



**HAL**  
open science

# A concurrent and compositional Petri net semantics of preemption

Hanna Klaudel, Franck Pommereau

► **To cite this version:**

Hanna Klaudel, Franck Pommereau. A concurrent and compositional Petri net semantics of preemption. *Integrated Formal Methods (IFM)*, 2006, Dagstuhl, Germany. pp.318-337, 10.1007/3-540-40911-4\_19 . hal-00114693

**HAL Id: hal-00114693**

**<https://hal.science/hal-00114693>**

Submitted on 17 Nov 2006

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# A concurrent and compositional Petri net semantics of preemption

Hanna Klaudel and Franck Pommereau

LACL, Université Paris XII  
61, avenue du Général de Gaulle  
94010 Créteil, France  
{klaudel,pommereau}@univ-paris12.fr

**Abstract.** The aim of this paper is the introduction of preemption in a compositional model, called *M-nets*, which is based on Petri nets and hence provided with a concurrent semantics. We propose a way to model preemptible systems by extending the M-net model with priorities and the M-net algebra with a preemption operator. We show that these extensions can be seen as a high-level version of the well studied model of *priority systems*, and so, can be reduced to Petri nets (without priorities) which retain as much as possible of the original concurrency. As a consequence, Petri nets appear as a model powerful enough to deal with preemption in a compositional way and with a concurrent semantics.

**Keywords.** Petri nets, Preemption, Concurrency, Compositionality.

## 1 Introduction

*Preemption* relates to controlling the execution of the processes composing a concurrent system. Such processes are said *preemptible* if they can be suspended at any point of their execution.

Preemption is often addressed in *reactive systems*, for instance in synchronous models and languages [14, 1]; for some of them, it is even an essential feature. In most cases, the underlying semantics is *sequential*, which is well suited to the modeling of systems in which the computation performed in response to an input coming from the environment is relatively simple. But when the structure of the computation becomes more important than the structure of the reaction, the sequential semantics may be not sufficient. A *concurrent* semantics is often more adapted to the modeling of *heterogeneous* architectures which combine software (distributed on several processors) and specialized hardware components. In particular, playing with the scheduling of operations often allows a better resource management.

Petri nets form an inherently *asynchronous* model in which concurrency can be represented explicitly. This model and some of its extensions [22, 23] have been used for works on preemption, but in an unstructured way (non compositional).

This paper addresses the question of preemption in the context of compositional Petri nets. This naturally leads to consider the framework defined by

the *Petri Box Calculus* (PBC [3, 2]). The proposed approach tends to be as conservative as possible with respect to the existing framework, with the goal to minimize the changes necessary in order to adapt the existing tools to the proposed model.

PBC is a process algebra with a syntactic domain of *box-expressions* and a corresponding semantic domain of *boxes*, a class of labelled 1-safe Petri nets provided with an algebraic structure. It has been introduced with the aim of modeling the semantics of concurrent systems and programming languages. In order to cope with the possibly huge size of the nets, higher level versions of PBC have been considered, and in particular an algebra of *M-expressions* (high-level equivalent of box-expressions [18, 16]) and *M-nets* (high-level Petri net version of boxes [4]) which allow to represent large (possibly infinite) systems in a clear and compact way. The high- and low-level domains are related by an operation of *unfolding* which associates a box-expression to each M-expression and a box to each M-net.

The PBC framework also features a parallel programming language,  $B(PN)^2$  [5], which can be seen as a “user friendly” syntax on the top of both, high- and low-level process algebras. It is implemented in PEP toolkit [13], allowing to simulate modeled systems and to verify their properties via model checking. Several contributions [5, 4, 20, 12, 19, 15] provide applications to the PBC theory where box-expressions, M-nets and M-expressions are used as the semantical domain for  $B(PN)^2$ .

In this paper, the M-net model is extended by considering *priority M-nets* as pairs  $(N, \rho)$  where  $N$  is an M-net and  $\rho$  a pairwise priority relation between its transitions. The M-net algebra is then enriched by a new operation,  $\pi$ , which allows to make preemptible any priority M-net and can be nested arbitrarily.

