An M-net Semantics of $B(PN)^2$

Eike Best[1], Hans Fleischhack[2], Wojciech Frączak[3], Richard P. Hopkins[4], Hanna Klaudel[3] and Elisabeth Pelz[5]

This work is partly supported by ESPRIT WG CALIBAN, no 6067.

Abstract

Using a class of high level Petri nets, M-nets, endowed with composition operators resembling those of CCS, we give the compositional semantics of $B(PN)^2$ - a syntactically simple but semantically powerful concurrent programming language. We also give an associated low level net semantics and show the consistency of these high and low level semantics, as well as consistency with a previously defined low level semantics of $B(PN)^2$.

1 Introduction

$B(PN)^2$ (a basic Petri net programming notation, [2]) is a notation for the specification and programming of concurrent algorithms which incorporates within a simple syntax many of the constructs used in concurrent programming languages - (nested) parallel composition, iteration, guarded commands, and communication via both handshake and buffered communication channels, as well as shared variables. $B(PN)^2$ has an existing compositional Petri net semantics, given via the Petri Box Calculus (PBC, [3]) which is a formalism developed in order to support the application of net theoretical methods to the verification of concurrent algorithms and also to achieve the compositional semantics of languages needed to express them. This formalism supports operators, similar to the operators of CCS [15], on a class of labelled 1-safe place/transition nets. Furthermore, the PBC has an action structure allowing partial synchronisation between multisets of actions, similar to that in SCCS [15], but within an asynchronous framework. This plays a crucial role in obtaining a parsimonious compositional semantics for the powerful shared variable accessing provided in $B(PN)^2$.

The PBC semantics have been applied to program verification, in the *PEP* system [4] which supports: generation of a low level net from a $B(PN)^2$ program; interactive simulation of the net; verification of a temporal logic formula using a fast model checker. For any practical use of the semantics such as this, the size of the nets produced by the semantics becomes an issue in view of their machine representability and amenability to automated techniques: the elementary net may need to be infinite, namely, when the program contains a declaration of a variable with infinite value domain.

[1]Institut für Informatik, Universität Hildesheim, Germany
[2]Fachbereich Informatik, Universität Oldenburg, Germany
[3]LRI, Université Paris Sud, Orsay, France
[4]Department of Computing Science, University of Newcastle upon Tyne, UK
[5]LRI, Université Paris Sud, Orsay, and UPVM, Créteil, France

An obvious way to deal with this problem was to give the language a high level net semantics [10, 11, 17, 18]. Because compositionality remained of primary importance to us, it was necessary to view high level nets not just as abbreviations of low level nets, but to define suitable operations (corresponding to those of the PBC) on the domain of these nets. In the literature there are some high level net models providing some forms of compositionality, for instance those involving the composition of net modules by place and/or transition fusion [8, 16] or by synchronisation expressions as in [7]. However, there is no model providing the systematic action synchronisation required for the $B(PN)^2$ semantics, i.e., the partial synchronisation on action multisets described later in this paper. For this reason, we have developed and studied two classes of composable high level nets: M-nets (multilabelled high level nets [5]), a class of nets useful for this paper; and A-nets (algebraic multilabelled nets [9, 13]), a generalisation of M-nets supporting abstract data types. M-nets are a class of high level nets providing both vertical unfolding as usual, and also suitable horizontal composition operations. In [5], the coherence of these two kinds of operations has been shown there; that is, all horizontal operations commute with unfolding. Following this initial development, it is now the purpose of the present paper to demonstrate the use of M-nets in giving the high-level net semantics of programming languages, by giving an M-net semantics for $B(PN)^2$. Whilst there are benefits of a high level net semantics, there are also benefits of a low level net semantics (for a program yielding a net of tractable size) due to the greater availability of analysis techniques. Thus we also give here a compositional low level net semantics in terms of a restricted class of M-nets which correspond to labelled place/transition nets. We present a theorem which shows the consistency of the two semantics, and thus also the consistency between the new high level net semantics and the existing PBC semantics [2]. This consistency theorem relies on the coherence results proved in [5].

The structure of this paper is as follows. In Section 2 we give an extended informal introduction to both the $B(PN)^2$ programming language and to M-nets[6]. Sections 3 to 6 describe the syntax and M-net semantics of $B(PN)^2$. Section 7 describes the main result, while some concluding remarks can be found in Section 8.

2 Key Concepts

In this section we consider the following example program fragment:

$$\mathcal{F} \qquad \texttt{begin var } x : \{0,1,2\}; \quad \texttt{begin var } y : \{0,1\}; \langle 'x = 0 \wedge x' = 'y + 1 \wedge y' = 'y \rangle \ \texttt{end} \quad \texttt{end}.$$

The construct var x:$\{0, 1, 2\}$ is the declaration of a program variable x with type $\{0, 1, 2\}$ as the set of its possible values. The var y:$\{0, 1\}$ is similarly the inner declaration of y. The $\langle 'x = 0 \wedge x' = 'y + 1 \wedge y' = 'y\rangle$ denotes an atomic action comprising a logical expression. This accesses the variables x and y with $'x$ and $'y$ denoting their values prior to execution of the action ('pre-values')

[6] Formal definitions can be found in [1, 5].

and x' and y' denoting their values afterwards ('post-values'). The action can be executed, and establishes particular post-values, if the existing pre-values and the established post-values are such that the logical expression evaluates to `true`. In this case execution depends on x being initially 0, by $'x = 0$, and the effect is $x := y + 1$ (where $+$ is addition modulo 3), by $x' = 'y + 1 \wedge 'y = y'$ (the latter condition forces the value of y to be unchanged). We will also allow this to be written $\langle 'x = 0 \wedge x' = y + 1 \rangle$, where the use of plain y means $'y$ with the constraint $y' = 'y$.

2.1 The low level net semantics

In Figure 1(i)-(iii) we show the low level net semantics for each of the constructs of the example individually. The net (i) for the declaration of x has three places, x_0, x_1 and x_2 for its three possible values, and nine transitions for each of the possible accesses to x. For example, the transition **tx** labelled $\{\overline{x^A}(0,1)\}$ represents an access in which the value of x changes from 0 to 1; that labelled $\{\overline{x^A}(0,0)\}$ represents an access in which the value is and remains 0; and so on. The net (iii) for the declaration of y is analogous.

The net (ii) for the atomic action of the program fragment has: places a and b for representing program control being respectively immediately before and immediately after its execution; and a transition for each potential execution, i.e., for each combination of the potential pre- and post-values of the accessed variables for which the expression yields `true`. For example, the transition **t** labelled $\{x^A(0,1), y^A(0,0)\}$ corresponds to an execution with $'x = 0$, $x' = 1$, $'y = 0$, $y' = 0$.

