Specification and Programming Environment for Communication Software

SPECS

RACE Ref: 1046

Title: Methods for the transformation and analysis of CRL

Author: WP5

Editor: LDM

Type: External Deliverable

SPECS Identifier: D5.7

Document Version: 0

Date: Nov 16, 1990

Status: Draft

Confidentiality: WP5 Internal

The SPECS consortium consists of the following companies:
GSI-TECSI, CNET, STET-CSELT, PTT-RNL, IBM France, NIHE, TFL, INESC, GPT, PKI, EB, BELL, CIT, ELIN, LdM, SESA.

Chapter 1

Analysis of CRL V2.1

1.1 .

1.1.1 Contents

The idea is that in *effective* μCRL we can specify a process in such a way that the structure of the originating *specification* defines a (single) EFSM with the "same" behavior in a canonical way. As the relevant *process-specification* is in that case defined by a very strict syntax, we start from *se*-μCRL, which constitutes a more basic and interesting fragment of μCRL.

We mainly describe techniques for extending a *se*-μCRL *specification* in such a way that any process of interest is bisimilar with a process defined in the extension by a *process-specification* that is suitable for canonical translation. Though the proof theory for (effective) μCRL is yet available, we only show such bisimilarity by means of examples and refrain from formal proofs.

Next we describe a (canonical) translation for a process specified in such a restricted way to an EFSM and we argue that the EFSM obtained from this translation has the "same" behavior. Typical for this translation is that the resulting EFSM's always have two (control) states: one "busy" state, and one state denoting termination.

We then show two alternative approaches, that may lead to EFSM's with a larger number of states.

We conclude with some remarks on 'many-sorted' actions in I-CRL and on the two alternative approaches.

Problems left open. We do not consider the question of the translation of processes that are defined in (non-sequential) effective μCRL to I-CRL.

1.1.2 The source of the translation

An effective μCRL *specification* E is a *sequential effective*$-\mu$CRL *specification*, for short a *se*-μCRL *specification*, iff all *process-declarations* occurring in E have in their right-hand sides *process-expressions* that are *sequential*:

Definition 1.1.1 The syntactical category *seq-process-expression* that constitutes the class of *sequential process-expressions* has the following BNF syntax, which also defines the precedence among operators:

$$
\begin{array}{rcl}
\textit{seq-process-expression} & ::= & \textit{seq-cond-expression} \\
& | & \textit{seq-cond-expression} + \textit{seq-process-expression} \\
\\
\textit{seq-cond-expression} & ::= & \textit{seq-dot-expression} \\
& | & \textit{seq-dot-expression} \lhd \textit{data-term} \rhd \textit{seq-dot-expression} \\
\\
\textit{seq-dot-expression} & ::= & \textit{seq-basic-expression} \\
& | & \textit{seq-basic-expression} \cdot \textit{seq-dot-expression} \\
\\
\textit{seq-basic-expression} & ::= & \delta \\
& | & \tau \\
& | & \textit{name} \\
& | & \textit{name}(\textit{data-term-list}) \\
& | & (\textit{seq-process-expression})
\end{array}
$$

As we consider in the sequel only sequential *process-expressions*, we further omit the adjective 'sequential' and just speak of *process-expressions*.

Example 1.1.2 Consider the following *se-μ*CRL *specification* E:

$$
\begin{array}{lllll}
E & \equiv & \textbf{sort} & \textbf{Bool} & \\
& & \textbf{func} & T, F : \to \textbf{Bool} & \\
& & \textbf{act} & a, b & \\
& & & c : \textbf{Bool} & \\
& & \textbf{proc} & X(x : \textbf{Bool}) & = \ Y \cdot X(x) + a(c(x) \lhd x \rhd (b + c(x) \cdot X(x) \cdot X(x))) \\
& & & Y & = \ b \cdot Y + b
\end{array}
$$

We will use the terms *process* and *action* as follows: let E be a *se-μ*CRL *specification* and p a *process-expression* that is SSC wrt. E and $\emptyset$, then p **from** E is called a process from E. Furthermore an *action* is a process that refers directly to an *action-specification* in E. So in the example above $c(T) + a \cdot X(F)$ is a process from E, and $a, b, c(T), c(F)$ are the actions from E. If E is fixed, we just speak of "the process p".

Given an effective 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 an effective 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 elements of $\boldsymbol{A}_{N_E}$. These transition systems are considered modulo bisimulation equivalence, notation $\leftrightarroweq_{\boldsymbol{A}_{N_E}}$, as this is the coarsest congruence that respects operational behavior.

