

ISSN 0105-8517

**SPECIFICATION and VERIFICATION of
NETWORKS in a PETRI NET based LANGUAGE**

Morten Kyng

DAIMI PB-153
October 1982

PB-153

M. Kyng: Specification and Verification of Networks

TRYK: DAIMI/RECAU

ience Der
1.1

SPECIFICATION and VERIFICATION of
NETWORKS in a PETRI NET based LANGUAGE

Morten Kyng
Computer Science Department
Aarhus University, Ny Munkegade
DK-8000 Aarhus C, Denmark

Abstract

In this paper we present a system description language and a technique of top-down specification and verification of distributed systems. Our language is called Epsilon, and it has been developed for the description and analysis of systems containing concurrent components. We have used and developed concepts from the Simula [2] and Delta [5] languages and from Petri net theory [1]. A system described in Epsilon consists of a number of concurrent objects. Each object has a set of attributes, e.g. variables and procedures, and executes a sequence of actions. Epsilon includes both normal algorithmic statements and first order predicate logic as description elements.

In the paper we bring out the main features of the language and of our specification and verification technique through an example. We describe a simple ring-structured transport system by a stepwise specification and verification of its properties. Properties of all reachable states (partial correctness) are expressed directly in the Epsilon description by first order predicate logic. Proof of partial correctness is a matter of proving that the Epsilon-description is consistent. Properties concerning progress are not specified directly in the Epsilon description, but stated separately. Proof of total correctness is a matter of proving that the Epsilon description meets the progress specifications.

The semantics of Epsilon is defined by means of a model based on high-level Petri nets, i.e. a model founded on the notion of concurrency. This implies that our proof techniques a priori cover concurrency in the specified systems. The use of high-level Petri nets keeps the size of the nets relatively small. The complexity of the proofs is further reduced by transforming the logic expressions specifying partial correctness into a "local" form. Our technique avoids the "state-space explosion" often encountered in verification based on finite state models. With more complicated systems the "factorization" of the proofs becomes increasingly important. However, the proof method works for non-local expressions as well, which gives one the possibility of allowing a few "global" expressions, e.g. if these do not have any natural local counterparts.

An abbreviated version of this paper has been presented at the third European Workshop on Applications and Theory of Petri Nets. A similar system is analysed by Owicki in [14].

Keywords: *communication networks, high-level Petri nets, specification, verification, distributed systems, concurrency.*

1. Specification of safe states

The example which we consider throughout this paper, is a simple ring-structured transport system in which packets are sent in one direction only. The system is composed of L aggregates, each of which may produce a packet for consumption by any other aggregate. An aggregate consists of a producer, PROD, an output-process, OUT, an input-process, IN, and a consumer, CONS. The OUT of aggregate i is connected to the IN of aggregate $i \oplus 1$ (where \oplus is cyclic addition in $1..L$). This, and the internal connections, are depicted in Fig. 1:

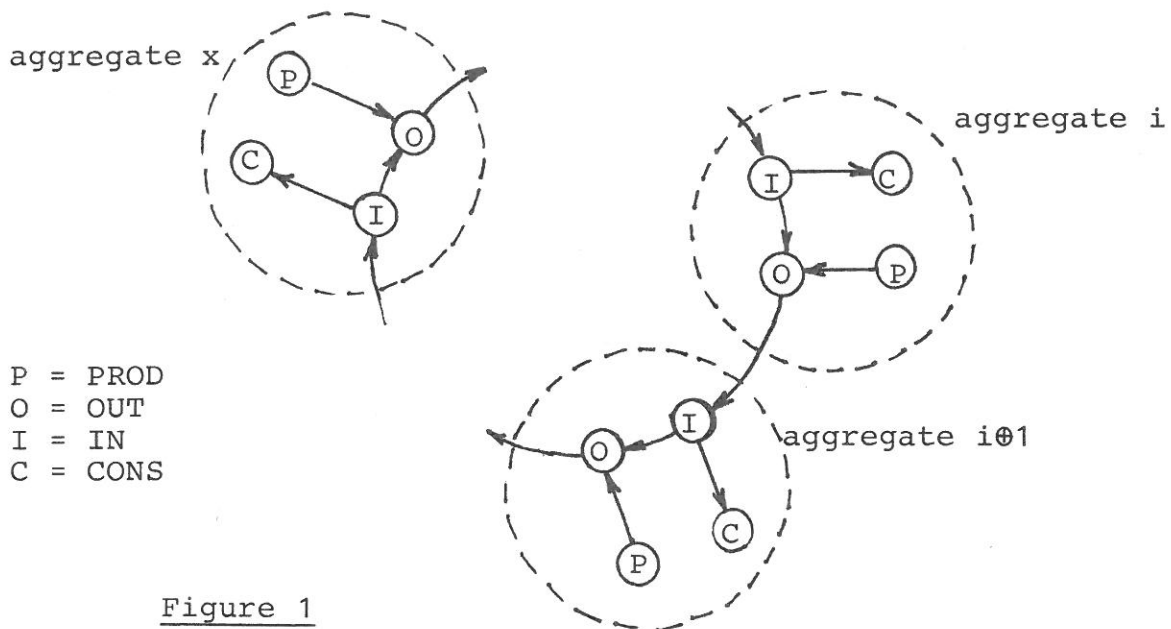


Figure 1

Stated simply, we want our system to behave in such a way that the packets sent from a producer to a consumer are delivered in the order in which they were sent without loss.

This rather informal description of the network is the basis for our first Epsilon description, given in Fig. 2. A system described in Epsilon consists of a number of objects, which each has a set of attributes and executes a sequence of actions. The attributes of an object may be constants, variables, types, functions, procedures, tasks and internal objects. The objects may synchronize their actions with each other via variables or by direct communication. The actions of an object consists of an alternating sequence of equational-actions and event-actions.

Most of the time objects execute equational-actions. Such an action is specified by an equation consisting of a list of changeable variables and a predicate to be kept unceasingly satisfied during the execution of the action. When a number of objects concurrently execute equational-actions the conjunction of their predicates constitutes the effective predicate, while the union of their variable-lists constitutes the set of changeable variables. Informally the semantics of a set of equational-actions concurrently executed by different objects can be described as follows: the effective predicate is kept unceasingly satisfied by varying the value of the changeable variables. In other words, the effective predicate defines an equation system, which is solved with respect to the set of changeable variables and these are changed accordingly. If several solutions exist one is chosen non-deterministically.

Event actions are instantaneous and specified by sequences of algorithmic commands. Each event-action constitutes one indivisible state transformation. The effect of an event-action does not depend on concurrently

executed actions. However in some cases two or more objects jointly execute a single event-action. The purpose of the description in Fig. 2 is to specify the safe states of the system, i.e. the properties which we want all states of the system to fulfill. Safe states have something of the flavour of partial correctness. The description contains some type and constant declarations and a number of different objects: PRODucers, OUTput-processes, INput-processes, and CONSumers. The elements transported by the system are PACKETs.

```

1  RING TRANSMISSION - VERSION 1:
2  (system
3      type N: 1..L; type M: finite
4      type PACKET: (record SOURCE : N
                        DEST      : N
                        MESSAGE: M record)
7      type PACKETSEQ: seq PACKET

8      PROD(ID:N):
9      (object
10         SENT: PACKETSEQ init EMPTY
11         (* SENT |  $\pi$ (ID,.)SENT *)
12         object)

13     OUT(ID:N), IN(ID:N):
14     (object
15         BUF: PACKETSEQ init EMPTY
16         (* BUF *)
17         object)

18     CONS(ID:N):
19     (object
20         REC: PACKETSEQ init EMPTY
21         (* REC |  $\forall i \in N$  [ $\Delta(i,.)$ REC $\neg \Delta(i, ID)$ route(i, ID)
22                                =  $\Delta(., ID)$ PROD(i).SENT] *)
23         object)

24  system)