A transition label, such as $\{\overline{x^A}(0,1)\}$ or $\{x^A(0,1), y^A(0,0)\}$, is a set of actions (generally, a multi-set, but here we may and will restrict ourselves to sets). Each such action can be thought of as a synchronisation capability, as actions in CCS. The action x^A(0,1) occurring in the label of transition **t** signifies that **t** needs to be synchronised with a transition having the conjugate action $\overline{x^A}(0,1)$ in its label - for instance, transition **tx** from the net for the declaration of x. We can think of action x^A(0,1) as representing the need for an access to x in which it changes from 0 to 1, and $\overline{x^A}(0,1)$ as representing that access. The $y^A(0,0)$ in the label of **t** means that **t** also needs to be synchronised with a transition having the conjugate action $\overline{y^A}(0,0)$ in its label - e.g., transition **ty** from the net for the declaration of y.

The complete semantics for the declaration of x and y with this atomic action as their scope would give the net shown in (viii) which can be obtained in terms of the operators on low level nets by parallel composition (i.e., juxtaposition) of the three nets (i)-(iii) and then applying synchronisation and then restriction, both over the appropriate action set - namely all actions of form $x^A(\ldots)$, $\overline{x^A}(\ldots)$, $y^A(\ldots)$ or $\overline{y^A}(\ldots)$.

The synchronisation would add the new transitions shown in (viii) as **t1** and **t2**. The transition **t1** brings together **tx**, **t** and **ty** which mutually satisfy each others' synchronisation needs, leaving no residual needs, and thus the empty set as label. The **t2** is likewise derived from **ux**, **u** and **uy**. These are the only possible complete synchronisations. Restriction over the action set then deletes any transitions having an action from that action set in its label, i.e., all the original transitions from (i)-(iii), leaving just the net shown in (viii).

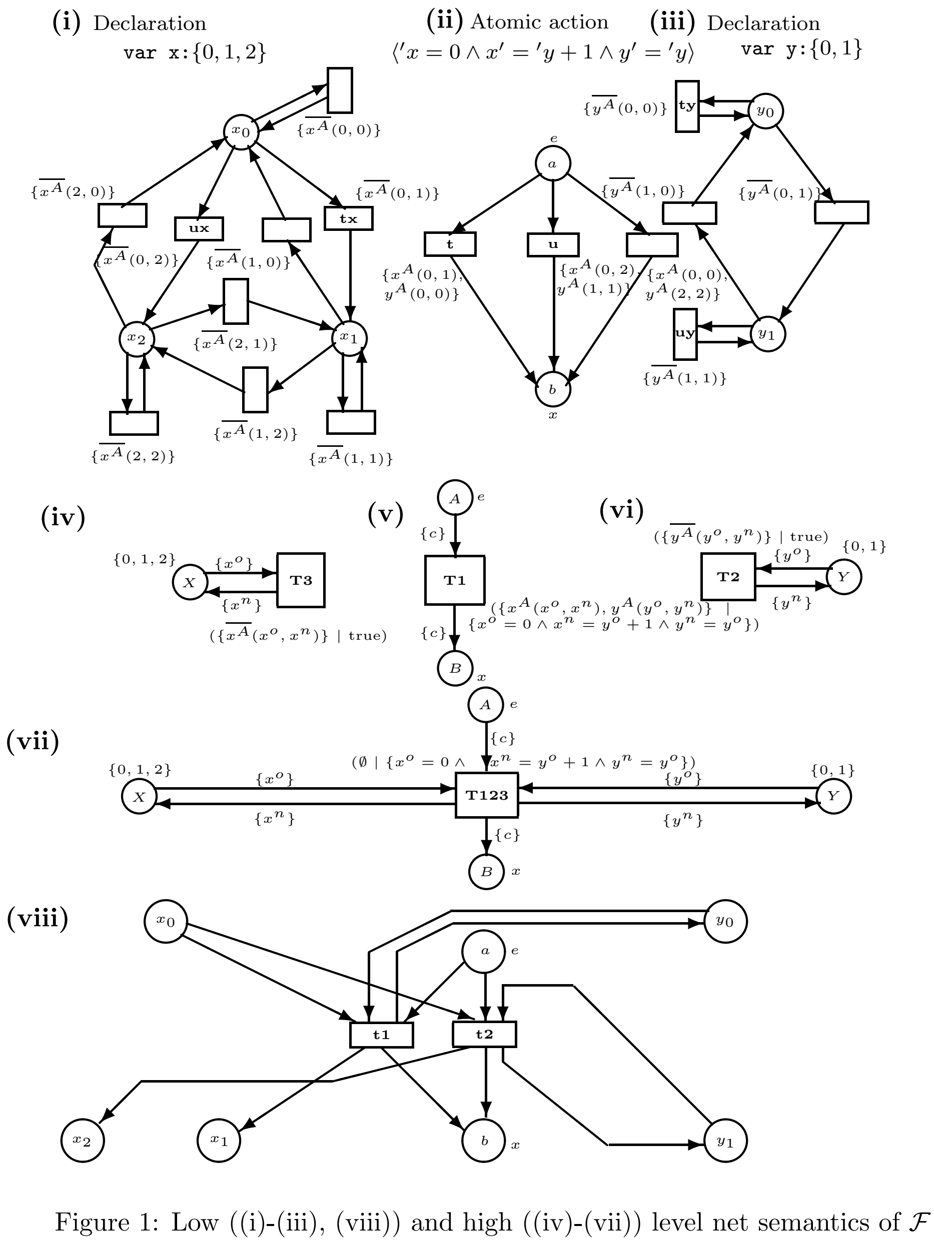


Figure 1: Low ((i)-(iii), (viii)) and high ((iv)-(vii)) level net semantics of $\mathcal{F}$

The transition **t1** represents the execution of the atomic action in the case of initially $x = 0$ and $y = 0$, requiring tokens on the corresponding places x_0 and y_0, and moving the former to the place x_1 for the updating of x to the value 1, and restoring the latter token to preserve the value of y. The transition **t2** is similarly the only other possible execution, in the case of initially $x = 0$ and $y = 1$.

There are two additional points to be made about the example so far.

The nets used have place labels as well as transition labels. In the net of (ii): Place a has label e, an 'entry' place[7]; place b has label x, an 'exit' place[8]. The third possibility for a place is to be 'internal' - for example, x_0, x_1, x_2, y_0, y_1 in (ii)/(iii) are internal. These have the empty label, which is omitted on figures. These labels are used in control structure compositions of nets. E.g., sequential composition of two nets is merging the exit places of the first net with the entry places of the second net.

