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

Copyright © by the SPECS consortium.
The SPECS consortium consists of the following companies:
GSI-TECSI, CNET, STET-CSELIT, PTT-RNL, IBM France, NIHE, TFL, INESC, GPT, PKI, EB, BELL, CIT, ELIN, LdM, SESA.
Chapter 1

Analysis of CRL V2.1

1.1 Contents

The idea is that in effective \(\mu\)CRL 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 \(se-\mu\)CRL, which constitutes a more basic and interesting fragment of \(\mu\)CRL.

We mainly describe techniques for extending a \(se-\mu\)CRL 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) \(\mu\)CRL 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 \(\mu\)CRL to I-CRL.

1.1.2 The source of the translation

An effective \(\mu\)CRL specification \(E\) is a sequential effective—\(\mu\)CRL specification, for short a \(se-\mu\)CRL specification, if and only if process declarations occurring in \(E\) have in their right-hand sides process expressions that are sequential:
Definition 1.1.1 The syntactical category \textit{seq-process-expression} that constitutes the class of \textit{sequential process-expressions} has the following BNF syntax, which also defines the precedence among operators:

\[
\begin{align*}
\text{seq-process-expression} & ::= \text{seq-cond-expression} \\
& \quad | \text{seq-cond-expression} + \text{seq-process-expression} \\
\text{seq-cond-expression} & ::= \text{seq-dot-expression} \\
& \quad | \text{seq-dot-expression} < \text{data-term} > \text{seq-dot-expression} \\
\text{seq-dot-expression} & ::= \text{seq-basic-expression} \\
& \quad | \text{seq-basic-expression} \cdot \text{seq-dot-expression} \\
\text{seq-basic-expression} & ::= \delta \\
& \quad | \tau \\
& \quad | \text{name} \\
& \quad | \text{name(}\text{data-term-list}\text{)} \\
& \quad | (\text{seq-process-expression})
\end{align*}
\]

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

Example 1.1.2 Consider the following \textit{se-μCRL specification} $E$:

\[
E = \begin{array}{l}
\text{sort } \text{Bool} \\
\text{func } T,F : \rightarrow \text{Bool} \\
\text{act } a,b \\
\text{name } c : \text{Bool} \\
\text{proc } X(x : \text{Bool}) = Y \cdot X(x) + a(c(x) < x > (b + c(x) \cdot X(x) \cdot X(x))) \\
\text{name } Y = b \cdot Y + b
\end{array}
\]

We will use the terms \textit{process} and \textit{action} as follows: let $E$ be a \textit{se-μCRL specification} and $p$ a \textit{process-expression} that is SSC wrt. $E$ and $\emptyset$, then $p$ from $E$ is called a process from $E$. Furthermore an \textit{action} is a process that refers directly to an \textit{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 \textit{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 $\mathcal{A}(\mathcal{A}_{N_E}, p \text{ from } E)$ where $\mathcal{A}_{N_E}$ is the canonical term algebra of $E$, and where the labels of transitions may be parameterized by the elements of $\mathcal{A}_{N_E}$. These transition systems are considered modulo bisimulation equivalence, notation $\cong_{\mathcal{A}_{N_E}}$, as this is the coarsest congruence that respects operational behavior.

Now processes from \textit{se-μCRL specifications} constitute the \textbf{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 ¬ and ∧ on the standard sort Bool. Applications of the function ∧ 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 p₁, p₂, ... be process-expressions, then the expression

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

abbreviates δ in case k = 0, and p₁ + p₂ + ... + pₖ otherwise.

**Definition 1.1.3** A process-specification of the form pd₁ ... pdₘ with m ≥ 1 from some se-µCRL specification E is in normal form iff for all 1 ≤ i ≤ m the declaration pdᵢ has a right-hand side of the form

\[ \sum_{j=1}^{k_{i}} p_{ij} \]

where each of the process-expressions pᵢⱼ is of the form

\[ \left( \sum_{k=1}^{k_{ij}} a_{ijk} \cdot X_{ijk}^{1} \cdot X_{ijk}^{2} + \sum_{k=1}^{t_{ij}} b_{ijk} \cdot X_{ijk}^{3} + \sum_{k=1}^{c_{ijk}} c_{ijk} \right) < t_{ij} > \delta \]

with the aᵢⱼ, bᵢⱼ, cᵢⱼ (possibly parameterized) process-expressions over the names in the action-specifications from E, and the X₁ᵢⱼ, X₂ᵢⱼ, X₃ᵢⱼ (possibly parameterized) process-expressions over the names in the left-hand sides of the declarations pd₁, ..., pdₘ.

