1992

A. Ponse

# Computable processes and bisimulation equivalence

Computer Science/Department of Software Technology Report CS-R9207 January

# Computable Processes and Bisimulation Equivalence

Alban Ponse
*CWI*
*P.O. Box 4079, 1009 AB Amsterdam, The Netherlands*
*e-mail:* alban@cwi.nl

**Abstract**

A process is 'computable' if it can be modelled by a transition system that has a recursive structure. The equivalence relation between transition systems considered is bisimulation. As a means for defining transition systems (modulo bisimulation), the specification language $\mu$CRL (*micro* Common Representation Language) is used. Two simple fragments of $\mu$CRL are singled out, yielding universal expressivity with respect to recursive and primitive recursive transition systems over a fixed, finite label set. For all transition systems defined by a $\mu$CRL specification over the recursive fragment, and for the class of all primitive recursive transition systems over some finite label set, the properties 'bisimilarity', 'deadlock freedom' and 'regularity' (i.e., having a bisimilar finite representation) are classified in the arithmetical hierarchy. Finally it is shown that in the domain of primitive recursive transition systems over a fixed, finite label set, a genuine hierarchy in bisimilarity can be defined via the complexity of the witnessing relations.

*Keywords & Phrases:* Process algebra, ACP, $\mu$CRL, computability.
*1985 Mathematics Subject Classification:* 68Q05, 68Q15, 68Q55, 68Q75.
*1987 Computing Review Categories:* D.3.1, D.3.3, F1.1.
*Note:* This work was supported by the European Communities under RACE project no. 1046, Specification and Programming Environment for Communication Software (SPECS). It represents the view of the author.

## Contents

Report CS-R9207
ISSN 0169-118X
CWI
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

## 1 Introduction

A process is the behaviour of a system, such as a computer or a coffee vending machine, in terms of (observable) actions that can be performed. In this paper transition systems are considered as mathematical representations of processes. Such a transition system consists of a set of states, a set of labels representing the actions, and a transition relation, prescribing for each state the possible 'next steps', i.e., what actions can be performed, and (per action) what state results. Distinguishing one state as 'initial' then yields a formal representation of a process. A process is called *computable* if it can be associated with a transition system that has a recursive structure: the sets of states and labels are recursive, and from each state all next steps can be computed as a finite set. A widely studied behavioural equivalence relation on transition systems is *(strong) bisimulation equivalence* [Par81, Mil89, GV89], which is the semantic equivalence relation considered in this paper.

The purpose of this paper is twofold: first transition systems and bisimulation equivalence are analysed in terms of (basic) recursion theory. Secondly, it is intended to illustrate that the particular specification language $\mu$CRL (*micro* Common Representation Language, [GP90, GP91a]) is a suitable means for the definition of transition systems (modulo bisimulation), and hence for the study of process theory.

The language $\mu$CRL is designed for the specification of (parallel, communicating) processes. This language is a fragment of the specification language CRL [SSA90]. Typically, $\mu$CRL supports only the most basic features of CRL or other specification languages such as LOTOS [ISO87], SDL [CCI87] or PSF [MV90], and is meant to be a vehicle for the study of these basic features. For $\mu$CRL there is a proof theory, providing the mathematical means to analyse behavioural aspects of processes, i.e., to reason about behavioural equivalences [GP91b]. This proof theory is based on an axiomatic approach, in the tradition of the framework ACP (the Algebra of Communicating Processes) developed by BERGSTRA and KLOP (for an overview and historical references see [BW90]). The language $\mu$CRL is just an extension of the ACP set-up in that it allows the specification of data, and that it contains operators for connecting data values to process behaviour. To be more precise: a $\mu$CRL-*specification* instantiates a formal language for defining processes.

Two particular types of transition systems over finite label sets play a role in this paper: "recursive" and "primitive recursive" transition systems. For each of these domains of transition systems, a fragment of $\mu$CRL is described that is universally expressive: any (primitive) recursive transition system can be defined (modulo bisimulation) using a 'canonical' $\mu$CRL-specification over the appropriate fragment.

A 'semantic' property of a transition system modelling some process should be bisimulation invariant, as it is on the level of bisimulation equivalence that transition systems are assumed to model processes. For instance the 'number of states' of a transition system is no such property, as even very simple transition systems can be bisimilar while having a different number of states. The following bisimulation invariant properties of recursive transition systems over a fixed, finite label set are investigated and classified with respect to their defining $\mu$CRL-specifications: *bisimilarity* which is complete in $\Pi^0_1$, *deadlock freedom* which is also complete in $\Pi^0_1$, and *regularity* (i.e., having a bisimilar *finite* representation) which is complete in $\Sigma^0_2$. It is argued that these classification results are also valid with respect to the class of all primitive recursive systems over a fixed, finite label set.

Finally the nature of bisimilarity itself (i.e., the existence of a relation that is a bisimulation) is investigated. It turns out that in the relatively simple domain of primitive recursive transition systems over a fixed, finite label set, one can distinguish between bisimilarity based on primitive recursive, recursive, recursively enumerable or unrestricted witnessing bisimulations.

**Related work.** In [BBK87] BAETEN, BERGSTRA and KLOP show that ACP with 'abstraction' is universally expressive over a comparable domain of recursive transition systems, and that the feature abstraction is really necessary in that case. DARONDEAU uses in [Dar90a] a wider notion of 'recursive transition systems' than is done here (the transition relation must be recursive as a *set* so that infinite branching may occur; see also FERNANDO in [Fer92]), whereby these results do not carry over to the present approach. However, in [Dar90b] DARONDEAU proves that "The maximal reduction of a primitive recursive and deterministic transition graph with *finite degree* and *finite labelling* is not always recursive." In Section 6 there is a result of BERGSTRA [Ber91] implying that it even need not be recursively enumerable.

**Acknowledgements.** I thank Jos Baeten, Jan Bergstra, Javier Blanco, Tim Fernando, Henri Korver, Joachim Parrow, Jan Rutten and Frits Vaandrager for discussions and critical remarks. In particular the observations of Jan Bergstra formed a fundamental inspiration.

## 2 Computable processes

This section introduces transition systems and bisimulation equivalence. Then 'computable' behaviour is defined by means of transition systems having a recursive structure. Finally the bisimulation invariant properties of transition systems that play a role in this paper are introduced.

### 2.1 Preliminaries

A *(rooted, labelled) transition system* is a quadruple $(S, L, \longrightarrow, s_0)$ with

1. $S \neq \emptyset$ a set of *states*,
2. $L \neq \emptyset$ a set of *labels* or *actions*,
3. $\longrightarrow \subseteq S \times L \times S$ a *transition relation*, and
4. $s_0 \in S$ its *root*.

Instead of writing $(s, l, s') \in \longrightarrow$, the more 'pictorial' notation $s \xrightarrow{l} s'$, in accordance with the way transition systems will be visualised (see Example 2.1.2), is used from now on.

In a rooted transition system the root represents the initial state of the process it models. The transition relation then prescribes for each state what actions may be performed (if any) and what state results per possible action. In fact the states of a transition system only play a role in structuring the actions a process may perform. The *operational behaviour* embodied by a transition system is the real object of interest. This behaviour can be captured by regarding transition systems modulo (strong) *bisimulation* equivalence [Par81]:

**Definition 2.1.1** (*Bisimulation equivalence and isomorphisms*). Let $T_1 = (S_1, L_1, \longrightarrow_1, s_1)$ and $T_2 = (S_2, L_2, \longrightarrow_2, s_2)$ be two transition systems. A relation $R \subseteq S_1 \times S_2$ is a *bisimulation* iff $R$ satisfies the *transfer property*, i.e., for each pair $(t_1, t_2) \in R$:

- $t_1 \xrightarrow{l}_1 u_1 \implies \exists u_2 \,.\, t_2 \xrightarrow{l}_2 u_2$ and $(u_1, u_2) \in R$,
- $t_2 \xrightarrow{l}_2 u_2 \implies \exists u_1 \,.\, t_1 \xrightarrow{l}_1 u_1$ and $(u_1, u_2) \in R$.

The transition systems $T_1$ and $T_2$ are *bisimilar*, notation

$$T_1 \leftrightarroweq T_2$$

iff there exists a bisimulation $R \subseteq S_1 \times S_2$ such that $(s_1, s_2) \in R$.

The transition systems $T_1$ and $T_2$ are called *isomorphic*, notation

$$T_1 \simeq T_2$$

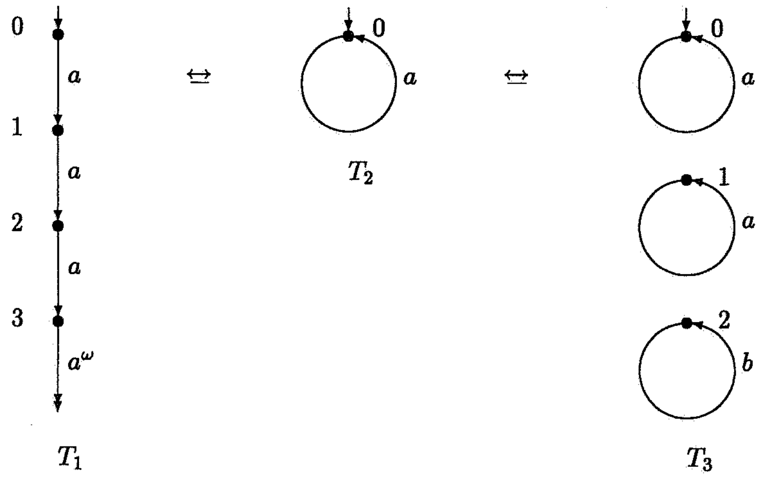
if there is bijective mapping between $S_1$ and $S_2$ that preserves the roots and the respective transition relations. ■

Observe that $\leftrightarroweq$ is an equivalence relation on transition systems, and that isomorphic transition systems are bisimilar.

**Example 2.1.2.** Consider the following three, bisimilar transition systems (*Nat* denoting the natural numbers):

$$\begin{aligned}
T_1 &\overset{def}{=} (Nat, \{a\}, \{n \xrightarrow{a} n+1 \mid n \in Nat\}, 0)\\
T_2 &\overset{def}{=} (\{0\}, \{a\}, \{0 \xrightarrow{a} 0\}, 0)\\
T_3 &\overset{def}{=} (\{0,1,2\}, \{a,b,c\}, \{0 \xrightarrow{a} 0, 1 \xrightarrow{a} 1, 2 \xrightarrow{b} 2\}, 0).
\end{aligned}$$

In the following picture all these transition systems are visualised, where the roots are indicated by a small downward arrow and $\overset{a^\omega}{\twoheadrightarrow}$ abbreviates $\omega$ consecutive $a$-transitions:



It follows easily that $Nat \times \{0\}$ is a bisimulation relating $T_1$ and $T_2$, and $T_1$ and $T_3$. The transition systems $T_2$ and $T_3$ are related by $\{(0,0)\}$. ■

An immediate consequence of regarding transition systems modulo bisimulation equivalence concerns *root connectedness*: only states that can be reached from the root play a role. More formally, given a transition system $T = (S, L, \longrightarrow, s_0)$, let

$$\begin{aligned}
S_0 &\overset{def}{=} \{s_0\},\\
S_{i+1} &\overset{def}{=} \{s' \mid \exists s \in S_i\ \exists l \in L \,.\, s \xrightarrow{l} s'\},\\
S_\omega &\overset{def}{=} \bigcup_{i<\omega} S_i.
\end{aligned}$$

Then $T \leftrightarroweq (S_\omega, L, \longrightarrow \cap (S_\omega \times L \times S_\omega), s_0)$. In a similar way also the set $L$ of labels can be restricted to those labels that occur in transitions from $S_\omega$. There is a sound reason for not defining a transition system right away as a *connected*, directed, labelled graph. In the spirit of a specification language for (equivalence classes) of transition systems, it is usual to define a transition relation via a calculus that operates on language expressions, i.e., on the structure of the states (as to obtain an operational semantics in the style of PLOTKIN [Plo81, GV89]). Therefore the transitions from any state may not depend on properties of the transition system, as for instance root connectedness.