The nets shown in (i)/(iii) for the variable declarations are not quite complete in that there would be two additional places, an entry place and an exit place, with initialisation and termination transitions. These additional elements have been omitted in order to avoid cluttering the figure, but they will be described later.

2.2 The high level net semantics

Figure 1(iv)-(vi) and (vii) show the high level net semantics analogously to the low level net semantics of (i)-(iii) and (viii), respectively. In (iv), X is the place holding the value of program variable x, and x^o and x^n (old x and new x) are net variables for its pre- and post- value; similarly in (vi), Y, y^o and y^n for variable y; A or B in (v) represent the program control points before and after the atomic action, respectively; they are able hold a black token for which net variable c is used. Let us consider (vii) and its unfolding to (viii) to cover those aspects of our high level nets which are similar to other high level net models, namely:

1. A place has a type, e.g. $\{0, 1, 2\}$ for X, which is the possible values for any token on that place. Omission of a type, as for A, means the singleton type $\{\bullet\}$, i.e. the place only sustains 'black' tokens.

2. An arc has an inscription, e.g. $\{x^o\}$ for the arc $X \to$ **T123** from X to **T123**, which in general is a multi-set of net variables. However, only singleton multisets are relevant here.

3. A transition has an annotation, e.g. the singleton set

   $$\{x^o = 0 \wedge x^n = y^o + 1 \wedge y^n = y^o\}$$

   for **T123**, which is a set of conjuncts defining an occurrence condition. The empty set will be shown as `true`.

4. A place unfolds to low level places for each value in its type. E.g., X unfolds to x_0, x_1 and x_2, denoting $x = 0$, $x = 1$ and $x = 2$, respectively.

[7] The initial marking of a net is a token on every entry place and no token elsewhere.
[8] By definition, a marking is final if all exit places (and no others) are marked.

5. A transition unfolds to low level transitions for each binding of its variables such that: all its conjuncts evaluate to `true`; each variable of an incident arc has a value consistent with the type of the connected place. E.g., **T123** unfolds to **t1** and **t2**, for bindings $\{x^o = 0, x^n = 1, y^o = 0, y^n = 0\}$ and $\{x^o = 0, x^n = 2, y^o = 1, y^n = 1\}$ respectively.

6. An arc $P \rightarrow \mathbf{T}$ with variable x unfolds to produce an arc $p \rightarrow \mathbf{t}$ for every instance of p, $\mathbf{t}$ and value v, such that p is the unfolding of P for v and $\mathbf{t}$ is an unfolding of $\mathbf{T}$ for a binding in which $x = v$. E.g., the arc $X \rightarrow$ **T123** unfolds (for $x^o = 0$) to $x_0 \rightarrow$ **t1** and $x_0 \rightarrow$ **t2**. The unfolding of an arc $\mathbf{T} \rightarrow P$ is analogous.

7. The behaviour of a high level net is consistent with that of the low level net to which it unfolds. E.g., if X, Y and A have tokens 0, 1 and $\bullet$, then there is an appropriate binding for the variables of **T123**, giving all its conjuncts as `true`, which allows its occurrence (corresponding to the occurrence of **t2**) by: removing the 0 token from X ($x^o = 0$); removing the 1 token from Y ($y^o = 1$); removing the $\bullet$ from A, and placing a $\bullet$ on B ($c = \bullet$); placing a 2 token on X ($x^n = 2$); placing a 1 token on Y ($y^n = 1$).

The aspects in which the high level net model here is less usual, are the place or transition inscriptions including not only type or occurrence condition, but also a label. The place labels identify each place as entry, exit or internal, and these have exactly the same roles as in the low level nets, that is (as already mentioned) to serve for the control structure compositions of nets. To illustrate high level transition labels we turn to Figure 1(iv)/(v) which shows the high level nets corresponding to the low level nets in (i)/(ii). A label, e.g. $\{x^A(x^o, x^n), y^A(y^o, y^n)\}$ for **T1**, is a multi-set of action terms, each being a 'parameterised' action which, for a particular binding of the variables in its parameter list, evaluates to an elementary action. E.g., for the binding $x^o = 0, x^n = 2$, the label $\overline{x^A}(x^o, x^n)$ of **T3** evaluates to $\overline{x^A}(0, 2)$, which is the label of the low level transition **ux** to which **T3** unfolds for that binding.

Our low level net domain, more precisely, is the class of elementary M-nets, which corresponds to labelled place/transition nets. An M-net is *elementary* if every place has singleton type and for every transition, its label consists of just elementary actions, and its annotation is the empty set. Types, transition annotations and arc inscriptions can thus be omitted from the representation.

2.3 Incremental partial synchronisation

In the program example, the variables x and y are declared by two distinct declarations in the program, and thus there will be two distinct semantic steps in the construction of (viii) from (i)-(iii), or (vii) from (iv)-(vi), i.e., first a partial synchronisation and restriction over y^A actions (for the declaration of y) and then a further partial synchronisation and restriction over x^A actions (for the declaration of x). In Figure 2 we show intermediate steps for the high level semantics, describing the high level net synchronisation mechanism. (The synchronisation mechanism for low level nets is a simplified version of that for high level nets.)

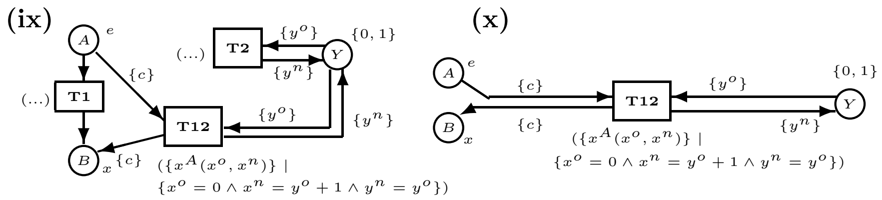


Figure 2: Synchronisation ((ix)) of (v) and (vi), followed by restriction ((x))

The parallel composition of the high level nets for the atomic action and the declaration of y would give the juxtaposition of nets (v) and (vi); and (ix) is the result of applying synchronisation over y^A actions to that. The conjugate actions $\overline{y^A}(y^o, y^n)$ in **T2** and $y^A(y^o, y^n)$ in **T1** allow these two to come together as **T12** which has: as its label, the multi-set sum of the labels of its two constituent transitions, but with the elimination of the pair of actions, y^A and $\overline{y^A}$, which allowed the synchronisation, leaving just $(\{x^A(x^o, x^n)\})$ as its annotation, the union of the annotations of the two constituent transitions, i.e., giving the occurrence condition as the conjunction of the constituents' occurrence conditions.

