{\mathsf{X} = \mathsf{a} \cdot \mathsf{X}\} \not\vdash \mathsf{a} = \mathsf{a} \cdot \mathsf{a}$).

Definition 2.11. Let G_1, G_2 be systems of process equations with $|G_1| = \{\mathsf{X}_1 \dots \mathsf{X}_n\}$ and $|G_2| = \{\mathsf{Y}_1 \dots \mathsf{Y}_m\}$. Let furthermore *DATA* be such that it contains $DATA(G_1, G_2)$, i.e., *DATA* contains all data sorts and associated proof rules of $DATA(G_1, G_2)$.

We say that $(\mathsf{X}_1(\overrightarrow{t_1}), G_1)$ *conditionally implies* $(\mathsf{Y}_1(\overrightarrow{t_2}), G_2)$ (notation $(\mathsf{X}_1(\overrightarrow{t_1}), G_1) \Rightarrow_c (\mathsf{Y}_1(\overrightarrow{t_2}), G_2)$) for some (possibly open) data term vectors $\overrightarrow{t_1}, \overrightarrow{t_2}$ over *DATA* if for $j = 1, \dots, m$ there is a set of

$$\sum_{d:D} x = x \quad \text{(SUM1)}$$

$$\sum_{e:D} r = \sum_{d:D}(r[e := d]) \quad \text{(SUM2)}$$

$$\sum_{d:D} p = \sum_{d:D} p + p \quad \text{(SUM3)}$$

$$\sum_{d:D}(p + q) = \sum_{d:D} p + \sum_{d:D} q \quad \text{(SUM4)}$$

$$\sum_{d:D}(p \cdot x) = (\sum_{d:D} p) \cdot x \quad \text{(SUM5)}$$

$$\sum_{d:D}(p \mathbin{\lfloor\!\lfloor} x) = (\sum_{d:D} p) \mathbin{\lfloor\!\lfloor} x \quad \text{(SUM6)}$$

$$\sum_{d:D}(p \mid x) = (\sum_{d:D} p) \mid x \quad \text{(SUM7)}$$

$$\sum_{d:D}(x \mid p) = x \mid (\sum_{d:D} p) \quad \text{(SUM7}'\text{)}$$

$$\sum_{d:D}(\partial_H(p)) = \partial_H(\sum_{d:D} p) \quad \text{(SUM8)}$$

$$\sum_{d:D}(\tau_I(p)) = \tau_I(\sum_{d:D} p) \quad \text{(SUM9)}$$

$$\sum_{d:D}(\rho_R(p)) = \rho_R(\sum_{d:D} p) \quad \text{(SUM10)}$$

$$\sum_{d:D}(p \lhd c \rhd \delta) = (\sum_{d:D} p) \lhd c \rhd \delta \quad \text{(SUM12)}$$

Table 5: Axioms for sums in μCRL.

mappings $g_{\mathsf{Y}_j} : \mathit{type}(\mathsf{Y}_j) \to \mathit{Terms}(\{\mathsf{X}_1 \dots \mathsf{X}_n\})$ such that

$$\{\mu\text{CRL}, \mathit{BOOL}, \mathit{DATA}\} \cup G_1 \vdash \mathsf{X}_1(\overrightarrow{t_1}) = g_{\mathsf{Y}_1}(\overrightarrow{t_2}) \text{ and}$$

$$\forall j \in 1..m \left(\{\mu\text{CRL}, \mathit{BOOL}, \mathit{DATA}\} \cup G_1 \vdash g_{\mathsf{Y}_j}(\overrightarrow{d_j'}) = \mathit{rhs}(\mathsf{Y}_j)\, [\forall k\ \mathsf{Y}_k(t') := g_{\mathsf{Y}_k}(t')]\right)$$

If DATA identities are not used in these derivations we say that $(\mathsf{X}_1(\overrightarrow{t_1}), G_1)$ *(unconditionally) implies* $(\mathsf{Y}_1(\overrightarrow{t_2}), G_2)$ (notation $(\mathsf{X}_1(\overrightarrow{t_1}), G_1) \Rightarrow (\mathsf{Y}_1(\overrightarrow{t_2}), G_2)$). In case $(\mathsf{X}(\mathit{pars}(\mathsf{X}, G_1)), G_1)$ (conditionally) implies $(\mathsf{Y}(\mathit{pars}(\mathsf{Y}, G_2)), G_2)$ we say that (X, G_1) (conditionally) implies (Y, G_2) (notation $(\mathsf{X}, G_1) \Rightarrow (\mathsf{Y}, G_2)$ ($(\mathsf{X}, G_1) \Rightarrow_c (\mathsf{Y}, G_2)$)).

We state without proof:

Lemma 2.12. *Let G_1 and G_2 be systems of process equations, and let the set H of process equations be such that $G_i \cup H$ is a system of process equations ($i = 1, 2$). If $G_1 \Rightarrow G_2$, then $G_1 \cup H \Rightarrow G_2 \cup H$, and if $G_1 \Rightarrow_c G_2$, then $G_1 \cup H \Rightarrow_c G_2 \cup H$.*

Definition 2.13. Process definition $(\mathsf{X}(\overrightarrow{t_1}), G_1)$ is *equivalent* to process definition $(\mathsf{Y}(\overrightarrow{t_2}), G_2)$ (notation $(\mathsf{X}(\overrightarrow{t_1}), G_1) = (\mathsf{Y}(\overrightarrow{t_2}), G_2)$) if both $(\mathsf{X}(\overrightarrow{t_1}), G_1) \Rightarrow (\mathsf{Y}(\overrightarrow{t_2}), G_2)$ and $(\mathsf{Y}(\overrightarrow{t_2}), G_2) \Rightarrow (\mathsf{X}(\overrightarrow{t_1}), G_1)$. Similarly, if $(\mathsf{X}(\mathit{pars}(\mathsf{X}, G_1)), G_1) = (\mathsf{Y}(\mathit{pars}(\mathsf{Y}, G_2)), G_2)$ we say that (X, G_1) is *equivalent* to (Y, G_2). The *conditional* equivalence (notation $=_c$) is defined in the same way.

$$\partial_H(b) = b \text{ if } b = \tau \text{ or } (b = \mathsf{a}(\overrightarrow{d}) \text{ and } \mathsf{a} \notin H) \tag{D1}$$
$$\partial_H(b) = \delta \text{ otherwise} \tag{D2}$$
$$\partial_H(x + y) = \partial_H(x) + \partial_H(y) \tag{D3}$$
$$\partial_H(x \cdot y) = \partial_H(x) \cdot \partial_H(y) \tag{D4}$$
$$\partial_H(x \lhd c \rhd \delta) = \partial_H(x) \lhd c \rhd \delta \tag{D5}$$
$$\tau_I(b) = b \text{ if } b = \delta \text{ or } (b = \mathsf{a}(\overrightarrow{d}) \text{ and } \mathsf{a} \notin I) \tag{T1}$$
$$\tau_I(b) = \tau \text{ otherwise} \tag{T2}$$
$$\tau_I(x + y) = \tau_I(x) + \tau_I(y) \tag{T3}$$
$$\tau_I(x \cdot y) = \tau_I(x) \cdot \tau_I(y) \tag{T4}$$
$$\tau_I(x \lhd c \rhd \delta) = \tau_I(x) \lhd c \rhd \delta \tag{T5}$$
$$\rho_R(\delta) = \delta \tag{RD}$$
$$\rho_R(\tau) = \tau \tag{RT}$$
$$\rho_R(\mathsf{a}(\overrightarrow{d})) = R(\mathsf{a})(\overrightarrow{d}) \tag{R1}$$
$$\rho_R(x + y) = \rho_R(x) + \rho_R(y) \tag{R3}$$
$$\rho_R(x \cdot y) = \rho_R(x) \cdot \rho_R(y) \tag{R4}$$
$$\rho_R(x \lhd c \rhd \delta) = \rho_R(x) \lhd c \rhd \delta \tag{R5}$$

Table 6: Axioms for renaming operators in μCRL.

$$(x \mathbin{\|\!\!\lfloor} y) \mathbin{\|\!\!\lfloor} z = x \mathbin{\|\!\!\lfloor} (y \parallel z) \tag{SC1}$$
$$x \mid y = y \mid x \tag{SC3}$$
$$(x \mid y) \mid z = x \mid (y \mid z) \tag{SC4}$$
$$x \mid (y \mathbin{\|\!\!\lfloor} z) = (x \mid y) \mathbin{\|\!\!\lfloor} z \tag{SC5}$$

Table 7: Axioms for Standard Concurrency in μCRL.

Finally, $G_1 = G_2$ if $|G_1| = |G_2|$ and for all $\mathsf{X} \in |G_1|$, $(\mathsf{X}, G_1) = (\mathsf{X}, G_2)$.

Note that on systems of process equations, the relations $=$ and $=_c$ are equivalences, and the relations $\Rightarrow$ and $\Rightarrow_c$ are reflexive and transitive. The following simple examples demonstrate the use of Definitions 2.13 and 2.11.

Example 2.14. Let $G_1 = \{\mathsf{X} = \mathsf{a} \cdot \mathsf{Y}, \mathsf{Y} = \mathsf{b} \cdot \mathsf{X}\}$ and $G_2 = \{\mathsf{X} = \mathsf{a} \cdot \mathsf{b} \cdot \mathsf{X}\}$. We can show that $(\mathsf{X}, G_1) = (\mathsf{X}, G_2)$. The implication from left to the right can be shown by choosing $g_{\mathsf{X}} = \mathsf{X}$. The reverse direction can be shown by choosing $g_{\mathsf{X}} = \mathsf{X}$ and $g_{\mathsf{Y}} = \mathsf{b} \cdot \mathsf{X}$.

Example 2.15. Let $G_1 = \{\mathsf{X}(b{:}Bool) = \mathsf{a}(b) \cdot \mathsf{X}(\neg b)\}$ and $G_2 = \{\mathsf{Y}(n{:}Nat) = \mathsf{a}(even(n)) \cdot \mathsf{Y}(S(n))\}$. We can show that $(\mathsf{X}(\mathbf{t}), G_1) \Rightarrow_c (\mathsf{Y}(0), G_2)$ by choosing $g_{\mathsf{Y}}(n) = \mathsf{X}(even(n))$. In this case we need to show that $\mathsf{X}(\mathbf{t}) = g_{\mathsf{Y}}(0)$ (which follows from $even(0) = \mathbf{t}$) and that $\mathsf{X}(even(n)) = \mathsf{a}(even(n)) \cdot \mathsf{X}(even(S(n)))$. This latter identity follows from $\mathsf{X}(b) = \mathsf{a}(b) \cdot \mathsf{X}(\neg b)$ and the data identity $even(S(n)) = \neg even(n)$. If we assume the existence of a function $n : Bool \to Nat$, defined by $n(\mathbf{t}) = 0$ and $n(\mathbf{f}) = 1$, we can prove that $(\mathsf{X}(b), G_1) \Rightarrow_c (\mathsf{Y}(n(b)), G_2)$ using the same function $g_{\mathsf{Y}}(n)$ and the data identities $even(n(b)) = b$ and $even(S(n(b))) = \neg b$, both of which seem reasonable.

We do not have any of the reverse implications: consider the model with carrier set Nat, in which $\mathsf{a}(b)$ is interpreted as 1, and sequential composition as $+$. Then $\mathsf{Y}(0)$ has many solutions, whereas $\mathsf{X}(\mathbf{t})$ has none.

Below we argue that the basic Definition 2.11 characterizes preservation of solutions.

Proposition 2.16. *Let G_1, G_2 be systems of process equations with $|G_1| = \{\mathsf{X}_1 \ldots \mathsf{X}_n\}$ and $|G_2| = \{\mathsf{Y}_1 \ldots \mathsf{Y}_m\}$. Let $(\mathsf{X}_1(\overrightarrow{t_1}), G_1) \Rightarrow_c (\mathsf{Y}_1(\overrightarrow{u_1}), G_2)$ and let $\mathcal{M}$ be a model of μCRL, Bool, DATA and G_1. If $P \in \mathcal{M}$ is a solution for $\mathsf{X}_1(\overrightarrow{t_1})$ then P is also a solution for $\mathsf{Y}_1(\overrightarrow{u_1})$.*

Proof. Let $P_1 \in \mathcal{M}$ be a solution for $\mathsf{X}_1(\overrightarrow{t_1})$. So, there are processes P_i $(i = 1, \ldots, n)$ that solve the equations of G_1 for $\mathsf{X}_i(\overrightarrow{t_i})$. By $(\mathsf{X}_1(\overrightarrow{t_1}), G_1) \Rightarrow_c (\mathsf{Y}_1(\overrightarrow{u_1}), G_2)$ there are functions g_{Y_i} $(i = 1, \ldots, m)$ such that $\mathcal{M} \models \mathsf{X}_1(\overrightarrow{t_1}) = g_{\mathsf{Y}_1}(\overrightarrow{u_1})$. Furthermore, the derivability of $g_{\mathsf{Y}_j}(\overrightarrow{d_j}') = rhs(\mathsf{Y}_j)\,[\forall k\ \mathsf{Y}_k(t') := g_{\mathsf{Y}_k}(t')]$ $(j = 1, \ldots, m)$ yields that P_1 is also a solution for $\mathsf{Y}_1(\overrightarrow{u_1})$ in G_2. □

The following lemma shows that by applying a μCRL axiom to the right hand side of an equation we get an equivalent system.