We are particularly interested in a sub-class of priority M-nets, called *pre-emptible M-nets* (*P/M-nets*), which fulfill some structural constraints. We show that the concurrent (step) semantics of P/M-nets is sound with respect to the semantics of preemption.

Moreover, applying results obtained in some related areas, we show for a large class of P/M-nets, that they can be transformed into 1-safe Petri nets (without priorities), retaining as much as possible of the concurrent semantics. The transformation leads to really huge nets which cannot be used in practice, nevertheless, this means that 1-safe Petri nets are expressive enough to model preemption in a compositional framework.

The rest of the paper is organized as follows. Section 2 gives some intuition about the aspects of M-nets which are important for our purpose. Section 3 discusses preemption and the impact of its introduction in the context of Petri nets. Section 4 introduces priority M-nets, defines operation  $\pi$ , and extends the usual M-net operations to such nets. Then, P/M-nets are introduced as a structurally restricted class of priority M-nets, and their concurrent semantics is shown sound with respect to the semantics of preemption. A detailed example with nested P/M-nets is given in section 5. Section 6 discusses some properties of P/M-nets

and, in particular, their transformation to 1-safe Petri nets. The paper ends with some concluding remarks.

## 2 M-net model

### 2.1 Basic definitions

Let  $E$  be a set. A *multi-set* over  $E$  is a function  $\mu : E \rightarrow \mathbb{N}$ , generally denoted with an extended set notation, *e.g.*,  $\{a, a, b\}$  for  $\mu(a) = 2$ ,  $\mu(b) = 1$  and  $\mu(e) = 0$  for all  $e \in E \setminus \{a, b\}$ .  $\mu$  is finite if so is its support set  $E \setminus \mu^{-1}(0)$ . We denote by  $\mathcal{M}(E)$  (resp.  $\mathcal{M}_f(E)$ ) the set of multi-sets (resp. finite multi-sets) over  $E$ , by  $\oplus$  and  $\ominus$  the sum and difference of multi-sets. We may also use the usual sets notations such as  $\subseteq$  or  $\in$ ; for instance,  $e \in \mu$  stands for  $\mu(e) > 0$ .

### 2.2 M-nets

M-nets [4] form a class of high-level Petri nets provided with a set of operations giving them a structure of process algebra. We use here the M-net model defined in [8], and its asynchronous links extension from [17].

An M-net  $N$  is a triple  $(S, T, \iota)$ , where  $S$  is the set of places,  $T$  is the set of transitions,  $(T \times S) \cup (S \times T)$  is the set of arcs, and  $\iota$  is the annotation function on places, transitions and arcs. The annotation of a place is of the form  $\lambda.\tau$ , where  $\lambda$  is a *label* (entry  $e$ , exit  $x$  or internal  $i$ ) and  $\tau$  is a *type* (a non-empty set of values from a fixed set  $Val$ ). As usual, for each node (place or transition)  $r \in S \cup T$ , we denote by  $\bullet r$  the set of nodes  $\{r' \in S \cup T \mid \iota(r', r) \neq \emptyset\}$  and, similarly,  $r\bullet = \{r' \in S \cup T \mid \iota(r, r') \neq \emptyset\}$ .

Transitions annotations are of the form  $\lambda.\gamma$  where  $\lambda$  is a *label* (which can be hierarchical or for communications) and  $\gamma$  is a *guard* (a finite set of predicates from a set  $Pr$ ). Hierarchical labels are composed out of a single hierarchical action (*e.g.*,  $\mathcal{X}$ ) indicating a future refinement (*i.e.*, a substitution) by an M-net. Communications may be:

- *synchronous*, similar to CCS ones [21], *e.g.*, between transitions labelled by synchronous communication actions such as  $A(a_1, \dots, a_n)$  or  $\hat{A}(a'_1, \dots, a'_n)$ , where  $A$  is a *synchronous communication symbol*,  $\hat{A}$  is its *conjugate* and each  $a_i$  and  $a'_i$  is a value or a variable (belonging to a fixed set  $Var$ );
- *asynchronous*, *e.g.*, between transitions labelled by asynchronous links such as  $b^+(a_1)$  or  $b^-(a_2)$ , where  $b$  is an *asynchronous communication symbol* and each  $a_i$  is a value or a variable (ranging in  $type(b) \subseteq Val$ ). The communication is done via a place  $s_b$  of type  $\tau(s_b) = type(b)$  which plays the rôle of a heap buffer. Link  $b^+(a_1)$  means that  $a_1$  can be sent to  $s_b$  and  $b^-(a_2)$  means that  $a_2$  can be received from  $s_b$ ;
- or possibly both at the same time.

Communication labels are then of the form  $\lambda = \alpha.\beta$  where  $\alpha$  is a finite multi-set of synchronous communication actions and  $\beta$  is a finite multi-set of asynchronous links.

Arcs are inscribed by multi-sets of *structured annotations* representing multi-sets of values consumed or produced by a transition in a place. Structured annotations are variables, values or more complex structures allowing to cope with place types generated by refinements [8, 10]. As usual, in figures, arcs with an empty annotation will be omitted; moreover, annotation  $\{\bullet\}$  on an arc is omitted most of time and singletons are often replaced by their unique element.

### 2.3 Dynamic behavior and concurrent semantics

For each transition  $t \in T$  we shall denote by  $var(t)$  the set of all the variables occurring in the annotations of  $t$  and in the arcs coming to and from  $t$ . A *binding* for a transition  $t$  is a substitution  $\sigma : var(t) \rightarrow Val$ ; it will be said *enabling* if it satisfies the guard, if it respects the types of the asynchronous links, and if the flow of tokens it implies respects the types of the places adjacent to  $t$ .

A *marking* of an M-net  $(S, T, \iota)$  is a mapping  $M : S \rightarrow \mathcal{M}(Val)$  which associates to each place  $s \in S$  a multi-set of values from  $\tau(s)$ . In particular, we shall distinguish the *entry marking*, denoted  $M_e$ , where, for each  $s \in S$ ,  $M_e(s) = \tau(s)$  if  $\lambda(s) = e$  and the empty multi-set otherwise; the *exit marking* is defined similarly. The dynamic behavior of an M-net starts with its entry marking; it ends (if ever) with the exit marking.

The transition rule specifies the circumstances under which a marking  $M'$  is reachable from a marking  $M$ . A transition  $t$  is *enabled* at a marking  $M$  (this is denoted  $M[t)$ ) if there is an enabling binding  $\sigma$  of  $t$  such that  $\forall s \in S : \iota(s, t)[\sigma] \subseteq M(s)$ , *i.e.*, there are enough tokens of each type to satisfy the required flow. The effect of an occurrence of  $t$  is to remove from its input places all the tokens used for the enabling binding  $\sigma$  and to add to its output places the tokens according to  $\sigma$ ; this leads to a marking  $M'$  such that  $\forall s \in S : M'(s) = M(s) \ominus \iota(s, t)[\sigma] \oplus \iota(t, s)[\sigma]$ .

The above transition rule defines the *interleaving semantics* of an M-net which consists in a set of occurrence sequences. This semantics can be generalized by introducing the *step sequence semantics* [9], which allows any number of transitions to occur simultaneously.

Given an M-net  $N = (S, T, \iota)$ , a multi-set  $\delta$  of transitions is said *concurrently enabled* at a marking  $M$  if there are enough tokens to allow the simultaneous firing of all the transitions in  $\delta$ . Such a  $\delta$  is called a *step*. A *step sequence* of  $N$  is a sequence  $D = (\delta_1, \delta_2, \dots)$  such that there are markings  $M_1, M_2, \dots$ , where  $M_1 = M_e$  and which satisfy  $M_i[\delta_i]M_{i+1}$  for  $i \geq 1$ . The set of step sequences of  $N$  is its step sequence semantics and is denoted by  $steps(N)$ . It is easy to see that  $steps(N)$  is stable under linearisation: if  $\delta$  belongs to a step sequence  $D$  in  $steps(N)$ , then, replacing  $\delta$  with any of its linearisation gives a step sequence which is also in  $steps(N)$  (*e.g.*,  $\{t_1, t_2\}$  can be replaced by  $\{t_1\}\{t_2\}$  or  $\{t_2\}\{t_1\}$ ).

