Formal Aspects of Computing (1996) 8: 648–678

Formal Aspects of Computing

Computable Processes and Bisimulation Equivalence

Alban Ponse

University of Amsterdam, Faculty of Mathematics, Computer Science, Physics and Astronomy, Amsterdam, The Netherlands

Keywords: Process algebra; Labelled transition system; Bisimulation equivalence; Regular process; ACP; μCRL; Computability

Abstract. A process is called *computable* if it can be modelled by a transition system that has a recursive structure—implying finite branching. The equivalence relation between transition systems considered is strong bisimulation equivalence. The transition systems studied in this paper can be associated to processes specified in common specification languages such as CCS, LOTOS, ACP and PSF. As a means for defining transition systems up to bisimulation equivalence, the specification language μCRL is used. Two simple fragments of μCRL are singled out, yielding universal expressivity with respect to recursive and primitive recursive transition systems. For both these domains the following properties are classified in the arithmetical hierarchy: *bisimilarity*, *perpetuity* (both Π_1^0), *regularity* (having a bisimilar, finite representation, Σ_2^0), *acyclic regularity* (Σ_1^0), and *deadlock freedom* (distinguishing deadlock from successful termination, Π_1^0). 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 by the complexity of the witnessing relations, which extends r.e. bisimilarity. Hence, primitive recursive transition systems already form an interesting class.

1. Introduction

In this paper, rooted labelled transition systems are considered as mathematical representations of processes. Such a transition system consists of a set of states,

Correspondence and offprint requests to: A. Ponse, University of Amsterdam, Faculty of Mathematics, Computer Science, Physics and Astronomy, Kruislaan 403, 1098 SJ Amsterdam, The Netherlands. E-mail: alban@fwi.uva.nl.

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. Selecting one state as the root (the *initial* state) then yields a formal representation of a process. Furthermore, one can add the facility to distinguish between *successful termination* and *deadlock* in transition systems. This is modelled by a special label representing successful termination. A widely studied behavioural equivalence relation on transition systems is *(strong) bisimulation equivalence* [Par81, Mil89, GrV92]: the bisimulation equivalence class of a transition system determines a process.

Two particular types of transition systems are studied in this paper: a transition system is *recursive* [1] if its set of states and its set of labels are recursive, and from each state all next steps can be computed as a finite set. It is *primitive recursive* if all these ingredients are so. A process is called *computable* if it can effectively be associated with a recursive transition system. In the setting of process specification formalisms, this seems a natural interpretation of computability.

The processes studied in this paper can be specified in common specification languages such as CCS [Mil89] and LOTOS [ISO87], or ACP [BeK84, BaW90] and PSF [MaV90, MaV93]. The ACP-based approaches comprise the CCS-like ones by employing the two types of termination, a more flexible communication format, and sequential composition as a primitive operator. Therefore, the set-up of this paper is ACP-based, though it is taken care that if one does not wish to distinguish between successful termination and deadlock, all remaining results refer to the setting of (value-passing) CCS and LOTOS (replacing sequential composition by action prefixing). The ACP-like approach of specifying processes solely by actions, a finite number of process operators, and guarded recursion is extended by including conditionals (if—then—else—fi constructs) and data-parametric recursion as specification primitives. As an example, consider the process $X(0)$ recursively defined by the one-liner:

$$X(n) \stackrel{def}{=} \textbf{if } [n \text{ is a prime number}] \textbf{ then } a \cdot X(n+1) \textbf{ else } b \cdot X(n+1) \textbf{ fi.}$$

Then $X(0)$ is a simple specification of the (primitive recursive) transition system

$$0 \xrightarrow{b} 1 \xrightarrow{b} 2 \xrightarrow{a} 3 \xrightarrow{a} 4 \xrightarrow{b} 5 \xrightarrow{a} \ldots.$$

having the naturals as states, and an a-transition between states n and $n+1$ whenever n is a prime number, and a b-transition otherwise. By a basic result in [BBK87], it is possible to specify this transition system up to weak bisimilarity in ACP with finite recursion and abstraction, but such a specification is not so simple. (Abstraction, based on Milner's silent steps [Mil89], is not considered in this paper.)

The purpose of this paper is twofold. First, it is intended to illustrate that a systematic inclusion of conditionals and data-parametric recursion in process specification languages provides a simple and powerful means for the specification of transition systems up to bisimulation equivalence, and hence for the study of process theory. Secondly, some properties of transition systems modulo bisimulation and bisimilarity itself are analyzed in terms of basic recursion theory. Restricting to computable processes, the first goal is pursued by using the particular specification language μCRL (*micro* Common Representation Language,

[1] Knowledge of basic recursion theory is assumed (although some fundamentals are recalled). Common references to recursion theory are [Rog67, Dav82].

[GrP91a, GrP95]). For each of the selected domains of transition systems, a simple fragment of μCRL is defined that is universally expressive: each recursive transition system can be expressed modulo bisimulation in a canonical way. In the case of *primitive* recursive transition systems without the distinction between successful termination and deadlock, this can be done in one single, effective μCRL 'specification'. Otherwise, a recursive transition system induces its own effective μCRL specification and therewith its own, enumerable process specification language. Both these μCRL fragments employ a restrictive, decidable form of guardedness. (Guardedness is a criterion for defining processes in a recursive way.)

A property of a transition system modelling some process should be bisimulation invariant, as it is on the level of bisimulation equivalence that transition systems model processes. For instance, *number of states* is no such property, as even very simple transition systems can be bisimilar while having a different number of states. Employing process algebraic techniques, the following bisimulation invariant properties of (primitive) recursive transition systems are investigated and classified:

- *bisimilarity* — complete in Π_1^0;
- *perpetuity* (no possibility to terminate) — complete in Π_1^0;
- *regularity* (having a bisimilar *finite* representation) — complete in Σ_2^0;
- *acyclic regularity* (regularity without cycles) — complete in Σ_1^0.

A fifth property distinguishes between successful termination and deadlock, and therefore applies not (so easily) to CCS (I am not aware of specific properties of CCS-like transition systems that model deadlock):

- *deadlock freedom* — complete in Π_1^0.

These properties are especially relevant when (bisimilarity classes of) transition systems are defined in a formalism for process specification: for complex specifications they are not obvious and can be essential for tooling or correctness. Bisimilarity is of interest by definition since it characterizes all of what is taken to be important of a transition system. Perpetual processes often occur in process theory (note that a perpetual process is deadlock free). Regularity 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: the presence of an "irregular trace" contradicts the Pumping Theorem for regular languages [HU79]. Finite transition systems are of interest, because they are easy comprehensible (e.g., for a computer tool). The interest of deadlock freedom can be motivated as follows. In ACP or μCRL, concurrent processes are often defined using parallel operators and communication declarations. The remnants of unsuccessful communications are then encapsulated: the corresponding transitions are removed. If at some point there is no communication possible, this causes a deadlock. (For details, see e.g. the text book [BaW90].)

Finally, the nature of bisimilarity itself (i.e., the existence of a relation that is a bisimulation) is given attention to. 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 more complex witnessing bisimulations.

The paper is organized as follows: in Section 2, transition systems are introduced. In particular, the two forms of termination and recursive transition

systems are defined. In Section 3, the fragments of μCRL are defined. An effective, operational semantics for these fragments is given in Section 4. Section 5 is on expressivity of the μCRL fragments. Section 6 contains the arithmetical classification of the properties of recursive transition systems under consideration, and in Section 7 some weaker forms of bisimilarity over primitive recursive transition systems are investigated. Finally, Section 8 contains some conclusions, and a discussion of related work.

2. Computable Processes

In this section transition systems comprising successful termination and deadlock, and bisimulation equivalence are introduced. Then 'computable' behaviour is defined by means of transition systems having a recursive structure.

2.1. Transition Systems, Bisimulation and Termination

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

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

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.

Transition systems $\mathscr{T}_1 = (S_1, L_1, Tr_1, s_1)$ and $\mathscr{T}_2 = (S_2, L_2, Tr_2, s_2)$ are called *isomorphic*, notation

$$\mathscr{T}_1 \simeq \mathscr{T}_2,$$

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

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 *(Bisimilarity).* Given transition systems $\mathscr{T}_1 = (S_1, L_1, Tr_1, s_1)$ and $\mathscr{T}_2 = (S_2, L_2, Tr_2, s_2)$, a relation $R \subseteq S_1 \times S_2$ is a *bisimulation* iff for each pair $(t_1, t_2) \in R$ the *transfer property* holds:

- $(t_1, l, u_1) \in Tr_1 \implies \exists u_2 \,.\, (t_2, l, u_2) \in Tr_2$ and $(u_1, u_2) \in R$,
- $(t_2, l, u_2) \in Tr_2 \implies \exists u_1 \,.\, (t_1, l, u_1) \in Tr_1$ and $(u_1, u_2) \in R$,

The transition systems $\mathscr{T}_1$ and $\mathscr{T}_2$ are *bisimilar*, notation

$$\mathscr{T}_1 \leftrightarrows \mathscr{T}_2$$

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

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

Let (S, L, Tr, s_0) be some transition system. Instead of writing $(s, l, s') \in Tr$, the more pictorial notation

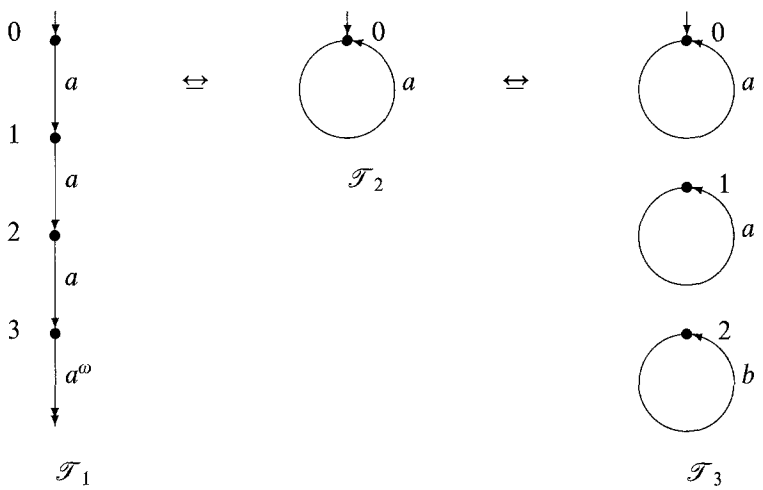
$$s \xrightarrow{l} s' \in Tr$$

is further used, in accordance with the way transition systems are visualized:

Example 2.1.2. Consider the following three transition systems, where *Nat* denotes the natural numbers:

$$\mathscr{T}_1 \stackrel{def}{=} (Nat, \{a\}, \{n \xrightarrow{a} n+1 \mid n \in Nat\}, 0)$$
$$\mathscr{T}_2 \stackrel{def}{=} (\{0\}, \{a\}, \{0 \xrightarrow{a} 0\}, 0)$$
$$\mathscr{T}_3 \stackrel{def}{=} (\{0,1,2\}, \{a,b,c\}, \{0 \xrightarrow{a} 0, 1 \xrightarrow{a} 1, 2 \xrightarrow{b} 2\}, 0).$$

These transition systems are depicted below, where the roots are indicated by a small downward arrow and $\xrightarrow{a^\omega}$ abbreviates ω successive a-transitions:



It can easily be seen that $Nat \times \{0\}$ is a bisimulation relating $\mathscr{T}_1$ and $\mathscr{T}_2$, and $\mathscr{T}_1$ and $\mathscr{T}_3$. The transition systems $\mathscr{T}_2$ and $\mathscr{T}_3$ are related by $\{(0,0)\}$.
(End example.)