Now processes from *se-μ*CRL *specifications* constitute the **source language** for the translation described in the sequel.

Conventions. For readability we adopt the following conventions.

- Instead of repeatedly denoting *se-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 *se-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 *se-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. □

1.1.3 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 *se-μCRL process-specifications*. In their definition we use the symbol Σ 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 1.1.3 A *process-specification* of the form $pd_1\ ...\ pd_m$ with $m \geq 1$ from some *se-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} is of the form

$$(\sum_{k=1}^{k_{ij}} a_{ijk} \cdot X^1_{ijk} \cdot X^2_{ijk} + \sum_{k=1}^{l_{ij}} b_{ijk} \cdot X^3_{ijk} + \sum_{k=1}^{m_{ij}} c_{ijk}) \lhd t_{ij} \rhd \delta$$

with the $a_{ijk}, b_{ijk}, c_{ijk}$ (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 an "single-linear" *process-specification*.

Definition 1.1.4 Let E be a *se-μCRL specification*. A *process-specification* occurring in E is *single-linear* iff it is in linear form and contains exactly one *process-declaration*.

Example 1.1.5 Consider the following specification:

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

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

1.1.4 From *se*-μCRL towards single-linear specifications

Given a *se*-μCRL *specification* E and a process p **from** E, we describe in this section the construction of an effective μCRL *specification* E' such that

- E' is a *se*-μCRL *specification*, obtained from E by the (possible) addition of *sort-*, *function-*, *rewrite-* and *process-specifications* (because E' is a *se*-μCRL *specification*, we have that E' is a *conservative extension* of E),
- there is a process p' from E' such that
  - p' satisfies p' **from** $E' \leftrightarrow_{\boldsymbol{A}_{N_{E'}}} p$ **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 *se*-μCRL *specification*, so in particular we assume that the newly added *sort-*, *function-* and *process-specifications* have fresh *names*.

Step 1. Let p **from** E be the object for translation. This step applies whenever p is not of the form n or $n(t_1, \ldots t_k)$ for some *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 $n(t_1, \ldots t_k)$ 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

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

(End example.)

Step 2. 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 $X(x:S)$ is specified as follows:

$$\begin{array}{rcl} X(x:S) & = & a \cdot X(x) \cdot Y(f(x)) \cdot X(x) + b \\ Y(y:S') & = & c \cdot Y(y) + d \end{array}$$

We sketch the technique to obtain a *process-specification* in normal form that defines the same process(es) as $X(x:S)$. The main problem here is the summand $a \cdot X(x) \cdot Y(f(x)) \cdot X(x)$, as it is essentially different from the 'normal form syntax'. We solve this problem as follows: Let $Z(x:S)$ be a (new) *process-specification*, defined by

$$Z(x:S) = X(x) \cdot Y(f(x))$$

then $X(x:S)$ could be exchanged by

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

which specifies the same processes. Having done this, we can replace the specification of the new process $Z(x:S)$ using the new specification of $X(x:S)$, i.e.

$$Z(x:S) = (a \cdot Z(x) \cdot X(x) + b) \cdot Y(f(x))$$

Application of a sound proof rule for μCRL leads to the following equivalences:

$$\begin{array}{rcl} Z(x) & = & a \cdot Z(x) \cdot X(x) \cdot Y(f(x)) + b \cdot Y(f(x)) \\ & = & a \cdot Z(x) \cdot Z(x) + b \cdot Y(f(x)) \end{array}$$

From this sketch it follows in what way we can extend E_1 to E_2 with a *process-specification* in *normal form* that defines a process behaving like $X(t)$:

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

We claim that

$$X(t) \textbf{ from } E_2 \leftrightarrows_{\boldsymbol{A}_{N_{E_2}}} X'(t) \textbf{ from } E_2.$$

(End example.)

We remark that a *process-specification* in normal form has a syntax comparable to the *restricted Greibach Normal Form* (rGNF) as defined in [17]. It is likely that the standard technique for the conversion of a (guarded) *process-specification* to a bisimilar rGNF *process-specification* can be extended to the setting of μCRL. Typical of this extension is then the conversion to 'explicit' guardedness and of conditional constructs to 'head-level'.

Step 3. 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.)

Step 4. 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 1.1.6** 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:

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

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

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

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.)

Step 5. 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 *se-μ*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:

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

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*

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

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.)