Lemma 2.17. *Let p_1, p_2 be process terms such that $p_1 = p_2$. Let G be a system of process equations, and X be a process name in it such that p_1 is a subterm of $rhs(\mathsf{X}, G)$. Let G' consist of equations in G, but in the equation defining X an occurrence of p_1 is replaced by p_2. Then $G = G'$.*

The following lemma shows that by replacing a subterm of the right hand side of an equation by a fresh process name, and adding the equation for it, we get an equivalent process definition for each process name in the original system.

Lemma 2.18. *Let G be a system of process equations, and X be a process name in it. Let p be a subterm of $rhs(\mathsf{X}, G)$ with free data variables $d^1{:}D^1, \ldots, d^n{:}D^n = \overrightarrow{d{:}D}$ in it. Let Y be a process name, $\mathsf{Y} \notin G$. Let G' consist of equations in G, but in the equation defining X an occurrence of p is replaced by $\mathsf{Y}(\overrightarrow{d})$, and the equation $\mathsf{Y}(\overrightarrow{d{:}D}) = p$ is added to G. Then for any $\mathsf{Z} \in |G|$ we have $(\mathsf{Z}, G) = (\mathsf{Z}, G')$.*

Proof. To prove that $(\mathsf{Z}, G) \Rightarrow (\mathsf{Z}, G')$ we take $g_{\mathsf{Z}}(pars(\mathsf{Z})) = \mathsf{Z}(pars(\mathsf{Z}))$ for all $\mathsf{Z} \in |G|$, and $g_{\mathsf{Y}} = p$. To prove the other direction we just take $g_{\mathsf{Z}}(pars(\mathsf{Z})) = \mathsf{Z}(pars(\mathsf{Z}))$ for all $\mathsf{Z} \in |G|$. □

The following lemma shows that under certain conditions we can substitute a process name by its right hand side in a right hand side of an equation.

Lemma 2.19. *Let G be a system of process equations, and X be a process name in it. Let $\mathsf{Y}(\overrightarrow{t})$ be a subterm of $rhs(\mathsf{X}, G)$ for some $\mathsf{Y} \neq \mathsf{X}$. Let G' consist of equations in G, but in the equation defining X an occurrence of $\mathsf{Y}(\overrightarrow{t})$ is replaced by $rhs(\mathsf{Y}, G)[pars(\mathsf{Y}, G) := \overrightarrow{t}]$. Then we have that $G = G'$.*

Proof. In both directions we take the mappings g_{X} to be the identity mappings. □

The following lemma says that we can add dummy data parameters to a process equation, or remove such parameters.

Lemma 2.20. *Let G be a system of process equations, and X be a process name in it with parameters $d^1, \ldots, d^n$. Suppose that d^i does not occur freely in $rhs(\mathsf{X}, G)$. Let G' be as G, but the process name X is replaced by X' and $pars(\mathsf{X}', G') = d^1, \ldots d^{i-1}, d^{i+1}, \ldots d^n$. Then for all $\mathsf{Y} \in |G| \wedge \mathsf{Y} \neq \mathsf{X}$ we have $(\mathsf{Y}, G) = (\mathsf{Y}, G')$, and $(\mathsf{X}(d^1, \ldots, d^n), G) = (\mathsf{X}'(d^1, \ldots d^{i-1}, d^{i+1}, \ldots d^n), G')$.*

Proof. In both directions we take the mappings g_{Y} (for $\mathsf{Y} \neq \mathsf{X}$) to be the identity mappings. In one direction $g_{\mathsf{X}'}(d^1, \ldots d^{i-1}, d^{i+1}, \ldots d^n) = \mathsf{X}(d^1, \ldots d^n)$ and $g_{\mathsf{X}}(d^1, \ldots d^n) = \mathsf{X}'(d^1, \ldots d^{i-1}, d^{i+1}, \ldots d^n)$. □

In many cases we are interested in a process definition (X, G) for a fixed process name X. The following lemma states that we can drop a defining equation for a process name $\mathsf{Y} \neq \mathsf{X}$, in cases when the X does not depend on Y, and Y does not depend on itself, under the condition that the resulting set of equations will form a system of process equations (Definition 2.7).

Lemma 2.21. *Let G be a system of process equations, and X, Y be process names in it such that X does not depend on Y, and Y does not depend on itself. Let G' contain all equations in G except the defining equation for Y. If G' is a system of process equations, then we have $(\mathsf{X}, G) = (\mathsf{X}, G')$.*

Proof. In the direction from left to the right we use the identity mapping for g_{Z}. In the reverse direction we use the same mapping, but $g_{\mathsf{Y}} = rhs(\mathsf{Y}, G)$. □

2.4 Guardedness

In this paper we use a slightly different notion of guardedness as the one used in [16].

Definition 2.22. An occurrence of a process name X in a process term p is *completely guarded* if there is a subterm p' of p of the form $q \cdot p''$ containing this occurrence of X, where q is a process term containing no process names.

A process term is called *completely guarded* if every occurrence of a process name in it is completely guarded. Note that a term that contains no process names is completely guarded.

A system of process equations G is *completely guarded* if for any $\mathsf{X} \in |G|$, $rhs(\mathsf{X}, G)$ is a completely guarded term.

Definition 2.23. A process definition (X, G) is *(unconditionally) guarded* if there is a process definition (X', G') such that G' is a completely guarded system of process equations, and $(\mathsf{X}, G) = (\mathsf{X}', G')$.

Definition 2.24. Let G be a system of process equations. A *Process Name Unguarded-Dependency Graph (PNUDG)* is an oriented graph with the set of nodes $|G|$, and edges defined as follows: $\mathsf{X} \to \mathsf{Y}$ belongs to the graph if Y is not completely guarded in $rhs(\mathsf{X}, G)$.

Lemma 2.25. *If the PNUDG of a finite system of process equations G is acyclic, then G is guarded.*

Proof. Given a system G we replace each unguarded occurrence of a process name by its right hand side. By Lemma 2.19 we get an equivalent system. Due to the fact that PNUDG is acyclic, we need to perform the replacement only finitely many times, and after that we get a completely guarded system. □

The following example shows that the converse of Lemma 2.25 does not hold.

Example 2.26. System G consisting of one equation $\mathsf{X} = \mathsf{X} \lhd \mathbf{f} \rhd \delta$ is guarded, but its PNUDG contains the cycle $\mathsf{X} \to \mathsf{X}$.

2.5 Parallel pCRL

We define (parallel) pCRL processes as a subset of μCRL processes.

Definition 2.27. Let G be a system of process equations. A process term in $Terms(|G|)$ is called a *pCRL process term* in G if it has the syntax

$$p ::= \mathsf{a}(\overrightarrow{t}) \mid \delta \mid \mathsf{Y}(\overrightarrow{t}) \mid p + p \mid p \cdot p \mid \sum_{d:D} p \mid p \lhd c \rhd p \tag{2.1}$$

and can directly depend only on process names whose right hand sides are also pCRL process terms. A process name is called a *pCRL process name* if its right hand side is a pCRL process term.

Definition 2.28. Let G be a system of process equations. A process term in $\mathit{Terms}(|G|)$ is called a *parallel pCRL process term* in G if it has the syntax

$$q ::= \mathsf{Y}(\overrightarrow{t}) \mid q \parallel q \mid \tau_I(q) \mid \partial_H(q) \mid \rho_R(q) \tag{2.2}$$

and directly depends only on process names whose right hand side are pCRL or parallel pCRL process terms. It is called a *parallel pCRL process name* if its right hand side is a parallel pCRL process term.

Example 2.29. Referring to G_1 and G_2 as defined in the previous Example 2.9, $\mathsf{X} + \mathsf{a}$ is a pCRL process term in G_1, and X, $\mathsf{X} \parallel \mathsf{X}$ and $\mathsf{X} \parallel Y$ are parallel pCRL process terms in G_1. Furthermore, $\mathsf{P}(S(n))$ with n a variable of sort Nat and $\mathsf{a}(\mathit{even}(0)) \cdot \mathsf{P}(0)$ are pCRL process terms in G_2. Finally, $\mathsf{X} \parallel \mathsf{a}$ is *not* a (parallel) pCRL process term in G_1.

In the following definition we define what a parallel pCRL process definition is. For this definition we assume that we have a μCRL specification that is Statically Semantically Correct (cf. [17]), that is, in which the data types, actions, communication functions and processes are all well-defined. The first two restrictions posed in the definition below distinguish parallel pCRL as a subset of μCRL. The third one is present to disallow parallel process names on which the head process name does not depend.

Definition 2.30. Let G be a finite system of process equations, and (X, G) be a process definition. (X, G) is called a *parallel pCRL process definition* if X is a (parallel) pCRL process name, and

- all of the process names in G are either pCRL or parallel pCRL process names;
- no parallel pCRL process name depends on itself;
- process name X depends on all parallel pCRL process names in G, but not on itself.

It is called a *pCRL system of process equations* if all process names in it are pCRL process names.

It follows from Definitions 2.30 and 2.28 that for every (parallel) pCRL process definition (X, G), either X is a pCRL process name, or it depends on a pCRL process name in G.

Example 2.31. Referring to G_1 as defined in Example 2.9, (Z, G_1) is a parallel pCRL process definition, but (X, G_1) is not.

3. Transformation to Extended Greibach Normal Form

As the input for the linearization procedure we take a (parallel) pCRL process definition (X, G) such that PNUDG of G is acyclic. The system of process equations G can be partitioned in two parts: G_1 and G_2, where G_1 has pCRL equations, and G_2 parallel pCRL equations. G_2 can be empty, in which case X is a pCRL process name. Otherwise X is a parallel pCRL process name.

In this section we transform G_1 into a system of process equations G'_1 in Extended Greibach Normal Form. The resulting system will contain process equations for all process names in $|G_1|$ with the same names and types of data parameters involved, as well as, possibly, other process equations. After that we need to linearize the process definition (X, G'), where $G' = G'_1 \cup G_2$.

Below we define the Extended Greibach Normal Form (EGNF) and pre-Extended Greibach Normal Form (pre-EGNF). From this point on we assume that $\mathsf{a}(\overrightarrow{t})$ with possible indices can also be an abbreviation for τ. This is done to make the normal form representations more concise.

Definition 3.1. A pCRL process equation is in *pre-EGNF* iff it is of the form:

$$\mathsf{X}(\overrightarrow{d{:}D}) = \sum_{i \in I} \sum_{\overrightarrow{e_i : E_i}} p_i(\overrightarrow{d, e_i}) \lhd c_i(\overrightarrow{d, e_i}) \rhd \delta$$

where $p_i(\overrightarrow{d,e_i})$ are terms of the following syntax:

$$p ::= \mathsf{a}(\overrightarrow{t}) \mid \mathsf{Y}(\overrightarrow{t}) \mid \mathsf{a}(\overrightarrow{t}) \cdot p \mid \mathsf{Y}(\overrightarrow{t}) \cdot p \tag{3.0}$$

A pCRL process equation is in *EGNF* iff it is of the form:

$$\begin{aligned}\mathsf{X}(\overrightarrow{d{:}D}) = &\sum_{i\in I} \sum_{\overrightarrow{e_i:E_i}} \mathsf{a}_i(\overrightarrow{f_i}(\overrightarrow{d,e_i})) \cdot p_i(\overrightarrow{d,e_i}) \lhd c_i(\overrightarrow{d,e_i}) \rhd \delta + \\ &\sum_{j\in J} \sum_{\overrightarrow{e_j:E_j}} \mathsf{a}_j(\overrightarrow{f_j}(\overrightarrow{d,e_j})) \lhd c_j(\overrightarrow{d,e_j}) \rhd \delta\end{aligned}$$

where I and J are disjoint, and all $p_i(\overrightarrow{d,e_i})$ are terms of the following syntax:

$$p ::= \mathsf{Y}(\overrightarrow{t}) \mid \mathsf{Y}(\overrightarrow{t}) \cdot p$$

Finally, a finite system of process equations is in (pre-)EGNF iff all its equations are.

Note (Sum Notation). Apart from functions $\sum_{d:D} p$ that are included in the syntax of process terms, we use the following abbreviations. Expression $\sum_{\overrightarrow{d:D}}$ is an abbreviation for $\sum_{d^1:D^1} \cdots \sum_{d^n:D^n}$. In case $n = 0$, $\sum_{\overrightarrow{d:D}} p$ is an abbreviation for p. Expression $\sum_{i\in I} p_i$, where I is a finite set, is an abbreviation for $p_{i_1} + \cdots + p_{i_n}$ such that $\{i_1, \ldots, i_n\} = I$. In case $I = \emptyset$, $\sum_{i\in I} p_i$ is an abbreviation for δ.

