Acta Informatica Manuscript-Nr.
(will be inserted by hand later)

M-nets: an Algebra of High-level Petri Nets, with an Application to the Semantics of Concurrent Programming Languages

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

[1] Fachbereich Informatik, Carl-von-Ossietzky-Universität zu Oldenburg, D-26111 Oldenburg, Germany; e-mail: e.best@informatik.uni-oldenburg.de
[2] Département d'Informatique, Université du Quebec à Hull, C.P. 1250, succ. B, Hull, Quebec, Canada J8X-3X7; e-mail: fraczak@UQAH.UQuebec.CA
[3] Department of Computing Science, University of Newcastle upon Tyne, UK; e-mail: R.P.Hopkins@newcastle.ac.uk
[4] Université Paris Val de Marne, Equipe d'Informatique, 61 Av. du Général de Gaulle, Bât.P3, 4^e étage, F-94010 Créteil, France; e-mail: {klaudel,pelz}@lri.fr

Received November 15, 1996; accepted April 23, 1997

Abstract. This paper describes a high-level Petri net model called M-nets (for modular multilabelled nets). A distinctive feature of this model is that it allows both: unfolding, as do most other high-level net models; and composition – in particular, synchronisation – in a process algebraic style, turning the set of M-nets into an algebraic domain. It turns out that the composition operations of this domain have various algebraic properties. Moreover, the model is such that composition operations are coherent with unfolding, in the sense that the unfolding of a composite high-level net is the composition of the unfoldings of its components.

One of the motivations for M-nets is that they be a vehicle for giving semantics of concurrent programming languages. To illustrate their capability for that, the compositional semantics of $B(PN)^2$ – a simple, expressive concurrent programming language – is given. An associated low-level net semantics is described, and the coherence of these high-level and low-level semantics is proved.

Key words: Compositional Semantics – Concurrent Programming Languages – High-level Petri Nets – M-nets

1. Introduction and Motivation

High-level Petri net models, as described for instance in [3, 17, 23, 36, 38, 39, 40], incorporate an unfolding rule. Unfolding takes any high-level net H (possibly together with an interpretation) and creates from this an elementary (low-level) net L. H can be seen as an abbreviation of L, and L as the semantics of H. Unfolding is governed by place and transition annotations. A place annotation can be a set of allowed tokens on that place, and in the unfolding, a place unfolds to distinct places for all the members of that set. A transition annotation can be a Boolean expression, and in the unfolding, a transition unfolds to distinct transitions for all the legal bindings (i.e., giving values to variables) making the expression true. Annotations also govern the transition rule of H, through which its behaviour is generated. Unfolding and behaviour are coherent, in the sense that behaviours of a high-level net and behaviours of the corresponding unfolded low-level net correspond to each other.

In this paper we choose a particular high-level net model, which we propose to extend by adding operations allowing large high-level nets to be composed from smaller parts in a modular way. They include operations such as sequential and parallel composition, transition synchronisation and restriction (and others), similar to a process algebraic setup [1, 20, 29, 30]. The way we define these operations is by using a further function from places and transitions to suitable objects. To distinguish them from annotations, we call these other objects 'labels'. This name is appropriate because the special case of elementary M-nets corresponds to what is widely known as 'labelled P/T-nets' [19, 22, 32, 33, 41]. Thus, in our enriched model, places and transitions are inscribed by pairs in the form (label, annotation).

We propose to consider elementary nets as special high-level nets. It follows that the definition of composition operations on the high level implies the same definitions for the low level, immediately giving rise to a question of coherence: *Does the unfolding of a composed net equal the composition of the unfolding(s) of its part(s)*? It is one of the main aims in designing the M-net model to answer this question positively for all operations that are defined on it. It will be seen that in order to satisfy this property, the synchronisation operation, in particular, requires a careful definition involving unification, logical conjunction and multiset addition of appropriate expressions and (multi)sets. The first half of this paper, comprising section 2, is devoted to defining M-nets, their unfolding and their composition operations, and finally proving the first main result of this paper, coherence between compositions and unfolding.

A second important aim of the M-net model has been to tune its composition operations in such a way that they can be used to define the compositional semantics of (nontrivial) concurrent programming languages as easily as possible. For this reason, the M-net operations are the same as those of the Petri Box Calculus (PBC, [4]) which has been defined for low-level nets with the same purpose. The PBC has been applied in giving the semantics of $B(PN)^2$ (Basic Petri Net Programming Notation, [8]) and a precursor to the PBC has been applied [21] in giving a semantics for occam [28] which were shown to be consistent with the standard occam semantics.

The PBC supports operators similar to the operators of CCS [30]. However, it has actions allowing partial synchronisation between multisets of actions, similar to that in SCCS [30], but within an asynchronous framework. This plays a crucial role in obtaining a parsimonious compositional semantics for the shared variable accessing provided in programming notations such as $B(PN)^2$. This language is a good test bed because it comprises shared memory parallelism, channel communication with arbitrary capacities and allows the nesting of parallel operators as well as of blocks. Therefore, in the second part of this paper (comprising section 3), we use $B(PN)^2$ as an example for obtaining a compositional semantics through the M-net operations defined in section 2. We will define, at the same time, both a high-level and a low-level semantics of $B(PN)^2$. The main result, described at the end of section 3, is that the two semantics are coherent, meaning that the low-level net of a program is the same, whether it is constructed directly or indirectly via the high-level net and unfolding.

This article presents M-nets in a slightly more general setting than [5, 6], where the model has been published first. The following gives an overview of its structure. In section 2, the domain of M-nets is defined. Sections 2.1 and 2.2 contain basic definitions of M-nets and their place, transition and arc inscriptions, labels and annotations. In section 2.3 the behaviour of M-nets is defined. Unfolding is defined as well; both notions depend on the annotations rather than the labels of an M-net. In section 2.4 a set of M-net composition operations are introduced. These operations are governed by the labels rather than by the annotations of participating M-net(s). Sections 2.5 and 2.6 contain theorems about coherence and algebraic properties. In section 3, the $B(PN)^2$ notation is introduced and its M-net semantics is given. Section 3.1 describes a motivating example. In section 3.2 the syntax and the semantics of $B(PN)^2$ are defined. Section 3.3 contains the coherence theorem and its proof. The paper ends with a concluding section and a list of references. It also contains an appendix with some technical definitions.

2. M-nets: a Domain of Modular Multilabelled Nets

The principal difference between M-nets and coloured nets [23] is that M-nets carry additional information on their places and transitions to support composition operations. Moreover, the high-level net structure of M-nets is less general than that of coloured nets. Besides annotations on places (sets of allowed tokens), arcs (multisets of variables) and transitions (occurrence conditions), M-nets have, additionally, labels on places and labels on transitions. The labels on transitions denote communication capabilities similar to action symbols in a CCS term. The labels on places denote their status as 'entry' (part of the initial marking), 'exit' (part of the final marking) or 'internal' (otherwise). Thus, every place and transition of an M-net carries an inscription of the form

$$\textit{inscription} \;=\; (\,\textit{label} \mid \textit{annotation}\,).$$

Labels drive composition operations and annotations drive unfolding.

2.1. Basic definitions

2.1.1. Multisets

Let Z be a set. A multiset over Z is a function $\mu: Z \to \mathbf{N}$; μ is finite if $\{a \in Z \mid \mu(a) > 0\}$ is finite. We denote by $\mathcal{M}_f(Z)$ the set of finite multisets over Z. The operations of sum and difference are defined on multisets as follows:

$$\begin{array}{lrcl} \text{Sum:} & (\mu_1 + \mu_2)(a) & = & \mu_1(a) + \mu_2(a) \\ \text{Difference:} & (\mu_1 - \mu_2)(a) & = & \max(\mu_1(a) - \mu_2(a), 0), \end{array}$$

for all $a \in Z$. Note that if μ_1 and μ_2 are finite then so are $\mu_1 + \mu_2$ and $\mu_1 - \mu_2$. Multiset inclusion is defined by $\mu_1 \subseteq \mu_2$ iff $\forall a \in Z: \mu_1(a) \leq \mu_2(a)$. The restriction $\mu\lceil_b$ (for $b \in Z$) is defined as the multiset

$$\mu\lceil_b(a) = \begin{cases} \mu(a) & \text{if } a = b \\ 0 & \text{if } a \neq b. \end{cases}$$

In our examples we use for multisets an extended set notation, e.g., we write $\{a, a, b\}$ for $\mu(a) = 2, \mu(b) = 1$, and $\mu(c) = 0$ for all $c \in Z \setminus \{a, b\}$.

2.1.2. Values, types, variables, terms and substitutions

Let *Val* be a fixed, nonempty, and suitably large, set of *values*. This set contains all the 'colours' of the high-level net model we define. In this paper, we assume that all integers, the Boolean values **false** and **true**, as well as the 'token' $\bullet$, are members of *Val*. A *type* is a subset of *Val*. The letters u, v, w towards the end of the alphabet will be used to denote fixed values in the set *Val*. We also need symbols to denote possibly changing values, i.e., variables. We normally use the small letters a, b, c near the front of the alphabet to denote them. Let the set *Var* be defined as the set of (symbols for) variables.

We will use standard definitions of substitutions and associated notions to describe the substitution of variables by values or, more generally, by terms. The set of all well-formed *terms* τ, built over the sets *Val*, *Var*, and a suitable set of operators *Op* (such as $+(5,7)$, $\mathbf{and}(a,\mathbf{true})$ or $\mathbf{eq}(a,5)$), is denoted by $\mathbf{T}$. For each term $\tau \in \mathbf{T}$ we denote by $var(\tau)$ the set of variables occurring freely in τ, and τ is called a *ground term* if $var(\tau) = \emptyset$.

A *substitution* is a mapping θ: $Var \to \mathbf{T}$ which differs from identity (id) for only a finite subset of *Var*. Let $a[\theta] = \theta(a)$ and let $dom(\theta)$ be the subset of *Var* where θ is different from identity: $dom(\theta) = \{a \in Var \mid a[\theta] \neq a\}$. If θ is a substitution with $dom(\theta) = \{a_1, \ldots, a_n\}$, then we denote θ by a list $(a_1/\tau_1, \ldots, a_n/\tau_n)$ which means that for $1 \leq i \leq n$, $a_i[\theta] = \tau_i$. Substitutions can be lifted from variables to terms in the usual way. For instance, substitution $\theta_0 = (a/\mathbf{false})$ can be applied as $+(5,7)[\theta_0] = +(5,7)$, $\mathbf{and}(a,\mathbf{true})[\theta_0] = \mathbf{and}(\mathbf{false},\mathbf{true})$ and as $\mathbf{eq}(a,5)[\theta_0] = \mathbf{eq}(\mathbf{false},5)$. A substitution can also be lifted in the standard way to sets or multisets or vectors of terms. For example, $\{+(5,7), \mathbf{and}(a,\mathbf{true})\}[\theta_0] = \{+(5,7), \mathbf{and}(\mathbf{false},\mathbf{true})\}$. Moreover, substitutions can be composed with each other: $\tau[\theta\theta'] = (\tau[\theta])[\theta']$.

A *renaming* is a substitution ρ: $Var \to Var$ which is bijective. If τ and τ' are two terms, we say that τ and τ' are *renaming equivalent*, denoted $\tau \approx \tau'$, if there are renamings ρ and ρ' such that $\tau[\rho] = \tau'[\rho']$. In this definition, the two bijective renamings could be replaced alternatively by a single non-bijective renaming $\rho'' = (a_1/b_1, \ldots, a_n/b_n)$ where all b_i are distinct variables. This relation can be extended canonically to sets, multisets and vectors of terms.

A *binding* is a substitution σ: $Var \to Var \cup Val$ which is, for each variable, either the identity, or a value from *Val*, i.e. $\forall a \in dom(\sigma)$: $a[\sigma] \in Val$. For each term τ, we say that σ is a binding of τ if $dom(\sigma) \supseteq var(\tau)$.

We assume that each operator $op \in Op$ of arity n determines a function f_{op}: $Val^n \rightarrow Val$. Then, the evaluation $eval_\sigma(\tau)$ of a term $\tau \in \mathbf{T}$ under a binding σ is defined as follows:

$$eval_\sigma(\tau) = \begin{cases} \tau & \text{if } \tau \in Val \\ \tau[\sigma] & \text{if } \tau \in Var \\ f_{op}(eval_\sigma(\tau_1), \ldots, eval_\sigma(\tau_n)) & \text{if } \tau = op(\tau_1, \ldots, \tau_n). \end{cases}$$

A substitution θ is called *at least as general* as a substitution θ' if there exists a substitution θ_0 such that $\theta' = \theta\theta_0$. A substitution θ is called a *unifier* of two terms τ_1 and τ_2 if $\tau_1[\theta] = \tau_2[\theta]$. A unifier θ_0 is said to be a *most general unifier* of τ_1 and τ_2 if every unifier θ of τ_1 and τ_2 equals $\theta_0\theta'$ for some substitution θ'. If two terms τ_1 and τ_2 are unifiable, then most general unifiers exist and they are all equal up to renaming. The corresponding class of substitutions is denoted by $mgu(\tau_1, \tau_2)$ and if θ is a most general unifier of τ_1 and τ_2, it belongs to $mgu(\tau_1, \tau_2)$. This notion can again be extended canonically to multisets and vectors of terms.

2.1.3. Action symbols and action terms

The framework defined in this section generalises, in a particular way, the basic structure of actions of Milner's CCS [29, 30]. We assume the existence of a fixed but sufficiently large set $\mathbf{A}$ of *action symbols* (sometimes also called *actions*, for short). We use capital letters A, B etc. to denote action symbols, so as not to confuse them with variables a, b etc. Each action symbol $A \in \mathbf{A}$ is assumed to have an arity $ar(A)$, which, by definition, is a natural number describing the number of its parameters. The set $\mathbf{A}$ is, by definition, the carrier of a bijection $\bar{\ }: \mathbf{A} \rightarrow \mathbf{A}$, called *conjugation*, satisfying

$$\forall A \in \mathbf{A}: \;\; \overline{A} \neq A \text{ and } \overline{\overline{A}} = A \text{ and } ar(\overline{A}) = ar(A).$$

That is, the mapping $\bar{\ }$ groups the elements of $\mathbf{A}$ into pairwise conjugates with the same arities. An *action term* is, by definition, a construct

$$A(\tau_1, \ldots, \tau_{ar(A)}),$$

where A is an action symbol and $\tau_j \in Var \cup Val$ for all $1 \leq j \leq ar(A)$. That τ_j may only be a value or a variable is not a true restriction, as we will explain later (at the end of section 2.3.2).

Let $\mathbf{AT}$ denote the set of all action terms. For instance, for $A, B, C \in \mathbf{A}$ and $ar(A)=2$, $ar(B)=1$, $ar(C)=0$, the following are action terms: $A(a, b)$,

$A(a,7)$, $A(5,7)$, $B(a)$, $B(7)$, and C. If an action term contains no variables of *Var* then it is called an *elementary action*. Of the above examples, $A(5,7)$, $B(7)$ and C are elementary actions while the others are not. Suppose that σ is a binding of all terms $\tau_1, \dots \tau_{ar(A)}$. Then the evaluation of $A(\tau_1, \dots, \tau_{ar(A)})$ under σ, by definition, yields an elementary action

$$eval_\sigma(A(\tau_1, \dots, \tau_{ar(A)})) = A(\tau_1, \dots, \tau_{ar(A)})[\sigma] = A(\tau_1[\sigma], \dots, \tau_{ar(A)}[\sigma]).$$

2.2. M-nets

2.2.1. Definition of M-nets

We generalise the (S,T,W) definition of nets [35] where

$$W\colon ((S\times T)\cup(T\times S)) \to \mathbf{N}$$

is a weight function (which yields 0 iff there is no connection).