```

Figure 2

In lines (8-12) L PRODucer objects are declared, each having a constant, ID, indicating that object. Each PRODucer has a sequence variable SENT as data attribute. It represents the sequence of PACKETs sent by the PRODucer. SENT is initialized to the empty sequence. A PRODucer executes only one action, described by the equational-statement (11). Such a statement is comprised of a list of changeable variables, a vertical bar and a predicate, enclosed in round, starred brackets. Its execution consists of changing the values of the variables to the left of the

vertical bar, i.e. SENT, in such a way that the predicate to the right, $\pi(ID, \cdot) \text{SENT}$, is satisfied. The functions π , and Δ used in (21-22), are defined on subsets of producer and consumer identifiers and yield functions on PACKETSEQUENCES:

$$\pi: P(N) \times P(N) \rightarrow \text{PACKETSEQ} \rightarrow \{\text{true}, \text{false}\}$$

$$\Delta: P(N) \times P(N) \rightarrow \text{PACKETSEQ} \rightarrow \text{PACKETSEQ}$$

The function $\pi(P_1, C_1)$ is used to test that all packets in a sequence has a SOURCE in P_1 and a DESTINATION in C_1 ; $\Delta(P_1, C_1)$ picks out the maximal subsequence with this property. Formally we define

$$\pi(P_1, C_1)\lambda \text{ is true, } \lambda \text{ is the empty sequence}$$

$$\pi(P_1, C_1)\sigma\rho \text{ is true iff } \pi(P_1, C_1)\sigma \text{ and } \rho.\text{SOURCE} \in P_1 \text{ and } \rho.\text{DEST} \in C_1, \sigma \in \text{PACKETSEQ}, \rho \in \text{PACKET}.$$

$\Delta(P_1, C_1)\sigma$ is the maximal subsequence, σ_m , of σ such that $\pi(P_1, C_1)\sigma_m$ is true. By convention $\pi(P_1, \cdot) = \pi(P_1, N)$; $\pi(P_1, c) = \pi(P_1, \{c\})$; and $\pi(P_1, \sim c) = \pi(P_1, N \setminus \{c\})$. Analogous conventions are used for the first parameter, for both in combination and for the function Δ .

Returning to Fig. 2, we may paraphrase the predicate $\pi(ID, \cdot) \text{SENT}$ in line (11) in the following way: all PACKETS SENT have the right SOURCE value.

In lines (13-17) L OUTPUT objects and L INPUT objects are declared. Each has a sequence variable BUF and executes an equational-statement, changing the value of BUF. It is not specified how the value is changed.

Each of the L CONSUMERS, (18-23), has a sequence variable, REC, which represents the sequence of PACKETS received. A CONSUMER executes one equational-statement, changing the value of REC in such a way that the predicate of the statement is satisfied. The predicate may be paraphrased as follows: The PACKETS RECEIVED from PROD(i) concatenated with the PACKETS on route(i, ID) having SOURCE=i and DEST=ID equals the PACKETS SENT by PROD(i) to CONS(ID). The function route: $N \times N \rightarrow \text{PACKETSEQ}$ concatenates the BUFFER sequences of the OUTPUTS and INPUTS on the route from a PRODUCER to a CONSUMER. It is defined in the following way (cf. Fig. 1, with $x\theta k=i$):

$$\begin{aligned} \text{route}(x\theta k, x) &= \text{IN}(x) \cdot \text{BUF} \wedge \\ &\quad \text{OUT}(x\theta 1) \cdot \text{BUF} \wedge \text{IN}(x\theta 1) \cdot \text{BUF} \wedge \\ &\quad \vdots \\ &\quad \text{OUT}(x\theta(k-1)) \cdot \text{BUF} \wedge \text{IN}(x\theta(k-1)) \cdot \text{BUF} \wedge \\ &\quad \text{OUT}(x\theta k) \cdot \text{BUF} \end{aligned}$$

The sequence variables SENT and REC of the PRODUCERS and CONSUMERS respectively, are needed to specify the safe states. In our final description, Fig. 7, they will function as history variables, [6], [15], and need not appear in a program implementing the specification.

Semantics

In [10] we define the semantics of Epsilon by means of a syntax-directed translation into Equation nets, which are high-level Petri nets, [3], [8], [9], with equations attached to the places. In this paper we do not define the semantics of Epsilon, but, for each Epsilon description, simply present the corresponding Equation net. In such a net places correspond to equational-statements and transitions to event-statements (these are discussed in a succeeding section). The semantics of the description in Fig. 2 is the Equation net in Fig. 3. In this paper we


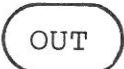
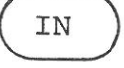
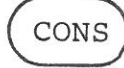
avoid aliasing, and this allows us to use a simplified form of net inscriptions, which combine the environment and store described in [10]. Furthermore we do not include the part of the net corresponding to the system object, since this plays no part in our analysis.

The net consists of four isolated places, PROD, OUT, IN and CONS, corresponding to the four equational-statements in Fig. 2. (Since the description in Fig. 2 has no event-statements, the net has no transitions.) Each place has attached a set of possible token-colours. All tokens marking a place have a colour (information content) belonging to the colour set of the place. In our Equation nets a token-colour always describes the state of the local data attributes of an object.

Tokens on place PROD are of the form

$$(i, \sigma_i^{ps}) \in N \times \text{PACKETSEQ}$$

where the first component represents the PRODucer IDentifier and the second SENT. To stress the relationship we may write ID*SENT instead of N*PACKETSEQ. ID*SENT is referred to as the token-colour set of PROD. The other places have token-colour sets with analogous interpretations. Below the following conventions are used: "i" ranges over PRODucers and OUTputs, "x" over INputs and CONSumers.

$(i, \sigma_i^{ps}) \in \text{ID} \times \text{SENT}$		$\{\sigma_i^{ps} \mid \Pi(i, \cdot) \sigma_i^{ps}\}$
$(i, \sigma_i^{ob}) \in \text{ID} \times \text{BUF}$		$\{\sigma_i^{ob}\}$
$(x, \sigma_x^{ib}) \in \text{ID} \times \text{BUF}$		$\{\sigma_x^{ib}\}$
$(x, \sigma_x^{cr}) \in \text{ID} \times \text{REC}$		$\{\sigma_x^{cr} \mid \forall i \in N [\Delta(i, \cdot) \sigma_x^{cr} \frown \Delta(i, x) \text{route}(i, x)$ $= \Delta(\cdot, x) \sigma_i^{ps}]\}$

The initial marking $m_0 = (\sum_{i \in N} (i, \lambda); \sum_{i \in N} (i, \lambda); \sum_{x \in N} (x, \lambda); \sum_{x \in N} (x, \lambda))$

Figure 3

In addition to a token-colour set an equation enclosed in braces is attached to each place. An equation consists of a list of changeable variables, a vertical bar, and a predicate. It is a straightforward translation of the equation of the equational-statement corresponding to the place. In our definition of the initial marking, m_0 , above, the sums refer to the markings of PROD, OUT, IN and CONS in that order.

Analysis of VERSION 1

We analyse the Epsilon description by considering the set of possible behaviours of the net above. For the purpose of this paper the following simplified definition is sufficient:

The behaviours, B , of an Equation net is the set of marking sequences which start in the initial marking and satisfies

$$B = \{(m_i)_{0 \leq i < n}, 0 < n \leq \infty \mid m_{i-1} \rightarrow m_i \text{ or } m_{i-1} \rightsquigarrow m_i \text{ for } 0 < i < n\},$$

where \rightarrow as usual denotes the firing of a set of transitions having concurrent concession, and \rightsquigarrow denotes a change only in the colour of some tokens within the limits of the equations of the marked places. \rightsquigarrow is defined in such a way that it can be applied to dead markings only, i.e.

$$m_{i-1} \rightsquigarrow m_i \Rightarrow \neg(\exists m': m_{i-1} \rightarrow m')$$

When we are not interested in the sequence in which markings appear, we may consider the reachability set, R , instead of the behaviours. A marking is reachable iff it is contained in a behaviour. R consists of all reachable markings. Throughout this paper we consider reachable markings only.

