Translating a Process Algebra with Symbolic Data Values to Linear Format

Doeko Bosscher
CWI
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands

Alban Ponse
Programming Research Group, University of Amsterdam
Kruislaan 403, 1098 SJ Amsterdam

The first author was supported by Esprit BRA 7166 concur 2.

Abstract

Historically, process algebras have been studied mostly *without* data. In this paper the transformation is described of the valued process algebra μCRL [GP95] to a symbolic transition system in the spirit of [Sch94]. The data oriented specifications thus obtained, seem to be in a better format for checking modal properties.

1. Introduction

Historically, much effort has been put into understanding the theories of pure process algebra calculi. Also, process algebra tools concentrate on process calculi with explicit data values as an input language. A few front ends have been developed which translate a calculus with symbolic values into a pure calculus, such as the value passer [Bru91], which translates value-passing CCS [Mil89] into pure CCS. In the LOTOS community several tools have been constructed which can translate (valued) LOTOS to C programs or labeled transitions system, e.g. [FGM$^+$92, KBG93].

We aim at building a tool which can do model checking for μCRL [GP95]. μCRL is a specification language for a process algebra with symbolic data values, where the process part is based on ACP [BW90] and the data part on algebraic specifications as in [EM85]. Until now μCRL has been mainly used for manual verification proving equivalences between processes (e.g. [BG94a]), but we are thinking of checking properties in a suitable logic. Sometimes we know only part of the desired behavior, as in the case of safety criteria for railroads [GKvV94].

In this respect [Sch94] is highly interesting, which treats a calculus with symbolic data values as a first class citizen. In [Sch94] value-passing CCS is mapped onto a data structure called *parametrized graphs*, which are essentially symbolic transition systems. This has two advantages over the conventional procedure of translating the valued calculus to the pure calculus and then perform modal checking. First, the structure of the processes is still visible in the parametrized graphs. Second, part of the state explosion is avoided because data is not expanded.

Whereas [Sch94] is mainly focused on checking various equivalences, we are interested in checking modal formulas as in [GvV94] or [HL93]. We think that translating the language μCRL to parametrized graphs is an interesting experiment in itself and a signal for model checking. We refer to the experience that Hennessy-Milner Logic seems to be checked more efficiently on a restricted form of pure CCS [Hol89]. A second point of interest is that the parameterized graphs are a kind of *data oriented* specifications. In several case studies verification starts by transforming specifications to such a form by hand and then performing further analysis, see e.g. [Bru95, GS95].

We describe the transformation of μCRL to parametrized graphs, which we will define syntactically.

We start with a translation of a fragment of μCRL to single-linear format by means of typical examples [1]. This format is a direct translation of a graph grammar and can be seen also as a fragment of value-passing CCS. Next we explain how a larger part of the μCRL specifications in the full calculus can be translated to this format. We end with some conclusions on the implementation of the transformation in the ASF+SDF system [Kli93].

2. Translating a Fragment of μCRL to a Single-Linear Specification

The specification language μCRL has come out of the SPECS project, as the essence of the language CRL [BDE+93]. It has been developed under the assumption that a study of the basic concepts of specification languages will yield more fundamental insights then studying the complete language.

The data part contains equational specifications. The process part contains processes described in the style of CCS, CSP or ACP, where the syntax has been taken from the last. It basically consists of a set of uninterpreted actions that may be parametrized by data. These actions represent various activities, depending on the usage of the language. There are sequential composition, alternative and parallel composition operators. Furthermore, recursive processes are specified in a simple way. See for a complete definition of jargon, syntax and semantics [PVvV95].

In this section we describe the translation of a basic fragment of μCRL to linear format. It is similar to BPA, in that it contains only alternative and sequential composition [2] It extends BPA by the presence of data and the if-then-else and sum construct. First we define this fragment and linear specifications in a precise way. Next we describe the translation by means of examples.

2.1 Specifications in BPS Format

A well-formed μCRL *specification* E is a *specification* in Basic Process Syntax, BPS for short, iff all *process-declarations* occurring in E have in their right-hand sides *process-expressions* that are in BPS:

Definition 2.1 The syntactical category BPS that constitutes the class of processes in BPS has the following BNF syntax.

$$
\begin{array}{lcl}
\textit{process-expression} & ::= & \textit{process-expression} + \textit{process-expression} \\
& \mid & \textit{process-expression} \lhd \textit{data-term} \rhd \textit{process-expression} \\
& \mid & \textit{process-expression} \cdot \textit{process-expression} \\
& \mid & \sum(\textit{single-variable-declaration}, \textit{process-expression}) \\
& \mid & \delta \\
& \mid & \textit{name} \\
& \mid & \textit{name}(\textit{data-term-list}) \\
& \mid & (\textit{process-expression}).
\end{array}
$$

In the above defintion $+$ is the choice operator, $\cdot$ sequential composition and $\lhd\ \rhd$ is the notation for the if-then-else construct in μCRL . $\sum$ is the notation for a summation over data. δ is the deadlocked process. The precedence is in the order $\cdot, \lhd\ \rhd, +$ (as can be seen seen from definition above).