Definition 1. *M-nets*

An *M-net* is a triple (S,T,ι) such that S is a set of *places*, T is a set of *transitions* with $S \cap T = \emptyset$, and ι is a function with domain $S \cup (S \times T) \cup (T \times S) \cup T$ (called *inscription*) such that:

- For every place $s \in S$, $\iota(s)$ is a pair $(\lambda_s \mid \alpha_s)$, where $\alpha_s \subseteq Val$ is a nonempty set called the *(place-)annotation* or the *type* of s and $\lambda_s\colon \alpha_s \to \{\mathbf{e}, \mathbf{i}, \mathbf{x}\}$ is a function called the *(place-)label*.
- For every arc $(s,t) \in (S\times T)$: $\iota(s,t) \in \mathcal{M}_f(Var \cup (Val \cap \alpha_s))$ (idem for arcs $(t,s) \in (T\times S)$), i.e., its inscription is a finite multiset of variables and values.
- For every transition $t \in T$, $\iota(t)$ is a pair $(\lambda_t \mid \alpha_t)$, where λ_t is a finite multiset of action terms called its *label* and α_t is a finite multiset of terms called its *annotation*. $\diamond$

Considering only the annotations rather than the labels, the differences to coloured nets are as follows. Place annotations are like the colour sets of [23]. Transition annotations give rise to legal and enabling bindings (as explained below), which correspond to the transition colour sets of [23]. Arc inscriptions are multisets of variables or values, and thus less expressive than the functions definable for coloured nets; in particular, M-nets – for the moment[1] – do not allow a transition to occur in some mode σ_1, taking a number of tokens from a certain subset of its input

[1] It may be feasible and desirable, in some future work, to relax this condition.

places, and to occur in a different mode σ_2, taking a different number of tokens from a different subset of its input places (compare also [9, 39] where this general occurrence is described in terms of linear algebra and unfolding, and [36, 40, 37] where it is disallowed for algebraic high-level nets).

We explain the three kinds of inscriptions in turn, beginning with arc inscriptions. In the explanation, we anticipate the intended transition rule semantics, which allows a transition t to occur with respect to a particular binding, σ, of variables. The arc inscriptions specify the variables and constants by which tokens flow; more formally, when tokens flow along an arc, they become bound to the variables on that arc. Thus the cardinality of the multiset $\iota(s,t)$ corresponds to the weight $W(s,t)$ in the usual definition of weighted place/transition nets. In particular, an arc inscription may be the empty multiset $\emptyset$. This signifies that no tokens may flow along that arc and that no effective connection exists along it. Arcs with annotation $\emptyset$ will not be represented graphically.

Each place s of an M-net is annotated by a pair $(\lambda_s \mid \alpha_s)$ where α_s is a subset of *Val* which delimits the set of tokens permissible on s, and λ_s is a mapping from α_s to $\{\mathbf{e}, \mathbf{i}, \mathbf{x}\}$ which describes the 'status' (entry, internal or exit, respectively) of a value in α_s. The status is important for certain composition operations; for example, to combine two M-nets in sequence we will combine the $\mathbf{x}$ values of places of the first with the $\mathbf{e}$ values of places of the second. It will often be the case that λ_s is one of the three possible constant functions, mapping every element of α_s to $\mathbf{e}$, $\mathbf{i}$ or $\mathbf{x}$. In that case we use the notation $\mathbf{e}$, $\mathbf{i}$ or $\mathbf{x}$, respectively, for λ_s. The reason we allow non-constant functions is to be conceptually consistent with the net unfolding operation. When a high-level net is unfolded, we cannot retrieve the places, but we can retrieve the place values, and thus λ_s should be defined in a consistent, unfolding-respecting, way as a function on place values rather than on places.

The first part, λ_t, of the inscription of a transition t is a multiset of action terms from $\mathbf{AT}$. A transition t can occur only under some binding σ which allows every term in that multiset to be evaluated; then $\lambda_t[\sigma]$ yields a multiset of elementary actions. The second part, $\alpha_t = \{\tau_1, \ldots, \tau_n\}$, which is a multiset of terms, plays the role of an *occurrence condition* or a *guard*, in the sense that t may occur only if the binding σ in question yields the evaluation **true** for all of the terms τ_i, i.e., if $eval_\sigma(\alpha_t)$ is a multiset over $\{\mathbf{true}\}$. Thus, the empty guard $\emptyset$ corresponds to a condition which is always satisfied, as do $\{\mathbf{true}\}$, $\{\mathbf{true}, \mathbf{true}\}$, etc.

Given a transition $t \in T$, the immediate environment of t in N, i.e. the transition as well as all its arcs, will be called the *area* of t:

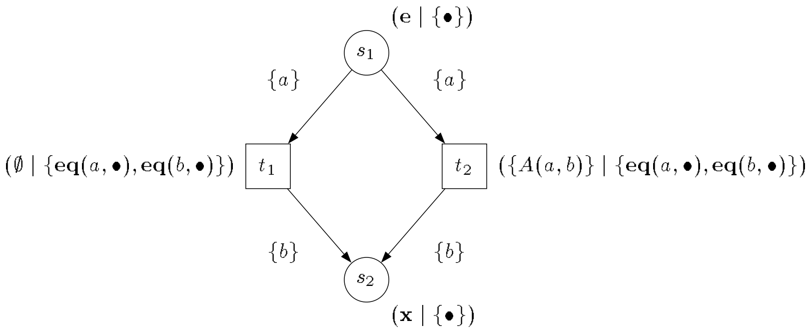


Fig. 1. An M-net.

$$area(t) \;=\; \{(s,t) \mid s \in S \wedge \iota(s,t) \neq \emptyset\} \cup \{t\} \cup \{(t,s) \mid s \in S \wedge \iota(t,s) \neq \emptyset\}.$$

Note that areas of different transitions are disjoint. As we will see when the behaviour of M-nets are defined, the area of a transition is the domain of relevance of variables occurring in inscriptions. This means that the name of a variable from the inscription of a transition t or an arc bordering on t is relevant only inside the area of t. The set of all variables occurring in the inscriptions of an area $area(t)$ will be denoted by $var(t)$. The set of all variables which belong to $var(t)$ but do not appear in an inscription of any adjacent arc of t are denoted by $free(t)$, i.e.,

$$free(t) \;=\; \{a \in var(t) \mid \forall s \in S\colon \iota(s,t)(a) = 0 = \iota(t,s)(a)\}.$$

Figure 1 shows an M-net. Consider the place s_1; its inscription $(\mathbf{e}|\{\bullet\})$ says that the value set associated with it is a singleton consisting only of the token $\bullet$ and the function λ_{s_1} is the constant function mapping $\bullet$ to $\mathbf{e}$. Consider the transition t_2 in the same net. Its inscription $(\{A(a,b)\}|\{\mathbf{eq}(a,\bullet), \mathbf{eq}(b,\bullet)\})$ says that its only action term is $A(a,b)$ and that it has two terms using the equality operator $\mathbf{eq}$ in its guard. Note that $var(t_2) = \{a,b\}$ and $free(t_2) = \emptyset$.

2.2.2. Bindings

In this section, we define the concept of a legal and enabling binding of a transition and some consequent concepts which underly the notion of occurrence of an M-net transition.

Definition 2. *Legal and enabling binding of a transition*

Let $N = (S, T, \iota)$ be an M-net and let $t \in T$ be a transition of N. Let σ be a binding of t, i.e., by definition, $dom(\sigma) \supseteq var(t)$. Then σ is called *legal* for t if

$$\forall s \in S \ \forall a \in (\iota(s,t) \cup \iota(t,s)) : a[\sigma] \in \alpha_s,$$

and *enabling* for t if it is legal and $eval_\sigma(\alpha_t)$ is a multiset over $\{\mathbf{true}\}$. $\diamond$

The legality requirement means that for all variables that occur on arcs between some place s and t, σ has to respect the type α_s of s, also all constants have to respect the type of s. In the following, we assume that all bindings considered are legal. Moreover, we do not distinguish two bindings σ and σ' for t if they only differ for variables not in $var(t)$.

Let σ be a binding which is legal for t. Then σ associates a value $a[\sigma]$ with every variable a in $var(t)$. Moreover, σ can be lifted to vectors and (multi-)sets of terms in the manner specified in sections 2.1.2 and 2.1.3. Similarly, let $A(\tau_1, \ldots, \tau_n)$ be an action term; then σ determines an elementary action $A(\tau_1, \ldots, \tau_n)[\sigma]$. It follows that if σ is a legal binding of t then inside the area of t we have:

— For every place s, $eval_\sigma(\iota(s,t)) = \iota(s,t)[\sigma]$ and $eval_\sigma(\iota(t,s)) = \iota(t,s)[\sigma]$ are well-defined multisets of values.
— $eval_\sigma(\lambda_t) = \lambda_t[\sigma]$ is a well-defined multiset of elementary actions.
— $eval_\sigma(\alpha_t)$ is a well-defined multiset of values (but in general $eval_\sigma(\alpha_t) \neq \alpha_t[\sigma]$, as the latter may be ground terms).

The requirement that σ is enabling for t means that the guard of t evaluates to a (possibly empty) multiset over **true**. In particular, guards such as, e.g., $\{+(a,5)\}$ have no enabling binding whatsoever.

As an example, consider the bindings $\sigma_1 = (a/3, b/\bullet)$, and $\sigma_2 = (a/\bullet, b/\bullet)$ of figure 1. The first binding is legal neither for t_1 nor for t_2, because the value of a does not match the value set of place s_1. The second binding is legal as well as enabling for t_1 and t_2. Consider, however, the M-nets in figure 2 and the binding $\sigma_3 = (a/1, b/4)$. Then σ_3 is legal but not enabling for the transition of figure 2(i) (using the usual interpretation of **eq** and $+$).

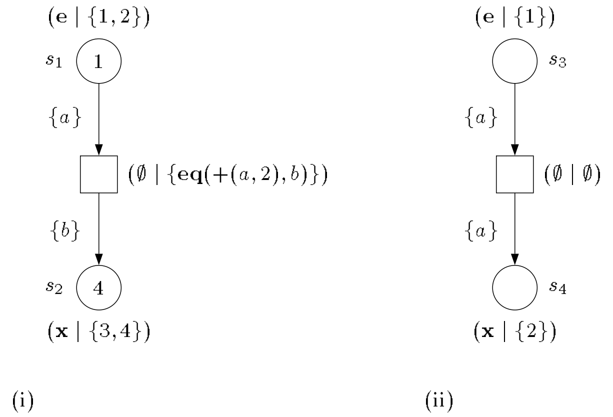


Fig. 2. Two non-elementary M-nets.

2.2.3. Elementary M-nets

Definition 3. *Elementary M-nets*
We call an M-net (S, T, ι) *elementary* if and only if

$\forall s \in S$: α_s is a singleton set.
$\forall t \in T$: λ_t is a multiset of elementary actions, α_t is a multiset over $\{\mathbf{true}\}$, and there exists a legal binding for t. $\diamond$

Note that a binding such as in the last sentence of this definition is always enabling. As usual, the exact identities of values and bindings don't matter, so that we may allow arbitrary singleton sets (rather than just the set $\{\bullet\}$) as place value sets of an elementary net. This makes sense, because the elementary net in question may have arisen by unfolding from a high-level net in which the black token $\bullet$ does not occur. The condition requiring a transition to have some legal and enabling binding is not implied by the other conditions for elementary M-net. Consider, for instance, the example shown in figure 2(ii) where the transition has no enabling binding because binding a both to 1 and to 2 is impossible. However, in the context of elementary M-nets, if some legal and enabling binding for a transition t exists, then it is unique with respect to $var(t)$.

Elementary M-nets and labelled arc-weighted P/T-nets [35] are in straightforward correspondence with each other. To transform a labelled arc-weighted P/T-net into an elementary M-net, attach to every place an arbitrary singleton value set (e.g. $\iota(s)=\{\bullet\}$) and to every transition a guard $\emptyset$. For arcs (s,t), add a multiset containing as many (occurrences of) variables as specified by its weight, and do the same for arcs (t, s). The labels remain the same. Conversely, to obtain a labelled arc-weighted P/T-net from an elementary M-net, apply the converse construction (simply forgetting all value sets and replacing them by the singleton set $\{\bullet\}$ and replacing all arc annotations by their cardinalities – see also [9, 24, 35]).

2.3. Behaviour and unfolding

In this section we define the behaviour of M-nets. This depends, first of all, on the notion of a marking (section 2.3.1). Marking changes are governed by the transition rule (section 2.3.2). Finally, we define the unfolding of a marked M-net (section 2.3.3).

2.3.1. Markings of an M-net

Definition 4. *Marking of an M-net*
A *marking* M of an M-net (S, T, ι) is a function M from S to the set of multisets over *Val* such that for every $s \in S$, $M(s) \in \mathcal{M}_f(\alpha_s)$. $\diamond$

The number $(M(s))(v)$ describes the number of occurrences of the value v in the place s at the marking M. Note that M may assign the empty multiset to s, indicating no marking on s. With each M-net (S, T, ι), we can associate in a canonical way an initial and a final marking. They are called its *standard initial marking* M^{e} and its *standard final marking* $M^{\mathbf{x}}$ and defined as follows for all values $v \in$ *Val* and for all places $s \in S$:

$$(M^{\mathrm{e}}(s))(v) = \begin{cases} 1 & \text{if } \lambda_s(v) = \mathbf{e} \\ 0 & \text{otherwise} \end{cases} \qquad (M^{\mathbf{x}}(s))(v) = \begin{cases} 1 & \text{if } \lambda_s(v) = \mathbf{x} \\ 0 & \text{otherwise.} \end{cases}$$

The initial marking is the one with which the net's behaviour is started, while reaching the final marking (which may or may not be possible) indicates proper termination.

2.3.2. The transition rule

The transition rule specifies the circumstances under which a marking M_2 is reachable from a marking M_1. Let $N = (S, T, \iota)$ be an M-net and let t be a transition of N. A marking M *activates* t *under* σ (or *in mode* σ) if σ is legal and enabling for t and $\forall s \in S \colon \iota(s, t)[\sigma] \subseteq M(s)$.

Definition 5. *Transition rule*

The *transition relation* is a relation $\rightarrow \subseteq \mathcal{M}_N \times T \times \mathcal{B} \times \mathcal{M}_N$, where $\mathcal{M}_N$ denotes the set of all markings of N and $\mathcal{B}$ denotes the set of all bindings. A quadruple (M_1, t, σ, M_2) is defined to be in $\rightarrow$ iff M_1 activates t in mode σ and $\forall s \in S\colon M_2(s) = (M_1(s) - \iota(s,t)[\sigma]) + \iota(t,s)[\sigma]$. $\diamond$

If (M_1, t, σ, M_2) is in $\rightarrow$ then we write $M_1 \xrightarrow{t:\sigma} M_2$ and say that the occurrence of t under the binding σ (or in mode σ) leads from M_1 to M_2. All the usual basic dynamic concepts (occurrence sequences, set of reachable markings, etc.) follow from the transition rule.