In (x) we see the result of applying restriction over y^A actions to (ix), which deletes all transitions with labels involving y^A actions. This completes the semantics of combining the declaration of y with the atomic action. The use of variable y has been internalised - the only external requirements are for access to a variable x, shown by the $x^A(x^o, x^n)$ action label. In particular, an outer declaration of a variable y would not affect this net.

In (vii) we see the result of further adding the declaration of x - parallel composition of (x) with (iv), then the application of synchronisation over x^A actions, and then the application of restriction over x^A actions – leaving just the transition **T123** which is the synchronisation of **T12** with **T3**.

This example is sufficient for understanding the use of the general synchronisation mechanism in the semantics of $B(PN)^2$, although the full definition of synchronisation is substantially more sophisticated, see [5].

3 Preliminaries to Syntax and Semantics

In the rest of the paper we present the syntax and both high and low level semantics of $B(PN)^2$ programs. We proceed top-down through the syntax, starting with program blocks and declarations (Section 4); then command connectives for parallel, sequence, choice and iteration (Section 5); and finally atomic actions (Section 6). The semantic functions are $Net_H(P)$ and $Net_L(P)$, yielding, respectively, the high level net and the low level net associated with some construct P.

The semantic definitions employ the unary net operations already described, namely for an M-net N and a set of action names A: unfolding, $U(N)$; synchronisation N **sy** A; restriction, N **rs** A. Unfolding is with respect to a set of values over which all net variables range, which here is taken as the set of all values supported by the particular realisation of $B(PN)^2$.

The semantics will also use some non-unary net composition operators, namely: parallel, $N1 \| N2$; sequential, $N1; N2$; choice, $N1 \,\Box\, N2$; and iteration, $[NI * NR * NF]$. Parallel composition is simply the disjoint union of the two nets (with renaming of the places and transitions to obtain the disjointness). Sequential composition, shown in Figure 3(i)-(iii), is parallel composition followed by a standard place multiplication construct (as in e.g. [12, 14]), to merge the exit places of the first net with the entry places of the second. Choice is a standard construction, similar to sequential composition, but with two place multiplications: of the entry places from one net with the entry places from the other; and of the exit places from one with the exit places from the other. Iteration, $[NI * NR * NF]$, is illustrated in Figure 3(iv)-(v). It involves three nets: initial, NI; repeated, NR; final, NF. The effect is one execution of NI; followed by zero or more executions of NR; followed by one execution of NF. The construction is to take two copies of each of the original nets and put them together, with the appropriate place multiplications, into the structure shown in (v) where **u1** and **u2** are copies of **u** etc.

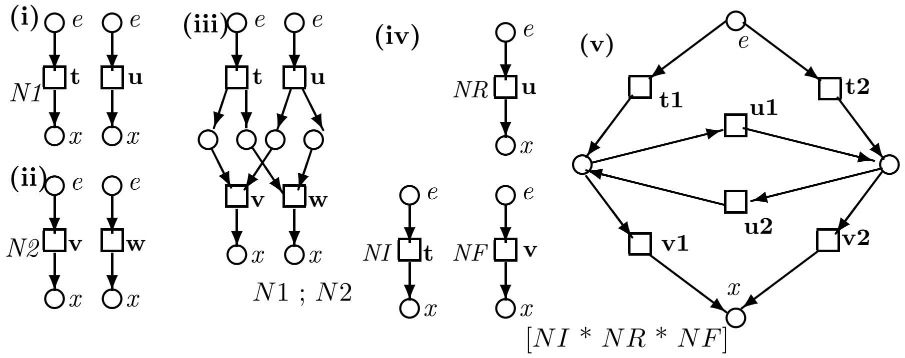


Figure 3: Illustrations of control structure compositions

4 Programs, Blocks and Declarations

The following is the syntax pertaining to programs *Prog*, blocks *Block*, (block) bodies B and declarations *Decl*:

$$\begin{array}{rcl} \mathit{Prog} ::= \mathit{Block} & ::= & \texttt{begin}\ B\ \texttt{end} \\ B & ::= & \mathit{Decl}\,; B \ \mid\ \mathit{Com} \\ \mathit{Decl} & ::= & \texttt{var}\ \mathit{name} : \mathit{set}\ \mid\ \texttt{var}\ \mathit{name} : \texttt{chan}\ b\ \texttt{of}\ \mathit{set} \\ b & ::= & 0 \mid 1 \mid \ldots \mid \omega \end{array}$$

A declaration involves a variable identifier *name* and the specification *set* of a type, i.e., a set of possible values (e.g. `int` or $\{0, 1, 2\}$). Either ordinary variables, or channel variables, may be declared. The declarations may be followed by a command *Com*. For the syntax and semantics of *Com*, the reader is referred to the next section.

4.1 Blocks

The main idea in describing a block is to juxtapose the nets for its declarations and the net for its command, and then to synchronise all matching transitions and to restrict in order to make them invisible outside the block (see Section 2). Just as a variable declared inside a block may gain an initial value on block entry, it must lose its value on block exit, because otherwise value inconsistencies might arise (for instance if the block in question is nested inside a loop). For this reason, we also associate a termination part $\tau(D)$ with every declaration D. More precisely, for both high and low level semantics, the following are the steps in obtaining the net semantics of a block body $D;B$, where D is the declaration of a program or channel variable w.

1. The sequential composition of the net for B and the net $\tau(D)$ for the termination of w, which comprises a single transition labelled with a special termination action for w called w^{A_T}.
2. The parallel composition of that with the net for the declaration.
3. The application to that of synchronisation and restriction over $\delta(D)$, the set of all actions pertaining to w.

Definition 4.1 Using the above notation and the parameterised class of nets $Net_H(D)$ defined in sections 4.2 and 4.3 below, and for I $\in$ {H,L}:

$$
\begin{array}{lll}
& Net_I(\texttt{begin } B \texttt{ end}) & = Net_I(B) \\
& Net_I(D;B) & = ((Net_I(D) \parallel (Net_I(B);\tau(D))) \textbf{ sy } \delta(D)) \textbf{ rs } \delta(D) \\
\text{and} & Net_L(D) & = U(Net_H(D)),
\end{array}
$$

$$
\text{where } \tau(D) = \left(\overset{\text{e}}{\bigcirc} \xrightarrow{\{c\}} \overset{\tau}{\square} \xrightarrow{\{c\}} \overset{\text{x}}{\bigcirc} \right)
$$

with $\tau = (\{w^{A_T}\} \mid \texttt{true})$ and $\delta(D) = \{w^A, w^{A_?}, w^{A_!}, w^{A_T}\}$. $\diamond$

Note the low level net for a declaration is defined via unfolding[9]. Next we complete the definition by treating the two types of variable declarations separately.