Example 2.2 Consider the following well-formed μCRL *specification* in BPS, E of the sender in the Alternating Bit Protocol of [GP95]:

[1] Formal approaches to μCRL proof theory are e.g., [GP93],[BG94b].

[2] In linear formats, sequential composition can actually be replaced by action prefixing.

$$
\begin{array}{lll}
E \equiv & \textbf{sort} & bit, D, error, \textbf{Bool} \\
& \textbf{func} & T, F :\rightarrow \textbf{Bool} \\
& & 0, 1 :\rightarrow bit \\
& & e :\rightarrow error \\
& & invert : bit \rightarrow bit \\
& & d_1, d_2, d_3 :\rightarrow D \\
& \textbf{act} & r1 : D \\
& & r6 : error \\
& & r6 : bit \\
& & s2 : D \times bit \\
& & c : \textbf{Bool} \\
& \textbf{rew} & invert(0) = 1 \\
& & invert(1) = 0 \\
& \textbf{proc} & S = S(0) \cdot S(1) \cdot S \\
& & S(n : bit) = sum(d : D, r1(d) \cdot S(d, n)) \\
& & S(d : D, n : bit) = s2(d, n) \cdot ((r6(invert(n)) + r6(e)) \cdot S(d, n) + r6(n))
\end{array}
$$

We will use the terms *process* and *action* as follows: let E be a μCRL *specification* and q a *process-expression* that is Statically and Semantically Correct (SSC, [GP95]) with respect to E and has no free data variables, then p **from** E is called a process from E. We will use the term *parametrized process name* for the name in the left-hand side of a process specification, which has a type given by the parametrization [3]. Furthermore an *action* is a process that refers directly to an *action-specification* in E and has no free data variables. So in the example above $S(0)$ is a process from E, and $r6(invert(0)), r6(e)$ are actions from E. If E is fixed, we just speak of "the process p".

We will restrict our attention to a decidable class of guarded specifications in BPS. We will admit only those specifications where the defining right hand side of every process name is such that the process name occurs only guarded, i.e. either directly or indirectly in the scope of an action.

Definition 2.3 Let P be the set of process names occurring in the specification E and $p, p_1, ..., p_n, q \in P$ (parametrized) process names. Let $UG(p, E)$ be a set of tuples of the form $< p, q >$ where q is a (parametrized) process name occurring unguarded in the declaration of p, i.e. not in the scope of a preceding action. E is *syntactically guarded* iff $\bigcup_{p \in P} UG(p, E)$ contains no cycle, i.e. a subset of the form $\{< p_1, p_2 >, < p_2, p_3 > ... < p_{n-1}, p_n >\}$ so that $p_1 \equiv p_n$.

Given a μCRL *specification* E, we associate with each process from E a (referential) transition system that describes its meaning. The intended semantics of a process p from a μCRL *specification* E is a transition system $\mathcal{A}(\boldsymbol{A}_{N_E}, p \textbf{ from } E)$ where $\boldsymbol{A}_{N_E}$ is the canonical term algebra of E, and where the labels of transitions may be parameterized by the fixed representations of the elements of $\boldsymbol{A}_{N_E}$. These transition systems are considered modulo bisimulation equivalence, notation $\leftrightarrows_{\boldsymbol{A}_{N_E}}$, as this is the coarsest congruence that respects operational behaviour.

Now processes from syntactically guarded μCRL *specifications* in BPS constitute the **source language** for the translation described in the sequel.

Conventions. For readability we adopt the following conventions.

- Binary operations associate to the right, brackets are omitted if possible.

[3] So in Example 2.2 the three process declarations have a *different* parametrized process name, although their name is the same.

- Instead of repeatedly denoting μCRL *specifications* in a syntactically correct way (as was done in the example above), we often only write down a *process-specification* without the keyword **proc**, and assume that it is part of some well-defined μCRL *specification*. In doing so we use $a, b, c, ...$ as syntactic variables for action *names* and $X, Y, Z, ...$ as syntactic variables for process *names*.
- Whenever convenient, we assume that any μCRL *specification* under consideration contains the (standard) functions $\neg$ and $\wedge$ on the standard sort **Bool**. Applications of the function $\wedge$ will always be written in an infix manner. Note that from the point of view of describing *processes* this convention causes no loss of generality, as we can always extend *specifications* with these functions. □

2.2 Single-Linear Process Specifications

In this section we define the syntax of “single-linear” *process-specifications* that play a crucial role in our canonical translation.

