Graph Isomorphism Models for Non Interleaving Process Algebra

J.C.M. Baeten
Department of Computer Science, Eindhoven University of Technology, Eindhoven, The Netherlands

J.A. Bergstra
Programming Research Group, University of Amsterdam, Amsterdam, The Netherlands; and Department of Philosophy, Utrecht University, Utrecht, The Netherlands

We present a simple and intuitive model for the syntax of ACP based on graph isomorphism. We prove an expressivity result, and use the model to determine the number of states of a process.
Note: This research was supported in part by ESPRIT basic research action 7166, CONCUR2. The second author is also partially supported by ESPRIT basic research action 6454, CONFER.

1. Introduction.

The purpose of this paper is to provide a very simple model of the syntax of ACP [7]. This model, based on graph isomorphism, provides a clear explanation of the meaning of the primitives of ACP but it satisfies fewer axioms. In particular, it is non-interleaving in the sense of [3].

We feel that the graph isomorphism model is the simplest and most convincing one presented thus far in the literature on ACP. Of course it has various drawbacks: because it is non-interleaving and because it is concrete (in the sense of [1]) it is not very well suited for equational protocol verification. The practical merit of the graph isomorphism model is that it allows a precise state count of systems. We propose to use this model if the number of states of a process description is referred to. To this end we provide some examples.

We are unaware of a similar model in the literature. Of course most constructions have been given already in [8] but that paper failed to identify the graph isomorphism structure as a model for the syntax of ACP in its own right.

Acknowledgement: The authors thank S. Mauw and P. Rambags (both Eindhoven University of Technology) for useful comments.

2. Process graphs modulo isomorphism.

2.1 Definition. We introduce a set of process graphs as follows. Let A be a given set, and let κ, λ be two infinite cardinal numbers with $\kappa \geq \lambda$. A process graph g of cardinality $< \kappa$, with out-degree (branching degree) $< \lambda$, over a set of labels A is a quadruple $\langle S, \rightarrow, \mathsf{begin}, \mathsf{end}\rangle$ where

- S is a set, the set of states,
- $\mathsf{begin} \in S$, the start state
- $\mathsf{end} \in S$, the end state
- $\rightarrow \subseteq S \times A \times S$ is the transition relation,

and we have the following conditions:

- $1 < |S| < \kappa$
- $\mathsf{begin} \neq \mathsf{end}$
- $\forall s \in S\ |\{t \in S : \exists a \in A\ \langle s,a,t\rangle \in \rightarrow\}| < \lambda$, the out-degree is $< \lambda$
- $\{s \in S : \exists a \in A\ \langle s, a, \mathsf{begin}\rangle \in \rightarrow\} = \varnothing$, the start state has no incoming edges
- $\{s \in S : \exists a \in A\ \langle \mathsf{end}, a, s\rangle \in \rightarrow\} = \varnothing$, the end state has no outgoing edges.

We call any state different from the start state or the end state an *interior state.* We write $s \xrightarrow{a} t$ for $\langle s,a,t\rangle \in \rightarrow$. We refer to the four components of a process graph g by $S(g)$, $\rightarrow(g)$, $\mathsf{begin}(g)$ and $\mathsf{end}(g)$, respectively. $\mathcal{G}(A, \kappa, \lambda)$ is the set of all process graphs over a set of labels A of cardinality $< \kappa$ with out-degree $< \lambda$. $\mathcal{G}(A, \aleph_0, \aleph_0)$ is the set of finite process graphs, $\mathcal{G}(A, \kappa, \aleph_0)$ contains only finitely branching process graphs.

Note that we require that start and end state are always different. This allows us to give an intuitive definition for alternative composition (without root unwinding). An alternative to the present definition is to allow several end states (and several start states). We restrict ourselves to the simplest definition here.

2.2 Definition. Let $g,h \in \mathcal{G}(A, \kappa, \lambda)$. A bijection ϕ between states of g and states of h is called an *isomorphism* if:

1. $\phi(\mathsf{begin}(g)) = \mathsf{begin}(h)$, $\phi(\mathsf{end}(g)) = \mathsf{end}(h)$
2. $s \xrightarrow{a} t \Leftrightarrow \phi(s) \xrightarrow{a} \phi(t)$.

We say g,h are *isomorphic*, $g \approx h$, if there is an isomorphism between g and h. Obviously, isomorphism is an equivalence relation on process graphs. We can divide out this equivalence, and obtain the algebras $\mathcal{G}(A, \kappa, \lambda)/\!\approx$. Basically, this means that in these algebras the names of the states do not matter. This allows us to take disjoint unions of state spaces in the following definitions of operators on process graphs modulo isomorphism.

2.3 Definition. We define several operators on process graphs modulo isomorphism, i.e. on the algebras $\mathcal{G}(A, \kappa, \lambda)/\!\approx$. First, constants.

1. *Atomic action.* Let $a \in A$. $\mathbf{a} = \langle\{b, e\}, \{\langle b, a, e\rangle\}, b, e\rangle$.
2. *Deadlock.* $\delta = \langle\{b, e\}, \varnothing, b, e\rangle$

Abusing notation, we usually write a for $\mathbf{a}$.

Next, operators.

3. Alternative composition. Let g,h $\in \mathcal{G}(A, \kappa, \lambda)$ be given. Assume that the set of states of g is disjoint from the set of states of h (since we consider process graphs modulo isomorphism, we can always ensure that this is the case). The set of states of g+h consists of the interior states of g, the interior states of h, and two new states begin, end. The transition relation is given by:

a. all transitions between interior states of g or h

b. a transition begin $\xrightarrow{a}$ end whenever there is a transition begin(g) $\xrightarrow{a}$ end(g) or begin(h) $\xrightarrow{a}$ end(h)

c. for interior s in g, a transition begin $\xrightarrow{a}$ s if begin(g) $\xrightarrow{a}$ s, a transition s $\xrightarrow{a}$ end if s $\xrightarrow{a}$ end(g)

d. for interior s in h, a transition begin $\xrightarrow{a}$ s if begin(h) $\xrightarrow{a}$ s, a transition s $\xrightarrow{a}$ end if s $\xrightarrow{a}$ end(h).

Note that $a + a \approx a$, and $g + h \approx h + g$, $g + (h + k) \approx (g + h) + k$, $g + \delta \approx g$ for all g,h,k.

4. Sequential composition. Let g,h $\in \mathcal{G}(A, \kappa, \lambda)$ be given with disjoint state sets. The set of states of g·h consists of the interior states of g, the interior states of h, the states begin(g), end(h) and a new state link. begin(g·h) = begin(g), end(g·h) = end(h). The transition relation is given by:

a. all transitions between states of g or h that are still in the set of states of g·h

b. a transition s $\xrightarrow{a}$ link whenever there is a transition s $\xrightarrow{a}$ end(g)

c. a transition link $\xrightarrow{a}$ s whenever there is a transition begin(h) $\xrightarrow{a}$ s.