We say that a marking is consistent iff it satisfies the predicates of the equations attached to all marked places. An Equation net and (if the net is a translation) the corresponding Epsilon description, is consistent iff its reachable markings are consistent. Notice that if a non-consistent marking is reached it will always be via a transition firing, " \rightarrow ", and not a change in colour only, " \rightsquigarrow ".

All we want to show about our first description is that it is consistent. This is trivial:

Theorem 1 VERSION 1, Fig. 2, is consistent.

Proof We must show that the reachable markings of the net in Fig. 3 are consistent. But this follows directly from the definition of " \rightsquigarrow " and the facts that the initial marking is consistent and that the net contains no transitions. \square

2. Local predicates

Some specification and verification techniques restrict the use of variables in predicates (invariants) to special kinds of local variables, when dealing with concurrent processes, [14]. This may complicate the task of making the first formal specification of an informally conceived system; and in turn obscure the inherently unformalizable process of convincing oneself that the first formalization is adequate. Our semantic model does not impose such restrictions. The CONSUMER predicates rather directly reflect our informal understanding of a system that transmits packets from producers to consumers, which means that it is a relatively simple task to convince oneself that the description is an adequate formal specification. On the other hand, the CONSUMER predicates tie together all objects in the system, which complicates both verification and modification. Therefore we shall give a modified description with local predicates; we give a description in which the predicate of an object only involves variables of the object itself and of objects from which it receives PACKETS (cf. Fig. 1).

To this end we introduce a pair of auxiliary sequences, REC and SENT, in each OUTPUT and INPUT. These play roles similar to REC and SENT of CONSUMERS and PRODUCERS. The new version is presented in Fig. 4.

The description of the PRODUCERS is not changed, since the predicate used in their specification only involves local variables. OUT is supplemented by predicates stating that what is SENT concatenated with the contents of the BUFFER is equal to what is RECEIVED. And that the

sequence RECeived is an element in the set of sequences consisting of $\Delta(\cdot, \sim ID) IN(ID).SENT$ merged with $PROD(ID).SENT$ (17), i.e. PACKETS with a DESTination different from ID, SENT from $IN(ID)$ merged with PACKETS SENT from $PROD(ID)$. The predicates of the INputs is similar, but a bit simpler, since they only RECeive PACKETS from the preceding OUTput. The new CONSUMER predicate states that what $CONS(ID)$ RECeives is equal to what $IN(ID)$ has SENT with DESTination equal to ID.

```

1  RING TRANSMISSION - VERSION 2:
2  (system
3    type N: 1..L; type M: finite
4    type PACKET: (record SOURCE : N
5                      DEST    : N
6                      MESSAGE: M    record)
7    type PACKETSEQ: seq  PACKET

8    PROD(ID:N):
9    (object
10     SENT: PACKETSEQ init EMPTY
11     (* SENT |  $\pi(ID, \cdot) SENT$  *)
12    object

13    OUT(ID:N):
14    (object
15     SENT, BUF, REC: PACKETSEQ init EMPTY
16     (* SENT, BUF, REC |  $SENT \sim BUF = REC$ ,
17                         $REC \in \{\Delta(\cdot, \sim ID) IN(ID).SENT \parallel PROD(ID).SENT\}$  *)
18    object)

19    IN(ID:N):
20    (object
21     SENT, BUF, REC: PACKETSEQ init EMPTY
22     (* SENT, BUF, REC |  $SENT \sim BUF = REC$ ,  $REC = OUT(ID \oplus 1).SENT$  *)
23    object)

24    CONS(ID:N):
25    (object
26     REC: PACKETSEQ init EMPTY
27     (* REC |  $REC = \Delta(\cdot, ID) IN(ID).SENT$  *)
28    object)
29  system)

```

Figure 4

Analysis of VERSION 2

The net of VERSION 2 has four places, like the net of VERSION 1, one corresponding to each of the equational-statements. The token-colour components representing the sequence variables SENT and RECEIVED. The four place equations are direct translations of the equations of the corresponding statements.

$$\begin{aligned}
 (i, \sigma_i^{ps}) &\in ID \times SENT \quad \text{PROD} \quad \{\sigma_i^{ps} \mid \pi(i, \cdot) \sigma_i^{ps}\} \\
 (i, \sigma_i^{os}, \sigma_i^{ob}, \sigma_i^{or}) &\in ID \times SENT \times BUF \times REC \quad \text{OUT} \quad \{\sigma_i^{os}, \sigma_i^{ob}, \sigma_i^{or} \mid \sigma_i^{os} \sigma_i^{ob} = \sigma_i^{or}, \\
 &\quad \sigma_i^{or} \in [\Delta(\cdot, \sim i) \sigma_i^{is} \parallel \sigma_i^{ps}]\} \\
 (x, \sigma_x^{is}, \sigma_x^{ib}, \sigma_x^{ir}) &\in ID \times SENT \times BUF \times REC \quad \text{IN} \quad \{\sigma_x^{is}, \sigma_x^{ib}, \sigma_x^{ir} \mid \sigma_x^{is} \sigma_x^{ib} = \sigma_x^{ir}, \\
 &\quad \sigma_x^{ir} = \sigma_{x\theta 1}^{os}\} \\
 (x, \sigma_x^{cr}) &\in ID \times REC \quad \text{CONS} \quad \{\sigma_x^{cr} \mid \sigma_x^{cr} = \Delta(\cdot, x) \sigma_x^{is}\}
 \end{aligned}$$

$$\text{Initial marking } m_0 = \left(\sum_{i \in N} (i, \lambda); \sum_{i \in N} (i, \lambda, \lambda, \lambda); \sum_{x \in N} (x, \lambda, \lambda, \lambda); \sum_{x \in N} (x, \lambda) \right)$$

Figure 5

First we observe that VERSION 2 is consistent, and then we show that it correctly implements VERSION 1.

Theorem 2 VERSION 2, Fig. 4, is consistent.

Proof As for VERSION 1 this follows directly from the definition of " \sim " and the facts that the initial marking is consistent and that the net contains no transitions. \square

Implementation

As we develop a sequence of descriptions which are gradually more detailed, we need to prove that a description A is correctly implemented by its successor B. Intuitively, we must map B onto A in some reasonable way. Formally we do this on the Equation net level. We define a mapping from the places of B to the places of A and a mapping from markings of B to markings of A. Then the reachability set of B must be mapped into the reachability set of A:

Let two Equation nets A and B, with places P_A and P_B , colour-functions C_A and C_B and initial markings m_A and m_B be given together with two mappings: $h_p: P_B \rightarrow P_A$, and h_c defined on P_B such that

$$\forall p_B \in P_B [h_c(p_B): C_B(p_B) \rightarrow C_A(h_p(p_B))].$$

$h_c(p_B)$ maps token-colours of $p_B \in P_B$ into token-colours of $h_p(p_B) \in P_A$. h_c can uniquely be extended to a linear function which maps sets of markings of B on sets of markings of A. For convenience we shall denote this extended function also h_c . This should cause no confusion.

Definition

B implements A with respect to (h_p, h_c) iff $R_A \subseteq h_c(R_B)$.

Theorem 3

where:

VERSION 2 implements VERSION 1 with respect to $(id, proj)$

id : maps a place of VERSION 2 on the place of VERSION 1 with the same name

$proj$: skips the SENT and REC component of OUT and IN token-colours, otherwise it is the identity mapping.

Proof

We have to prove that $proj$ maps any reachable marking m (of VERSION 2) into a reachable marking (of VERSION 1).