We start by introducing the following two archetypes of μCRL *process-specifications* in BPS. In their definition we use the symbol Σ also as a shorthand to denote **finite** sums (not to be confused with the *sum operator* of μCRL): let $p_1, p_2, ...$ be *process-expressions*, then the expression

$$\sum_{i=1}^{k} p_i$$

abbreviates δ in case $k = 0$, and $p_1 + p_2 + ... + p_k$ otherwise.

Definition 2.4 A *process-specification* of the form $pd_1 \; ... \; pd_m$ with $m \geq 1$ from some μCRL *specification* E is in *normal form* iff for all $1 \leq i \leq m$ the declaration pd_i has a right-hand side of the form

$$\sum_{j=1}^{k_i} p_{ij}$$

where each of the *process-expressions* p_{ij} [4] is of the form

$$\begin{aligned}&(\textstyle\sum_{k=1}^{k_{ij}} \Sigma(d_{ijk} : D_{ijk}, a_{ijk} \cdot X^1_{ijk} \cdot X^2_{ijk}) + \\ &\textstyle\sum_{l=1}^{l_{ij}} \Sigma(d_{ijl} : D_{ijk}, b_{ijl} \cdot X^3_{ijl}) + \\ &\textstyle\sum_{m=1}^{m_{ij}} \Sigma(d_{ijm} : D_{ijm}, c_{ijm})) \lhd t_{ij} \rhd \delta\end{aligned}$$

with the d_{ijk} single variables over data types D_{ijk}, $a_{ijk}, b_{ijk}, c_{ijk}$ (possibly parameterized) *process-expressions* over the *names* in the *action-specifications* from E, and the $X^1_{ijk}, X^2_{ijk}, X^3_{ijk}$ (possibly parameterized) *process- expressions* over the *names* in the left-hand sides of the declarations $pd_1, ..., pd_m$.
In the special case that $k_{ij} = 0$ for all appropriate i, j we say that the *process-specification* $pd_1 \; ... \; pd_m$ is in *linear form*.

Now we can define what is meant by a “single-linear” *process-specification*.

Definition 2.5 Let E be a μCRL *specification*. A *process-specification* occurring in E is *single-linear* iff it is in linear form and contains exactly one *process-declaration*.

[4] We use of course the axiom $\sum(d : D, p) = p$, d *not* free in p, to remove summations.

Example 2.6 Consider the following specification:

$$
\begin{array}{lll}
E \equiv & \textbf{sort} & \textbf{Bool}, S \\
& \textbf{func} & T, F :\to \textbf{Bool} \\
& & C :\to S \\
& & f : \textbf{Bool} \to S \\
& & g : S \to \textbf{Bool} \\
& \textbf{act} & a, d \\
& & b : \textbf{Bool} \\
& & c : S \times \textbf{Bool} \\
& \textbf{proc} & X(x : \textbf{Bool}, y : S) = \quad (a \cdot X(x, f(x)) + b(x)) \triangleleft x \triangleright \delta \\
& & \qquad\qquad +(c(y, g(y)) \cdot X(g(y), f(x)) + d) \triangleleft g(y) \triangleright \delta
\end{array}
$$

that has a single-linear *process-specification*.

2.3 The Translation
Given a syntactically guarded well-formed μCRL *specification* E in BPS and a process p **from** E, we describe in this section the construction of a syntactically guarded μCRL *specification* E' such that

- E' is a μCRL *specification*, obtained from E by the (possible) addition of *sort-*, *function-*, *rewrite-* and *process-specifications* in such a way that p **from** E $\leftrightarrows_{\boldsymbol{A}_{N_{E'}}}$ p **from** E'.
- there is a process p' from E' such that
  - p' satisfies p' **from** E' $\leftrightarrows_{\boldsymbol{A}_{N_{E'}}}$ p **from** E', i.e. p and p' behave the same,
  - p' is a process that is specified in a single-linear way, i.e. the *name* of p' is declared in a single-linear *process-specification* contained in E'.

We just describe the construction of E' by means of examples, and refrain from formal descriptions which are required for a correctness proof. We hope that the suggestion of provability is sufficiently clear.

