A Constructive Version of the Approximation Induction Principle

S. Mauw
Computer Science Department,
University of Amsterdam,
P.O. Box 41882, 1009 DB Amsterdam, The Netherlands.

A constructive version of the Approximation Induction Principle is formulated, which states that equality of processes can be decided after observing the initial behaviour of the processes. This principle is proved correct for the class of regular processes. The number of states that the processes have, determines the number of steps that have to be considered. This principle can be used to formulate a complete inference system for regular processes and algorithms to decide upon equality.
Note: This work was sponsored in part by ESPRIT contract nr. 432, Meteor.

1. INTRODUCTION

In Mauw [9] an algebraic specification of some topics in process algebra was given. It turned out that for some items no operational definition was known, so no algebraic specification was possible. Some research led to the observation that for the class of regular processes the Approximation Induction Principle (AIP) could be reformulated in an effective way. AIP states that two processes are equal if all their projections are equal. The projection of a process at level *n* can be viewed as the initial behaviour of this process up to the first *n* steps.
The constructive version of this principle (AIP^c) states that not all projections have to be considered, but only some projection at a large enough level. So just by comparing the initial behaviour of two regular processes, it is possible to decide upon equality.

The depth depends only on the number of states both processes have, in fact it is the sum of these numbers. If the processes are defined by means of linear specifications, the number of states is equal to the number of equations used.

Furthermore it can be proved that this result is best possible, in the sense that for every pair of positive integers n and m there are processes with n, respectively m states, which are equal up to depth $n+m-1$, but unequal at depth $n+m$.

Because it states exactly which processes are equal and which are not, AIP^c can be used to give a complete axiomatisation of the class of regular processes. In Milner [11] and Bergstra & Klop [2] other complete systems were described.

The constructive version of AIP leads us to consider algorithms that can be used to determine equality of regular processes. At first glance the equality problem for regular process graphs could be mapped to the equality problem for finite process trees. This problem has an easy solution, but the number of nodes increases exponentially. Another possible algorithm has as its origin an operationalisation of the proof of AIP^c. It yields the same complexity as an algorithm in Kanellakis & Smolka [8].

I would like to express my thanks to Jos Baeten, Jan Bergstra, Henk Goeman, Marjan van Herwijnen, Jan-Willem Klop, and Frits Vaandrager. Special thanks go to Freek Wiedijk, who suggested an improved version of AIP^c to me.

2. PROCESS ALGEBRA

2.1. BPA

In this section some topics in process algebra will be introduced (more on process algebra can be found in Bergstra & Klop [3], [6]). Process algebra studies the mathematical abstraction of the physical notion of a process. A process is considered as being constructed from a number of atomic actions from some finite set A. Two major ways in which processes are constructed out of simpler ones are alternative composition (+) and sequential composition (.). The process $x+y$ can be read as "do either x or y", and the process $x.y$ as "first do x and then y". The axioms of Basic Process Algebra (BPA) determine which processes should be identified:

(A1)	$x+y = y+x$
(A2)	$(x+y)+z = x+(y+z)$
(A3)	$x+x = x$
(A4)	$(x+y)z = xz+yz$
(A5)	$(xy)z = x(yz)$

As a convention we shall omit the dot-sign and reduce the number of parentheses by giving this function a higher priority than addition.

EXAMPLE: If $A=\{a,b,c\}$ then

$$(b+c)ab + bab = bab + cab, \text{ while}$$

$$a(b+c) \neq ab + ac$$

2.2. RECURSION

The introduction of guarded recursive specifications enables us to consider also infinite processes. A recursive specification is a system of recursive equations:

$$x_1 = s_1(x_1,\dots,x_n)$$
$$x_2 = s_2(x_1,\dots,x_n)$$
$$\dots\dots$$
$$x_n = s_n(x_1,\dots,x_n).$$

The terms $s_1,\dots,s_n$ may contain the variables $x_1,\dots,x_n$. A recursive specification is **guarded** if all occurences of variables in all terms are guarded by an atom. An occurrence of variable x in a term t is said to be guarded if some subterm $a.s$ exists such that a is an atom (the guard) and s is a term containing the occurrence of x. We also consider a specification guarded if it is possible to transform the specification into one which is guarded, by use of the axioms and substitution.