The only non-trivial part of this is to prove that the CONS predicates of VERSION 1 are fulfilled, i.e. that $proj(m)$ fulfills:

$$\forall i, x \in N [\Delta(i, \cdot) \sigma_x^{cr} \wedge \Delta(i, x) route(i, x) = \Delta(\cdot, x) \sigma_i^{ps}]$$

First we prove (intuitively "no packets cycle in the ring"):

Lemma

Any reachable marking, m , of VERSION 2 satisfies the predicate:

$$\Delta(i, x) \sigma_i^{ps} = \Delta(i, x) \sigma_i^{or}$$

Proof

From the OUT predicate we get

$$\sigma_i^{or} \in [\Delta(\cdot, \sim i) \sigma_i^{is} \parallel \sigma_i^{ps}]$$

so it is sufficient to prove that $\Delta(i, \sim i) \sigma_i^{is} = \lambda$. We do this by contradiction.

Assume that $\rho \in \text{PACKET}$ is contained in $\Delta(i, \sim i) \sigma_i^{is}$, i.e. $\rho = (i, i \oplus k, a)$, $k \in 1..(L-1)$.

From the PROD predicates we get that ρ is not contained in $\sigma_{i \oplus 1}^{ps}, \dots, \sigma_{i \oplus k}^{ps}$. From the IN and OUT predicates we get that ρ is contained in $\sigma_{i \oplus 1}^{or}, \dots, \sigma_{i \oplus k}^{or}$, and finally that ρ is contained in $\Delta(\cdot, \sim (i \oplus k)) \sigma_{i \oplus k}^{is}$. But this contradicts our initial assumption that $\rho = (i, i \oplus k, a)$. \square

Then, using this, we prove that m itself satisfies the CONS predicates of VERSION 1.

$$\begin{aligned} & \Delta(\cdot, x) \sigma_{x \oplus k}^{ps} \\ &= \Delta(x \oplus k, x) \sigma_{x \oplus k}^{ps} && \text{(m fulfills PROD predicate of VERSION 2)} \\ &= \Delta(x \oplus k, x) \sigma_{x \oplus k}^{or} && \text{(lemma)} \\ &= \Delta(x \oplus k, x) \left[\sigma_{x \oplus k}^{os} \sigma_{x \oplus k}^{ob} \right] && \text{(OUT predicate)} \\ &= \Delta(x \oplus k, x) \left[\sigma_{x \oplus (k-1)}^{ir} \sigma_{x \oplus k}^{ob} \right] && \text{(IN predicate)} \\ &= \Delta(x \oplus k, x) \left[\sigma_{x \oplus (k-1)}^{is} \sigma_{x \oplus (k-1)}^{ib} \sigma_{x \oplus k}^{ob} \right] && \text{(IN)} \\ &= \Delta(x \oplus k, x) \left[\sigma_{x \oplus (k-1)}^{or} \sigma_{x \oplus (k-1)}^{ib} \sigma_{x \oplus k}^{ob} \right] && \text{(PROD and OUT)} \\ & \vdots \end{aligned}$$

$$\begin{aligned}
&= \Delta(x\theta k, x) \left[\sigma_x^{is} \sigma_x^{ib} \sigma_{x\theta 1}^{ob} \dots \sigma_{x\theta k}^{ob} \right] \\
&= \Delta(x\theta k, \cdot) \Delta(\cdot, x) \sigma_x^{is} \Delta(x\theta k, x) \left[\sigma_x^{ib} \sigma_{x\theta 1}^{ob} \dots \sigma_{x\theta k}^{ob} \right] \\
&= \Delta(x\theta k, \cdot) \sigma_x^{cr} \Delta(x\theta k, x) \left[\sigma_x^{ib} \dots \sigma_{x\theta k}^{ob} \right] \quad (\text{CONS}) \\
&= \Delta(x\theta k, \cdot) \sigma_x^{cr} \Delta(x\theta k, x) \text{route}(x\theta k, x)
\end{aligned}$$

Finally we have to show that $\text{proj}(m)$ also fulfills the above predicates, but this follows directly because proj does not change (skip) any of the involved token-colour components. \square

3. Communication and progress

In the next step we describe the transfer of PACKETS explicitly. As we shall see, the description restricts the behaviours of the system in a way that ensures progress of information flow. (The descriptions of VERSION 1 and 2 in fact allowed the system to "run backwards"). Furthermore, since this is the last description of a system intended for implementation as a set of computer programs, we want all equational-statements to be without any effect on the state of the system. This is achieved simply by demanding that all equational-statements have empty variable lists. (Such statements play a role similar to that of invariants in Hoare-style proof techniques, see e.g. [14].) Descriptions of this kind are well suited for implementation in a programming language with CSP-like communication primitives, [4], [7].

In order to simplify the analysis of progression we now assume that all PACKETS are produced before the transmission begins and contained in a sequence variable NOTS, "not sent". The essence of this restriction is that once a message has been wrapped up as a packet at the transport level (corresponding to the producers/consumers) the system will try to deliver it until it succeeds (or deadlocks) - a packet cannot be cancelled or called back. Furthermore, since this last description should be easy to implement in a programming language, we limit the size of the buffers. For the sake of simplicity we set the limit to one. The restrictions on the communications between the objects resulting from the limit on buffer size is summarized in Fig. 6, which may be paraphrased as follows: whenever a PRODUCER has PACKETS to send, it is ready to send to the OUTPUT of the aggregate. When the BUFFER of the OUTPUT is empty, it is ready to accept a PACKET etc.

The third description is presented in Fig. 7. The new constant $\text{SEQ}(N)$ of PACKETSEQUENCES contains the PACKETS to be sent by the PRODUCERS. It is for all $i \in N$ used to initialize NOTS(i) in such a way that all PACKETS have SOURCE equal to i .