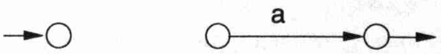

Fig. 1a. δ·a.

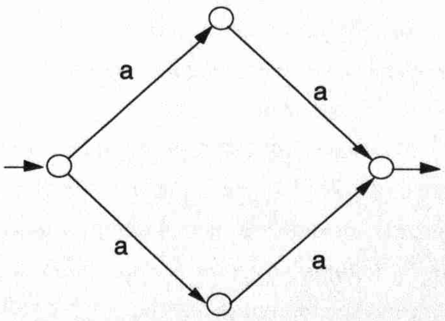

Fig. 1b. a·a + a·a.

Examples: fig. 1a shows a process graph (modulo isomorphism) for δ·a. The start state is denoted by a small unlabeled incoming arrow, the end state by a small unlabeled outgoing arrow. Note that this graph is not isomorphic to the graph of δ. Fig. 1b shows a process graph for $a \cdot a + a \cdot a$. Note that this graph is not isomorphic to the graph of $a \cdot a$, so $g + g \approx g$ does not hold for all process graphs. Similarly, $(a + b) \cdot a$ is not isomorphic to $a \cdot a + b \cdot a$ (if $b \neq a$), so $(g + h) \cdot k \approx g \cdot k + h \cdot k$ does not hold in general. We do have the identity $(g \cdot h) \cdot k \approx g \cdot (h \cdot k)$ for all graphs.

5. *Parallel composition.* Let $g, h \in \mathcal{G}(A, \kappa, \lambda)$ be given. Let a partial, commutative and associative function $\gamma: A \times A \rightarrow A$ be given, the communication function. The set of states of $g \| h$ is the cartesian product of the states of g and the states of h. $begin(g \| h) = \langle begin(g), begin(h) \rangle$, $end(g \| h) = \langle end(g), end(h) \rangle$.

The transition relation is given by:

a. for each state s in g, and each transition $t \xrightarrow{a} t'$ in h, there is a transition $\langle s,t \rangle \xrightarrow{a} \langle s,t' \rangle$

b. for each state t in h, and each transition $s \xrightarrow{a} s'$ in h, there is a transition $\langle s,t \rangle \xrightarrow{a} \langle s',t \rangle$

c. for each pair of transitions $\langle s,t \rangle \xrightarrow{a} \langle s',t \rangle$, $\langle s,t \rangle \xrightarrow{b} \langle s,t' \rangle$ such that $\gamma(a,b)$ is defined, say $\gamma(a,b) = c$, there is a transition $\langle s,t \rangle \xrightarrow{c} \langle s',t' \rangle$.

Note that $a \| b \approx a \cdot b + b \cdot a$ (if $\gamma(a,b)$ is undefined). However, $a \cdot a \ \| \ b$ is not isomorphic to $a \cdot (a \cdot b + b \cdot a) + b \cdot a \cdot a$ (the former graph has 6 states, the latter 7).

6. *Left merge.* The graph of $g \mathbin{\lfloor\!\lfloor} h$ has the same states as the graph of $g \| h$, and the same transitions except that the transitions $\langle begin(g), begin(h) \rangle \xrightarrow{a} \langle s, t \rangle$ with $t \neq begin(h)$ are omitted.

7. *Communication merge.* The graph of $g \mid h$ has the same states as the graph of $g \| h$, and the same transitions except that the transitions $\langle begin(g), begin(h) \rangle \xrightarrow{a} \langle s, t \rangle$ with $t = begin(h)$ or $s = begin(g)$ are omitted.

8. *Encapsulation.* Let $g \in \mathcal{G}(A, \kappa, \lambda)$ be given, and let $H \subseteq A$. The graph of $\partial_H(g)$ has the same states as the graph of g, and the same transitions except that all transitions $s \xrightarrow{a} s'$ with $a \in H$ are omitted.

9. *Renaming.* Let $g \in \mathcal{G}(A, \kappa, \lambda)$ be given, and let $f: A \rightarrow A$ be a given function. The graph of $\rho_f(g)$ has the same states as the graph of g, the same $begin$ and end, and each transition $s \xrightarrow{a} s'$ is replaced by a transition $s \xrightarrow{f(a)} s'$.

10. *Conditional operator.* Let $g \in \mathcal{G}(A, \kappa, \lambda)$ be given. The graph of $true :\rightarrow g$ is the same as the graph of g, and the graph of $false :\rightarrow g$ is obtained from the graph of g by removing all edges starting in the begin state. The ternary *if...then...else...* operator is defined by (b is a boolean):

$$g \lhd b \rhd h = (b :\rightarrow g) + (\neg g :\rightarrow h).$$

It is more involved to define a conditional operator over a general boolean algebra (other than $\{true, false\}$), as we did in [2]. We sketch part of this in section 6.

11. *Finite state operator.* Let $g \in \mathcal{G}(A, \kappa, \lambda)$ be given, let St be a finite set, let $act: A \times St \rightarrow A$ be a partial function, let $eff: A \times St \rightarrow St$ be a total function and let $T \in St$. The

set of states of $\lambda_T(g)$ is the the cartesian product $(S(g) - \{end(g)\}) \times St$ together with the singleton $\{end(g)\}$. $begin(\lambda_T(g)) = \langle begin(g), T\rangle$, $end(\lambda_T(g)) = end(g)$.
The transition relation is given by:

a. Suppose $s \xrightarrow{a} s'$ is a transition in g, $t \in St$ and $act(a, t)$ is undefined. In this case, we do not have a transition.

b. Suppose $s \xrightarrow{a} s'$ is a transition in g, $s' \neq end(g)$, $t \in St$ and $act(a, t)$ is defined. In this case, we have a transition $\langle s,t\rangle \xrightarrow{b} \langle s',u\rangle$, where $b = act(a,t)$ and $u = eff(a,t)$.

c. Suppose $s \xrightarrow{a} end(g)$ is a transition in g, $t \in St$ and $act(a, t)$ is defined. In this case, we have a transition $\langle s,t\rangle \xrightarrow{b} end(g)$, where $b = act(a,t)$.

12. Priority operator. Let $g \in \mathcal{G}(A, \kappa, \lambda)$ be given, and let $<$ be a given partial ordering on A. The set of states of $\theta_<(g)$ is the set of states of g, and the set of transitions is a subset of the set of transitions of g, given by:
$s \xrightarrow{a} s'$ is a transition in $\theta_<(g)$ if for all $b > a$ we do not have a transition $s \xrightarrow{b} s''$ in g.

13. Ternary Kleene star operator. Let $g,h,k \in \mathcal{G}(A, \kappa, \lambda)$ be given with disjoint state sets. The set of states of $tks(g,h,k)$ is the set of states of $g{\cdot}k$ together with the set of the interior states of h. The begin state is $begin(g)$, the end state is $end(k)$. The transitions are those of $g{\cdot}k$, as given in 4, the transitions between interior states of h, and moreover:

a. a transition $s \xrightarrow{a} link$ whenever there is a transition $s \xrightarrow{a} end(h)$ in h (s interior)
b. a transition $link \xrightarrow{a} s$ whenever there is a transition $begin(h) \xrightarrow{a} s$ in h (s interior)
c. a transition $link \xrightarrow{a} link$ whenever there is a transition $begin(h) \xrightarrow{a} end(h)$ in h.