EXAMPLE:

The specification

$$x = a(x + byy)$$
$$y = cay + abc + x$$

is guarded, while

$$x = ax + x$$

is not.

It is consistent to assume that each guarded recursive specification determines a unique process, called the **solution** of the specification. This process may be infinite.

EXAMPLE:

The equation

$$x=ax$$

determines the process a.a.a. ..., also denoted by a^{ω}.

A **linear** specification contains only linear equations. This means that every equation is of the form

$$x_i = \Sigma_{s\in S}\, a_s x_s + \Sigma_{t\in T}\, b_t.$$

The summation symbol is used to denote a (finite) number of additions. Every linear specification is obviously guarded. If a process is a solution of some linear specification, it is said to be a **regular** process.

2.3. PROJECTION

The projection function π_n ($n\geq 1$) can be used to describe the initial behaviour of a process.

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

EXAMPLE:

1) $\pi_3(a(bc+bca)a) = a(bc+bc) = abc$

2) If x is the solution of

$$x = ax + by + a$$
$$y = by$$

then $\pi_3(x) = a\pi_2(x) + b\pi_2(y) + a = a(a\pi_1(x) + b\pi_1(y) + a) + bb\pi_1(y) + a$
$= a(a(a+b) + bb + a) + bbb + a$

The second example shows that the projection function is also defined for (infinite) processes which are defined by some guarded recursive specification. In fact, if all projections of some process are given, the process is completely known. This is known as the Approximation Induction Principle (see Bergstra & Klop [7]):

$$\text{(AIP)} \qquad \forall_{k>0}\ \pi_k(x) = \pi_k(y) \quad \Rightarrow \quad x=y.$$

EXAMPLE:

AIP can be used to prove that the following two systems define the same process

$$x_0 = ax_0 \qquad \text{and} \qquad \begin{aligned} y_0 &= ay_0 + ay_1 \\ y_1 &= ay_1 \end{aligned}$$

2.4. GRAPH MODEL

The graph model (see Bergstra & Klop [3]) is often considered to be the most natural model for the theory of processes. Every graph determines some process, and every solution of a guarded specification determines some equivalence class of graphs.

We only consider rooted directed finitely branching labeled graphs. This means that in every graph one node is selected to be the root of the graph, that every node has finitely many outgoing edges and that every edge has some atom assigned to it. An edge from node s to node s', labeled with an atom $a\in A$ will be denoted by s-a->s'.

Two graphs g and h determine the same process if they are **bisimilar** (notation: $g \leftrightarrow h$). This is the case if there exists a relation R between nodes of g and nodes of h, such that:

(i) The roots of g and h are in relation R.

(ii) For every edge s-a->s' in g and for every node t in h such that R(s,t), there is an edge t-a->t' in h such that R(s',t').

(iii) For every edge t-a->t' in h and for some node s in g such that R(s,t), there is an edge s-a->s' in g such that R(s',t').

The relation R is called a **bisimulation** between the graphs g and h. The notion of bisimulation is from Park [12] and is comparable to Milner's observational congruence [10]. There is a canonical way to associate a graph to each linear specification. Two processes are called equal in bisimulation semantics if their graphs are bisimilar.

EXAMPLE:

The guarded specification

$x=ax+aby$

$y=cy$

has a solution which is represented by either of the two following graphs. They are bisimilar.

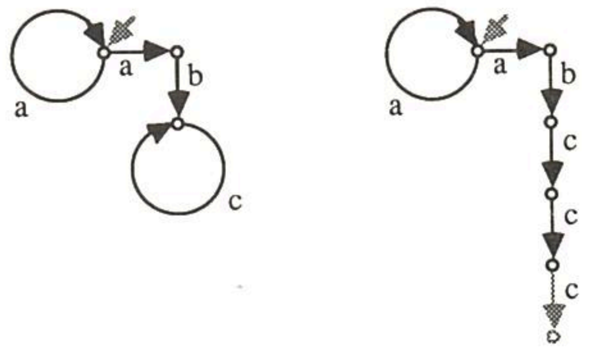

Fig. 1

To define the operators +, . and π_n on graphs the notion of unwinding a graph should be introduced. If the root of a graph is cyclic, it can be transformed to one with an acyclic root, which is bisimilar with the original. This is simply done by adding a new root and new edges from this new root to all nodes which were accessible from the old root. Then all unaccesible nodes can be deleted. By recursively doing this operation on all nodes, it is possible to unwind the entire graph.