An immediate consequence of regarding transition systems modulo bisimulation equivalence concerns *root connectedness* or *reachability*: only states that can be reached from the root play a role. Similarly, also the set L of labels can be restricted to those that occur in root connected transitions. 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 common practice 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, GrV92]). Therefore the transitions from any state may not depend on properties of the transition system, such as for instance root connectedness.

The property root connectedness can be used to define transition systems in which two types of *termination* are distinguished:

Definition 2.1.3 *(Termination in transition systems).* Let $\mathcal{T} = (S, L, Tr, s_0)$ 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.

Then $\mathcal{T}$ is *termination consistent*, *TC* for short, if for each $s, s' \in S$ satisfying

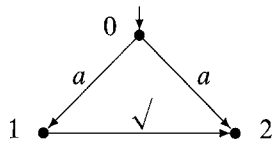
$$s \stackrel{\surd}{\longrightarrow} s' \in Tr$$

it holds that

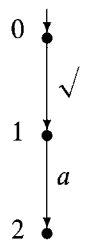
- for no $s'' \in S$ and $l \in L$, $s \stackrel{l}{\longrightarrow} s'' \in Tr$,
- for no $s'' \in S$ and $l \in L$, $s' \stackrel{l}{\longrightarrow} s'' \in Tr$,
- $s \neq s_0$.

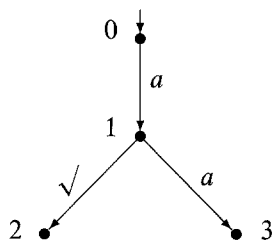
Assume $\mathcal{T}$ is *TC*. A root connected state s for which $s \stackrel{\surd}{\longrightarrow} s'$ represents *successful termination.* A state s that has no outgoing transitions, and that can be reached from the root by transitions labelled from $L \setminus \{\surd\}$ represents *deadlock.* □

Some examples. The leftmost transition system is *TC* and has two termination states: 1 represents successful termination, and 2 represents deadlock (this is still the case if $1 \stackrel{\surd}{\longrightarrow} 2$ is replaced by $1 \stackrel{\surd}{\longrightarrow} 3$).



A *TC* transition system.





Two transition systems that are not *TC*.

Note that a transition system in which $\surd$ does not occur as a label is by definition *TC*.

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 can be modelled by a *recursive* transition system. In the formal definition below 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 $x \leq j(x,y) \geq y$.

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

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

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

   is recursive.
3. The transition relation Tr 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:

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

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

If appropriate, (S, L, Tr, s_0) is sometimes denoted as

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

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

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*.

A transition system $(S, L, \to, s_0)$ is called *finite* iff both S and L are finite (so a finite transition system is always isomorphic with a primitive recursive transition system).

3. The Language μCRL, Two Simple Fragments

In this section, two simple fragments of the specification language μCRL (*micro* Common Representation Language, [GrP91a, GrP95]) are introduced. These fragments shall be used to specify recursive or primitive recursive transition systems modulo bisimulation.

A (well-formed) μCRL *specification* consists of a finite number of declaration units: some of these constitute the 'data part' of the specification, others the 'process part'. These units are introduced below. As there is in this paper only a restricted use of the language (especially concerning concurrency, parameterization and recursion), the syntax given here is a simplification.

3.1. Data Specification

Only two data types are explicitly used in this paper: the Booleans of which the constants **t** (true) and **f** (false) must be declared in any (well-formed) data specification, and the natural numbers (Nat) with constant 0 and successor

[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.

Table 1. A μCRL specification of some familiar data.

sort	**Bool**
func	$\mathbf{t}, \mathbf{f} :\rightarrow$ **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	$eq : Nat \times Nat \rightarrow$ **Bool**
var	$x, y : Nat$
rew	$eq(x, x) = \mathbf{t}$
	$eq(S(x), S(y)) = eq(x, y)$
	$eq(S(x), 0) = \mathbf{f}$
	$eq(0, S(x)) = \mathbf{f}$

function S. Furthermore, a data specification may contain a finite number of total recursive functions (declared in an algebraic way). In Table 1 a data specification of some familiar functions is displayed. The keyword **rew** ('rewriting rules') precedes the actual definitions of the functions (using the variables declared by **var**).

In the following some conventions for data specification are introduced. The Boolean standard functions $\neg, \wedge, \vee$ are used in the common way. Letters v, w, x, y, z, ... are reserved for variables declared over *Nat*, and the letters $k, l, m, n, ...$ range over numerals. Finally, Kleene's primitive recursive *T-predicate* [Kle52, Dav82] is often used. To recall and fix notation: let a coding of Turing Machines (or any other equivalent computing device) be fixed and let $m \geq 1 \in Nat$. Then

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

holds if z codes the unique computation of the Turing Machine encoded by x for arguments $(y_1, ..., y_m)$. For a fixed m, Kleene's T-predicate can be defined in a data specification by a Boolean valued characteristic function. [3] In the remainder, the letter T will always be used for this function (omitting the subscript m). The data part of any specification is interpreted in the canonical term algebra over the domains $D(Nat) = \{0, S(0), ...\}$ and $D(\mathbf{Bool}) = \{\mathbf{t}, \mathbf{f}\}$. So any function declared is regarded as yielding the usual normal forms in the appropriate domain.

[3] Using sequences of naturals as a sort.

3.2. Process Specification

The most simple processes are *(atomic) actions*, which must be explicitly declared in a specification. Actions represent basic activity, and will be associated to the labels of transition systems. Letters $a, b, c, \ldots$ are used to represent actions. Furthermore, actions can be data-parametric. For example, given an action declaration $a, b : Nat$, actions have the form $a(0), \ldots, b(17), \ldots$. It is further assumed that data occurring in actions are always in normal form (by which equality over labels in bisimulations is syntactic equivalence).

More complex processes can be declared by means of (parameterized) process *identifiers*, possibly in a recursive way. For example

$$\begin{aligned} Counter(x) &= p \\ Buffer &= q \end{aligned}$$

In the first line a counter is declared. It is a process with one parameter x of sort *Nat*. The parameter x and the identifier *Counter* may be used in the *process term* p and have no wider scope; p specifies the counter's behaviour. 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 parameterized, or parameterized over *Nat* (so the sort of the variables possibly occurring in process identifiers is always *Nat*).[4] Apart from some expressivity results, all actions considered in this paper are not data-parametric.

In the basic μCRL fragments considered in this paper, process terms may be constructed according to the following syntax:

$$\begin{aligned} p \quad ::= \quad & (p + p) \\ \mid \quad & (p \cdot p) \\ \mid \quad & (p \lhd t \rhd p) \\ \mid \quad & \delta \\ \mid \quad & n \\ \mid \quad & n(t_1, \ldots, t_m) \end{aligned}$$

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—fi expression introduced by Hoare *et al.* [HHJ87] (see also [BaB92]). The data-term t is supposed to be of the standard sort of the Booleans (**Bool**). The left argument is executed if t evaluates to true (**t**) and the right argument is executed if t evaluates to false (**f**). (Recall that all possible functions occurring in t are assumed to be total recursive.) Furthermore, δ 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 $t_1, \ldots, t_m$ are data terms. In process terms, 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 all these operators associate to the right.

A *specification* over the fragment of μCRL used in this paper, is a sequence of data and process declarations (with certain well-formedness criteria, excluding ambiguity in overloading).

[4] In full μCRL, typing of data parameters in process and action declarations is necessary.

In (effective) μCRL, specifications have to be *guarded* as to safeguard that any process term is associated to a (recursive) transition system. Guardedness is an umbrella term for conditions on the way recursion may be used in process declarations (in [BaW90] a common definition and some historical references can be found). Typically, unguarded specifications may either not define any behaviour at all (e.g., the declaration $P = P$), or may have different, uncomparable "solutions". A *syntactically guarded* specification satisfies a syntactic, decidable requirement, that implies guardedness:

Definition 3.2.1 *(Syntactic guardedness).* Let $\mathcal{S}$ be a specification.

1. Let p, q be process terms over $\mathcal{S}$ with p a (parameterized) process identifier. Then *p is (locally) syntactically guarded in q* iff one of the following conditions is satisfied:
   - $q \equiv q_1 + q_2$ and p is syntactically guarded in q_1 and q_2,
   - $q \equiv q_1 \lhd t \rhd q_2$, p is syntactically guarded in q_1 and q_2 (t a Boolean),
   - $q \equiv q_1 \cdot q_2$, and p is syntactically guarded in q_1,
   - q is any action or δ.
2. The specification $\mathcal{S}$ is *(locally) 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').

□

Due to parametrization, "locality" is an issue in the case of μCRL: a single equation can define an infinite number of processes. Note that this is relative to the interpretation of the data involved—in this paper the standard models of the naturals and the Booleans.

Example 3.2.2. Consider the following process declaration:

$$Q(x) \quad = \quad a \cdot \delta \lhd eq(x,0) \rhd a \cdot Q(x \dot{-} 1).$$

This declaration can be associated with a specification that is syntactically guarded by assuming the contents of Table 1 (written with infix notation $\dot{-}$ instead of $monus(,)$), and the declaration of a as an atomic action. For each $k \in Nat$, the process $Q(k)$ behaves as $a^{k+1} \cdot \delta$. *(End example.)*

Syntactic guardedness is a strong requirement on specifications. It implies that each recursively defined process has in each of its states a finite upper bound on the number of actions that can be performed (i.e., on the number of 'outgoing transitions'; see further the next section on operational semantics). Hence, the example above cannot be extended to a syntactically guarded specification that defines a recursive process $P(k)$ behaving like $Q(k) + Q(k \dot{-} 1) + ... + Q(0)$ for each k.