Note that $g{\cdot}h \approx tks(g, \delta, h)$.

14. Proper iteration and binary Kleene star. With the help of the ternary Kleene star operator as defined above, we can easily define the binary Kleene star operator of [6] and the proper iteration operator of [9].

Proper iteration: $g^{\oplus}h = tks(g, g, h)$
Binary Kleene star: $g^{*}h = h + g^{\oplus}h$.

2.4 Definition.

Our model makes it possible to define a cardinality function on process graphs modulo isomorphism. Further, we can define the reverse of a process. Finally, we define an extra operator ξ, that limits a process to its set of reachable states.

1. The *cardinality* of a process graph, $|g|$, is the cardinality of its set of states. We can compute:

a. $|g \cdot h| = |g| + |h| - 1$
b. $|g + h| = |g| + |h| - 2$
c. $|g \parallel h| = |g| \times |h|$.

2. *Reverse operator.* If $g \in \mathcal{G}(A, \kappa, \lambda)$, then g^{-1} has the same set of states as g, $begin(g^{-1}) = end(g)$, $end(g^{-1}) = begin(g)$ and $s \xrightarrow{a} t$ is a transition in g^{-1} whenever $t \xrightarrow{a} s$ is a transition in g. Note that this operator commutes with all operators defined so far. It does not, however, commute with the following operator.

3. *Reachability operator.* Let a process graph $g \in \mathcal{G}(A, \kappa, \lambda)$ be given. We define its set of reachable states, $reach(g) \subseteq S(g)$ inductively:

   a. $begin(g) \in reach(g)$

   b. if $s \in reach(g)$, and $s \xrightarrow{a} s'$ is a transition in g, then $s' \in reach(g)$.

Now we define $\xi(g)$ as follows. The set of states of $\xi(g)$ is $reach(g) \cup \{end(g)\}$, with same begin state and end state, and only the transitions between reachable states. With the help of this reachability operator, we can formulate new versions of well-known identities, for instance we have $\xi(\delta \cdot g) \approx \delta$ for all process graphs g.

2.5 Theorem. The models $\mathcal{G}(A, \kappa, \lambda)$ are non-interleaving (in the sense of [3]).
Proof: Consider the process $a \cdot a \parallel b$ ($a \neq b$, and $\gamma(a,b)$ is undefined). If we have an interleaving model, then this process should equal $a \cdot (a \cdot b + b \cdot a) + b \cdot a \cdot a$. However, we found in 2.3.5 that this is not the case.

We conclude that the expansion theorem does not hold in the models $\mathcal{G}(A, \kappa, \lambda)$, and thus they are non-interleaving models.

3. Bisimulation.

We look at the familiar notion of bisimulation in the present setting. To this end, consider the following definition.

3.1 Definition.

We define the familiar notion of bisimulation on process graphs. Let $g,h \in \mathcal{G}(A, \kappa, \lambda)$. A relation R between states of g and states of h is called a *bisimulation* if:

1. $R(begin(g), begin(h))$, $R(end(g), end(h))$ and a begin or end state is not related to another state;
2. if $R(s, t)$ and $s \xrightarrow{a} s'$, then there is a t' such that $t \xrightarrow{a} t'$ and $R(s', t')$;
3. if $R(s, t)$ and $t \xrightarrow{a} t'$, then there is a s' such that $s \xrightarrow{a} s'$ and $R(s', t')$.

We say g,h are *bisimilar*, $g \leftrightarrows h$, if there is an bisimulation between g and h.
It is well-known that bisimulation is an equivalence relation on process graphs. We can divide out this equivalence, and obtain the algebras $\mathcal{G}(A, \kappa, \lambda)/\leftrightarrows$. Since bisimulation is also a congruence for all operators defined in section 2.3, we can define these operators on these algebras. We cannot, however, define the cardinality operator or the reverse

operator any more. For the reachability operator, we have $\xi(g) \leftrightarrow g$, so this operator becomes the identity on process graphs modulo bisimulation.

3.2 Definition.

Let a process graph $g \in \mathcal{G}(A, \kappa, \lambda)$ be given. We say states s,t of g are *bisimulation equivalent*, $s \leftrightarrow t$, iff there is an bisimulation R between g and g such that $R(s, t)$.

It is obvious that this defines an equivalence relation on states of g. We can divide out this equivalence relation, and obtain the reduced graph of g.

3.3 Definition. Let a process graph $g \in \mathcal{G}(A, \kappa, \lambda)$ be given. The *reduced graph* of g, $g/\leftrightarrow$ has as states the set of equivalence classes of bisimulation equivalent states of g, $\mathrm{begin}(g/\leftrightarrow) = \{\mathrm{begin}(g)\}$, $\mathrm{end}(g/\leftrightarrow) = \{\mathrm{end}(g)\}$ and $s/\leftrightarrow \xrightarrow{a} s'/\leftrightarrow$ iff $s \xrightarrow{a} s'$.

3.4 Remark. The graph isomorphism model is non-interleaving. The bisimulation model can be obtained from this model as a homomorphic image, and is interleaving. The bisimulation model, however, is not the *least* identifying model that is interleaving; in other words, there are models in between the graph isomorphism model and the bisimulation model that are still interleaving. Whether such model are useful, we do not know. Therefore, we choose not to present such a model here.

4. Expressivity.

In this section, we prove an expressivity result for the algebras $\mathcal{G}(A, \aleph_0, \aleph_0)/\approx$. We show that every finite process graph modulo isomorphism can be obtained from a single graph by using alternative, sequential, parallel composition and iteration, renaming and encapsulation operators. We do need an infinite set of atomic actions and an appropriate choice of the communication function in order to obtain this result.

4.1 Atomic actions.

Suppose we have a countable set of atomic actions A. Divide A into a countable set B and a disjoint countable set C. Suppose we have a bijection i from the set of finite subsets of B to C. Define a communication function γ as follows:

1. $\gamma(a,b) = i(\{a,b\})$	if $a,b \in B$
2. $\gamma(a, i(s)) = \gamma(i(s), a) = i(\{a\} \cup s)$	if $a \in B$, s a finite subset of B
3. $\gamma(i(s), i(s')) = i(s \cup s')$	if s,s' are finite subsets of B.

Note that this definition makes γ commutative and associative. Notice that this definition amounts to a *free* communication function on the set B. This is similar to the approach we used in [3].

4.2 The seed process.

Let a,b,c,d,e,f,g,h,k be distinct atoms from B. The seed process P is given in fig. 2. This process has 4 states, and is maximally connected: every state except end has an outgoing edge to every state except begin and every state except begin has an incoming edge from every state except end.

If Q is the same as process P except that the k-edge is omitted, then we have $P \approx Q + k$. Note that a further decomposition of P using the operators of section 2 is not possible.