In the special case that kᵢⱼ = 0 for all appropriate i, j we say that the process-specification pd₁ ... pdₘ 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:

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

that has a single-linear \textit{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 processSpecifications (because \( E' \) is a se-μCRL specification, we have that \( E' \) is a \textit{conservative extension} of \( E \)),

- there is a process \( p' \) from \( E' \) such that

  - \( p' \) satisfies \( p' \text{ from } E' \Leftrightarrow A_{E'}^p p \text{ from } E' \), 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 \textit{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 \textit{specification} with at least a \textit{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 processSpecifications have fresh names.

Step 1. Let \( p \text{ from } E \) 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 \textit{process-specification} that specifies a process \( p_1 \) of the form \( n \) or \( n(t_1, \ldots, t_k) \) that behaves the same as \( p \text{ from } E_1 \).
Example of step 1. Let \( p \equiv X(t) + b(u) \) where \( X(x : S) \) is specified as follows:

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

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

\[
X'(x : S, y : S') = X(x) + b(y)
\]

Note that

\[
X(t) + b(u) \text{ from } E_1 \Leftrightarrow A_{x_1} X'(t, u) \text{ from } E_1.
\]

(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:

\[
X(x : S) = a \cdot X(x) \cdot Y(f(x)) \cdot X(x) + b
\]
\[
Y(y : S') = c \cdot Y(y) + d
\]

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 \( \mu \text{CRL} \) leads to the following equivalences:

\[
Z(x) = a \cdot Z(x) \cdot X(x) \cdot Y(f(x)) + b \cdot Y(f(x))
\]
\[
= a \cdot Z(x) \cdot Z(x) + b \cdot Y(f(x))
\]
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 $X(t)$:

\[
\begin{align*}
X'(x : S) &= (a \cdot Z(x) \cdot X'(x) + b) < T > \delta \\
Y'(y : S') &= (c \cdot Y'(y) + d) < T > \delta \\
Z'(x : S) &= (a \cdot Z'(x) \cdot Z'(x) + b \cdot Y'(f(x))) < T > \delta
\end{align*}
\]

We claim that

\[X(t) \text{ from } E_2 \equiv \mathcal{A}_{\kappa E_2} X'(t) \text{ from } E_2.\]

(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 $\mu$CRL. 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 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:

\[
\begin{align*}
X(x : S) &= (a \cdot Y(f(x)) + b) < t > \delta \\
Y(x : S') &= (c \cdot X(g(x)) + d(x)) < h(x) > \delta
\end{align*}
\]

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

\[
\begin{align*}
X'(x : S) &= (a \cdot Y'(f(x)) + b) < t > \delta \\
Y'(y : S') &= (c \cdot X'(g(y)) + d(y)) < h(y) > \delta
\end{align*}
\]

Note that

\[X(t) \text{ from } E_3 \equiv \mathcal{A}_{\kappa E_3} X'(t) \text{ from } E_3.\]

(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:

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

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

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

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

\[
X(t) \text{ from } E_4 \Leftrightarrow_{\mathbb{A}_{X_{E_4}}} X'(t, u) \text{ from } E_4
\]

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-$\mu$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:

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

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

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

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

\[
X'(t, u) \text{ from } E_5 \Leftrightarrow_{\mathbb{A}_{X_{E_5}}} Z(X', t, u) \text{ from } E_5.
\]

*(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:

$$Z(n : Sort, x : S, y : S') = (a \cdot Z(X', x, f(x)) \cdot Z(X', g(x), y) + b(x)) + t_1 \wedge n = X' \triangleright \delta$$

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 $\mu$CRL 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 $\lambda$ 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

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

$$(a \cdot Z'(Y', x, f(x), push(X', g(x), y, s)) + b(x)) + t_1 \wedge n = X' \triangleright \delta$$

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 $\mu$CRL 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 $\lambda$ 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

$$Z(X', t, u) \text{ from } E_6 \Leftrightarrow A_{x_0} Z'(X', t, u, \lambda) \text{ from } E_6.$$

(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.