Step 6. 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:

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

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*

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

Note that

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

(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.

1.1.5 From single-linear specifications to I-CRL

We do not yet need to consider EFSM's that contain *system rules*, meant to define *Networks* of EFSM's. The (simple) EFSM's without system rules constitute the **target language** of our translation.

Given a *se*-μCRL *specification* E and a process p **from** E defined in a single-linearway, we can define the EFSM $M[p$ **from** $E]$ in a canonical way. We show this by means of an example, in which we furthermore define the concept of *pseudo bisimilarity*. As any EFSM is also associated with a transition system, we can show that p **from** E and $M[p$ **from** $E]$ are in a sense bisimilar. We conclude that our translation yields pseudo bisimilarity.

Example 1.1.7 As an example let $p \equiv X(T, C)$, where E is specified as follows (cf. example 1.1.5):

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

The EFSM $M[p$ **from** $E]$ is instantiated with the 'data-world' of E, i.e. all the sorts, functions and rewrite rules that are defined in E are taken to be present. It is further instantiated with the action *names* declared in E. By default it contains

- the set of (control) states $\{\top, \bot\}$,
- the initial state $\top$,
- the final state $\bot$.

The *process-specification* of $X(T, C)$ further determines the definition of $M[p$ **from** $E]$ in the following canonical way: it is defined over the *state variables* x of sort **Bool** and y of sort S, and has

- the rules with *many-sorted*[1] actions

$$
\begin{array}{lllll}
\langle a, & \top, & \top, & x, & y := f(x)\rangle, \\
\langle b!x, & \top, & \bot, & x, & nop\rangle, \\
\langle c!y!g(y), & \top, & \top, & g(y), & x := g(y), y := f(x)\rangle, \\
\langle d, & \top, & \bot, & g(y), & nop\rangle,
\end{array}
$$

- the initialization statement $x := T, y := C$.

We now argue that the transition system $\mathcal{A}(\boldsymbol{A}_{N_E}, X(T)$ **from** $E)$ and the transition system for $M[X(T)$ **from** $E]$ are in a sense bisimilar.

Let Θ be the set of all ground substitutions over the set of variables $\{\langle x : \textbf{Bool}\rangle, \langle y : S\rangle\}$, and let the relation

$$R \subseteq \langle S(E) \cup \{\surd\}\rangle \times \langle\{\top, \bot\} \times \Theta\rangle$$

[1] We return to this point in section 1.1.8.

where $S(E)$ is the set of processes from E be defined by

$$\begin{array}{rcll} X(\theta(x),\theta(y)) & R & \langle\top,\theta\rangle & \text{for all } \theta\in\Theta \\ \surd & R & \langle\bot,\theta\rangle & \text{for all } \theta\in\Theta \end{array}$$

We show by two typical cases that R satisfies a transfer property:

$$\begin{array}{ll} & X(\theta(x),\theta(y)) \xrightarrow{a()} X(\theta'(x),\theta'(y)) \\ \text{iff} & \\ & [\![x]\!]^{\theta} = T \text{ and } \theta' = Env(y := f(x),\theta) \\ \text{iff} & \\ & \langle\top,\theta\rangle \xrightarrow{a} \langle\top,\theta'\rangle \end{array}$$

and

$$\begin{array}{ll} & X(\theta(x),\theta(y)) \xrightarrow{b(N_E(\theta(x)))} \surd \\ \text{iff} & \\ & [\![x]\!]^{\theta} = T \text{ (and } \theta = Env(nop,\theta)) \\ \text{iff} & \\ & \langle\top,\theta\rangle \xrightarrow{b!\theta(x)} \langle\bot,\theta\rangle \end{array}$$

The second case shows that the data in the labels may be (syntactically) different, as these are always *normal forms* in effective μCRL (for any closed *data-term* t, the expression $N_E(t)$ denotes its normal form). Because the relation R satisfies the transfer property as illustrated above, we say that $X(T)$ **from** E and $M[X(T)$ **from** $E]$ are *pseudo bisimilar*, notation

$$X(T) \textbf{ from } E \leftrightarroweq_{\boldsymbol{A}_{N_E}} M[X(T) \textbf{ from } E].$$

1.1.6 Correctness of the translation

Given a *se-*μCRL *specification* E and a process p **from** E, the extension of E to E_6 as described in the six steps in section 1.1.4 defines a process p_6 **from** E_6 in a single-linear way that satisfies

$$p \textbf{ from } E_6 \leftrightarroweq_{\boldsymbol{A}_{N_{E_6}}} p_6 \textbf{ from } E_6.$$