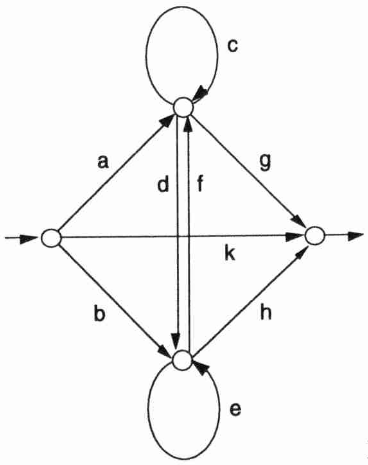

Figure 2. Seed process.

4.3 Theorem. Let $F \in \mathcal{G}(A, \aleph_0, \aleph_0)$. Then there is a graph $G \approx F$ and G can be constructed using only:

1. the seed process P
2. the operators $+, \cdot, \|, \partial_H, \rho_f, ^*$
3. additional atoms from B.

We will construct the graph G in several stages.

4.4 Obtaining enough nodes.

First of all, we construct a graph G_1 that has at least as many nodes as F, and is maximally connected. Moreover, all edges of G_1 have distinct labels. Take a number N ≥ 1 such that $2^N \geq |F| - 2$ (the number of internal nodes of F). Choose a set of distinct atoms $\{a_{k,j} : 1 \leq k \leq 9, 1 \leq j \leq N\} \subseteq B$. Define for each j the renaming f_j by:

$f_j(a)=a_{1,j}$, $f_j(b)=a_{2,j}$, $f_j(c)=a_{3,j}$, $f_j(d)=a_{4,j}$, $f_j(e)=a_{5,j}$, $f_j(f)=a_{6,j}$, $f_j(g)=a_{7,j}$, $f_j(h)=a_{8,j}$, $f_j(k)=a_{9,j}$

Define $P_j=\rho_{f_j}(P)$. This gives N copies of P, with all edge-labels distinct.

Now put $G_1 = \partial_H(P_1 \parallel P_2 \parallel ... \parallel P_N)$, with

$H = H_0 \cup H_1 \cup H_2$

$H_0 = \{a_{k,j} : 1 \le k \le 9, 1 \le j \le N\}$

$H_1 = \{i(s) \in C : |s| < N\}$

$H_2 = \{i(s) \in C : \exists k \in \{7,8,9\}, j \in \{1,...,N\}\ a_{k,j} \in s \wedge \exists k \in \{1,...,6\}, j \in \{1,...,N\}\ a_{k,j} \in s\}$.

H_0 and H_1 together ensure that the only steps possible in G_1 are communications involving a step from each of the components, H_2 ensures that if one component does a termination step (a step to end) then all components do so simultaneously.

Now put $G_2 = \xi(G_1)$. Note that G_2 has exactly 2^N internal nodes and

- exactly one transition from begin to each internal node and to end
- exactly one transition from each internal node to each internal node (including itself)
- exactly one transition from each internal node to end.

Moreover, all transitions have distinct labels.

4.5 Exact number of nodes.

The second step is to reduce G_2 so that we obtain exactly the right number of nodes. Let ϕ be an injection from S(F) into $S(G_2)$ that respects begin and end. Such an injection exists by choice of N. Let H_3 contain all labels of edges of h_2 that start from or end in a node outside the range of ϕ.

Put $G_3 = \xi \circ \partial_{H_3}(G_2)$. Then G_3 has exactly $|F|$ nodes and still has the further properties of G_2 above.

4.6 Multiple steps.

Let M be the maximum number of distinct edges between any pair of nodes in F. We will modify G_3, so that each transition is replaced by M distinct transitions. To this end, let $b_1, ..., b_M, c_1, ..., c_M$ be fresh atoms in B (distinct from all $a_{i,j}$ and pairwise distinct). Put $G_4 = b_1 + ... + b_M$, $G_5 = c_1 + ... + c_M$. Next,

$G_6 = \partial_{H'}(G_3 \parallel G_4{}^*G_5)$, with

$H' = H_4 \cup H_5 \cup H_6 \cup H_7$

$H_4 = \{b_m, c_m : 1 \le m \le M\}$

$H_5 = \{i(s) \in C : |s| = N\}$

$H_6 = \{i(s \cup \{c_m\}) \in C : i(s)$ labels a non-terminating step in G_3 and $1 \le m \le M\}$

$H_7 = \{i(s \cup \{b_m\}) \in C : i(s)$ labels a terminating step in G_3 and $1 \leq m \leq M\}$.

The encapsulation here ensures that the only steps possible in G_3 are communications between a step of G_3 and a step of the other component, and moreover that both components only terminate together.

Now put $G_7 = \xi(G_6)$. G_7 has exactly $|F|$ nodes and

- exactly M transitions from begin to each internal node and to end
- exactly M transitions from each internal node to each internal node (including itself)
- exactly M transitions from each internal node to end.

Moreover, all transitions have distinct labels.

We have now constructed a graph into which F can be embedded, after a suitable relabeling of edges. What remains now is to define this renaming, and trim away all superfluous edges.

4.7 Number of edges, labels of edges.

Now we define a renaming function f and an encapsulation set H'' as follows:

Take a pair of nodes $\langle n,m \rangle$ in G_7. If either $n = end(G_7)$ or $m = begin(G_7)$, do nothing. Otherwise, there are M edges from n to m, say with labels $d_1, ..., d_M$.

Since ϕ is a bijection between $S(F)$ and $S(G_7)$, $n = \phi(s)$, $m = \phi(t)$ for certain nodes s,t in F. Suppose there are K edges between s and t in F, with labels $e_1, ..., e_K$, $0 \leq K \leq M$. Put $f(d_1) = e_1, ..., f(d_K) = e_K$, and put $d_{K+1}, ..., d_M$ into H''.

Do the same for every node-pair in G_7. Since all labels in G_7 are distinct, f is well-defined, and $dom(f)$ and H'' are disjoint. Define

$G = \partial_{H''} \circ \rho_f(G4)$. By construction we have $G \approx F$.

5. Application: counting states.

The graph isomorphism model allows us to determine the number of states of a process. As an example, we consider the familiar Alternating Bit Protocol. We take the description of [4], and recast this in order to use iteration operators instead of recursive equations. We remark that [5] contains a description and verification of a simplified ABP (due to [10]), also using iteration instead of recursion.

We assume that we have a data set D, with $|D| = n$, that are to be transmitted from sender S to receiver R using unreliable channels K,L. $B = \{0,1\}$. The communication links are as shown in fig. 3. We use the standard communication function given by $\gamma(r_k(x), s_k(x)) = c_k(x)$ (see [4]). We have the following specifications.

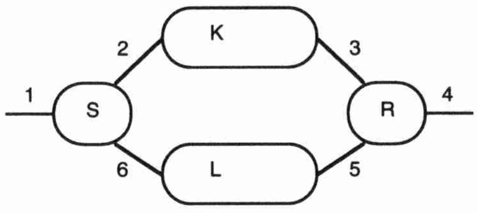

Figure 3. ABP.