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-$\mu$CRL specification $E$ and a process $p$ from $E$ defined in a single-linearway, we can define the EFSM $M[p \text{ from } E]$ 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):

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

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 \( \text{Bool} \) and \( y \) of sort \( S \), and has

- the rules with many-sorted\(^1\) actions

\[
\begin{align*}
\langle a, & \top, \top, x, \; y := f(x) \rangle, \\
\langle b!x, & \top, \bot, x, \; \text{nop} \rangle, \\
\langle c!y!g(y), & \top, \top, g(y), \; x := g(y), y := f(x) \rangle, \\
\langle d, & \top, \bot, g(y), \; \text{nop} \rangle,
\end{align*}
\]

- the initialization statement \( x := T, y := C \).

We now argue that the transition system \( A(A_{N_e}, X(T) \) from \( E \) and the transition system for \( M[X(T) \) from \( E \) are in a sense bisimilar.

Let \( \Theta \) be the set of all ground substitutions over the set of variables

\( \{\langle x : \text{Bool} \rangle, \langle y : S \rangle\} \), and let the relation

\[
R \subseteq \langle S(E) \cup \{\sqrt{\}\} \rangle \times \langle \{\top, \bot\} \times \Theta \rangle
\]

\(^{1}\)We return to this point in section 1.1.8.
where \( S(E) \) is the set of processes from \( E \) be defined by

\[
X(\theta(x), \theta(y)) \xrightarrow{R} \langle \top, \theta \rangle \quad \text{for all } \theta \in \Theta \\
\sqrt{R} \langle \bot, \theta \rangle \quad \text{for all } \theta \in \Theta
\]

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

\[
X(\theta(x), \theta(y)) \xrightarrow{a} X(\theta'(x), \theta'(y))
\]

iff

\[
[x]^{\theta} = T \quad \text{and} \quad \theta' = \text{Env}(y := f(x), \theta)
\]

iff

\[
\langle \top, \theta \rangle \xrightarrow{a} \langle \top, \theta' \rangle
\]

and

\[
X(\theta(x), \theta(y)) \xrightarrow{b(N_E(\theta(x))]} \sqrt{
\]

iff

\[
[x]^{\theta} = T \quad \text{(and} \quad \theta = \text{Env}(\text{nop}, \theta))
\]

iff

\[
\langle \top, \theta \rangle \xrightarrow{b(\theta(x))} \langle \bot, \theta \rangle
\]

The second case shows that the data in the labels may be (syntactically) different, as these are always normal forms in effective \( \mu \)CRL (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 \( X(T) \) \textbf{from} \( E \) and \( M[X(T) \textbf{from} E] \) are pseudo bisimilar, notation

\[
X(T) \textbf{from} E \equiv_{A_{N_E}} M[X(T) \textbf{from} E].
\]

### 1.1.6 Correctness of the translation

Given a se-\( \mu \)CRL specification \( E \) and a process \( p \) \textbf{from} \( E \), the extension of \( E \) to \( E_6 \) as described in the six steps in section 1.1.4 defines a process \( p_6 \) \textbf{from} \( E_6 \) in a single-linear way that satisfies

\[
p \textbf{from} E_6 \equiv_{A_{N_E}} p_6 \textbf{from} E_6.
\]

The conversion of the process \( p_6 \) \textbf{from} \( E_6 \) to the EFSM \( M[p_6 \textbf{from} E_6] \) as described in section 1.1.5 satisfies

\[
p_6 \textbf{from} E_6 \equiv_{A_{N_E}} M[p_6 \textbf{from} E_6]
\]

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 \( \mu \)CRL this leads to

\[
p \textbf{from} E_6 \equiv_{A_{N_E}} M[p_6 \textbf{from} E_6].
\]

Though bisimilarity is in \( \mu \)CRL parameterized by one specification, we know here that \( E_6 \) is a conservative extension of \( E \), and therefore we may as well write

\[
p \textbf{from} E \equiv_{A_{N_E}} p_6 \textbf{from} E_6
\]
and therefore also
\[ p \text{ from } E \leftrightarrow A_{E_{k_0}} M[p_{k_0} \text{ from } E_0]. \]

Hence \( M[p_{k_0} \text{ from } E_0] \) can be qualified as a correct translation of the initial object of translation \( p \text{ from } E \).