The conversion of the process p_6 **from** E_6 to the EFSM $M[p_6$ **from** $E_6]$ as described in section 1.1.5 satisfies

$$p_6 \textbf{ from } E_6 \leftrightarroweq_{\boldsymbol{A}_{N_{E_6}}} M[p_6 \textbf{ from } E_6]$$

because our translation always admits the (canonical) definition of a relation like R in example 1.1.7 that satisfies a transfer property as illustrated there. By definition of bisimulation equivalence in μCRL this leads to

$$p \textbf{ from } E_6 \leftrightarroweq_{\boldsymbol{A}_{N_{E_6}}} M[p_6 \textbf{ from } E_6].$$

Though bisimilarity is in μCRL parameterized by *one* specification, we know here that E_6 is a conservative extension of E, and therefore we may as well write

$$p \textbf{ from } E \leftrightarroweq_{\boldsymbol{A}_{N_{E_6}}} p_6 \textbf{ from } E_6$$

and therefore also

$$p \textbf{ from } E \leftrightarroweq_{\mathbf{A}_{N_{E_6}}} M[p_6 \textbf{ from } E_6].$$

Hence $M[p_6 \textbf{ from } E_6]$ can be qualified as a correct translation of the initial object of translation p **from** E.

1.1.7 Two alternative approaches

First alternative. An alternative approach is to define a more liberal format of a *process-specification* that allows a canonical translation to an EFSM of which the *number* of states depends on the number of declarations:

Definition 1.1.8 Let E be a *se-μ*CRL *specification*. A *process-specification* occurring in E is *EFSM-like* iff it is in linear form and contains no overloading of variable *names*.

Example 1.1.9 Consider the following specification E of the process $X(T)$:

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

Note that the first three steps in section 1.1.4 *may* already lead to a defining *process-specification* that is EFSM-like, namely in the case that summands of the form aXY are absent.

We show by means of an example how a process defined by an EFSM-like *process-specification* also defines an EFSM in a canonical way. The difference with the translation described in section 1.1.5 is now that each process *name* defines a separate state.

Example 1.1.10 Let $p \equiv X(T)$, where $X(x : \textbf{Bool})$ is specified as in example 1.1.9. The EFSM $M^1[p \textbf{ from } E]$ is again instantiated with the 'data-world' of E, i.e. all the sorts, functions and rewrite rules that are defined in E are taken to be present. It is further instantiated with the action *names* declared in E. By default it contains

- the final state $\perp$.

The *process-specification* of $X(T)$ further determines the definition of $M^1[p \textbf{ from } E]$ in the following canonical way: it is defined over the *state variables* x of sort **Bool** and y of sort S, and has

- the set of (control) states $\{X, Y\}$,
- the initial state X,
- the rules

$$\begin{array}{lllll} \langle a, & X, & Y, & x, & y := f(x)\rangle, \\ \langle b!x, & X, & \perp, & x, & nop\rangle, \\ \langle c!y!g(y), & Y, & X, & g(y), & x := g(y)\rangle, \\ \langle d, & Y, & \perp, & g(y), & nop\rangle, \end{array}$$

- the initialization statement $x := T, y := C$, where C is an arbitrary closed data-term of sort S (note that by effectiveness of E the sort S is non-empty).

It is not hard to see that p **from** $E \leftrightarrow_{\boldsymbol{A}_{N_E}} M^1[p$ **from** $E]$.

So this alternative approach comes down to

1. applying the first three steps of the construction described in section 1.1.4,
2. in case this yields a bisimilar process specified by an EFSM-like *process-specification*, then to apply the canonical translation as sketched above,
3. in case this yields a bisimilar process specified by a *process-specification* that is *not* EFSM-like, then to continue the procedure as described in sections 1.1.4 and 1.1.5.