Note (Conditions). As follows from the above definition, any process equation in pre-EGNF or EGNF must have a condition in each summand. However, this is not a necessary restriction. In case a summand q does not have a condition, it is an abbreviation for $q \lhd \mathbf{t} \rhd \delta$.

3.1 Preprocessing

We first transform G_1 into G_1^1. This can be seen as a preprocessing step that possibly renames bound data variables. For instance $\sum_{d:D}((\sum_{d:E} \mathsf{a}(d)) \cdot \mathsf{b}(d))$ is replaced by $\sum_{d:D}((\sum_{e:E} \mathsf{a}(e)) \cdot \mathsf{b}(d))$, where e is a fresh variable. We replace each equation $\mathsf{X}(\overrightarrow{d_\mathsf{X}{:}D_\mathsf{X}}) = p_\mathsf{X}$ in G_1 with the equation $X(\overrightarrow{d_\mathsf{X}{:}D_\mathsf{X}}) = S_0(\{\overrightarrow{d_\mathsf{X}}\}, p_\mathsf{X})$, where $S_0 : DVar \times Terms(|G_1|) \to Terms(|G_1|)$ is defined in the following way:

$$S_0(S, f(p^1, \ldots, p^n)) \to f(S_0(S, p^1), \ldots, S_0(S, p^n)) \text{ if } f \text{ is not } \sum_{d:D}$$

$$S_0\Big(S, \sum_{d:D} p\Big) \to \begin{cases} \sum_{d:D} S_0(S \cup \{d\}, p) & \text{if } d \notin S \\ \sum_{e:D} S_0(S \cup \{e\}, p[d := e]) & \text{if } d \in S \end{cases}$$

where e is a fresh variable.

Proposition 3.2. *Let G_1^1 be the result of applying the preprocessing to G_1. Then $G_1^1 = G_1$.*

Proof. The statement follows from Lemma 2.17 if we apply axiom (SUM2). □

As can easily be seen, the preprocessing step does not increase the size or the number of equations in the system.

$$
\begin{aligned}
p &::= \mathsf{a}(\overrightarrow{t}) \mid \delta \mid \mathsf{X}(\overrightarrow{t}) \mid p^1 \cdot p \mid p^2 + p^2 \mid p^3 \lhd c \rhd \delta \mid \sum_{d:D} p^4 \\
p^1 &::= \mathsf{a}(\overrightarrow{t}) \mid \mathsf{X}(\overrightarrow{t}) \mid p^1 \cdot p \mid p^2 + p^2 \\
p^2 &::= \mathsf{a}(\overrightarrow{t}) \mid \mathsf{X}(\overrightarrow{t}) \mid p^1 \cdot p \mid p^2 + p^2 \mid p^3 \lhd c \rhd \delta \mid \sum_{d:D} p^4 \\
p^3 &::= \mathsf{a}(\overrightarrow{t}) \mid \mathsf{X}(\overrightarrow{t}) \mid p^1 \cdot p \\
p^4 &::= \mathsf{a}(\overrightarrow{t}) \mid \mathsf{X}(\overrightarrow{t}) \mid p^1 \cdot p \mid p^3 \lhd c \rhd \delta \mid \sum_{d:D} p^4
\end{aligned}
$$

Table 8: Syntax of terms after simple rewriting.

3.2 Reduction by Simple Rewriting

By applying term rewriting we get an equivalent set of process equations to the given one, but with terms in right hand sides having the more restricted form as presented in Table 8.

The rewrite rules that we apply to the right hand sides of the equations are listed in Table 9. The symbols $\sum_{d:D}$ are treated in this rewrite system as function symbols, not as binders. This is justified by the fact that we have renamed all nested bound variables, which allows the use of first order term rewriting. We call the function induced by the rewrite rules $rewr : \mathit{Terms}(|G|) \to \mathit{Terms}(|G|)$ for a given system of process equations G.

$$x + \delta \to x \tag{RA6}$$

$$\delta \cdot x \to \delta \tag{RA7}$$

$$\sum_{d:D} \delta \to \delta \tag{RSUM1$'$}$$

$$\sum_{d:D}(x+y) \to \sum_{d:D} x + \sum_{d:D} y \tag{RSUM4}$$

$$\Big(\sum_{d:D} x\Big) \cdot y \to \sum_{d:D}(x \cdot y) \tag{RSUM5}$$

$$\Big(\sum_{d:D} x\Big) \lhd c \rhd \delta \to \sum_{d:D} x \lhd c \rhd \delta \tag{RSUM12}$$

$$\delta \lhd c \rhd \delta \to \delta \tag{RCOND0$'$}$$

$$(x \lhd c_1 \rhd \delta) \lhd c_2 \rhd \delta \to x \lhd c_1 \wedge c_2 \rhd \delta \tag{RCOND4}$$

$$(x + y) \lhd c \rhd \delta \to x \lhd c \rhd \delta + y \lhd c \rhd \delta \tag{RCOND7}$$

$$(x \lhd c \rhd \delta) \cdot y \to (x \cdot y) \lhd c \rhd \delta \tag{RCOND6}$$

Table 9: Rewrite rules defining *rewr*.

Before applying the rewriting we eliminate all terms of the form $\_ \lhd \_ \rhd \_$ with the third argument being different from δ with the following rule:

$$y \not\equiv \delta \implies x \lhd c \rhd y \to x \lhd c \rhd \delta + y \lhd \neg c \rhd \delta \tag{RCOND3}$$

The rewriting is performed modulo the following rules:

$$\begin{aligned} x + y &= y + x \\ x + (y + z) &= (x + y) + z \\ (x \cdot y) \cdot z &= x \cdot (y \cdot z) \end{aligned}$$

The optimization rules presented in Table 10 are not needed to get the desired restricted syntactic form, but can be used to simplify the terms. They could be applied with higher priority than the rules in Table 9 to achieve possible reductions. Note that the rule (RSCA′) could lead to optimizations only in cases when x is completely guarded, and y or z are not.

$$x + x \to x \tag{RA3}$$
$$x \lhd c \rhd x \to x \tag{RCOND0}$$
$$x \lhd \mathbf{t} \rhd y \to x \tag{RCOND1}$$
$$x \lhd \mathbf{f} \rhd y \to y \tag{RCOND2}$$
$$x \lhd c_1 \rhd \delta + x \lhd c_2 \rhd \delta \to x \lhd c_1 \vee c_2 \rhd \delta \tag{RCOND5}$$
$$(x_1 \lhd c \rhd x_2) \cdot (y_1 \lhd c \rhd y_2) \to x_1 \cdot y_1 \lhd c \rhd x_2 \cdot y_2 \tag{RSCA}$$
$$x \cdot (y \lhd c \rhd z) \to x \cdot y \lhd c \rhd x \cdot z \tag{RSCA′}$$

Table 10: Optimization rules.

Proposition 3.3. *The commutative/associative term rewriting system of Table 9 is strongly terminating.*

Proof. Termination can be proved by using the following order on the operations: $\_ \lhd c \rhd \_ > \cdot > \_ \lhd c \rhd \delta > \sum > +$. □

Lemma 3.4. *For any process term p not containing $p_1 \lhd c \rhd p_2$, where $p_2 \not\equiv \delta$, we have that $rewr(p)$ has the syntax defined in Table 8.*

Proof. Let $q = rewr(p)$. It can be seen from the rewrite rules that they preserve the syntax in Definition 2.27. Suppose q does not satisfy the syntax defined in Table 8. The following possibilities exist, and all of them imply that q is reducible.

- $q = \delta \cdot p_1$. Can be reduced by (RA7).
- $q = (p_1 \lhd c \rhd \delta) \cdot p_2$. Can be reduced by (RCOND6).
- $q = (\sum_{d:D} p_1) \cdot p_2$. Can be reduced by (RSUM5).
- $q = \delta + p_1$. Can be reduced by (RA6).
- $q = \delta \lhd c \rhd \delta$. Can be reduced by (RCOND0′).
- $q = (p_1 + p_2) \lhd c \rhd \delta$. Can be reduced by (RCOND7).
- $q = (p_1 \lhd c_1 \rhd \delta) \lhd c_2 \rhd \delta$. Can be reduced by (RCOND4).
- $q = (\sum_{d:D} p_1) \lhd c \rhd \delta$. Can be reduced by (RSUM12).

- $q = \sum_{d:D} \delta$. Can be reduced by (RSUM1′).
- $q = \sum_{d:D}(p_1 + p_2)$. Can be reduced by (RSUM4).

□

Proposition 3.5. *Let G_1^2 be the result of applying the rewriting to G_1^1. Then $G_1^2 = G_1^1$.*

Proof. Taking into account that G_1^1 does not contain nested occurrences of bound variables, each rewrite rule is a consequence of the axioms of μCRL. By Lemma 2.17 we get $G_1^2 = G_1^1$. □

As the result of applying simple rewriting the number of equations obviously remains the same. The process terms may grow with a constant factor, but the number of occurrences of action labels and process names does not increase. The data terms and the number of their occurrences may grow with a constant factor, too.

3.3 Adding New Process Equations

In this step we reduce the complexity of terms in the right hand sides of the G_1^2 equations even further by the introduction of new process equations. In some cases we take a subterm of a right hand side and substitute it by a fresh process name parameterized by (at least) all free variables that appear in that subterm. As the result we get a system of process equations G_1^3 with equations in pre-EGNF. Such a transformation can be done for all equations $\mathsf{X}(\overrightarrow{d_\mathsf{X}{:}D_\mathsf{X}}) = p_\mathsf{X}$ by replacing them with $\mathsf{X}(\overrightarrow{d_\mathsf{X}{:}D_\mathsf{X}}) = S_1(\overrightarrow{d_\mathsf{X}{:}D_\mathsf{X}}, p_\mathsf{X})$.

$$
\begin{aligned}
S_1(S, \mathsf{a}(\overrightarrow{t})) &\to \mathsf{a}(\overrightarrow{t})\\
S_1(S, \delta) &\to \delta\\
S_1(S, \mathsf{X}(\overrightarrow{t})) &\to \mathsf{X}(\overrightarrow{t})\\
S_1(S, p_1 \cdot p_2) &\to S_2(S, p_1 \cdot p_2)\\
S_1(S, p_1 + p_2) &\to S_1(S, p_1) + S_1(S, p_2)\\
S_1(S, p \lhd c \rhd \delta) &\to S_2(S, p) \lhd c \rhd \delta\\
S_1\Big(S, \sum_{d:D} p\Big) &\to \sum_{d:D} S_1(S \mathbin{\&} d{:}D, p)\\
S_2(S, \mathsf{a}(\overrightarrow{t})) &\to \mathsf{a}(\overrightarrow{t})\\
S_2(S, \delta) &\to (\mathsf{Y} := \mathit{fresh\_var});\, \mathit{add}\big(\mathsf{Y} = \delta\big)\\
S_2(S, \mathsf{X}(\overrightarrow{t})) &\to \mathsf{X}(\overrightarrow{t})\\
S_2(S, p_1 \cdot p_2) &\to S_2(S, p_1) \cdot S_2(S, p_2)\\
S_2(S, p_1 + p_2) &\to (\mathsf{Y} := \mathit{fresh\_var})(S);\, \mathit{add}\big(\mathsf{Y}(S) = S_1(S, p_1 + p_2)\big)\\
S_2(S, p \lhd c \rhd \delta) &\to (\mathsf{Y} := \mathit{fresh\_var})(S);\, \mathit{add}\big(\mathsf{Y}(S) = S_1(S, p \lhd c \rhd \delta)\big)\\
S_2\Big(S, \sum_{d:D} p\Big) &\to (\mathsf{Y} := \mathit{fresh\_var})(S);\, \mathit{add}\Big(\mathsf{Y}(S) = S_1\Big(S, \sum_{d:D} p\Big)\Big)
\end{aligned}
$$

Here *fresh_var* represents a fresh process name, and *add* represents addition of the equation to the resulting system. Thus formally, S_1 and S_2 operate on sets of equations, not on equations themselves. In the following we provide a simple example of the transformation.

Example 3.6. Let $G = \{\mathsf{X}(d{:}D) = \mathsf{a}(d) \cdot (\mathsf{b}(d) + \mathsf{X}(f(d)))\}$ be a given system of process equations. After applying the transformation we get the system $G' = \{\mathsf{X}(d{:}D) = \mathsf{a}(d) \cdot \mathsf{Y}(d), \mathsf{Y}(d{:}D) = \mathsf{b}(d) + \mathsf{X}(f(d))\}$ which is in pre-EGNF.