4.2 Ordinary variables

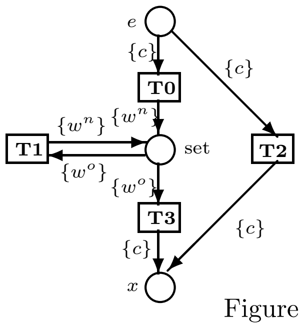
Definition 4.2 $Net_H(\texttt{var w : set}) = M_{data}(w, \texttt{set})$, that is, the net shown in Figure 4. $\diamond$

The high level net semantics of an ordinary variable thus consists of a central place for the value of the variable together with a transition, **T1**, for the possible value changes. In addition, there are initialising and terminating transitions **T0**, **T2** and **T3**. The transitions **T2** and **T3** have to synchronise with the termination action τ associated to D via the net $\tau(D)$ according to Definition 4.1. The initialisation transition **T0** represents the first access to the variable,

[9] For a more explicit representation of this net, compare [2].

which may assume any pre-value of the variable. E.g. in any of the following the first atomic action could successfully be executed -

$$\texttt{var}\ w : \{0,1,2\}; \langle 'w = 0 \wedge w' = 2\rangle \ldots$$
$$\texttt{var}\ w : \{0,1,2\}; \langle 'w = 1 \wedge w' = 2\rangle \ldots$$
$$\texttt{var}\ w : \{0,1,2\}; \langle w' = 2\rangle \ldots$$



Inscriptions:

T0: $(\{\overline{w^A}(w^o, w^n)\} \mid \text{true})$
T1: $(\{\overline{w^A}(w^o, w^n)\} \mid \text{true})$
T2: $(\{\overline{w^{A_T}}\} \mid \text{true})$
T3: $(\{\overline{w^{A_T}}\} \mid \text{true})$

Figure 4: $M_{data}(w, set)$

4.3 Channel variables

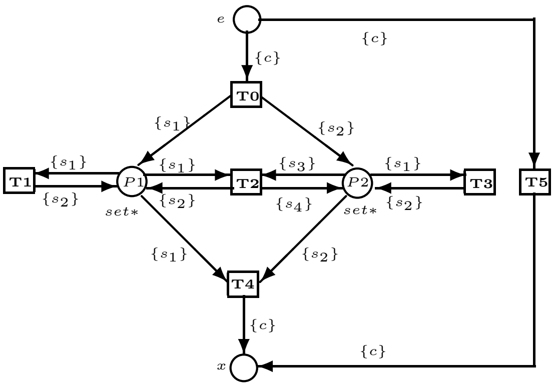
The declaration, `var w : chan b of set`, with $b \neq 0$, declares a channel of capacity b which acts as a FIFO buffer of values. A $w!$ (send) which appears in an atomic action within the scope of the declaration causes a value to be appended to the end of the buffer and a $w?$ (receive) causes the first value to be removed from the buffer. The special case of $b = \omega$ is for a buffer of unbounded capacity. For the case of $b = 0$, the buffer has capacity 0 which enforces handshake sychronisation between a send and a receive.

Definition 4.3 $Net_H($`var w : chan b of set`$) = M_{chan}(w, b, set)$, the nets shown in Figures 5 and 6. $\diamond$

In Figure 5, to model the queue of values in a channel we consider a (complex) type $set*$ of tokens. $set*$ is the set of all sequences with entries from set. In general, sequences are denoted by s; a value $v \in set$ is identified with a sequence of length 1 (singleton sequence); the sequential composition of sequences is denoted by a dot .; ϵ denotes the empty sequence; and, finally, the length of sequence s, is denoted by $lh(s)$.

To enable concurrent send and receive actions, the contents of a channel is modelled by two sequences: that held on place $P2$, either empty or the first element in the channel; that held on place $P1$, the rest of the channel (which is the whole channel if $P2$ holds empty). The channel is initialised by the first sending of a value v to the channel. The initialisation transition **T0** places this value on $P1$, and the empty sequence on $P2$. Then there is one transition, **T1**, for the general send action, appending the communicated value to the content of place $P1$, provided this could not result in the channel capacity being exceeded - condition $lh(s_2) < b$ which will always yield `true` if $b = \omega$ (unbounded capacity), and otherwise ensures at most $b-1$ values on $P1$.

There is one transition, **T3**, for the receive action, removing the communicated value from place $P2$ and replacing it by ϵ. An internal transition, **T2**, transfers the first entry of the sequence on place $P1$ to place $P2$, if this contains ϵ. In addition, we have two terminating transitions, **T4** and **T5** which are analogous to those of an ordinary variable declaration (and which allow termination even when the channel is non-empty).



T0: $\{\overline{w^{A_!}(w^!)}\} \mid \{s_1 = w^!, s_2 = \epsilon\}$
T1: $(\{\overline{w^{A_!}(w^!)}\} \mid \{lh(s_2) < b,$
$s_2 = s_1.w^!\})$
T2: $(\emptyset \mid \{s_1 = v.s, lh(v) = 1,$
$s_2 = s, s_3 = \epsilon, s_4 = v\})$
T3: $(\{\overline{w^{A_?}(w^?)}\} \mid \{s_1 = w^?, s_2 = \epsilon\})$
T4: $(\{\overline{w^{A_T}}\} \mid \text{true})$
T5: $(\{\overline{w^{A_T}}\} \mid \text{true})$

Figure 5: $M_{chan}(w, b, set), b \neq 0, b \neq 1$

For the case, $b = 1$, the channel may either be empty, or contain a singleton sequence of values. The net for this case is shown in Figure 6 (i). It can be seen as being obtained from $M_{chan}(w, b, set)$ by merging the two places $P1$ and $P2$.