In general, if ρ is a renaming and σ is a binding with domain $dom(\sigma) = V \subseteq Var$ then $\rho^{-1}\sigma$ is a binding with $dom(\rho^{-1}\sigma) = V[\rho]$; conversely, if σ' is a binding with domain $V[\rho]$ then $\rho\sigma'$ is an inversely corresponding binding with domain V. In particular, by this inverse transformation, if the variables in areas of transitions other than t are renamed, this has no influence on the transition rule for t; every binding that was legal for t before the renaming is also legal after the renaming, and conversely. Similarly, any legal binding for a transition t is also legal (modulo the corresponding renamings) for any transition that is renaming equivalent with t.

The transition rule (and also the unfolding rule below) is hence invariant over renamings, as long as they are consistently done within the area of a transition. For this reason, every M-net can be transformed into one in which the variable sets in different areas are disjoint. Another observation is that our definition of action terms might be generalised by allowing an arbitrary term τ (rather than just a variable or constant) as a parameter of an action term $A(\ldots,\tau,\ldots)$ belonging to λ_t. However, this is not a true generalisation, because such a term τ can be replaced by a fresh variable a and the term $\mathbf{eq}(a,\tau)$ be added to the multiset α_t. Hence, any $A(\ldots,\tau,\ldots)$ containing a τ which is not either a value or a variable can be brought into the form of an action term.

2.3.3. The unfolding rule

The unfolding rule associates an elementary M-net $U(N)$ with every M-net N, as well as a marking $U(M)$ of $U(N)$ with every marking M of N. Let $N = (S, T, \iota)$; then $U(N) = (S_U, T_U, \iota_U)$ is defined as follows (where s_v and t_σ are the new places and transitions):

- $S_U = \{s_v \mid s \in S \text{ and } v \in \iota(s)\}$, where for each $s_v \in S_U$: $\alpha_{s_v} = \{v\}$ and $\lambda_{s_v}(v) = \lambda_s(v)$.

– $T_U = \{t_\sigma \mid t \in T$ and σ is an enabling binding of $t\}$,
 where for each $t_\sigma \in T_U$: $\lambda_{t_\sigma} = \lambda_t[\sigma]$ and $\alpha_{t_\sigma} = eval_\sigma(\alpha_t)$.
– $\iota_U(s_v, t_\sigma) = (\iota(s,t)[\sigma])\lceil_v$ and $\iota_U(t_\sigma, s_v) = (\iota(t,s)[\sigma])\lceil_v$.

Clearly, $U(N)$ is an elementary M-net. Figure 3 shows the unfolding of the net shown in figure 2(i). In this figure, σ_1 is the binding mapping a to 1 and b to 3 and σ_2 is the binding mapping a to 2 and b to 4. Note, for instance, that there is no arc between transition t_{σ_1} and place s'_4 because the restriction of $\iota(t, s')[\sigma_1]$ to the value 4 evaluates to the empty multiset.

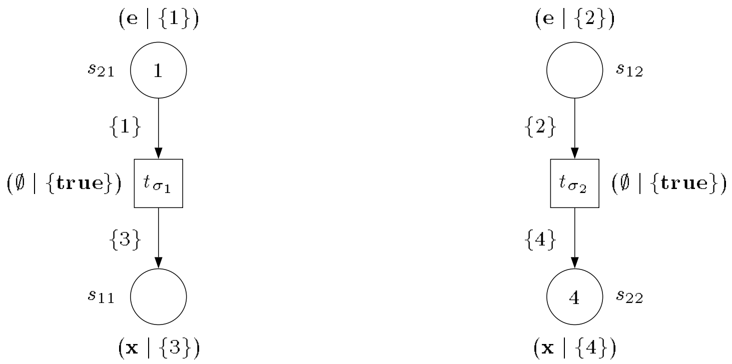


Fig. 3. Unfolding of figure 2(i).

Next, let M be any marking of N. We define a marking $U(M)$ of $U(N)$ as follows: for every place $s_v \in S_U$, $U(M)(s_v) = M(s)\lceil_v$. Thus, each elementary place $s_v \in S_U$ contains as many values v as the number of times this value occurs in the marking $M(s)$. The marking shown in figure 3 corresponds in this way to the marking shown in figure 2(i). The following lemma expresses the coherence of behaviour with respect to unfolding. An analogous lemma can also be found in [23].

Lemma 1. Unfolding and transition rules commute.

With notations as above and M_1, M_2 two markings of N, the following are equivalent:

$$\begin{array}{lll} \text{(i)} & M_1 \stackrel{t:\sigma}{\rightarrow} M_2 & \textit{in } N \\ \text{(ii)} & U(M_1) \stackrel{t_\sigma:id}{\longrightarrow} U(M_2) & \textit{in } U(N). \end{array}$$

Proof. First, we prove (i)⇒(ii). Let s_v be any place in $U(N)$. For the activation condition we have:

$$\begin{array}{rcl}
(\iota_U(s_v,t_\sigma))[id] & = & (\text{definition of } \iota_U \text{ and } id) \\
 & & (\iota(s,t)[\sigma])\lceil_v \\
 & \subseteq & (\text{by activation condition (i), applied to } v) \\
 & & (M_1(s))\lceil_v \\
 & = & (\text{definition of } U(M_1)) \\
 & & (U(M_1))(s_v).
\end{array}$$

For the calculation of the new marking we have:

$$\begin{array}{rcl}
(U(M_2))(s_v) & = & (\text{definition of } U(M_2)) \\
 & & (M_2(s))\lceil_v \\
 & \subseteq & (\text{marking calculation for (i), applied to } v) \\
 & & (M_1(s))\lceil_v - (\iota(s,t)[\sigma])\lceil_v + (\iota(t,s)[\sigma])\lceil_v \\
 & = & (\text{definition of } U(M_1) \text{ and } \iota_U \text{ and } id) \\
 & & ((U(M_1))(s_v) - (\iota_U(s_v,t_\sigma))[id]) + (\iota_U(t_\sigma,s_v))[id].
\end{array}$$

To prove (ii)$\Rightarrow$(i), the above calculation can be reversed and done for every value v. ■

2.3.4. Isomorphism of M-nets

Two marked M-nets are called isomorphic iff there are marking-preserving, inscription-preserving and arc-annotation-preserving bijections between their places and their transitions. Let $LE(t)$ denote the set of legal and enabling bindings of transition t.

The nets (S, T, ι, M) and (S', T', ι', M') are called *isomorphic* iff there exists

- a bijection $\beta : S \cup T \to S' \cup T'$ (with $\beta(S) \subseteq S'$ and $\beta(T) \subseteq T'$),
- a bijection $\beta_s : \alpha(s) \to \alpha(\beta(s))$ for every $s \in S$, and its canonical (componentwise) extension $\tilde{\beta}_s$ on transition labels and arc annotations,
- a bijection $\beta_t : LE(t) \to LE(\beta(t))$ for every $t \in T$, satisfying

(1) $\forall s \in S,\ \forall v \in \alpha(s)$: $\lambda_s(v) = \lambda_{\beta(s)}(\beta_s(v))$ (place-label-preservation)

(2) $\forall t \in T,\ \forall \sigma \in LE(t) :\ \tilde{\beta}_s(\lambda_t[\sigma]) = \lambda_{\beta(t)}[\beta_t(\sigma)]$ (transition-label-preservation)

(3) $\forall s \in S,\ \forall t \in T,\ \forall \sigma \in LE(t)$: $\tilde{\beta}_s(\iota(s,t)[\sigma]) = \iota(\beta(s), \beta(t))[\beta_t(\sigma)]$, and the same for (t, s) (arc-annotation-preservation)

(4) $\forall s \in S, \forall v \in \alpha(s)$: $(M(s))(v) = (M'(\beta(s)))(\beta_s(v))$ (marking-preservation).

Lemma 2.
An elementary net is isomorphic to its unfolding.

Proof. Directly from the definitions. ■

We will call two M-nets *equal* (denoted by the symbol =) iff they are equal up to isomorphism. In the following we will prove net properties up to this kind of isomorphism.

2.3.5. Duplication equivalence of M-nets

The most frequently used one is the duplication equivalence, which allows to add/drop duplicate places and/or transitions (with the same label and the same arc inscriptions to successors/predecessors). In this article we address in particular an equivalence allowing identifications of transitions.

Definition 6. *Transition duplication*
Let $N = (S, T; \iota)$ be an M-net. Two transitions $t, t' \in T$ are said to *duplicate* each other (or to be *renaming equivalent*) in N, denoted by $t \approx t'$, iff their areas are renaming equivalent:

$$\lambda(area(t)) \approx \lambda(area(t')). \qquad \diamond$$

This definition may be extended to transitions belonging to different nets, if they have the same set of places, as it will occur in definition 7 and theorems 2 and 1.

The following definition introduce the $\equiv_T$-equivalence identifying M-nets which are identical modulo duplication of transitions.

Definition 7. *Duplication equivalence*
Two M-nets having the same set of places, $N_1 = (S, T_1; \iota_1)$ and $N_2 = (S, T_2; \iota_2)$, are said to be $T-$*equivalent*, denoted by $N_1 \equiv_T N_2$, iff the following conditions hold:

1. $\forall s \in S \;\; \iota_1(s) = \iota_2(s)$,
2. $\forall t_1 \in T_1 \;\; \exists t_2 \in T_2$ such that $t_2 \approx t_1$,
3. $\forall t_2 \in T_2 \;\; \exists t_1 \in T_1$ such that $t_1 \approx t_2$. $\diamond$

This $T-$equivalence can be extended by net-isomorphism.

2.4. Operations on M-nets

In this section we define composition operations on M-nets which correspond to the PBC operators proposed in [4]. These operations create a new (not necessarily elementary) M-net out of one, two, or three given M-nets. We consider operators of two different kinds: those concerning place interfaces and thus the flow of control (which comprise sequential composition, choice, parallel composition and iteration) and those concerning transition interfaces and thus capabilities for communication (comprising synchronisation and restriction). Operations of the first kind are defined in section 2.4.1; for the time being, and by analogy to [5, 6], we will assume a uniform place interface which will be denoted by *ex*-goodness. Operations of the second kind are defined in sections 2.4.2 and 2.4.3; such operations will be defined without assuming the requirement of *ex*-goodness.

2.4.1. Sequence, choice, parallel composition and iteration

These operations are defined in such a way that they conform to the intuition associated with their names. Sequential composition is a binary operator having two M-nets as arguments and creating a new M-net by identifying the **x** places of the first with the **e** places of the second, thus creating the effect that the second net's execution can start if and only if the executing of the first has ended. The choice operator identifies the two sets of **e** places, and similarly the two sets of **x** places, of its two operand nets. The parallel composition operator creates the disjoint union of its two operand nets. The iteration operator is ternary. Its first operand is the initialisation, its second operand is the body, and its third operand is the finalisation of the loop it describes.

Similar definitions of these operations for place/transition nets can be found, e.g., in [4, 11, 18, 31]. In order to abbreviate the technical part of the definitions, and since in our later application, we will encounter this situation only, we assume the control flow interfaces of the M-nets, which should be composable sequentially (by choice, in parallel, etc.), to be like those for Petri Boxes in the low level:

Definition 8. *ex-goodness*
An M-net N is called *ex-good* if

(i) Every place s has as label a constant function, i.e. s is either an *entry place* with λ_s=**e**, or an *exit place* with λ_s=**x**, or an *internal place* with λ_s=**i**.
(ii) There are at least one entry place and at least one exit place.
(iii) Every entry or exit place has a singleton type, i.e., $\alpha_s=\{\bullet\}$ for every s with $\lambda_s\neq\mathbf{i}$.[2]

[2] This is an arbitrary choice; we could have chosen any other element of *Val*.

(iv) An entry place has no incoming arcs, i.e., if λ_s=**e** then $\forall t \in T\colon \iota(t,s)=\emptyset$, and an exit place has no outgoing arcs, i.e., if λ_s=**x** then $\forall t \in T\colon \iota(s,t)=\emptyset$. $\diamond$

Thus the 'plain token' $\bullet$ gains a special significance in being the only value for an entry or exit place and also, in examples, for representing flows of control. As a convenience in diagrams, we will assume the default for an arc inscription to be $\{\bullet\}$, unless explicitly inscribed otherwise.

In this section all M-nets (N, N_1, N_2 and N_3) are assumed to be *ex*-good. We define ${}^\bullet N$ and $N^\bullet$ as the sets of, respectively, entry places and exit places of N. Moreover, we use the auxiliary net manipulation operators defined in Appendix A. Their informal description is as follows: iN creates a copy of N tagged by the integer i; $\otimes\{S_1,\ldots,S_n\}$ multiplies n sets of places; $N \oplus S'$ adds a new set of places S' to N and at the same time removes the old places from which S' has been constructed; and $N_1 \cup N_2$ creates the union of N_1 and N_2.

First, we define sequential composition:

$$N_1;N_2 \;\;=\;\; (1N_1 \cup 2N_2) \;\oplus\; \otimes\{(1N_1)^\bullet, {}^\bullet(2N_2)\}.$$

This definition should be read as follows. To obtain the net $N_1;N_2$, first create two disjoint copies $1N_1$ of N_1 and $2N_2$ of N_2. Then multiply ($\otimes$) the exit places of $1N_1$ pairwise with the entry places of $2N_2$. Finally, connect those new places into the net like the old ones, give them the label **i** and remove the old ones (all of this is achieved by $\oplus$).

The other constructions should be read in similar ways. Choice $\square$ is defined as follows:

$$N_1 \,\square\, N_2 \;\;=\;\; (1N_1 \cup 2N_2) \;\oplus\; \otimes\{{}^\bullet(1N_1), {}^\bullet(2N_2)\} \;\oplus\; \otimes\{(1N_1)^\bullet, (2N_2)^\bullet\}.$$

Parallel composition $\|$ is defined as independent juxtaposition:

$$N_1 \| N_2 \;\;=\;\; 1N_1 \cup 2N_2.$$

The iteration construct is $[N_1 * N_2 * N_3]$. It produces the effect of one execution of N_1, followed by zero or more executions of N_2, followed by one execution of N_3. The net construction used is informally shown in figure 4, where 1 and 2 are two copies of N_1 (the initialisation), 3, 4 are copies of N_2 (the body), and 5, 6 are copies of N_3 (the finalisation).

$$\begin{aligned}[N_1 * N_2 * N_3] \;\;=\;\; & (1N_1 \cup 2N_1 \cup 3N_2 \cup 4N_2 \cup 5N_3 \cup 6N_3) \\ & \oplus \otimes \{{}^\bullet(1N_1), {}^\bullet(2N_1)\} \\ & \oplus \otimes \{(1N_1)^\bullet, {}^\bullet(3N_2), (4N_2)^\bullet, {}^\bullet(5N_3)\} \\ & \oplus \otimes \{(2N_1)^\bullet, {}^\bullet(4N_2), (3N_2)^\bullet, {}^\bullet(6N_3)\} \\ & \oplus \otimes \{(5N_3)^\bullet, (6N_3)^\bullet\}.\end{aligned}$$

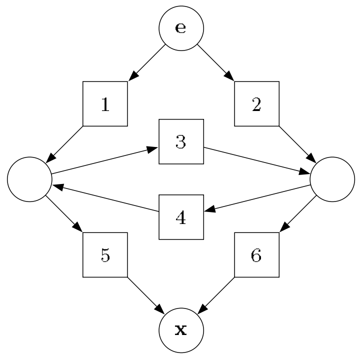


Fig. 4. Scheme for the definition of iteration.

