SPECIFICATION and VERIFICATION of NETWORKS in a PETRI NET based LANGUAGE

Morten Kyng

DAIMI PB-153
October 1982
SPECIFICATION and VERIFICATION of NETWORKS in a PETRI NET based LANGUAGE

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 in-
ternal connections, are depicted in Fig. 1:

![Diagram showing the connections between aggregates]

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 attrib-
tutes 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 ob-
ject 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 vary-
ing 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 transfor-
mation. 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
5        DEST : N
6        MESSAGE: M record)
7    type PACKETSEQ: seq PACKET
8
9 PROD(ID:N):
10 (object
11    SENT: PACKETSEQ init EMPTY
12    (* SENT \[\pi(ID,\cdot)\]SENT *)
13    object)
14
15 OUT(ID:N), IN(ID:N):
16 (object
17    BUF: PACKETSEQ init EMPTY
18    (* BUF *)
19    object)
20
21 CONS(ID:N):
22 (object
23    REC: PACKETSEQ init EMPTY
24    (* REC \[\forall i \in N [\Delta(i,\cdot)REC \Delta(i,ID)route(i,ID)
25       = \Delta(\cdot,ID)PROD(i).SENT ] *)
26    object)
27
28 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, π(ID, ·)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:

\[
\begin{align*}
\pi : \mathcal{P}(N) \times \mathcal{P}(N) & \to \text{PACKETSEQ} \to \{\text{true, false}\} \\
\Delta : \mathcal{P}(N) \times \mathcal{P}(N) & \to \text{PACKETSEQ} \to \text{PACKETSEQ}
\end{align*}
\]

The function π(P, C) is used to test that all packets in a sequence has a SOURCE in P and a DESTination in C; Δ(P, C) picks out the maximal subsequence with this property. Formally we define

- \( \pi(P, C) \) is true, λ is the empty sequence
- \( \pi(P, C) \sigma \rho \) is true iff \( \pi(P, C) \sigma \) and \( \rho.\text{SOURCE} \in P \) and \( \rho.\text{DEST} \in C \), σ ∈ PACKETSEQ, ρ ∈ PACKET.

\( \Delta(P, C) \sigma \) is the maximal subsequence, \( \sigma_m \), of \( \sigma \) such that \( \pi(P, C) \sigma_m \) is true. By convention \( \pi(P, \cdot) = \pi(P, N) \); \( \pi(P, c) = \pi(P, \{c\}) \); and \( \pi(P, \cdot; c) = \pi(P, N(\{c\})) \). 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 π(ID, ·)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 × N → 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 k=1):

\[
\text{route}(x@k, x) = \begin{align*}
\text{IN}(x).\text{BUF} & \\
\text{OUT}(x@1).\text{BUF} & \leftarrow \text{IN}(x@1).\text{BUF} \\
& \vdots \\
\text{OUT}(x@(k-1)).\text{BUF} & \leftarrow \text{IN}(x@(k-1)).\text{BUF} \\
\text{OUT}(x@k).\text{BUF}
\end{align*}
\]

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 [70] 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 

\[(i, \sigma_i^{ps}) \in N \times PACKETSEQ\]

where the first component represents the PRODucer IDentifier and the second SENT. To stress the relationship we may write ID\times SENT instead of N\times PACKETSEQ. ID\times 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.

\[(i, \sigma_i^{ps}) \in ID\times SENT \quad \{\sigma_i^{ps} \mid \Pi(i, \cdot) \sigma_i^{ps}\}\]

\[(i, \sigma_i^{ob}) \in ID\times BUF \quad \{\sigma_i^{ob}\}\]

\[(x, \sigma_x^{ib}) \in ID\times BUF \quad \{\sigma_x^{ib}\}\]

\[(x, \sigma_x^{cr}) \in ID\times REC \quad \{\sigma_x^{cr} \mid \forall i \in N[\Delta(i, \cdot) \sigma_x^{cr} \cap \Delta(i,x) \text{ route } (i,x)] = \Delta(\cdot, x) \sigma_i^{ps}\}\]

The initial marking \(m_0 = (\Sigma (i, \lambda); \Sigma (i, \lambda); \Sigma (x, \lambda); \Sigma (x, \lambda))\)

\[\begin{align*}
\text{i} & \in N \\
\text{i} & \in N \\
x & \in N \\
x & \in N
\end{align*}\]

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

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

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

\[
m_{i-1} \bowtie m_i \Rightarrow \forall (3m!:\ m_{i-1} \rightarrow m!)
\]

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, "\( \bowtie \)".