We distinguish six consecutive steps in this type of construction, each of which should be applied in case its conditions hold. Application of such a step extends the *specification* with at least a *process-specification*. We assume that these extensions always yield a μCRL *specification*, so in particular we assume that the newly added *sort-*, *function-* and *process-specifications* have fresh *names*.

1. Introducing a process expression as a new declaration. Let p **from** E be the object for translation. This step applies whenever p is not of the form n or $n(t_1, ..., t_k)$ for some *process name* n. In this case we extend E to E_1 by adding a *process-specification* that specifies a process p_1 of the form n or $n(t_1, ...t_k)$ that behaves the same as p **from** E_1.

Example of step 1. Let $p \equiv X(t) + b(u)$ where $X(x : S)$ is specified as follows:

$$X(x : S) \quad = \quad a(x) \cdot X(x) + a(x)$$

and the *action-specification* **act** $b : S'$ is also contained in E. We extend E to E_1 by adding the *process-specification*

$$X'(x : S, y : S') \quad = \quad X(x) + b(y)$$

Note that

$$X(t) + b(u) \textbf{ from } E_1 \leftrightarrows_{\boldsymbol{A}_{N_{E_1}}} X'(t, u) \textbf{ from } E_1.$$

(End example.)

2. Translating the process declarations to normal form. Let p_1 **from** E_1 satisfy $p_1 \equiv n$ or $p_1 \equiv n(t_1, ..., t_k)$. This step applies whenever the *process-specification* of p_1 is not in normal form. In this case we extend E_1 to E_2 by adding a *process-specification* in normal form of a process p_2 that behaves the same as p_1 **from** E_2.

Example of step 2. Let $p_1 \equiv X(t)$ where $X(d : D)$, is specified as follows, with $d_0 \in D$ a constant:

$$X(d : D) \quad = \quad \textstyle\sum(e : D, a(d) \cdot X(d_0) \cdot X(e) \cdot X(d)) + b$$

We sketch the technique to obtain a *process-specification* in normal form that defines the same process(es) as $X(d : D)$. The main problem here is the summand $\sum(e : D, a(d) \cdot X(d_0) \cdot X(e) \cdot X(d))$, as it is essentially different from the 'normal form syntax'. We start by replacing this subterm by the term $\sum(e : D, a(d) \cdot X_1(d, e))$. We add the new process declaration

$$X_1(d : D, e : D) \quad = \quad X(d_0) \cdot X(e) \cdot X(d)$$

and thus obtain the specification

$$\begin{array}{lcl} X(d : D) & = & \sum(e : D, a(d) \cdot X_1(d, e)) + b \\ X_1(d : D, e : D) & = & X(d_0) \cdot X(e) \cdot X(d). \end{array}$$

The process declaration for X is now essentially in normal form. We repeat the same step for the process declaration for X_1. The new specification becomes

$$\begin{array}{lcl} X(d : D) & = & \sum(e : D, a(d) \cdot X_1(d, e)) + b \\ X_1(d : D, e : D) & = & X(d_0) \cdot X_2(d, e) \\ X_2(d : D, e : D) & = & X(e) \cdot X(d). \end{array}$$

Having done this, we can replace the specification using the *new* declaration for X, i.e.,

$$\begin{array}{lcl} X(d : D) & = & \sum(e : D, a(d) \cdot X_1(d, e)) + b \\ X_1(d : D, e : D) & = & (\sum(e : D, a(d_0) \cdot X_1(d_0, e)) + b) \cdot X_2(d, e) \\ X_2(d : D, e : D) & = & (\sum(e' : D, a(e) \cdot X_1(e, e')) + b) \cdot X(d). \end{array}$$

Using the axioms for the sum operator distributivity and the conditional construct this gives a specification which is in normal form. From this sketch it follows in what we can extend E_1 to E_2 with a *process-specification* in *normal form* that defines a process behaving like $X(t)$:

$$\begin{array}{lcl} X'(d : D) & = & \sum(e : D, a(d) \cdot X'_1(d, e)) + b \lhd T \rhd \delta \\ X'_1(d : D, e : D) & = & \sum(e : D, a(d_0) \cdot X'_1(d_0, e) \cdot X'_2(d, e)) + b \cdot X'_2(d, e) \lhd T \rhd \delta \\ X'_2(d : D, e : D) & = & \sum(e' : D, a(e) \cdot X'_1(e, e') \cdot X'(d)) + b \cdot X'(d) \lhd T \rhd \delta. \end{array}$$

We remark that a *process-specification* in normal form has a syntax comparable to the *restricted Greibach Normal form* (rGNF) as defined in [BBK87]. They do not give an explicit method to obtain this form but give a sketch in the proof. We believe that their method is more difficult to implement than the method presented above, as we restrict ourselves to syntactically guarded specifications.