As one of the aims of this paper is to present a simple *and* powerful specification format, syntactic guardedness is relaxed to *lspd-guardedness* — **l**ocal, **s**yntactic guardedness modulo **p**rimitive recursion, relative to a **d**ata interpretation. The following criterion for lspd-guardedness is sufficient, but somewhat ad hoc. Its extra primitive recursive ingredients only are $eq(x,0)$ and $x \dot{-} 1$.

Definition 3.2.3 *(Lspd-guardedness).* Let $\mathcal{S}$ be a specification that contains the data specified in Table 1 (written with infix notation).

1. Let p, q be process terms over $\mathscr{S}$ with p a (parameterized) process identifier. Then p *is lspd-guarded in* q iff one of the following conditions is satisfied:
   - $q \equiv q_1 + q_2$ and p is lspd-guarded in q_1 and q_2,
   - $q \equiv q_1 \lhd t \rhd q_2$, p is lspd-guarded in q_1 and q_2 (t a Boolean),
   - $q \equiv q_1 \cdot q_2$, and p is lspd-guarded in q_1,
   - q is any action or δ,
   - $p \equiv n(x, y_1, ..., y_k)$ and $q \equiv r \lhd eq(x, 0) \rhd n(x \dot{-} 1, y_1 ..., y_k)$, and p is lspd-guarded in r.
2. The specification $\mathscr{S}$ is *lspd-guarded* iff in each of its process declarations, the left-hand side is lspd-guarded in the right-hand side.

□

Lspd-guardedness also is based on the syntax of specifications, and it is a decidable property. In Section 5.1, it is shown that lspd-guardedness is not a restriction in terms of expressivity up to bisimulation equivalence, and in Section 8, a more common definition of guardedness is discussed.

Example 3.2.4. Consider the following process declaration (cf. Example 3.2.2):

$$\begin{aligned} P(x) &= a \cdot \delta \lhd eq(x,0) \rhd a \cdot Q(x \dot{-} 1) + \\ &\quad\ \delta \lhd eq(x,0) \rhd P(x \dot{-} 1) \\ Q(x) &= a \cdot \delta \lhd eq(x,0) \rhd a \cdot Q(x \dot{-} 1). \end{aligned}$$

This declaration can be associated with a specification that is lspd-guarded by assuming the contents of Table 1, and the declaration of a as an atomic action. The process $P(k)$ behaves as $a \cdot Q(k \dot{-} 1) + a \cdot Q(k \dot{-} 2) + ... + a \cdot Q(0) + a \cdot \delta$, i.e. as $Q(k) + ... + Q(0)$. In the next section, this example is continued.
(End example.)

Now the fragments of effective μCRL 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 3.2.5. A specification $\mathscr{S}$ belongs to $\mu\mathrm{CRL}_{\mathrm{TREC}}$ ($\mu\mathrm{CRL}_{\mathrm{PRIM}}$, respectively) iff

- $\mathscr{S}$ contains the data specified in Table 1, and all other functions in $\mathscr{S}$ are total recursive (primitive recursive, respectively),
- $\mathscr{S}$ is lspd-guarded.

□

In the sequel specifications are abbreviated by only describing the occurring process declarations, and even these in an informal way: the restriction to lspd-guardedness is relaxed in favour of readability. For example, given total recursive functions f and g, the specification

$$\begin{aligned} A &= B(f(3)) \\ B(x) &= C(f(x), g(x)) \\ C(x,y) &= a \cdot B(g(x)) \lhd eq(f(x), y) \rhd C(x \dot{-} 1, y) \end{aligned}$$

informally introduces process identifiers A and $B(x)$. In this example, the declaration of $B(x)$ serves as an abbreviation for the less readable formal declaration of $C(x, y)$, which is the following:

$$C(x, y) = a \cdot C(f(g(x)), g(g(x))) \lhd eq(f(x), y) \rhd C(x \dot{-} 1, y)$$

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

4. Operational Semantics for μCRL$_{\text{TREC}}$ and μCRL$_{\text{PRIM}}$

In this section, the interpretation of the process part of a specification is defined. This is based on the interpretation of data, mentioned in Section 3.1. A so called Structured Operational Semantics (SOS) is defined. This approach is based on the work of Plotkin [Plo81], and associates a transition system to each process (closed term) defined in some specification. The states of such a transition system are closed process terms or $\surd$ (expressing successful termination), and its transitions are defined by conditional rules, based on structural induction. Some general references to SOS and bisimulation are [Sim85, GrV92, Vaa93].

4.1. Transition Systems

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

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

where for a specification $\mathcal{S}$ over μCRL$_{\text{TREC}}$,

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

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

$$SOS(\mathcal{S})(p) = (\mathbb{P}(\mathcal{S}) \cup \{\surd\}, \mathbb{A}(\mathcal{S}) \cup \{\surd\}, Tr(\mathcal{S}), p)$$

with the transition relation $Tr(\mathcal{S})$ defined by the transition rules below, where

- variables x, y, z range over $\mathbb{P}(\mathcal{S})$ and primed variables x', y' over $\mathbb{P}(\mathcal{S}) \cup \{\surd\}$,
- in the rule introducing $\lhd t \rhd$, t must be a Boolean declared in $\mathcal{S}$,
- in the rule introducing recursively defined processes, the notation $\mathcal{S} \vdash P = x$ refers to a process declaration: $\mathcal{S} \vdash P = x$ iff $P = x$ is a closed instance of a process declaration in $\mathcal{S}$, in which all data are in normal form.

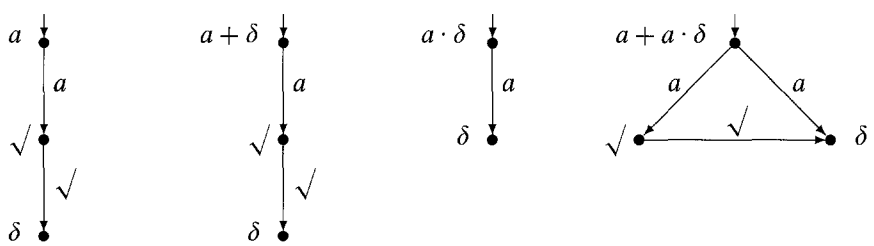
successful termination	$\surd \overset{\surd}{\longrightarrow} \delta$	
$a \in \mathbb{A}(\mathscr{S})$	$a \overset{a}{\longrightarrow} \surd$	
$+$	$\dfrac{x \overset{a}{\longrightarrow} x'}{x+y \overset{a}{\longrightarrow} x'}$	$\dfrac{y \overset{a}{\longrightarrow} y'}{x+y \overset{a}{\longrightarrow} y'}$
	$\dfrac{x \overset{a}{\longrightarrow} y}{x \cdot z \overset{a}{\longrightarrow} y \cdot z}$	$\dfrac{x \overset{a}{\longrightarrow} \surd}{x \cdot y \overset{a}{\longrightarrow} y}$
$\lhd t \rhd$	$\dfrac{x \overset{a}{\longrightarrow} x'}{x \lhd t \rhd y \overset{a}{\longrightarrow} x'}$ if $t = \mathbf{t}$	$\dfrac{y \overset{a}{\longrightarrow} y'}{x \lhd t \rhd y \overset{a}{\longrightarrow} y'}$ if $t = \mathbf{f}$
recursion	$\dfrac{\mathscr{S} \vdash P = x \quad x \overset{a}{\longrightarrow} x'}{P \overset{a}{\longrightarrow} x'}$	

Typically, a transition $\surd \overset{\surd}{\longrightarrow} \delta$ signals *successful termination*, and a transition $x \overset{a}{\longrightarrow} \delta$ models *deadlock* ($a \in \mathbb{A}(\mathscr{S})$). Note that neither δ, nor $\delta \cdot x$ has outgoing transitions. The following example illustrates the transition rules:

Example 4.1.1. Recall the specification $\mathscr{S}$ defined in Example 3.2.4:

$$\begin{aligned} P(x) &= a \cdot \delta \lhd eq(x,0) \rhd a \cdot Q(x \dot{-} 1) + \\ &\quad\ \delta \lhd eq(x,0) \rhd P(x \dot{-} 1) \\ Q(x) &= a \cdot \delta \lhd eq(x,0) \rhd a \cdot Q(x \dot{-} 1). \end{aligned}$$

So, $a \in \mathbb{A}(\mathscr{S})$ is a $\mathbb{P}(\mathscr{S})$ term. With $a \overset{a}{\longrightarrow} \surd$ and the rules for $+$ and $\cdot$ it follows that $a + \delta \overset{a}{\longrightarrow} \surd$ and $a \cdot \delta \overset{a}{\longrightarrow} \delta$. The root connected parts of the transition systems associated to a, $a + \delta$, $a \cdot \delta$ and $a + a \cdot \delta$ can be visualized as follows (observe that only the two leftmost transition systems are deadlock free):



The root connected transitions of $P(n)$ can be derived in the following way. With the rules for $\lhd t \rhd$ it follows that $a \cdot \delta \lhd eq(0,0) \rhd a \cdot Q(0) \overset{a}{\longrightarrow} \delta$. With the rule for $+$ it follows that

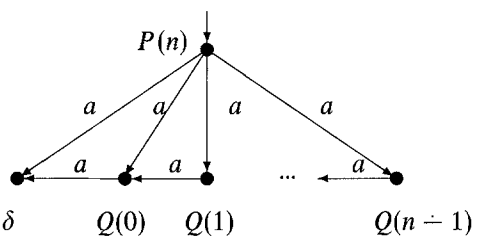
$$a \cdot \delta \lhd eq(0,0) \rhd a \cdot Q(0) + \delta \lhd eq(0,0) \rhd P(0) \overset{a}{\longrightarrow} \delta.$$

Because $\mathscr{S} \vdash P(0) = a \cdot \delta \lhd eq(0,0) \rhd a \cdot Q(0) + \delta \lhd eq(0,0) \rhd P(0)$, the recursion rule yields $P(0) \overset{a}{\longrightarrow} \delta$. The process $P(1)$ has by its last summand all transitions

of $P(0)$. By its first summand there is an a-transition to $Q(0)$, which has an a-transition to δ. In this way one can derive:

$$P(0) \xrightarrow{a} \delta,$$
$$P(1) \xrightarrow{a} \delta, \quad P(1) \xrightarrow{a} Q(0) \xrightarrow{a} \delta,$$
$$P(2) \xrightarrow{a} \delta, \quad P(2) \xrightarrow{a} Q(0) \xrightarrow{a} \delta, \quad P(2) \xrightarrow{a} Q(1) \xrightarrow{a} Q(0) \xrightarrow{a} \delta.$$

Clearly, $SOS(\mathcal{S})(P(n))$ consists of $n+1$ paths a^{k+1} to δ for all $k \leq n$, and its root connected part can be depicted as follows (for $n \geq 3$):



(End example.)