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 "\( \bowtie \)" and the facts that the initial marking is consistent and that the net contains no transitions. \( \Box \)

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
9     PROD(ID:N):
10           (object
11              SENT: PACKETSEQ init EMPTY
12           (* SENT | $\pi(ID, \cdot)SENT$)
13          object)
14
15     OUT(ID:N):
16           (object
17              SENT,BUF,REC: PACKETSEQ init EMPTY
18           (* SENT,BUF,REC | SENT$\sim$BUF = REC,
19               REC $\in$ \{ $\Delta(\cdot, \sim ID)IN(ID).SENT \|$ PROD(ID).SENT \}
20          object)
21
22     IN(ID:N):
23           (object
24              SENT,BUF,REC: PACKETSEQ init EMPTY
25           (* SENT,BUF,REC | SENT$\sim$BUF = REC, REC = OUT(ID$\emptyset$).SENT)
26          object)
27
28     CONS(ID:N):
29           (object
30              REC: PACKETSEQ init EMPTY
31           (* REC | REC = $\Delta(\cdot, ID)IN(ID).SENT$)
32          object)
33
34     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 set of OUT and IN have, compared to VERSION 1, been extended with components representing the sequence variables SENT and RECEIVED. The four place equations are direct translations of the equations of the corresponding statements.

\[(i, \sigma^{ps}_i) \in ID \times SENT \quad PROD \quad \{ \sigma^{ps}_i | \pi(i, \cdot) \sigma^{ps}_i \}\]

\[(i, \sigma^{os}_i, \sigma^{ob}_i, \sigma^{or}_i) \in ID \times SENT \times BUF \times REC \quad OUT \quad \{ \sigma^{os}_i, \sigma^{ob}_i, \sigma^{or}_i | \sigma^{os}_i \sigma^{ob}_i = \sigma^{or}_i, \sigma^{or}_i \in [\Delta(\cdot, \cdot)i \sigma^{is}_i \| \sigma^{ps}_i] \}

\[(x, \sigma^{is}_x, \sigma^{ib}_x, \sigma^{ir}_x) \in ID \times SENT \times BUF \times REC \quad IN \quad \{ \sigma^{is}_x, \sigma^{ib}_x, \sigma^{ir}_x | \sigma^{is}_x \sigma^{ib}_x = \sigma^{ir}_x, \sigma^{ir}_x = \sigma^{os}_x \sigma^{is}_x \}

\[(x, \sigma^{cr}_x) \in ID \times REC \quad CONS \quad \{ \sigma^{cr}_x | \sigma^{cr}_x = \Delta(\cdot, x) \sigma^{is}_x \}

Initial marking \(m_0 = \left( \Sigma (i, \lambda); \Sigma (i, \lambda, \lambda, \lambda); \Sigma (x, \lambda, \lambda, \lambda); \Sigma (x, \lambda) \right) \)

\[\text{Figure 5}\]

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 "\(\cong\)" and the facts that the initial marking is consistent and that the net contains no transitions. \(\Box\)

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

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

\(h_c\) 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** VERSION 2 implements VERSION 1 with respect to \((id, proj)\)

where:

- **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 \(\text{proj}(m)\) fulfills:

\[
\forall i, x \in N[\Delta(i, \cdot) \circ_X \cap \Delta(i, x) \text{route}(i, x) = \Delta(\cdot, x) \circ_1 ps]
\]

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

**Lemma** Any reachable marking, \(m\), of VERSION 2 satisfies the predicate:

\[
\Delta(i, x) \circ_1 ps = \Delta(i, x) \circ_1 or
\]

**Proof** From the OUT predicate we get

\[
\circ_1 or \in [\Delta(\cdot, \sim i) \circ_1 is \| \circ_1 ps]
\]

so it is sufficient to prove that \(\Delta(i, \sim i) \circ_1 is = \lambda\). We do this by contradiction.

Assume that \(\rho \in \text{PACKET}\) is contained in \(\Delta(i, \sim i) \circ_1 is\), i.e. \(\rho = (i, i \theta k, a), k \in 1..(L-1)\).

