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 $\theta$CRL [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 transition system, e.g. [FGM+92, KBG93].

We aim at building a tool which can do model checking for $\theta$CRL [GP95]. $\mu$CRL 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 $\mu$CRL 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 $\mu$CRL 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 $\mu$CRL to parametrized graphs, which we will define syntactically.
We start with a translation of a fragment of $\mu$CRL to single-linear format by means of typical examples. 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 $\mu$CRL 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 $\mu$CRL to a Single-Linear Specification

The specification language $\mu$CRL 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 $\mu$CRL to linear format. It is similar to BPA, in that it contains only alternative and sequential composition. 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 $\mu$CRL 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.

$$
\text{process-expression} ::= \begin{align*}
&\text{process-expression + process-expression} \\
&\text{process-expression} \circ \text{data-term}\circ \text{process-expression} \\
&\sum(\text{single-variable-declaration, process-expression}) \\
&\delta \\
&\text{name} \\
&\text{name(data-term-list)} \\
&(\text{process-expression}).
\end{align*}
$$

In the above definition $+$ is the choice operator, $\circ$ sequential composition and $\circ\triangleright$ is the notation for the if-then-else construct in $\mu$CRL. $\sum$ is the notation for a summation over data. $\delta$ is the deadlocked process. The precedence is in the order $\circ, \circ\triangleright, +$ (as can be seen from definition above).

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

1. Formal approaches to $\mu$CRL proof theory are e.g., [GP93],[BG94b].
2. In linear formats, sequential composition can actually be replaced by action prefixing.
We will use the terms process and action as follows: let \( E \) be a \( \mu \)CRL 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 \( r6(invert(0)) \) \( \hookrightarrow r6(e) \) 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 \( \mu \)CRL 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 \( \mu \)CRL specification \( E \) is a transition system \( \mathcal{A}(A_{N_E}, p \text{ from } E) \) where \( A_{N_E} \) is the canonical term algebra of \( E \), and where the labels of transitions may be parameterized by the fixed representations of the elements of \( A_{N_E} \). These transition systems are considered modulo bisimulation equivalence, notation \( \equiv \) \( A_{N_E} \), as this is the coarsest congruence that respects operational behaviour.

Now processes from syntactically guarded \( \mu \)CRL 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.
2. Translating a Fragment of \( \mu \text{CRL} \) to a Single-Linear Specification

- Instead of repeatedly denoting \( \mu \text{CRL} \) 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 \( \mu \text{CRL} \) 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 \( \mu \text{CRL} \) specification under consideration contains the (standard) functions \( \triangleright \) and \( \triangleright \) on the standard sort \( \text{Bool} \). Applications of the function \( \triangleright \) 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 \( \mu \text{CRL} \) process-specifications in BPS. In their definition we use the symbol \( \beta \) also as a shorthand to denote finite sums (not to be confused with the sum operator of \( \mu \text{CRL} \)):

\[
\sum_{i=1}^{k} p_i
\]

abbreviates \( \delta \) in case \( k = 0 \), and \( p_1 + p_2 + \ldots + p_k \) otherwise.

**Definition 2.4** A process-specification of the form \( p_1 \ldots p_m \) with \( m \geq 1 \) from some \( \mu \text{CRL} \) specification \( E \) is in normal form if for all \( 1 \leq i \leq m \) the declaration \( p_i \) 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

\[
\sum_{j=1}^{k_{ij}} \sum_{k=1}^{X} \Sigma(d_{ijk} : D_{ijk} \cdot a_{ijk} \cdot X_{ijk}^1 \cdot X_{ijk}^2) + \sum_{m=1}^{X} \Sigma(d_{ijm} : D_{ijm} \cdot a_{ijm} \cdot X_{ijm}^3) \]

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 \( X_{ijk}^1, X_{ijk}^2, X_{ijm}^3 \) (possibly parameterized) process-expressions over the names in the left-hand sides of the declarations \( p_{1}, \ldots, p_{m} \).

In the special case that \( k_{ij} = 0 \) for all appropriate \( i, j \) we say that the process-specification \( p_{1} \ldots p_{m} \) is in linear form.