4.2. SOS and Computability

For any $\mu\mathrm{CRL}_{\mathrm{TREC}}$ specification $\mathcal{S}$ it is the case that $SOS(\mathcal{S})$ yields (isomorphic images of) *recursive* transition systems. (In fact this is the case for the standard operational semantics of *effective* μCRL [GrP91a, GrP95], of which *SOS* is the restriction to $\mu\mathrm{CRL}_{\mathrm{TREC}}$ specifications):

Theorem 4.2.1. Let p be some process specified over a specification $\mathcal{S}$ in $\mu\mathrm{CRL}_{\mathrm{TREC}}$, then $SOS(\mathcal{S})(p)$ is recursively isomorphic with a recursive transition system. Moreover, this transition system is termination consistent (see Definition 2.1.3).

Proof. First note that $SOS(\mathcal{S})(p)$ is finitely branching: by lspd-guardedness any closed process term q can be equated to a term (using the process declarations in $\mathcal{S}$) for which the next steps do not depend on terms headed by process identifiers. It follows from the calculus for $Tr(\mathcal{S})$ 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 $\mathbb{P}(\mathcal{S}) \cup \{\surd\}$ can be recursively encoded as recursive sets (even as primitive recursive 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

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

for this coding. Using the calculus for $Tr(\mathcal{S})$, define a total recursive 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 total recursive as it must be able to compute the total recursive functions defined in $\mathcal{S}$. Given p as in the theorem, define a coding

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

by $\llcorner p \lrcorner = 0$ (the root) 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

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

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

Finally, it follows immediately from the rules of $Tr(\mathscr{S})$ that both these transition systems are termination consistent. □

Corollary 4.2.2. Let p be some process specified over a specification $\mathscr{S}$ in $\mu\mathrm{CRL}_{\mathrm{PRIM}}$, then $SOS(\mathscr{S})(p)$ is primitive recursively isomorphic with a primitive recursive transition system.

Another important (algebraic) property of $SOS(\mathscr{S})$ is that the bisimilarity induced by it is a congruence with respect to the process operators of $\mu\mathrm{CRL}_{\mathrm{TREC}}$ [GrP91a, GrP95].

A relevant question is whether $SOS(\mathscr{S})$ itself is 'nice enough' as an operational semantics. Are there no effective semantics for $\mathscr{S}$ that respect $SOS(\mathscr{S})$ up to bisimulation equivalence, and that yield smaller transition systems (in terms of number of states and transitions), in particular 'minimal' transition systems? Generally this is not the case, not even for the restriction to $\mu\mathrm{CRL}_{\mathrm{PRIM}}$ specifications, as this problem easily reduces to the Halting Problem.

Theorem 4.2.3. There exists a $\mu\mathrm{CRL}_{\mathrm{PRIM}}$ specification $\mathscr{S}$ for which no effective operational semantics yields minimal transition systems.

Proof. Consider the $\mu\mathrm{CRL}_{\mathrm{PRIM}}$ specification $\mathscr{S}$ defined by

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

Let k, l be fixed.

In the case that $\neg\exists z \,.\, T(k,l,z)$, the finite transition system $\mathscr{T}_2$ defined in Example 2.1.2 is bisimilar with $SOS(\mathscr{S})(K(k,l,0))$, while $SOS(\mathscr{S})(K(k,l,0))$ is isomorphic with $\mathscr{T}_1$ in that example.

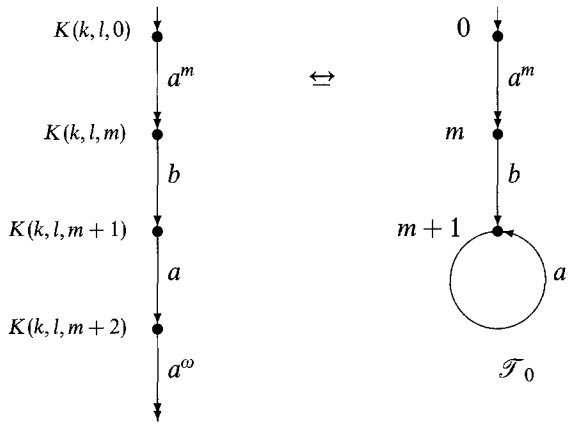
Assume $T(k,l,m)$ for some (unique) m. In this case consider the finite transition system

$$\mathscr{T}_0 \stackrel{def}{=} (\{0,1,...,m+1\}, \{a,b\}, Tr, 0)$$

with the transition relation

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

The root connected part of $SOS(\mathscr{S})(K(k,l,0))$ and $\mathscr{T}_0$ can be depicted as follows:



Obviously, $\mathscr{T}_0 \leftrightarrows SOS(\mathscr{S})(K(k,l,0))$, and $\mathscr{T}_0$ is smallest.

Hence for each pair k,l it follows that $SOS(\mathscr{S})(K(k,l,0))$ is regular. This example shows that an operational semantics for $\mathscr{S}$ yielding transition systems with minimal sets of states (and labels) must be able to decide for each k,l whether $\exists z \,.\, T(k,l,z)$—i.e., the Halting Problem—, so cannot be effective. □

5. Expressivity of μCRL$_{\text{TREC}}$ and μCRL$_{\text{PRIM}}$

In this section a relation between recursive transition systems and μCRL$_{\text{TREC}}$ as a language for specifying these up to bisimulation equivalence is established. First, it is shown that the selected fragments of μCRL have universal expressivity over the two selected domains of transition systems—recursive and primitive recursive—if one restricts to actions as labels (and possibly the successful termination label $\surd$, see Definition 2.1.3). To provide an immediate correspondence with CCS and LOTOS, a distinction is made between CCS-like transition systems not containing $\surd$, and ACP-like ones in which both successful termination and deadlock can be modelled. Then it is shown that also modulo bisimulation, the two selected domains are different.

5.1. Universal Expressivity

It is first shown that each (primitive) recursive transition system over a recursive set of actions as labels can be represented by a μCRL$_{\text{TREC}}$ process term (respectively a process specified over μCRL$_{\text{PRIM}}$). The resulting specification is CCS-like—based on the correspondence of $a \cdot \delta$ with $a.\mathbf{0}$ in CCS.

Theorem 5.1.1. Let L be a recursive set of actions. Each recursive transition system over L can be specified up to bisimulation equivalence in a μCRL$_{\text{TREC}}$ specification. If L is primitive recursive, this can be done in μCRL$_{\text{PRIM}}$.

Proof. Let $\mathscr{T} = (S, L, next, i)$ and let $Action(x)$ be a process term that *decodes* the actions encoded by i. For example, if L contains action names $a_0, ..., a_m$ of which some are parameterized over Nat, and i is defined by

$$\begin{aligned} i(a_k(x)) &\stackrel{def}{=} j(k, x+1) && \text{(in case } a_k \text{ is data-parametric)} \\ i(a_k) &\stackrel{def}{=} j(k, 0) && \text{(in case } a_k \text{ is not data-parametric),} \end{aligned}$$

then

$$\begin{aligned} Action(x) = \; & a_0(j_2(x) \dot{-} 1) \lhd eq(j_1(x), 0) \wedge \neg eq(j_2(x), 0) \rhd \delta + \\ & a_0 \lhd eq(j_1(x), 0) \wedge eq(j_2(x), 0) \rhd \delta + \\ & \vdots \\ & a_m(j_2(x) \dot{-} 1) \lhd eq(j_1(x), m) \wedge \neg eq(j_2(x), 0) \rhd \delta + \\ & a_m \lhd eq(j_1(x), m) \wedge eq(j_2(x), 0) \rhd \delta. \end{aligned}$$

So for any $k \in Nat$, $Action(k)$ is a process term of $2m$ summands, of which at most one is an action from L and all others equal δ.

Consider the process P specified in the following specification $\mathscr{S}$ (recall that in a recursive transition system the initial state is 0):

$$\begin{aligned} P &= Q(0) \\ Q(x) &= R(next(x), next(x)) \\ R(x, y) &= Action(j_1(x)) \lhd x \in D_y \rhd \delta + \\ &\quad \delta \lhd eq(x, 0) \rhd R(x \dot{-} 1, y). \end{aligned}$$

By definition of the function $next$, the property $n \in D_m \implies n < m$, and by unraveling the specification of P it follows that $SOS(\mathscr{S})(P) \leftrightarrows \mathscr{T}$. Note that:

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

□

This result can be generalized to termination consistent transition systems (in which successful termination and deadlock are distinguished, Definition 2.1.3):

Theorem 5.1.2. Let L be a recursive set of actions. Each termination consistent recursive transition system over $L \cup \{\surd\}$ can be specified up to bisimulation equivalence in a $\mu\mathrm{CRL}_{\mathrm{TREC}}$ specification. If L is primitive recursive, this can be done in $\mu\mathrm{CRL}_{\mathrm{PRIM}}$.

Proof. Let $\mathscr{T} = (S, L \cup \{\surd\}, next, i)$ and let $Action(x)$ be defined as in the proof above. Now successful termination states have to be distinguished from the others.

Consider the process P specified in the following specification $\mathscr{S}$:

$$
\begin{aligned}
P &= Q(0)\\
Q(x) &= R(next(x), next(x))\\
R(x,y) &= Action(j_1(x)) \cdot Q(j_2(x)) \lhd x \in D_y \wedge NoTick(next(j_2(x))) \rhd \delta +\\
&\quad \delta \lhd eq(x,0) \rhd R(x \dot{-} 1, y) +\\
&\quad Action(j_1(x)) \lhd x \in D_y \wedge Tick(next(j_2(x))) \rhd \delta
\end{aligned}
$$

where the primitive recursive predicates *NoTick* (modelling the absence of successful termination) and *Tick* (modelling successful termination) are defined by

$$
\begin{aligned}
NoTick(x) &= \forall z \leq x \,.\, \neg(z \in D_x \wedge eq(j_1(z), i(\surd)))\\
Tick(x) &= \exists z \leq x \,.\, z \in D_x \wedge eq(j_1(z), i(\surd)).
\end{aligned}
$$

Then $SOS(\mathscr{S})(p) \leftrightarrows \mathscr{T}$. This can be argued in the same way as in the proof of Theorem 5.1.1 because all conditions are still total recursive (primitive recursive, respectively). Another difference with the specification in that proof is the last summand of $R(x,y)$, which possibly generates successful termination states. □

For *primitive* recursive transition systems over a label set not including $\surd$ there is the following corollary:

Corollary 5.1.3. (Cf. [Gla95].) Let L be a primitive recursive set of actions. Each primitive recursive transition system over L can be specified up to bisimulation equivalence in one single $\mu\mathrm{CRL}_{\mathrm{TREC}}$ specification.