The operators can now be defined by:

g+h is the graph derived from g and h by unwinding their roots and then identifying the new roots.

g.h is the graph which is derived by identifying all leaves of g with the root of h. If g has no leaves, only g remains.

$\pi_n(g)$ is derived by unwinding g completely and deleting all nodes and associated edges which are not accessible from the root in n "steps".

The class $\mathbb{R}$ of graphs with a finite number of nodes and edges is the class of regular graphs. $\mathbb{R}/\underline{\leftrightarrow}$ is a model for the regular processes.

3. A CONSTRUCTIVE VERSION OF AIP

A well known fact about projections is the following (see e.g. Bergstra & Klop [4]):

> If two processes are equal (in bisimulation semantics) then their projections are equal. (or: bisimulation is a congruence relation on $\mathbb{R}$ with respect to the projection function)

The converse of this fact is known as the Approximation Induction Principle (AIP):

> If all projections of two processes are equal then the two processes are equal.
>
> (AIP) $\quad \forall_{k>0}\ \pi_k(x) = \pi_k(y) \quad \Rightarrow \quad x=y.$

This principle has been shown to be valid in the model $\mathbb{R}/{\underline{\leftrightarrow}}$ of finite graphs modulo bisimulation (see [3]). At first sight it could very well serve to construct a complete axiomatisation of process algebra with bisimulation semantics, but there is no effective way to check the infinite number of premises. However, for the class of regular processes, a constructive version of AIP can be derived (AIP^c). This version states that only a finite number of projections have to be considered, in order to determine the equality of two regular processes. The upper bound is dependent on the number of states that both processes have; in fact it is the sum of both numbers.

3.1. THEOREM: *If two regular processes,* x_0 *and* y_0*, are defined by the linear recursive specifications E and E', having n and m equations respectively, then*

$$(AIP^c) \qquad \pi_{n+m}(x_0) = \pi_{n+m}(y_0) \quad \Rightarrow \quad x_0=y_0.$$

PROOF:

We can suppose that the sets of variables of E and E' are disjoint. Let V be the set of variables that are used in the definitions of x_0 and y_0. Then V contains n+m elements. Consider, for k>0, the relation $\equiv_k$ on V, defined by:

$$x \equiv_k y \quad \Leftrightarrow \quad \pi_k(x) = \pi_k(y).$$

It is easy to prove that this defines an equivalence relation.

Because

$$\pi_{k+1}(x) = \pi_{k+1}(y) \quad \Rightarrow \quad \pi_k(x) = \pi_k(y),$$

a non-increasing sequence of relations is defined:

$$\equiv_1 \ \supseteq \ \equiv_2 \ \supseteq \ \equiv_3 \ \supseteq \ldots$$

CLAIM 1. Initially this sequence is strictly decreasing, while the rest of the sequence is constant:

$$(\cong_k = \cong_{k+1}) \quad \Rightarrow \quad (\cong_{k+1} = \cong_{k+2})$$

PROOF OF CLAIM 1:

Let $\cong_k$ and $\cong_{k+1}$ be equal. Since the sequence is non-increasing, we only have to prove:

$$\cong_{k+1} \subseteq \cong_{k+2}.$$

Supposing this is false, there are x and y in V, such that

$$\pi_{k+1}(x) = \pi_{k+1}(y) \quad \text{and} \quad \pi_{k+2}(x) \neq \pi_{k+2}(y).$$

Now let x and y be defined by the linear equations

$$x = \Sigma_{p\in P}\, a_p x_p \ + \Sigma_{q\in Q}\, b_q$$
$$y = \Sigma_{r\in R}\, c_r y_r \ + \Sigma_{s\in S}\, d_s.$$