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

**Definition 2.5** Let \( E \) be a \( \mu \text{CRL} \) specification. A process-specification occurring in \( E \) is single-linear if 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:

\[
E \equiv \begin{cases} 
\text{sort} & \text{Bool, } S \\
\text{func} & T, F \rightarrow \text{Bool} \\
& C \rightarrow S \\
& f : \text{Bool} \rightarrow S \\
& g : S \rightarrow \text{Bool} \\
\text{act} & a, d \\
& b : \text{Bool} \\
& c : S \times \text{Bool} \\
\text{proc} & X(x : \text{Bool}, y : S) = (a \cdot X(x, f(x)) + b(x) \cdot x \triangleright \delta + (c(y, g(y)) \cdot X(y(g(y), f(x)) + d) + g(y) \triangleright \delta)
\end{cases}
\]

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 \( E \) \( \equiv A_{E'} \) from \( E' \).
- there is a process \( p' \) from \( E' \) such that
  - \( p' \) satisfies \( p' \) from \( E' \) \( \equiv A_{E'} \) 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 \( E \).

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

\[
X(x : S) = a(x) \cdot X(x) + a(x)
\]

and the action-specification \( \text{act} b : S' \) is also contained in \( E \). We extend \( E \) to \( E_1 \) by adding the process-specification

\[
X'(x : S, y : S') = X(x) + (b(y)
\]
2. **Translating a Fragment of μCRL to a Single-Linear Specification**

Note that

\[ X(t) + b(u) \text{ from } E_1 \equiv A_{\subseteq E_1} X'(t, u) \text{ from } E_1. \]

(End example.)

2. **Translating the process declarations to normal form.** Let \( p_t \) from \( E_1 \) satisfy \( p_t \equiv n \) or \( p_t \equiv n(t_1, \ldots, t_k) \). This step applies whenever the process specification of \( p_t \) 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 \( p_t \) from \( E_2 \).

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

\[
X(d : D) = \sum (e : D, a(d) \cdot X(d_0) \cdot X(e) \cdot X(d)) + b
\]

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

\[
X_1(d : D, e : D) = X(d_0) \cdot X(e) \cdot X(d)
\]

and thus obtain the specification

\[
\begin{align*}
X(d : D) &= \sum (e : D, a(d) \cdot X_1(d, e)) + b \\
X_1(d : D, e : D) &= X(d_0) \cdot X(e) \cdot X(d).
\end{align*}
\]

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

\[
\begin{align*}
X(d : D) &= \sum (e : D, a(d) \cdot X_1(d, e)) + b \\
X_1(d : D, e : D) &= X(d_0) \cdot X_2(d, e) \\
X_2(d : D, e : D) &= X(e) \cdot X(d).
\end{align*}
\]

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

\[
\begin{align*}
X(d : D) &= \sum (e : D, a(d) \cdot X_1(d, e)) + b \\
X_1(d : D, e : D) &= \sum (e : D, a(d_0) \cdot X_1(d_0, e)) + b \cdot X_2(d, e) \\
X_2(d : D, e : D) &= \sum (e' : D, a(e) \cdot X_1(e, e')) + b \cdot X(d).
\end{align*}
\]

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 \( X(t) \):

\[
\begin{align*}
X'(d : D) &= \sum (e : D, a(d) \cdot X_1'(d, e)) + b \cdot T \bowtie \delta \\
X_1'(d : D, e : D) &= \sum (e : D, a(d_0) \cdot X_1'(d_0, e) \cdot X_2'(d, e)) + b \cdot X_2'(d, e) \cdot T \bowtie \delta \\
X_2'(d : D, e : D) &= \sum (e' : D, a(e) \cdot X_1'(e, e') \cdot X'(d)) + b \cdot X'(d) \cdot T \bowtie \delta.
\end{align*}
\]

124
2. Translating a Fragment of $\mu$CRL to a Single-Linear Specification

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 Staticaly 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:

\[
X(x : S) = (a \cdot Y(f(x)) + b) \triangleleft t \triangleright \delta
\]
\[
Y(x : S') = (c \cdot X(g(x)) + d(x)) \triangleleft h(x) \triangleright \delta
\]

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

\[
X'(x : S) = (a \cdot Y'(f(x)) + b) \triangleleft t \triangleright \delta
\]
\[
Y'(y : S') = (c \cdot X'(g(y)) + d(y)) \triangleleft h(y) \triangleright \delta
\]

Note that

\[X(t) \text{ from } E_3 \equiv A_{E_3} X'(t) \text{ from } E_3.\]