Proof. Let a primitive recursive label coding i of L be fixed and $Action(x)$ be defined as in the preceding proofs. Code all unary primitive recursive functions, and define a total recursive function *Eval* that satisfies

$$Eval(k,x) = f(x)$$

if k is the code of f, and 0 if k does not code any primitive recursive function.[5] Now consider the following specification (cf. the one in the proof of Theorem 5.1.1):

$$
\begin{aligned}
P_{prim}(z) &= Q_{prim}(0,z)\\
Q_{prim}(x,z) &= R_{prim}(Eval(z,x), Eval(z,x), z)\\
R_{prim}(x,y,z) &= Action(j_1(x)) \cdot Q_{prim}(j_2(x), z) \lhd x \in D_y \rhd \delta +\\
&\quad \delta \lhd eq(x,0) \rhd R_{prim}(x \dot{-} 1, y, z).
\end{aligned}
$$

Then, as follows from the proof of Theorem 5.1.1, $P_{prim}(k)$ specifies the primitive recursive transition system for which k encodes the *next* function. Furthermore, as k varies, each primitive recursive transition system over L is specified up to bisimilarity. □

There is no generalization of this corollary to primitive recursive ACP-like transition systems (involving $\surd$) because termination consistency is not decidable (otherwise, for each $l \in L$, the set $\{(s,s') \mid s \xrightarrow{l} s'\}$ is decidable, which is a contradiction, cf. the proof of Theorem 4.2.3).

[5] Recall that *Eval* itself cannot be primitive recursive (otherwise $g(x) = Eval(x,x) + 1$ would be as well, say with code l, and $Eval(l,l) = g(l) = Eval(l,l) + 1$, contradiction).

So each (primitive) recursive transition system $\mathscr{T} = (S, L, next, i)$ with L finite and $\mathscr{T}$ termination consistent, gives rise to a specification $\mathscr{S}$ over $\mu\mathrm{CRL}_{\mathrm{TREC}}$ (or $\mu\mathrm{CRL}_{\mathrm{PRIM}}$). Such a specification thus defines a *canonical* process term P for which $SOS(\mathscr{S})(P) \leftrightarrows \mathscr{T}$. In Section 8 more on canonicity.

5.2. Difference between $\mu\mathrm{CRL}_{\mathrm{TREC}}$ and $\mu\mathrm{CRL}_{\mathrm{PRIM}}$

As can be expected, the restriction to primitive recursive 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 primitive recursive function (but total recursive) can be used to define a process that has a 'branching degree' growing faster than any $\mu\mathrm{CRL}_{\mathrm{PRIM}}$ specification can handle:

Theorem 5.2.1. There is a recursive transition system over a finite set of labels that cannot be specified modulo bisimulation in $\mu\mathrm{CRL}_{\mathrm{PRIM}}$.

Proof. Consider the following function *Ack*, (a version of) the Ackermann generalized exponential [Kle52, Rog67], which is total recursive but not primitive recursive:

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

Let the specification $\mathscr{S}$ over $\mu\mathrm{CRL}_{\mathrm{TREC}}$ be defined by

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

Now assume $\mathscr{T} = (S, \{a,b,c\}, next, i)$ is a primitive recursive transition system that satisfies $\mathscr{T} \leftrightarrows SOS(\mathscr{S})(P(0))$. Let $F : Nat \to Nat$ be such that $j_1(F(k))$ is the code of the state characterized by the trace a^k and $j_2(F(k))$ is the code of the state characterized by the trace $a^k \cdot b$. To make this precise, let G^l be a function that satisfies

$$G^l(next(s)) = \begin{cases} s' & \text{if } s \xrightarrow{l} s' \\ 0 & \text{otherwise,} \end{cases}$$

i.e., $G^l(x) = j_2(\mu y < x.[y \in D_x \wedge j_1(y) = i(l)])$. Then F can be defined by primitive recursion:

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

So F is primitive recursive 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 successive c-transitions). Hence, the CI of the set coding all these labels and resulting states is certainly larger than $Ack(k,k)$ by which *next* cannot be a *primitive* recursive function, contradicting the assumption. So $SOS(\mathscr{S})(P(0))$

is a recursive transition system over a finite label set that is not bisimilar with any primitive recursive transition system. □

6. Arithmetical Classification of Properties

In this section, the properties of (pairs of) recursive transition systems that were introduced in Section 1 are defined and classified in the arithmetical hierarchy [Rog67]. The arithmetical characterization takes place on the level of $\mu\mathrm{CRL}_{\mathrm{TREC}}$ and $\mu\mathrm{CRL}_{\mathrm{PRIM}}$ specifications. By the Expressivity Theorems 5.1.1 and 5.1.2 this is sufficient. Moreover, these properties are more apparent in a process term than in a representation of the form $\mathcal{T} = (S, L, next, i)$. This is useful for completeness results.

6.1. Characterizing Properties with Process Algebra

Properties of recursive transition systems have to be invariant under bisimulation equivalence, as it are such equivalence classes that represent 'operational behaviour'. Typically, neither *number of states*, nor *being a tree* is a bisimulation invariant property (even not for the restriction to root connected transition systems), as was illustrated in Example 2.1.2. Obviously, all properties considered refer to the root connected part of transition systems.

A first property is *bisimilarity* itself (because $\leftrightarrows$ is an equivalence relation, it is a property over two transition systems that is bisimulation invariant). A transition system is *perpetual* if each root-connected state has at least one outgoing transition. A transition system is *regular* if it is bisimilar with some finite transition system. A transition system is *acyclic regular* if it is bisimilar with a finite acyclic transition system. A transition system is *deadlock free* if it is has no deadlock states, i.e., states that are root connected by transitions not having the label $\surd$, and that have no outgoing transitions (see Definition 2.1.3).

These properties can easily be characterized by means of *basic process algebra* and *projections*. Given a $\mu\mathrm{CRL}_{\mathrm{TREC}}$ specification $\mathcal{S}$, the set of $\mathrm{BPA}_\delta(\mathcal{S})$*-terms* over $\mathcal{S}$ consists of the process terms that can be constructed out of δ, the actions declared in $\mathcal{S}$ (a recursive set), and the operators $+$ and $\cdot$. So $\mathrm{BPA}_\delta(\mathcal{S})$ characterizes the class of acyclic regular transition systems over $\mathbb{A}(\mathcal{S}) \cup \{\surd\}$ up to bisimulation equivalence.

The coming proofs employ some standard ACP-results, all of which can be found in for instance [BaW90, BaV95]. The basic axiom system BPA_δ (Basic Process Algebra with δ) consists of the axioms in Table 2.

The following completeness result is standard: for any two $\mathrm{BPA}_\delta(\mathcal{S})$-terms

Table 2. The axioms of BPA_δ.

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

p, q it holds that

$$\mathrm{BPA}_\delta(\mathscr{S}) \vdash p = q \iff SOS(\mathscr{S})(p) \leftrightarrows SOS(\mathscr{S})(q).$$

In this section we further assume that $\ulcorner . \urcorner$ is a primitive recursive coding of $\mathbb{P}(\mathscr{S})$ such that

- the set of codes, say $\ulcorner \mathbb{P}(\mathscr{S}) \urcorner$, is primitive recursive and $0 \notin \ulcorner \mathbb{P}(\mathscr{S}) \urcorner$,
- the code of a term is larger than those of its proper subterms,
- if $\ulcorner p_i \urcorner < \ulcorner q_i \urcorner$, then $\ulcorner p_1 + p_2 \urcorner < \ulcorner q_1 + q_2 \urcorner$ and $\ulcorner p_1 \cdot p_2 \urcorner < \ulcorner q_1 \cdot q_2 \urcorner$.

(For a syntactic coding, these requirements are easy to meat.)

The relation $Eq \subseteq Nat \times Nat$ defined by

$$Eq \stackrel{def}{=} \{(\ulcorner p \urcorner, \ulcorner q \urcorner) \mid p, q \in \mathrm{BPA}_\delta(\mathscr{S}) \text{ and } \mathrm{BPA}_\delta(\mathscr{S}) \vdash p = q\}$$

is primitive recursive: $\mathrm{BPA}_\delta(\mathscr{S})$ terms can effectively be reduced to normal forms modulo commutativity and associativity of the $+$ (see e.g., [BaW90, BaV95]). Furthermore, the relation $Df \subseteq Nat$ (Deadlock-free) over codes of $\mathrm{BPA}_\delta(\mathscr{S})$ terms defined by

$$Df \stackrel{def}{=} \{x \mid \exists y \leq x \,.\, Eq(x, y) \wedge \delta\text{-}free(y)\}$$

is a primitive recursive relation (where $\delta\text{-}free(y)$ holds iff δ does not occur in the term coded by y).

Because the booleans occurring in $\mathbb{P}(\mathscr{S})$ can be computed effectively to either $\mathbf{t}$ or $\mathbf{f}$, terms $p \lhd t \rhd q$ can be reduced (in terms of obtaining a smaller code) to either p or q with the axioms

$$x \lhd \mathbf{t} \rhd y = x \qquad \text{and} \qquad x \lhd \mathbf{f} \rhd y = y$$

(this reduction need not be primitive recursive, as it may involve evaluation of total recursive functions).

Furthermore, projections of the processes definable over $\mathscr{S}$ shall be used: $\pi_n(p)$ is the process that can perform the first n steps of p, and the π_n operators are axiomatized in Table 3. By the lspd-guardedness of $\mathscr{S}$, each $\pi_n(p)$ can be reduced effectively to a $\mathrm{BPA}_\delta(\mathscr{S})$ process: the total recursive functions in $\mathscr{S}$ used in data-parametric recursion and conditionals must be computed. For example, given $\mathscr{S}$ as in Examples 3.2.4 and 4.1.1, i.e.,

$$\begin{aligned} P(x) &= a \cdot \delta \lhd eq(x, 0) \rhd a \cdot Q(x \dot{-} 1) + \\ &\quad\ \delta \lhd eq(x, 0) \rhd P(x \dot{-} 1) \\ Q(x) &= a \cdot \delta \lhd eq(x, 0) \rhd a \cdot Q(x \dot{-} 1), \end{aligned}$$

one can derive

Table 3. Projection axioms, where a is an action or δ and $n > 0$.