Proposition 3.7. *The functions S_1 and S_2 are well defined.*

Proof. Using the order on the operations $S_1 > +, S_1 > \sum, S_2 > \cdot$ it can be shown that the infinite recursion is not possible for any admissible arguments given. □

Lemma 3.8. *All process equations in G_1^3 are in pre-EGNF.*

Proof. It is easy to see that S_2 produces terms that satisfy the syntax (3.0) from Definition 3.1. The transformation S_1 can add only $+$, $\sum$ or $\lhd\rhd$ operations to them at the correct places. The only interesting transformation to consider is $S_1\Big(S, \sum_{d:D} p\Big) \to \sum_{d:D} S_1(S \mathbin{\&} d{:}D, p)$, as we need to show that p is not of the form $p_1 + p_2$. This follows from the fact that p satisfies the syntax defined in Table 8. □

Proposition 3.9. *For any process name X in G_1^2 we have $(\mathsf{X}, G_1^3) = (\mathsf{X}, G_1^2)$.*

Proof. The statement follows from Lemma 2.18. □

The transformation described in this subsection does not increase the size of terms. The number of processes may increase linearly in the size of terms in the original system.

3.4 Guarding

Next we transform the equations of G_1^3 in such a way that each sequential term starts with an action (or τ). To this end, we define the function $guard : DVar \times Terms(|G|) \to Terms(|G|)$ in the following way:

$$guard\Big(S, \sum_{i\in I} \sum_{\overrightarrow{e_i:E_i}} p_i \lhd c_i \rhd \delta\Big) = rewr\Big(\sum_{i\in I} \sum_{\overrightarrow{e_i:E_i}} guard(S \cup \{\overrightarrow{e_i}\}, p_i) \lhd c_i \rhd \delta\Big)$$
$$guard(S, \mathsf{a}(\overrightarrow{t})) = \mathsf{a}(\overrightarrow{t})$$
$$guard(S, \mathsf{Y}(\overrightarrow{t})) = guard\Big(S, S_0\big(S \setminus \{pars(\mathsf{Y})\}, rhs(\mathsf{Y})\big)\big[pars(\mathsf{Y}) := \overrightarrow{t}\,\big]\Big)$$
$$guard(S, p_1 \cdot p_2) = rewr'(guard(S, p_1) \cdot p_2)$$

Here we use functions *rewr* and S_0 from previous subsections. The function $rewr'$ represents the rewrite system of *rewr* extended with the following rule.

$$(x + y) \cdot z \to x \cdot z + y \cdot z \tag{RA4}$$

Proposition 3.10. *For any finite system G_1^3 with acyclic PNUDG, and any process name X in it, the function guard is well-defined on $rhs(\mathsf{X}, G_1^3)$.*

Proof. Let n be the number of equations in G_1^3, and m be the maximal number of process names in sequences p_i for all $i \in I$. Suppose that *guard* is applied more than $n \cdot m$ times on a term. This means that a process name Y is substituted more than once, which contradicts to the fact that PNUDG is acyclic. □

We define the system G_1^4 in the following way. For each equation

$$\mathsf{X}(\overrightarrow{d{:}D}) = \sum_{i\in I} \sum_{\overrightarrow{e_i:E_i}} p_i(\overrightarrow{d, e_i}) \lhd c_i(\overrightarrow{d, e_i}) \rhd \delta$$

in G_1^3 we put

$$\mathsf{X}(\overrightarrow{d{:}D}) = \mathit{guard}\Big(\{\overrightarrow{d{:}D}\}, \sum_{i\in I} \sum_{\overrightarrow{e_i:E_i}} p_i(\overrightarrow{d,e_i}) \lhd c_i(\overrightarrow{d,e_i}) \rhd \delta\Big)$$

into G_1^4.

Lemma 3.11. *The equations in G_1^4 are in pre-EGNF and all sequential process terms in the right hand sides of its equations start with an action.*

Proof. Due to Proposition 3.10 we can apply induction on the definition of *guard*. The second and third clauses of the definition are trivial. The first one is brought to the desired form by applying (RCOND4) and (RSUM4) from Table 9. The fourth clause is brought to the desired form by applying (RA4), and then (RSUM5) and (RCOND6) from Table 9. □

Proposition 3.12. *Let G_1^3 and G_1^4 be defined as above. Then $G_1^3 = G_1^4$.*

Proof. According to Lemma 2.19 and Lemma 2.17 all transformations performed by *guard* lead to equivalent systems. We note that care has been taken to rename some data variables during the substitution (in the third clause of *guard* definition) in order to make the substitution and the following applications of the axioms sound. □

The transformation performed in this step does not increase the number of equations, but their sizes may grow exponentially, due to application of (RA4). An example of such an exponential growth is given below.

Example 3.13. Let n be a natural number and let the system of process equations G contain the following n equations.

$$\mathsf{X}_0 = \mathsf{a} + \mathsf{b}$$
$$\dots$$
$$\mathsf{X}_n = \mathsf{X}_{n-1} \cdot \mathsf{a} + \mathsf{X}_{n-1} \cdot \mathsf{b}$$

By induction on n it is easy to show that after applying guarding we get $\mathsf{X}_n = \sum_{p\in\{\mathsf{a},\mathsf{b}\}^n} p$ where $\{\mathsf{a},\mathsf{b}\}^n$ is a set of all strings of length n consisting of a and b occurrences. Indeed, for $n = 0$ this is trivial. For $n > 0$ we get

$$\mathsf{X}_n = \left(\sum_{p\in\{\mathsf{a},\mathsf{b}\}^{n-1}} p\right) \cdot \mathsf{a} + \left(\sum_{p\in\{\mathsf{a},\mathsf{b}\}^{n-1}} p\right) \cdot \mathsf{b} = \sum_{p\in\{\mathsf{a},\mathsf{b}\}^{n-1}} (p \cdot \mathsf{a}) + \sum_{p\in\{\mathsf{a},\mathsf{b}\}^{n-1}} (p \cdot \mathsf{b}) = \sum_{p\in\{\mathsf{a},\mathsf{b}\}^{n}} p$$

This example shows that the term in the right hand side of the equation for X_n contains 2^n summands after the transformation.

3.5 Postprocessing

Finally, we transform all equations of G_1^4 into EGNF. This transformation can be seen as a simple postprocessing step in which we eliminate all actions that appear not leftmost in the right hand sides in the equations. This elimination is obtained by introducing a new process name X_a for each action a that occurs inside the process terms p_i, with parameters corresponding to those of the action. Thus we add equations $\mathsf{X}_\mathsf{a}(\overrightarrow{d_\mathsf{a}{:}D_\mathsf{a}}) = \mathsf{a}(\overrightarrow{d_\mathsf{a}})$ to the system, and replace the occurrences of the action $\mathsf{a}(\overrightarrow{t})$ by $\mathsf{X}_\mathsf{a}(\overrightarrow{t})$.

Proposition 3.14. *Let the system G_1' of process equations be obtained after the postprocessing of the system G_1^4 as described above. Then for all $\mathsf{X} \in G_1^4$ we have $(\mathsf{X}, G_1') = (\mathsf{X}, G_1^4)$ and G_1' is in EGNF.*

Proof. According to Lemma 2.18 this transformation is correct and leads to a system that obviously is in EGNF. □

As a possible optimization during the postprocessing step, the following slightly different strategy can be applied. If we encounter a subterm $\mathsf{a} \cdot \mathsf{Y}$ in p_i, we replace it by a new process name (with the parameters for both a and Y), and add the equation for it to the system. This optimization goes along the lines of a so-called *regular linearization procedure* (see Conclusion), which is a more general case of such an optimization.

Summary. In this section we described the transformation of a finite system $G = G_1 \cup G_2$ with acyclic PNUDG and G_2 containing all parallel pCRL process equations into a system $G' = G_1' \cup G_2$ with G_1' in EGNF. For each $\mathsf{X} \in |G_1|$,

$$\begin{array}{rcll}
(\mathsf{X}, G_1) & = & (\mathsf{X}, G_1^1) & (\text{“Preprocessing”, by Proposition 3.2}) \\
& = & (\mathsf{X}, G_1^2) & (\text{“Rewriting”, by Proposition 3.5}) \\
& = & (\mathsf{X}, G_1^3) & (\text{“Adding new equations”, by Proposition 3.9}) \\
& = & (\mathsf{X}, G_1^4) & (\text{“Guarding”, by Proposition 3.12}) \\
& = & (\mathsf{X}, G_1') & (\text{“Postprocessing”, by Proposition 3.14}).
\end{array}$$

By Lemma 2.12 it follows that $(\mathsf{X}, G) = (\mathsf{X}, G')$ for each $\mathsf{X} \in |G|$.

4. From EGNF to LPE

In this section we transform the system of process equations $G' = G_1' \cup G_2$ where G_1' is in EGNF (cf. Definition 3.1) into $G'' = G_1'' \cup G_2'$, where

- G_1'' consists of a single linear process equation with a specially constructed parameter list;
- if G_2 is not empty, it is transformed into G_2' with the same set $|G_2|$ of process names, but taking the effect of the transformation from G_1' into G_1'' into account (references to G_1' process identifiers may have to be adapted).

Definition 4.1. A process equation is called a *linear process equation (LPE)* if it is of the form

$$\begin{aligned}
\mathsf{X}(\overrightarrow{d{:}D}) = & \sum_{i \in I} \sum_{\overrightarrow{e_i:E_i}} \mathsf{a}_i(\overrightarrow{f_i}(\overrightarrow{d, e_i})) \cdot \mathsf{X}(\overrightarrow{g_i}(\overrightarrow{d, e_i})) \lhd c_i(\overrightarrow{d, e_i}) \rhd \delta + \\
& \sum_{j \in J} \sum_{\overrightarrow{e_j:E_j}} \mathsf{a}_j(\overrightarrow{f_j}(\overrightarrow{d, e_j})) \lhd c_j(\overrightarrow{d, e_j}) \rhd \delta
\end{aligned}$$

where I and J are disjoint sets of indices.

We note that the transformation described in this section is *uni-directional*, i.e., is formulated in terms of $\Rightarrow_c$. We again give counter examples for the associated reverse implications.

4.1 Formal Parameters Harmonization

In this subsection we make the formal parameters of all (non-parallel) pCRL process names in G_1' to be the same, and adapt the parallel pCRL equations in G_2 in an appropriate way. This is done to be able to compress all (non-parallel) pCRL equations in one process equation. The harmonization is defined by the following steps.

1. We rename the data variables with the same names, but different types in different processes. This can be easily done (see Section 3.1).

2. We create the common list of data parameters $\overrightarrow{d{:}D}$ by taking the *set* of all data parameters in the pCRL equations, and giving some order to it.

3. For each pCRL process name X in G_1' we define a mapping M_{X} from its parameter list $\overrightarrow{D_{\mathsf{X}}}$ to the common parameter list $\overrightarrow{D}$. This mapping is such that each newly created parameter is a constant. (Recall that a correct μCRL specification contains constants for each declared data sort.)

4. Then we replace all left hand sides of the pCRL process equations $\mathsf{X}(\overrightarrow{d_{\mathsf{X}}{:}D_{\mathsf{X}}})$ by $\mathsf{X}(\overrightarrow{d{:}D})$, and *all* pCRL process name occurrences $\mathsf{Y}(\overrightarrow{t})$ in the right hand sides of *all* the equations in G' by $\mathsf{Y}(M_{\mathsf{Y}}(\overrightarrow{t}))$.

Proposition 4.2. *Let the system $G_1^5 \cup G_2^1$ of process equations be obtained after harmonization of the system $G_1' \cup G_2$ as described above. Then for all $\mathsf{X} \in |G_1'|$, $(\mathsf{X}(\overrightarrow{d{:}D}), G_1^5) = (\mathsf{X}(\overrightarrow{d_{\mathsf{X}}{:}D_{\mathsf{X}}}), G_1')$, and for all $\mathsf{X} \in |G_2|$, $(\mathsf{X}, G_1^5 \cup G_2^1) = (\mathsf{X}, G_1' \cup G_2)$.*

Proof. By Lemma 2.20 it follows that this transformation yields an equivalent system of equations. $\square$

We remark that a more optimal strategy than 'global harmonization' is to merge as many data parameters as possible. This can be achieved by renaming parameters of some processes so that they match the parameters of other processes, and therefore are not introduced in the general parameter list. In this case the number of parameters of some type s in the general list will be the maximal number of parameters of this type in an equation. A drawback of this optimization is the fact that we may lose parameter name information for some process names.