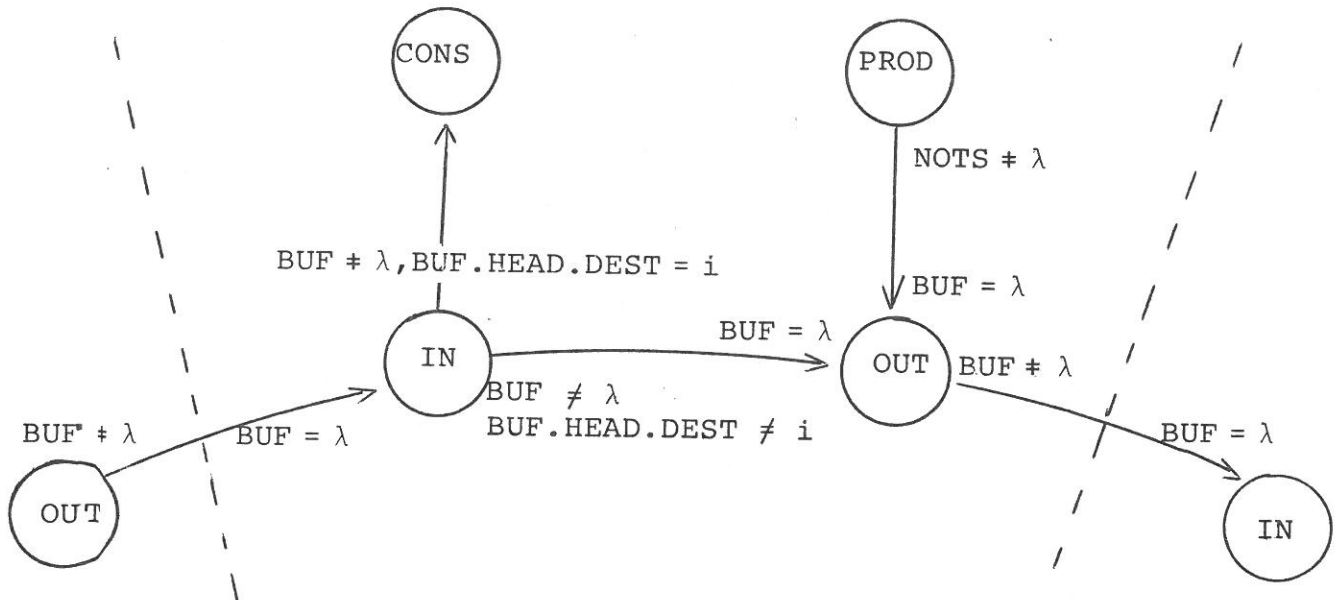


Figure 6

In the description of the PRODucers we have added a sequence variable, *NOTS*, representing those of the produced PACKETS which are not sent yet. The actions of a PRODucers are described by an equational-statement (14) controlled by an event-statement (16-17). Execution of an event-statement is always instantaneous (and indivisible). When constructing the net of an Epsilon description event-statements are translated into single transitions. A PRODucers begins its actions by executing the equational-statement. During its execution none of the variables of the PRODucers are changed since the statement has an empty variable list. But for the system to be consistent the predicate has to be fulfilled. Thus the initial state should satisfy the predicate. Execution of the equational-statement continues until the controlling event-statement is executed (see below). When this happens the procedure PUT is executed and then execution of the equational-statement continues. Thus PUT should not destroy the predicate.

The event-statement of the PRODucers, (16-17), consists of a match-clause, a when clause and a do-clause. The match-clause specifies another object and a procedure of that object (with an actual input parameter), which must be executed jointly with the local procedure specified in the do-clause. The when-clause contains a predicate, which must be satisfied, when execution begins. In this case the event-statement in PRODucers matches (28) in OUT: PROD(i).PUT is executed jointly with OUT(i).GET, and only when PROD(i).NOTS≠EMPTY and OUT(i).BUF=EMPTY.

The equational-statement of the OUTputs is controlled by three event-statements. If, in any state, it is possible to execute more than one of these, a non-deterministic choice is made. After execution of one of the controlling event-statements execution of the equational-statement continues.

The repeated declarations of identical or similar data and procedures, e.g. the three declarations of the PUT procedures, could be reduced to one by the use of the SIMULA prefix-concept. This is discussed in [10], but will not be considered in this paper.

The net of VERSION 3

The net has four places, one for each of the equational-statements. In this version all place equations consist of predicates only, and this means that they have no effect on the markings. The net has four tran-

```

1  RING TRANSMISSION - VERSION 3:
2  (system
3      type N: 1..L; type M: finite
4      type PACKET: (record SOURCE : N
5                      DEST      : N
6                      MESSAGE: M   record)
7      type PACKETSEQ: seq PACKET
8      SEQ(N): const PACKETSEQ where  $\forall i \in N[\Pi(i, \cdot) \text{SEQ}(i)]$ 

9  PROD(ID:N):
10 (object
11     SENT,NOTS: PACKETSEQ init EMPTY,SEQ(ID)
12     PUT: (procedure SENT:=^NOTS.HEAD
13           NOTS.DELETE      procedure)

14     (* SENT^NOTS = SEQ(ID) *)
15     control
16     → [* match OUT(ID).GET(NOTS.HEAD)
17        when NOTS ≠ EMPTY do PUT *]
18 object)

19 OUT(ID:N):
20 (object
21     SENT,BUF,REC: PACKETSEQ init EMPTY
22     GET(PCK:PACKET): (procedure REC,BUF:=^PCK   procedure)
23     PUT: (procedure SENT:=^BUF.HEAD
24           BUF.DELETE      procedure)

25     (*  $0 \leq |BUF| \leq 1$ , SENT^BUF = REC,
26     REC  $\in \{\Delta(\cdot, \sim ID) \text{IN} (ID). \text{SENT} \parallel \text{PROD}(ID). \text{SENT}\}$  *)
27     control
28     → [* match PROD(ID).PUT when BUF=EMPTY do GET *]

29     → [* match IN(ID).PUT when BUF=EMPTY do GET *]

30     → [* match IN(ID⊕1).GET(BUF.HEAD)
31        when BUF ≠ EMPTY do PUT *]
32 object)

```

```

33   IN(ID:N):
34   (object
35       SENT,BUF,REC: PACKETSEQ init EMPTY
36       GET(PCK:PACKET):
37       (procedure REC,BUF:=^PCK procedure)
38       PUT: (procedure SENT:=^BUF.HEAD
39               BUF.DELETE procedure)

40       (*  $0 \leq |BUF| \leq 1$ ,  $SENT \wedge BUF = REC$ ,  $REC = OUT(ID\theta 1).SENT$  *)
41       control
42       → [* match OUT(ID $\theta$ 1).PUT when BUF=EMPTY do GET *]

43       → [* match CONS(ID).GET(BUF.HEAD)
44           when BUF $\neq$ EMPTY, BUF.HEAD.DEST=ID do PUT *]

45       → [* match OUT(ID).GET(BUF.HEAD)
46           when BUF $\neq$ EMPTY, BUF.HEAD.DEST $\neq$ ID do PUT *]
47   object)

48   CONS(ID:N):
49   (object
50       REC: PACKETSEQ init EMPTY
51       GET(PCK:PACKET):
52       (procedure REC:=^PCK procedure)

53       (*  $REC = \Delta(\cdot, ID)IN(ID).SENT$  *)
54       control
55       → [* match IN(ID).PUT do GET *]
56   object)

57 system)

```

Figure 7

sitions, one for each matching pair of event-statements in Fig. 7. Transition $P \rightarrow 0$ corresponds to the match (16-17)-(28), i.e. transfer of a PACKET from PRODucer to OUTput. $I \rightarrow 0$ corresponds to the match (29)-(45-46), $0 \rightarrow I$ to (30-31)-(42), and $I \rightarrow C$ to (43-44)-(55).

In our formulas we use the following conventions: i ranges over PRODucers and OUTputs, x over INputs and CONSumers. When $P \rightarrow 0$ fires, j denotes the IDentity of the involved PRODucer and OUTput. When $0 \rightarrow I$ fires (transmission of a PACKET from one aggregate to the next) i denotes the IDentity of the OUTput and $i\theta 1$ that of the INput, etc. As before $\sigma \in \text{PACKETSEQ}$,