(PR1)	$\pi_n(a)$	$=$	a	(PR3)	$\pi_{n+1}(a \cdot x)$	$=$	$a \cdot \pi_n(x)$
(PR2)	$\pi_1(a \cdot x)$	$=$	a	(PR4)	$\pi_n(x + y)$	$=$	$\pi_n(x) + \pi_n(y)$

$$\begin{aligned}\pi_2(P(2)) &= \pi_2(a\cdot\delta \lhd eq(2,0)\rhd a\cdot Q(2\dot{-}1)+\delta \lhd eq(2,0)\rhd P(2\dot{-}1))\\ &= \pi_2(a\cdot Q(1))+\pi_2(P(1))\\ &= a\cdot\pi_1(Q(1))+\pi_2(a\cdot Q(0))+\pi_2(P(0))\\ &\vdots\\ &= a\cdot a+a\cdot a+a\cdot\delta.\end{aligned}$$

As a consequence, the following properties (cf. Section 6.1) can easily be characterized with projections and equality in $\mathrm{BPA}_\delta(\mathscr{S})$ (i.e., derivability or bisimilarity):

- $SOS(\mathscr{S})(p) \leftrightarrows SOS(\mathscr{S})(q)$, i.e. *bisimilarity* [6] — $\forall n>0\,.\,\pi_n(p)=\pi_n(q)$.
- $SOS(\mathscr{S})(p)$ is *deadlock free* — $\forall n>0\,.\,Df(\ulcorner\pi_n(p)\urcorner)$.
- $SOS(\mathscr{S})(p)$ is *perpetual* — $\forall n>0\,.\,[Df(\ulcorner\pi_n(p)\urcorner)\ \&\ \pi_n(p)=\pi_n(\pi_n(p)\cdot p)]$.
- $SOS(\mathscr{S})(p)$ is *acyclic regular* — $\exists n>0\,.\,\pi_n(p)=\pi_{n+1}(p)$.

For *regularity*, the characterization is based on projections *and* a coding of finite transition systems (see next section).

In the classification, the following primitive recursive function Pr (projection on codes of $\mathbb{P}(\mathscr{S})$) shall be used.

$$Pr(x,y)=\begin{cases}\ulcorner\pi_{x+1}(p)\urcorner & \text{if } \ulcorner p\urcorner = y\\ 0 & \text{otherwise.}\end{cases}$$

6.2. Classification

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

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

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

Now E_n is complete in Σ^0_n, and $\neg E_n$ —the complement of E_n— is complete in Π^0_n.

Given a certain class $\mathscr{C}$ in the arithmetical hierarchy and one of the properties, completeness in $\mathscr{C}$ for both $\mu\mathrm{CRL}_{\mathrm{PRIM}}$ and $\mu\mathrm{CRL}_{\mathrm{TREC}}$ is proved simultaneously in the following way:

1. Show that for any $\mu\mathrm{CRL}_{\mathrm{TREC}}$ specification the property is equivalent to a relation in $\mathscr{C}$ (based on a primitive recursive relation when restricting to $\mu\mathrm{CRL}_{\mathrm{PRIM}}$).
2. Show the *completeness* in $\mathscr{C}$ by giving a particular $\mu\mathrm{CRL}_{\mathrm{PRIM}}$ specification for which the property is equivalent to a relation that is complete in $\mathscr{C}$.

In the rest of this section, the five properties mentioned above are characterized in the arithmetical hierarchy. Given a $\mu\mathrm{CRL}_{\mathrm{TREC}}$ specification $\mathscr{S}$, let $\ulcorner.\urcorner$, Eq, Df and Pr be defined as in the previous section.

[6] In terms of the theory of ACP, this equivalence comprises application of the Approximation Induction Principle (AIP^-) (see e.g. [BBK87, BaW90, BaV95]). This principle implies that two finitely branching transition systems are bisimilar if all their projections are.

Bisimilarity — complete in Π_1^0.

1. Let $\mathcal{S}$ in $\mu\mathrm{CRL}_{\mathrm{TREC}}$ be given and $p, q \in \mathbb{P}(\mathcal{S})$. Then $SOS(\mathcal{S})(p) \leftrightarrows SOS(\mathcal{S})(q)$ iff
$$(\ulcorner p \urcorner, \ulcorner q \urcorner) \in \{(x, y) \mid \forall z \,.\, Eq(Pr(z, x), Pr(z, y))\}.$$
Hence, bisimilarity between two processes over $\mathcal{S}$ is in the class Π_1^0.
2. *Completeness* in Π_1^0 can be proved with the following specification $\mathcal{S}_p$ over $\mu\mathrm{CRL}_{\mathrm{PRIM}}$:
$$\begin{aligned} K(x, y, z) &= b \cdot \delta \lhd T(x, y, z) \rhd a \cdot K(x, y, z + 1) \\ M &= a \cdot M. \end{aligned}$$
Now $SOS(\mathcal{S}_p)(K(k, l, 0)) \leftrightarrows M \iff \forall z \,.\, \neg T(k, l, z)$
$\iff (k, l) \in \neg E_1$.
The latter problem is complete in Π_1^0.

Deadlock Freedom — complete in Π_1^0.

1. Let $\mathcal{S}$ in $\mu\mathrm{CRL}_{\mathrm{TREC}}$ be given and $p \in \mathbb{P}(\mathcal{S})$. Then $SOS(\mathcal{S})(p)$ is deadlock free iff
$$\ulcorner p \urcorner \in \{x \mid \forall y \,.\, Df(Pr(y, x))\}.$$
Hence, this property is in the class Π_1^0.
2. Deadlock freedom is *complete* in Π_1^0 by the bisimilarity example above:
$$SOS(\mathcal{S}_p)(K(k, l, 0)) \text{ is deadlock free} \iff \forall z \,.\, \neg T(k, l, z)$$
$$\iff (k, l) \in \neg E_1.$$

Perpetuity — complete in Π_1^0.

1. Let $\mathcal{S}$ in $\mu\mathrm{CRL}_{\mathrm{TREC}}$ be given and $p \in \mathbb{P}(\mathcal{S})$. Two cases can be distinguished. In the degenerated case that $\mathcal{S}$ contains no actions, perpetuity is not meaningful (only the bisimilarity class of δ is definable). Otherwise, let a be an action of $\mathcal{S}$, and let the function *Pr-a* (projection followed by a) be defined as follows:
$$Pr\text{-}a(x, y) = \begin{cases} \ulcorner \pi_{x+1}(\pi_{x+1}(p) \cdot a) \urcorner & \text{if } \ulcorner p \urcorner = y \\ 0 & \text{otherwise.} \end{cases}$$
Then $SOS(\mathcal{S})(p)$ is perpetual iff
$$\ulcorner p \urcorner \in \{x \mid \forall y \,.\, Eq(Pr(y, x), Pr\text{-}a(y, x)) \wedge Df(x)\}.$$
Hence this property is in the class Π_1^0.
2. Perpetuity is *complete* in Π_1^0 using the bisimilarity example above:
$$SOS(\mathcal{S}_p)(K(k, l, 0)) \text{ is perpetual} \iff \forall z \,.\, \neg T(k, l, z)$$
$$\iff (k, l) \in \neg E_1.$$

Acyclic regularity — complete in Σ_1^0.

1. Let $\mathcal{S}$ in $\mu\mathrm{CRL}_{\mathrm{TREC}}$ be given and $p \in \mathbb{P}(\mathcal{S})$. Then $SOS(\mathcal{S})(p)$ is acyclic regular iff
$$\ulcorner p \urcorner \in \{x \mid \exists y \,.\, Eq(Pr(y, x), Pr(y + 1, x))\}.$$
Hence this property is in the class Σ_1^0.

2. This property is *complete* in Σ_1^0 using the example above:

$$\begin{array}{lll} SOS(\mathscr{S}_p)(K(k,l,0)) \text{ is acyclic regular} & \Longleftrightarrow & \exists z \,.\, T(k,l,z) \\ & \Longleftrightarrow & (k,l) \in E_1. \end{array}$$

Regularity — complete in Σ_2^0.

1. Let $\mathscr{S}$ in $\mu\mathrm{CRL}_{\mathrm{TREC}}$ be given and $p \in \mathbb{P}(\mathscr{S})$. Assume some coding of the termination consistent (see Definition 2.1.3), finite transition systems over $\mathbb{A}(\mathscr{S})$ or, if one likes, over $\mathbb{A}(\mathscr{S}) \cup \{\surd\}$, and a (primitive recursive) relation *Fin* that characterizes these. For example, $n \in Fin$ iff $j_2(n)$ is the CI of codes of transitions $k \xrightarrow{a} l$ with $a \in \mathbb{A}(\mathscr{S})$ and $k, l \leq j_1(n)$. Let the function $Pr' : Nat \times Nat \to Nat$ be such that $Pr'(x,y) = Pr(x,y)$ in case $y \in Fin$, and 0 otherwise. Then $SOS(\mathscr{S})(p)$ is regular iff

$$\ulcorner p \urcorner \in \{x \mid \exists y \forall z \,.\, Eq(Pr(z,x), Pr'(z,y))\}.$$

   Hence, regularity over $\mathscr{S}$ can be defined as a Σ_2^0 relation.

2. For the completeness in Σ_2^0, 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 (cf. the Pumping Theorem for regular languages [HU79]). A $\mu\mathrm{CRL}_{\mathrm{PRIM}}$ specification $\mathscr{S}'_p$ for defining this trace is

$$\begin{array}{lcl} K(v,w,x,y,z) & = & a \cdot L(v,w,x,y,z) \\ & & \quad \lhd eq(z,0) \rhd a \cdot K(v,w,x,y,z \dot{-} 1) \\ L(v,w,x,y,z) & = & b \cdot K(v,w,x+1,0,x+1) \\ & & \quad \lhd T(v,w,x,y) \rhd b \cdot L(v,w,x,y+1,z). \end{array}$$

   Now $SOS(\mathscr{S}'_p)(K(k,l,0,0,0))$ is regular $\Longleftrightarrow$ $\exists x \forall y \,.\, \neg T(k,l,x,y)$ $\Longleftrightarrow$ $(k,l) \in E_2$.

   The latter problem is complete in Σ_2^0.

This section is concluded with some comments on these properties. In [MaM94], it is shown that in $\mathrm{BPA}_\delta(\mathscr{S})$ with (syntactically guarded) recursion and without data, *regularity* of specifications is a decidable property. More decidability results on regularity can be found in [BoG96].