The property root connectedness can be used to define transition systems that distinguish two types of *termination*: 'deadlock' and 'successful termination'.

**Definition 2.1.3** (*Termination in transition systems*). Let $T = (S, L, \longrightarrow, s_0)$ be a transition system such that the distinguished symbol $\surd$ ("tick") is in $L$, and $L \setminus \{\surd\} \neq \emptyset$. The label $\surd$ is used to signal successful termination.

1. A state $s \in S$ is called a *termination* state iff $s$ has no outgoing transitions having a label in $L \setminus \{\surd\}$, i.e.,

$$\longrightarrow \cap (\{s\} \times L \setminus \{\surd\} \times S) = \emptyset,$$

and either $s = s_0$ or there is a root connected state $s' \in S$ and a label $a \in L \setminus \{\surd\}$ such that $s' \xrightarrow{a} s \in \longrightarrow$.

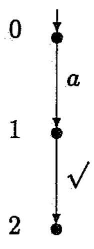
2. A termination state $s \in S$ represents *deadlock* iff $s$ has no outgoing transitions.
3. A termination state $s \in S$ represents *successful termination* iff $s \neq s_0$, there is *one* outgoing transition of the form

$$s \xrightarrow{\surd} s' \in \longrightarrow,$$

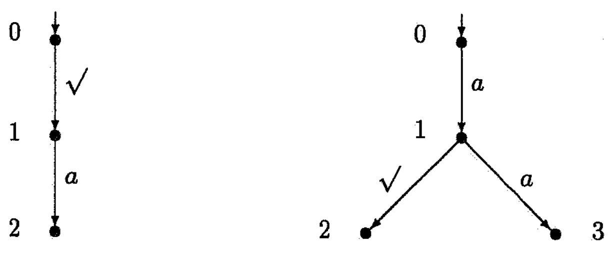
and for this transition the resulting state $s'$ has no outgoing transitions.

The transition system $T$ is *properly terminating* ($PT$) iff for all $s, s' \in S$ that are root connected and that satisfy $s \xrightarrow{\surd} s' \in \longrightarrow$, the state $s$ is a termination state, and every termination state in $T$ represents either deadlock or successful termination. ■

Some examples:



A $PT$ transition system



Two transition systems that are not $PT$

## 2.2 Recursive transition systems

Following BERGSTRA and KLOP [BBK87], a 'computable' process is a process of which in any state all possible next steps are finite in number and can be computed. Such a process is modelled by a 'recursive' transition system. [1] In the formal definition of such transition systems the following standard primitive recursive (de)coding functions relating $Nat \times Nat$ and $Nat$ [Dav82] are used:

$$\begin{aligned} j(x,y) &= \tfrac{1}{2} \cdot ((x+y)^2 + 3y + x) \\ j_1(x) &= \mu y \leq x \,.\, [\exists z \leq x \,.\, j(y,z) = x] \\ j_2(x) &= \mu z \leq x \,.\, [\exists y \leq x \,.\, j(y,z) = x]. \end{aligned}$$

Typically $j(j_1(x), j_2(x)) = x$ and $j_i(j(x_1, x_2)) = x_i$. Moreover, a useful property is $x \leq j(x,y) \geq y$.

[1]Some common references to recursion theory are [Rog67, Dav82].

**Definition 2.2.1.** A transition system $(S, L, \longrightarrow, s_0)$ is *recursive* iff

1. The set $S \subseteq Nat$ of states is a recursive set.
2. The set $L$ of labels can be coded as a recursive set, i.e., there is an injective function $i : L \rightarrow Nat$ such that

   $$i(L) \overset{def}{=} \{i(l) \mid l \in L\}$$

   is recursive.
3. The relation $\longrightarrow$ can be represented by a (total) recursive function *next* such that for all $s \in S$ the value of $next(s)$ is the canonical index[2] (CI) of the finite set coding all next steps from $s$:

   $$next(s) = \mathrm{CI}(\{j(i(l), s') \mid s \xrightarrow{l} s'\})$$

   (so $D_{next(s)}$ contains all values $j(i(l), s')$ for which $s \xrightarrow{l} s'$).
4. The root $s_0$ is 0.

If appropriate, $(S, L, \longrightarrow, s_0)$ is sometimes denoted as

$$(S, L, next, i).$$

A transition system is *primitive recursive*, PRIM for short, iff the sets $S$ and $i(L)$, and the function *next* are PRIM. ■

In the case that $L$ is (isomorphic with) $Nat$, it is assumed that the coding function $i$ itself is recursive. Note that in a recursive transition system the number of next steps is always finite: such systems are called *finitely branching*.

An alternative approach would be to define a recursive transition system as one that may be countably branching and has a recursive structure. In [Dar90a] this approach is followed, and there a transition system is "recursive" iff its transition relation is a recursive *set*.

A transition system $(S, L, \rightarrow, s_0)$ is called *finite* iff both $S$ and $L$ are finite (so a finite transition system is always isomorphic with a PRIM transition system). Finite transition systems are of interest, because they are easy comprehensible and immediately reveal a simple type of behaviour ('regular' behaviour). The question whether a (primitive) recursive transition system is bisimilar to a finite one is highly undecidable. This question is further dealt with in Section 5.

Any (primitive) recursive transition system can be unfolded into a bisimilar one that is *tree-like*, i.e., of which the root connected part is a tree (cf. Example 2.1.2, where $T_1$ is an unfolded version of $T_2$). This can be done by the following transformation. Take $Nat$ as the set of states of the system to be defined. Given a recursive transition system

$$T = (S, L, next, i),$$

the idea is to define a new function *tree-next*(.) on $Nat$ that defines transitions $s \xrightarrow{l} s'$ such that $s < s'$ and such that all root connected states contain a reference to *next* that guarantees bisimilarity. Assume a state $s$ of $T$ is related to a state $j(t, s)$ of the system under construction (initially the root 0 is related to $j(0, 0) = 0$), and in $T$ there is a transition

$$s \xrightarrow{l} s'.$$

[2]The canonical index of $\emptyset$ is 0, of $\{k_1, k_2, ..., k_l\}$ it is the number $2^{k_1} + 2^{k_2} + ... + 2^{k_l}$, and $D_x$ is the finite set with canonical index $x$.

Now take as corresponding transition

$$j(t,s) \xrightarrow{l} j(f(t,s,i(l)),s')$$

with $f$ a PRIM function satisfying $\forall x_1, x_2, x_3, y.\ j(x_1,x_2) < j(f(x_1,x_2,x_3),y)$. Due to this inequality, it follows that these new transitions never generate cycles. The $j_2$ values of the new states can be used to determine (using *next*) what transitions are defined in the original system $T$, whence guaranteeing bisimilarity. Now a formal definition of this new system, fixing $f$, can be given. Let

$$C[y \overset{?}{\in} D_x]$$

be the (PRIM) characteristic function that yields 1 in case of membership, and 0 otherwise. Then $T' = (Nat, L, \mathit{tree\text{-}next}, i)$ with

$$\mathit{tree\text{-}next}(x) \overset{def}{=} \sum_{y \leq \mathit{next}(j_2(x))} C[y \overset{?}{\in} D_{\mathit{next}(j_2(x))}] \cdot 2^{j(j_1(y), j(j_1(y)+x+1, j_2(y)))}$$

is a recursive tree-like transition system yielding $T \leftrightarrow T'$ by definition.

Observe that in the transformation described above, the function *tree-next* is PRIM if *next* is. Hence transforming a *primitive* recursive transition system in this way yields a bisimilar tree-like transition system that is again PRIM.

## 2.3 Bisimulation invariant properties

The properties of recursive transition systems over a fixed, finite label set that play a role in this paper are now introduced. All of these have to be invariant under bisimulation equivalence, as it are such equivalence classes that represent 'operational behaviour'. So, if $T_0, ..., T_n$ are transition systems for which property $\mathcal{P}$ holds, say $\mathcal{P}(T_0, ..., T_n)$, and $T_i \leftrightarrow T_i'$, then also $\mathcal{P}(T_0', ..., T_n')$. Typically, neither 'number of states', nor 'being tree-like' is a bisimulation invariant property of a single transition system (even not for the restriction to root connected transition systems), as was illustrated in Example 2.1.2.

The first property that will be considered is *bisimilarity* itself (because $\leftrightarrow$ is an equivalence relation, it is a property over two transition systems that is bisimulation invariant). This property is of interest 'by definition': it characterises all of what is taken to be important of a transition system.

A second property of interest concerns properly terminating ($PT$) transition systems (see Definition 2.1.3). A $PT$ transition system is *deadlock free* if all termination states represent successful termination. Obviously deadlock freedom is invariant under bisimilarity. The interest of this property can be motivated as follows. In $\mu$CRL, concurrent processes are defined using parallel operators and communication declarations. The remnants of unsuccessful communications are then encapsulated: the corresponding transitions are removed [BW90]. If at some point there is no communication possible, this causes a deadlock.

The property that goes with finiteness modulo bisimulation is *regularity*: a transition system is regular iff it is bisimilar with some finite transition system. The term 'regular' refers to the theory of formal languages [HU79], from which also standard techniques can be used to prove that a transition system is not regular. In particular, the presence of an "irregular trace" contradicts the Pumping Theorem for regular languages [HU79]. As noted before, regular transition systems are of interest because they are 'finitely representable' (cf. Example 2.1.2).

For a tree-like transition system, let its $n_{th}$ *projection* be its (rooted) subtree of depth $n$. So modulo bisimulation, the $n_{th}$ projection of a recursive transition system is a well defined notion modelling the first $n$ steps that can be performed. Projections play a crucial role in the way properties of recursive transition systems are classified in Section 5. For instance, two recursive transition systems are bisimilar if all their projections are.

## 3 The language $\mu$CRL, two simple fragments

This section introduces the fragments of the specification language $\mu$CRL that are used to specify (primitive) recursive transition systems over a fixed, finite label set (modulo bisimulation). Any closed process term is associated with such a (primitive) recursive transition system.

The language $\mu$CRL [GP90, GP91a] can be characterised as an extension of the process specification language ACP [BW90] with data. A (well-formed) $\mu$CRL *specification* may contain the declaration of abstract data types and defines a process specification environment in which such data may be used to specify data dependent process behaviour. This paper concentrates on some fragments of *effective* $\mu$CRL, the adjective referring to the effective computability of associated notions. Before explaining these notions, it makes sense to impose the following general restrictions, adopted throughout the paper:

1. There are no *parameterised* actions (this corresponds with the restriction to recursive transition systems over *finite* label sets).
2. With regard to data, only (total) recursive functions and relations are considered (or sometimes even only primitive recursive ones).
3. Only three basic $\mu$CRL process operators are used: choice, sequential composition and the conditional (this operator resembles the programming construct **if - then - else -**).