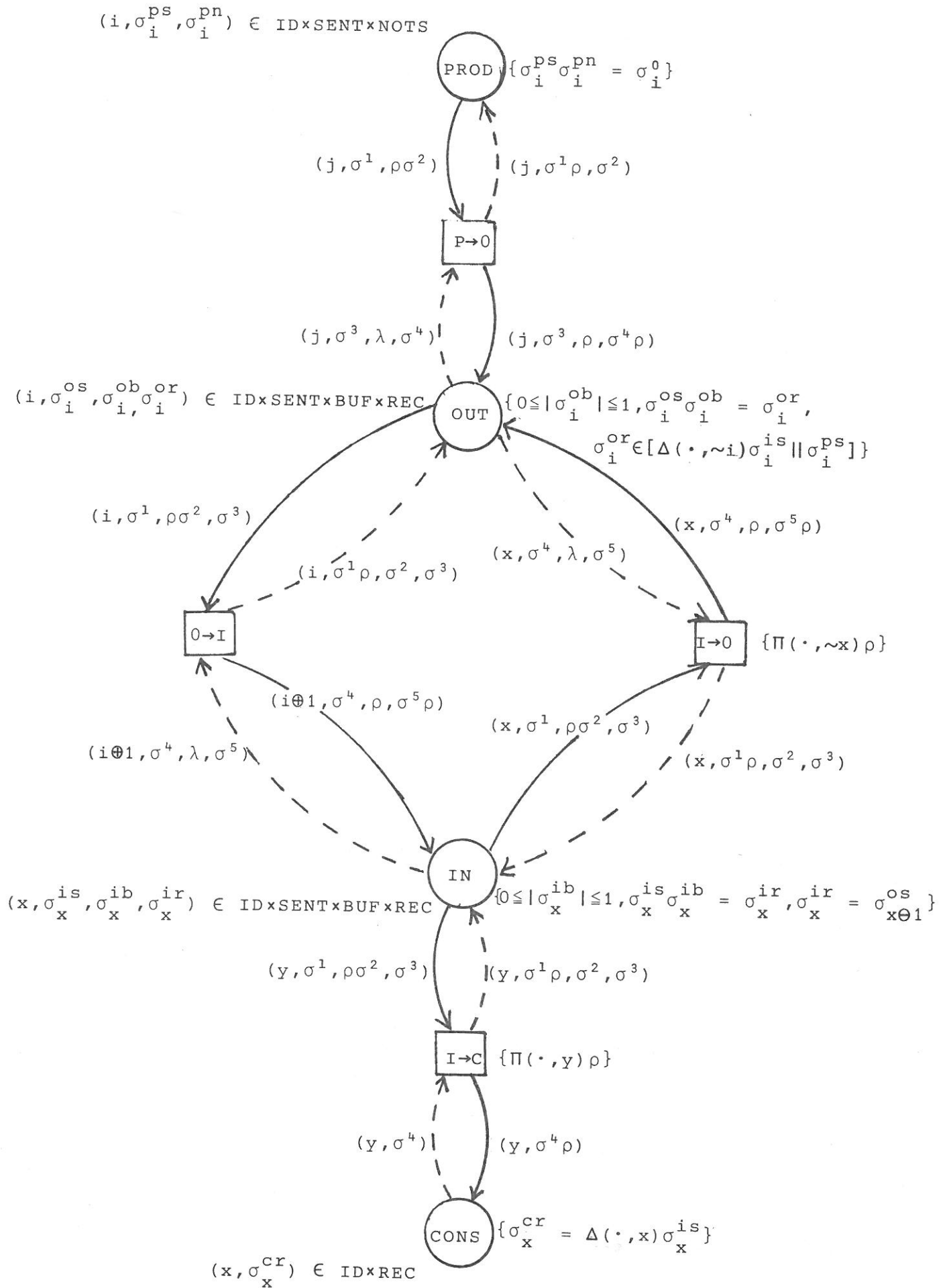


Figure 8

$\rho \in \text{PACKET}$. Formally all arrows are of the same type. However, as an aid to the reader the unbroken arrows indicate the direction of the transmission of PACKETS.

A PACKET is sent from the PRODUCER in aggregate j to the CONSUMER in aggregate y by a firing of $P \rightarrow 0$ followed by a sequence of firings $0 \rightarrow I$, $(I \rightarrow O, O \rightarrow I)^*$ ending with the firing of $I \rightarrow C$ when the PACKET reaches aggregate y .

The relation between an old marking, m , and a new is incorporated in the arc-inscriptions, except for the two transition-predicates: $\Pi(\cdot, \sim x)\rho$, which says that $I \rightarrow 0$ fires only when the DESTINATION of the involved PACKET, ρ , is different from the IDENTITY of the INPUT and OUTPUT. And $\Pi(\cdot, y)\rho$, which says that $I \rightarrow C$ fires only when the DESTINATION of ρ equals the IDENTITY of INPUT and CONSUMER.

The initial marking, m_0 , is:

$$(\sum_{i \in N} (i, \lambda, \sigma_i^0); \sum_{i \in N} (i, \lambda, \lambda, \lambda); \sum_{x \in N} (x, \lambda, \lambda, \lambda); \sum_{x \in N} (x, \lambda))$$

where the sums refer to PROD, OUT, IN and CONS in that order and $\Pi(i, \cdot)\sigma_i^0$ for all $i \in N$, (σ_i^0 corresponds to $\text{SEQ}(i)$).

It is easy to verify that all reachable markings may be written on the following form:

$$m = (\sum_{i \in N} (i, \sigma_i^{\text{ps}}, \sigma_i^{\text{pn}}); \sum_{i \in N} (i, \sigma_i^{\text{os}}, \sigma_i^{\text{ob}}, \sigma_i^{\text{or}}); \sum_{x \in N} (x, \sigma_x^{\text{is}}, \sigma_x^{\text{ib}}, \sigma_x^{\text{ir}}); \sum_{x \in N} (x, \sigma_x^{\text{cr}}))$$

Since all place equations consist of predicates only (no lists of changeable variables), the only way to change a marking is by transition firings, i.e. by means of \rightarrow . Thus every reachable marking can be obtained as the result of a suitable sequence of transition firings starting from the initial marking.

When we consider a step in a firing sequence, we use the notation $m \rightarrow \hat{m}$, where the components of m are named as above and:

$$\hat{m} = (\sum_{i \in N} (i, \hat{\sigma}_i^{\text{ps}}, \hat{\sigma}_i^{\text{pn}}); \sum_{i \in N} (i, \hat{\sigma}_i^{\text{os}}, \hat{\sigma}_i^{\text{ob}}, \hat{\sigma}_i^{\text{or}}); \sum_{x \in N} (x, \hat{\sigma}_x^{\text{is}}, \hat{\sigma}_x^{\text{ib}}, \hat{\sigma}_x^{\text{ir}}); \sum_{x \in N} (x, \hat{\sigma}_x^{\text{cr}}))$$

Analysis of VERSION 3

We state that VERSION 3 is consistent and implements VERSION 2 with respect to the obvious mappings. We use the introduction of the "not sent" sequence to define a set of final markings and show that this is a subset of the dead markings. And we state that any transition firing gets the system "closer" to a final marking.

Theorem 4 VERSION 3, Fig. 7, is consistent.

Proof We must show that all reachable markings satisfy the predicates of all marked places:

$$m_0 \rightarrow^* m \Rightarrow$$

$$\text{PROD-predicates: } \sigma_i^{\text{ps}} \sigma_i^{\text{pn}} = \sigma_i^0,$$

$$\text{OUT-predicates: } 0 \leq |\sigma_i^{\text{ob}}| \leq 1, \quad \sigma_i^{\text{os}} \sigma_i^{\text{ob}} = \sigma_i^{\text{or}}, \quad \sigma_i^{\text{or}} \in [\Delta(\cdot, \sim i) \sigma_i^{\text{is}} || \sigma_i^{\text{ps}}],$$

IN-predicates : $0 \leq |\sigma_x^{ib}| \leq 1$, $\sigma_x^{is} \sigma_x^{ib} = \sigma_x^{ir}$, $\sigma_x^{ir} = \sigma_{x\theta 1}^{os}$,

CONS-predicates: $\sigma_x^{cr} = \Delta(\cdot, x) \sigma_x^{is}$,

where m_0 is the initial marking and m is on the form described above.

We prove the theorem by induction on single transition firings. Since all predicates are local, a transition firing affects at most the predicates corresponding to the involved objects and objects who receive PACKETS from the involved objects.

Basis

$m=m_0$ implies that the predicates are trivially true.

Firing

We assume that $m_0 \xrightarrow{*m} \hat{m}$. And we shall prove that \hat{m} fulfills the place-predicates under the assumption that m does.

$t=P \rightarrow 0$

Due to the local predicates this transition can at most affect the PROD- and OUT-predicate corresponding to the two tokens involved and the IN-predicate corresponding to the succeeding INput. From the inscriptions on the net we get the following, where j is the involved PRODucer and OUTput:

$$\begin{array}{ll} \text{ps)} & \hat{\sigma}_j^{ps} = \sigma_j^{ps} \rho; \quad \text{pn)} \quad \hat{\sigma}_j^{pn} = \sigma_j^{pn}; \\ \text{os)} & \hat{\sigma}_j^{os} = \sigma_j^{os} \quad \text{ob)} \quad \hat{\sigma}_j^{ob} = \rho; \quad \sigma_j^{ob} = \lambda; \\ \text{or)} & \hat{\sigma}_j^{or} = \sigma_j^{or} \rho; \end{array}$$

and otherwise \hat{m} equals m .