Then the k+2nd projections look like

$$\pi_{k+2}(x) = \Sigma_{p\in P}\, a_p \pi_{k+1}(x_p) \ + \Sigma_{q\in Q}\, b_q$$
$$\pi_{k+2}(y) = \Sigma_{r\in R}\, c_r \pi_{k+1}(y_r) \ + \Sigma_{s\in S}\, d_s.$$

If these are unequal it is easy to see that at least one of the following expressions is true:

(1) $\exists_{q\in Q}\forall_{s\in S}\, b_q \neq d_s \qquad \vee \qquad \exists_{s\in S}\forall_{q\in Q}\, b_q \neq d_s$

(2) $\exists_{p\in P}\forall_{r\in R}\, (a_p \neq c_r \vee \pi_{k+1}(x_p) \neq \pi_{k+1}(y_r)) \qquad \vee$

$\exists_{r\in R}\forall_{p\in P}\, (a_p \neq c_r \vee \pi_{k+1}(x_p) \neq \pi_{k+1}(y_r))$

Now we can use the equality of $\cong_k$ and $\cong_{k+1}$ to obtain an expression which is equivalent to the second one:

(2') $\exists_{p\in P}\forall_{r\in R}\, (a_p \neq c_r \vee \pi_k(x_p) \neq \pi_k(y_r)) \qquad \vee$

$\exists_{r\in R}\forall_{p\in P}\, (a_p \neq c_r \vee \pi_k(x_p) \neq \pi_k(y_r))$

So at least one of the expressions (1) and (2') is true. From this we can conclude:

$$\pi_{k+1}(x) = \Sigma_{p\in P}\, a_p \pi_k(x_p) \ + \Sigma_{q\in Q}\, b_q \quad \neq$$
$$\Sigma_{r\in R}\, c_r \pi_k(y_r) \ + \Sigma_{s\in S}\, d_s \ = \pi_{k+1}(y).$$

This is in contradiction with the assumptions.

CLAIM 2. The sequence is constant from $\cong_{n+m}$ at the latest:

$$\cong_{n+m} = \cong_{n+m+1}$$

PROOF OF CLAIM 2:

Note that for two equivalence relations R and R', if R⊂R' and R≠R', the number of equivalence classes generated by R is strictly greater then the number of equivalence classes generated by R'. Now, since the sequence of equivalence relations we defined is initially strictly decreasing, and since the maximum number of equivalence classes of V is n+m, we can conclude that at most the first n+m relations in the sequence can be unequal. Therefore

$$\cong_{n+m} = \cong_{n+m+1}$$

Now we come to the final part of the proof. The two claims can now be used to infer from

$$\pi_{n+m}(x_0) = \pi_{n+m}(y_0)$$

the equality of all projections:

$$\forall k>0\ \pi_k(x_0) = \pi_k(y_0).$$

Now, using AIP, we can conclude

$$x_0=y_0.$$

NOTE: The equivalence relation $\cong_k$ is discussed in Kanellakis & Smolka [8], where it is called k-limited observation equivalence. Congruence on observable processes there is defined by:

$$\cong = \cap_{k>0} \cong_k.$$

This formula is equivalent to AIP. In this setting $\cong_k$coincides with Milner's k-string equivalence.

4. TIGHTNESS

The question whether the result derived in the previous section is tight, in the sense that the bound n+m cannot be reduced, can be answered affirmatively.

4.1. THEOREM: *For every pair of positive integers* n *and* m *there are regular processes* x *and* y, *defined by* n *and* m *linear equations respectively, such that:*

$$\pi_{n+m-1}(x) = \pi_{n+m-1}(y) \ \wedge \ x \neq y.$$

PROOF:

Let n and m be positive integers, and let a be some arbitrary atom. Without loss of generality it may be assumed that $m \geq n$. Let r be the remainder of m divided by n ($r = m \bmod n$). Now define the process x_0 using the following system of n linear equations ($0 \leq k \leq n-1$):

$$x_k = \begin{cases} a.x_{k+1(\text{mod } n)} & \text{if } k \neq r-2 \ (\text{mod } n) \\ \\ a.x_{k+1(\text{mod } n)} + a & \text{if } k = r-2 \ (\text{mod } n) \end{cases}$$