5.1 Channels.

$$K = \Big(\sum_{d\in D, b\in B} r_2(db)\cdot(i\cdot s_3(db) + i\cdot s_3(\perp)) \Big) * \delta$$

$$L = \Big(\sum_{b\in B} r_5(b)\cdot(i\cdot s_6(b) + i\cdot s_6(\perp)) \Big) * \delta$$

Following the definitions in section 2, we find that K has 12n + 3 states, and L has 15 states. The graph of L is shown in fig. 4. The labels in the second part are the same as those in the first part, and are omitted. The graph of K is similar, except that the branching in the begin state and the middle state is of size 2n.

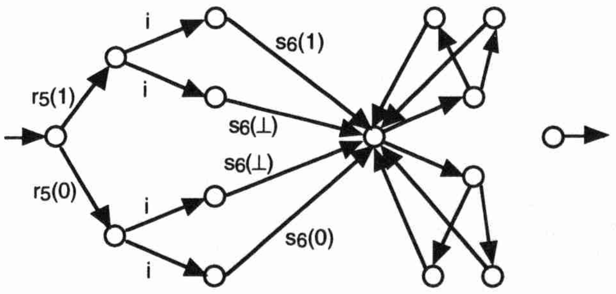

Figure 4. Acknowledgement channel L.

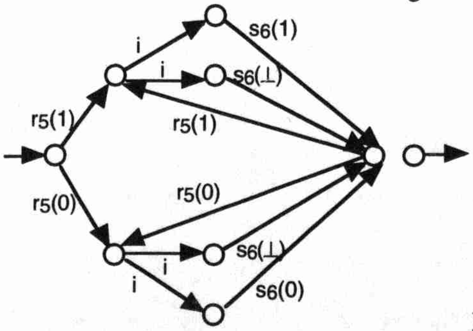

Figure 5. Reduced graph of L.

We can divide out bisimulation equivalence, and then the states in the second part of the graph are cancelled. We show the reduced graph of L in fig. 5. We find in this case that

K has $6n + 3$ states, and L 9. If we want to define these reduced processes in our syntax, it is not sufficient to use the operators of section 2, but have to follow in essence the construction of section 4.

To sketch this in case of L, we take three copies of the seed process P, with sets of atoms $a_1,...,a_9$ resp. $b_1,...,b_9$ resp. $c_1,...,c_9$, take the parallel composition, encapsulate all actions except the following 12 ternary communications, apply the following renaming and lastly apply the reachability operator.

- rename the communication of a_1, b_2 and c_2 and of a_6, b_4 and c_5 into $r_5(1)$
- rename the communication of a_2, b_2 and c_1 and of a_5, b_4 and c_6 into $r_5(0)$
- rename the communication of a_3, b_6 and c_5, of a_3, b_5 and c_6, of a_5, b_6 and c_3 and of a_6, b_6 and c_3 into i
- rename the communication of a_4, b_3 and c_5 into $s_6(1)$
- rename the communication of a_4, b_3 and c_4 into $s_6(0)$
- rename the communication of a_4, b_6 and c_4 and of a_5, b_3 and c_4 into $s_6(\perp)$.

It is obvious that this definition of L does not add to our understanding of the process. We will omit such descriptions with a minimal number of states in the sequel.

We conclude that the number of states of a process depends very much on the specification of the process in the syntax, and that the simplest and most intuitive notation usually does not have a minimal number of states.

5.2 Sender.

$S = (S0 \cdot S1) * \delta$

$$Sb = \sum_{d \in D} r_1(d) \cdot tks(s_2(db), (r_6(1\text{-}b)+r_6(\perp)) \cdot s_2(db), r_6(b)) \qquad \text{for } b = 0,1.$$

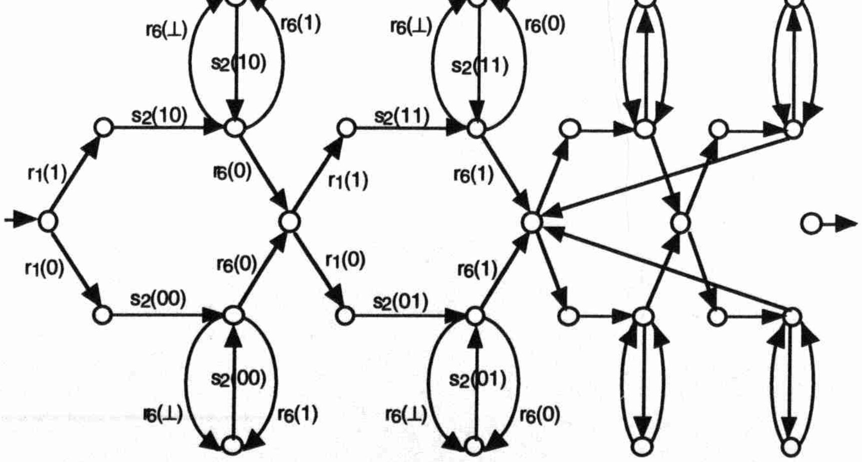

Figure 6. Graph of S, for D = {0, 1}.

We find that each Sb has 3n + 2 states, and S has 12n + 5 states. The sender will have even more states if we do not use the ternary tks operator, but instead the binary iteration operators. The graph of S is shown in fig. 6, in case D = {0, 1}. Again, the labels in the second part are the same as in the first part, and are omitted. Dividing out bisimulation equivalence, we get that each Sb has 2n + 2 states, and S has 4n + 4. We show the reduced graph of S in fig. 7.

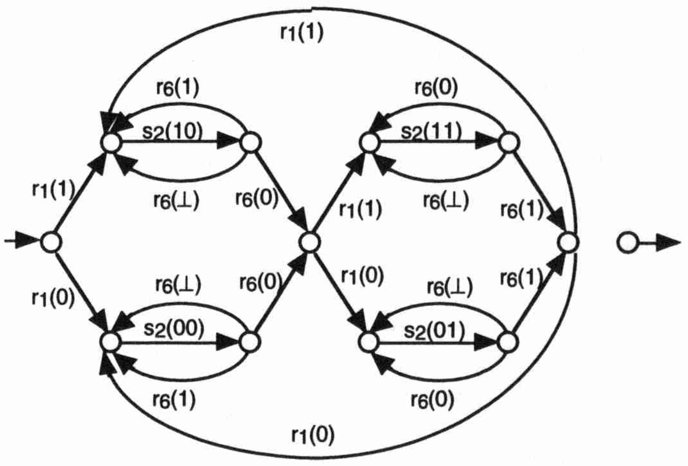

Figure 7. Reduced graph of S, for D = {0, 1}.