(End example.)

3. Disambiguate the formal parameters. Let p_2 **from** E_2 be specified in a *process-specification* that is in normal form. This step applies whenever it is the case that the *process-specification* of p_2 has overloading of variable *names*. By definition of E_2 being Statically Semantically Correct (SSC), this can only be the case if the *process-specification* of p_2 contains more than one declaration. In this case we extend E_2 to E_3 by adding a *process-specification* in normal form that has uniquely typed variable *names*, and that defines a process p_3 that behaves like p_2 **from** E_3.

Example of step 3. Let $p_2 \equiv X(t)$ where $X(x : S)$ is specified as follows:

$$\begin{array}{lcl} X(x:S) & = & (a \cdot Y(f(x)) + b) \lhd t \rhd \delta \\ Y(x:S') & = & (c \cdot X(g(x)) + d(x)) \lhd h(x) \rhd \delta \end{array}$$

We extend E_2 to E_3 by adding the *process-specification*

$$\begin{array}{lcl} X'(x:S) & = & (a \cdot Y'(f(x)) + b) \lhd t \rhd \delta \\ Y'(y:S') & = & (c \cdot X'(g(y)) + d(y)) \lhd h(y) \rhd \delta \end{array}$$

Note that

$$X(t) \textbf{ from } E_3 \leftrightarrows_{\boldsymbol{A}_{N_{E_3}}} X'(t) \textbf{ from } E_3.$$

(End example.)

4. Globalize formal parameters. Let p_3 **from** E_3 be specified in a *process-specification* that is in normal form and that has uniquely typed variable *names*. This step applies whenever it is not the case that the *process-specification* of p_3 has *global parameterization*:

> **Definition 2.7** A *process-specification* in normal form with uniquely typed variable *names* has *global parameterization* iff each occurring variable *name* is declared in *all* of its declarations, that is in all occurring process parameter lists.

Note that a single-linear *process-specification* has by definition global parameterization. If step 4 applies, we extend E_3 to E_4 by adding a *process-specification* in normal form and with uniquely typed variables that has global parameterization, and that defines a process p_4 that behaves like p_3 **from** E_4. The next step will show the purpose of this extension.

Example of step 4. Let $p_3 \equiv X(t)$ and let $X(x : S)$ be specified as follows:

$$\begin{array}{lcl} X(x:S) & = & (a \cdot Y(f(x)) \cdot X(g(x)) + b(x)) \lhd t_1 \rhd \delta \\ Y(y:S') & = & (c \cdot Y(h(y)) + d(y)) \lhd t_2 \rhd \delta \end{array}$$

We extend E_3 to E_4 by adding the *process-specification*

$$\begin{array}{lcl} X'(x:S, y:S') & = & (a \cdot Y'(x, f(x)) \cdot X'(g(x), y) + b(x)) \lhd t_1 \rhd \delta \\ Y'(x:S, y:S') & = & (c \cdot Y'(x, h(y)) + d(y)) \lhd t_2 \rhd \delta \end{array}$$

Note that x and y being different *names* is essential for application of this step. This extension has the following property:

$$X(t) \textbf{ from } E_4 \leftrightarrows_{\boldsymbol{A}_{N_{E_4}}} X'(t, u) \textbf{ from } E_4$$

for any closed *data-term* u of sort S'.
(End example.)

5. Form single declaration. Let p_4 **from** E_4 be specified in a *process-specification* in normal form that has uniquely typed variable *names* and global parameterization. This step applies whenever the *process-specification* of p_4 contains more than one *process-declaration*. In this case we extend E_4 to E_5 by adding a *sort-specification*, a *function-specification* and a *process-specification* containing only one declaration that defines a process p_5 which behaves the same as p_4 **from** E_5. The following example also shows how the data-part of μCRL may be used, and the purpose of global parameterization (step 4).

Example of step 5. Let $p_4 \equiv X'(t, u)$ where $X'(x : S, y : S')$ is specified as in the example of step 4:

$$\begin{array}{lcl} X'(x:S, y:S') & = & (a \cdot Y'(x, f(x)) \cdot X'(g(x), y) + b(x)) \lhd t_1 \rhd \delta \\ Y'(x:S, y:S') & = & (c \cdot Y'(x, h(y)) + d(y)) \lhd t_2 \rhd \delta \end{array}$$

We extend E_4 to E_5 by adding a new sort *Sort* with constants X', Y', an equality function on *Sort* (we use infix notation) and the *process-specification*

$$\begin{array}{rl} Z(n:Sort, x:S, y:S') = (a \cdot Z(Y', x, f(x)) \cdot Z(X', g(x), y) + b(x)) & \lhd\ t_1 \wedge n = X' \rhd \delta \\ +(c \cdot Z(Y', x, h(y)) + d(y)) & \lhd\ t_2 \wedge n = Y' \rhd \delta \end{array}$$

The summands $b(x)$ and $d(y)$ show the purpose of global parameterization: the process Z has to be parameterized with both the sorts S and S' in order to have the *specification* E_5 SSC. Note that indeed

$$X'(t, u) \textbf{ from } E_5 \leftrightarrows_{\boldsymbol{A}_{N_{E_5}}} Z(X', t, u) \textbf{ from } E_5.$$

(End example.)