From os) we get that the IN-predicate is not affected.

In the following analysis, inscriptions on the equality signs refer to one of the five predicates above (ps, pn, os, ob or or) or to one of the predicates which we assume to be true for m (P, 0).

PROD-predicates

We shall prove that $\hat{\sigma}_j^{ps} \hat{\sigma}_j^{pn} = \sigma_j^0$. We get:

$$\hat{\sigma}_j^{ps} \hat{\sigma}_j^{pn} \stackrel{\text{ps}}{=} (\sigma_j^{ps} \rho) \hat{\sigma}_j^{pn} = \sigma_j^{ps} (\rho \hat{\sigma}_j^{pn}) \stackrel{\text{pn}}{=} \sigma_j^{ps} \sigma_j^{pn} \stackrel{P}{=} \sigma_j^0$$

OUT-predicates

We shall prove that $0 \leq |\hat{\sigma}_j^{ob}| \leq 1$, $\hat{\sigma}_j^{os} \hat{\sigma}_j^{ob} = \hat{\sigma}_j^{or}$ and $\hat{\sigma}_j^{or} \in [\Delta(\cdot, \sim j) \hat{\sigma}_j^{is} \parallel \hat{\sigma}_j^{ps}]$. We get:

$$\hat{\sigma}_j^{ob} \stackrel{\text{ob}}{=} \rho, \text{ i.e. } |\hat{\sigma}_j^{ob}| = 1$$

$$\hat{\sigma}_j^{os} \hat{\sigma}_j^{ob} \stackrel{\text{os, ob}}{=} \sigma_j^{os} \sigma_j^{ob} \rho \stackrel{0}{=} \sigma_j^{or} \rho \stackrel{\text{or}}{=} \hat{\sigma}_j^{or}$$

$$\hat{\sigma}_j^{or} \stackrel{\text{or}}{=} \sigma_j^{or} \rho \stackrel{0}{\in} [\Delta(\cdot, \sim j) \sigma_j^{is} \parallel (\sigma_j^{ps} \rho)] \stackrel{\text{ps}}{=} [\Delta(\cdot, \sim j) \hat{\sigma}_j^{is} \parallel \hat{\sigma}_j^{ps}]$$

Analogously, we may show that the firing of the three other transitions also preserves the predicates. \square

Theorem 5 VERSION 3 implements VERSION 2 with respect to (id,proj), where id and proj are constructed as in Theorem 3.

Proof Trivial. \square

Final markings

Informally a marking is final iff all packets initially contained in the "not sent" sequences of the PRODucers have been correctly delivered to the CONSUMers. This is formally defined below.

Definition The set of final markings, FM, is:

$$\{ (\sum_{i \in N} (i, \sigma_i^{ps}, \sigma_i^{pn}); \sum_{i \in N} (i, \sigma_i^{os}, \sigma_i^{ob}, \sigma_i^{or}); \sum_{x \in N} (x, \sigma_x^{is}, \sigma_x^{ib}, \sigma_x^{ir}); \sum_{x \in N} (x, \sigma_x^{cr})) \mid \forall i, x \in N: \Delta(i, \cdot) \sigma_x^{cr} = \Delta(\cdot, x) \sigma_i^0 \}$$

Theorem 6 If a final marking is reached then the system is terminated. More precisely

$$\forall m \in R: m \in FM \Rightarrow \neg (\exists m' \in R: m \rightarrow m' \vee m \sim m').$$

Proof Intuitively we need the "global" specification of our system, represented by VERSION 1, in our proof. It is a simple task to use VERSION 1 via our previous theorems:

Assume

$$m \in FM$$

(definition)

$$\Downarrow \Delta(i, \cdot) \sigma_x^{cr} = \Delta(\cdot, x) \sigma_i^0 \text{ for all } i, x \in N$$

(Theorem 5, 3 and 1 and CONS-predicates of VERSION 1)

$$\Downarrow \Delta(\cdot, x) \sigma_i^0 \sim \Delta(i, x) \text{route}(i, x) = \Delta(\cdot, x) \sigma_i^{ps} \text{ for all } i, x \in N$$

(Theorem 4 and PROD-predicates of VERSION 3)

$$\Downarrow \text{route}(i, x) = \lambda \wedge \sigma_i^{pn} = \lambda \text{ for all } i, x \in N$$

(definition)

$$\Downarrow \sigma_i^{ob} = \lambda \wedge \sigma_x^{ib} = \lambda \wedge \sigma_i^{pn} = \lambda \text{ for all } i, x \in N$$

(net-inscriptions)

$$\Downarrow \text{no transition is enabled}$$

(definition)

$$\Downarrow m \text{ is dead.}$$

And since there are no changeable variables, " \sim " cannot be applied. \square

Deadlock

The specified transport system, with its unrestricted use of buffers of limited size, may deadlock before a final state is reached.

Example Deadlock in a not final marking. The net is characterized by:

$$N=1..3 \text{ and } \sigma_1^0 = (1,3) (1,2), \sigma_2^0 = (2,1) (2,3) \text{ and } \sigma_3^0 = (3,2) (3,1)$$

With $T_1 = \{P \rightarrow 0 (j=1,2,3)\}$, $T_2 = \{0 \rightarrow I (i=1,2,3)\}$ and $T_3 = \{P \rightarrow 0 (j=1,2,3)\}$ we have that

$$m_0 \xrightarrow{T_1} m_1 \xrightarrow{T_2} m_2 \xrightarrow{T_3} m_3$$

where m_3 is dead and $m_3 \notin FM$. \square

If we want to exclude the possibility of deadlock in a non-final state, we can do this by preventing undelivered packets from filling the buffers. This may be done in a number of different ways, but we will not discuss the subject in this paper.

Progression

Finally we consider progression. We define a function, Λ , from markings

$$m = (\Sigma(i, \sigma_i^{ps}, \sigma_i^{pn}); \Sigma(i, \sigma_i^{os}, \sigma_i^{ob}, \sigma_i^{or}); \Sigma(x, \sigma_x^{is}, \sigma_x^{ib}, \sigma_x^{ir}); \Sigma(x, \sigma_x^{cr}))$$

into non-negative integers:

$$\Lambda(m) = \sum_{i \in N} (\Lambda_P(i, \sigma_i^{pn}) + \Lambda_0(i, \sigma_i^{ob}) + \Lambda_I(i, \sigma_i^{ib}))$$

where

$$\begin{aligned} \Lambda_P(j, (i_1, x_1, m_1) \dots (i_k, x_k, m_k)) &= \sum_{h=1}^k (1 + 2 \times (x_h \Theta j)) \\ \Lambda_0(j, (i_1, x_1, m_1) \dots (i_k, x_k, m_k)) &= \sum_{h=1}^k 2 \times (x_h \Theta j) \\ \Lambda_I(j, (i_1, x_1, m_1) \dots (i_k, x_k, m_k)) &= \sum_{h=1}^k (1 + 2 \times [(x_k \Theta j) \bmod L]) \end{aligned}$$

As we shall see below, Λ measures the number of single transition-firings necessary to go from m to a final marking. (The system may however deadlock before a final marking is reached.)