The uniform token interface ensures that sequence, choice, parallel composition and iteration preserve *ex*-goodness.

2.4.2. Synchronisation

As this is the most complex and novel operation described in this paper, we explain it more fully than the previous operations. The intuitive idea behind the synchronisation operation of an M-net consists of a conglomeration of certain basic synchronisations. The operation will be defined as follows:

$$N \ \mathbf{sy} \ A,$$

where $N = (S, T, \iota)$ is a given M-net and A is a given action symbol. We first describe informally the basic synchronisation.

As an example, consider Figure 5. The upper part of the figure shows two transitions, t_1 and t_2 which contain conjugate action symbols in their labels: t_1 contains $A(.,.)$ (twice) and t_2 contains $\overline{A}(.,.)$ (once). This implies, by our definition, that t_1 and t_2 can be synchronised – as we shall say, 'over A' – and yield two new transitions, depending on which of the terms $A(.,.)$ is chosen from the label of t_1 to be matched with the term $\overline{A}(a,d)$ of the label of t_2.

Let us choose the first of the two, i.e., $A(a,b)$. Matching $A(a,b)$ with $\overline{A}(a,d)$ creates a new transition t_{12} from t_1 and t_2 in two steps, which are depicted in figure 5. First, the variables in the areas of t_1 and t_2 are substituted in order to make $var(t_1)$ and $var(t_2)$ disjoint. This is necessary because by synchronisation, the two areas of t_1 and t_2 are combined to a new single area. Then it is necessary, for instance, to make explicit the previously implicit distinction between the d belonging to t_1 and the d belonging to t_2.

Thus we consider two renamings ρ_1 (pertaining to t_1) and ρ_2 (pertaining to t_2). On the left-hand side of figure 5, a is substituted through ρ_1 by itself, a, and on the right-hand side of the figure a is substituted through ρ_2 by a'. The same happens with b and d: on the left they are substituted by b, d and on the right they are substituted by b', d'. The middle row of figure 5 shows the areas of the two transition after these renamings.

Next, we look for a unifier θ for the vectors (a, b) and (a', d') corresponding to the parameters of A and $\overline{A}$, respectively. We may take $\theta = (a/a', b/d')$. In the second step, a new transition (called t_{12}) is created. The label of t_{12} is (by definition) the multiset sum of the two constituent labels, substituted by θ, minus the matching pair of action terms. The annotation of t_{12} is the union of the two constituent annotations (modulo θ). The same happens with the arcs around t_{12}: they are the multi sets sums of the constituent arcs modulo θ. The last row of the figure shows t_{12} with its full annotation, but, for brevity, the annotations of t_1 and t_2 have been omitted (they are the same as in the top row of the figure).

Had we initially chosen the term $A(c, d)$ to be synchronised with the term $\overline{A}(a, d)$ of t_2, then there would have had to be different substitutions, and a different (not renaming equivalent) transition t'_{12} would have resulted. The transition t_{12} of the last part of figure 5 contains a term with A, and hence, it can again be synchronised with the transition t_2 which contains a term with $\overline{A}$ (this is what was meant by a conglomeration of the basic synchronisation). Since the transitions t_2 and t_{12} share an input place, the resulting arc inscription would then be a non-singleton multiset of variable names.

We split the formal definition into two parts: general synchronisation and basic synchronisation.

Definition 9. *Synchronisation*

Let $N = (S, T, \iota)$ be an M-net and A an action symbol. The net

$$N' \quad = \quad N \ \mathbf{sy}\ A \quad = \quad (S', T', \iota')$$

is defined as the smallest M-net satisfying:

(a) The set of places of N and N' (and their inscriptions) are the same: $S = S'$.

(b) Every transition of N (and its set of surrounding arcs) is also in N', with the same inscriptions as in N.

(c) If t_1 is a transition of N and t_2 a transition of N', such that one of them contains an action term with A in its label and the other one an action term with $\overline{A}$, then any transition t arising through a basic

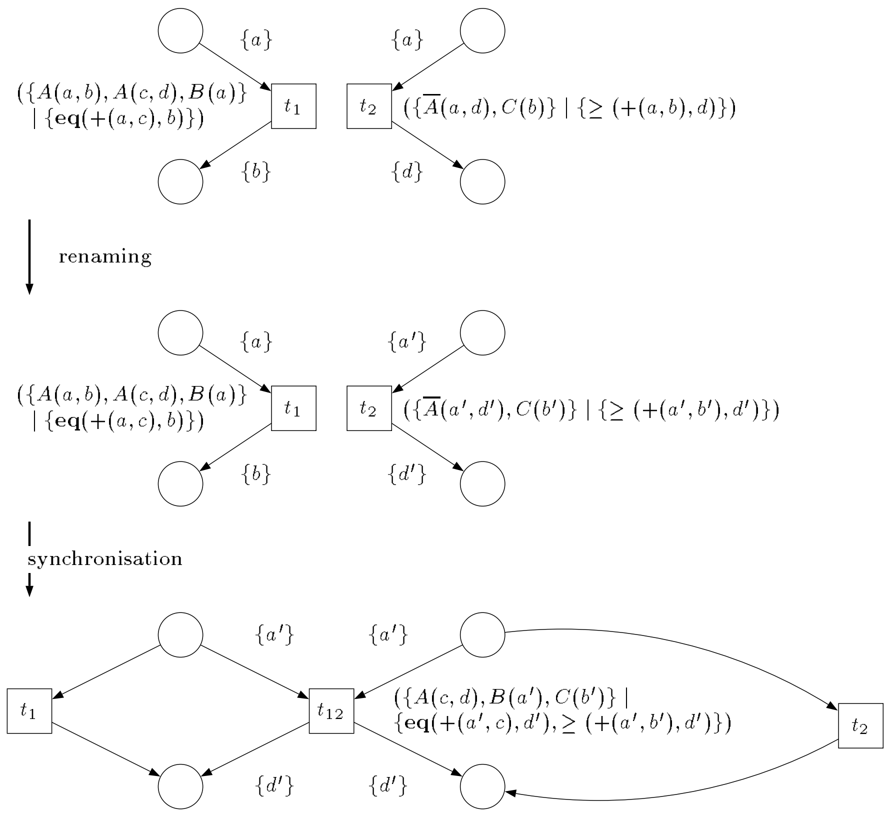


Fig. 5. Explanation of the basic synchronisation in creating N **sy** A.

synchronisation out of t_1 and t_2 over A (and its surrounding arcs) is also in N'. ◇

Note the asymmetry between t_1 and t_2 in step (c). This makes the definition more amenable to an induction argument. Later, at the end of this section, we show that the definition could have been phrased symmetrically without change of generality. The symmetric definition is essential for the proof of commutativity while the asymmetrical one is useful in the inductive proof of coherence of unfolding. Note also that in step (c) it may be the case that $t_1 = t_2$, namely when the label of this transition contains both a term with A and a term with $\overline{A}$.

Definition 10. *Basic synchronisation of two transitions*

Let t_1 and t_2 be transitions of N and N', respectively, as above. A transition t (and its surrounding arcs) *arises out of* t_1 *and* t_2 *through a basic synchronisation over* A, if there is an action term $A(a_1, \ldots, a_m)$ in the label of t_1 and an action term $\overline{A}(b_1, \ldots, b_m)$ in the label of t_2 ($ar(A){=}m{=}ar(\overline{A})$), as well as two substitutions θ_1 and θ_2 which are decomposable with two renamings ρ_1, ρ_2 and a substitution θ as follows: $\theta_1 = \rho_1\theta$ and $\theta_2 = \rho_2\theta$, such that five conditions hold:

1. ρ_1, ρ_2 are renamings for the areas of t_1 and t_2 such that $var(t_1)[\rho_1] \cap var(t_2)[\rho_2] = \emptyset$.
2. θ is the most general unifier of parameters[3] of the two action terms in question:
   $(a_1, \ldots, a_m)[\rho_1\theta] = (b_1, \ldots, b_m)[\rho_2\theta]$, and such that:
   $$\begin{array}{l}\forall s \in S, \forall x \in Var \cap (\iota(s,t_1) + \iota(t_1,s)){:}\ \ \theta_1(x) \in Val \ \ \Rightarrow \ \ \theta_1(x) \in \alpha_s \\ \forall s \in S, \forall x \in Var \cap (\iota(s,t_2) + \iota(t_2,s)){:}\ \ \theta_2(x) \in Val \ \ \Rightarrow \ \ \theta_2(x) \in \alpha_s.\end{array}$$
3. The arc inscriptions around t satisfy the following:
   $$\begin{array}{rcl}\forall s \in S{:}\ \ \iota(s,t) & = & \iota(s,t_1)[\rho_1\theta] + \iota'(s,t_2)[\rho_2\theta] \\ \forall s \in S{:}\ \ \iota(t,s) & = & \iota(t_1,s)[\rho_1\theta] + \iota'(t_2,s)[\rho_2\theta].\end{array}$$
4. The label of t contains all parameterised actions from $\lambda_{t_1}[\theta_1]$ and from $\lambda_{t_2}[\theta_2]$ less the conjugate pair $A(a_1, \ldots, a_m)[\theta_1], \overline{A}(b_1, \ldots, b_m)[\theta_2]$:
   $$\lambda_t \ = \ (\lambda_{t_1}[\theta_1] + \lambda_{t_2}[\theta_2]) - \{A(a_1, \ldots, a_m)[\theta_1], \overline{A}(b_1, \ldots, b_m)[\theta_2]\}.$$
5. The annotation of t is the union of the annotations of t_1 and t_2 after variable substitutions:
   $$\alpha_t = \alpha_{t_1}[\theta_1] + \alpha_{t_2}[\theta_2].$$ ◇

[3] Recall, at this point, that parameters can be only variables or constants.

Note that, if such a pair of substitutions θ_1, θ_2 exists, it is unique up to renaming.

As mentioned before, definition 9 can be rephrased by changing item (c) into:

> (c)$'$ If t_1 and t_2 are transitions of N' such that one of them contains an action term with A and the other one with $\overline{A}$, then any transition t arising through a basic synchronisation out of t_1 and t_2 over A (and its surrounding arcs) is also in N'.

To switch from one definition to the other is possible by the following proposition.

Proposition 1. *The definitions* 9*, and* 9 *with* (c) *replaced by* (c)$'$*, are equivalent.*

Proof. Let us call (a)-(c) the asymmetrical definition, (a)-(c)$'$ the symmetrical definition. It is clear that every transition created by the asymmetrical definition can also be created by the symmetrical definition. Conversely, let t be created by the symmetrical definition. This means that there are two N'-transitions t' and t'' such that t arises by a basic synchronisation out of t' and t''. By an induction hypothesis, we may assume that t' and t'' satisfy the asymmetrical definition, i.e., that there are transitions t_1' and t_2' such that t_1' is in N (not N'!) and t_2' is in N' and t' arises from a basic synchronisation out of t_1' and t_2'; and similarly that t'' arises from a basic synchronisation out of t_1'' and t_2'', for some transitions t_1'' in N and t_2'' in N'.

In the language of [4], any conglomerate of such synchronisations forms a finite spanning tree in the graph of possible conjugate pair connections generated by the labels of all N-transitions involved in it, counting multiplicities properly. Any nontrivial spanning tree (i.e., one not just consisting of a single node) has at least two endpoint nodes that border on exactly one edge because otherwise it would contain a cycle. This means that t' and t'' have at least four such nodes, and no matter how t' and t'' are synchronised together by a basic step, at most two of them can disappear and in the resulting synchronised conglomerate still at least two such endpoints are available. Any one of these endpoints will do as a transition t_1 in N, and the rest as a transition t_2 in N' which, when synchronised together, yield t. ∎

As a final example (in this section), consider an M-net N with three transitions, t_1, t_2, t_3, such that the first has label $\{A\}$, the second has label $\{A\}$ and the third has label $\{\overline{A}, \overline{A}\}$. We explain some of the transitions of $N' = N$ **sy** A, using the asymmetric definition. By item (b) of the definition, all of t_1–t_3 are also in N'. By a basic synchronisation, $t_1 \in N$ and $t_3 \in N'$ give rise to a transition t_4 in N' with label $\{\overline{A}\}$. By a basic synchronisation again, $t_2 \in N$ and $t_4 \in N'$ give rise to a transition t_5 in N', with label $\emptyset$, which synchronises all three original transitions.

2.4.3. Restriction

Intuitively, the restriction operation N **rs** A removes from N all transitions that mention either A or $\overline{A}$, and hence all synchronisation capabilities for A.

Definition 11. *Restriction*
Given an M-net $N = (S, T, \iota)$ and an action symbol A, we define N **rs** A as the M-net $N' = (S', T', \iota')$ where

- $S' = S$.
- $T' = \{\, t \in T \mid \iota(t) = (\lambda_t \mid \alpha_t)$ and λ_t contains no action term with either A or $\overline{A}\,\}$.
- $\iota' = \iota$ restricted to S' and T'. $\diamond$

2.5. Coherence properties of the M-net algebra

This section states coherence results of the form: the unfolding of a composite object is the composition of its unfoldings.

2.5.1. Coherence of sequence, concurrent composition, choice and iteration

The next result states that unfolding commutes with the compositions of sequence, concurrent composition, choice and iteration introduced in the previous section.

Proposition 2. *Coherence of unfolding and composition*
Let N_1, N_2 and N_3 be ex-good M-nets. Then

$$\begin{aligned} U(N_1; N_2) &= U(N_1); U(N_2) \\ U(N_1 \| N_2) &= U(N_1) \| U(N_2) \\ U(N_1 \,\square\, N_2) &= U(N_1) \,\square\, U(N_2) \\ U([N_1 * N_2 * N_3]) &= [U(N_1) * U(N_2) * U(N_3)] \end{aligned}$$

Proof. Directly from the definitions, since by the ex-goodness properties, the interfaces of an ex-good M-net and its unfolding are the same. ∎

2.5.2. Coherence property of synchronisation

In this section we state and prove a result for synchronisation that is analogous to proposition 2. First, let us remark that $\{A, \overline{A}\}$, the pair of conjugate action symbols under consideration, is the same (with the same arity) in an M-net and in its unfolding. The only difference is that its parameter list may contain variables and constants in the former but only constants and no variables in the latter.

Theorem 1. *Coherence of unfolding and synchronisation*
Let N be an M-net and A an action symbol.
Then $U(N \textbf{ sy } A) \equiv_T U(N) \textbf{ sy } A$.

Proof. The sets of places of $U(N \textbf{ sy } A)$ and $U(N) \textbf{ sy } A$ are the same by definition. It needs to be shown that any transition of $U(N \textbf{ sy } A)$ is also a transition of $U(N) \textbf{ sy } A$ and conversely.

($\subseteq$): Let t_0 be a transition of $U(N \textbf{ sy } A)$.

By the definition of unfolding, there exist a legal and enabling binding σ and a transition t in $N \textbf{ sy } A$ such that $t_0 = t_\sigma$.

By proposition 1 and definition 10, there exist two transitions t' in N and t'' in $N \textbf{ sy } A$, as well as two substitutions $\theta_1 = \rho_1\theta$ and $\theta_2 = \rho_2\theta$, such that t arises out of t' and t'' through a basic synchronisation.