Compared to the common system ACP, the additional feature of the $\mu$CRL-fragments considered is that process identifiers (used to specify processes recursively) may be parameterised with conceptually the most simple data type, viz. the natural numbers on which only total recursive operations are allowed, [3] plus the availability of the conditional by which process behaviour can be specified as depending on data values. Therefore it is so to speak the smallest extension of the ACP framework (discarding parallelism and abstraction) in the direction of 'involving data in specifications', i.e., in the direction of $\mu$CRL. It turns out that with respect to expressivity and some computational properties of associated transition systems, this fragment constitutes already an interesting extension.

### 3.1 Syntax of $\mu$CRL(TREC) and $\mu$CRL(PRIM)

A (well-formed) $\mu$CRL specification contains a finite number of two types of declaration units: one constituting the 'data part' of the specification, and one forming the 'process part'. In the following these are first described separately. As there is in this paper only a restricted use of the language (especially concerning parameterisation), the syntax given here is simpler than in [GP90].

**The data part.** Only two data sorts play a role: the Booleans of which the sort name **Bool** and the constants **T** (true) and **F** (false) must be declared in any (well-formed) specification; and the natural numbers, represented as the sort *Nat* with constant 0 and successor function $S$. Furthermore a specification may contain a finite number of 'function declarations', provided these are total recursive (TREC for short). As an example of the syntax of data specification in $\mu$CRL, the specification of some familiar functions is shown in Table 1, where the keyword **rew** ('rewriting rules') precedes the actual definitions of the functions (using the variables declared by **var**).

Let for the remainder of this paper a coding of Turing Machines (or any other equivalent computing device) be fixed. In particular KLEENE'S well-known *T-predicate* [Kle52, Dav82] plays a crucial role in the example specifications to come: let $m \geq 1 \in Nat$, then

$$T_m(x, y_1, ..., y_m, z)$$

[3]Parameterisation of process identifiers in ACP with a *finite* data type is not considered as an extension: it can always be mimicked by using all 'instantiated' process identifiers.

| | |
|---|---|
| **sort** | **Bool** |
| **func** | $\mathbf{T}, \mathbf{F} :\rightarrow \mathbf{Bool}$ |
| | |
| **sort** | $Nat$ |
| **func** | $0 :\rightarrow Nat$ |
| | $S, pd : Nat \rightarrow Nat$ |
| | $add, monus, times : Nat \times Nat \rightarrow Nat$ |
| **var** | $x, y : Nat$ |
| **rew** | $pd(0) = 0$ |
| | $pd(S(x)) = x$ |
| | $add(x, 0) = x$ |
| | $add(x, S(y)) = S(add(x, y))$ |
| | $monus(x, 0) = x$ |
| | $monus(x, S(y)) = pd(monus(x, y))$ |
| | $times(x, 0) = 0$ |
| | $times(x, S(y)) = add(times(x, y), x)$ |
| | |
| **func** | $eq : Nat \times Nat \rightarrow \mathbf{Bool}$ |
| **var** | $x, y : Nat$ |
| **rew** | $eq(x, x) = \mathbf{T}$ |
| | $eq(S(x), S(y)) = eq(x, y)$ |
| | $eq(S(x), 0) = \mathbf{F}$ |
| | $eq(0, S(x)) = \mathbf{F}$ |

Table 1: A data specification.

holds if $z$ codes a computation according to the Turing Machine coded by $x$ for arguments $(y_1, ..., y_m)$. It should be stressed that once such a coding is defined, the predicate $T_m$ (for a fixed value $m$) is PRIM. Consequently, $T_m(x, y_1, ..., y_m, z)$ can be defined in a specification by a Boolean valued characteristic function.

**The process part.** The most simple processes are *(atomic) actions.* These must be declared explicitly using the keyword **act**. For example

**act** $a, b$

is a declaration of actions with names $a, b$.

More complex processes can be declared by means of (parameterised) process *identifiers*, possibly in a recursive way. Such declarations must be preceded by the keyword **proc**. For example

**proc** $counter(x) = p$
$buffer = q$

In the first line a counter is declared. It is a process with one parameter $x$ of sort $Nat$. The parameter $x$ may be used in the *process term* $p$ that specifies its behaviour and has no wider scope. The syntax of process terms is defined below. In the second line of the example a parameterless process *buffer* is declared. Its behaviour is given by the process term $q$. In this paper all process declarations are either not parameterised, or parameterised over $Nat$ (so the sort of the variables possibly occurring in process identifiers is always $Nat$). [4]

[4]In full $\mu$CRL typing of data parameters in process declarations is necessary.

Process terms may be constructed according to the following syntax:

$$\begin{array}{rcl} p & ::= & (p+p) \\ & | & (p \cdot p) \\ & | & (p \lhd t \rhd p) \\ & | & \delta \\ & | & n \\ & | & n(t_1, ..., t_m) \end{array}$$

Here the $+$ represents choice and the $\cdot$ stands for sequential composition. The *conditional* construct $p \lhd t \rhd p$ is an alternative way to write an **if - then - else** expression introduced by HOARE *et al.* [HHJ$^+$87] (see also [BB90]). The data-term $t$ is supposed to be of the standard sort of the Booleans (**Bool**). The $\lhd$-part is executed if the data-term $t$ evaluates to true (**T**) and the $\rhd$-part is executed if the $t$ evaluates to false (**F**). Furthermore $\delta$ is a constant called *deadlock* or *inaction*, and represents the situation in which no steps can be performed. Finally $n$ is the name of some declared action or process identifier, and $t_1, ..., t_m$ are data terms.

**Specifications in $\mu$CRL(TREC) and $\mu$CRL(PRIM).** A *specification* over the fragment of $\mu$CRL used here is a sequence starting with the declaration given in Table 1, possibly followed by a number of other function declarations, action declarations and process declarations.

In effective $\mu$CRL, specifications have to be *guarded* as to safeguard that any process term is associated to a transition system that is recursive. Guardedness is a condition on the way recursion may be used in process declarations. Typically, unguarded specifications may represent transition systems that are infinitely branching. The following criterion for *syntactic guardedness* (a sort of 'primitive recursion scheme') is further used. It is based on the syntax of specifications, and it is a decidable property. (In Section 3.2 a much wider, but undecidable criterion for guardedness is discussed.)

**Definition 3.1.1** (*Syntactic guardedness*). Let $E$ be a specification that contains the data part introduced above.

1. Let $p, q$ be process terms over $E$ with $p$ a (parameterised) process identifier, then $p$ *is syntactically guarded in* $q$ iff one of the following conditions is satisfied:
   - $q \equiv (q_1 + q_2)$ or $q \equiv (q_1 \lhd t \rhd q_2)$, and $p$ is syntactically guarded in $q_1$ and $q_2$,
   - $p \equiv n(..., x, ...)$ and $q \equiv (r \lhd eq(x, 0) = \mathbf{T} \rhd n(..., monus(x, 1), ...))$, and $p$ is syntactically guarded in $r$,
   - $q \equiv (q_1 \cdot q_2)$, and $p$ is syntactically guarded in $q_1$,
   - $q$ is any action or $\delta$.
2. The specification $E$ is *syntactically guarded* iff in each of its process declarations the left-hand side (the process identifier) is syntactically guarded in the right-hand side (the 'body').

■

Now the fragments of effective $\mu$CRL that play a role in this paper can be defined. Given a finite set of actions (labels), these fragments turn out to have universal expressivity with respect to the class of recursive and primitive recursive transition systems over that label set.

**Definition 3.1.2.** A specification $E$ belongs to $\mu$CRL(TREC) (respectively $\mu$CRL(PRIM)) iff

- $E$ contains the declarations given in Table 1,
- all functions in $E$ are total recursive (primitive recursive, respectively),
- $E$ is syntactically guarded.

■

**Informal abbreviation of specifications.** In the sequel specifications are abbreviated by describing only the occurring process declarations, and even these in an informal way. As an example consider the specification $E$ of the form given in Table 2.

| | |
|---|---|
| The contents of Table 1<br>... some more function declarations ... | } data part of $E$ |
| **act** $a\ b$<br>**proc** $K(x,y,z) = \mathit{body}$ | } process part of $E$ |

Table 2: A $\mu$CRL(TREC) specification $E$.

In the sequel $E$ will be described by

$$K(x,y,z) = \mathit{body}.$$

It is then assumed that any data function occurring in *body* is familiar to the reader and declared in the data part of $E$. A further convention is to use $a, b, c, ...$ as identifiers for actions, plus the assumption that all actions occurring in *body* are declared in $E$.

In process terms (as *body*), brackets are omitted according to the convention that $\cdot$ binds stronger than $.\!\lhd t \rhd\!.$ (regarding $.\!\lhd t \rhd\!.$ as a binary process operator for any closed data term $t$ over the Booleans), which in turn binds stronger than $+$, and that these operators associate to the right.

In a harmless way, the restriction to syntactic guardedness is relaxed in favour of readability. For example the specification

$$\begin{array}{rcl} A & = & P(f(3)) \\ P(x) & = & Q(f(x), g(x)) \\ Q(x,y) & = & a \cdot P(g(x)) \end{array}$$

informally introduces process identifiers $A$ and $P(x)$. In this case the declaration of $P(x)$ serves as a readable abbreviation for the less readable formal declaration of $Q(x,y)$, which is then the following:

$$Q(x,y) = a \cdot Q(f(g(x)), g(g(x)))$$

obtained by straightforward syntactic substitution, and the process identifier $A$ is used to abbreviate the process term $Q(f(f(3)), g(f(3)))$.

With respect to the data used in specifications, the following conventions are adopted. For the binary functions declared in Table 1 infix notation and standard symbols will be further used: $t + u$, $t \dot{-} u$ and $t \cdot u$ for $\mathit{add}(t,u)$, $\mathit{monus}(t,u)$ and $\mathit{times}(t,u)$, respectively. Furthermore $t = u$ and $t \neq u$ are used as abbreviations for $\mathit{eq}(t,u) = \mathbf{T}$ and $\mathit{eq}(t,u) = \mathbf{F}$, respectively. Also the Boolean standard functions $\neg, \wedge, \vee$ are often used. To increase readability, brackets are omitted when possible and infix notation will be used. Letters $v, w, x, y, z, ...$ are reserved for variables declared over *Nat*, and the letters $k, l, m, n, ...$ range over numerals. Finally, the domain of $T_m$ occurring in an (informal) specification is given by context (number of arguments) and the subscript $m$ is omitted.

## 3.2 Semantics of $\mu$CRL(TREC) and $\mu$CRL(PRIM)

The data part of any specification over $\mu$CRL(TREC) is interpreted in the canonical term algebra over the domains $D(\mathit{Nat}) = \{0, S(0), ...\}$ and $D(\mathbf{Bool}) = \{\mathbf{T}, \mathbf{F}\}$. So any function declared is regarded as yielding the usual normal forms in the appropriate domain.

The operational semantics of $\mu$CRL(TREC) is given by an interpretation function *SOS* (Structured Operational Semantics) that, if instantiated with (the signature of) some $\mu$CRL(TREC) specification $E$, assigns to each closed process term over $E$ a transition system. Thus

$$SOS : \mu\text{CRL(TREC)} \rightarrow (\mathbb{P}(.) \rightarrow TS[\mathbb{P}(.) \cup \{\surd\}\ ,\ \mathbb{A}(.) \cup \{\surd\}]$$

where for a specification $E$ over $\mu$CRL(TREC),

- $\mathbb{P}(E)$ is the set of *closed* process terms over $E$,
- $\mathbb{A}(E) \subseteq \mathbb{P}(E)$ is the set of actions declared in $E$,
- the expression $TS[\mathbb{P}(E) \cup \{\surd\}\ ,\ \mathbb{A}(E) \cup \{\surd\}]$ abbreviates the domain of recursive transition systems over states $\mathbb{P}(E) \cup \{\surd\}$ and labels $\mathbb{A}(E) \cup \{\surd\}$, where $\surd$ is used to express successful termination (cf. Definition 2.1.3). Note that $\mathbb{P}(E)$ is denumerably infinite, as $\delta, \delta+\delta, ..., \delta\cdot\delta, ... \in \mathbb{P}(E)$.