The graph associated with this system consists of a cycle with one "handle" attached to it (see the example below). Define the process y_0 using the following system of m linear equations ($0 \leq k \leq m-1$):

$$y_k = \begin{cases} a.y_{k+1} & \text{if } k \neq r-2 \ (\text{mod } n) \text{ and } k \neq m-1 \\ a.y_{k+1} + a & \text{if } k = r-2 \ (\text{mod } n) \text{ and } k \neq m-1 \\ a.y_{m-1} & \text{if } k = m-1. \end{cases}$$

This system determines a linear graph, with a cycle at the end. At every nth node a "handle" is attached, such that the last handle is attached to the penultimate node (see the example below). Now, if d is the integer division of m by n ($d = m$ div n), the projections can be calculated as:

$$\pi_{n+m-1}(x_0) = \pi_{(d+1)n+r-1}(x_0) = a^{r-2(\text{mod } n)}.(a+a^n.(a+a^n.(...(a+a^n(a+a))...))).$$
$$\pi_{n+m-1}(y_0) = \pi_{(d+1)n+r-1}(y_0) = a^{r-2(\text{mod } n)}.(a+a^n.(a+a^n.(...(a+a^n(a))...))).$$

The repeating part is repeated d+1 times if $r \neq 0$ and $r \neq 1$, and d times if $r=0$ or $r=1$. The atom a with superscript n denotes a repetition of n times the atom a. These two projections are obviously equal, while the following two are not:

$$\pi_{n+m}(x_0) = \pi_{(d+1)n+r}(x_0) = a^{r-2(\text{mod } n)}.(a+a^n.(a+a^n.(...(a+a^n(a+a^2))...))).$$
$$\pi_{n+m}(y_0) = \pi_{(d+1)n+r}(y_0) = a^{r-2(\text{mod } n)}.(a+a^n.(a+a^n.(...(a+a^n(a^2))...))).$$

So x_0 and y_0 are not equal, while projections at level n+m-1 are.

EXAMPLE: Let n=5, m=13, r = 13(mod 5) = 3, d = 13(div 5) = 2.

$$\begin{array}{l|lll}
x_0 = a.x_1 & y_0 = a.y_1 & y_5 = a.y_6 & y_{10} = a.y_{11} \\
x_1 = a.x_2 + a & y_1 = a.y_2 + a & y_6 = a.y_7 + a & y_{11} = a.y_{12} + a \\
x_2 = a.x_3 & y_2 = a.y_3 & y_7 = a.y_8 & y_{12} = a.y_{12} \\
x_3 = a.x_4 & y_3 = a.y_4 & y_8 = a.y_9 & \\
x_4 = a.x_0 & y_4 = a.y_5 & y_9 = a.y_{10} &
\end{array}$$

The projections at level 17 are equal, while the projections at level 18 are not:

$$\pi_{18}(x_0) = a(a+a^5(a+a^5(a+a^5(a+a^2))))$$

$$\pi_{18}(y_0) = a(a+a^5(a+a^5(a+a^5(a^2))))$$

The graphs associated with x_0 and y_0 are:

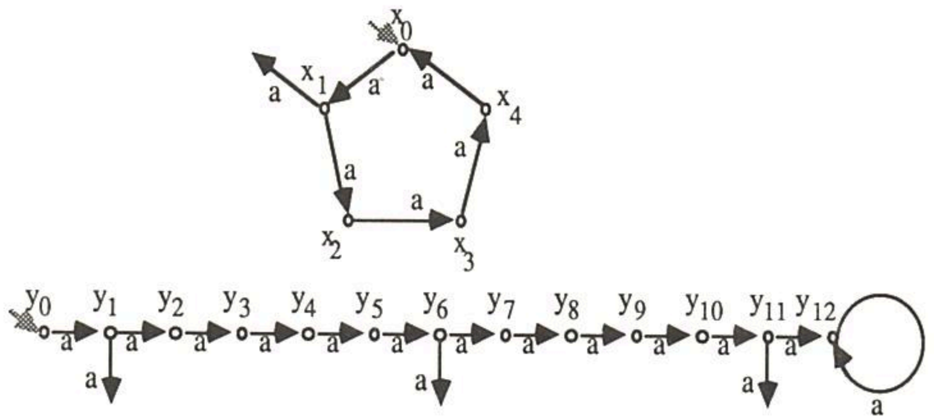

Fig. 2

5. A COMPLETE INFERENCE SYSTEM FOR REGULAR PROCESSES

5.1. REGULAR PROCESSES

In [11] Milner studied the notion of regular behaviours (or regular processes) and gave a complete inference system. Bergstra & Klop [2] gave an alternative axiomatisation of regular processes, in order to expand it with the silent step τ. In essence this inference system was the same as the one proposed by Milner, only phrased in terms of process algebra. Now, using the constructive version of the Approximation Induction Principle, yet another axiomatisation can be given. To do so, the following definitions will be needed:

Let Var be a denumerable infinite set of variables, which will be denoted by X, Y, ...
Let A be the finite set of atomic actions, denoted by a, b, ... Some extra symbols are used: <, |, >, . , , and +.

The class RHS of linear right hand sides is inductively defined by:

$a \in RHS$	if $a\in A$
$a.X \in RHS$	if $a\in A$ and $X\in Var$
$r_1+r_2 \in RHS$	if $r_1\in RHS$ and $r_2\in RHS$

The class SYS of systems of linear equations, separated by commas, is defined by:

$X=rhs \in SYS$	if $X\in Var$ and $rhs\in RHS$
$s,t \in SYS$	if $s\in SYS$, $t\in SYS$ and $lvar(s)\cap lvar(t)=\varnothing$

The function lvar: $SYS\rightarrow P(VAR)$ determines which variables are involved on the lefthand side of a system. It is defined by::

$lvar(X=rhs) = \{X\}$

$lvar(s,t) = lvar(s)\cup lvar(t)$.

Also two functions $rvar':RHS\rightarrow P(VAR)$ and $rvar:SYS\rightarrow P(VAR)$ can be defined, which determine the variables involved on the righthand side:

$rvar'(a) = \varnothing$

$rvar'(a.X) = \{X\}$

$rvar'(x+y) = rvar'(x)\cup rvar'(y)$

$rvar(X=rhs) = rvar'(rhs)$

$rvar(s,t) = rvar(s)\cup rvar(t)$

The class R of regular processes can now be defined to be the least class containing all atoms and well formed linear systems, which is closed under the operations multiplication (.) and addition (+):

$a \in R$	if $a \in A$
$<X\|s> \in R$	if $X\in Var$, $s\in SYS$, $X\in lvar(s)$ and $rvar(s)\subset lvar(s)$
$(x.y) \in R$	if $x\in R$ and $y\in R$
$(x+y) \in R$	if $x\in R$ and $y\in R$

Given some system $s\in SYS$ and a variable $X\in lvar(s)$, the righthand side of the definition for X is denoted by $(s)_X$. This expression can be transformed to a regular process by replacing the symbols + and . by the binary functions + and . and substituting for each variable $Y \in \mathrm{rvar}'((s)_X)$ the corresponding process $<Y|s>$. Notation: $(s)_X\,[Y \rightarrow <Y|s>]$. This operation is defined by:

$(s)_X = \mathrm{rhs}$ if $s = s_1, X=\mathrm{rhs}, s_2$ where s_1, s_2 may be not present

$a[Y \rightarrow <Y|s>] = a$

$a.Z[Y \rightarrow <Y|s>] = a.<Z|s>$

$(r_1+r_2)[Y \rightarrow <Y|s>] = r_1[Y \rightarrow <Y|s>] + r_2[Y \rightarrow <Y|s>]$

The following axioms are defined for $x,y \in R$, $a \in A$, $s,t \in SYS$, $X \in \mathrm{lvar}(s)$ and $Y \in \mathrm{lvar}(t)$:

(A1)	$x+y = y+x$
(A2)	$(x+y)+z = x+(y+z)$
(A3)	$x+x = x$
(A4)	$(x+y).z = x.z + y.z$
(A5)	$(x.y).z = x.(y.z)$
(R1)	$<X\|s> = (s)_X\,[Y \rightarrow <Y\|s>]$
(PR1)	$\pi_n(a) = a$
(PR2)	$\pi_1(a.x) = a$
(PR3)	$\pi_{n+1}(a.x) = a.\pi_n(x)$
(PR4)	$\pi_n(x+y) = \pi_n(x) + \pi_n(y)$
(AIP^c)	$\pi_{n+m}(<X\|s>) = \pi_{n+m}(<Y\|t>) \Rightarrow <X\|s> = <Y\|t>$ if $n=\|\mathrm{lvar}(s)\|$ and $m=\|\mathrm{lvar}(t)\|$

$\mathrm{BPA}_{\mathrm{Reg}}$

Axioms (A1)-(R1) are identical to the axioms stated in [2]. Axiom (R1) is needed to define the projection function on systems. It also enables us to consider every finite process as a system, so AIP^c can be formulated in an easy way.

5.2. COMPLETENESS AND SOUNDNESS

As the semantics of the class R we take the class of finite rooted process graphs modulo bisimulation equivalence $\mathbb{R}/\leftrightarrow$ (see paragraph 2.4.). Let the interpretation [<X|s>] of a term <X|s>∈R be the graph with nodes N=lvar(s) ∪ {√}, edges E={X-a->Y | $(s)_X$ has a summand a.Y} ∪ {X-a->√ | $(s)_X$ has a summand a} and root X. If +, . and π_n are defined on graphs as in paragraph 2.4., we define:

$[S.T] = [S].[T]$

$[S+T] = [S]+[T]$

$[\pi_n(S)] = \pi_n([S])$

It is easy to check that [.] is a surjective function on $\mathbb{R}/\leftrightarrow$

EXAMPLE:

Let <X|s> be

$X = aX+aY+a$

$Y = bY+aZ+a+b$

$Z = b$

Then the interpretation of <X|s> is the graph:

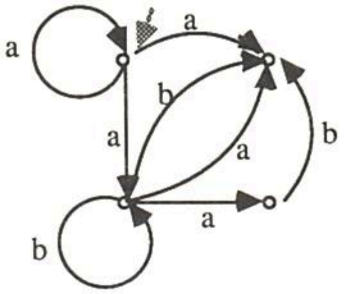

Fig. 3

5.2.1. THEOREM:

The axiom system BPA_{Reg} *is sound and complete with respect to the model of regular process graphs modulo bisimulation. For closed terms* S, T∈R:

$BPA_{Reg} \vdash S=T \iff \mathbb{R}/\leftrightarrow \models [S]=[T]$

PROOF:

Soundness of (A1)-(A5) and (PR1)-(PR4) is easy to verify. Soundness of (R1) is verified in [2]. Soundness of (AIP^c) follows directly from the theorem stated in paragraph 3.

AIP^c also provides the completeness of the system. Suppose [S]=[T]. In presence of (R1), S and T may be thought to be of the form <X|s> and <Y|t>. Let n=||var(s)| and m=||var(t)| and let the graphs g and h be representations in $\mathbb{R}$ of the classes [S] and [T]. Because g and h are bisimilar in $\mathbb{R}$, the n+m^th projections of g and h are also bisimilar. These projections are finite processes, which can be represented by closed terms without recursion parts <X|s>. Now, because the axioms (A1)-(A5) form a complete axiomatisation of finite processes without silent moves (see Bergstra & Klop [5]), the following is provable:

$$\pi_{n+m}(<X|s>) = \pi_{n+m}(<Y|t>)$$

Hence <X|s> = <Y|t>.

6. IMPLEMENTATION

In Kanellakis & Smolka [8] an algorithm is described which decides upon equality of regular processes. It is possible to use AIP^c for the construction of an algorithm, which has the same complexity. Several approaches are possible to transform AIP^c into a computer program.

The most naive way is to determine the n+m^th projection of both processes and decide whether they are equal. The algorithm in Aho, Hopcroft & Ullman [1] for determining equivalence of finite trees can be modified so that it determines bisimulation equivalence of finite trees in time linearly proportional to the number of nodes. Unfortunately, when unwinding a graph to a tree, the number of nodes increases exponentially. So this algorithm takes time exponential to the total number of states. By keeping track of all comparisons of projections at a lower level, this could be improved to an algorithm which takes polynomial time.