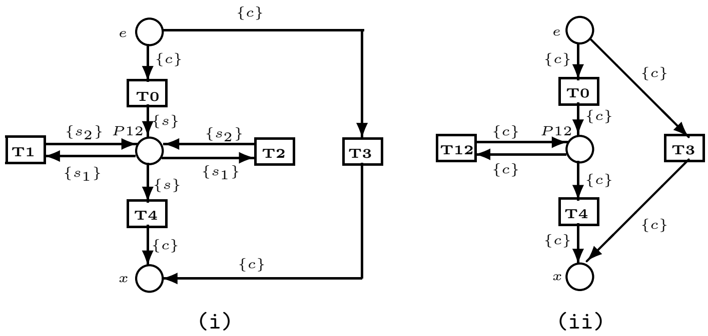
For the case, $b = 0$ (i. e., in the case of synchronous communication), the net of Figure 6(i) is modified in the following way to give Figure 6(ii). The value set of the internal place becomes $\{\bullet\}$. The send transition and the receive transition are folded into a single transition, which has to synchronise with a send and a receive action and whose occurrence condition assures that both the communicated values coincide.

5 Command Connectives

The following is the syntax for a command, which can be: a block (see previous section); an atomic action, *AtAct*, which is the basic unit of commands and is described in the next section; or a combination of commands by one of the three connectives dealt with in this section, namely sequential composition, parallel composition and choice/iteration (which are both described by the `do`–`od` construct):

$$\begin{array}{rcl} \mathit{Com} & ::= & \mathit{Block} \mid \mathit{AtAct} \mid \mathit{Com}; \mathit{Com} \mid \mathit{Com} \| \mathit{Com} \mid \texttt{do}\ \mathit{alt\text{-}set}\ \texttt{od} \\ \mathit{alt\text{-}set} & ::= & \mathit{alt\text{-}set} \ \Box\ \mathit{alt\text{-}set} \mid \mathit{Com}; \texttt{exit} \mid \mathit{Com}; \texttt{repeat}. \end{array}$$

Within a `do`–`od` construct, we may have `repeat` clauses, the 'repeat-set', and `exit` clauses, the 'exit-set'. The effect is repeated executions of arbitrarily chosen executable clauses of the repeat-set, possibly followed by a single



$M_{chan}(w, 1, set)$:

$P12$:	$\{s \in set* \mid lh(s) \leq 1\}$
T0:	$(\{\overline{w^{A!}}(w^!)\} \mid \{s = w^!\})$
T1:	$(\{\overline{w^{A!}}(w^!)\} \mid s_1 = \epsilon, s_2 = w^!\})$
T2:	$(\{\overline{w^{A?}}(w^?)\} \mid \{s_1 = w^?, s_2 = \epsilon\})$
T3:	$(\{\overline{w^{A_T}}\} \mid \text{true})$
T4:	$(\{\overline{w^{A_T}}\} \mid \text{true})$

$M_{chan}(w, 0, set)$:

$P12$:	$\{\bullet\}$
T0:	$(\{\overline{w^{A!}}(w^!), \overline{w^{A?}}(w^?)\} \mid \{w^! = w^?, w^! \in set\})$
T12:	$(\{\overline{w^{A!}}(w^!), \overline{w^{A?}}(w^?)\} \mid \{w^! = w^?, w^! \in set\})$
T3:	$(\{\overline{w^{A_T}}\} \mid \text{true})$
T4:	$(\{\overline{w^{A_T}}\} \mid \text{true})$

Figure 6: (i) $M_{chan}(w, 1, set)$ (ii) $M_{chan}(w, 0, set)$

execution of one clause of the exit-set. If a clause of the exit-set is chosen, termination of the whole construct ensues (unless, of course, this clause has a deadlock or an infinite loop). Two examples are -

1. `do` $\langle n > 0\rangle; \langle y' = {}'y + n\rangle; \langle n' = {}'n - 1\rangle;$ `repeat` $\square$ $\langle n \leq 0\rangle;$ `exit od`,

   which is equivalent to `while` $n > 0$ `do begin` $y := y + n; n := n - 1$ `end`.

2. `do` $\langle c? = x'\rangle; \langle n' = {}'n + 1\rangle;$ `exit` $\square$ $\langle d? = x'\rangle; \langle m' = {}'m + 1\rangle;$ `exit od`,

   which is equivalent to `do` $c?x \rightarrow n := n + 1 \;\square\; d?x \rightarrow m := m + 1$ `od`.

The semantics of the $B(PN)^2$ `do`–`od` construct involves two supplementary semantic functions relating to the net iteration operator $[NI * NR * NF]$ used, namely R to give, for use in NR part, the net which is choice over the nets from all the repeat-set clauses, and E similarly for the exit-set clauses and the NF part. The NI part is the net for a single silent transition. Thus, for $I \in \{H, L\}$:

$$\begin{aligned}
R_I(\mathit{alt\text{-}set}_1 \square \mathit{alt\text{-}set}_2) &= R_I(\mathit{alt\text{-}set}_1) \square R_I(\mathit{alt\text{-}set}_2) \\
E_I(\mathit{alt\text{-}set}_1 \square \mathit{alt\text{-}set}_2) &= E_I(\mathit{alt\text{-}set}_1) \square E_I(\mathit{alt\text{-}set}_2) \\
R_I(\mathit{Com}; \texttt{repeat}) = E_I(\mathit{Com}; \texttt{exit}) &= \mathit{Net}_I(\mathit{Com}) \\
R_I(\mathit{Com}; \texttt{exit}) = E_I(\mathit{Com}; \texttt{repeat}) &= N_{stop} = \left(\begin{matrix} e & & x \\ \bigcirc & & \bigcirc \end{matrix} \right).
\end{aligned}$$

The high and low level semantics of program connectives are the same, involving applications of the appropriate net operators:

Definition 5.1 For $I \in \{H, L\}$:

$$\begin{aligned}
\mathit{Net}_I(P_1 \| P_2) &= \mathit{Net}_I(P_1) \| \mathit{Net}_I(P_2) \\
\mathit{Net}_I(P_1; P_2) &= \mathit{Net}_I(P_1); \mathit{Net}_I(P_2) \\
\mathit{Net}_I(\texttt{do}\ \mathit{alt\text{-}set}\ \texttt{od}) &= [N_{silent} * R_I(\mathit{alt\text{-}set}) * E_I(\mathit{alt\text{-}set})]
\end{aligned}$$

where $N_{silent} = \left(\overset{\mathrm{e}}{\bigcirc} \xrightarrow{\{c\}} \overset{\sigma}{\Box} \xrightarrow{\{c\}} \overset{\mathrm{x}}{\bigcirc} \right)$ with $\sigma = (\emptyset \mid \texttt{true})$. $\diamond$