For each closed process term $p \in \mathbb{P}(E)$, the transition system $SOS(E)(p)$ is defined by:

$$(\mathbb{P}(E) \cup \{\surd\}\ ,\ \mathbb{A}(E) \cup \{\surd\}\ ,\ \longrightarrow_E\ ,\ p)$$

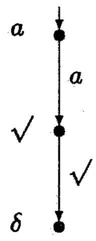
where the transition relation $\longrightarrow_E$ , further abbreviated by $\longrightarrow$, is provided by the calculus given in Table 3 and the following three rules:

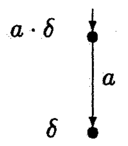
| | | |
|---|---|---|
| $\mathbb{A}(E)$ | $a \xrightarrow{a} \surd$ | $\surd \xrightarrow{\surd} \delta$ |
| $+$ | $\dfrac{x \xrightarrow{a} x'}{x+y \xrightarrow{a} x'}$ | $\dfrac{y \xrightarrow{a} y'}{x+y \xrightarrow{a} y'}$ |
| $\cdot$ | $\dfrac{x \xrightarrow{a} x'}{x\cdot y \xrightarrow{a} x'\cdot y}$ if $x' \not\equiv \surd$ | $\dfrac{x \xrightarrow{a} \surd}{x\cdot y \xrightarrow{a} y}$ |
| $\lhd\, t \,\rhd$ | $\dfrac{x \xrightarrow{a} x'}{x \lhd t \rhd y \xrightarrow{a} x'}$ if $t = \mathbf{T}$ | $\dfrac{y \xrightarrow{a} y'}{x \lhd t \rhd y \xrightarrow{a} y'}$ if $t = \mathbf{F}$ |
| recursion | $\dfrac{x \xrightarrow{a} x'}{y \xrightarrow{a} x'}$ if $y = x$ refers to a *process instantiation in E* | |

Table 3: Transition rules for a $\mu$CRL(TREC) specification $E$.

1. The variables $x, y$ range over $\mathbb{P}(E)$ and the primed variables $x', y'$ over $\mathbb{P}(E) \cup \{\surd\}$,
2. In the rule introducing $\lhd\, t \,\rhd$, the side conditions $t = \mathbf{T}$ and $t = \mathbf{F}$ refer to the normal form of $t$,
3. In the recursion rule a *process instantiation in E* is a closed instance of a process declaration in $E$, with all data parameters in the right-hand side represented by their associated normal forms (numerals).

Typically, the root connected parts of the transition systems associated to an action $a \in \mathbb{A}(E)$ and the term $a \cdot \delta \in \mathbb{P}(E)$ can be visualised as follows (observe that the right-hand side transition system is not deadlock free):





As another example, assume that $E$ contains the process declaration $K(x) = a \cdot K(x+1)$. Using $a \xrightarrow{a} \surd$ and the second rule for sequential composition it follows that $a \cdot K(1) \xrightarrow{a} K(1)$, and using the process instantiation $K(0) = a \cdot K(1)$, application of the recursion rule yields $K(0) \xrightarrow{a} K(1)$. Similarly, $K(0+0) \xrightarrow{a} K(1)$.

The specific operational semantics *SOS* introduced here is of a referential nature: any such semantics yielding transition systems that are (strongly) bisimilar with the *SOS* ones will be of further interest. More on this in the next section.

The restriction to μCRL(TREC) now implies that *SOS* yields (isomorphic images of) *recursive* transition systems. (In fact this holds for the standard operational semantics for *effective* μCRL, of which *SOS* is the restriction to the particular μCRL(TREC) syntax [GP90]):

**Theorem 3.2.1.** *Let $p$ be some process specified over a specification $E$ in* μCRL(TREC), *then $SOS(E)(p)$ is isomorphic with a recursive transition system. Moreover, this transition system is properly terminating (see Definition 2.1.3).*

**Proof.** (Sketch) First note that $SOS(E)(p)$ is finitely branching: by syntactic guardedness any closed process term $q$ can be expanded (using the declarations in $E$) into a term for which the next steps do not depend on terms headed by process identifiers. It follows from the calculus for $\longrightarrow_E$ that only finitely many next steps from $q$ can be derived.

Secondly, both the set of (syntactic well-formed) closed data terms and the set $\mathbb{P}(E) \cup \{\surd\}$ can be coded as recursive sets (even as PRIM sets), where the latter coding has the property that the code of a term is larger than those of its proper subterms, and that 0 is not in its range. Write

$$\ulcorner . \urcorner : \mathbb{P}(E) \cup \{\surd\} \to Nat \setminus \{0\}$$

for this coding. Using the calculus for $\longrightarrow_E$, define a TREC function $next'(.)$ that computes the next steps of any code of a closed term in the style of Definition 2.2.1 (yielding some CI). The function $next'(.)$ is TREC as it must be able to evaluate the TREC functions defined in $E$. Now given $p$ as in the theorem, define a coding

$$\llcorner . \lrcorner : \mathbb{P}(E) \cup \{\surd\} \to Nat$$

by $\llcorner p \lrcorner = 0$ and $\llcorner q \lrcorner = \ulcorner q \urcorner$ for $q \not\equiv p$. Adjusting $next'(.)$ to $next(.)$ by taking the difference between $\ulcorner . \urcorner$ and $\llcorner . \lrcorner$ into account, it then follows that

$$SOS(E)(p) \simeq (\llcorner \mathbb{P}(E) \cup \{\surd\} \lrcorner, \mathbb{A}(E) \cup \{\surd\}, next, i)$$

with $i$ defined as $\llcorner . \lrcorner$ on the appropriate subdomain $\mathbb{A}(E) \cup \{\surd\}$.

Finally, it follows immediately from the rules of $\longrightarrow_E$ that both these transition systems are properly terminating. ■

**Corollary 3.2.2.** *Let $p$ be some process specified over a specification $E$ in $\mu$CRL(PRIM), then $SOS(E)(p)$ is isomorphic with a PRIM transition system.*

Another important (algebraic) property of $SOS$ is that the bisimilarity induced by it is a congruence with respect to the process operators of $\mu$CRL(TREC) [GP90].

As to illustrate the claim that $\mu$CRL(TREC) provides an elegant mechanism for the specification of transition systems, consider the following example:

**Example 3.2.3.** Let the specification $E$ be defined by

$$K(x,y) \quad = \quad a \cdot K(y,x) \triangleleft x = 0 \triangleright b \cdot K(x \dot{-} 1, y+2)$$

so that $SOS(E)(K(0,0))$ consists of the infinite trace $a^\omega$ (its root connected part is isomorphic with the transition system $T_2$ from Example 2.1.2). Hence $SOS(E)(K(0,0))$ is regular.

In the case that $n+m>0$, the transition system $SOS(E)(K(n,m))$ has the infinite trace (starting from the root)

$$b^n \cdot a \cdot b^{2n+m} \cdot a \cdot b^{2(2n+m)} \cdot a \cdot \ldots \cdot b^{k(2n+m)} \cdot a \cdot \ldots$$

(omitting $b^n$ if $n=0$). Recall that a transition system is 'regular' if it is bisimilar with a finite transition system. It follows that $SOS(E)(K(n,m))$ is regular iff $n=m=0$. (The assumption that if $n+m>0$, there is a finite automaton that accepts all prefixes of the latter, infinite trace contradicts the 'Pumping Theorem' for regular languages [HU79].) ■

This section is concluded with some reflections on guardedness. The motivation for (syntactic) guardedness is made explicit in the following example.

**Example 3.2.4.** Consider the specification $E$ given by

$$X = X \cdot a + a.$$

Then $SOS(E)(X \cdot a)$ has an $a$-transition to $SOS(E)(a^k)$ for any $k \geq 1$, as can be seen from the following derivations for the cases $k=1$ and $k=2$:

$$\begin{array}{c} a \xrightarrow{a} \surd \\ \hline X \cdot a + a \xrightarrow{a} \surd \\ \hline X \xrightarrow{a} \surd \\ \hline X \cdot a \xrightarrow{a} a \end{array} \quad \text{and using this result derive} \quad \begin{array}{c} X \cdot a \xrightarrow{a} a \\ \hline X \cdot a + a \xrightarrow{a} a \\ \hline X \xrightarrow{a} a \\ \hline X \cdot a \xrightarrow{a} a \cdot a \end{array}$$

With the conclusion of the second derivation it can be derived that

$$X \cdot a \xrightarrow{a} a^3$$

and so on and so forth. So $SOS(E)(X \cdot a)$ is *infinitely* branching, and hence not recursive. ■

The most general definition of 'guardedness' guaranteeing effective behaviour is in the case of $\mu$CRL(TREC) (or $\mu$CRL(PRIM)) the following.[5]

**Definition 3.2.5** (*Guardedness*)**.** Let $E$ be a specification with data declaration as in $\mu$CRL(TREC) (or as in $\mu$CRL(PRIM)) that is not necessarily syntactically guarded.

1. Let $p, q \in \mathbb{P}(E)$ and $p$ be a process identifier. Then $p$ is *guarded* in $q$ iff one of the following conditions is satisfied:

   - $q \equiv q_1 + q_2$, and $p$ is guarded in $q_1$ and $q_2$,

[5]In [GP90] guardedness is defined relative to the algebraic semantics, and, moreover, involves all the process operators from $\mu$CRL.

- $q \equiv q_1 \lhd t \rhd q_2$ and either $c = \mathbf{T}$ and $p$ is guarded in $q_1$, or $c = \mathbf{F}$ and $p$ is guarded in $q_2$,
- $q \equiv q_1 \cdot q_2$, and $p$ is guarded in $q_1$,
- $q$ is any action or $\delta$,
- $q$ is a process identifier different from $p$, that is, there must be a semantical difference: eg. $p(k)$ and $p(k+0)$ are not considered to be different.

2. The *Process Name Dependency Graph* of $E$, notation

   $$PNDG(E)$$

   is the graph that has as its node set all (closed) process identifiers defined in $E$ and an arc from $p_1$ to $p_2$ iff $p_1 = q$ is a closed instance of a process declaration in $E$ and $p_2$ is not guarded in $q$.