Second alternative. A second alternative for translation is to encode all *finite* parameters in the 'control' of a *process-specification* before translation, thus obtaining in general a larger number of control states after translation. We illustrate this technique again by an example:

Example 1.1.11 Consider the following extension of the *specification* E from example 1.1.9 that defines the process X_T behaving like $X(T)$. The finite parameter **Bool** gives rise to the new process *names* X_T and X_F.

$$\begin{array}{llcl} \textbf{proc} & X_T & = & (a \cdot Y'(f(T)) + b(T)) \triangleleft T \triangleright \delta \\ & X_F & = & (a \cdot Y'(f(F)) + b(F)) \triangleleft F \triangleright \delta \\ & Y'(y:S) & = & (c(y, g(y)) \cdot X_T + d) \triangleleft g(y) \triangleright \delta \\ & Y'(y:S) & = & (c(y, g(y)) \cdot X_F + d) \triangleleft \neg(g(y)) \triangleright \delta \end{array}$$

Note that this *process-specification* is EFSM-like and leads to a canonical translation in the same way as sketched above, but now with *four* different (control) states: $\{X_T, X_F, Y', \perp\}$, the initial state X_T and the initialization statement $y := C$ for some closed *data-term* C of sort S.

1.1.8 Remarks

8.1 Many-sorted actions. We slightly extended the definition of *actions* in I-CRL to *many-sorted* actions, i.e. expressions like

$$a?x!f(y) \quad \text{or} \quad b$$

where a and b are gate names. We feel that such an extension corresponds with the fact that the *states* of an EFSM are also subject to many-sorted parameterization.

If, however, one would insist on only allowing 'single-sorted' actions, then we can extend μCRL *specifications* (or for that matter of course also the data-world of EFSM's in our 'extended' I-CRL) with new sorts, appropriate function- and *rewrite-specifications* such that any parameterization can be mimicked by single-sorted parameterization over one of the new sorts. This can be obtained by standard embedding techniques, or the addition of 'dummy' sorts.

Example 1.1.12 The *action-specification*

$$\textbf{act } a$$

could give rise to the extension

$$\textbf{act } a : Dummy$$

where $Dummy$ is a newly added sort containing one (irrelevant) constant $dummy$.

Of course the $\stackrel{a()}{\longrightarrow}$ and $\stackrel{a(dummy)}{\longrightarrow}$ transitions illustrate the necessity of adapting the notion of 'bisimilarity' in this case.

8.2 Actions containing input offers. A possible employ for actions containing input offers in EFSM's obtained from translation is to admit the *sum operator* of μCRL in sequential *process-expressions*. In that case we can allow in definition 1.1.3 that the parameterization is organized by this operator. A summand of the form

$$\sum(x : S, a(x)...$$

would then translate to an action

$$a?x$$

and the canonical translation thus obtained also yields pseudo bisimilarity. Note however that in *se*-μCRL this means that the sort S has to be finite.

8.3 EFSM-like versus single-linear. In case the first three steps of the construction described in section 1.1.4 yield a process bisimilar with the object for translation, but defined by a process-specification that is in normal, *non-linear* form (so that is *not* EFSM-like), we cannot (yet) provide a technique for conversion to an EFSM-like, *non-single-linear specification*. Reason for this is that in order to keep track of termination options, we use a single *name* that organizes control.

Chapter 2

Bibliography

[1] G.J. Akkerman and J.C.M. Baeten
Term Rewriting Analysis in Process Algebra.
Report P9006, Programming Research Group, Univerisity of Amsterdam, 1990.

[2] T. Bolognesi and S.A. Smolka
Fundamental results for the verification of observational equivalence: a survey.
Proceedings 7^{th} IFIP, WG6.1 International Symposium on Protocol Specification, Testing, and Verification Zürich, Switserland, May 1987

[3] R. Cleaveland
On Automatically Distinguishing Inequivalent Processes
Proceedings: 1990 Workshop on Computer-Aided Verification, DIMACS technical report 90-31, Vol. 2, New Yersey, 1990. To appear in Lecture Notes in Computer Science.

[4] R. Cleaveland and M. Hennessy
Testing Equivalence as a Bisimulation Equivalence
In Proceedings of the Workshop on Automatic Verification Methods for Finite-State Systems, Lecture Notes in Computer Science 407, 11–23. Springer-Verlag, Berlin, 1990.

[5] R. DeNicola and F.W. Vaandrager
Three logics for branching bisimulation (extended abstract)
In Proceedings 5^{th} Annual Symposium on Logic in Computer Science, Philadelphia, USA, pages 118–129, Los Alamitos, CA, 1990. IEEE Computer Society Press. Full version appeared as CWI Report CS-R9012.

[6] R. DeNicola and F.W. Vaandrager
Action versus state based logics for transition systems.
In Proceedings Ecole de Printemps on Semantics of Concurrency, April 90, (I. Guessarian), Springer-Verlag, LNCS, 1990.