(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:

\[
X(x : S) = (a \cdot Y(f(x)) \cdot X(g(x)) + b(x)) \triangleleft t_1 \triangleright \delta
\]
\[
Y(y : S') = (c \cdot Y(h(y)) + d(y)) \triangleleft t_2 \triangleright \delta
\]

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

125
2. **Translating a Fragment of μCRL to a Single-Linear Specification**

\[
\begin{align*}
X'(x : S, y : S') &= (a \cdot Y'(x, f(x)) \cdot X'(g(x), y) + b(x)) \triangleq t_1 \triangleright \delta \\
Y'(x : S, y : S') &= (c \cdot Y'(x, h(y)) + d(y)) \triangleq t_2 \triangleright \delta
\end{align*}
\]

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

\[
X(t) \text{ from } E_4 \models \mathcal{A}_{X,t} X'(t, u) \text{ from } E_4
\]

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 \(E_4\). The following example also shows how the data-part of \(\mu\)CRL 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:

\[
\begin{align*}
X'(x : S, y : S') &= (a \cdot Y'(x, f(x)) \cdot X'(g(x), y) + b(x)) \triangleq t_1 \triangleright \delta \\
Y'(x : S, y : S') &= (c \cdot Y'(x, h(y)) + d(y)) \triangleq t_2 \triangleright \delta
\end{align*}
\]

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

\[
Z(n : \text{Sort}, x : S, y : S') = (a \cdot Z(Y', x, f(x)) \cdot Z(X', g(x), y) + b(x)) \triangleq t_1 \land n = X' \triangleright \delta \\
+ (c \cdot Z(Y', x, h(y)) + d(y)) \triangleq t_2 \land n = Y' \triangleright \delta
\]

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

\[
X'(t, u) \text{ from } E_5 \models \mathcal{A}_{X,t} Z(X', t, u) \text{ from } E_5.
\]

(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 \(p_6\) from \(E_6\).

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

\[
\begin{align*}
Z(n : \text{Sort}, x : S, y : S') &= (a \cdot Z(Y', x, f(x)) \cdot Z(X', g(x), y) + b(x)) \triangleq t_1 \land n = X' \triangleright \delta \\
+ (c \cdot Z(Y', x, h(y)) + d(y)) \triangleq t_2 \land n = Y' \triangleright \delta
\end{align*}
\]

126
3. From μCRL to Single-Linear Specifications

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

$$Z'(n : S, x : S, y : S', s : Stack) =$$

$$(a \cdot Z'(Y', x, f(x), push(X', g(x), y, s)) + b(x)) \quad \triangleq t_1 \land n = X' \land is-empty(s) \triangleright \delta$$

$$+ (a \cdot Z'(Y', x, f(x), push(X', g(x), y, s)) + b(x) \cdot Z'(pop(s), rest(s))) \quad \triangleq t_1 \land n = X' \land \neg(is-empty(s)) \triangleright \delta$$

$$+ (c \cdot Z'(Y', x, h(y), s) + d(y)) \quad \triangleq t_2 \land n = Y' \land is-empty(s) \triangleright \delta$$

$$+ (c \cdot Z'(Y', x, h(y), s) + d(y) \cdot Z'(pop(s), rest(s))) \quad \triangleq t_2 \land n = Y' \land \neg(is-empty(s)) \triangleright \delta$$

Note that

$$Z(X', t, u) \text{ from } E_6 \equiv A_{E_6} Z'(X', t, u, \lambda) \text{ from } E_6.$$  

(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
4. Conclusions and Future Work

1. $E$ is the extension of a syntactically guarded $\mu$CRL specification $E_{\text{syng}}$ with (parametrized) process names $N_{\text{syng}}$ and,

2. All right hand sides of process declarations in the extension $E - E_{\text{syng}}$ are process expressions in which only (parametrized) process names in $N_{\text{syng}}$ occur.

Without proof we state that well-formed $\mu$CRL 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 straightforward 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 à 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, the steps 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 $\mu$CRL data and process specification is ASF like. We are aiming to integrate this “linearizer” with the well-formedness checker [HK95] developed for $\mu$CRL.

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 $\mu$CRL. 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 Madeleine and Frits Vaandrager are thanked for helpful suggestions.

References
References


[Hol89] Uno Holmer. Translating Static CCS Agents into Regular Form. PMG report 51, De-
References

Department of Computer Science, Chalmers University of Technology and the University of Göteborg, 1989.