3. The specification $E$ is *guarded* iff $PNDG(E)$ is well founded, i.e., does not contain an infinite path.

■

This means that a specification $E$ is guarded iff *for any* node in $PNDG(E)$ *there exists* an upper bound to the length of its paths. This suggests that the question whether $E$ is guarded is undecidable. Indeed there is the following result:

**Lemma 3.2.6.** *In* μCRL *with data declaration restricted as in* μCRL(PRIM), *guardedness is complete in* $\Pi^0_2$.

**Proof.** Define a specification $E$ such that for fixed $k, l$ the graph $PNDG(E)$ has for process identifiers $K(m,n,n')$ the following properties: there is an arc

$$K(m,n+1,n') \longrightarrow K(m,n,n'+1) \qquad \text{iff} \qquad \neg T(k,l,m,n+1),$$

and an arc

$$K(m,0,n') \longrightarrow K(m,n'+1,0)$$

so that the question of the well foundedness of $PNDG(E)$, and hence of the guardedness of $E$, is related to satisfaction of the $T$-predicate. Let $E$ be defined as follows:

$$K(x,y,z) \quad = \quad K(x,z+1,0) \lhd y=0 \rhd \big(a \cdot K(x+1,0,0) \lhd T(k,l,x,y) \rhd K(x, y \dot{-} 1, z+1)\big).$$

The variable $z$ only has a 'local' purpose in the systematic testing of $T(k,l,x,y)$ that goes on for a particular value of $x$ as long as the $T$-predicate is not satisfied: it induces testing down from consecutively larger values than the initial value of $y$. So, for any node $K(m,n,n') \in PNDG(E)$ there is an infinite branch iff $\forall y \,.\, \neg T(k,l,m,y)$ holds.

Hence the specification $E$ is guarded iff for all $m$ (instantiating $x$) there is an $n$ (the $y$ instance) such that $T(k,l,m,n)$. The upper bound of the path length from any node in $PNDG(E)$ can then be computed from the initial $K$ values in that node.

$$\text{So } E \text{ is guarded} \quad \Longleftrightarrow \quad \forall x \exists y \,.\, T(k,l,x,y).$$

The right-hand side is a standard condition for completeness in $\Pi^0_2$ (for references see Section 5). Though in this specification $k, l$ are fixed, and the question of the latter condition can be very simple in some particular cases, there is no *algorithmic* procedure to check whether it holds. Hence, the property of a μCRL(PRIM) specification being guarded is complete in $\Pi^0_2$. ■

This motivates the restriction to *syntactic* guardedness as defined in 3.1.1, rather than to guardedness as such. In the next section it is shown that syntactic guardedness is not a restriction in terms of expressivity (modulo strong bisimulation).

# 4 Symbolic implementations and expressivity

In this section the relation between recursive transition systems over a fixed, finite label set and $\mu$CRL(TREC) as a language for specifying these (modulo bisimulation) is further investigated. It is shown that the selected fragments of $\mu$CRL have universal expressivity over the two selected domains of transition systems. However, the distinction between successful termination and deadlock introduces a subtlety that is somewhat difficult. Finally, it is proved that modulo bisimulation these two domains (over a fixed, finite label set) are different.

## 4.1 Symbolic implementations

A transition system $T$ is a *symbolic implementation* of a $\mu$CRL(TREC) process term $p$ over some $\mu$CRL(TREC) specification $E$ iff

$$T \leftrightarrow SOS(E)(p)$$

where $SOS$ is the function defined in Section 3.2. Conversely, the process term $p$ (and its defining specification $E$) is said to *specify* the transition system $T$, in this case. Any function that assigns symbolic implementations to closed process terms specified over $\mu$CRL(TREC) is further referred to as a *symbolic operational semantics.*

An important question is whether $SOS$ itself is 'nice enough' as a symbolic operational semantics. Are there no such semantics that yield simpler transition systems, in particular 'minimal' transition systems? This is not even the case for the restriction to $\mu$CRL(PRIM):

**Theorem 4.1.1.** *Any symbolic operational semantics for* $\mu$CRL(PRIM) *yields too large transition systems.*

**Proof.** Consider for example the $\mu$CRL(PRIM) specification $E$ defined by

$$K(x,y,z) = a \cdot K(x,y,z+1) \triangleleft T(x,y,z) \triangleright b \cdot K(x,y,z+1).$$

Let $k,l$ be fixed. First assume $T(k,l,m)$ for some $m$. In this case consider the transition system

$$T_0 \overset{def}{=} (\{0,1,...,m+1\},\{a,b\},Tr,0)$$

with the transition relation

$$Tr \overset{def}{=} \{x \xrightarrow{a} x+1 \mid x<m\} \cup \{m \xrightarrow{b} m+1\} \cup \{m+1 \xrightarrow{a} m+1\}.$$

Then $T_0$ is a finite symbolic implementation of $K(k,l,0)$, and is certainly 'smaller' than $SOS(E)(K(k,l,0))$. In the case that $T(k,l,m)$ holds, the root connected part of $SOS(E)(K(k,l,0))$ and $T_0$ are depicted in Figure 1. In the case that $\neg\exists z\,.\,T(k,l,z)$, the transition system $T_2$ defined in Example 2.1.2 would be a minimal symbolic implementation of $K(k,l,0)$ (and $SOS(E)(K(k,l,0))$ would resemble $T_1$ in that example).

Hence for each pair $k,l$ it follows that $SOS(E)(K(k,l,0))$ is regular. This example shows that a symbolic operational semantics yielding transition systems with minimal sets of states (and labels) must be able to decide for each $k,l$ whether $\exists z\,.T(k,l,z)$, so must be able to solve the halting problem. ■

## 4.2 Expressivity

It is shown that any (primitive) recursive transition system over a finite label set can be represented by a $\mu$CRL(TREC) process term (respectively a process specified over $\mu$CRL(PRIM)). The first result excludes the successful termination label $\surd$ (see Definition 2.1.3) and is on transition systems having one type of termination states.

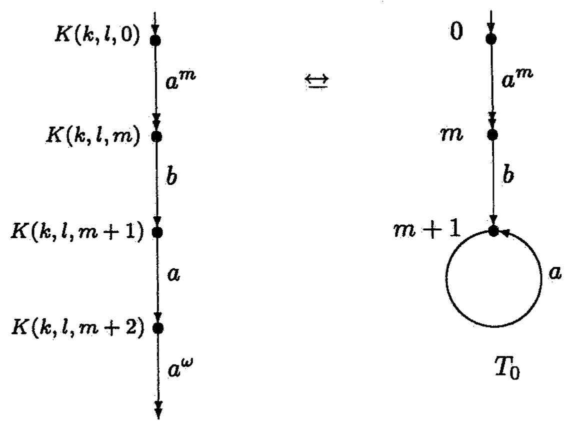


Figure 1: The root connected part of $SOS(E)(K(k,l,0))$ and a symbolic implementation $T_0$.

**Theorem 4.2.1.** *Every (primitive) recursive transition system over a finite set of labels not including $\surd$ can be specified modulo bisimulation over $\mu$CRL(TREC) (over $\mu$CRL(PRIM), respectively).*

**Proof.** Assume $T = (S, \{a_1, ..., a_n\}, next, i)$ and recall that in a recursive transition system the initial state is 0. Let $AA(x)$ abbreviate the process term

$$(a_1 \lhd x = i(a_1) \rhd \delta + ... + a_n \lhd x = i(a_n) \rhd \delta)$$

so $AA(k)$ is for any $k \in Nat$ a closed term of $n$ summands, one of which possibly represents a label from $T$ as an action, and all others are $\delta$.

Now $T$ is a symbolic implementation of the process $P$ specified by the following specification $E$:

$$\begin{array}{rcl} P & = & Q(0) \\ Q(x) & = & R(next(x), next(x)) \\ R(x,y) & = & AA(j_1(x)) \cdot Q(j_2(x)) \lhd x \in D_y \rhd \delta + \\ & & \delta \lhd x = 0 \rhd R(x \dot{-} 1, y). \end{array}$$

This can be seen as follows:

1. The function $next(.)$ can be specified in $\mu$CRL(TREC) as it is a total recursive function, or in case $T$ is PRIM in $\mu$CRL(PRIM).
2. Both decoding functions $j_1$ and $j_2$ are PRIM,
3. All conditions are PRIM (membership of finite sets coded by a CI; equality).

By definition of the function $next$, the property $n \in D_m \implies n < m$, and by unravelling the specification of $P$ it follows that $SOS(E)(P) \leftrightarrow T$, as the root connected parts of both transition systems are even isomorphic: (with some abuse of notation) $Q(k) \simeq k$ for any state $k$ of $T$. ∎

The analogue of this theorem to transition systems that are properly terminating, i.e., that distinguish successful termination and deadlock (see Definition 2.1.3), is slightly more complex:

**Theorem 4.2.2.** *Every properly terminating (primitive) recursive transition system over a finite set of labels can be specified modulo bisimulation over $\mu$CRL(TREC) (over $\mu$CRL(PRIM), respectively).*

**Proof.** Let $T = (S, \{a_1, ..., a_n, \surd\}, next, i)$ and let $AA(x)$ abbreviate the process term

$$(a_1 \lhd x = i(a_1) \rhd \delta + ... + a_n \lhd x = i(a_n) \rhd \delta).$$

Then $T$ is a symbolic implementation of the process $P$ specified by

$$\begin{array}{rcl} P & = & Q(0) \\ Q(x) & = & R(next(x), next(x)) \\ R(x,y) & = & AA(j_1(x)) \cdot Q(j_2(x)) \lhd x \in D_y \wedge \forall z \leq x \,.\, \neg(z \in D_{j_2(x)} \wedge j_1(z) = i(\surd)) \rhd \delta + \\ & & \delta \lhd x = 0 \rhd R(x \dot{-} 1, y) + \\ & & AA(j_1(x)) \lhd x \in D_y \wedge \forall z \leq x \,.\, z \in D_{j_2(x)} \wedge j_1(z) = i(\surd) \rhd \delta. \end{array}$$

This can be argued in the same way as in the proof above because all (slightly extended) conditions are still TREC (PRIM, respectively). Another difference with the specification in the preceding proof is the last summand of $R(x, y)$, which possibly generates successful termination states. ■

So any (primitive) recursive transition system $T = (S, L, next, i)$ with $L$ finite and either $\surd \notin L$ or $T$ properly terminating, gives rise to a specification $E$ over $\mu$CRL(TREC) (or $\mu$CRL(PRIM)) that defines a 'canonical' process term $P$ for which $SOS(E)(P) \leftrightarrow T$. In particular this holds for any process specified over $\mu$CRL(TREC) (by Theorem 3.2.1), or, more generally, for any process specified over effective $\mu$CRL without parameterised actions [GP90]. Theorem 4.2.2 then suggests a "normal form theorem", the proof of which is based on strong bisimulation semantics and on the coding of processes as a data type.