[7] P. Ernberg, L. Fredlund and B. Jonsson
Specification and Validation of a Simple Overtaking Protocol using LOTOS

SICS technical report, T90006, ISSN 1100-3154, 1990.

[8] R.J. van Glabbeek and W.P. Weijland
Branching time and abstraction in bisimulation semantics (extended abstract)
Information Processing 89, pages 613–618. North-Holland, 1989.

[9] R.J. van Glabbeek and W.P. Weijland
Refinement in branching time semantics.
Report CS-R8922, CWI, Amsterdam, 1989. Also appeared in Proceedings AMAST Conference, Iowa, USA, pp. 197–201.1989.

[10] J. Godskesen, K. Larsen and M. Zeeberg
TAV User Manual
Technical report R 89-19, 1989.

[11] J.F. Groote and F.W. Vaandrager
An efficient algorithm for branching bisimulation and stuttering equivalence.
In Proceedings 17^{th} ICALP, Warwick, volume 443 of LNCS, pages 626–638. Springer-Verlag, 1990.

[12] M. Hennessy and R. Milner
Algebraic Laws for Nondeterminism and Concurrency.
Journal of the Association for Computing Machinery, v.S 32, n. 1, pages 137-161, January 1985.

[13] M. Hillerström
Verification of CCS-processes.
M.Sc. Thesis, Computer Science Department, Aalborg University, 1987.

[14] P. Kanellakis and S.A. Smolka
CCS Expressions, Finite State Processes, and Three Problems of Equivalence.
In Proceedings of the Second ACM Symposium on the Principles of Distributed Computing, 1983.

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

[16] R. Paige and R.E. Tarjan
Three Partition Refinement Algorithms.
In SIAM Journal of Computing, v. 16, n. 6, pages 973-989, December 1987.

[17] J.C.M. Baeten, J.A. Bergstra & J.W. Klop
Decidability of bisimulation equivalence for processes generating context-free languages
In Proceedings PARLE, Lecture Notes in Computer Science 259, 94–113. Springer-Verlag, Berlin, 1987.

[18] Günter Karjoth
Definition of I-CRL
SPECS Draft MP.WP6.4.03.,

[19] J.F. Groote and A. Ponse
The syntax and semantics of μCRL
SPECS A.WP5.1401-1.,

[20] J.F. Groote and A. Ponse
The syntax and semantics of μCRL
SPECS A.WP5.1401-1.,

[21] J.C.M. Baeten and J.A. Bergstra
Process algebra with signals and conditions
Report P9008, Programming Research Group, Univerisity of Amsterdam, 1990.

[22] J.A. Bergstra and J.W. Klop
Process algebra for synchronous communication
Information and Computation, v.S 60, n. 1/3, pages 109–137, 1984.

[23] J.C.M. Baeten and W.P. Weijland
Process Algebra.
Cambridge University Press, 1990,

[24] D. van Dalen
Logic and Structure.
Springer-Verlag, 1983,

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

[26] R.J. van Glabbeek
Comparative Concurrency Semantics and Refinement of Actions.
PhD Thesis, Free University, Amsterdam, 1990.

[27] J.F. Groote and F.W. Vaandrager
Structured operational semantics and bisimulation as a congruence (extended abstract).
Proceedings 16^{th} ICALP, Stresa, LNCS 372, Springer-Verlag, 1989. Full version to appear in *Information and Computation*

[28] C.A.R. Hoare, I.J. Hayes, He Jifeng, C.C. Morgan, A.W. Roscoe, J.W. Sanders, I.H. Sorensen, J.M. Spivey and B.A. Sufrin
Laws of programming.
Communications of the Association for Computing Machinery, v.S 30, n. 8, pages 672–686, August 1987

[29] C.A.R. Hoare
Communicating Sequential Processes.
Prentice Hall, 1985.,

[30] R. Milner
A Calculus of Communicating Systems.
LNCS 92, Springer-Verlag, 1980.,

[31] S. Mauw and G.J. Veltink
A process specification formalism.
Fundamenta Informaticae, v.S XIII, pages 85–139, 1990.

[32] D.M.R. Park
Concurrency and automata on infinite sequences.
LNCS 104, Springer-Verlag, 1981.,

[33] SPECS-semantics
Definition of MR and CRL Version 2.1
1990.,

[34] M.E. Szabo
The Collected Papers of Gerhard Gentzen.
North-Holland, 1969.,

[35] A.S. Troelstra and D. van Dalen
Constructivism in Mathematics, An Introduction (vol I).
North-Holland, 1988.,