Of course, combinations of properties now can also be classified. For instance, the property *acyclic regularity & deadlock free* is complete in Σ_0^1, as the number of relevant projections is bounded:

$$\ulcorner p \urcorner \in \{x \mid \exists y \,.\, Eq(Pr(y,x), Pr(y+1,x)) \wedge \forall z \leq y \,.\, Df(Pr(z,x))\}.$$

Completeness follows from the $\mu\mathrm{CRL}_{\mathrm{PRIM}}$ specification $\mathscr{S}''$ defined by

$$\begin{array}{lcl} K(x,y,z) & = & (b \lhd Even(z) \rhd b \cdot \delta) \\ & & \quad \lhd T(x,y,z) \rhd a \cdot K(x,y,z+1). \end{array}$$

Now $SOS(\mathscr{S}'')(K(k,l,0))$ is 'acyclic regular & deadlock free' iff $\exists z \,.\, T(k,l,z) \wedge Even(z)$.

7. Restricted Forms of Bisimilarity

In this section different restricted forms of bisimilarity are investigated. The motivation for this is the observation of Bergstra [Ber91] that not each pair of bisimilar primitive recursive transition systems over a *finite* label set can be related by a *recursively enumerable* bisimulation. So even over a relatively simple domain, bisimilarity is a complex relation. Finally, two more forms of bisimilarity are distinguished that are both weaker than bisimilarity defined by the existence of a r.e. bisimulation.

7.1. 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 7.1.1 (*Bergstra* [Ber91]). There are two primitive recursive transition systems over a finite set of actions as 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 $\mathscr{S}$ over $\mu\mathrm{CRL}_{\mathrm{PRIM}}$:

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

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

Then $SOS(\mathscr{S})(A(0))$ and $SOS(\mathscr{S})(B(0))$ are primitive recursive transition systems (cf. Corollary 4.2.2). Observe that any trace of $A(0)$ or $B(0)$ has at most *one* of the $b \cdot \delta$ or $c \cdot \delta$ options. It is proved that

$$SOS(\mathscr{S})(A(0)) \leftrightarrows SOS(\mathscr{S})(B(0))$$

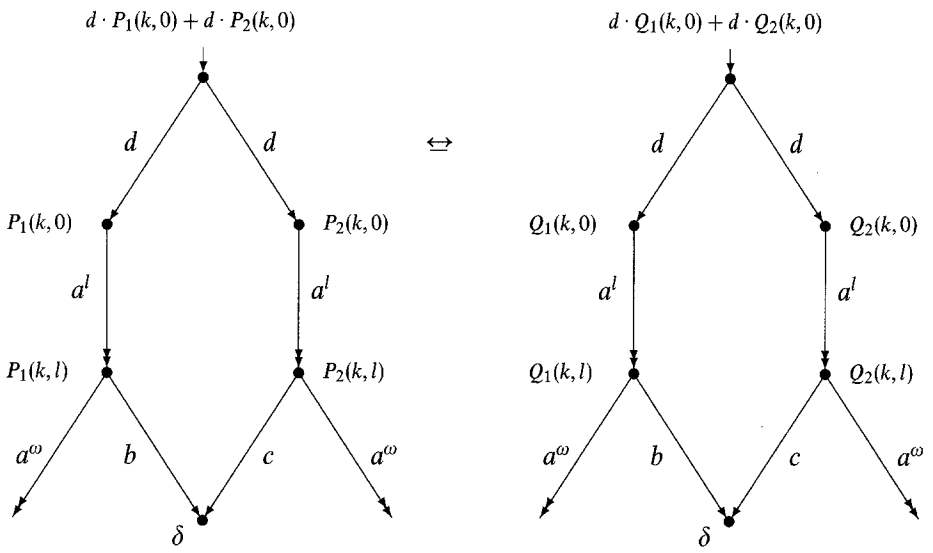
and that each witnessing bisimulation is not recursively enumerable.

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

$$SOS(\mathscr{S})(d \cdot P_1(k,0) + d \cdot P_2(k,0)) \leftrightarrows SOS(\mathscr{S})(d \cdot Q_1(k,0) + d \cdot Q_2(k,0)).$$

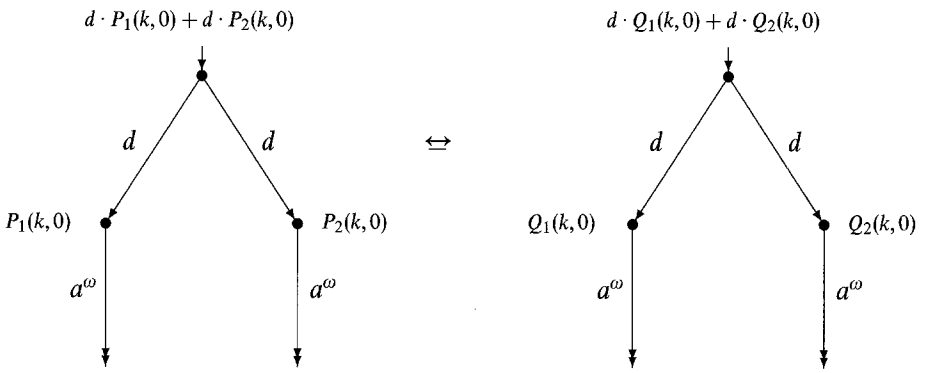
Distinguishing the three cases $k \in W_{e_1}$, $k \in W_{e_2}$ and $k \notin W_{e_1} \cup W_{e_2}$, this can most easily be shown by pictures suggesting the bisimulations to be used.

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 $SOS(\mathscr{S})(d \cdot P_1(k,0) + d \cdot P_2(k,0))$ above should be reversed, and again bisimulation is obvious;

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



So for any $k \in Nat$ it follows that $SOS(\mathscr{S})(A(k)) \leftrightarrows SOS(\mathscr{S})(B(k))$. As bisimilarity is a congruence relation, it follows easily that

$$SOS(\mathscr{S})(A(0)) \leftrightarrows SOS(\mathscr{S})(B(0)).$$

It remains to be shown that any bisimulation relating $SOS(\mathscr{S})(A(0))$ and $SOS(\mathscr{S})(B(0))$ cannot be recursively enumerable. Assume the contrary for a relation S with $\ulcorner S \urcorner \stackrel{def}{=} \{(\ulcorner p \urcorner, \ulcorner q \urcorner) \mid (p,q) \in S\}$, then both

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

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

are also recursively enumerable, assuming that $\ulcorner . \urcorner$ is a function in the style of the proof of Theorem 4.2.1, and $\ulcorner P_1(k) \urcorner$ and $\ulcorner Q_1(k) \urcorner$ are total recursive functions on k. Let for $i = 1, 2$

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

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 5.1.1 and 5.1.2 do not hold modulo $\leftrightarrows_{r.e.}$ (as $\leftrightarrows_{r.e.}$ is a transitive relation).

7.2. Weaker Bisimulations

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

Theorem 7.2.1. There are two primitive recursive transition systems over a finite set of actions as labels that are recursively bisimilar, but not primitive recursively bisimilar.

Proof. Consider the process declarations from the proof of Bergstra's Theorem 7.1.1, but now take W_{e_1} a recursive set that is not primitive recursive, and $W_{e_2} = Nat \setminus W_{e_1}$. Proceeding as in the proof of 7.1.1, it follows that the primitive recursive transition systems $SOS(\mathscr{S})(A(0))$ and $SOS(\mathscr{S})(B(0))$ are recursively bisimilar, but not primitive recursively 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 7.2.2. There are two primitive recursive transition systems over a finite set of actions as 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:

$$\begin{aligned} A &= a \cdot A \\ B(x,y) &= a \cdot B(x, y+1) \; + \\ &\quad b \cdot \delta \lhd \exists y' \leq y \,.\, T(e_2, x, y') \rhd \delta \; + \\ &\quad \Sigma_{x<z<y}(a \cdot B(z,0) \lhd K(x,y) \rhd \delta) \end{aligned}$$

where $K(x,y)$ abbreviates the primitive recursive predicate

$$\exists y' \leq y \,.\, T(e_1,z,y') \;\wedge\; \neg\exists z' < z, y'' < y \,.\, x < z' \wedge T(e_1,z',y'')$$

and a formal, but less readable description of the summand(s)

$$\Sigma_{x<z<y}(a \cdot B(z,0) \lhd K(x,y) \rhd \delta)$$

can easily be defined (this expression equals δ whenever $\neg(x < z < y)$).

Let $k \stackrel{def}{=} \mu x \,.\, [x \in W_{e_1}]$. Typically, in $SOS(\mathscr{S})(B(k,0))$ 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 $SOS(\mathscr{S})(A) \leftrightarrows_{r.e.} SOS(\mathscr{S})(B(k,0))$, for given a suitable 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 $SOS(\mathscr{S})(A)$ and $SOS(\mathscr{S})(B(k,0))$ thus assumes a recursive separation of W_{e_1} and W_{e_2}. □

8. Conclusions and Comparison with Related Work

The transition systems studied in this paper can be associated to processes specified in common specification languages such as CCS [Mil89], LOTOS [ISO87], ACP [BeK84, BaW90] and PSF [MaV90, MaV93]. As a means for defining transition systems up to bisimulation equivalence, the specification language μCRL (*micro* Common Representation Language [GrP91a, GrP95]) is used. Two simple fragments of μCRL are distinguished, involving a decidable form of guardedness. These fragments—$\mu\text{CRL}_{\text{TREC}}$ and $\mu\text{CRL}_{\text{PRIM}}$—are up to bisimulation equivalence universally expressive with respect to recursive and primitive recursive transition systems. For both these domains, the following properties are classified in the arithmetical hierarchy: *bisimilarity*, *perpetuity* and *deadlock freedom*, (all Π_1^0), *acyclic regularity* (Σ_1^0) and *regularity* (Σ_2^0). In expressivity and classification proofs, all occurrences of sequential composition can be replaced by action prefixing, by which these results also refer to (value-passing) CCS and LOTOS. 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 by the complexity of the witnessing relations, which extends r.e. bisimilarity.