First we state that each time the marking changes Λ is decreased. We do this by showing that a single transition-firing decreases Λ by one:

Theorem 7

$$\forall m \in R \quad \forall t \in T: m \xrightarrow{t} \hat{m} \Rightarrow \Lambda(m) = \Lambda(\hat{m}) + 1$$

Proof

$t = P \rightarrow 0$, involved PRODUCER j . From the inscriptions on the net we get (cf. Theorem 4):

$$\Lambda_P(j, \sigma_j^{pn}) = \Lambda_P(j, \rho \hat{\sigma}_j^{pn}), \text{ where } \rho = (\cdot, y, \cdot)$$

$$= \Lambda_P(j, \hat{\sigma}_j^{pn}) + (1 + 2 \times (y \Theta j))$$

$$\Lambda_0(j, \sigma_j^{ob}) = \Lambda_0(j, \lambda) = 0$$

$$\Lambda_0(j, \hat{\sigma}_j^{ob}) = \Lambda_0(j, \rho) = 2 \times (y \Theta j).$$

For $k \in N \setminus \{j\}$ Λ_P and Λ_0 are unchanged, and for $i \in N$ Λ_I is unchanged. Thus we get:

$$\begin{aligned} \Lambda(m) &= \Lambda(\hat{m}) + (1 + 2 \times (y \Theta j)) - 2 \times (y \Theta j) \\ &= \Lambda(\hat{m}) + 1 \end{aligned}$$

$t = 0 \rightarrow I$, involved OUTPut i . We get:

$$\Lambda_0(i, \sigma_i^{ob}) = \Lambda_0(i, \rho \hat{\sigma}_i^{ob}), \text{ where } \rho = (\cdot, v, \cdot)$$

$$= \Lambda_0(i, \sigma_i^{ob}) + 2 \times (v\theta i)$$

$$\Lambda_I(i\oplus 1, \sigma_{i\oplus 1}^{ib}) = \Lambda_I(i\oplus 1, \lambda) = 0$$

$$\Lambda_I(i\oplus 1, \sigma_{i\oplus 1}^{ib}) = \Lambda_I(i\oplus 1, \rho) \\ 1 + 2 \times [(v\theta(i\oplus 1)) \bmod L]$$

otherwise Λ_0 and Λ_I are unchanged and Λ_P is unchanged. Thus we get:

$$\Lambda(m) = \Lambda(\hat{m}) + 2 \times (v\theta i) - (1 + 2 \times [(v\theta(i\oplus 1)) \bmod L]) \\ = \Lambda(\hat{m}) + 1$$

The cases $t=I \rightarrow 0$ and $t=I \rightarrow C$ may be treated analogously. \square

Theorem 8

$$\forall m \in R: m \in FM \Leftrightarrow \Lambda(m) = 0$$

Proof

" \Rightarrow " Assume

$$m \in FM$$

(cf. proof of
Theorem 6) \Downarrow

$$\sigma_i^{pn} = \lambda \wedge \sigma_i^{ob} = \lambda \wedge \sigma_i^{ib} = \lambda \quad \text{for all } i \in N$$

(definition) \Downarrow

$$\Lambda(m) = 0$$

" \Leftarrow " Assume

$$\Lambda(m) = 0$$

(definition) \Downarrow

$$\sigma_i^{pn} = \lambda \wedge \sigma_i^{ob} = \lambda \wedge \sigma_i^{ib} = \lambda \quad \text{for all } i \in N$$

(Theorem 5, 3 and
1 and CONS-
predicates of
VERSION 1) \Downarrow

$$\Delta(i, \cdot) \sigma_x^{cr} \wedge \lambda = \Delta(\cdot, x) \sigma_i^{ps} \wedge \sigma_i^{pn} = \lambda \quad \text{for all } i, x \in N$$

(Theorem 4 and
PROD-predicates
of VERSION 3) \Downarrow

$$\Delta(i, \cdot) \sigma_x^{cr} = \Delta(\cdot, x) \sigma_i^0 \quad \text{for all } i, x \in N$$

(definition) \Downarrow

$$m \in FM \quad \square$$

To sum up: VERSION 3 is consistent and implements the safe states specified by VERSION 1 (Theorems 4, 5, 3 and 1). Furthermore VERSION 3 meets the progress specifications describes by Theorems 7 and 8, which may be paraphrased as follows: the length of any firing sequence of VERSION 3 is less than or equal to $\Lambda(m_0)$; and if it equals $\Lambda(m_0)$ the last element is a final marking.

4. Conclusion

We have defined a simple ring-structured transport system by a step-wise specification of its properties. Properties of all reachable states are expressed directly in the Epsilon language by predicates. Properties concerning progress are not formulated directly in Epsilon, but stated separately.

The first formal description of the safe states uses predicates which directly model the informal specification. It is important that the specification language enables one to do so since this simplifies the inherently unformalizable task of convincing oneself that the formal specification is adequate.

The complexity of the proofs is reduced by the use of local predicates; and with more complicated systems this "factorization" becomes increasingly important. However, the proof method works for non-local predicates as well, which gives one the possibility of allowing a few "global" predicates, e.g. if these do not have any natural local counterparts.

In an earlier paper [11] we defined the semantics of Epsilon by a model containing a condition-event net (to model the control flow) and a separate set of variables (to model all data-attributes in a system). That approach was inspired by Keller [12] and Mazurkiewicz [13]. Our present model, which uses the high-level Petri nets of Genrich and Lautenbach [3] and Jensen [8],[9], integrates the data state in the token-colours of a net. It is simpler than the one presented in [11], and so are proofs based on the model.

At present we are considering the inclusion of a temporal logic in Epsilon to be able to formulate properties concerning progress directly in the Epsilon description.

Acknowledgements

The work on this paper has benefited from many discussions on earlier versions with Kurt Jensen, Ole Lehrmann Madsen, Mogens Nielsen, Karsten Bank Petersen and P.S. Thiagarajan.

References

- [1] Brauer, W. (ed.): Net theory and applications. Proceedings of the Advanced Course on General Net Theory of Processes and Systems, Hamburg 1979. LNCS 84, Springer-Verlag, 1980.
- [2] Dahl, O.-J., Myhrhaug, B. and Nygaard, K.: Common Base Language. Norwegian Computing Center, Oslo, 1970.
- [3] Genrich, H.J. and Lautenbach, K.: System modelling with high-level Petri nets. Theoretical Computer Science 13 (1981), 109-136.
- [4] Hoare, C.A.R.: Communicating sequential processes. Comm. ACM 21, 8 (August 1978), 666-677.
- [5] Holbæk-Hanssen, E., Håndlykken, P. and Nygaard, K.: System description and the Delta language. Norwegian Computing Center, Oslo 1975.
- [6] Howard, J.H.: Proving Monitors. Comm. ACM 19, 5 (May 1976), 273-279.

- [7] Ichbiah, J.D. et al.: Reference manual for the ADA programming language. Proposed standard document. United States Department of Defense, July 1980.
- [8] Jensen, K.: Coloured Petri nets and the invariant-method. Theoretical Computer Science 14 (1981), 317-336.
- [9] Jensen, K.: High-level Petri nets. To appear in the proceedings of the Third European Workshop on Applications and Theory of Petri nets. Informatik-Fachberichte, Springer Verlag.
- [10] Jensen, K. and Kyng, M.: Epsilon - a system description language. DAIMI PB-150, Computer Science Department, Aarhus University, September 1982.
- [11] Jensen, K., Kyng, M. and Madsen, O.L.: A Petri net definition of a system description language. Semantics of Concurrent Computation, Evian 1979, G. Kahn (ed.), LNCS 70, Springer-Verlag 1979, 348-368.
- [12] Keller, R.M.: Formal verification of parallel programs. Comm. ACM 19, 7 (July 1976), 371-384.
- [13] Mazurkiewicz, A.: Concurrent programschemes and their interpretations. DAIMI PB-78, Computer Science Department, Aarhus University, July 1977.
- [14] Owicki, S.: Specification and verification of a network mail system. Program Construction, F.L. Bauer and M. Broy (eds.), LNCS 69, Springer-Verlag 1979, 198-234.
- [15] Wang, A.: Generalized types in high-level programming languages. Research Reports in Informatics, No. 1, University of Oslo, 1975.