4.2 Making One Process Equation

Let G_1^5 be a system of n pCRL process equations in EGNF with the same formal parameters.

$$
\begin{aligned}
\mathsf{X}^1(\overrightarrow{d{:}D}) = &\sum_{i \in I^1} \sum_{\overrightarrow{e_i:E_i^1}} \mathsf{a}_i^1(\overrightarrow{f_i^1}(\overrightarrow{d,e_i})) \cdot p_i^1(\overrightarrow{d,e_i}) \triangleleft c_i^1(\overrightarrow{d,e_i}) \triangleright \delta + \\
&\sum_{j \in J^1} \sum_{\overrightarrow{e_j:E_j^1}} \mathsf{a}_j^1(\overrightarrow{f_j^1}(\overrightarrow{d,e_j})) \triangleleft c_j^1(\overrightarrow{d,e_j}) \triangleright \delta \\
\ldots & \\
\mathsf{X}^n(\overrightarrow{d{:}D}) = &\sum_{i \in I^n} \sum_{\overrightarrow{e_i:E_i^n}} \mathsf{a}_i^n(\overrightarrow{f_i^n}(\overrightarrow{d,e_i})) \cdot p_i^n(\overrightarrow{d,e_i}) \triangleleft c_i^n(\overrightarrow{d,e_i}) \triangleright \delta + \\
&\sum_{j \in J^n} \sum_{\overrightarrow{e_j:E_j^n}} \mathsf{a}_j^n(\overrightarrow{f_j^n}(\overrightarrow{d,e_j})) \triangleleft c_j^n(\overrightarrow{d,e_j}) \triangleright \delta
\end{aligned}
$$

We define the system G_1^6 as a single EGNF process equation in the following way:

$$\begin{aligned}
\mathsf{X}(s{:}State, \overrightarrow{d{:}D}) = &\sum_{i \in I^1} \sum_{\overrightarrow{e_i{:}E_i^1}} \mathsf{a}_i^1(\overrightarrow{f_i^1}(\overrightarrow{d, e_i})) \cdot S(p_i^1(\overrightarrow{d, e_i})) \lhd c_i^1(\overrightarrow{d, e_i}) \wedge s = 1 \rhd \delta + \\
&\sum_{j \in J^1} \sum_{\overrightarrow{e_j{:}E_j^1}} \mathsf{a}_j^1(\overrightarrow{f_j^1}(\overrightarrow{d, e_j})) \lhd c_j^1(\overrightarrow{d, e_j}) \wedge s = 1 \rhd \delta \\
&+ \cdots + \\
&\sum_{i \in I^n} \sum_{\overrightarrow{e_i{:}E_i^n}} \mathsf{a}_i^n(\overrightarrow{f_i^n}(\overrightarrow{d, e_i})) \cdot S(p_i^n(\overrightarrow{d, e_i})) \lhd c_i^n(\overrightarrow{d, e_i}) \wedge s = n \rhd \delta + \\
&\sum_{j \in J^n} \sum_{\overrightarrow{e_j{:}E_j^n}} \mathsf{a}_j^n(\overrightarrow{f_j^n}(\overrightarrow{d, e_j})) \lhd c_j^n(\overrightarrow{d, e_j}) \wedge s = n \rhd \delta
\end{aligned}$$

where $S(\mathsf{X}^s(\overrightarrow{t})) = \mathsf{X}(s, \overrightarrow{t})$, and $S(\mathsf{X}^s(\overrightarrow{t}) \cdot p) = \mathsf{X}(s, \overrightarrow{t}) \cdot S(p)$.

The data type *State* is an enumerated data type with equality predicate. Natural numbers are normally used for *State*, though a finite data type is, of course, sufficient.

Let the system $G_1^5 \cup G_2^1$ of process equations be obtained after harmonization of the system $G_1' \cup G_2$ as described above. Then for all $\mathsf{X} \in |G_1'|$, $(\mathsf{X}(\overrightarrow{d{:}D}), G_1^5) = (\mathsf{X}(\overrightarrow{d_\mathsf{X}{:}D_\mathsf{X}}), G_1')$, and for all $\mathsf{X} \in |G_2|$, $(\mathsf{X}, G_1^5 \cup G_2^1) = (\mathsf{X}, G_1' \cup G_2)$. During the current step we construct the system G_1^6 consisting of the single equation for X and the set G_2^2 being G_2^1 with all pCRL process terms $\mathsf{X}^i(\overrightarrow{t})$ replaced by $\mathsf{X}(i, \overrightarrow{t})$ for each $1 \leq i \leq n$.

Proposition 4.3. *Let G_1^5 be a system of n process equations in EGNF, each with formal parameters $\overrightarrow{d{:}D}$, and let State enumerate $1, \ldots, n$. Let furthermore $G_1^5 \cup G_2^1$ be a system of parallel pCRL process equations and $G_1^6 \cup G_2^2$ be the result of the transformation described above. Then for any $s{:}State$, data term vector $\overrightarrow{t}$, and any $\mathsf{X} \in |G_1^5|$, $(\mathsf{X}(s, \overrightarrow{t}), G_1^6) =_c (\mathsf{X}^s(\overrightarrow{t}), G_1^5)$. Finally, for each $\mathsf{X} \in |G_2^1|$, $(\mathsf{X}, G_1^6 \cup G_2^2) =_c (\mathsf{X}, G_1^5 \cup G_2^1)$.*

Proof. The equivalence is easy to derive with the following functions: $g_{\mathsf{X}^i}(\overrightarrow{t}) = \mathsf{X}(i, \overrightarrow{t})$ for each $i{:}State$, and $g_\mathsf{X}(s, \overrightarrow{t}) = \mathsf{X}^s(\overrightarrow{t})$. Note that identities of sort *State* are used in the derivations. □

4.3 Introduction of a Stack

The final step in the linearization of pCRL processes consists of the introduction of a stack parameter which allows to model a sequential composition of process names with parameters as a single process term. In the case that such sequential compositions do not occur in the equation, we do not apply this step. For the particular transformation described here, it is necessary that the process equation to be transformed is data-parametric. This need not be the case after application of all preceding transformation steps. For instance the equation $\mathsf{X} = \mathsf{a} \cdot \mathsf{X} \cdot \ldots \cdot \mathsf{X} + \mathsf{b}$ does not have a data parameter. In this case we need to add a dummy data parameter (over a singleton data type, cf. Lemma 2.20) to apply the following transformation.

Let G_1^6 be a single pCRL process equation in EGNF:

$$\begin{aligned}
\mathsf{X}(\overrightarrow{d{:}D}) = &\sum_{i \in I} \sum_{\overrightarrow{e_i{:}E_i}} \mathsf{a}_i(\overrightarrow{f_i}(\overrightarrow{d, e_i})) \cdot \mathsf{X}(\overrightarrow{t_i^1}) \cdot \ldots \cdot \mathsf{X}(\overrightarrow{t_i^{n_i}}) \lhd c_i(\overrightarrow{d, e_i}) \rhd \delta + \\
&\sum_{j \in J} \sum_{\overrightarrow{e_j{:}E_j}} \mathsf{a}_j(\overrightarrow{f_j}(\overrightarrow{d, e_j})) \lhd c_j(\overrightarrow{d, e_j}) \rhd \delta
\end{aligned}$$

We define G''_1 by the single process equation for Z in the following way:

$$\begin{aligned}
&\mathsf{Z}(st{:}Stack, \overrightarrow{d{:}D}) = \\
&\quad \sum_{i\in I}\sum_{\overrightarrow{e_i:E_i}} \mathsf{a}_i(\overrightarrow{f_i}(\overrightarrow{d,e_i}))\cdot \mathsf{Z}(push(\overrightarrow{t_i^2},\ldots,push(\overrightarrow{t_i^{n_i}},st)\ldots),\overrightarrow{t_i^1}) \lhd c_i(\overrightarrow{d,e_i}) \rhd \delta \\
&\quad +\sum_{j\in J}\sum_{\overrightarrow{e_j:E_j}} \mathsf{a}_j(\overrightarrow{f_j}(\overrightarrow{d,e_j}))\cdot \mathsf{Z}(pop(st),\overrightarrow{get(st)}) \lhd st \neq \langle\,\rangle \wedge c_j(\overrightarrow{d,e_j}) \rhd \delta \\
&\quad +\sum_{j\in J}\sum_{\overrightarrow{e_j:E_j}} \mathsf{a}_j(\overrightarrow{f_j}(\overrightarrow{d,e_j})) \lhd st = \langle\,\rangle \wedge c_j(\overrightarrow{d,e_j}) \rhd \delta
\end{aligned}$$

where $\overrightarrow{get(st)} = get_1(st),\ldots,get_n(st)$.

The data type *Stack* is a standard stack data type with constructors $\langle\,\rangle$ representing the empty stack, and $push(\overrightarrow{t},st)$ inserting the new element $\overrightarrow{t}$ to the top of the stack st. We use the equality predicate on stacks, but a predicate that checks if a stack is empty can be used instead. The function $get_i(st)$ returns the ith element of the top of st, and the function $pop(st)$ returns the stack value st without its top element. See [19] for details on implementing data types in μCRL. To prove the following proposition we use an induction principle on the data type *Stack*, namely that every value of type stack is either empty or the result of an insertion to another value of this type.

During the current step we construct the system G''_1 consisting of the single equation for X and the set G'_2 being G^2_2 with all pCRL process terms $\mathsf{X}(\overrightarrow{t})$ replaced by $\mathsf{Z}(\langle\,\rangle,\overrightarrow{t})$.

Proposition 4.4. *Let systems G^6_1 and G''_1 as described above be given. Then for any data term vector $\overrightarrow{t}$ we have $(\mathsf{X}(\overrightarrow{t}),G^6_1) \Rightarrow_c (\mathsf{Z}(\langle\,\rangle,\overrightarrow{t}),G''_1)$. Let furthermore $G^6_1\cup G^2_2$ be a system of parallel pCRL process equations and $G''_1\cup G'_2$ be the result of the transformation described above. Then for any $\mathsf{X}\in|G^2_2|$, $(\mathsf{X},G^6_1\cup G^2_2)\Rightarrow_c(\mathsf{X},G''_1\cup G'_2)$.*

Proof. We define $g_{\mathsf{Z}}(st,\overrightarrow{d}) = \mathsf{X}(\overrightarrow{d}) \lhd st=\langle\,\rangle \rhd \mathsf{X}(\overrightarrow{d})\cdot g_{\mathsf{Z}}(pop(st),\overrightarrow{get(st)})$. To prove the implication we consider two cases. First, if the stack st is empty we have $g_{\mathsf{Z}}(st,\overrightarrow{t}) = \mathsf{X}(\overrightarrow{t})$. It can be shown by induction on n that

$$g_{\mathsf{Z}}(push(\overrightarrow{t^2},\ldots,push(\overrightarrow{t^n},\langle\,\rangle)\ldots),\overrightarrow{t^1}) = \mathsf{X}(\overrightarrow{t^1})\cdot\ldots\cdot\mathsf{X}(\overrightarrow{t^n})$$

When we apply this g_{Z} to the equation for Z and use the identities of the sort *Stack*, we get an identity which is the same as the equation for X.

In second case, if the stack $st = push(st',\overrightarrow{t'})$ for some stack value st' and data term vector $\overrightarrow{t'}$, we have $g_{\mathsf{Z}}(st,\overrightarrow{t}) = \mathsf{X}(\overrightarrow{t})\cdot g_{\mathsf{Z}}(st',\overrightarrow{t'})$. By induction on n it can be shown that

$$g_{\mathsf{Z}}(push(\overrightarrow{t^2},\ldots,push(\overrightarrow{t^n},st)\ldots),\overrightarrow{t^1}) = \mathsf{X}(\overrightarrow{t^1})\cdot\ldots\cdot\mathsf{X}(\overrightarrow{t^n})\cdot g_{\mathsf{Z}}(st',\overrightarrow{t'})$$

When we apply this g_{Z} to the equation for Z and use the identities of the sort *Stack*, we get the following identity:

$$\begin{aligned}
\mathsf{X}(\overrightarrow{d})\cdot g_{\mathsf{Z}}(st',\overrightarrow{t'}) = &\sum_{i\in I}\sum_{\overrightarrow{e_i:E_i}} \mathsf{a}_i(\overrightarrow{f_i}(\overrightarrow{d,e_i}))\cdot\mathsf{X}(\overrightarrow{t_i^1})\cdot\ldots\cdot\mathsf{X}(\overrightarrow{t_i^{n_i}})\cdot g_{\mathsf{Z}}(st',\overrightarrow{t'}) \lhd c_i(\overrightarrow{d,e_i}) \rhd \delta \\
&+\sum_{j\in J}\sum_{\overrightarrow{e_j:E_j}} \mathsf{a}_j(\overrightarrow{f_j}(\overrightarrow{d,e_j}))\cdot g_{\mathsf{Z}}(st',\overrightarrow{t'}) \lhd c_j(\overrightarrow{d,e_j}) \rhd \delta
\end{aligned}$$

This identity is derivable from the equation for X by applying axioms (A4), (SUM5) and (Cond6). $\square$

The following example [26] shows that the reverse implication does not hold in every model. It is easy to see that if data parameters do not matter, the stack is isomorphic to a counter which can be implemented by means of natural numbers.

Example 4.5. Let $G_1 = \{\mathsf{X} = \mathsf{a} \cdot \mathsf{X} \cdot \mathsf{X}\}$ and $G_2 = \{\mathsf{Z}(n{:}Nat) = \mathsf{a} \cdot \mathsf{Z}(succ(n))\}$. Consider the model with integers $\mathbb{Z}$ as the carrier set, and the operations $\cdot \to +, \mathsf{a} \to -1$. The equation in G_1 has the unique solution $\mathsf{X} = 1$, while the equation in G_2 has infinitely many solutions $\mathsf{Z}(n) = n + c$, where $c \in \mathbb{Z}$. For a more elaborated model that includes interpretations of other μCRL operations see Example 5.2.