Define $\sigma' = \theta_1\sigma$ and $\sigma'' = \theta_2\sigma$; then σ' and σ'' are legal and enabling for t' and t'', respectively. Define $t'_0 = t'_{\sigma'}$ and $t''_0 = t''_{\sigma''}$. Then t'_0 is a transition of $U(N)$ and t''_0 is a transition of $U(N \textbf{ sy } A)$. Moreover, t_0 arises from t'_0 and t''_0 through a basic synchronisation. Hence t_0 is a transition of $U(N) \textbf{ sy } A$.

($\supseteq$): Let t_0 be a transition of $U(N) \textbf{ sy } A$. Then by proposition 1, t_0 arises from a transition t_1 of $U(N)$ and a transition t_2 of $U(N) \textbf{ sy } A$ by a basic synchronisation. We proceed by induction on the size of the synchronisation (spanning) tree of t_0.

Base: t_2 is a transition of $U(N)$. Since $U(N)$ is a subnet of $U(N) \textbf{ sy } A$, this case is analogous to the induction step.

Step: t_2 is a transition of $U(N) \textbf{ sy } A$ and we may assume that t_2 is also a transition of $U(N \textbf{ sy } A)$.

By the definition of unfolding, there exist two legal bindings σ_1 and σ_2, and two transitions t_1^h of N and t_2^h of $N \textbf{ sy } A$, such that the unfolding of t_1^h under σ_1 and the unfolding of t_2^h under σ_2 equals t_2. We assume σ_1 and σ_2 chosen in such a way that the sets of variables on which σ_1 and σ_2 differ from identity are disjoint. This is always possible since we may choose two separating renamings ρ_1 and ρ_2 for the areas of t_1^h and t_2^h.

By definition 10, there exists a transition t in N **sy** A which arises out of t_1^h and t_2^h through the basic synchronisation with substitutions $\theta_1 = \rho_1\theta$ and $\theta_2 = \rho_2\theta$, where θ is a substitution as in definition 10. Since σ_1 and σ_2 are also substitutions, we can write $\sigma_1 = \rho_1\theta\xi_1$ and $\sigma_2 = \rho_2\theta\xi_2$. Because ρ_1 and ρ_2 are separating for the areas of t_1^h and t_2^h, and θ, ξ_1 and ξ_2 commute.

By unfolding with the legal and enabling binding $\sigma = \theta\xi_1\xi_2$, we find a transition t_σ in $U(N$ **sy** $A)$, such that t_σ is the unfolding of t under σ. But t_σ is exactly t_0, whence t_0 is in $U(N$ **sy** $A)$. ■

2.5.3. Coherence property of restriction

An analogous result can be proved with less effort for the restriction operation.

Proposition 3. *Coherence of unfolding and restriction*
Let N be an M-net and A an action symbol.
Then $U(N \textbf{ rs } A) = U(N) \textbf{ rs } A$.

Proof. Directly from the definitions of the restriction and the unfolding of an M-net. ■

2.6. Algebraic Properties

There are a number of commutativity, associativity and idempotence properties of the M-net operations. For elementary nets, except for the property of commutativity of synchronisation, these are well-known results (see e.g. [4, 11, 18, 31]).

Proposition 4. *Associativity/commutativity*
Let N_1, N_2 and N_3 be ex-good M-nets. Then

$$
\begin{aligned}
N_1; (N_2; N_3) &= (N_1; N_2); N_3 \\
N_1 \| (N_2 \| N_3) &= (N_1 \| N_2) \| N_3 \\
N_1 \square (N_2 \square N_3) &= (N_1 \square N_2) \square N_3 \\
N_1 \| N_2 &= N_2 \| N_1 \\
N_1 \square N_2 &= N_2 \square N_1
\end{aligned}
$$

Proof. Mutatis mutandis, by [4, 11, 18, 31]. ■

Proposition 5. *Commutativity and idempotence of restriction*
Let N be an M-net and A, B two action symbols. Then

$$
\begin{aligned}
(N \textbf{ rs } A) \textbf{ rs } B &= (N \textbf{ rs } B) \textbf{ rs } A. \\
(N \textbf{ rs } A) \textbf{ rs } A &= N \textbf{ rs } A.
\end{aligned}
$$

Proof. Immediate from the definition of restriction. ■

Theorem 2. *Commutativity and idempotence of synchronisation*
Let N be an M-net and A and B two action symbols. Then

$$\begin{array}{lcl} (N \ \mathbf{sy}\ A)\ \mathbf{sy}\ B & \equiv_T & (N\ \mathbf{sy}\ B)\ \mathbf{sy}\ A. \\ (N\ \mathbf{sy}\ A)\ \mathbf{sy}\ A & \equiv_T & N\ \mathbf{sy}\ A. \end{array}$$

Proof. We omit a proof of the first claim in this paper. There are two reasons for this. Firstly, using the synchronisation spanning trees defined in the proof of proposition 1, the detailed proof given in [4] for the commutativity of synchronisation at the elementary level can be transferred almost *verbatim.* Secondly, there is also a detailed proof of the commutativity of synchronisation of A-nets in [25], and M-nets are a class of A-nets with synchronisation in M-nets being a special case of synchronisation in A-nets.

The second claim is an immediate consequence of the definitions. ■

3. A Parallel Language and its M-net Semantics

$B(PN)^2$ (a basic Petri net programming notation, [8]) 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 via shared variables.

3.1. A motivating example

To start the discussion we consider a $B(PN)^2$ fragment which specifies a block with local variable X of type $\{0,1\}$ (i.e. its possible values) and global variable Y assumed to be declared of type $\{0,1\}$ in some outer block:

$$\begin{array}{ll} \mathbf{begin} & \mathbf{var}\, X : \{0,1\}; \\ & \langle X'{=}'Y \wedge Y'{=}'Y\rangle \parallel \langle X'{=}1{-}'X\rangle \\ \mathbf{end}. & \end{array}$$

The block body is a parallel composition of two atomic actions. An atomic action defines the conditions required for its execution, in terms of the pre- (e.g. $'X$) and post- (e.g. X') values of variables it accesses. The above two atomic actions thus correspond to $X := Y$ and $X := 1{-}X$, respectively.

Even though the above is not a closed program we still wish to give it an M-net semantics. In order to achieve this, we use an idea from [29, 30]. The locality of X and non-locality of Y must be reflected in some way in the M-net associated with the program. In the following we describe the steps for obtaining this M-net. In giving the M-nets we will use a different notation for the terms in transition inscriptions, using the infix form, for compatibility with the programming language notation, rather than the prefix form more appropriate for the formal definition of M-nets in the previous section. We will write for instance,

$$X'{=}'Y \wedge Y'{=}'Y \quad \text{instead of} \quad \mathbf{and}(\mathbf{eq}(X', 'Y), \mathbf{eq}(Y', 'Y)),$$
$$\text{and} \quad X'{=}1{-}'X \quad \text{instead of} \quad \mathbf{eq}(X', -(1, 'X)).$$

Note that the block has three basic constituents, namely a declaration and two atomic actions. The M-nets corresponding to these constituents can be given directly. Figure 6 shows an M-net describing the declaration of X and Figure 7 shows M-nets describing the two atomic actions.

3.1.1. Basic Interaction between Atomic Actions and Declarations

The essential part of the declaration M-net in Figure 6 is the central place and its connected transition t; for the moment we will ignore the rest of the M-net, which is to do with initialisation and termination. That central place, the X-value place, can hold a token of the same type as X, representing the current value of X when it has one. The transition t can come into play when X is accessed through some atomic action in the body of the block. Its arcs have inscriptions $\{X^i\}$ and $\{X^o\}$ where the superscripts i and o stand for 'input' and 'output', respectively. X^i and X^o should be considered to be net variables (such as a, b, c etc. as defined in section 2.1.2), which by their names make a connection to program variable X. The idea is that in binding X^i to some value, a token (on the X-value place) is consumed which describes the old value of X (which, as X^i borders on a place with the same type as X, must be a value from the value set of the declaration of X). Binding a value to X^o means that a value is produced which is the new value of X.

The transition uses action symbol $\overline{X}$ of arity 2, which is an action (in the sense of section 2.1.3) derived from program variable X. It is waiting to be synchronised with other transitions involving the conjugate actions $X(.,.)$, also of arity 2; these other transitions will be described next.

Consider Figure 7. For the transition t_2 of this figure, the guard $\{X^o{=}1 - X^i\}$ describes the semantics of the action $\langle X' = 1 - 'X\rangle$; in words, the output value of X is 1 minus the input value of X. For the

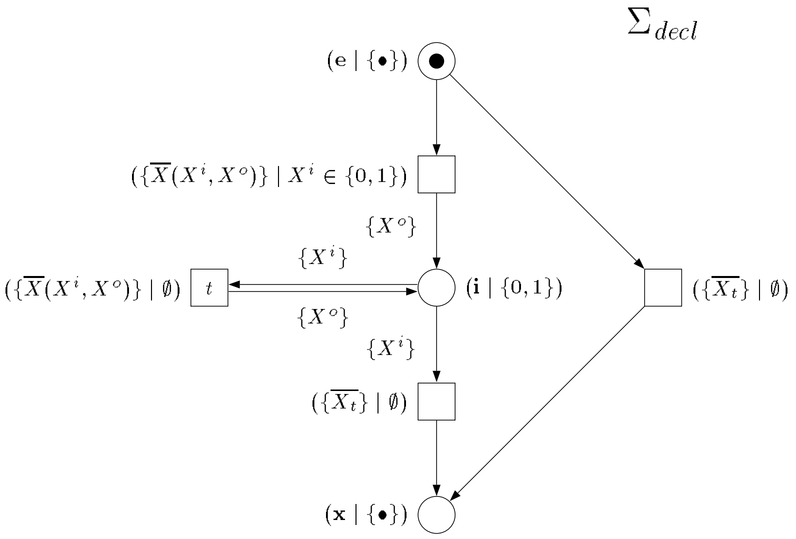


Fig. 6. An M-net corresponding to **var** $X : \{0,1\}$.

transition t_1 of this figure, the guard $\{X^o{=}Y^i \wedge Y^o{=}Y^i\}$ describes the semantics of action $\langle X'{=}'Y \wedge Y'{=}'Y\rangle$: in words, for that transition to occur, the output value of X, X^o, must equal the input value of Y, Y^i, and the latter must equal the output value of Y, Y^o, as well.

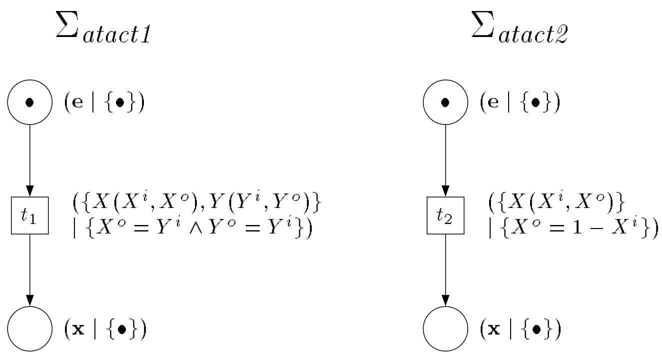


Fig. 7. Two M-nets corresponding to $\langle X'{=}'Y \wedge Y'{=}'Y\rangle$ and $\langle X'{=}1{-}'X\rangle$, resp.

Note that the transitions contain labels with action symbols $X(.,.)$ and $Y(.,.)$ of arity 2. The action symbols $X(.,.)$ represent the accesses to a variable X specified by the atomic action in question. When the block containing these atomic actions and the declaration of X is described then this involves synchronisations and restrictions of the transitions from the declaration (Figure 6), containing $\overline{X}(.,.)$, with transitions from the atomic actions (Figure 7), containing $X(.,.)$.

The synchronisation of t from the declaration and t_2 from the atomic action can produce (as described in more detail below) a transition which has:

(a) the guard $\{X^o{=}1-X^i\}$ (inherited from t_2);
(b) the empty label (resulting from the mutual cancellation of action names $X(.,.)$ and $\overline{X}(.,.)$ from t_2 and t respectively);
(c) arcs inscribed with X^i and X^o from and to the central X-value place of the declaration M-net (inherited from t);
(d) arcs from the pre-place and to the post-place of t_2 (inherited from t_2)

Any execution of this synchronised transition will achieve: the execution of the atomic action represented by t_2, as a result of (d); the change of the value on the X-value place (to 1 minus its previous value), as a result of the common variables X^o and X^i in (a) and (c).

Considering now the other atomic action, its transition t_1 can synchronise with t from the declaration to produce a transition which is the same as above except in:

(a) the guard is $\{X^o{=}Y^i \wedge Y^o{=}Y^i\}$, for the semantics of this atomic action;
(b) the label is $Y(.,.)$. Action X is cancelled out by the synchronisation, but action $Y(.,.)$ remains in the label. By these means, the locality of X and the globality of Y are described, respectively.

3.1.2. Initialisation and Termination

The M-net for the declaration (Figure 6) has, in addition to the central X-value place, an entry place which can hold tokens $\bullet$ and an exit place which can hold tokens $\bullet$. The entry place holds a token at the point where program control enters this block, and the exit place holds a token at the point where program control leaves that block. The transition connected to the entry and X-value places is the initialisation transition. It comes into play on the first (if any) access to the variable following each entry of control into the block (which may occur several times if the block is within a loop). The label and out-going arc inscription are the same as for t discussed above, allowing it to synchronise with an atomic action transition and establish an output value, X^o, for the X-value place. The input value X^i in the label is a notional initial value for X, which, by the guard, is allowed to be any value in X's type. For the synchronisation with

t_2 from Figure 7, the only possible bindings would be X^i=1 and X^o=0. The incoming arc simply removes the • token from the entry place.

The transition connected to the X-value place and the exit place is a termination transition which comes into play when control leaves the block, to effect loss of value for X; in this way, the net is well-behaved when, say, the block is enclosed within a loop. This behaviour is achieved by the label $\{\overline{X_t}\}$ which will synchronise with a transition, having label $\{X_t\}$, which is appended to the end of the M-net for the block body (in this case the parallel composition of the two M-nets in Figure 7). The final transition, connected to the entry and exit places, is another termination transition to allow for the possibility that control may leave the block without the variable ever being accessed.

Variable initialisation is the one area where we have considered alternative semantics to those given in this paper, in two respects. Firstly, one could have the guard in the initialisation transition as say X^i=0, to fix a definite initial value. This would be appropriate if the syntax allowed for explicit initialisations of variables. Secondly one could eliminate the label (i.e. replace it by $\emptyset$) from the initialisation transition, so that this would be a silent transition and the first access to the variable would be via transition t. That would have the advantage of simplicity, particularly in removing the need for the transition connected to the entry and exit places, but the disadvantage of introducing silent commitment for a block within a choice. For example, compare the following two constructs, where $commZ$ is synchronous communication on channel Z which must synchronise with conjugate communication $\overline{commZ}$,

$$((commC) \square (\textbf{begin var } X{:}\{0,1\}; commD; \ldots \textbf{end})) \parallel \overline{commC}$$

and

$$((commC) \square (commD; \textbf{begin var } X{:}\{0,1\}; \ldots \textbf{end}) \parallel \overline{commC}$$

If the declaration of X has a silent transition for its initialisation, then the apparently trivial change in the position of that declaration has a substantial effect on the overall behaviour in that the first construct has the possibility of deadlock (silently entering the block without being able to execute $commC \| \overline{commC}$ after that), whereas the second does not.