6. Linearize the process declaration. Let p_5 **from** E_5 be specified in a *process-specification* in normal form containing one *process-declaration*. This step applies whenever the *process-specification* of p_5 is not linear. In this case we extend E_5 to E_6 by adding *sort-*, *function-* and *rewrite-specifications*, and a single-linear *process-specification* that defines a process p_6 that behaves the same as p_5 **from** E_6.

Example of step 6. Let $p_5 \equiv Z(X', t, u)$ where $Z(n : Sort, x : S, y : S')$ is specified as in the example of step 5:

$$\begin{array}{rl} Z(n:Sort, x:S, y:S') = (a \cdot Z(Y', x, f(x)) \cdot Z(X', g(x), y) + b(x)) & \lhd\ t_1 \wedge n = X' \rhd \delta \\ +(c \cdot Z(Y', x, h(y)) + d(y)) & \lhd\ t_2 \wedge n = Y' \rhd \delta \end{array}$$

We add two sorts to E_5. First a sort *Unproper* over which the *data-terms* are of the form X', t', u' and Y', t', u', for all *data-terms* t', u' over the sorts S and S', respectively. Note that this cannot be proper μCRL syntax, as *names* may not contain commas. However, for the purpose of readability we do not care for the moment and underline the elements of the unproper sort.

Next we add a sort *Stack* defined over *Unproper* and the constant λ for the empty stack, and the functions *pop*, *push*, *rest* and *is-empty* with rewrite rules as expected. We extend E_5 to E_6 by also adding the *process-specification*

$$\begin{array}{ll}
Z'(n:S, x:S, y:S', s:Stack) = & \\
\quad (a \cdot Z'(Y', x, f(x), push(\underline{X', g(x), y}, s)) + b(x)) & \lhd\, t_1 \wedge n = X' \wedge \mathit{is\text{-}empty}(s) \rhd \delta \\
+\ (a \cdot Z'(Y', x, f(x), push(\underline{X', g(x), y}, s)) + b(x) \cdot Z'(pop(s), rest(s))) & \\
 & \lhd\, t_1 \wedge n = X' \wedge \neg(\mathit{is\text{-}empty}(s)) \rhd \delta \\
 & \\
+\ (c \cdot Z'(Y', x, h(y), s) + d(y)) & \lhd\, t_2 \wedge n = Y' \wedge \mathit{is\text{-}empty}(s) \rhd \delta \\
+\ (c \cdot Z'(Y', x, h(y), s) + d(y) \cdot Z'(pop(s), rest(s))) & \lhd\, t_2 \wedge n = Y' \wedge \neg(\mathit{is\text{-}empty}(s)) \rhd \delta
\end{array}$$

Note that

$$Z(X', t, u) \text{ from } E_6 \leftrightarrows_{\boldsymbol{A}_{N_{E_6}}} Z'(X', t, u, \lambda) \text{ from } E_6.$$

(End example.)

The general idea behind step 6 is that we can define a sort that has a class of (properly encoded) *process-expressions* as its closed *data-terms*, and a sort *Stack* of stacks over this sort. Upon a summand of the form $a \cdot X \cdot Y$ we stack the subprocess Y, and upon a non-recursive summand of the form a and a non-empty stack, we pop the first subprocess for execution.

3. From μCRL to Single-Linear Specifications

The Basic Process Syntax of the previous section is a concise way to specify processes with data, but somewhat inconvenient to specify protocols. Usually protocols are specified as a parallel composition of processes. Therefore we reintroduce more involved operators (merge, encapsulation etc.) into the syntax. This will make specifying easier, but at the same time we have to be attentive that the specifications we allow can be translated to a linear format.

It is well-known that regularity (and hence linearity) is undecidable when the occurrence of parallelism in the syntax is unrestricted [BK89]. Moreover finiteness conditions as in the case of process algebra *without* data such as in [MV90] become undecidable if processes and data interact.