Summary. This section is about the transformation of a finite system $G' = G'_1 \cup G_2$ with acyclic PNUDG and G'_1 in EGNF into a system $G'' = G''_1 \cup G'_2$ with G''_1 an LPE and G'_2 appropriately updated. For each $\mathsf{X} \in |G'|$,

$$\begin{array}{llll}
(\mathsf{X}, G') & = & (\mathsf{X}', G_1^5 \cup G_2^1) & (\text{“Harmonization”, by Proposition 4.2}) \\
 & =_c & (\mathsf{X}'', G_1^6 \cup G_2^2) & (\text{“One equation”, by Proposition 4.3}) \\
 & \Rightarrow_c & (\mathsf{X}''', G'') & (\text{“One LPE”, by Proposition 4.4}).
\end{array}$$

Here the primed versions of X represent the possible updates of parameters, as prescribed by the propositions mentioned.

5. From Parallel pCRL to LPE

As the result of the previous section we have obtained $G'' = G''_1 \cup G'_2$, where G''_1 is an LPE and G'_2 a (possibly empty) set of parallel pCRL process equations. In this section we show that the parallel part of G'' can be eliminated. First we take a general point of view, and show that LPEs are closed under the parallel pCRL process operations, viz. parallel composition, encapsulation, hiding, and renaming (see Definition 2.28). Then we show that with these results and those from Sections 3 and 4, the transformation of G'' into a single LPE can be carried out. We note that the transformation described in this section is uni-directional, and we give counterexamples for the associated reverse implications.

5.1 Parallel Composition of LPEs

Let G be a system of process equations in which each of $(\mathsf{X}(\overrightarrow{d_\mathsf{X}}), G)$ and $(\mathsf{Y}(\overrightarrow{d_\mathsf{Y}}), G)$ is defined by an LPE, and that contains an equation $\mathsf{Z}(\overrightarrow{d_\mathsf{X}, d_\mathsf{Y}}) = \mathsf{X}(\overrightarrow{d_\mathsf{X}}) \parallel \mathsf{Y}(\overrightarrow{d_\mathsf{Y}})$. Assume that the LPEs for X and Y have no common data variables, and are defined in the following way:

$$\begin{aligned}
\mathsf{X}(\overrightarrow{d_\mathsf{X}{:}D_\mathsf{X}}) = & \sum_{i \in I} \sum_{\overrightarrow{e_i{:}E_i}} \mathsf{a}_i(\overrightarrow{f_i}(\overrightarrow{d_\mathsf{X}, e_i})) \cdot \mathsf{X}(\overrightarrow{g_i}(\overrightarrow{d_\mathsf{X}, e_i})) \lhd c_i(\overrightarrow{d_\mathsf{X}, e_i}) \rhd \delta + \\
& \sum_{j \in J} \sum_{\overrightarrow{e_j{:}E_j}} \mathsf{a}_j(\overrightarrow{f_j}(\overrightarrow{d_\mathsf{X}, e_j})) \lhd c_j(\overrightarrow{d_\mathsf{X}, e_j}) \rhd \delta \\
\mathsf{Y}(\overrightarrow{d_\mathsf{Y}{:}D_\mathsf{Y}}) = & \sum_{i \in I'} \sum_{\overrightarrow{e'_i{:}E'_i}} \mathsf{a}'_i(\overrightarrow{f'_i}(\overrightarrow{d_\mathsf{Y}, e'_i})) \cdot \mathsf{Y}(\overrightarrow{g'_i}(\overrightarrow{d_\mathsf{Y}, e'_i})) \lhd c'_i(\overrightarrow{d_\mathsf{Y}, e'_i}) \rhd \delta + \\
& \sum_{j \in J'} \sum_{\overrightarrow{e'_j{:}E'_j}} \mathsf{a}'_j(\overrightarrow{f'_j}(\overrightarrow{d_\mathsf{Y}, e'_j})) \lhd c'_j(\overrightarrow{d_\mathsf{Y}, e'_j}) \rhd \delta
\end{aligned}$$

where $I \cap J = I' \cap J' = \emptyset$. We construct the equation for $\mathsf{Z}(\overrightarrow{d_\mathsf{X}{:}D_\mathsf{X}, d_\mathsf{Y}{:}D_\mathsf{Y}})$, being equal to $\mathsf{X}(\overrightarrow{d_\mathsf{X}}) \| \mathsf{Y}(\overrightarrow{d_\mathsf{Y}})$, as follows.

$$
\begin{aligned}
&\mathsf{Z}(\overrightarrow{d_\mathsf{X}{:}D_\mathsf{X}, d_\mathsf{Y}{:}D_\mathsf{Y}}) = \\
&\quad \sum_{i \in I} \sum_{\overrightarrow{e_i:E_i}} \mathsf{a}_i(\overrightarrow{f_i}(\overrightarrow{d_\mathsf{X}, e_i})) \cdot \mathsf{Z}(\overrightarrow{g_i}(\overrightarrow{d_\mathsf{X}, e_i}), \overrightarrow{d_\mathsf{Y}}) \lhd c_i(\overrightarrow{d_\mathsf{X}, e_i}) \rhd \delta \\
&\quad + \sum_{j \in J} \sum_{\overrightarrow{e_j:E_j}} \mathsf{a}_j(\overrightarrow{f_j}(\overrightarrow{d_\mathsf{X}, e_j})) \cdot \mathsf{Y}(\overrightarrow{d_\mathsf{Y}}) \lhd c_j(\overrightarrow{d_\mathsf{X}, e_j}) \rhd \delta \\
&\quad + \sum_{i \in I'} \sum_{\overrightarrow{e'_i:E'_i}} \mathsf{a}'_i(\overrightarrow{f'_i}(\overrightarrow{d_\mathsf{Y}, e'_i})) \cdot \mathsf{Z}(\overrightarrow{d_\mathsf{X}}, \overrightarrow{g'_i}(\overrightarrow{d_\mathsf{Y}, e'_i})) \lhd c'_i(\overrightarrow{d_\mathsf{Y}, e'_i}) \rhd \delta \\
&\quad + \sum_{j \in J'} \sum_{\overrightarrow{e'_j:E'_j}} \mathsf{a}'_j(\overrightarrow{f'_j}(\overrightarrow{d_\mathsf{Y}, e'_j})) \cdot \mathsf{X}(\overrightarrow{d_\mathsf{X}}) \lhd c'_j(\overrightarrow{d_\mathsf{Y}, e'_j}) \rhd \delta \\
&\quad + \sum_{(k,l) \in I \gamma I'} \sum_{\overrightarrow{e_k:E_k, e_l:E'_l}} \gamma(\mathsf{a}_k, \mathsf{a}'_l)(\overrightarrow{f_k}(\overrightarrow{d_\mathsf{X}, e_k})) \cdot \mathsf{Z}(\overrightarrow{g_k}(\overrightarrow{d_\mathsf{X}, e_k}), \overrightarrow{g'_l}(\overrightarrow{d_\mathsf{Y}, e'_l})) \\
&\qquad \lhd \overrightarrow{f_k}(\overrightarrow{d_\mathsf{X}, e_k}) = \overrightarrow{f'_l}(\overrightarrow{d_\mathsf{Y}, e'_l}) \wedge c_k(\overrightarrow{d_\mathsf{X}, e_i}) \wedge c'_l(\overrightarrow{d_\mathsf{Y}, e'_l}) \rhd \delta \\
&\quad + \sum_{(k,l) \in I \gamma J'} \sum_{\overrightarrow{e_k:E_k, e_l:E'_l}} \gamma(\mathsf{a}_k, \mathsf{a}'_l)(\overrightarrow{f_k}(\overrightarrow{d_\mathsf{X}, e_k})) \cdot \mathsf{Y}(\overrightarrow{g'_l}(\overrightarrow{d_\mathsf{Y}, e'_l})) \\
&\qquad \lhd \overrightarrow{f_k}(\overrightarrow{d_\mathsf{X}, e_k}) = \overrightarrow{f'_l}(\overrightarrow{d_\mathsf{Y}, e'_l}) \wedge c_k(\overrightarrow{d_\mathsf{X}, e_i}) \wedge c'_l(\overrightarrow{d_\mathsf{Y}, e'_l}) \rhd \delta \\
&\quad + \sum_{(k,l) \in J \gamma I'} \sum_{\overrightarrow{e_k:E_k, e_l:E'_l}} \gamma(\mathsf{a}_k, \mathsf{a}'_l)(\overrightarrow{f_k}(\overrightarrow{d_\mathsf{X}, e_k})) \cdot \mathsf{X}(\overrightarrow{g_k}(\overrightarrow{d_\mathsf{X}, e_k})) \\
&\qquad \lhd \overrightarrow{f_k}(\overrightarrow{d_\mathsf{X}, e_k}) = \overrightarrow{f'_l}(\overrightarrow{d_\mathsf{Y}, e'_l}) \wedge c_k(\overrightarrow{d_\mathsf{X}, e_i}) \wedge c'_l(\overrightarrow{d_\mathsf{Y}, e'_l}) \rhd \delta \\
&\quad + \sum_{(k,l) \in J \gamma J'} \sum_{\overrightarrow{e_k:E_k, e_l:E'_l}} \gamma(\mathsf{a}_k, \mathsf{a}'_l)(\overrightarrow{f_k}(\overrightarrow{d_\mathsf{X}, e_k})) \\
&\qquad \lhd \overrightarrow{f_k}(\overrightarrow{d_\mathsf{X}, e_k}) = \overrightarrow{f'_l}(\overrightarrow{d_\mathsf{Y}, e'_l}) \wedge c_k(\overrightarrow{d_\mathsf{X}, e_i}) \wedge c'_l(\overrightarrow{d_\mathsf{Y}, e'_l}) \rhd \delta
\end{aligned}
$$

where $P \gamma Q = \{(p, q) \in P \times Q \mid \gamma(\mathsf{a}_p, \mathsf{a}'_q) \text{ is defined}\}$.

Proposition 5.1. *Let G' contain the equations for X, Y and Z defined above. Let G contain the equations for X and Y, and the equation $\mathsf{Z}(\overrightarrow{d_\mathsf{X}, d_\mathsf{Y}}) = \mathsf{X}(\overrightarrow{d_\mathsf{X}}) \parallel \mathsf{Y}(\overrightarrow{d_\mathsf{Y}})$. Then $(\mathsf{Z}, G) \Rightarrow (\mathsf{Z}, G')$.*

Proof. We use the identity mapping for $g_\mathsf{X}, g_\mathsf{Y}, g_\mathsf{Z}$. Then the equations for X and Y are proven trivially because they are the same in G and G'. To prove the equation for Z first apply the axiom (CM1) to get $\mathsf{Z} = (\mathsf{X}(\overrightarrow{d_\mathsf{X}}) \mathbin{⫫} \mathsf{Y}(\overrightarrow{d_\mathsf{Y}}) + \mathsf{Y}(\overrightarrow{d_\mathsf{Y}}) \mathbin{⫫} \mathsf{X}(\overrightarrow{d_\mathsf{X}})) + \mathsf{X}(\overrightarrow{d_\mathsf{X}}) | \mathsf{Y}(\overrightarrow{d_\mathsf{Y}})$. Then we replace X and Y in the left hand sides of $⫫$ and in both sides of $|$ by their right hand sides. After that we apply the axioms (CM4), (SUM6), (Cond8), (CM2) and (CM3) to eliminate $⫫$, and the axioms (CM8), (CM9), (SUM7), (SUM7′), (Cond9), (Cond9′), (CM5), (CM6), (CM7), (CF1), (CF2),(CT1), (CT2), (CD1), (CD2) to eliminate $|$. Note that before applying the axioms for sums we might need to apply (SUM2), and after elimination $⫫$ and $|$ we might need to apply (A7) and (A6). After that we apply the identity $x \parallel y = y \parallel x$, which is derivable from axioms (CM1), (A1) and (SC3), to replace all occurrences of $\mathsf{Y}(\overrightarrow{t'}) \parallel \mathsf{X}(\overrightarrow{t})$ by $\mathsf{X}(\overrightarrow{t}) \parallel \mathsf{Y}(\overrightarrow{t'})$, and finally we replace all $\mathsf{X}(\overrightarrow{t}) \parallel \mathsf{Y}(\overrightarrow{t'})$ by $\mathsf{Z}(\overrightarrow{t}, \overrightarrow{t'})$ using the equation for Z in G. As the result we get the equation for Z in G'. □

In the following example we present a model of μCRL based on the trace model [12], but in which the sequential composition operation is commutative and idempotent. This model is used in Example 5.3 to show that the reverse implication of Proposition 5.1 does not hold in every model.