The second approach is derived from the proof of AIP^c. It is possible to determine the equivalence relations $\equiv_1$ until $\equiv_{n+m}$ and then check whether the two processes are in the same equivalence class. Determining the equivalence relations can be done by the following algorithm. By using the lexicographic sort algorithm of Aho, Hopcroft & Ullman [1] it is possible to determine $\equiv_1$. Sort all atoms in every righthand side and then, considering the righthand sides as strings, sort these strings. This can be done in time linearly proportional to the number of states (S=n+m) plus the number of outgoing edges (T). When $\equiv_k$ is known, $\equiv_{k+1}$ can be calculated by assigning to each equivalence class of $\equiv_k$ a unique number and substituting for each variable at each righthand side the number of its class. Then again sorting the righthand sides individually and sorting all righthand sides considered as strings results in $\equiv_{k+1}$. This operation takes also O(T+S) time. Repeating this operation S times results in a total of O(S(T+S)). This can be reduced to O(S.T) because only processes are considered with at least one outgoing edge per state.

7. FINAL REMARKS

Some questions whether the results in this paper could be extended arise. Firstly, there is the question whether some AIP^c-like principle could be formulated in the presence of the silent step τ. This is important, because all significant applications of process algebra make use of this special process. Automatic verification of protocols then comes into sight.

The question whether the class of processes for which AIP^c holds could be extended beyond the class of regular processes is also of interest. Enlarging this class would add to a solution of the decidability problem for BPA-processes.

It can be expected that the bound n+m is not tight for all regular processes. For example if the states of a process are not ordered linearly, but e.g. in "clusters", not the total number of states has to be considered, but the number of states in the largest cluster.

8. REFERENCES

[1] A.V. Aho, J.E. Hopcroft & J.D. Ullman, *The design and analysis of computer algorithms,* Addison-Wesley, 1974.

[2] J.A. Bergstra & J.W.Klop, *A complete inference system for regular processes with silent moves*, Report CS-R8420, Centre for Math. and Comp. Sci., Amsterdam, 1984.

[3] J.A. Bergstra & J.W. Klop, *Algebra of communicating processes*, Proc. CWI Symp. Math. & Comp. Sci. (J.W. de Bakker, M. Hazewinkel & J.K. Lenstra, eds.), pp. 89-138, North-Holland, 1986.

[4] J.A. Bergstra & J.W. Klop, *Process algebra: Specification and Verification in Bisimulation Semantics*, Proc. CWI Symp. Math. & Comp. Sci. II, (M. Hazewinkel, J.K. Lenstra & L.G.L.T. Meertens, eds.), pp. 61-94, North-Holland, 1986.

[5] J.A. Bergstra & J.W.Klop, *Algebra of communicating processes with abstraction,* TCS 37 (1), pp. 77-121, 1985.

[6] J.A. Bergstra & J.W.Klop, *Process algebra for synchronous communication*, Inf. & Control 60 (1/3), pp. 109-137, 1984.

[7] J.A. Bergstra & J.W.Klop, *Verification of an alternating bit protocol by means of process algebra*, Report CS-R8404, Centre for Math. and Comp. Sci., Amsterdam, 1984.

[8] P.C. Kanellakis & S.A. Smolka, *CCS Expressions, Finite State Processes and Three Problems of Equivalence,* Proc. 2nd. Ann. ACM Symp. Principles of Distributed Computing, Montreal, Canada, pp. 228-240, Aug. 1983.

[9] S. Mauw, *An Algebraic Specification of Process Algebra, including two examples,* FVI report 87-06, University of Amsterdam, 1987.

[10] R. Milner, *A calculus of communicating systems,* Springer LNCS 92, 1980.

[11] R. Milner, *A complete inference system for a class of regular behaviours,* Journal of Computer and Systems Sciences, Vol. 28, Nr. 3, June 1984, pp. 439-466.

[12] D.M.R. Park, *Concurrency and automata on infinite sequences,* Proc. 5th GI Conference, Springer LNCS 104, 1981.