As can be expected, the restriction to PRIM functions in the data part of specifications does not make all recursive transition systems over a finite label set specifiable. The idea is that a non PRIM function (that is total recursive) can be used to define a process that has a 'branching degree' growing faster than any PRIM function (no smart use of coding and PRIM data functions can be of help):

**Theorem 4.2.3.** *There is a recursive transition system over a finite set of labels that cannot be specified modulo bisimulation over $\mu$CRL(PRIM).*

**Proof.** Consider the following function $Ack$, (a version of) the Ackermann generalised exponential [Kle52, Rog67], which is TREC but not PRIM:

$$\begin{array}{rcl} Ack(0,y) & = & y+1 \\ Ack(x+1,0) & = & Ack(x,1) \\ Ack(x+1,y+1) & = & Ack(x, Ack(x+1,y)). \end{array}$$

Let the specification $E$ over $\mu$CRL(TREC) be defined by

$$\begin{array}{rcl} P(x) & = & a \cdot P(x+1) + \\ & & b \cdot Q(Ack(x,x))) \\ Q(x) & = & c \cdot R(x) + \\ & & \delta \lhd x = 0 \rhd Q(x \dot{-} 1) \\ R(x) & = & \delta \lhd x = 0 \rhd c \cdot R(x \dot{-} 1). \end{array}$$

Now assume there is a PRIM transition system $T = (S, \{a, b, c\}, next, i)$ that is a symbolic implementation of $P(0)$. Let $F : Nat \to Nat$ be such that $j_1(F(k))$ is the code of the state characterised by the trace $a^k$ and $j_2(F(k))$ is the code of the state characterised by the trace $a^k \cdot b$. Formally:

$$\begin{aligned}
G^a(x) &= \mu y < x.[y \in D_x \wedge j_1(y) = i(a)] \\
G^b(x) &= \mu y < x.[y \in D_x \wedge j_1(y) = i(b)]
\end{aligned}$$

$$\begin{aligned}
F(0) &= j(0, j_2(G^b(next(0)))) \\
F(x+1) &= j(\ j_2(G^a(next(j_1(F(x))))) \ , \\
&\quad\ j_2(G^b(next(j_2(G^a(next(j_1(F(x)))))))) \ ).
\end{aligned}$$

Observe that $F$ is PRIM if *next* is. Now for any $k \in Nat$ it holds that

$$next(j_2(F(k))) > Ack(k,k).$$

This follows from the fact that this particular value of *next* codes a state that must have $Ack(k,k)+1$ different outgoing $c$-transitions (each of these entails its own number of consecutive $c$-transitions). Consequently the CI of the set coding all these labels and resulting states is certainly larger than $Ack(k,k)$. Hence, *next* cannot be a *primitive* recursive function, contradicting the assumption. So $SOS(E)(P(0))$ is a recursive transition system over a finite label set that is not PRIM. ■

## 5 Arithmetical classification of properties

In this section the properties *bisimilarity, deadlock freedom* and *regularity* over recursive transition systems (introduced in Section 2.3) are classified in the arithmetical hierarchy. [6] The results proven state that each of these properties is *complete* in a particular class, meaning that its definition is exactly as complex as characteristic for that class.

The arithmetical characterisation takes place on the level of $\mu$CRL(TREC) specifications. By the Expressivity Theorems 4.2.1 and 4.2.2 (and by the interest in specification languages plus their associated proof theory) it is interesting to characterise properties on this level. The section is concluded with an arithmetical characterisation of the properties mentioned above with respect to the class of *all* primitive recursive transition systems over a fixed, finite label set.

### 5.1 The approach

Given a certain class $\mathcal{C}$ in the arithmetical hierarchy and one of the properties, completeness in $\mathcal{C}$ is proved as follows:

1. Show that for any specification the property is equivalent to a relation in $\mathcal{C}$. For instance, given a specification $E$ and a recursive coding $\ulcorner . \urcorner$ of $\mathbb{P}(E)$, show that $SOS(E)(p) \leftrightarrows SOS(E)(q)$ iff $(\ulcorner p \urcorner, \ulcorner q \urcorner) \in R \in \mathcal{C}$.
2. Show the *completeness* in $\mathcal{C}$ by giving a specification for which a special case of the property is equivalent to a relation that is *complete* in $\mathcal{C}$. For instance, given some specification $E$ and process terms $p(k), q$ over $E$, show for some $R$ complete in $\mathcal{C}$ that $SOS(E)(p(k)) \leftrightarrows SOS(E)(q)$ iff $k \in R$.

So 1 implies that (in general) the property is not *more* complex than any relation in $\mathcal{C}$, and 2 implies that it is *as least as* complex as a relation that is complete in $\mathcal{C}$.

For reference to arithmetical completeness consider the following special relations, referring to the Enumeration Theorem of Kleene [Kle52]. Let $n \geq 1$, then the binary relation $E_n$ is defined as

$\{(z,x) \mid \exists y_1 \forall y_2 ... \exists y_n \,.\, T_n(z,x,y_1,...y_n)\}$ in case $n$ is odd,
$\{(z,x) \mid \exists y_1 \forall y_2 ... \forall y_n \,.\, \neg T_n(z,x,y_1,...y_n)\}$ in case $n$ is even.

Now $E_n$ is complete in $\Sigma^0_n$, and $\neg E_n$ (i.e., the complement of $E_n$) is complete in $\Pi^0_n$.

[6] This means that each of these properties is related to a numerical relation of which the 'complexity of definition' is subject to a clear cut criterion for classification: the 'arithmetical hierarchy' [Rog67].

A crucial step in the proofs to come is based on some standard facts about the basic theory of ACP [BW90]. The standard axiom system BPA$_\delta$ (Basic Process Algebra with $\delta$) consists of the axioms in Table 4.

| | | | |
|---|---|---|---|
| $x+(y+z)=(x+y)+z$ | A1 | $x+\delta=x$ | A6 |
| $x+y=y+x$ | A2 | $\delta\cdot x=\delta$ | A7 |
| $x+x=x$ | A3 | | |
| $(x+y)\cdot z=x\cdot z+y\cdot z$ | A4 | | |
| $(x\cdot y)\cdot z=x\cdot(y\cdot z)$ | A5 | | |

Table 4: The axioms of BPA$_\delta$.

Given a $\mu$CRL(TREC) specification $E$, the set of BPA$_\delta$-*terms* over $E$ consists of the process terms that can be constructed out of $\delta$, the actions declared in $E$ and the operators $+$ and $\cdot$. There is the following standard completeness result: for any two BPA$_\delta$-terms $p, q$ over $E$ it holds that

$$\mathrm{BPA}_\delta \vdash p=q \iff SOS(E)(p) \leftrightarrow SOS(E)(q).$$

Moreover, given a coding $\ulcorner.\urcorner$ of the BPA$_\delta$-terms over $E$ as a recursive set, the relation $A \subseteq Nat \times Nat$ defined by

$$A \overset{def}{=} \{(\ulcorner p\urcorner, \ulcorner q\urcorner) \mid \mathrm{BPA}_\delta \vdash p=q\}$$

is recursive (the idea is that the axioms of BPA$_\delta \setminus \{$A1, A2$\}$, when read as rewriting rules from left to right form a strongly normalising rewriting system [BW90]).

## 5.2 Bisimilarity

Naively one may expect from the definition of bisimilarity (over some $\mu$CRL(TREC) specification) that this property is $\Sigma^1_1$ hard: *there exists* a relation $R$ such that *for all* pairs $(p,q) \in R$ the transfer property (see 2.1.1) is satisfied and that contains the roots. However, bisimilarity turns out to be complete in $\Pi^0_1$. Application of the two steps mentioned above:

1. In terms of the theory of ACP, this step comprises application of the "Approximation Induction Principle (AIP$^-$)" from [BW90]. This principle states that two finitely branching transition systems are bisimilar if all their projections are. Let $E$ be some arbitrary specification over $\mu$CRL(TREC).

   (a) Define a function $\ulcorner.\urcorner$ that codes the closed process terms over $E$ as a recursive set, and observe that a PRIM predicate can then be defined that singles out the codes of the BPA$_\delta$-terms over $E$.

   (b) Define a TREC function $Exp(n,x)$ that expands $n$ times the terms headed by process identifier(s) that occur in the term coded by $x$. A simple example to explain this function:

   $$M = a\cdot M$$

   then $Exp(0,\ulcorner M\urcorner)=\ulcorner M\urcorner$, $Exp(1,\ulcorner M\urcorner)=\ulcorner a\cdot M\urcorner$, $Exp(2,\ulcorner M\urcorner)=\ulcorner a\cdot a\cdot M\urcorner$,....

   (c) Define a TREC function $Pr(n,x)$ that yields a BPA$_\delta$-term of at most 'depth' $n$ with the property that

   $$Pr(n, Exp(n, \ulcorner p\urcorner))$$

codes a BPA$_\delta$-term, say $p_n$, such that $SOS(E)(p_n)$ is bisimilar with the $n_{th}$ projection of $SOS(E)(p)$. Using the example above: $Pr(0,\ulcorner M\urcorner) = Pr(1,\ulcorner M\urcorner) = \ulcorner\delta\urcorner$, and $Pr(1,\ulcorner a\cdot M\urcorner) = Pr(1,\ulcorner a\cdot a\cdot M\urcorner) = \ulcorner a\urcorner$. (In the terminology of [BW90], *Pr* represents only a pseudo projection function on process terms, for it yields $\ulcorner\delta\urcorner$ when applied to terms headed by a process identifier. However, the combination of syntactic guardedness and applying *Exp* sufficiently often makes this deviation irrelevant.)

Then $SOS(E)(p) \leftrightarrow SOS(E)(q)$ iff

$$(\ulcorner p\urcorner,\ulcorner q\urcorner) \in \{(x,y) \mid \forall z \,.\, A(Pr(z,Exp(z,x)),Pr(z,Exp(z,y)))\}$$

with $A$ as defined in the previous section 5.1. As the defining relation is recursive, it follows that $\leftrightarrow$ over $E$ is in the class $\Pi^0_1$.

2. Bisimilarity can be proven *complete* in $\Pi^0_1$ using the following specification $E$ over $\mu$CRL(PRIM):

$$\begin{aligned} K(x,y,z) &= a\cdot\delta \lhd T(x,y,z) \rhd a\cdot K(x,y,z+1)\\ M &= a\cdot M \end{aligned}$$

then $$SOS(E)(K(k,l,0)) \leftrightarrow M \iff \forall z\,.\,\neg T(k,l,z) \iff (k,l)\in\neg E_1.$$

The latter problem is complete in $\Pi^0_1$.

## 5.3 Deadlock freedom and regularity

The arithmetical characterisation of 'deadlock freedom' is also complete in $\Pi^0_1$. Application of the two classification steps:

1. Let $E$ be fixed over $\mu$CRL(TREC) and $\ulcorner.\urcorner$ be a coding function such that $\ulcorner\mathbb{P}(E)\urcorner$ is recursive. It is not hard to define a PRIM relation that characterises the codes of BPA$_\delta$-terms over $E$ that can be proven $\delta$-free in the system BPA$_\delta$, say *DF*. Using the functions *Exp* and *Pr* introduced above, $SOS(E)(p)$ is deadlock free iff

$$\ulcorner p\urcorner \in \{x \mid \forall y\,.\,DF(Pr(y,Exp(y,x))) \vee y=0\}$$

(characterising that all (non-zero) projections on process terms are deadlock free). As the defining relation is recursive, it follows that deadlock freedom over $E$ is in the class $\Pi^0_1$.

2. Deadlock freedom is *complete* in $\Pi^0_1$ by the example in the previous section:

$$SOS(E)(K(k,l,0)) \text{ is deadlock free} \iff \forall z\,.\,\neg T(k,l,z) \iff (k,l)\in\neg E_1.$$

The property 'regularity' is complete in $\Sigma^0_2$:

1. Let $E$ be a specification over $\mu$CRL(TREC). Assume some coding of the finite transition systems over $\mathbb{A}(E)$ and a (PRIM) relation *Fin* that characterises these. For example, $n\in Fin$ iff $j_2(n)$ is the CI of codes of transitions $k \xrightarrow{\ulcorner a\urcorner} l$ with $a\in\mathbb{A}(E)$ and $k,l\le j_1(n)$. Let the function $Pr' : Nat\times Nat \to Nat$ be such that $Pr'(x,y)$ yields the code of $\delta$ in case $x\notin Fin$, and yields the (code of) the $y_{th}$ projection of $x$ as a BPA$_\delta$-term otherwise. Now $SOS(E)(p)$ is regular iff

$$\ulcorner p \urcorner \in \{x \mid \exists y \forall z \,.\, A(Pr(z, Exp(z,x)), Pr'(y,z)) \vee z = 0\}$$

where the relation $A$ denoted $\mathrm{BPA}_\delta$ derivability. Hence, regularity over $E$ can be defined as a $\Sigma^0_2$ relation.