Example 5.2. Let *ActLab* be a finite set of action labels and γ be the totally undefined function. Consider the model with carrier set $\left(2^{\left(2^{ActLab}\setminus\emptyset\right)} \setminus \emptyset\right) \cup \{\top, \bot\}$, and the operations defined as follows:

- For each $\mathsf{a} \in ActLab$ $\mathsf{a}(\overrightarrow{t}) \to \{\{\mathsf{a}\}\}$
- $\delta \to \top$ and $\tau \to \bot$
- $+ \to \cup$, where $S \cup \top = \top \cup S = S$ and $S \cup \bot = \bot \cup S = \bot$
- $\cdot, \|, \lfloor\!\lfloor, | \to *$, where $S * S' = \{s \cup s' \mid s \in S \wedge s' \in S'\}$, $S * \top = \top * S = \top$ and $S * \bot = \bot * S = S$.
- $\partial_H \to e_H$, where $e_H(\{\{\mathsf{a}\}\}) = \{\{\mathsf{a}\}\}$ if $\mathsf{a} \notin H$, $e_H(\{\{\mathsf{a}\}\}) = \top$ if $\mathsf{a} \in H$, $e_H(S \cup S') = e_H(S) \cup e_H(S')$, $e_H(S * S') = e_H(S) * e_H(S')$, $e_H(\top) = \top$, $e_H(\bot) = \bot$
- $\tau_I \to h_I$, where h_I is defined in a similar way as e_H.
- $\sum_{d:D} \to id$, where id is the identity mapping.
- $x \lhd c \rhd y \to \mathit{if}(c, x, y)$, where $\mathit{if}(c, x, y)$ is the if-then-else mapping.

Example 5.3. Let $G = \{\mathsf{X} = \mathsf{a}\cdot\mathsf{X}, \mathsf{Y} = \mathsf{b}\cdot\mathsf{Y}, \mathsf{Z} = \mathsf{X}\|\mathsf{Y}\}$ and $G' = \{\mathsf{X} = \mathsf{a}\cdot\mathsf{X}, \mathsf{Y} = \mathsf{b}\cdot\mathsf{Y}, \mathsf{Z} = \mathsf{a}\cdot\mathsf{Z}+\mathsf{b}\cdot\mathsf{Z}\}$. In the model defined in Example 5.2 the equations for X in both G and G' have the following solutions:

$$\{\{\mathsf{a}\}\}, \quad \{\{\mathsf{a},\mathsf{b}\}\}, \quad \{\{\mathsf{a}\},\{\mathsf{a},\mathsf{b}\}\}, \quad \top$$

while the equations for Y have the following solutions:

$$\{\{\mathsf{b}\}\}, \quad \{\{\mathsf{a},\mathsf{b}\}\}, \quad \{\{\mathsf{b}\},\{\mathsf{a},\mathsf{b}\}\}, \quad \top$$

The equation for Z in G has two solutions $\{\{\mathsf{a},\mathsf{b}\}\}$ and $\top$, while the equation for Z in G' has five solutions $\{\{\mathsf{a},\mathsf{b}\}\}$, $\{\{\mathsf{a}\},\{\mathsf{a},\mathsf{b}\}\}$, $\{\{\mathsf{b}\},\{\mathsf{a},\mathsf{b}\}\}$, $\{\{\mathsf{a}\},\{\mathsf{b}\},\{\mathsf{a},\mathsf{b}\}\}$ and $\top$.

5.2 Encapsulation, Hiding and Renaming of LPEs

Let G be an LPE defining X as in the previous section, A be a set of action labels, and R be a renaming function. We construct LPEs for Z_1 being equal to $\partial_A(\mathsf{X})$, Z_2 being equal to $\tau_A(\mathsf{X})$, and Z_3 being equal to $\rho_R(\mathsf{X})$, in the following way:

$$\begin{aligned}
\mathsf{Z}_1(\overrightarrow{d_\mathsf{X}{:}D_\mathsf{X}}) = & \sum_{i\in I_1}\sum_{\overrightarrow{e_i:E_i}} \mathsf{a}_i(\overrightarrow{f_i}(\overrightarrow{d_\mathsf{X}, e_i}))\cdot \mathsf{Z}_1(\overrightarrow{g_i}(\overrightarrow{d_\mathsf{X}, e_i})) \lhd c_i(\overrightarrow{d_\mathsf{X}, e_i}) \rhd \delta \\
& + \sum_{j\in J_1}\sum_{\overrightarrow{e_j:E_j}} \mathsf{a}_j(\overrightarrow{f_j}(\overrightarrow{d_\mathsf{X}, e_j})) \lhd c_j(\overrightarrow{d_\mathsf{X}, e_j}) \rhd \delta
\end{aligned}$$

Here and in the equations below we assume that $I_1 = \{i \in I \mid \mathsf{a}_i \notin A\}$ and $J_1 = \{j \in J \mid \mathsf{a}_j \notin A\}$.

$$\begin{aligned}
\mathsf{Z}_2(\overrightarrow{d_\mathsf{X}{:}D_\mathsf{X}}) = & \sum_{i\in I_1}\sum_{\overrightarrow{e_i:E_i}} \mathsf{a}_i(\overrightarrow{f_i}(\overrightarrow{d_\mathsf{X}, e_i}))\cdot \mathsf{Z}_2(\overrightarrow{g_i}(\overrightarrow{d_\mathsf{X}, e_i})) \lhd c_i(\overrightarrow{d_\mathsf{X}, e_i}) \rhd \delta \\
& + \sum_{j\in J_1}\sum_{\overrightarrow{e_j:E_j}} \mathsf{a}_j(\overrightarrow{f_j}(\overrightarrow{d_\mathsf{X}, e_j})) \lhd c_j(\overrightarrow{d_\mathsf{X}, e_j}) \rhd \delta \\
& + \sum_{i\in I\setminus I_1}\sum_{\overrightarrow{e_i:E_i}} \tau\cdot \mathsf{Z}_2(\overrightarrow{g_i}(\overrightarrow{d_\mathsf{X}, e_i})) \lhd c_i(\overrightarrow{d_\mathsf{X}, e_i}) \rhd \delta \\
& + \sum_{j\in J\setminus J_1}\sum_{\overrightarrow{e_j:E_j}} \tau \lhd c_j(\overrightarrow{d_\mathsf{X}, e_j}) \rhd \delta
\end{aligned}$$

$$\begin{aligned}\mathsf{Z}_3(\overrightarrow{d_{\mathsf{X}}{:}D_{\mathsf{X}}}) = &\sum_{i\in I}\sum_{\overrightarrow{e_i:E_i}} R(\mathsf{a}_i)(\overrightarrow{f_i}(\overrightarrow{d_{\mathsf{X}}, e_i}))\cdot \mathsf{Z}_3(\overrightarrow{g_i}(\overrightarrow{d_{\mathsf{X}}, e_i})) \lhd c_i(\overrightarrow{d_{\mathsf{X}}, e_i}) \rhd \delta\\ &+\sum_{j\in J}\sum_{\overrightarrow{e_j:E_j}} R(\mathsf{a}_j)(\overrightarrow{f_j}(\overrightarrow{d_{\mathsf{X}}, e_j})) \lhd c_j(\overrightarrow{d_{\mathsf{X}}, e_j}) \rhd \delta\end{aligned}$$

Proposition 5.4. *Let G_1' contain the equations for X and Z_1 defined above, G_2' contain the equations for X and Z_2 defined above, and G_3' contain the equations for X and Z_3 defined above. Let G_1 contain the equations for X and $\mathsf{Z}_1(\overrightarrow{d_{\mathsf{X}}{:}D_{\mathsf{X}}}) = \partial_A(\mathsf{X}(\overrightarrow{d_{\mathsf{X}}}))$, G_2 contain the equations for X and $\mathsf{Z}_2(\overrightarrow{d_{\mathsf{X}}{:}D_{\mathsf{X}}}) = \tau_A(\mathsf{X}(\overrightarrow{d_{\mathsf{X}}}))$, and G_3 contain the equations for X and $\mathsf{Z}_3(\overrightarrow{d_{\mathsf{X}}{:}D_{\mathsf{X}}}) = \rho_R(\mathsf{X}(\overrightarrow{d_{\mathsf{X}}}))$. Then we have $G_1 \Rightarrow G_1'$, $G_2 \Rightarrow G_2'$ and $G_3 \Rightarrow G_3'$.*

Proof. To prove the implications we use the identity mappings for $g_{\mathsf{X}}, g_{\mathsf{Z}_1}, g_{\mathsf{Z}_2}$ and g_{Z_3}. The equations for X are proven trivially. For the other equations we substitute X by its right hand side and apply the axioms (D3), (SUM8), (D5), (D4), (D1), (D2), (A7), (A6) to push ∂_A inside; the axioms (T3), (SUM9), (T5), (T4), (T1), (T2) to push τ_A inside; the axioms (R3), (SUM10), (R5), (R4), (R1), (RT), (RD) to push ρ_R inside. After that we use the equations for $\mathsf{Z}_1, \mathsf{Z}_2, \mathsf{Z}_3$ in G_1, G_2, G_3 respectively to to eliminate the operators ∂_A, τ_A and ρ_R completely and arrive at equations for $\mathsf{Z}_1, \mathsf{Z}_2, \mathsf{Z}_3$ in G_1', G_2', G_3' respectively. □

The following examples show that the reverse implications of the latter proposition do not hold in every model.

Example 5.5. Let $G_1 = \{\mathsf{X} = \mathsf{a}\cdot\mathsf{X} + \mathsf{b}\cdot\mathsf{X}, \mathsf{Z}_1 = \partial_{\{\mathsf{b}\}}(\mathsf{X})\}$ and $G_1' = \{\mathsf{X} = \mathsf{a}\cdot\mathsf{X} + \mathsf{b}\cdot\mathsf{X}, \mathsf{Z}_1 = \mathsf{a}\cdot\mathsf{Z}_1\}$. Consider the model from Example 5.2. The equations for X in both G_1 and G_1' have the following solutions:

$$\{\{\mathsf{a},\mathsf{b}\}\},\quad \{\{\mathsf{a}\},\{\mathsf{a},\mathsf{b}\}\},\quad \{\{\mathsf{b}\},\{\mathsf{a},\mathsf{b}\}\},\quad \{\{\mathsf{a}\},\{\mathsf{b}\},\{\mathsf{a},\mathsf{b}\}\},\quad \top$$

The equation for Z_1 in G_1 has two solutions $\{\{\mathsf{a}\}\}$ and $\top$, while the equation for Z_1 in G_1' has four solutions $\{\{\mathsf{a}\}\}$, $\{\{\mathsf{a},\mathsf{b}\}\}$, $\{\{\mathsf{a}\},\{\mathsf{a},\mathsf{b}\}\}$ and $\top$.

Example 5.6. Let $G_2 = \{\mathsf{X} = \mathsf{a}\cdot\mathsf{X}, \mathsf{Z}_2 = \tau_{\{\mathsf{a}\}}(\mathsf{X})\}$ and $G_2' = \{\mathsf{X} = \mathsf{a}\cdot\mathsf{X}, \mathsf{Z}_2 = \tau\cdot\mathsf{Z}_2\}$. Consider the branching bisimulation model [12]. The equation for Z_2 in G_2 has the unique solution $\mathsf{Z}_2 = \tau$, while the equation for Z_2 in G_2' has infinitely many solutions $\mathsf{Z}_2 = \tau\cdot p$, where p is any element of the model.

Example 5.7. Let $G_3 = \{\mathsf{X} = \mathsf{a}\cdot\mathsf{X}+\mathsf{b}\cdot\mathsf{X}, \mathsf{Z}_3 = \rho_R(\mathsf{X})\}$ and $G_3' = \{\mathsf{X} = \mathsf{a}\cdot\mathsf{X}+\mathsf{b}\cdot\mathsf{X}, \mathsf{Z}_3 = \mathsf{a}\cdot\mathsf{Z}_3\}$, where $R(\mathsf{a}) = R(\mathsf{b}) = \mathsf{a}$. Consider the model from Example 5.2. The equation for Z_3 in G_3 has two solutions $\{\{\mathsf{a}\}\}$ and $\top$, while the equation for Z_3 in G_3' has four solutions $\{\{\mathsf{a}\}\}$, $\{\{\mathsf{a},\mathsf{b}\}\}$, $\{\{\mathsf{a}\},\{\mathsf{a},\mathsf{b}\}\}$ and $\top$.

5.3 Towards an LPE

Let $G'' = G_1'' \cup G_2'$ be a system of process equations with G_1'' an LPE and G_2' containing parallel pCRL process equations. If G_2' is empty we are done. Otherwise, let (X, G'') be the process definition to be transformed. We substitute the right hand sides for all parallel pCRL process names (other than X) in G_2' and obtain the set G_2'' with a single process equation for X, such that $(\mathsf{X}, G'') = (\mathsf{X}, G_1'' \cup G_2'')$. We finish the description of our transformation of G'' into a single LPE by describing how G_2'' can be integrated with G_1''. A general strategy is to apply an innermost/outermost reduction along the lines of Propositions 5.1 and 5.4, occasionally adding or replacing process equations.