### 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-\( \mu \)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 \( X(T) \):

\[
E \equiv \begin{align*}
\text{sort} & \quad \text{Bool, } S \\
\text{func} & \quad T, F : \to \text{Bool} \\
& \quad f : \text{Bool} \to S \\
& \quad g : S \to \text{Bool} \\
\text{var} & \quad \ldots \\
\text{rew} & \quad \ldots \\
\text{act} & \quad a, d \\
& \quad b : \text{Bool} \\
& \quad c : S \times \text{Bool} \\
\text{proc} & \quad X(x : \text{Bool}) = (a \cdot Y(f(x)) + b(x)) \\& \quad Y(y : S) = (c(y, g(y)) \cdot X(g(y)) + d) \\& \quad \delta
\end{align*}
\]

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 \( aX Y \) 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 \( X(x : \text{Bool}) \) is specified as in example 1.1.9. The EFSM \( M^1[p \text{ from } E] \) 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 \( \bot \).

The process-specification of \( X(T) \) further determines the definition of \( M^1[p \text{ from } E] \) in the following canonical way: it is defined over the state variables \( x \) of sort \( \text{Bool} \) and \( y \) of sort \( S \), and has
• the set of (control) states \( \{X, Y\} \),
• the initial state \( X \),
• the rules
\[
\begin{align*}
\langle a, X, Y, x, y := f(x) \rangle, \\
\langle b!x, X, \bot, x, \text{nop} \rangle, \\
\langle c!y!g(y), Y, X, g(y), x := g(y) \rangle, \\
\langle d, Y, \bot, g(y), \text{nop} \rangle,
\end{align*}
\]
• 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 \( p \text{ from } E \Leftrightarrow A_{\kappa E} M^1[p \text{ from } E] \).

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 \( \text{Bool} \) gives rise to the new process names \( X_T \) and \( X_F \).

\[
\begin{align*}
\text{proc } X_T &= (a \cdot Y'(f(T)) + b(T)) \triangleright T \triangleright \delta \\
\text{proc } X_F &= (a \cdot Y'(f(F)) + b(F)) \triangleright F \triangleright \delta \\
Y'(y : S) &= (c(y, g(y)) \cdot X_T + d) \triangleright g(y) \triangleright \delta \\
Y'(y : S) &= (c(y, g(y)) \cdot X_F + d) \triangleright \neg (g(y)) \triangleright \delta
\end{align*}
\]

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: \( \{X_T, X_F, Y', \bot\} \), 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

\[
a ? x! f(y) \text{ or } b
\]
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 \(\mu\text{CRL specifications}\) (or for that matter of course also the data-world of EFSM’s in our ‘extended’ \(I\text{-CRL}\)) 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*

\[
\text{act } a
\]

could give rise to the extension

\[
\text{act } a : \text{Dummy}
\]

where \(\text{Dummy}\) is a newly added sort containing one (irrelevant) constant \(\text{dummy}\).

Of course the \(a(\text{dummy})\) 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 \(\mu\text{CRL}\) 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 \(se-\mu\text{CRL}\) 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

   *Term Rewriting Analysis in Process Algebra.*

   *Fundamental results for the verification of observational equivalence: a survey.*

   *On Automatically Distinguishing Inequivalent Processes*

   *Testing Equivalence as a Bisimulation Equivalence*

   *Three logics for branching bisimulation (extended abstract)*

   *Action versus state based logics for transition systems.*

   *Specification and Validation of a Simple Overtaking Protocol using LOTOS*
SICS technical report, T90006, ISSN 1100-3154, 1990.


Branching time and abstraction in bisimulation semantics (extended abstract)


Refinement in branching time semantics.


TAV User Manual


An efficient algorithm for branching bisimulation and stuttering equivalence.


Algebraic Laws for Nondeterminism and Concurrency.


Verification of CCS-processes.


Communication and Concurrency.


Three Partition Refinement Algorithms.


Decidability of bisimulation equivalence for processes generating context-free languages
[18] Günter Karjoth  
*Definition of I-CRL*  
SPECS Draft MP.WP6.4.03.,

*The syntax and semantics of μCRL*  
SPECS A.WP5.1401-1.,

*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*  

[22] J.A. Bergstra and J.W. Klop  
*Process algebra for synchronous communication*  

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

*Logic and Structure.*  
Springer-Verlag, 1983,

*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.*  

[27] J.F. Groote and F.W. Vaandrager  
*Structured operational semantics and bisimulation as a congruence (extended abstract).*  

*Laws of programming.*  

*CHAPTER 2. BIBLIOGRAPHY*
[29] C.A.R. Hoare
   *Communicating Sequential Processes.*

   *A Calculus of Communicating Systems.*
   LNCS 92, Springer-Verlag, 1980.

[31] S. Mauw and G.J. Veltingk
   *A process specification formalism.*

   *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.*

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