2. For the completeness in $\Sigma^0_2$, consider for fixed $k, l$ the trace

$$a \cdot b^{1+\mu y . T(k,l,0,y)} \cdot a^2 \cdot b^{1+\mu y . T(k,l,1,y)} \cdot \ldots \cdot a^{m+1} \cdot b^{1+\mu y . T(k,l,m,y)} \cdot \ldots$$

This trace is "regular" iff $\exists x \forall y . \neg T(k,l,x,y)$, for the trace then ends in a $b$-loop. A $\mu$CRL(PRIM) specification $E$ for defining this trace is

$$\begin{array}{rcl} K(v,w,x,y,z) & = & a \cdot L(v,w,x,y,z) \triangleleft z = 0 \triangleright a \cdot K(v,w,x,y,z \dot{-} 1) \\ L(v,w,x,y,z) & = & b \cdot K(v,w,x+1,0,x+1) \triangleleft T(v,w,x,y) \triangleright b \cdot L(v,w,x,y+1,z). \end{array}$$

$$\begin{array}{lcl} \text{So } \ SOS(E)(K(k,l,0,0,0)) \text{ is regular} & \Longleftrightarrow & \exists x \forall y \,.\, \neg T(k,l,x,y) \\ & \Longleftrightarrow & (k,l) \in E^1_2. \end{array}$$

The latter problem is complete in $\Sigma^0_2$.

## 5.4 Properties of primitive recursive transition systems

The classification results from above also hold over the domain of all PRIM transition systems over a fixed, finite label set. In the following it is argued (somewhat sketchy) that these classification results also hold for PRIM transition systems that are properly terminating (see Definition 2.1.3). The slightly simpler case restricting to transition systems over a finite label set that do not distinguish successful termination from deadlock can be dealt with in a similar way.

Let a finite set $L$ of labels including $\surd$ be given. Fix a coding of all unary PRIM functions, and define a PRIM predicate *Prim* that characterises these codes and a binary PRIM function *Eval* that evaluates application (standard). In order to characterise $PT$ transition systems by such codes, assume a surjective function

$$i : Nat \rightarrow L$$

be given (so any label is coded by at least one element of $Nat$).

First of all the property $PT$ itself should be defined over these codes. This property can now be defined as a unary predicate using the following auxiliary function and relation:

- Define the PRIM function $TL$ (Tree Like conversion) that transforms codes of unary PRIM functions into those of bisimilar tree-like versions with the property that $s \xrightarrow{l} s' \implies s < s'$ (cf. the transformation in Section 2.2).
- Define the binary PRIM predicate $RC$ (Root Connectedness) such that $RC(x,y)$ iff $y$ is a root connected state in $TL(x)$. By the property of $TL$ mentioned above, this can be done by primitive recursion on the second argument.

As the predicate $PT$ should only test values in the codomain of $TL$, it can be defined in $\Pi^0_1$.

Further define a binary PRIM projection function (operating on a transition system $TL(x)$ and a state $y$ for which $RC(x,y)$) that extracts the labels of next steps as codes of actions for constructing process terms that are:

- possibly an action followed by $\delta$,
- possibly an action followed by nothing (successful termination),

- possibly an action followed by the remaining smaller projection of the root connected subtree of $TL(x)$ from this next state.

In the same style as described above, the properties in question can now be defined. Note that concerning all *recursive* transition systems over a finite label set (characterised by standard codes for recursive functions), a property like bisimilarity can be defined by demanding totality (complete in $\Pi^0_2$) plus the BPA$_\delta$-derivability between all finite projections as sketched above.

## 6 Restricted forms of bisimilarity

In this section different restricted forms of bisimilarity are investigated. The motivation for this is an observation of BERGSTRA [Ber91] that there are bisimilar primitive recursive transition systems over a finite label set that cannot be related by means of a recursively enumerable bisimulation. So even over a relatively simple domain, bisimilarity is a complex relation. Furthermore two more forms of bisimilarity are distinguished that are both weaker than bisimilarity based upon the existence of a r.e. bisimulation.

### 6.1 Recursively enumerable bisimulations

In the following theorem it is shown that recursively enumerable bisimulations do not identify all bisimilar primitive recursive transition systems (over a fixed, finite label set). Its proof uses recursively inseparable sets [Rog67] in the specification of processes that are bisimilar, but for which the existence of a recursively enumerable bisimulation implies the existence of a recursive separation.

**Theorem 6.1.1** (BERGSTRA [Ber91]). *There are two primitive recursive transition systems over a finite set of labels that are bisimilar, but cannot be related by means of a recursively enumerable bisimulation.*

**Proof.** Let $W_{e_1}$ and $W_{e_2}$ be recursively inseparable sets. Consider the following specification $E$ over $\mu$CRL(PRIM):

$$\begin{array}{rclrcl}
A(x) & = & e\cdot A(x+1) + & B(x) & = & e\cdot B(x+1) + \\
 & & d\cdot P_1(x,0) + & & & d\cdot Q_1(x,0) + \\
 & & d\cdot P_2(x,0) & & & d\cdot Q_2(x,0) \\
 & & & & & \\
P_1(x,y) & = & a\cdot P_1(x,y+1) + & Q_1(x,y) & = & a\cdot Q_1(x,y+1) + \\
 & & b\cdot\delta \lhd T(e_1,x,y) \rhd \delta + & & & b\cdot\delta \lhd T(e_1,x,y) \rhd \delta + \\
 & & c\cdot\delta \lhd T(e_2,x,y) \rhd \delta & & & b\cdot\delta \lhd T(e_2,x,y) \rhd \delta \\
 & & & & & \\
P_2(x,y) & = & a\cdot P_2(x,y+1) + & Q_2(x,y) & = & a\cdot Q_2(x,y+1) + \\
 & & c\cdot\delta \lhd T(e_1,x,y) \rhd \delta + & & & c\cdot\delta \lhd T(e_1,x,y) \rhd \delta + \\
 & & b\cdot\delta \lhd T(e_2,x,y) \rhd \delta & & & c\cdot\delta \lhd T(e_2,x,y) \rhd \delta.
\end{array}$$

Then $SOS(E)(A(0))$ and $SOS(E)(B(0))$ are PRIM transition systems (cf. Corollary 3.2.2). Now

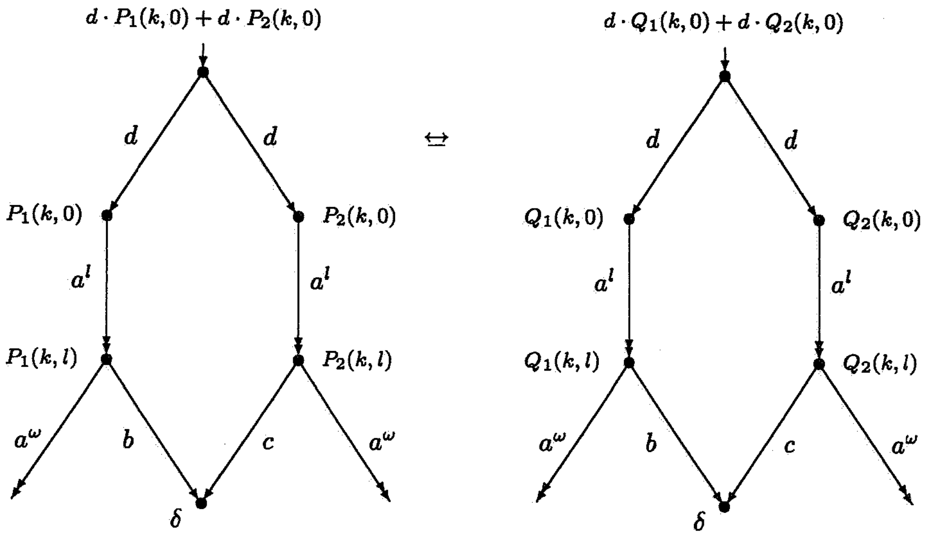
$$SOS(E)(A(0)) \leftrightarrow SOS(E)(B(0)).$$

To show this, it is first argued that for any $k \in Nat$ one has

$$SOS(E)(d\cdot P_1(k,0) + d\cdot P_2(k,0)) \leftrightarrow SOS(E)(d\cdot Q_1(k,0) + d\cdot Q_2(k,0)).$$

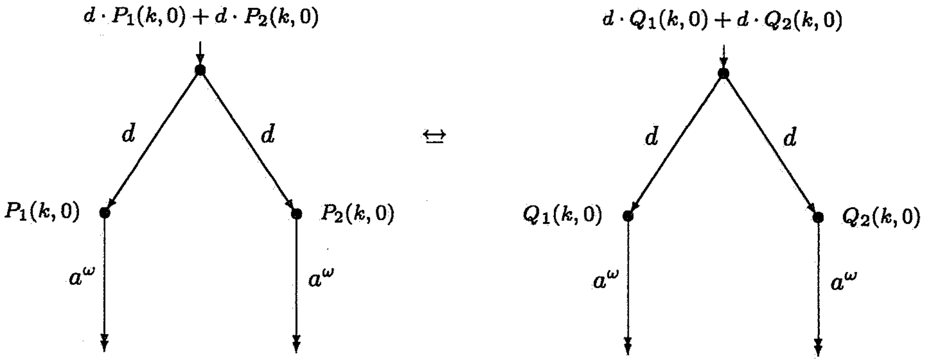
Distinguishing the three cases $k \in W_{e_1}$, $k \in W_{e_2}$ and $k \notin W_{e_1} \cup W_{e_2}$, this can most easily be shown by pictures suggesting the isomorphisms to be used.

1. If $k \in W_{e_1}$, say $T(e_1, k, l)$:



2. If $k \in W_{e_2}$ the $b$ and $c$ labels of $SOS(E)(d \cdot P_1(k,0) + d \cdot P_2(k,0))$ above should be reversed, and again bisimulation is obvious;

3. If $k \notin W_{e_1} \cup W_{e_2}$ :


$d \cdot P_1(k,0) + d \cdot P_2(k,0)$
$d \cdot Q_1(k,0) + d \cdot Q_2(k,0)$
$d$
$\leftrightarrow$
$P_1(k,0)$
$P_2(k,0)$
$Q_1(k,0)$
$Q_2(k,0)$
$a^\omega$

So for any $k \in Nat$ it follows that $SOS(E)(A(k)) \leftrightarrow SOS(E)(B(k))$. As bisimilarity is a congruence relation, it follows easily that

$$SOS(E)(A(0)) \leftrightarrow SOS(E)(B(0)).$$

It remains to be shown that any (isomorphic image of a) bisimulation relating $SOS(E)(A(0))$ and $SOS(E)(B(0))$ cannot be recursively enumerable. Assume the contrary for a relation $S$, then both

$$S_1 \overset{def}{=} \{(\ulcorner p \urcorner, \ulcorner q \urcorner) \mid (p,q) \in S\} \cap \{(\ulcorner P_1(n,0) \urcorner, \ulcorner Q_1(n,0) \urcorner) \mid n \in Nat\}$$

and

$$S_2 \overset{def}{=} \{(\ulcorner p \urcorner, \ulcorner q \urcorner) \mid (p,q) \in S\} \cap \left(Nat^2 \setminus \{(\ulcorner P_1(n,0) \urcorner, \ulcorner Q_1(n,0) \urcorner) \mid n \in Nat\}\right)$$

are also recursively enumerable, assuming that $\ulcorner . \urcorner$ is a function in the style of the proof of Theorem 3.2.1, and $\ulcorner P_1(k) \urcorner$ and $\ulcorner Q_1(k) \urcorner$ are TREC functions on $k$. Let for $i = 1, 2$

$$S'_i \overset{def}{=} \{n \mid (\ulcorner P_1(n,0) \urcorner, \ulcorner Q_i(n,0) \urcorner) \in S_i\}.$$

Then also $S'_1$ and $S'_2$ are recursively enumerable. As $S'_1$ and $S'_2$ are complementary, both are recursive. But this is a contradiction, as $S'_1$ constitutes a recursive separation of $W_{e_1}$ and $W_{e_2}$: first observe that for any $n \in Nat$ it must hold that $(A(n), B(n)) \in S$. Secondly,

$n \in W_{e_1}$ : as $A(n) \xrightarrow{d} P_1(n,0)$ and $B(n) \xrightarrow{d} Q_i(n,0)$ $(i = 1,2)$, at least one of $(P_1(n,0), Q_i(n,0))$ should be in $S$. By bisimilarity and $n \in W_{e_1}$, this must be the case for $i = 1$, and not for $i = 2$. Hence $n \in S'_1$.