6 Atomic Actions

The following is the syntax for atomic actions, *AtAct*, and constituent expressions, *expr*:

$$
\begin{array}{rcl}
\mathit{AtAct} & ::= & \langle\, \mathit{expr} \,\rangle \\
\mathit{expr} & ::= & {'w} \;\mid\; w' \;\mid\; w? \;\mid\; w! \;\mid\; \mathit{const} \;\mid\; \mathit{expr}\ \mathit{op}\ \mathit{expr} \;\mid\; \mathit{op}\ \mathit{expr}.
\end{array}
$$

Operators (*op*, e.g. $+$) and constants (*const*, e.g. 1) are standard programming constructs, details of which depend on a particular realisation of $B(PN)^2$. The identifier w is a program variable, which may be declared as an ordinary variable, in which case the forms $'w$ and w' are meaningful, as discussed in Section 2. A program variable may also be declared as a channel variable in which case the forms $w?$ and $w!$ are meaningful. The construct $w?$ requires an input on the channel w and denotes the value received. The construct $w!$ requires an output on the channel w and denotes the value sent. E.g., $\langle w? = x' \rangle$ and $\langle w! = x + 2 \rangle$ correspond to, respectively, `occam` notation $w?x$ and $w!(x + 2)$.

First we define a supplementary semantic function, *Ins*, to yield the semantics of an expression as an action-expression pair $(AS|E)$, which is similar to a transitions inscription. AS is a set of action terms, the action names used being: w^A for access to an ordinary variable w; $w^{A_?}$ for input from, and $w^{A_!}$ for output to, a channel variable w. E is a term, which is e with program variables appropriately replaced by net variables: w^o for $'w$; w^n for w'; $w^?$ for $w?$; $w^!$ for $w!$.

$$
\begin{array}{lcl}
\mathit{Ins}('w) & = & (\{w^A(w^o, w^n)\} \mid w^o) \\
\mathit{Ins}(w') & = & (\{w^A(w^o, w^n)\} \mid w^n) \\
\mathit{Ins}(w?) & = & (\{w^{A_?}(w^?)\} \mid w^?) \\
\mathit{Ins}(w!) & = & (\{w^{A_!}(w^!)\} \mid w^!) \\
\mathit{Ins}(\mathit{const}) & = & (\emptyset \mid \mathit{const}) \\
\mathit{Ins}(e_1\, \mathit{op}\, e_2) & = & (AS_1 \cup AS_2 \mid E_1\, \mathit{op}\, E_2) \text{ for } \mathit{Ins}(e_i) = (AS_i \mid E_i) \\
\mathit{Ins}(\mathit{op}\, e) & = & (AS \mid \mathit{op}\, E) \text{ for } \mathit{Ins}(e) = (AS \mid E).
\end{array}
$$

E.g., for $\langle 'x = 0 \wedge x' = c? + 1 \rangle$, the semantics is obtained via:

$$
\begin{array}{lcl}
\mathit{Ins}('x = 0) & = & (\{x^A(x^o, x^n)\} \mid x^o = 0) \\
\mathit{Ins}(x' = c^? + 1) & = & (\{x^A(x^o, x^n), c^{A_?}(c^?)\} \mid x^n = c^? + 1) \\
\mathit{Ins}('x = 0 \wedge x' = c^? + 1) & = & (\{x^A(x^o, x^n), c^{A_?}(c^?)\} \mid x^o = 0 \wedge x^n = c^? + 1)
\end{array}
$$

By the following definition, this would give a net like Figure 1(v), except that the transition inscription is $(\{x^A(x^o, x^n), c^{A_?}(c^?)\} \mid \{x^o = 0 \wedge x^n = c^? + 1)\})$.

Definition 6.1 For $\mathit{Ins}(\mathit{expr}) = (AS|E)$,

$$
\begin{array}{rcl}
\mathit{Net}_H(\langle \mathit{expr} \rangle) & = & \left(\overset{\mathrm{e}}{\bigcirc} \xrightarrow{\{c\}} \overset{\alpha}{\Box} \xrightarrow{\{c\}} \overset{\mathrm{x}}{\bigcirc} \right), \quad \text{where} \quad \alpha = (AS \mid \{E\}) \\
\mathit{Net}_L(\langle \mathit{expr} \rangle) & = & U(\mathit{Net}_H(\langle \mathit{expr} \rangle)).
\end{array}
$$

$\diamond$

Note that, as for declarations, the low level semantics is defined via unfolding.

7 Consistency Theorem

The following theorem shows the consistency of the high and low level semantics.

Theorem 7.1 *(Commutativity of semantics and unfolding)*
For any syntactic construct P for which is defined $Net_H(P)$ (and so too $Net_L(P)$), we have $U(Net_H(P)) = Net_L(P)$.

Proof: By induction on the structure of P. The theorem is trivial for the base cases, viz. for declarations and for atomic actions, for which it is established by definition. The induction steps rely on some non-trivial results for M-nets [5]. The results used are: first, for elementary M-nets N (such as N_{silent} or N_{stop}), unfolding is identity, i.e., $U(N) = N$, and second, all operators commute with unfolding, i.e.:

$$\begin{array}{lcll} U(N_1 \; op \; N_2) & = & U(N_1) \; op \; U(N_2) & \text{for } op \in \{ \; \| \; , \; ; \; , \; \square \; \} \\ U(N \; op \; A) & = & U(N) \; op \; A & \text{for } op \in \{ \; \mathbf{sy} \; , \; \mathbf{rs} \; \} \\ U([NI * NR * NF]) & = & [U(NI) * U(NR) * U(NF)]. & \end{array}$$

For the declaration, D, of an (ordinary or channel) variable w together with its scope B, i.e. $D; B$, the induction step requires $Net_L(D; B) = U(Net_H(D; B))$, which is proved as follows:

$$\begin{array}{lll} & Net_L(D; B) & \\ = & (Net_L(D) \| (Net_L(B); \tau(D))) \; \mathbf{sy} \; \delta(D) \; \mathbf{rs} \; \delta(D) & \text{– definition of } Net_L(D; B) \\ = & (U(Net_H(D)) \| (U(Net_H(B)); U(\tau(D)))) \; \mathbf{sy} \; \delta(D) \; \mathbf{rs} \; \delta(D) & \text{– induction hypoth.} \\ = & U(Net_H(D) \| (Net_H(B); \tau(D)) \; \mathbf{sy} \; \delta(D) \; \mathbf{rs} \; \delta(D)) & \text{– commutativity of unfolding,} \\ & & \text{and } \tau(D) \text{ being elementary} \\ = & U(Net_H(D; B)) & \text{– definition of } Net_H(D; B) \end{array}$$

For the induction steps pertaining to the parallel and sequential constructs, viz.

$$Net_L(P_1 \| P_2) = U(Net_H(P_1 \| P_2)) \;\; \text{and} \;\; Net_L(P_1; P_2) = U(Net_H(P_1; P_2)),$$

the proofs are similar to the above. For iteration and choice, there need to be subsidiary inductions to show

$$U(R_H(\textit{alt-set})) = R_L(\textit{alt-set}) \;\; \text{and} \;\; U(R_H(\textit{alt-set})) = R_L(\textit{alt-set}).$$

8 Conclusion

In this paper we have used the domain of M-nets and the operations defined on this domain to give both a high level and low level net semantics of the specification and programming language $B(PN)^2$; and have shown the consistency of these two semantics and thus the consistency of the high level net semantics with the original semantics of [2]. These semantics (both high and low level) have four important properties -

P1 They are compositional in that every program construct has a semantics which is independent of its context, and (in the case of a compound construct) is defined only in terms of the semantics of its constituents.