5.3 Receiver.

$R = (R1 \cdot R0) * \delta$

$$Rb = \Big(\big((\sum_{d\in D} r_3(db) + r_3(\perp)\ ) \cdot s5(b)\big) * \big(\sum_{d\in D} r_3(d(1\text{-}b)) \cdot s_4(d)\ \big)\Big) \cdot s_5(1\text{-}b)$$

We find that each Rb has 2n + 6 states, and R has 8n + 20 states. Dividing out bisimulation equivalence, we get that each Rb has n + 5 states, and R has 2n + 9. We show the graph of R in fig. 8, in case D = {0, 1}, with the same conventions as above, and the reduced graph in fig. 9.

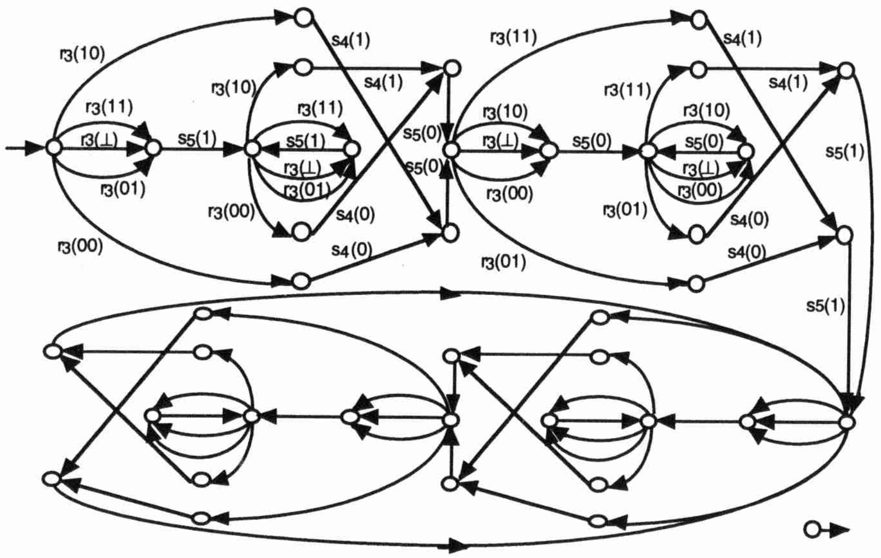

Figure 8. Graph of R, for D = {0, 1}.

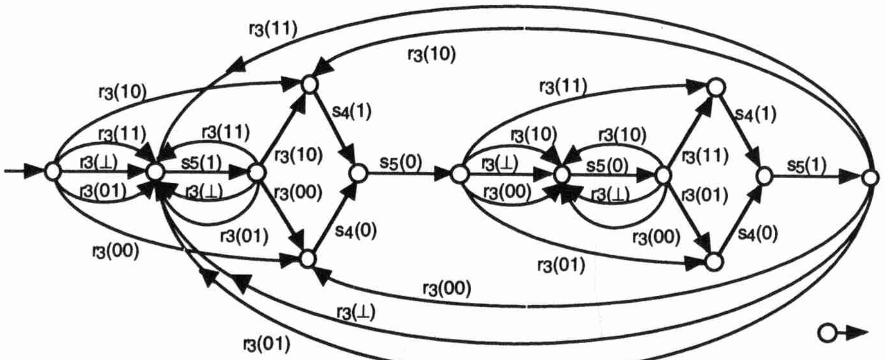

Figure 9. Reduced graph of R, for D = {0, 1}.

5.4 Protocol.

The protocol is now given by $ABP = \partial_H(S \parallel K \parallel L \parallel R)$, with

$$H = \{r_k(x), s_k(x) : k \in \{2,3,5,6\}, x \in (D \times B) \cup B \cup \{\perp\}\}.$$

By section 2, the process ABP has $(12n + 3)(12n + 5)(8n + 20)15 = 17280n^3 + 54720n^2 + 30600n + 4500$ states. Of course, most of these states are not reachable. It is much more interesting to determine the number of states of $\xi(ABP)$, i.e. the number of reachable states of the protocol.

Although the expansion theorem does not hold in our graph isomorphism model, a restricted form of it does hold, and we can use this to calculate the number of reachable states. This calculation is based on the following two identities:

- $\xi(ax \mathbin{⫫} y) \approx a\cdot\xi(x \parallel y)$
- $\xi((ax \mid by) \mathbin{⫫} z) \approx c\cdot\xi(x \parallel y \parallel z)$ if $\gamma(a, b) = c$.

Thus, if a parallel composition of processes yields a process that is really sequential (i.e. in all states there is only one possibility to proceed, either an autonomous step of one component, or a synchronisation between two components), we can use these identities to calculate the state graph. In the following, we present the results for the Alternating Bit Protocol.

5.5 Number of reachable states.

The ABP starts with an initialisation phase, where the components have not all reached the iteration parts of their behaviour. We show this initialisation phase in fig. 10. Open circles denote n states, one for every element of D, closed circles denote just one state. The initialisation phase ends in one of the two closed circles on the top left. These two are states the protocol can return to later. We show the iteration behaviour in fig. 11. We count $67n + 3$ states in the initialisation phase, plus 1 end state.

The rest of the reachable states of the ABP are shown in fig. 11. The two closed circles on the top left are the same as those shown in fig. 10. We count $58n + 4$ states in this part. Thus, in total we have $125n + 8$ states. Dividing out bisimulation equivalence, this reduces to $34n + 4$ states. Without separate begin and end states, we have the traditional $34n + 2$ states, as shown e.g. in [4].