$n \in W_{e_2}$ : in a similar way it follows that $n \in S'_2 = \neg S'_1$.

■

Write $\leftrightarrows_{r.e.}$ for bisimilarity induced by a recursively enumerable bisimulation. An immediate consequence of the theorem above is that both Expressivity Theorems 4.2.1 and 4.2.2 do not hold modulo $\leftrightarrows_{r.e.}$ (as $\leftrightarrows_{r.e.}$ is a transitive relation).

## 6.2 Weaker bisimulations

It is shown that in the domain of PRIM transition systems over a fixed, finite label set 'PRIM bisimilarity' identifies less than 'recursive bisimilarity', which in turn identifies less than r.e. bisimilarity. The first result uses processes defined as in the proof of the preceding theorem.

**Theorem 6.2.1.** *There are two primitive recursive transition systems over a finite set of labels that are recursively bisimilar, but not PRIMly bisimilar.*

**Proof.** Consider the process declarations from the proof of BERGSTRA'S Theorem 6.1.1, but now take $W_{e_1}$ a recursive set that is not PRIM, and $W_{e_2} = Nat \setminus W_{e_1}$. Proceeding as in the proof of 6.1.1, it follows that the PRIM transition systems $SOS(E)(A(0))$ and $SOS(E)(B(0))$ are recursively bisimilar, but not PRIMly bisimilar. ■

The next result again uses recursively inseparable sets. In its proof two r.e. bisimilar processes are defined for which the assumption of a recursive bisimulation implies a recursive separation.

**Theorem 6.2.2.** *There are two primitive recursive transition systems over a finite set of labels that are r.e. bisimilar, but not recursively bisimilar.*

**Proof.** Let $W_{e_1}$ and $W_{e_2}$ be recursively inseparable sets. Consider the following specification:

$$A = a \cdot A$$

$$\begin{aligned} B(x,y) &= a \cdot B(x, y+1) \; + \\ &\quad C(x,y) \lhd \exists y' \le y \,.\, T(e_1, x, y') \rhd \delta \; + \\ &\quad b \cdot \delta \lhd \exists y' \le y \,.\, T(e_2, x, y') \rhd \delta \end{aligned}$$

where $C(x,y) \equiv$

$$\Sigma_{x<z<y}(a \cdot B(z,0) \lhd \exists y' \le y \,.\, T(e_1, z, y') \;\wedge\; \neg \exists z' < z, y'' < y \,.\, x < z' \wedge T(e_1, z', y'') \rhd \delta).$$

(A formal, but less readable description of $C(x,y)$ can be obtained by inserting appropriate conditionals and defining a recursion for this PRIMly bounded sum.)

Let $k \stackrel{def}{=} \mu x \,.\, [x \in W_{e_1}]$. Typically, in $SOS(E)(B(k,0))$ the root $B(k,0)$ is connected to all states $B(m,n)$ with $m \in W_{e_1}$ via $a$-transitions, so all of these must be related to $A$ in a bisimulation. Now $SOS(E)(A) \leftrightarrow_{r.e.} SOS(E)(B(k,0))$, for given an appropriate coding function $\ulcorner . \urcorner$ of closed process terms, $\{(\ulcorner A \urcorner, \ulcorner B(x,y) \urcorner) \mid x \in W_{e_1}, y \in Nat\}$ is a r.e. bisimulation. Furthermore $B(m,n)$ with $m \in W_{e_2}$ cannot be related to $A$ because of the $b$-transition. The assumption that there is a recursive bisimulation relating $SOS(E)(A)$ and $SOS(E)(B(k,0))$ thus assumes a recursive separation of $W_{e_1}$ and $W_{e_2}$. ∎

## 7 Conclusions

Essentially based on [BBK87], the property of a transition system being (primitive) recursive is defined. Also two fragments of the specification language $\mu$CRL [GP90, GP91a] are introduced, $\mu$CRL(TREC) and $\mu$CRL(PRIM), and their correspondence with transition systems is established: $\mu$CRL(TREC) is universally expressive (modulo bisimulation) with respect to all recursive transition systems over a fixed, finite label set, and so is $\mu$CRL(PRIM) if one restricts to the primitive recursive transition systems. Given a $\mu$CRL(TREC) specification, the basic properties bisimilarity, deadlock freedom and regularity all turn out to be undecidable. Furthermore, if two primitive recursive transition systems are bisimilar, it may well be that the witnessing relation is of a rather complex nature. For two process terms $p,q$, write $SOS(E)(p) \leftrightarrow_{rec} SOS(E)(q)$ if the witnessing bisimulation is recursive, and let $\leftrightarrow_{prim.rec}$ be defined similarly. It is shown that $\leftrightarrow_{prim.rec}$ identifies less than $\leftrightarrow_{rec}$ , which in turn identifies less than $\leftrightarrow_{r.e.}$ , which in turn identifies less than bisimilarity as such (i.e., $\leftrightarrow$ ). One more consequence addresses a proof theoretic phenomenon: consider some axiomatic, finitary proof system for $\mu$CRL(PRIM), say $\vdash$. Proving for any two closed process terms $p,q$ over some $\mu$CRL(PRIM) specification $E$ that

$$\vdash p = q \implies SOS(E)(p) \leftrightarrow_{r.e.} SOS(E)(q)$$

shows by the result of Bergstra (Theorem 6.1.1) and the Expressivity Theorem 4.2.2 that $\vdash$ is not complete (relative to $E$). As the implication above can be shown for the $\mu$CRL(PRIM) fragment of the proof system for $\mu$CRL defined in [GP91b], it follows that this system is not complete with respect to this fragment. This applies also to the $\mu$CRL(TREC) fragment. A conclusion of this may be that other process algebras, for example those defined by recursively enumerable bisimilarity, have their right of existence.

A final remark is that all the results of this paper can be generalised to a restricted case of countably many labels or actions. Let the following example be illustrative: assume $a,b$ are names of actions that are parameterised over $\mathbb{N}$, so eg. $a(0), b(17), \ldots$ are then actions. [7] In this case both Expressivity Theorems, and hence all preceding results are still preserved. To see this let $i$, the function coding the labels, be defined as:

$$i(a(k)) = j(0,k) \text{ and } i(b(k)) = j(1,k).$$

Let $AA(x)$ abbreviate the process term

$$a(j_2(x)) \triangleleft j_1(x) = 0 \triangleright \delta + b(j_2(x)) \triangleleft j_1(x) = 1 \triangleright \delta.$$

It is immediately clear how the proofs of Theorems 4.2.1 and 4.2.2 should be modified.

[7] In (effective) $\mu$CRL actions may be data parameterised.

## References

[BB90] J.C.M. Baeten and J.A. Bergstra. Process algebra with signals and conditions. Report P9008, University of Amsterdam, Amsterdam, 1990. To appear in *Proceedings NATO ASI Summer School*, Marktoberdorf 1990, LNCS.

[BBK87] J.C.M. Baeten, J.A. Bergstra, and J.W. Klop. On the consistency of Koomen's fair abstraction rule. *Theoretical Computer Science*, 51(1/2):129–176, 1987.

[Ber91] J.A. Bergstra, 1991. Personal Communications.

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

[CCI87] CCITT Working Party X/1. *Recommendation Z.100 (SDL)*, 1987.

[Dar90a] Ph. Darondeau. Concurrency and computability. In I. Guessarian, editor, *Semantics of Systems of Concurrent Processes, Proceedings LITP Spring School on Theoretical Computer Science,* La Roche Posay, France, volume 469 of *Lecture Notes in Computer Science*, pages 223–238. Springer-Verlag, 1990.

[Dar90b] Ph. Darondeau. Recursive graphs are not stable under maximal reduction. Note, IRISA, Campus de Beaulieu, F35042 Rennes cedex, December 1990.

[Dav82] M. Davis. *Computability and Unsolvability.* Dover Publications, Inc., 1982.

[Fer92] T. Fernando. Between programs and processes: absoluteness and open endedness. Technical Report IAM 92-011, Institut für Informatik, Universität Bern, 1992.

[GP90] J.F. Groote and A. Ponse. The syntax and semantics of $\mu$CRL. Report CS-R9076, CWI, Amsterdam, 1990.

[GP91a] J.F. Groote and A. Ponse. $\mu$CRL: A base for analysing processes with data. In E. Best and G. Rozenberg, editors, *Proceedings 3rd Workshop on Concurrency and Compositionality, Goslar, GMD-Studien Nr. 191*, pages 125–130. Universität Hildesheim, 1991.

[GP91b] J.F. Groote and A. Ponse. Proof theory for $\mu$CRL. Report CS-R9138, CWI, 1991.

[GV89] J.F. Groote and F.W. Vaandrager. Structured operational semantics and bisimulation as a congruence (extended abstract). In G. Ausiello, M. Dezani-Ciancaglini, and S. Ronchi Della Rocca, editors, *Proceedings 16th ICALP,* Stresa, volume 372 of *Lecture Notes in Computer Science*, pages 423–438. Springer-Verlag, 1989. Full version to appear in *Information and Computation.*

[HHJ+87] 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 ACM*, 30(8):672–686, August 1987.

[HU79] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.

[ISO87] ISO. *Information processing systems – open systems interconnection – LOTOS – a formal description technique based on the temporal ordering of observational behaviour* ISO/TC97/SC21/N DIS8807, 1987.

[Kle52] S.C. Kleene. *Introduction to Meta Mathematics.* North-Holland, 1952.

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

[MV90] S. Mauw and G.J. Veltink. A process specification formalism. *Fundamenta Informaticae*, XIII:85–139, 1990.

[Par81] D.M.R. Park. Concurrency and automata on infinite sequences. In P. Deussen, editor, 5th *GI Conference*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183. Springer-Verlag, 1981.

[Plo81] G.D. Plotkin. A structural approach to operational semantics. Report DAIMI FN-19, Computer Science Department, Aarhus University, 1981.

[Rog67] H. Rogers. *Theory of Recursive Functions and Effective Computability.* McGraw-Hill Book Co., 1967.

[SSA90] SPECS-Semantics and Analysis. *Definition of MR and CRL Version 2.1.* Specification and Programming Environment for Communicating Software (SPECS), RACE Ref: 1046, Report 46/SPE/WP5/DS/A/017/b1, December 1990.