P2 They yield a single transition occurrence for an atomic action execution, despite the generality of the atomic action construct in supporting write access to multiple shared variables together with simultaneous communication on multiple channels.

P3 They yield a concurrent semantics for the parallel execution construct of the language.

P4 They are systematic, and thus expressing an algorithm as a $B(PN)^2$ program fixes a standard Petri net representation of the algorithm (avoiding the need to invent *ad hoc* representations whenever some analysis is to be performed).

It is the synchronisation structure on multi-sets of actions which allows the combination of P1 and P2, as was illustrated in the discussion of Figure 2. In contrast, if we used the simpler CCS style, then property P2 could not be achieved (as indeed it is not in the CCS approach to programming language semantics illustrated in [15]). Moreover, if we used the SCCS style, which has a somewhat similar synchronisation on multi-sets of actions, then property P3 could not be achieved.

The semantics was developed not just for theoretical reasons, but also for practical application in program verification, to which all of the above properties are relevant (P1 – P4 all contributing to the comprehensibility of the net produced by the semantics and P3 enabling various efficiency techniques in state space searching). The *PEP* system, originally supporting simulation and verification on the low level net semantics of $B(PN)^2$, has now been enhanced to a system (exhibited at the Hannover Computer Fair CeBIT'95, March 1995) which incorporates: the high level net semantics defined here, yielding smaller and more readable net models for programs than does the low level net translation; a fast interpreter for a $B(PN)^2$ program via the simulation of its high level net semantics; the unfolding of those semantics for the purpose of verification on the resulting low level nets. Work is in progress on developing verification techniques to be applied directly to the M-net semantics.

Petri net research in general has so far yielded fewer (and less powerful) analysis techniques for high level nets than for low level nets. Thus at the present time there is a trade-off: using low-level nets gives more analysis power but less easy machine representability; while using high level nets allows nice visualisation of every program in a small net but less analysis capabilities. However, as there is an ever-growing body of results pertaining to the analysis of high level nets, investing in the implementation of a high level net semantics can be done in anticipation of future results in that area.

References

[1] Eike Best, Raymond Devillers, Elisabeth Pelz, Arend Rensink, Manuel Silva and Enrique Teruel: Causal Calculi Based on Nets (CALIBAN). In this volume.

[2] Eike Best and Richard P. Hopkins: B(PN)^2 - a Basic Petri Net Programming Notation. In *Proceedings of PARLE'93*, Volume 694 of LNCS, pages 379–390. Springer Verlag, 1993.

[3] Eike Best, Raymond Devillers and Jon Hall: The Box Calculus: a New Causal Algebra with Multi-label Communication. In *Advances in Petri Nets 1992*, Volume 609 of *LNCS*, pages 21-69. Springer Verlag, 1992.

[4] Eike Best and Hans Fleischhack: *PEP - Programming Environment Based on Petri Nets* (project supported by the Deutsche Forschungsgemeinschaft, 1993-1994).

[5] Eike Best, Hans Fleischhack, Wojciech Frączak, Richard P. Hopkins, Hanna Klaudel and Elisabeth Pelz: A Class of Composable High Level Petri Nets, to appear in *Proc. of 16th ICPN*, Torino, LNCS, Springer Verlag, June 1995.

[6] Eike Best, Hans Fleischhack, Wojciech Frączak, Richard P. Hopkins, Hanna Klaudel and Elisabeth Pelz: An M-net Semantics of $B(PN)^2$, Hildesheimer Informatikbericht 10/95, Hildesheim, 1995.

[7] Didier Buchs and Nicolas Guelfi: CO-OPN: A Concurrent Object-Oriented Petri Nets Approach for System Specification. In *Proc. of 12th ICPN*, Århus, 1991, also LRI Technical Report 616, Orsay, 1990.

[8] Soren Christensen and Laure Petrucci: Towards a Modular Analysis of Coloured Petri Nets. In *Advances in Petri Nets 1992*, Volume 609 of LNCS, pages 113-133. Springer Verlag, 1992.

[9] Raymond Devillers and Hanna Klaudel: Refinement and Recursion in a High Level Petri Box Calculus. In this volume.

[10] Hartmann Genrich: Predicate-transition Nets. In *High Level Petri Nets: Theory and Application*, pages 3-43. Springer Verlag, 1991.

[11] Kurt Jensen: Coloured Petri Nets. In *High Level Petri Nets: Theory and Application*, pages 44-122. Springer Verlag, 1991.

[12] Ursula Goltz: On Representing CCS Programs by Finite Petri Nets. In *Proc. MFCS'88*, Volume 324 of LNCS, pages 339-350. Springer Verlag, 1988.

[13] Hanna Klaudel and Elisabeth Pelz: *Communication as Unification in the Petri Box Calculus*, to appear in *Proc. FCT'95*, Dresden, LNCS, Springer Verlag, August 1995.

[14] Vadim Kotov: An Algebra for Parallelism based on Petri Nets. In *Proc. MFCS'78*, Volume 64 of LNCS, pages 39-55. Springer Verlag, 1978.

[15] Robin Milner: *Communication and Concurrency.* Prentice Hall (1989).

[16] Giuseppe Pinna: *Petri Nets and Their Composition Problem.* PhD Thesis: TD-2/90, Dipartimento di Informatica, Università degli Studi di Pisa, March 1990.

[17] Wolfgang Reisig: Petri Nets and Algebraic Specifications. In *Theoretical Computer Science* Vol.80, 1-34 (1991).

[18] Jacques Vautherin: Parallel systems specification with colored Petri nets and algebraic specification. In *Advances in Petri Nets 87*, Volume 266 of LNCS. Springer Verlag, 1987.