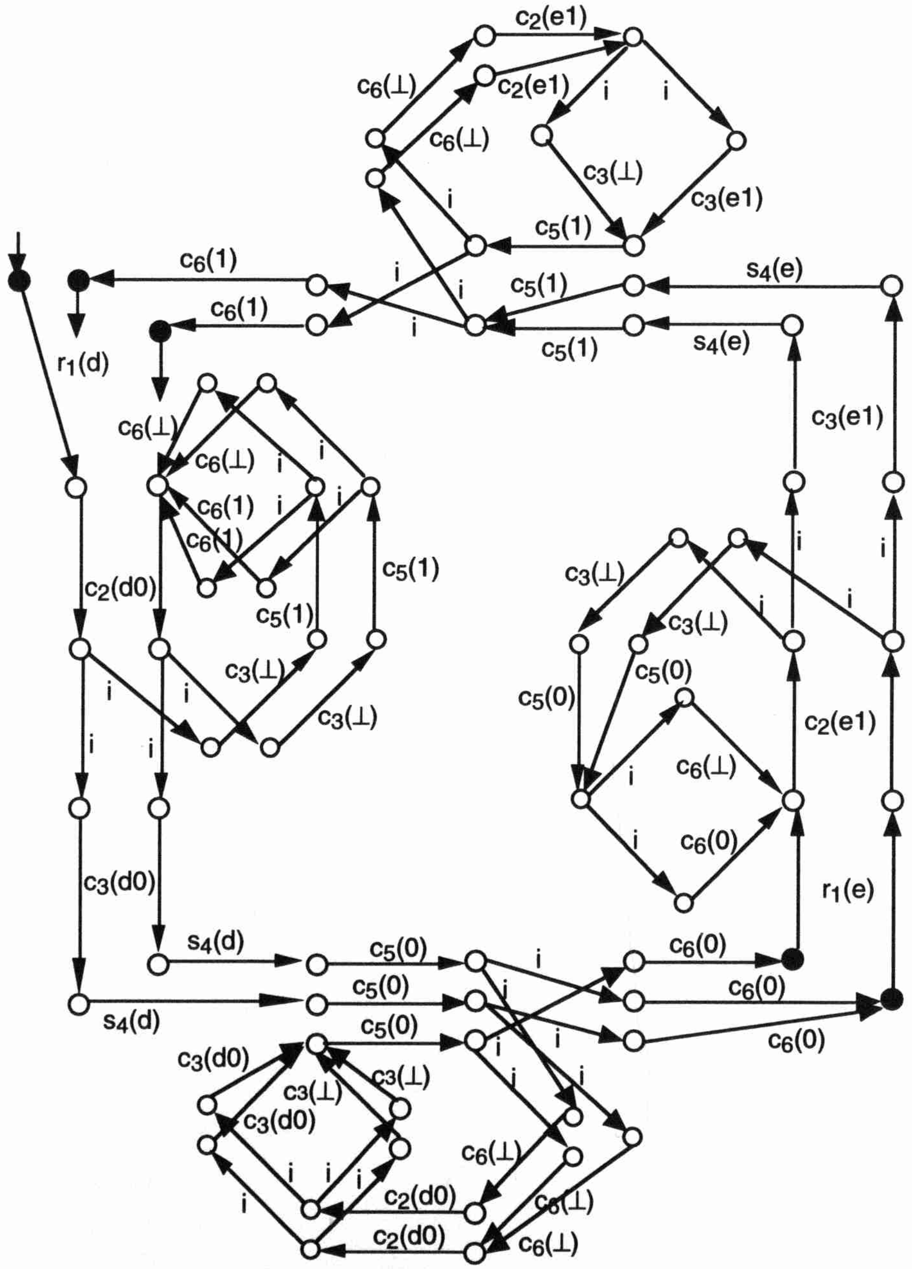

Figure 10. Initialisation phase of ABP.

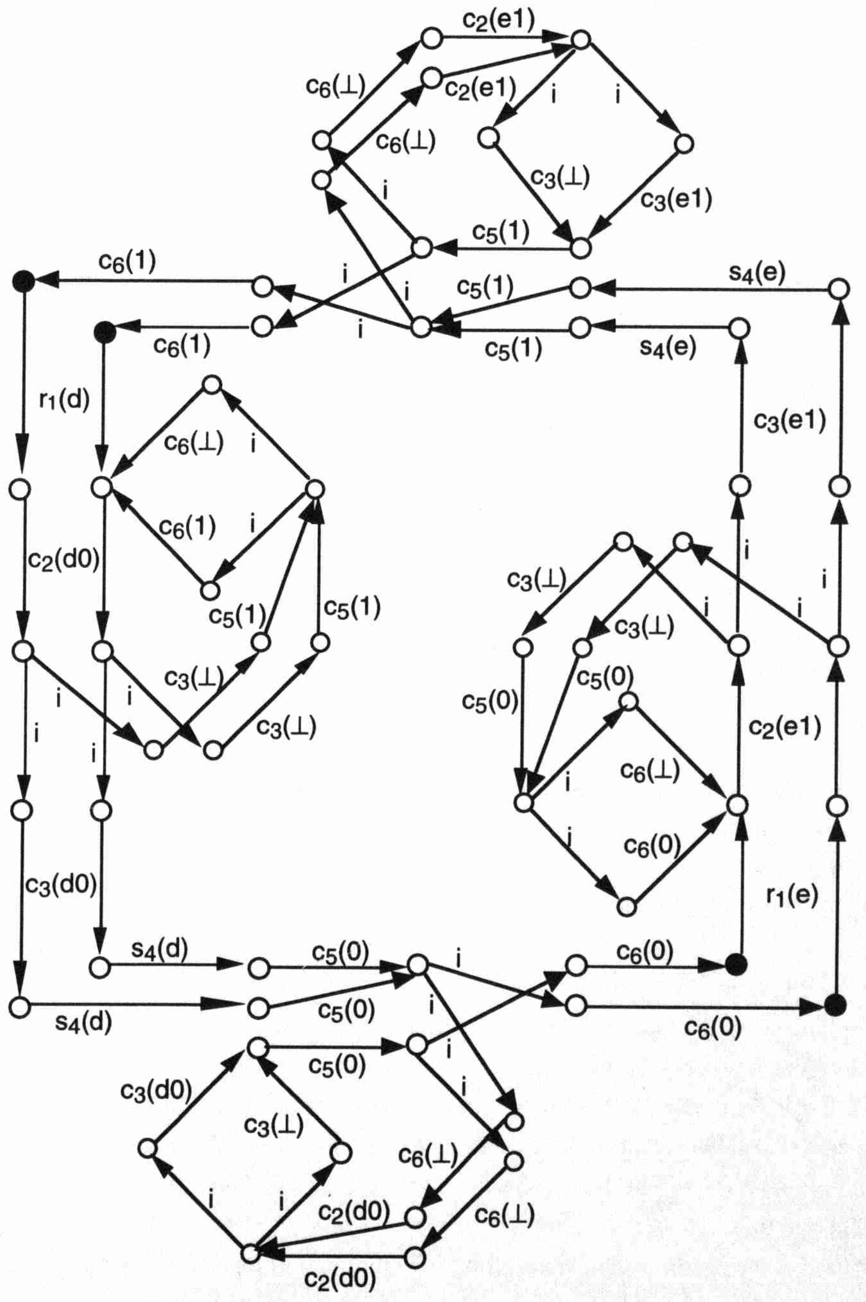

Figure 11. Iteration phase of ABP.

6. Conditions.

We can redo the theory of the previous sections in case we have conditions over a general boolean algebra. We use the theory developed in [2], and just mention a few key items, glossing over many details.

6.1 Boolean algebra.

Let $\mathbb{B}$ be a boolean algebra, with constants true, false and operators $\vee$, $\wedge$, $\neg$ We use letters ϕ, ψ to range over $\mathbb{B}$. A *valuation* is a homomorphism from $\mathbb{B}$ into {true, false}. For each process x, there is a process $\phi :\to x$ (if ϕ, then x). We redefine the model of process graphs, in order to take conditions on edges into account.

6.2 Process graphs over a boolean algebra.

Before, we had that a process graph is a quadruple $\langle S, \to, \text{begin}, \text{end}\rangle$ with $\to \subseteq S \times A \times S$, or, equivalently, the transition relation is a mapping from $S \times A \times S$ into {true, false}. Over a general boolean algebra $\mathbb{B}$, the transition relation is a mapping from $S \times A \times S$ into $\mathbb{B}$, and thus $(s \xrightarrow{a} t) \in \mathbb{B}$. The conditions on the transition relation are reformulated as follows:

- for all valuations v, states s $|\{t \in S : \exists a \in A\ v(s \xrightarrow{a} t) = \text{true}\}| < \lambda$,
- $\forall s \in S\ \forall a \in A\ (s \xrightarrow{a} \text{begin}(g)) = \text{false}$,
- $\forall s \in S\ \forall a \in A\ (\text{end}(g) \xrightarrow{a} s) = \text{false}$.