3.1.3. Complete Block Semantics

The overall semantics of the block in our running example is described as follows. Let D denote the declaration inside the block, i.e. $D = \textbf{var } X$:

$\{0,1\}$. At the end of the block, a termination net $\tau(D)$ with one transition labelled $\{X_t\}$ (i.e., the conjugate of the termination transitions in figure 6) is appended. Then, synchronisation and restriction are applied. That is, the semantics of the block is taken as:

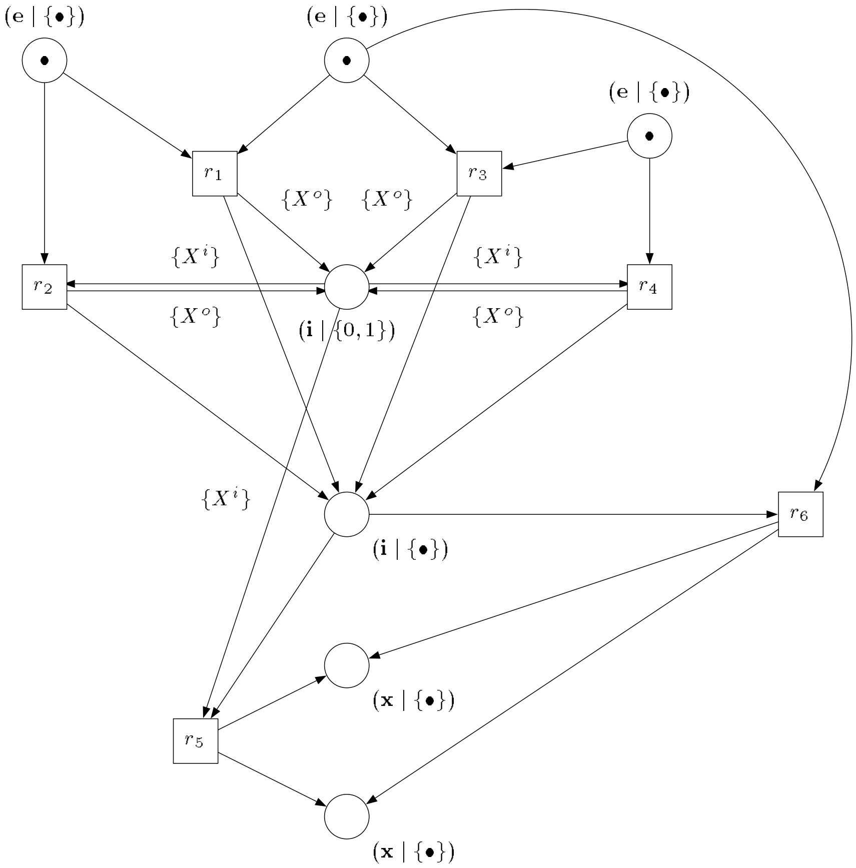
$$(\Sigma_{decl} \| ((\Sigma_{atact1} \| \Sigma_{atact2}); \tau(D))) \textbf{ sy } X \textbf{ rs } X, \tag{1}$$

where Σ_{decl}, Σ_{atact1} and Σ_{atact2} are the nets shown in figures 6 and 7, respectively. The net resulting from (1) is displayed in figure 8. Note that the value changes of X are now described explicitly by appropriate connections to the central X-value place which models X, rather than implicitly by the labels. The value changes of Y, however, are still described by labels. This label can now be synchronised with the corresponding conjugate label coming from the declaration of Y in an outer block, extending the construction until only $\emptyset$-labelled transitions remain in the net of program – provided all variables are declared properly in some enclosing block.

We explain briefly – referring back to the definition of synchronisation in section 2.4.2 – how the net shown in figure 8 arises as a result of the compound expression (1). Let us explain how the first transition r_2 shown in figure 8 arises by synchronisation from the transitions of (1). Consider transition t of figure 6 and transition t_1 of figure 7. They contain conjugate action terms $\overline{X}(X^i, X^o)$ and $X(X^i, X^o)$. When they are synchronised by the operator **sy** X, the two terms are first unified, that is, their parameter lists are considered the same (they are already the same, so there is a trivial renaming). Transition r_2 of figure 8 is, modulo this (trivial) renaming, the multiset sum of t and t_1 in all respects, except that the two action terms $\overline{X}(X^i, X^o)$ and $X(X^i, X^o)$, which were the cause of the synchronisation, are subtracted from the resulting label. Note that forming the multiset sum for the guards of t and t_1 has the effect of logical conjunction. The transition r_4 of figure 8 is a synchronisation of transitions t of figure 6 and t_2 of figure 7 according to the same rules. Note that by the subtraction of conjugate pairs, these two transitions survive the restriction **rs** X, but t, t_1 and t_2 do not. The other parts of figure 8 are constructed similarly; for instance, r_5 and r_6 arise from synchronising transitions labelled $\{\overline{X_t}\}$ of figure 6 with the termination transition from the net $\tau(D)$.

3.1.4. Unfolding and Behaviour

For the sake of completeness, we illustrate behaviour and unfolding on the running example. In figures 6 and 7, for instance, we have



where the inscriptions of transitions are as follows:

$$\begin{array}{lllll} \iota(r_1) & = & \iota(r_2) & = & (\{Y(Y^i,Y^o)\}|\{X^o{=}Y^i \wedge Y^o{=}Y^i\}) \\ \iota(r_3) & = & \iota(r_4) & = & (\emptyset|\{X^o{=}1{-}X^i\}) \\ \iota(r_5) & = & \iota(r_6) & = & (\emptyset|\emptyset). \end{array}$$

Fig. 8. Net arising from (1), i.e. $(\Sigma_{decl}\|((\Sigma_{atact1}\|\Sigma_{atact2});\tau(D)))$ **sy** X **rs** X.

$$\begin{array}{rrclrcl} & var(t) & = & \{X^i, X^o\}, & \mathit{free}(t) & = & \emptyset, \\ & var(t_1) & = & \{X^i, X^o, Y^i, Y^o\}, & \mathit{free}(t_1) & = & \{X^i, X^o, Y^i, Y^o\}, \\ \text{and} & var(t_2) & = & \{X^i, X^o\}, & \mathit{free}(t_2) & = & \{X^i, X^o\}. \end{array}$$

In the net shown on the left of figure 7, $(X^i/0, X^o/1, Y^i/1, Y^o/1)$ is a legal binding enabling the occurrence of t_1 under the shown marking. If t_1 occurs with this binding then the effect is to remove a token from its input place and to add a token to its output place. In the unfolded net (which is not shown in this paper), this occurrence corresponds to an execution of a transition labelled by the multiset $\{X(0,1), Y(1,1)\}$, because this is what the label of t_2 evaluates to under the given binding.

Figure 9 shows the unfolded M-net associated with the M-net in figure 6. The two middle places of figure 9 correspond to the two possible tokens on the middle place of Σ_{decl}, namely 0 on the left and 1 on the right.

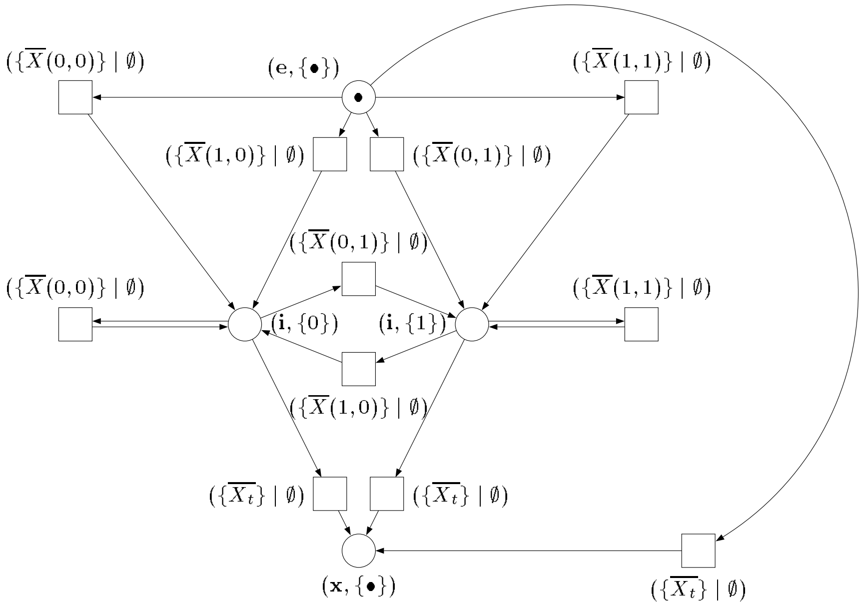


Fig. 9. Unfolding of the net of figure 6.

3.2. Syntax and semantics

In this section we present the syntax and both the high-level and low-level semantics of $B(PN)^2$ programs. We proceed top-down through the syntax, starting with program blocks (section 3.2.1) where the operations of synchronisation and restriction defined in sections 2.4.2 and 2.4.3 are instrumental; then declarations (whose nets are given directly, section 3.2.2); then command connectives for parallel composition, sequence, choice and iteration (section 3.2.3), where the operations defined in section 2.4.1 are instrumental; and finally atomic actions (whose nets are again given directly). Let P be any program construct of $B(PN)^2$. Then the high-level net and the low-level net associated with P are denoted by $Net_H(P)$ and $Net_L(P)$, respectively. All nets involved are ex-good, because the directly given nets are ex-good and all the net operations are ex-goodness preserving.

3.2.1. Programs and Blocks

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

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

A trailer B may consist of a (possibly empty) sequence of declarations followed by a command *Com*. For the syntax and semantics of *Com*, the reader is referred to section 3.2.3 (*Com* may itself be a compound structure of commands). A declaration involves a variable identifier W and the specification *set* of a type (e.g. $\mathbf{N}$ or $\{0,1,2\}$). Either ordinary variables or channel variables may be declared.

As has been illustrated in the previous section, the main idea in describing a block is to juxtapose the nets for its declarations and the net for its command (followed by a termination action), to synchronise all matching data / command transitions, and then to restrict them in order to make local variables invisible outside the block. The termination net associated with declaration D will be called $\tau(D)$ (see figure 10), and the set of all action symbols pertaining to W will be called $\delta(D)$: by definition, $\delta(D) = \{W, W_t\}$ if D is a declaration of the form **var** W : *set* and $\delta(D) = \{W_?, W_!, W_t\}$ if D is a declaration **var** W : **chan** b **of** *set*. Action $W(.,.)$ is of arity 2 and plays the same role as $X(.,.)$ in the example described in section 3.1. W_t is the termination action of arity 0; it has also been explained above. Actions $W_?$ and $W_!$ pertain to channel declarations. They are of arity 1 and will be described below.

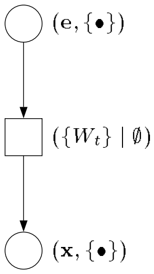


Fig. 10. Termination net $\tau(D)$ associated with a declaration $D = \textbf{var}\ W : \ldots$

Definition 12. *Semantics of blocks*
Using the above notation, for $I \in \{H, L\}$:

$$\begin{aligned} Net_I(\textbf{begin}\ B\ \textbf{end}) &= Net_I(B) \\ Net_I(D; B) &= \Big(\Big(Net_I(D) \parallel (\,Net_I(B); \tau(D)\,)\Big)\ \textbf{sy}\ \delta(D)\Big)\ \textbf{rs}\ \delta(D) \\ Net_L(D) &= U(Net_H(D)). \end{aligned}$$

◇

This definition must be completed by defining a parameterised class of declaration nets $Net_H(D)$ and nets $Net_I(Com)$ for a command Com. These definitions are given below.

Note that the low-level net for a declaration is defined via unfolding. For a more explicit representation of this net, compare figure 9, and [8] gives a the general inductive definition.

3.2.2. Declarations

Definition 13. *Semantics of ordinary variable declarations*
$Net_H(\textbf{var}\ W : set) = M_{data}(W, set)$, i.e. the net shown in figure 11.

◇

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, q, for the possible value changes. In addition, there are initialising and terminating transitions q_1, q_2 and q_3. The transitions q_2 and q_3 have to synchronise with the termination action τ associated to D via the net $\tau(D)$ according to Definition 12. The initialisation transition q_1 represents the first access to the variable, which – by $free(q_1)$=$\{W^i\}$ – may assume any pre-value of the variable.

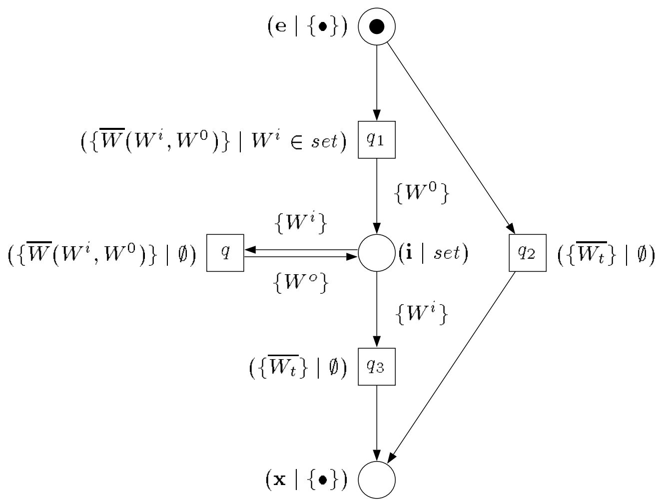


Fig. 11. The net $M_{data}(W, set)$ associated with a variable declaration.

The declaration **var** W : **chan** b **of** set declares a channel of capacity b which acts as a FIFO buffer of values supporting the operations of appending a value to the end of the buffer (send) and taking the oldest value from the buffer (receive). For $b = \omega$, the buffer has unbounded capacity. For $b = 0$, the buffer has zero capacity which gives handshake communication between send and receive of the same value.

The actions associated with a declaration of channel W, i.e. which occur in the transition labels of its net, are $\overline{W_t}$, $\overline{W_!}$ and $\overline{W_?}$. The $\overline{W_t}$ is for termination of the declaration, exactly as for termination of a variable declaration. The $\overline{W_!}$ and $\overline{W_?}$ will appear as $\overline{W_!}(W^!)$, for the sending of the value bound to net variable $W^!$, and $\overline{W_?}(W^?)$, for the receiving of the value bound to net variable $W^?$. The transitions carrying these labels are waiting to synchronise with transitions, carrying labels with the corresponding actions $W_!$ and $W_?$, which will be generated by atomic actions accessing the channel.

Definition 14. *Semantics of channel variable declarations*
$Net_H($**var** W : **chan** b **of** $set) = M_{chan}(W, b, set)$, as shown in figures 12 and 13. $\diamond$

In figure 12, 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 $lh(s)$.