In the formal definition of μCRL its authors adopted a (common) definition of guardedness (cf. [BBK87]): a specification $\mathscr{S}$ is "guarded" whenever the next steps of each closed process term in $\mathbb{P}(\mathscr{S})$ can be computed and are finite in number (so each closed process term determines a recursive transition system). Of course, this is relative to an interpretation of the data declared in a specification. This notion of guardedness is not decidable; it implies for each recursively defined process term the existence of a finite upper bound to the number of expansions (replacing identifiers by their defining right-hand sides) necessary to compute its next steps. Indeed, even restricting to primitive recursive data types, this general form of guardedness is complete in Π_2^0. This motivates the restriction to the decidable property "lspd-guardedness" (Definition 3.2.3) in $\mu\text{CRL}_{\text{TREC}}$ and $\mu\text{CRL}_{\text{PRIM}}$.

In terms of expressiveness, $\mu\text{CRL}_{\text{TREC}}$, $\mu\text{CRL}_{\text{PRIM}}$ and lspd-guardedness form a reasonable point of departure. In the following some other expressiveness results are discussed.

Two basic results of De Simone [Sim85] imply that:

1. Languages such as CCS, SCCS [Mil83], and ACP can up to bisimulation equivalence be defined by transition rules in a particular format, and hence be expressed in MEIJE [AuB84], and
2. Any recursively enumerable process graph is up to bisimulation equivalence representable in MEIJE.

In [BBK87], Baeten, Bergstra and Klop show that each recursive transition system over a finite set of actions is expressible in ACP with finite, guarded recursion and 'abstraction' (based on Milner's silent steps [Mil89] and weak bisimulation equivalence, see further [BeK85]), and that the feature abstraction is necessary for this result. In particular, they provide a counterexample for the case without abstraction.

In [Vaa93], Vaandrager investigates the expressive power of process algebras in the setting of structural operational semantics. Based on the above-mentioned counterexample in [BBK87], it is shown that no effective operational semantics for an enumerable language can specify all effective transition systems up to trace equivalence. Further results in this paper are on calculi for transition rules. In particular, a format is identified that guarantees an effective operational semantics, and that contains the guarded versions of CCS, SCCS, MEIJE and ACP. Hence, the above-mentioned expressiveness results of De Simone both depend on the use of unguarded recursion. Furthermore, Vaandrager defines an effective process language PC in his format, that is more expressive than any effective version of CCS, SCCS, MEIJE and ACP with finite, guarded recursion (due to a "relational renaming operator").

Van Glabbeek recreates in [Gla95] the expressiveness results of De Simone in variants of ACP without sequential composition, to which prefixing and renaming operators are added—either functional: aprACP$_F$, or relational: aprACP$_R$, having the expressive power of PC [Vaa93]. He only uses guarded recursion, and presents an extended, simultaneous classification of transition systems and process expressions. In particular, Van Glabbeek defines a primitive effective version of aprACP$_F$ that is universally expressive for primitive recursive transition systems up to bisimulation equivalence (cf. Corollary 5.1.3). In this case, infinite—but primitive recursive—guarded recursion is used (cf. lspd-guardedness).

In [BeG94], Bezem and Groote define *linear process operators* in the setting of μCRL with silent steps (τ-steps). In this paper, a general approach to verification with invariant techniques is presented. From the proof of Expressivity Theorem 5.1.2, it can be inferred that linear process operators are not a restriction in terms of expressiveness. First observe that this proof suggests a 'normal form theorem', the proof of which is based on strong bisimulation semantics, the coding of processes as a data type, the total recursive function $SOS(\mathscr{S})$, and application of Theorem 4.2.1. (In fact, this applies to *effective* μCRL, [GrP91a, GrP95].) Because the specification given in the proof of Expressivity Theorem 5.1.2 can be written as a linear process operator (essentially by replacing the R-equation by one with a sum operation over syntactically guarded subterms), the above-mentioned expressiveness of linear process operators follows.

As for the complexity of bisimilarity, Darondeau approaches this topic from a different point of view. In [Dar90] he gives an effective transition system that is infinitely branching—states and labels are recursive sets, and the transitions are recursively enumerable as a subset of Nat^3—, and for which the quotient

of the largest bisimulation is *not* effective. In [Dar91], this is sharpened to hold for a deterministic, primitive recursive transition graph with a finite number of labels. A consequence of the distinction between the various types of bisimilarity addresses in the case of μCRL a proof theoretic phenomenon. Consider some axiomatic, finitary proof system for $\mu\mathrm{CRL}_{\mathrm{PRIM}}$, say $\vdash$. Proving for any two closed process terms p, q over some $\mu\mathrm{CRL}_{\mathrm{PRIM}}$ specification $\mathscr{S}$

$$\mathscr{S} \vdash p = q \implies SOS(\mathscr{S})(p) \leftrightarrows_{r.e.} SOS(\mathscr{S})(q)$$

shows by the result of Bergstra (Theorem 7.1.1) and the Expressivity Theorem 5.1.2 that $\vdash$ cannot be complete with respect to bisimulation equivalence. As the implication above can be shown for the $\mu\mathrm{CRL}_{\mathrm{PRIM}}$ fragment of the proof system for μCRL defined in [GrP91b, GrP93], it follows that this system is not complete with respect to this fragment. This applies also to the $\mu\mathrm{CRL}_{\mathrm{TREC}}$ fragment. A conclusion of this may be that other process algebras, for example those defined by recursively enumerable bisimilarity, have a right to exist.

Acknowledgement

Observations of Jan Bergstra formed a fundamental inspiration for this paper. I further thank Jos Baeten, Jan Bergstra, Javier Blanco, Doeko Bosscher, Tim Fernando, Henri Korver, Joachim Parrow, Jan Rutten and Frits Vaandrager for discussions and critical remarks. Finally, I thank the referees and the editor for useful comments and suggestions.

References

[AuB84] Austry, D. and Boudol, G.: Algèbre de processus et synchronisations. *Theoretical Computer Science*, 30(1):91–131, 1984.

[BaB92] Baeten, J.C.M. and Bergstra, J.A.: Process algebra with signals and conditions. In M. Broy, editor, *Programming and Mathematical Methods, Proceedings Summer School Marktoberdorf 1991*, pages 273–323. Springer-Verlag, 1992. NATO ASI Series F88.

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

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

[BeG94] Bezem, M.A. and Groote, J.F.: Invariants in process algebra with data. In [JoP94], pages 401–416, 1994.

[BoG96] Bosscher, D.J.B. and Griffioen, W.O.D.: Regularity for a class of context-free processes is decidable. In proceedings of ICALP'96, to appear.

[BeK84] Bergstra, J.A. and Klop, J.W.: Process algebra for synchronous communication. *Information and Computation*, 60(1/3):109–137, 1984.

[BeK85] Bergstra, J.A. and Klop, J.W.: Algebra of communicating processes with abstraction. *Theoretical Computer Science*, 37(1):77–121, 1985.

[BaV95] Baeten, J.C.M. and Verhoef, C.: Concrete process algebra. In S. Abramsky, D.M. Gabbay, and T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science, Volume IV, Syntactical Methods*, pages 149–268. Oxford University Press, 1995.

[BaW90] Baeten, J.C.M. and Weijland, W.P.: *Process Algebra*. Cambridge Tracts in Theoretical Computer Science 18. Cambridge University Press, 1990.

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

[Dar90] Darondeau, Ph.: 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, LNCS 469, pages 223–238. Springer-Verlag, 1990.

[Dar91] Darondeau, Ph.: Recursive graphs are not stable under maximal reduction. *Bulletin of the European Association for Theoretical Computer Science*, 44:186–189, 1991.

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

[Gla95] van Glabbeek, R.J.: On the expressiveness of ACP (extended abstract). In [PVV95], pages 188–217, 1995.

[GrP90] Groote, J.F. and Ponse, A.: The syntax and semantics of μCRL. Report CS-R9076, CWI, Amsterdam, 1990.

[GrP91a] Groote, J.F. and Ponse, A.: μCRL: 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.

[GrP91b] Groote, J.F. and Ponse, A.: Proof theory for μCRL. Report CS-R9138, CWI, 1991.

[GrP93] Groote, J.F. and Ponse, A.: Proof theory for μCRL: a language for processes with data. In D.J. Andrews, J.F. Groote, and C.A. Middelburg, editors, *Proceedings of the International Workshop on Semantics of Specification Languages*, pages 232–251. Workshops in Computing, Springer-Verlag, 1994.

[GrP95] Groote, J.F. and Ponse, A.: The syntax and semantics of μCRL. In [PVV95], pages 26–62, 1995. (Appeared earlier as [GrP90].)

[GrV92] Groote, J.F. and Vaandrager, F.W.: Structured operational semantics and bisimulation as a congruence. *Information and Computation*, 100(2):202–260, 1992.

[HHJ87] Hoare, C.A.R., Hayes, I.J., Jifeng, He., Morgan, C.C., Roscoe, A.W., Sanders, J.W., Sorensen, I.H., Spivey, J.M. and Sufrin, B.A.: Laws of programming. *Communications of the ACM*, 30(8):672–686, August 1987.

[HU79] Hopcroft, J.E. and Ullman, J.D.: *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.

[JoP94] Jonsson, B. and Parrow, J.: editors, *Proceedings CONCUR 94*, Uppsala, Sweden, LNCS 836. Springer-Verlag, 1994.

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

[Mil83] Milner, R.: Calculi for synchrony and asynchrony. *Theoretical Computer Science*, 25:267–310, 1983.

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

[MaM94] Mauw, S. and Mulder, H.: Regularity of BPA-systems is decidable. In [JoP94], pages 34–47, 1994.

[MaV90] Mauw, S. and Veltink, G.J.: A process specification formalism. *Fundamenta Informaticae*, XIII:85–139, 1990.

[MaV93] Mauw, S. and Veltink, G.J.: editors, *Algebraic Specification of Communication Protocols.* Cambridge Tracts in Theoretical Computer Science 36. Cambridge University Press, 1993.

[Par81] Park, D.M.R.: Concurrency and automata on infinite sequences. In P. Deussen, editor, *5th GI Conference*, LNCS 104, pages 167–183. Springer-Verlag, 1981.

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

[PVV95] Ponse, A., Verhoef, C. and van Vlijmen, S.F.M.: editors, *Algebra of Communicating Processes, Utrecht 1994.* Workshops in Computing, Springer-Verlag, 1995.

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

[Sim85] de Simone, R.: Higher-level synchronising devices in MEIJE–SCCS. *Theoretical Computer Science*, 37:245–267, 1985.

[Vaa93] Vaandrager, F.W.: Expressiveness results for process algebras. In J.W. de Bakker, W.P. de Roever, and G. Rozenberg, editors, *Proceedings REX Workshop on Semantics: Foundations and Applications*, Beekbergen, The Netherlands, June 1992, LNCS 666, pages 609–638. Springer-Verlag, 1993.

Received August 1992
Accepted in revised form March 1996 by J. Parrow