It will be sufficient for our purposes to exclude specifications like

$$X(n : Int) = a(n) \parallel a(n+1) \cdot X(n+2)$$

where a merge operator is used in the scope of the recursion. For convenience the above mentioned operators will only be used to compose processes which are in BPS, or can be translated to it. Such a strategy is straightforward and is used in e.g. the AUTO tool [SR91] to specify processes. In [Sch94] syntactic conditions similar to ours are formulated and motivated with examples.

We formalize the restriction to a specification with a safe use of parallel operators with the aid of syntactic guardedness.

Definition 3.1 Let E be a well-formed μCRL *specification*. E is *safely linearizable* iff

1. E is the extension of a syntactically guarded μCRL *specification* E_{syng} with (parametrized) process names N_{syng} and,
2. All right hand sides of process declarations in the extension $E - E_{syng}$ are process expressions in which only (parametrized) process names in N_{syng} occur.

Without proof we state that well-formed μCRL *specifications*, which are safely linearizable are bisimilar to linear process specifications (see e.g. [BP94]). The receipt to obtain such a specification is obvious. We translate in an innermost-outermost fashion all process declarations to single-linear format, starting with the declarations in BPS. The other operators are eliminated in the usual way, by expansion and straight forward data parametric substitution, using the recursive specification principle RSP [GP93].

Of course the conditions of Definition 3.1 can be relaxed to allow more nesting. For this an iteration á la syntactic guardedness suffices.

4. Conclusions and Future Work

In this paper we aim at arriving at a single-linear format. We believe that this is a natural format for a parametrized graph or a symbolic transition system. Of course other formats are possible. The use of steps 3–5 can be avoided if we had aimed at a *linear format*, i.e. several coexisting linear declarations. One could say that this is a matter of taste, but we feel that Step 6 becomes more difficult and the resulting specification is less insightful. If several (mutually dependent) process declarations remain, process calls are not uniform and explicit list access has to be introduced, instead of implicit bindings. Also extra control information has to be supplied to process calls, to allow correct selection of the called process. Also in some way or another, process calls have to be stacked with varying types of parameters. The data structure needed will be a list of lists of varying types, and hence be complicated.

At the moment the first author is implementing the above described translation in the ASF+SDF system [Kli93]. This general purpose term rewriting system has several built-in possibilities, among them the possibility to compile rewriting systems to C code. We can make ample use of the fact that μCRL data and process specification is ASF like. We are aiming to integrate this “linearizer” with the well-formedness checker [HK95] developed for μCRL .

We see several next steps. A first (conservative) next step is to build an “instantiator”, a front end which can translate single-linear specifications to labeled transition systems. These can then be interfaced with the tools in the Concur 2 project, which offer various model checking facilities for pure calculi. Of course it will then be essential that all data types are finitary.

A second, more ambitious step is to implement a part of the logic of [GvV94], which is tailored to the syntax of μCRL . An obvious strategy would be to expand modal formulas, to instantiate data and check the pure formulas on a labeled transition system.

Third, we can make a detailed investigation of the complexity of the various steps and suggest optimizations. Furthermore we can look for a class of specifications for which the stacking of processes in Step 6 of Section 2 can be avoided, using the results of [MM94].

Acknowledgements. Jan Bergstra, David Griffioen, Jan-Friso Groote, Wilco Koorn, Eric Madelaine and Frits Vaandrager are thanked for helpful suggestions.

References

[BBK87] J.C.M. Baeten, J.A. Bergstra, and J.W. Klop. Decidability of bisimulation equivalence for processes generating context-free languages. In J.W. de Bakker, A.J. Nijman, and P.C. Treleaven, editors, *Proceedings PARLE conference,* Eindhoven*, Vol. II (Parallel Languages)*, volume 259 of *Lecture Notes in Computer Science*, pages 94–113. Springer-Verlag, 1987.

References

[BDE+93] W. Bouma, M. Dauphin, G.D. Evans, M. Michel, and R. Reed, editors. *SPECS-Specification and Programming Environment for Communicating Software*. North Holland, 1993.

[BG94a] M.A. Bezem and J.F. Groote. A correctness proof of a one bit sliding window protocol in μCRL. *The Computer Journal*, 37(4):289–307, 1994.

[BG94b] M.A. Bezem and J.F. Groote. Invariants in process algebra with data. In Bengt Jonsson and Joachim Parrow, editors, *CONCUR'94: Concurrency Theory*, volume 836 of *Lecture Notes in Computer Science*, pages 401–416. Springer-Verlag, 1994.