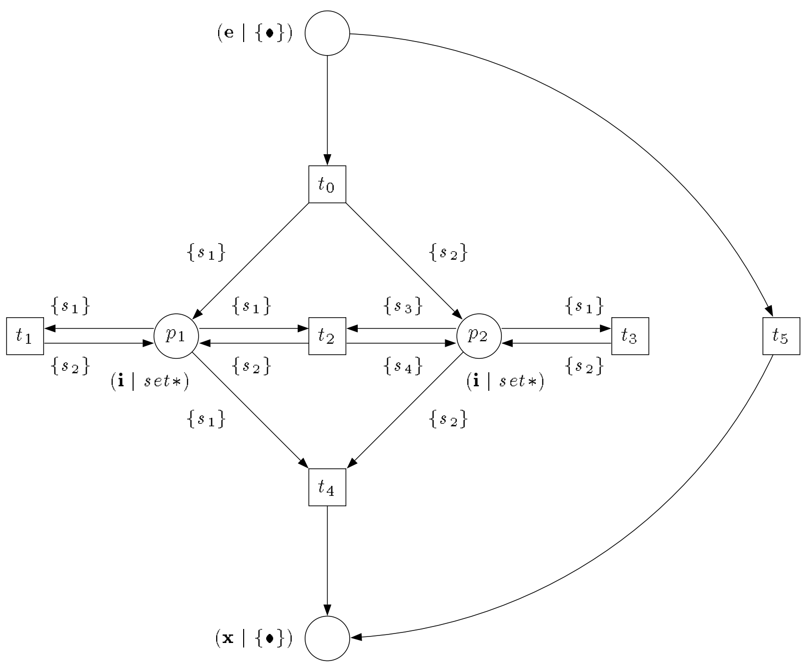
To enable concurrent send and receive actions, the contents of a channel is modelled by two sequences: that held on place p_2, either empty or the first element in the channel; and that held on place p_1, the rest of the channel (which is the whole channel if p_2 holds empty). The channel is initialised by the first sending of a value v to the channel. The initialisation transition t_0 places this value on p_1, and the empty sequence on p_2. Then there is one transition, t_1, for the general send action, appending the communicated value to the content of place p_1, 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 p_1. There is one transition, t_3, for the receive action, removing the communicated value from place p_2 and replacing it by ϵ. An internal transition, t_2, transfers the first entry of the sequence on place $P1$ to place p_2, if this contains ϵ. In addition, we have two terminating transitions, t_4 and t_5 which are analogous to those of an ordinary variable declaration (and which allow termination even when the channel is non-empty).

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 13(i). It can be seen as being obtained from $M_{chan}(W, b, set)$ (for $b > 1$) by merging the two places p_1 and p_2.

For the case $b = 0$ (i.e., in the case of handshake communication), the net of figure 13(i) is modified in the following way to give figure 13(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 ensures that both the communicated values coincide.

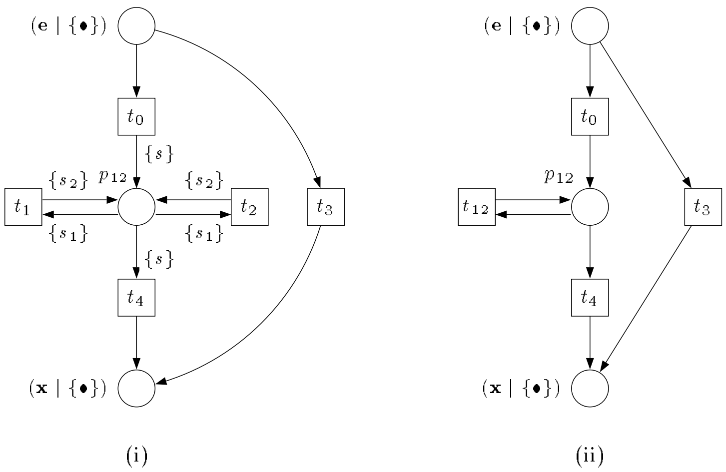
3.2.3. Command Connectives

The following is the syntax for a command, which can be: a block; an atomic action, *AtAct*, which is the basic unit of commands and is described in section 3.2.4; 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 subsumed in the **do**–**od** construct):



t_0: $\{\overline{W_!}(W^!)\} \mid \{s_1 = W^!, s_2 = \epsilon\}$
t_1: $(\{\overline{W_!}(W^!)\} \mid \{lh(s_2) < b, s_2 = s_1.W^!\})$
t_2: $(\emptyset \mid \{s_1 = v.s, lh(v) = 1, s_2 = s, s_3 = \epsilon, s_4 = v\})$
t_3: $(\{\overline{W_?}(W^?)\} \mid \{s_1 = W^?, s_2 = \epsilon\})$
t_4: $(\{\overline{W_t}\} \mid \emptyset)$
t_5: $(\{\overline{W_t}\} \mid \emptyset)$

Fig. 12. $M_{chan}(W, b, set)$, $b \neq 0$, $b \neq 1$.



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

$$\begin{array}{rl}
p_{12}: & (\mathbf{i} \mid \{s \in set* \mid lh(s) \leq 1\}) \\
t_0: & (\{\overline{W_!}(W^!)\} \mid \{s = W^!\}) \\
t_1: & (\{\overline{W_!}(W^!)\} \mid \{s_1 = \epsilon, s_2 = W^!\}) \\
t_2: & (\{\overline{W_?}(W^?)\} \mid \{s_1 = W^?, s_2 = \epsilon\}) \\
t_3: & (\{\overline{W_t}\} \mid \emptyset) \\
t_4: & (\{\overline{W_t}\} \mid \emptyset)
\end{array}$$

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

$$\begin{array}{rl}
p_{12}: & (\mathbf{i} \mid \{\bullet\}) \\
t_0: & (\{\overline{W_!}(W^!), \overline{W_?}(W^?)\} \\
 & \mid \{W^! = W^?, W^! \in set\}) \\
t_{12}: & (\{\overline{W_!}(W^!), \overline{W_?}(W^?)\} \\
 & \mid \{W^! = W^?, W^! \in set\}) \\
t_3: & (\{\overline{W_t}\} \mid \emptyset) \\
t_4: & (\{\overline{W_t}\} \mid \emptyset)
\end{array}$$

Fig. 13. (i) $M_{chan}(W,1,set)$, (ii) $M_{chan}(W,0,set)$.

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

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 execution of one clause of the exit-set. If a clause of the exit-set is chosen, termination of the whole construct ensues. Two examples are:[4]

1. $\mathbf{do}\ \langle N > 0\rangle; \langle Y' = {'Y} + N\rangle; \langle N' = {'N} - 1\rangle;\ \mathbf{repeat}\ \square$ $\langle N \leq 0\rangle;\ \mathbf{exit\ od}$, which, in a sequential context, is equivalent with $\mathbf{while}\ N > 0\ \mathbf{do\ begin}\ Y := Y + N; N := N - 1\ \mathbf{end};$
2. $\mathbf{do}\ \langle C? = X'\rangle; \langle N' = {'N} + 1\rangle; \mathbf{exit}\ \ \square\ \ \langle D? = X'\rangle;$ $\langle M' = {'M} + 1\rangle; \mathbf{exit\ od}$, which, in a sequential context, is equivalent with
   $\mathbf{if}\ C?X \to N := N + 1\ \ \square\ \ D?X \to M := M + 1\ \mathbf{fi}.$

The semantics of the **do–od** construct involves the iteration operator $[NI * NR * NF]$. Recall that the net produced for this (figure 4) gives the effect of one execution of the net NI (initial net), followed by zero or more executions of the net NR (repeated net), followed by one execution of NE (exit net). For the semantics here we clearly need NR to be choice over each of the nets obtained as the semantics of one of the **repeat** clauses and NE to be choice over each of the nets obtained as the semantics of one of the **exit** clauses. For the NI net we will use the net for a silent transition, N_{silent} shown in Figure 14.

The NR and NE nets are produced by supplementary semantic functions, R and E (each with different versions for high level and low level net semantics). For $I \in \{H, L\}$:

$$\begin{array}{rcl} R_I(alt\text{-}set_1 \,\square\, alt\text{-}set_2) & = & R_I(alt\text{-}set_1) \,\square\, R_I(alt\text{-}set_2) \\ E_I(alt\text{-}set_1 \,\square\, alt\text{-}set_2) & = & E_I(alt\text{-}set_1) \,\square\, E_I(alt\text{-}set_2) \\ R_I(Com; \mathbf{repeat}) = E_I(Com; \mathbf{exit}) & = & Net_I(Com) \\ R_I(Com; \mathbf{exit}) = E_I(Com; \mathbf{repeat}) & = & N_{stop} \end{array}$$

where the net N_{stop} is shown in figure 14.

The R function recursively follows the nested structure of *alt-set* constructs, producing the same structure of nested choice net operations. This includes all the **repeat** clauses by mapping a base case which is a **repeat** clause to the net for its command – $R_I(Com; \mathbf{repeat}) = Net_I(Com)$.

---

[4] When we use N in an expression, this is an abbreviation for using either ${'N}$ or N' and adding the conjunct ${'N}{=}N'$. Thus, for instance, $\langle N{>}0\rangle$ is equivalent with $\langle {'N}{>}0 \wedge {'N}{=}N'\rangle$.

It effectively discards all the **exit** clauses by mapping a base case which is an **exit** clause to the net N_{stop}, shown in Figure 14, which is a unit of the net choice operator - $R_I(Com; \textbf{exit}) = N_{stop}$. The E function is the same but with the **repeat** and **exit** roles interchanged.

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

Definition 15. *Semantics of command connectives*
For $I \in \{H, L\}$,

$$\begin{array}{rcl} Net_I(P_1 \| P_2) & = & Net_I(P_1) \| Net_I(P_2) \\ Net_I(P_1; P_2) & = & Net_I(P_1); Net_I(P_2) \\ \text{and} \quad Net_I(\textbf{do } alt\text{-}set \textbf{ od}) & = & [\ N_{silent} * R_I(alt\text{-}set) * E_I(alt\text{-}set)\ ] \end{array}$$

where N_{silent} is defined in figure 14. $\diamond$

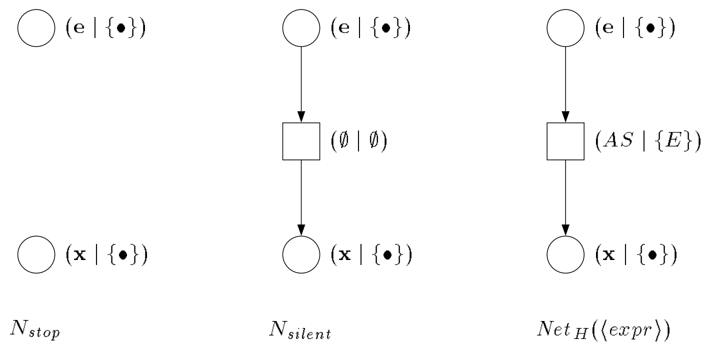


Fig. 14. Nets N_{stop}, N_{silent} and $Net_H(\langle expr \rangle)$ – see section 3.2.4 for AS and E.

3.2.4. Atomic Actions

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

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

Operators (op, e.g. $+$) and constants ($const$, e.g. 1) are standard programming constructs, details of which depend on a particular realisation of $B(PN)^2$. As far as the semantics is concerned, we assume every program constant to be included in *Val* and every program operator to be included in *Op*. The identifier W is a program variable. This may be declared as an ordinary variable, in which case the forms $'W$ and W' are meaningful, as discussed previously, or it may 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 $\langle W! = 'X + 2 \wedge X' = 'X \rangle$ correspond to, respectively, occam [28] notation $W?X$ and $W!(X+2)$. (However, note that W could be a channel of arbitrary capacity, while in occam, its capacity could only be zero.)

First we define a supplementary semantic function, *Ins*, to yield the semantics of an expression, e, as an action-expression pair $(AS \mid E)$, which is similar to a transition inscription $(AS \mid \{E\})$. AS is a set of action terms, the action names used being: W (arity 2) for access to an ordinary variable W; $W_?$ (arity 1) for input from, and $W_!$ (arity 1) for output to, a channel variable W. E is a term, which is e with program variables appropriately replaced by net variables: W^i for $'W$; W^o for W'; $W^?$ for $W?$; $W^!$ for $W!$.

$$
\begin{array}{lcl}
Ins('W) & = & (\{W(W^i, W^o)\} \mid W^i) \\
Ins(W') & = & (\{W(W^i, W^o)\} \mid W^o) \\
Ins(W?) & = & (\{W_?(W^?)\} \mid W^?) \\
Ins(W!) & = & (\{W_!(W^!)\} \mid W^!) \\
Ins(const) & = & (\emptyset \mid const) \\
Ins(e_1\, op\, e_2) & = & (AS_1 \cup AS_2 \mid E_1\, op\, E_2) \\
 & & \quad \text{for } Ins(e_i) = (AS_i \mid E_i) \text{ and } i \in \{1,2\} \\
Ins(op\, e) & = & (AS \mid op\, E) \ \text{ for } Ins(e) = (AS \mid E).
\end{array}
$$

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

$$
\begin{array}{lcl}
Ins('X{=}0) & = & (\{X(X^i, X^o)\} \mid X^i{=}0) \\
Ins(X'{=}C^?{+}1) & = & (\{X(X^i, X^o), C_?(C^?)\} \mid X^o{=}C^?{+}1) \\
Ins('X{=}0 \wedge X'{=}C^?{+}1) & = & (\{X(X^i, X^o), C_?(C^?)\} \mid X^i{=}0 \wedge X^o{=}C^?{+}1)
\end{array}
$$

By the following definition, this would give a net $Net_H(\langle expr \rangle)$ as shown on the right-hand side of figure 14, except that the transition inscription is

$$(\{X(X^i, X^o), C_?(C^?)\} \mid \{X^i = 0 \wedge X^o = C^? + 1)\}).$$

Definition 16. *Semantics of atomic actions*

For $Ins(expr) = (AS|E)$, define $Net_H(\langle expr \rangle)$ as shown on the right-hand side of figure 14 (with transition inscription $(AS \mid \{E\})$) and $Net_L(\langle expr \rangle) = U(Net_H(\langle expr \rangle))$. $\diamond$

3.3. A coherence property

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

Theorem 3. 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 $U(Net_H(P)) \equiv_T Net_L(P)$.

Proof. By induction on the structure of P. For the base cases, viz. for declarations and for atomic actions, the theorem is established by definition. The induction steps rely on the previous coherence results for M-nets proved in section 2.5. More precisely, the results used are: first, lemma 2, which establishes $U(N) = N$ for $N = N_{silent}$ or $N = N_{stop}$, i.e. isomorphism of unfolding for these nets; and second, that all operators commute with unfolding.

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

$$
\begin{aligned}
Net_L(D;B) \;\; \equiv_T & \;\; \text{(definition of } Net_L(D;B)) \\
& (Net_L(D) \| (Net_L(B); \tau(D))) \;\textbf{sy}\; \delta(D) \;\textbf{rs}\; \delta(D) \\
= & \;\; \text{(induction hypothesis)} \\
& (U(Net_H(D)) \| (U(Net_H(B)); U(\tau(D)))) \;\textbf{sy}\; \delta(D) \;\textbf{rs}\; \delta(D) \\
\equiv_T & \;\; \text{(propositions 1 and 3, lemma 2)} \\
& U(Net_H(D) \| (Net_H(B); \tau(D)) \;\textbf{sy}\; \delta(D) \;\textbf{rs}\; \delta(D)) \\
= & \;\; \text{(definition of } Net_H(D;B)) \\
& U(Net_H(D;B)).
\end{aligned}
$$

For the induction steps pertaining to the composition operators, the proofs are similar. ■

By observing the connection between arc-weighted labelled place/transition-nets and elementary M-nets and by theorem 3, the coherence of the original semantics from [8] with both semantics defined in this paper (high-level and low-level ones) is obtained.