From the PROD predicates we get that \(\rho\) is not contained in \(\circ_1 ps \cdots \circ_1 i \theta k\). From the IN and OUT predicates we get that \(\rho\) is contained in \(\circ_1 is \cdots \circ_1 i \theta k\) and finally that \(\rho\) is contained in \(\Delta(\cdot, \sim (i \theta k)) \circ_1 is\). But this contradicts our initial assumption that \(\rho = (i, i \theta k, a)\).

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

\[
\Delta(\cdot, x) \circ_1 ps \times \theta k
\]

\[
= \Delta(x \theta k, x) \circ_1 ps \times \theta k
\]

\[
= \Delta(x \theta k, x) \circ_1 or \times \theta k
\]

\[
= \Delta(x \theta k, x) \left[ \circ_1 os \circ_1 ob \times \theta k \times \theta k \right]
\]

\[
= \Delta(x \theta k, x) \left[ \circ_1 ir \times \theta (k-1) \circ_1 ob \times \theta k \times \theta k \right]
\]

\[
= \Delta(x \theta k, x) \left[ \circ_1 is \times \theta (k-1) \circ_1 ib \times \theta (k-1) \circ_1 ob \times \theta k \times \theta k \right]
\]

\[
= \Delta(x \theta k, x) \left[ \circ_1 or \times \theta (k-1) \circ_1 ib \times \theta (k-1) \circ_1 ob \times \theta k \times \theta k \right]
\]

\[
\vdots
\]
\[
\Delta(x\theta k, x) \left[ c_{x}^{ib} c_{x}^{ob} \ldots c_{x}^{ob} \right] \\
\Delta(x\theta k, \cdot) \Delta(\cdot, x) \circ_{x}^{is} \Delta(x\theta k, x) \left[ c_{x}^{ib} c_{x}^{ob} \ldots c_{x}^{ob} \right] \\
\Delta(x\theta k, \cdot) \circ_{x}^{cr} \Delta(x\theta k, x) \left[ c_{x}^{ib} \ldots c_{x}^{ob} \right] \text{(CONS)} \\
\Delta(x\theta k, \cdot) \circ_{x}^{cr} \Delta(x\theta k, x) \text{route}(x\theta k, x)
\]

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

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 SEQ(N) of PACKETSEQences contains the PACKETS to be sent by the PRODucers. It is for all \text{i} \in \text{N} used to initialize NOTS(i) in such a way that all PACKETS have SOURCE equal to \text{i}.
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 PRODucer 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 PRODucer begins its actions by executing the equational-statement. During its execution none of the variables of the PRODucer 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 PRODucer, (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 PRODucer 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 ∀ i ∈ N[Π(i,·)SEQ(i)]
9
10      PROD(ID:N):
11  (object
12      SENT,NOTS: PACKETSEQ init EMPTY,SEQ(ID)
13      PUT: (procedure SENT:=•NOTS.HEAD
14          NOTS.DELETE procedure)
15
16 (* SENT•NOTS = SEQ(ID) *)
17
18      control
19      → [* match OUT(ID).GET(NOTS.HEAD)
20          when NOTS ≠ EMPTY do PUT *]
21  object)
22
23      OUT(ID:N):
24  (object
25      SENT,BUF,REC: PACKETSEQ init EMPTY
26      GET(PCK:PACKET): (procedure REC,BUF:=•PCK procedure)
27      PUT: (procedure SENT:=•BUF.HEAD
28          BUF.DELETE procedure)
29
30 (* 0≤|BUF|≤1, SENT•BUF = REC,
31     REC ∈ [Δ(·,~ID)IN(ID).SENT||PROD(ID).SENT] *)
32
33      control
34      → [* match PROD(ID).PUT when BUF=EMPTY do GET *]
35
36      → [* match IN(ID).PUT when BUF=EMPTY do GET *]
37
38      → [* match IN(ID@1).GET(BUF.HEAD)
39          when BUF ≠ EMPTY do PUT *]
40  object)
IN(ID:N):

(object

    SENT,BUF,REC: PACKETSEQ init EMPTY

    GET(PCK: PACKET):

        (procedure REC,BUF:=\^PCK procedure)

        PUT: (procedure SENT:=\^BUF.HEAD

            BUF.DELETE procedure)