[BK89] J.A. Bergstra and J.W. Klop. Process theory based on bisimulation semantics. In J.W. de Bakker, W.P. de Roever, and G. Rozenberg, editors, *REX School/Workshop on Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency,* Noordwijkerhout, The Netherlands, May/June 1988, volume 354 of *Lecture Notes in Computer Science*, pages 50–122. Springer-Verlag, 1989.

[BP94] M.A. Bezem and A. Ponse. Two finite specifications of a queue. Report P9424, Programming Research Group, University of Amsterdam, 1994.

[Bru91] Glenn Bruns. A language for value-passing CCS. Technical Report ECS-LFCS-91-175, Laboratory for Foundations of Computer Science, University of Edinburgh, 1991.

[Bru95] J.J. Brunekreef. Process Specification in a UNITY format. In Ponse et al. [PVvV95], pages 319–337.

[BW90] J.C.M. Baeten and W.P. Weijland. *Process Algebra.* Cambridge Tracts in Theoretical Computer Science 18. Cambridge University Press, 1990.

[EM85] H. Ehrig and B. Mahr. *Fundamentals of algebraic specifications I*, volume 6 of *EATCS Monographs on Theoretical Computer Science.* Springer-Verlag, 1985.

[FGM+92] Jean-Claude Fernandez, Hubert Garavel, Laurent Mounier, Anne Rasse, Carlos Rodríguez, and Joseph Sifakis. A toolbox for the verification of lotos programs. In *Proceedings of the 14th International Conference on Software Engineering ICSE'14 Melbourne, Australia*, 1992.

[GKvV94] J.F. Groote, J.W.C. Koorn, and S.F.M. van Vlijmen. The Safety Guaranteeing System at Station Hoorn-Kersenboogerd. Report 121, Logic Group, Preprint Series, Utrecht University, 1994.

[GP93] 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, *Proceedings of the International Workshop on Semantics of Specification Languages*, Workshops in Computer Science, pages 231–250. Springer-Verlag, 1993.

[GP95] J.F. Groote and A. Ponse. The Syntax and Semantics of μCRL. In Ponse et al. [PVvV95], pages 26–62.

[GS95] J.F. Groote and A. Sellink. Confluence for Process Verification, 1995. To be published in the proceedings of *CONCUR'95.*

[GvV94] J.F. Groote and S.F.M. van Vlijmen. A Modal Logic for μCRL. Research Report 114, Dept. of Philosophy, Utrecht University, May 1994. Also in: *Modal Logic and Process Algebra*, A. Ponse, Y. Venema, and M. de Rijke, editors, CSLI Publications, to appear.

[HK95] J.A. Hillebrand and H. Korver. A Well-formedness Checker for μCRL. Report P9501, Programming Research Group, University of Amsterdam, February 1995.

[HL93] M. Hennessy and X. Liu. A modal logic for message passing processes. Research Report 3/93, University of Sussex, January 1993.

[Hol89] Uno Holmer. Translating Static CCS Agents into Regular Form. PMG report 51, Department of Computer Science, Chalmers University of Technology and the University of Göteborg, 1989.

[KBG93] G. Karjoth, C. Binding, and J. Gustafsson. LOEWE: A LOTOS engineering workbench. *Computer Networks and ISDN Systems*, 25:853–874, 1993.

[Kli93] P. Klint. A meta-environment for generating programming environments. *ACM Transactions on Software Engineering and Methodology*, 2(2):176–201, 1993.

[Mil89] R. Milner. *Communication and Concurrency.* Prentice-Hall International, Englewood Cliffs, 1989.

[MM94] Sjouke Mauw and Hans Mulder. Regularity of BPA-Systems is Decidable. In Bengt Jonsson and Joachim Parrow, editors, *CONCUR'94: Concurrency Theory*, volume 836, pages 34–47. Springer-Verlag, 1994.

[MV90] E. Madelaine and D. Vergamini. Finiteness Conditions and Structural Construction of Automata for all Process Algebras. In *Proceedings of CAV '90*, New Brunswick (NJ), 1990.

[PVvV95] A. Ponse, C. Verhoef, and S.F.M. van Vlijmen, editors. *Algebra of Communicating Processes, Utrecht '94*, Workshops in Computing, Utrecht, 1995. Springer-Verlag.

[Sch94] Marcel Zvi Schreiber. *Value-passing Process Calculi as a Formal Method.* PhD thesis, University of London, 1994.

[SR91] R. de Simone and V. Roy. Auto/Autograph. In E.M. Clarke and R.P. Kurshan, editors, *Proceedings of the 2nd International Conference on Computer-Aided Verification,* New Brunswick, NJ, USA, volume 531 of *Lecture Notes in Computer Science*, pages 65–75. Springer-Verlag, 1991.