We consider a typical case (but note that many variants are conceivable):

$$\begin{aligned} G_1'' &= \{\mathsf{Y}(\overrightarrow{d_{\mathsf{Y}}{:}D_{\mathsf{Y}}}) = p_{\mathsf{Y}}\}\\ G_2' &= \{\mathsf{X}(\overrightarrow{d_{\mathsf{X}}{:}D_{\mathsf{X}}}) = \tau_I(\partial_H(\mathsf{Y}(\overrightarrow{t}) \parallel \mathsf{Y}(\overrightarrow{u})))\}\end{aligned}$$

and proceed in a stepwise manner. First we reduce the $\|$-occurrence, so transform G_2' into

$$G_2^3 = \{\mathsf{X}(\overrightarrow{d_\mathsf{X}{:}D_\mathsf{X}}) = \tau_I(\partial_H(\mathsf{Z}(\overrightarrow{t}, \overrightarrow{u}))),\ \mathsf{Z}(\overrightarrow{d_\mathsf{Y}{:}D_\mathsf{Y}}, \overrightarrow{e_\mathsf{Y}{:}D_\mathsf{Y}}) = \mathsf{Y}(\overrightarrow{d_\mathsf{Y}}) \parallel \mathsf{Y}(\overrightarrow{e_\mathsf{Y}})\}$$

where $\overrightarrow{e_\mathsf{Y}}$ is a fresh copy of $\overrightarrow{d_\mathsf{Y}}$. With Lemma 2.18 it follows that for all $\mathsf{Y} \in |G''|$, $(\mathsf{Y}, G'') = (\mathsf{Y}, G_1'' \cup G_2^3)$. According to Proposition 5.1, there exists a system H with Z defined by a number of *linear* equations in the process names Z and Y such that $(\mathsf{Z}, G_1'' \cup G_2^3) \Rightarrow_c (\mathsf{Z}, H)$, and for the remaining process names $\mathsf{Y} \in |G''|$, $(\mathsf{Y}, G'') = (\mathsf{Y}, G_1'' \cup G_2^3) \Rightarrow_c (\mathsf{Y}, H)$. Comparing the newly created system H of process equations with G'', we see that it contains one parallel pCRL operation less, and one more pCRL process equation consisting of the linear equation for Z. Next, with Propositions 4.2 and 4.3 this system can be transformed into a system H' that contains a single LPE, say over process name U, and the equation $\mathsf{X}(\overrightarrow{d_\mathsf{X}{:}D_\mathsf{X}}) = \tau_I(\partial_H(\mathsf{U}(\overrightarrow{u})))$ where application of these propositions prescribes the value vector $\overrightarrow{u}$. With Proposition 5.4 we can resolve the encapsulation and hiding operation in a similar fashion. This yields a system of process equations H'' that consists of an LPE over process name V and the equation $\mathsf{X}(\overrightarrow{d_\mathsf{X}{:}D_\mathsf{X}}) = \mathsf{V}(\overrightarrow{v})$, and $(\mathsf{X}, H') \Rightarrow (\mathsf{X}, H'')$. Now the last step of this final transformation is the conclusion $(\mathsf{X}, H'') = (\mathsf{V}(\overrightarrow{v}), G_{lin})$, where G_{lin} contains only the LPE for V.

The description above illustrates the last part of our transformation. Without further proof we state the following result.

Proposition 5.8. *Let $G'' = G_1'' \cup G_2'$ be a system of process equations as described above (G_1'' an LPE, and G_2' containing parallel pCRL process equations). Then G'' can be transformed via innermost/outermost reduction into a system G_{lin} that contains one single LPE, and that satisfies $(\mathsf{X}, G'') \Rightarrow_c (\mathsf{X}'(\overrightarrow{t_{\mathsf{X}'}}), G_{lin})$ for a certain value vector $\overrightarrow{t_{\mathsf{X}'}}$.*

6. Conclusions

We described a transformation of parallel pCRL process definitions into a linear format, and argued that this transformation is correct. Our correctness argument is not tied to some particular model, and also applies to process definitions that do not necessarily imply that the models have unique solutions. Furthermore, this transformation is idempotent in the following sense: applying the transformation to an LPE yields the same LPE.

The algorithm underlying the transformation into LPE format basically matches the one that is currently implemented in the μCRL toolset [14]. Of course, during the process of linearization many optimizations are conceivable, some of which can only be applied in a certain context. We have already mentioned some optimization rewrite rules (Table 10) that can be applied during one of the linearization steps. Another optimization can be performed in the cases where a new process name is introduced. There can be a choice of what parameters to use for the new process name in order to fetch the complicated structure of data terms involved. Furthermore, there are many (minor) optimizations, such as the rewriting of conditions or the elimination of constant parameters. Due to the fact that the LPE format provides such a simple process structure, we feel that this type of optimizations can be best performed after the transformation into the LPE format. Such optimizations include rewriting of data terms, eliminations of redundant variables and constants, abstract interpretation, and so on.

There are two particular optimizations that we want to mention here in more detail: *regular linearization* and *clustering of actions*. The first of these is based on [22], and applies to the situation where regularity follows from the absence of termination in a recursion, like in $\mathsf{X} = \mathsf{a} \cdot \mathsf{X} \cdot \mathsf{X}$. Restricting to standard process semantics for μCRL, an LPE that specifies the same behavior is $\mathsf{X} = \mathsf{a} \cdot \mathsf{X}$. However, this optimization is model dependent, as there can be models in which the two equations have different sets of solutions. For some other cases, also dealt with in [22] and used in the μCRL toolset, these optimizations can be justified on a general level using the equivalence of systems of process equations. For example, the system $G_1 = \{\mathsf{X} = \mathsf{a} \cdot \mathsf{Y} \cdot \mathsf{X}, \mathsf{Y} = \mathsf{b}\}$ can be transformed into $G_2 = \{\mathsf{X} = \mathsf{a} \cdot \mathsf{Z}, \mathsf{Z} = \mathsf{b} \cdot \mathsf{X}\}$, and we can prove that $(X, G_1) = (X, G_2)$, thus showing that this transformation is sound in every model. As for 'clustering of actions', we refer to Definition 2.7, Theorem 2.8 and Theorem A.4 in [18]. The transformation allows to optimize an LPE to a form in which every action label occurs at most twice (either as a termination action or not). The constructed LPE is equivalent to the original one. During the transformation the sums $\sum_{i \in I}$ and $\sum_{j \in J}$ which in Definition 4.1 represent the abbreviations for alternative compositions, are changed to the 'real' sums over enumerated data types. We note that both these latter optimizations are implemented in the current version of the μCRL toolset.

In the future we plan to work on extending the linearization procedure to cover the full syntax of μCRL. Furthermore, the procedure can be extended to handle the timed version of the language. Finally, additional extensions to the language like interrupts, process creation and priorities could be investigated, as there is a practical demand for these facilities.

References

[1] J.C.M. Baeten, J.A. Bergstra, and J.W. Klop. Decidability of bisimulation equivalence for processes generating context-free languages. *Journal of the ACM*, 40(3):653–682, July 1993.

[2] J.C.M. Baeten and W.P. Weijland. *Process Algebra*. Cambridge Tracts in Theoretical Computer Science 18. Cambridge University Press, 1990.

[3] J.A. Bergstra and J.W. Klop. Process algebra for synchronous communication. *Information and Computation*, 60(1/3):109–137, 1984.

[4] J.A. Bergstra and J.W. Klop. A complete inference system for regular processes with silent moves. In F.R. Drake and J.K. Truss, editors, *Proceedings Logic Colloquium 1986*, pages 21–81, Hull, 1988. North-Holland. First appeared as: Report CS-R8420, CWI, Amsterdam, 1984.

[5] J.A. Bergstra and A. Ponse. Translation of a muCRL-fragment to I-CRL. In *Methods for the Transformation and analysis of CRL. Deliverable 46/SPE/WP5/DS/A/007/b1*, SPECS RACE Project no. 1046, pages 125–148. Available through GSI Tecsi, May 1991.

[6] M.A. Bezem and J.F. Groote. Invariants in process algebra with data. In B. Jonsson and J. Parrow, editors, *CONCUR'94*, pages 401–416. LNCS 836, Springer-Verlag, 1994.

[7] D.J.B. Bosscher and A. Ponse. Translating a process algebra with symbolic data values to linear format. In U.H. Engberg, K.G. Larsen, and A. Skou, editors, *Proceedings of the Workshop on Tools and Algorithms for the Construction and Analysis of Systems,* Aarhus, Denmark, volume NS-95-2 of *BRICS Notes Series*, pages 119–130. Department of Computer Science, University of Aarhus, May 1995.

[8] J.J. Brunekreef. Process specification in a UNITY format. In A. Ponse, C. Verhoef, and S.F.M. van Vlijmen, editors, *Algebra of Communicating Processes, Utrecht 1994*, Workshops in Computing, pages 319–337. Springer-Verlag, 1995.

[9] S. Burris and H.P. Sankappanavar. *A Course in Universal Algebra.* Number 78 in Graduate Texts in Mathematics. Springer-Verlag, 1981.

[10] K.M. Chandy and J. Misra. *Parallel Program Design. A Foundation.* Addison-Wesley, 1988.

[11] W.J. Fokkink. *Introduction to Process Algebra.* Texts in Theoretical Computer Science. An EATCS Series. Springer-Verlag, 2000.

[12] R.J. van Glabbeek. The linear time – branching time spectrum II; the semantics of sequential systems with silent moves. Manuscript. Preliminary version available by ftp at `ftp://boole.stanford.edu/pub/spectrum.ps.gz`, 1993. Extended abstract in E. Best, editor: Proceedings *CONCUR'93*, 4^{th} International Conference on Concurrency Theory, Hildesheim, Germany, August 1993, LNCS 715, Springer, pp. 66–81.

[13] J.F. Groote. The syntax and semantics of timed μCRL. Report SEN-R9709, CWI, The Netherlands, 1997.

[14] J.F. Groote and B. Lisser. *Tutorial and Reference Guide for the μCRL toolset version 1.0*. CWI, 1999. Available from http://www.cwi.nl/∼mcrl/mutool.html.

[15] J.F. Groote and S.P. Luttik. Undecidability and completeness results for process algebras with alternative quantification over data. Report SEN-R9806, CWI, The Netherlands, July 1998. Available from `http://www.cwi.nl/∼luttik/`; submitted for publication.

[16] J.F. Groote and A. Ponse. Proof theory for μCRL: A language for processes with data. In D.J. Andrews, J.F. Groote, and C.A. Middelburg, editors, *Semantics of Specification Languages*, pages 232–251. Workshop in Computing Series, Springer-Verlag, 1994.

[17] J.F. Groote and A. Ponse. The syntax and semantics of μCRL. In A. Ponse, C. Verhoef, and S.F.M. van Vlijmen, editors, *Algebra of Communicating Processes 1994*, pages 26–62. Workshop in Computing Series, Springer-Verlag, 1995.

[18] J.F. Groote and J. Springintveld. Focus points and convergent process operators. A proof strategy for protocol verification. Technical Report 142, Department of Philosophy, Utrecht University, 1995. ftp://ftp.phil.uu.nl/pub/logic/PREPRINTS/preprint142.ps.Z.

[19] J.F. Groote and J.J. van Wamel. Algebraic data types and induction in μCRL. Technical Report P9409, University of Amsterdam, Programming Research Group, 1994.

[20] Y. Hirshfeld and F. Moller. Decidability results in automata and process theory. In F. Moller and G. Birtwistle, editors, *Logics for Concurrency: Structure versus Automata*, LNCS 1043, pages 102–148, Berlin, 1996. Springer-Verlag.

[21] ISO/IEC. *LOTOS — a formal description technique based on the temporal ordering of observational behaviour*. International Standard 8807. International Organization for Standardization, — Information Processing Systems — Open Systems Interconection, Genève, September 1988.

[22] S. Mauw and J.C. Mulder. Regularity of BPA-systems is decidable. In B. Jonsson and J. Parrow, editors, *Proc. CONCUR '94*, LNCS 836, pages 34–47. Springer Verlag, 1994.

[23] S. Mauw and G.J. Veltink. A process specification formalism. *Fundamenta Informaticae*, XIII:85–139, 1990.

[24] S. Mauw and G.J. Veltink, editors. *Algebraic Specification of Communication Protocols*. Cambridge Tracts in Theoretical Computer Science 36. Cambridge University Press, 1993.

[25] R. Milner. A complete inference system for a class of regular behaviours. *Journal of Computer and System Sciences*, 28:439–466, 1984.

[26] V. van Oostrom. Personal communications, 2000.

[27] A. Ponse. Computable processes and bisimulation equivalence. *Formal Aspects of Computing*, 8(6):648–678, 1996.

[28] SPECS-Semantics and Analysis. *Definition of MR and CRL Version 2.1. Deliverable 46/SPE/WP5/DS/A/017/b1.* SPECS RACE Project no. 1046. Available through GSI Tecsi, 1990.

[29] Specification and description language (SDL). ITU-T Recommendation Z.100, 1994.