    (* 0\leq|BUF|\leq1, SENT*BUF = REC, REC = OUT(ID\varnothing1).SENT *)

    control

    \rightarrow [* match OUT(ID\varnothing1).PUT when BUF=EMPTY do GET *]

    \rightarrow [* match CONS(ID).GET(BUF.HEAD)

        when BUF*EMPTY, BUF.HEAD.DEST=ID do PUT *]

    \rightarrow [* match OUT(ID).GET(BUF.HEAD)

        when BUF*EMPTY, BUF.HEAD.DEST*ID do PUT *]

    object)

CONS(ID:N):

(object

    REC: PACKETSEQ init EMPTY

    GET(PCK: PACKET):

        (procedure REC:=\^PCK procedure)

    (* REC = \Delta(*, ID)IN(ID).SENT *)

    control

    \rightarrow [* match IN(ID).PUT do GET *]

    object)

system)

Figure 7

sitions, one for each matching pair of event-statements in Fig. 7. Transition P\to0 corresponds to the match (16-17)-(28), i.e. transfer of a PACKET from PRODucer to OUTput. I\to0 corresponds to the match (29)-(45-46), 0\toI to (30-31)-(42), and I\toC 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\to0 fires, j denotes the IDentity of the involved PRODucer and OUTput. When 0\toI fires (transmis-

sion of a PACKET from one aggregate to the next) i denotes the IDentity of the OUTput and i\varnothing1 that of the INput, etc. As before \epsilon\in PACKETSEQ,
Figure 8
A PACKET is sent from the PRODucer in aggregate j to the CONSumer in aggregate y by a firing of P→0 followed by a sequence of firings 0→I, (I→0, 0→I)* ending with the firing of I→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: \( \Pi(\cdot, \cdot\neg x) \)\( ^{p} \), which says that I→0 fires only when the DESTination of the involved PACKET, \( \rho \), is different from the IDentity of the INput and OUTput. And \( \Pi(\cdot, \cdot) \)\( ^{0} \), which says that I→C fires only when the DESTination of \( \rho \) equals the IDentity of INput and CONSumer.

The initial marking, \( m_{0} \), is:

\[
\left( \sum_{i \in N} (i_{0}^{i}, \lambda_{i}^{0}), \sum_{i \in N} (i_{0}^{i}, \lambda_{i}, \lambda_{i}), \sum_{x \in N} (x_{0}^{i}, \lambda_{x}, \lambda_{x}), \sum_{x \in N} (x_{0}^{i}, \lambda_{x}) \right)
\]

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

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

\[
m = \left( \sum_{i \in N} (i_{0}^{i}, \lambda_{i}^{ps}, \lambda_{i}^{pn}), \sum_{i \in N} (i_{0}^{i}, \lambda_{i}^{os}, \lambda_{i}^{ob}, \lambda_{i}^{or}), \sum_{x \in N} (x_{0}^{i}, \lambda_{x}^{is}, \lambda_{x}^{ib}, \lambda_{x}^{ir}), \sum_{x \in N} (x_{0}^{i}, \lambda_{x}^{cr}) \right)
\]

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:

\[
\hat{m} = \left( \sum_{i \in N} (i_{0}^{i}, \lambda_{i}^{ps}, \lambda_{i}^{pn}), \sum_{i \in N} (i_{0}^{i}, \lambda_{i}^{os}, \lambda_{i}^{ob}, \lambda_{i}^{or}), \sum_{x \in N} (x_{0}^{i}, \lambda_{x}^{is}, \lambda_{x}^{ib}, \lambda_{x}^{ir}), \sum_{x \in N} (x_{0}^{i}, \lambda_{x}^{cr}) \right)
\]

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:

\[
m_{0} \rightarrow* m \Rightarrow
\]

PROD-predicates: \( \lambda_{i}^{ps} = \lambda_{i}^{0} \),

OUT-predicates: \( 0 \leq \lambda_{i}^{ob} \leq 1, \lambda_{i}^{os} \lambda_{i}^{ob} = \lambda_{i}^{or}, \lambda_{i}^{or} \in \Delta(\cdot, \cdot) \lambda_{i}^{is} \leq \lambda_{i}^{ps} \).
IN-predicates: \( 0 \leq \sigma^ib_x \leq 1, \sigma^ib_x = \sigma^ib_x^r, \sigma^ir_x = \sigma^os_x^r \)