Isomorphism between two graphs g,h is now defined as expected. A bijection F between states of g and states of h is called an *isomorphism* if:

1. $F(\text{begin}(g)) = \text{begin}(h)$, $F(\text{end}(g)) = \text{end}(h)$
2. for all valuations v, $v(s \xrightarrow{a} t) = v(F(s) \xrightarrow{a} F(t))$.

Again, g,h are *isomorphic*, $g \approx h$, if there is an isomorphism between g and h.

6.3 Operators.

Definition of the operators in 2.3 is fairly straightforward. Writing down four interesting cases, condition b. in the definition of g+h becomes:

b'. $\text{begin} \xrightarrow{a} \text{end} = (\text{begin}(g) \xrightarrow{a} \text{end}(g)) \vee (\text{begin}(h) \xrightarrow{a} \text{end}(h))$.

In case of parallel composition, we calculate $\langle s,t\rangle$ \o\al(^{a},$\to$) $\langle s',t'\rangle$ as the disjunction of all $(s \xrightarrow{b} s') \wedge (t \xrightarrow{c} t')$ for pairs b,c with $\gamma(b,c) = a$, plus $(s \xrightarrow{a} s')$ in case t=t', plus $(t \xrightarrow{a} t')$ in case s=s'.

In case of the conditional operator, $(\text{begin} \xrightarrow{a} s)$ in $\phi{:}\to g$ equals $\phi \wedge (\text{begin} \xrightarrow{a} s)$ in g.

Finally, in case of the priority operator, the value of a transition $s \xrightarrow{a} s'$ in $\theta_<(g)$ becomes the conjunction of its value in g and all $\neg(s \xrightarrow{b} s'')$ for b > a.

6.4 Reachability.

Let a process graph $g \in \mathcal{G}(A, \kappa, \lambda)$ be given. As in 2.4.3, we define its set of reachable states, $\mathrm{reach}(g) \subseteq S(g)$ inductively:

a. $\mathrm{begin}(g) \in \mathrm{reach}(g)$

b. if $s \in \mathrm{reach}(g)$, and there is a valuation v and an action a such that $v(s \xrightarrow{a} s') =$ true, then $s' \in \mathrm{reach}(g)$.

Again, $\xi(g)$ is obtained from g by restriction to the set of reachable states.

6.5 Bisimulation.

Conditions 2 and 3 of the definition of bisimulation in 3.1 must be reformulated as follows (cf. [BAB92], section 8.2):

2'. if $R(s, t)$ and v is a valuation such that $v(s \xrightarrow{a} s') = \mathrm{true}$, then there is a t' such that $v(t \xrightarrow{a} t') = \mathrm{true}$ and $R(s', t')$;

3'. if $R(s, t)$ and v is a valuation such that $v(t \xrightarrow{a} t') = \mathrm{true}$, then there is a s' such that $v(s \xrightarrow{a} s') = \mathrm{true}$ and $R(s', t')$.

6.6 Expressivity.

We can still obtain an analogue of the expressivity result of section 4 in the present setting. Basically, the steps in 4.4, 4.5 remain the same, giving the required number of nodes, with exactly one transition between each pair (with first component not end, second component not begin). Next, we do not proceed as in 4.6, but instead, handle each node pair separately. Let $\langle n,m\rangle$ be a node pair. Since ϕ is a bijection, we can take certain nodes s,t in F with $n = \phi(s)$, $m = \phi(t)$. In case all transitions between s and t in F have condition false, do nothing. Otherwise there are $K \geq 1$ edges between s and t in g, with conditions $\phi_1, ..., \phi_K$ different from false. Take fresh atomic actions $b_0, b_1, ..., b_K$ in B.

We now consider two different cases. First, the case where $n = s = \mathrm{begin}$ or $m = t = \mathrm{end}$. In this case, we know that a transition under consideration can be executed at most once. In this case, we can put $G_6 = \xi \circ \partial_H(G_3 \parallel (\phi_1 :\rightarrow b_1 + ... + \phi_k :\rightarrow b_k))$, where the encapsulation enforces a communication between the edge between n and m in G_3 and one of the b_i. This replaces the step in 4.6. Next, we proceed by a renaming as in 4.7. Otherwise, the transition under consideration can be executed several times, and we have to use an iteration construct. We also need a termination clause. We take in this case $G_6 = \xi \circ \partial_H(G_3 \parallel (\phi_1 :\rightarrow b_1 + ... + \phi_k :\rightarrow b_k)^* b_0)$,
where b_0 will synchronize with each termination step (again enforced by encapsulation).

7. Conclusion.

We have presented a simple and intuitive model for the syntax of ACP. This model is non-interleaving. We can use this model to calculate the number of states of a process. We found that this number of states depends very much on the representation of a process in the syntax. The most intuitive representation usually does not yield the minimal number of states. We have presented an expressivity result, that shows that every finite state graph in the model can be expressed in our syntax, starting from one seed process.

References.

1. J.C.M. Baeten & J.A. Bergstra. Global renaming operators in concrete process algebra. Inf. & Comp. 1988; 78:205-245.

2. J.C.M. Baeten & J.A. Bergstra. Process algebra with signals and conditions. In: M. Broy (ed.), Programming and mathematical method, Proc. Summer School, Marktoberdorf 1990, Springer Verlag 1992, pp. 273-323 (NATO ASI Series F 88).

3. J.C.M. Baeten & J.A. Bergstra. Non interleaving process algebra. In: E. Best (ed.), Proc. CONCUR'93, Hildesheim, Springer Verlag 1993, pp. 308-323 (Lecture Notes in Computer Science 715).

4. J.C.M. Baeten & W.P. Weijland. Process algebra. Cambridge Tracts in Theoretical Computer Science 18, Cambridge University Press 1990.

5. J.A. Bergstra, I. Bethke & A. Ponse. Process algebra with combinators. Technical Report P9319, Programming Research Group, University of Amsterdam 1993.

6. J.A. Bergstra, I. Bethke & A. Ponse. Process algebra with iteration and nesting. The Computer Journal 1994; 37 (to appear).

7. J.A. Bergstra & J.W. Klop. Process algebra for synchronous communication. Inf. & Control 1984; 60:109-137.

8. J.A. Bergstra & J.W. Klop. Algebra of communicating processes with abstraction. Theor. Comp. Sci. 1985; 37:77-121.

9. W.J. Fokkink & H. Zantema. Basic process algebra with iteration: completeness of its equational axioms. The Computer Journal 1994; 37 (to appear).

10. J. Parrow. Fairness properties in process algebra – with applications in communication protocol verification. Ph.D. Thesis, DoCS 85/03, Dept. of Computer Systems, Uppsala University 1985.