4. Concluding Remarks

In this paper, an algebra on the domain of high-level nets has been proposed. It is based on [5] and [6] where the M-net model has first been published. The class of high-level nets defined here is unique in having labels for transitions and places on which are defined process algebraic composition operations. The most novel and interesting operation is synchronisation. This involves a unification step on labels, a logical conjunction step on annotations and a multiset addition step on surroundings arcs. Moreover this operation is commutative, a non-trivial property ensuring coherence with the elementary level box algebra and being important for compositionality.

Other high-level net models that provide some form of compositionality understand modularity more directly either in terms of place and/or transition fusion (e.g. [2, 12, 34]) or in terms of synchronisation expressions (e.g. [10]), but not driven by general place/transition labels. The kind of category style commutative diagram results we need (expressed in this paper as coherence) cannot be found in the other papers mentioned above.

M-nets have been generalised to A-nets (algebraic multilabelled nets [13, 25]), a class of nets supporting abstract data types (rather than just plain value sets). In [25] there is a generalisation of the proof of lemma 1 given in this paper, which shows that commutativity of synchronisation holds on the A-net level as well as on the M-net level.

In the second part of this paper the M-net domain has been used in order to give both a high-level and low-level net semantics of the specification / programming language $B(PN)^2$; and the coherence of these two semantics and thus the coherence of the high-level net semantics with the original semantics of [8] was shown. These semantics (both high-level 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 global-time independent 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 every time some analysis of the algorithm is to be performed).

It is the synchronisation structure on multisets of actions which allows the combination of P1 and P2, as was illustrated on an example in section 3. By 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 [30]). Moreover, if we used the SCCS style, which has a somewhat closer synchronisation on multisets of actions, then property P3 could not be achieved, as discussed in [4].

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 efficient techniques in state space searching [15]). The *PEP* system, originally supporting simulation and verification on the low-level net semantics of $B(PN)^2$ [7], has now been enhanced to a system 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.

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 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 as 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, see e.g. [37]. Work is also in progress to enrich the M-net domain by a sophisticated refinement operation [14], and to apply M-net and A-net ideas to an extension of $B(PN)^2$ by abstract data types [24] and procedures [16, 27].

Acknowledgments

This work has partly been supported by Esprit Basic Research Working Group CALIBAN (Causal Calculi Based on Nets), no 6067. We thank

Hans Fleischhack for his contributions to first versions of this model. We are grateful to Raymond Devillers, Bernd Grahlmann and the reviewers for helpful comments and Ingo Brückner for help with LaTeXing the figures.

References

1. Jos C.M. Baeten and W. P. Weijland: *Process Algebra*. Cambridge Tracts in Theoretical Computer Science 18 (1990).
2. Eugenio Battiston, Fiorella De Cindio, Giancarlo Mauri: OBJ-SA, a Class of High-level Nets Having Objects as Domains. Springer Verlag, Lecture Notes in Computer Science Vol. 340, 20-43 (1988).
3. Bernd Baumgarten: *Petri-Netze. Grundlagen und Anwendungen*. BI-Wissenschaftsverlag (1990).
4. Eike Best, Raymond Devillers, and Jon Hall: The Petri Box Calculus: a New Causal Algebra with Multilabel Communication. *Advances in Petri Nets'92*, G. Rozenberg (ed.). Springer-Verlag, Lecture Notes in Computer Science Vol. 609, 21–69 (1992).
5. Eike Best, Hans Fleischhack, Wojciech Frączak, Richard P. Hopkins, Hanna Klaudel and Elisabeth Pelz: A Class of Composable High Level Petri Nets. Proceedings of Petri Nets'95, Torino, G. De Michelis, M. Diaz (eds). Springer-Verlag, Lecture Notes in Computer Science Vol. 935, 103–120 (1995).
6. Eike Best, Hans Fleischhack, Wojciech Frączak, Richard P. Hopkins, Hanna Klaudel and Elisabeth Pelz: An M-net Semantics of $B(PN)^2$. Proc. *STRICT'95*, Berlin, J. Desel (ed.). Springer-Verlag, Workshops in Computing, 85–100 (1995).
7. Eike Best and Bernd Grahlmann: PEP - more than a Petri Net Tool. Proc. *Tools and Algorithms for the Construction and Analysis of Systems*, 2nd International Workshop, TACAS'96, Passau, March 1996, T. Margaria, B. Steffen (eds). Springer-Verlag, Lecture Notes in Computer Science Vol. 1055, 397–401 (1996).
8. Eike Best and Richard P. Hopkins: $B(PN)^2$ – a Basic Petri Net Programming Notation. Proc. *PARLE'93*, A. Bode, M. Reeve, G. Wolf (eds). Springer-Verlag, Lecture Notes in Computer Science Vol. 694, 379–390 (1993).
9. Eike Best and Thomas Thielke: Orthogonal Transformations for Coloured Petri Nets. Proc. of the eighteenth European Workshop on Applications and Theory of Petri Nets, ICATPN'97, Toulouse, P. Azéma, G. Balbo (eds), LNCS Vol. 1248, 447–466 (1997).
10. Didier Buchs and Nicolas Guelfi: CO-OPN: A Concurrent Object-Oriented Petri Nets Approach for System Specification. Proc. *12th ICPN*, Århus (1991). Also LRI Technical Report 616, Orsay, 1990.
11. Ludmila Cherkasova and Vadim Kotov: Descriptive and Analytical Process Algebras. *Advances in Petri Nets 1989*, G. Rozenberg (ed.). Springer-Verlag, Lecture Notes in Computer Science, Vol. 424, 77–104 (1990).
12. Soren Christensen and Laure Petrucci: Towards a Modular Analysis of Coloured Petri Nets. editor: Kurt Jensen, Lecture Notes in Computer Science Vol. 616, Springer-Verlag, 113-133 (1992).
13. Raymond Devillers and Hanna Klaudel: Refinement and Recursion in a High-level Petri Box Calculus. Proc. *STRICT'95*, Berlin, J. Desel (ed.). Springer-Verlag, Workshops in Computing, 144–195 (1995).
14. R. Devillers, H. Klaudel and R.-C. Riemann: General Refinement in the M-net Calculus. Proc. *FST-TCS'97*, Springer-Verlag, Lecture Notes in Computer Science Vol. 1346, 297–311, (1997).
15. J. Esparza: Model Checking based on Branching Processes. Habilitation, Hildesheim (1993). Published as: Model Checking Using Net Unfoldings. Proc. *TAPSOFT'93* (1993), M.C. Gaudel, J.P. Jouannaud (eds). Springer-Verlag, Lecture Notes in Computer Science Vol. 668, 613–628 (1993). Full version in *Science of Computer Programming* Vol. 23, 151–195 (1994).

16. H. Fleischhack und B. Grahlmann: A Petri Net Semantics for $B(PN)^2$ with Procedures. Proc. PDSE'97 (Parallel and Distributed Software Engineering), Boston, IEEE Coputer Society Press (1997).
17. Hartmann Genrich: Predicate-transition Nets. In *Petri Nets: Central Models and their Properties*, Advances in Petri Nets 1986, Part I. Springer-Verlag, Lecture Notes in Computer Science Vol.254, 207-247 (1987).
18. Ursula Goltz and Alan Mycroft: On the Relationship of CCS and Petri Nets. Proc. *ICALP'84*, J. Paredaens (ed.). Springer-Verlag, Lecture Notes in Computer Science Vol. 172, 196–208 (1984).
19. Michael Hack: Petri Net Languages, C.S.G. Memo 124, Project MAC, MIT (1975).
20. C.A.R. Hoare: *Communicating Sequential Processes*. Prentice Hall (1985).
21. R.P. Hopkins, J.G. Hall, and O. Botti: A basic-net algebra for program semantics and its application to occam. *Advances in Petri Nets 1992*, G. Rozenberg (ed.). Springer-Verlag, Lecture Notes in Computer Science Vol. 609, 179-214 (1992).
22. Matthias Jantzen: Languages Theory of Petri Nets, In Petri Nets: Central Models and their Properties, Advances in Petri Nets 1986, Part I. Lecture Notes in Computer Science Vol. 254, 397-412. Springer-Verlag, (1987).
23. Kurt Jensen: *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 1*. EATCS Monographs on Theoretical Computer Science (1992).
24. Hanna Klaudel: *Modèles algébriques, basés sur les réseaux de Petri, pour la sémantique des langages de programmation concurrents*. PhD Thesis, Université Paris-Sud, Orsay, (1995).
25. Hanna Klaudel and Elisabeth Pelz: Communication as Unification in the Petri Box Calculus. Proc. *FCT'95*. Springer-Verlag, Lecture Notes in Computer Science Vol. 965, 303-312 (1995).
26. H. Klaudel, R.-C. Riemann: *Consistent Equivalence Notion for a Class of High Level Petri Nets*. Eleventh International Symposium on Computer and System Science, Antalya (1996).
27. Johan Lilius and Elisabeth Pelz: An M-net Semantics for $B(PN)^2$ with Procedures. Eleventh International Symposium on Computer and System Science, Vol.I, 365–374, Antalya, Middle East Technical University (1996).
28. David May: occam. *SIGPLAN Notices* Vol.18(4), 69-79 (1983).
29. Robin Milner: *A Calculus of Communicating Systems*. Springer Verlag, Lecture Notes in Computer Science Vol. 92 (1980).
30. Robin Milner: *Communication and Concurrency*. Prentice Hall (1989).
31. Ernst-Rüdiger Olderog: *Nets, Terms and Formulas*. Cambridge Tracts in Theoretical Computer Science 23 (1991).
32. Elisabeth Pelz and Michel Parigot: A logical approach of Petri net languages, *Theoretical Computer Science* Vol.39, 155–169 (1985).
33. James L. Peterson: *Petri Net Theory and the Modelling of Systems*. Prentice Hall (1981).
34. Giuseppe Pinna: *Petri Nets and Their Composition Problem*. PhD Thesis: TD-2/90, Dipartimento di Informatica, Universitá degli Studi di Pisa, March 1990.
35. Wolfgang Reisig: *Petri Nets. An Introduction*. EATCS Monographs on Theoretical Computer Science Vol. 4, Springer-Verlag (1985).
36. Wolfgang Reisig: Petri Nets and Algebraic Specifications. *Theoretical Computer Science* Vol.80, 1-34. 1991.
37. Karsten Schmidt: *Symbolische Analysealgorithmen für algebraische Petrinetze*. Dissertation (1996).
38. Einar Smith: A Primer on High-Level Petri-Net Theory. Informatik-Bericht Nr. 25 der Humboldt-Universität zu Berlin, 1993.
39. Einar Smith and Wolfgang Reisig: The Semantics of a Net is a Net – an Exercise in General Net Theory. *Advances in Petri Nets'87*, G. Rozenberg (ed.). Springer-Verlag, Lecture Notes in Computer Science Vol. 266, 461-480 (1987).
40. Jacques Vautherin: Parallel systems specification with colored Petri nets and algebraic specification. Springer-Verlag, Lecture Notes in Computer Science Vol. 266, 293-308 (1987).
41. Walter Vogler: *Modular construction and Partial Order Semantics of Petri Nets*. Springer-Verlag, Lecture Notes in Computer Science Vol. 625 (1991).

A. Auxiliary net operators

We assume that all nets are *ex*-good.

Definition 17. *Making disjoint copies of a net*
For an integer i and an M-net $N = (S, T, \iota)$ we define $iN = (\{(s,i) \mid s \in S\}, \{(t,i) \mid t \in T\}, \iota')$ where $\forall x \in S \cup T\colon \iota'(x,i) = \iota(x)$ and
$\forall (x,y) \in (S \times T) \cup (T \times S)\colon \iota'((x,i),(y,i)) = \iota(x,y)$. $\diamond$

Definition 18. *Place multiplication*
For a set $\mathcal{S} = \{S_1, \ldots, S_n\}$ of mutually disjoint sets (of places) $\iota_1, \ldots, \iota_n$, we define

$$\otimes\mathcal{S} \quad = \quad \{\pi \subseteq \bigcup_{i=1}^{n} S_i \mid |\pi \cap S_i| = 1 \text{ for each } i \in \{1..n\}\}.$$

Each of the S_i is a set of places which participates in the place multiplication. A new place $\pi = \{s_1, \ldots, s_n\}$ with $s_i \in S_i$ for each $i \in \{1..n\}$ is obtained for every possible way of taking one place from each participating set. The inscription $\iota(\pi)$ is obtained as follows:

$$\iota(\pi) \quad = \quad \begin{cases} (\lambda \mid \alpha) & \text{if} \forall i \in \{1, \ldots, n\}\colon \iota(s_i) = (\lambda \mid \alpha) \\ (\emptyset, \{\bullet\}) & \text{otherwise.} \end{cases}$$ $\diamond$

Note that place multiplication is symmetrical; e.g., the multiplication of two sets of places $\{r_1, r_2\}$ and $\{s_1, s_2\}$ is

$$\otimes\{\{r_1, r_2\}, \{s_1, s_2\}\} \quad = \quad \{\{r_1, s_1\}, \{r_1, s_2\}, \{r_2, s_1\}, \{r_2, s_2\}\}.$$

The next operation is used to add into a net a set of new places S', normally created by a place multiplication on sets of places of the net, and delete the original places used in the multiplication.

Definition 19. *Place substitution*
Let $N = (S, T, \iota)$ be an M-net and $S' = \{\pi_1, \ldots, \pi_n\}$ such that for all $\pi_i \in S'$, $\pi_i \subseteq S$ and $\pi_i \neq \emptyset$. Taking $\Pi = \pi_1 \cup \ldots \cup \pi_n$, we define

$$N \oplus S' = ((S \backslash \Pi) \cup S', T, \iota')$$

where ι' is defined as follows:

$$\begin{array}{l} \forall x \in (S \backslash \Pi) \cup S' \cup T\colon \iota'(x) = \iota(x) \\ \forall x \in ((S \backslash \Pi) \times T) \cup (T \times (S \backslash \Pi))\colon \iota'(x) = \iota(x) \\ \forall (\pi, t) \in (S' \times T)\colon \iota'(\pi, t) = \sum_{s \in \pi} \iota(s, t) \\ \forall (t, \pi) \in (T \times S')\colon \iota'(t, \pi) = \sum_{s \in \pi} \iota(t, s). \end{array}$$

The sum in the last two lines denotes multiset sum (of multisets of variables). $\diamond$

As it happens, we shall use this construction in such a way that the following holds in addition to the properties (1.) and (2.):

$$\forall t \in T \colon |{}^\bullet t \cap \pi_i| \leq 1 \geq |t^\bullet \cap \pi_i|,$$

for all elements π_i of S'. Here, ${}^\bullet t$ and $t^\bullet$ denote the set of predecessor and successor places of t, respectively. Place multiplication and substitution are geared to our present needs and would need to be modified or extended in other contexts.

Definition 20. *Net union*
Let $N = (S, T, \iota)$ and $N' = (S', T', \iota')$ be two disjoint (i.e. $S \cap S' = \emptyset$ and $T \cap T' = \emptyset$) M-nets. Their union is defined as

$$N \cup N' \;\; = \;\; (S'', T'', \iota''),$$

where $S'' = S \cup S'$, $T'' = T \cup T''$, and all inscriptions of ι'' are carried over from ι or ι', respectively, whatever applies. $\diamond$

This article was processed by the author using the LaTeX style file *LJour1* from Springer-Verlag.