CONS-predicates: \( \sigma^cr_x = \Delta(\cdot, x)\sigma^is_x \)

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 \( m_0 \rightarrow^t m \). 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:

\[
\begin{align*}
\text{ps): } & \sigma^ps_j \sigma^pn_j = \sigma^ps_j \rho; \\
& \sigma^pn_j = \sigma^pn_j; \\
\text{os): } & \sigma^os_j = \sigma^os_j \\
& \sigma^ob_j = \rho; \sigma^ob_j = \lambda; \\
\text{or): } & \sigma^or_j = \sigma^or_j \rho;
\end{align*}
\]

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 \( \sigma^ps_j \sigma^pn_j = \sigma^0_j \). We get:

\[
\begin{align*}
\sigma^ps_j \sigma^pn_j &= (\sigma^ps_j \rho) \sigma^pn_j = \sigma^ps_j (\rho \sigma^pn_j) \quad \text{for } j \in \sigma^ps_j \sigma^pn_j = \sigma^0_j
\end{align*}
\]

**OUT-predicates**

We shall prove that \( 0 \leq \sigma^ob_j \leq 1, \sigma^os_j \sigma^ob_j = \sigma^or_j \) and \( \sigma^or_j \in [\Delta(\cdot, \sim) \sigma^is_j \| \sigma^ps_j] \).

We get:

\[
\begin{align*}
\sigma^ob_j &= \rho, \text{ i.e. } |\sigma^ob_j| = 1 \\
\sigma^os_j \sigma^ob_j &= \sigma^os_j \sigma^ob_j \sigma^or_j = \sigma^or_j \sigma^or_j = \sigma^or_j \\
\sigma^or_j &= \sigma^or_j \sigma^or_j \sigma^os_j \sim [\Delta(\cdot, \sim) \sigma^is_j \| (\sigma^ps_j \rho)] \sigma^ps_j [\Delta(\cdot, \sim) \sigma^is_j \| \sigma^ps_j]
\end{align*}
\]

Analogously, we may show that the firing of the three other transitions also preserves the predicates.
Theorem 5: VERSION 3 implements VERSION 2 with respect to \((id, \text{proj})\), where \(id\) and \(\text{proj}\) 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:
\[
\{(i, o^{ps}_i, o^{pn}_i); \ (i, o^{os}_i, o^{ob}_i, o^{cr}_i); \ (x, o^{is}_x, o^{ib}_x, o^{ir}_x); \ x \in N \} \\
\Sigma (x, o^{cr}_x) \ | \ \forall i, x \in N: \Delta(i, \cdot) o^{cr}_x = \Delta(\cdot, x) o^0_i, \ x \in N
\]

Theorem 6: If a final marking is reached then the system is terminated. More precisely
\[
\forall m \in R: m \in FM \Rightarrow \neg(\exists m' \in R: \text{m} \rightarrow m' \land m' \in FM)
\]

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)
\[
\Delta(i, \cdot) o^{cr}_x = \Delta(\cdot, x) o^0_i \quad \forall i, x \in N
\]

(Theorem 5, 3 and 1 and CONS-predicates of VERSION 1)
\[
\Delta(\cdot, x) o^0_i \wedge \Delta(i, x) \text{route}(i, x) = \Delta(\cdot, x) o^{ps}_i \quad \forall i, x \in N
\]

(Theorem 4 and PROD-predicates of VERSION 3)
\[
\text{route}(i, x) = \lambda \wedge o^{ps}_i = \lambda \quad \forall i, x \in N
\]

(definition)
\[
o^{ob}_i = \lambda \wedge o^{ib}_x = \lambda \wedge o^{pn}_i = \lambda \quad \forall i, x \in N
\]

(net-inscriptions)
\[
\text{no transition is enabled}
\]

(definition)
\[
m \text{ is dead.}
\]

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:

\(N = 1..3\) and \(o^{0}_i = (1, 3)(1, 2), o^{0}_2 = (2, 1)(2, 3)\) and \(o^{0}_3 = (3, 2)(3, 1)\)

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 \( m_3 \notin \text{FM} \).

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, \( \Lambda \), 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

\[ \Lambda_p(j, (i_1, x_1, m_1), (i_2, x_2, m_2), \ldots (i_k, x_k, m_k)) = \sum_{h=1}^{k} (1 + 2x(x_h \Theta j)) \]

\[ \Lambda_0(j, (i_1, x_1, m_1), \ldots (i_k, x_k, m_k)) = \sum_{h=1}^{k} 2x(x_h \Theta j) \]

\[ \Lambda_I(j, (i_1, x_1, m_1), \ldots (i_k, x_k, m_k)) = \sum_{h=1}^{k} (1 + 2x[(x_h \Theta j) \text{mod} L]) \]

As we shall see below, \( \Lambda \) 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 \( \Lambda \) is decreased. We do this by showing that a single transition-firing decreases \( \Lambda \) by one:

**Theorem 7**

\[ \forall m \in \mathcal{R}, \forall t \in \mathcal{T}: m \xrightarrow{t} \hat{m} \Rightarrow \Lambda(m) = \Lambda(\hat{m}) + 1 \]

**Proof**

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

\[ \Lambda_p(j, \sigma_j^{PN}) = \Lambda_p(j, \sigma^{PN}), \text{ where } \rho = (\ldots, y, \ldots) \]

\[ = \Lambda_p(j, \sigma_j^{PN}) + (1 + 2x(y \Theta j)) \]

\[ \Lambda_0(j, \sigma_j^{OB}) = \Lambda_0(j, \lambda) = 0 \]

\[ \Lambda_0(j, \sigma_j^{OB}) = \Lambda_0(j, \rho) = 2x(y \Theta j). \]

For \( k \in N \setminus \{ j \} \) \( \Lambda_p \) and \( \Lambda_0 \) are unchanged, and for \( i \in N \) \( \Lambda_I \) is unchanged. Thus we get:

\[ \Lambda(m) = \Lambda(\hat{m}) + (1 + 2x(y \Theta j)) - 2x(y \Theta j) \]

\[ = \Lambda(\hat{m}) + 1 \]

\( t = 0 \to I \), involved OUTput \( i \). We get:

\[ \Lambda_0(i, \sigma_i^{OB}) = \Lambda_0(i, \rho \sigma_i^{OB}), \text{ where } \rho = (\ldots, v, \ldots) \]
\[ \Lambda_0(i, \sigma_i^{\text{ob}}) + 2(x(v \Theta i)) \]

\[ \Lambda_I(i \oplus 1, \sigma_i^{\text{ib}}) = \Lambda_I(i \oplus 1, \lambda) = 0 \]

\[ \Lambda_I(i \oplus 1, \sigma_i^{\text{b}}) = \Lambda_I(i \oplus 1, \rho) \]

\[ 1 + 2x[(v \Theta (i \oplus 1)) \mod L] \]

otherwise \( \Lambda_0 \) and \( \Lambda_I \) are unchanged and \( \Lambda_p \) is unchanged. Thus we get:

\[ \Lambda(m) = \Lambda(\hat{m}) + 2(x(v \Theta i)) - (1 + 2x[(v \Theta (i \oplus 1)) \mod L]) = \Lambda(\hat{m}) + 1 \]

The cases \( t = I \rightarrow 0 \) and \( t = I \rightarrow C \) may be treated analogously. \( \Box \)

**Theorem 8**

\[ \forall m \in R: m \in \text{FM} \Leftrightarrow \Lambda(m) = 0 \]

**Proof**

\[ \Rightarrow \] Assume

\[ m \in \text{FM} \]

(cf. proof of \( \Downarrow \)

Theorem 6)

\[ \sigma_i^{\text{pn}} = \lambda \land \sigma_i^{\text{ob}} = \lambda \land \sigma_i^{\text{ib}} = \lambda \quad \text{for all } i \in \mathbb{N} \]

(definition) \( \Downarrow \)

\[ \Lambda(m) = 0 \]

\[ \Leftarrow \] Assume

\[ \Lambda(m) = 0 \]

(definition) \( \Downarrow \)

\[ \sigma_i^{\text{pn}} = \lambda \land \sigma_i^{\text{ob}} = \lambda \land \sigma_i^{\text{ib}} = \lambda \quad \text{for all } i \in \mathbb{N} \]

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

VERSION 1)

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

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

VERSION 3)

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

(definition) \( \Downarrow \)

\[ m \in \text{FM} \quad \Box \]

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 described 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