### 2.4 Unfolding

Let  $N = (S, T, \iota)$  be an M-net. The unfolding of  $N$  is the labelled Petri net  $\mathcal{U}(N) = (\mathcal{U}(S), \mathcal{U}(T), W, \lambda)$ , where  $\mathcal{U}(S)$  is the set of places,  $\mathcal{U}(T)$  the set of

transitions,  $W$  the weight function on arcs and  $\lambda$  the labelling function on places and transitions, defined as follows:

- $\mathcal{U}(S) = \{(s, v) \mid s \in S \text{ and } v \in \tau(s)\}$ , and  $\forall (s, v) \in \mathcal{U}(S) : \lambda((s, v)) = \lambda(s)$ ;
- $\mathcal{U}(T) = \{(t, \sigma) \mid t \in T \text{ and } \sigma \text{ is an enabling binding of } t\}$ ,  
and  $\forall (t, \sigma) \in \mathcal{U}(T) :$
- $\lambda((t, \sigma)) = \begin{cases} \alpha(t)[\sigma], \beta(t)[\sigma] & \text{if } t \text{ is a communication transition,} \\ \lambda(t) & \text{if } t \text{ is a hierarchical transition;} \end{cases}$
- $W((s, v), (t, \sigma)) = \sum_{x \in \iota(s, t)} \iota(s, t)(x) \cdot x[\sigma](v)$ , where  $x[\sigma](v)$  is the number of values  $v$  occurring in the structured annotation  $x$  evaluated under  $\sigma$ ;
- $W((t, \sigma), (s, v))$  is defined analogously.

If  $M$  is a marking of  $N$ , the marking  $\mathcal{U}(M)$  of  $\mathcal{U}(N)$  is defined as follows: for every place  $(s, v) \in \mathcal{U}(S)$ ,  $\mathcal{U}(M)((s, v)) = M(s)(v)$ , *i.e.*, each low-level place  $(s, v) \in \mathcal{U}(S)$  contains as many tokens as the number of occurrences of  $v$  in the marking  $M$  of  $s$ .

The unfolding can easily be extended to steps and step sequences, and one can observe that the step semantics obtained by unfolding the step semantics of an M-net  $N$  equals the step semantics obtained from  $\mathcal{U}(N)$ .

**Theorem 1.** *Let  $N$  be an M-net. Then,  $\mathcal{U}(\text{steps}(N)) = \text{steps}(\mathcal{U}(N))$ .*

*Proof.* By definition of the unfolding and by the analogous property for interleaving semantics [4].

## 2.5 Algebra of M-nets

For compositionality, we are particularly interested in a sub-class of M-nets: we assume that each M-net has at least one entry and one exit place, that each transition has at least one input and one output place (*T-restrictness* property), and that there are neither arcs going to entry places nor from exit places. Such M-nets are said *ex-good*.

The algebra of ex-good M-nets comprises the operations listed below, where  $N_1$ ,  $N_2$  and  $N_3$  are M-nets,  $\mathcal{X}$  is a hierarchical symbol,  $A$  is a synchronous communication symbol,  $b$  is an asynchronous link symbol and  $f$  is a renaming function on synchronous and asynchronous symbols. Detailed explanations and some examples of these operations are given in [4, 10, 17].

$N_1[\mathcal{X} \leftarrow N_2]$	refinement	$N_1[f]$	renaming
$N_1 \parallel N_2$	parallel composition	$N_1 \mathbf{sy} A$	synchronization
$N_1; N_2$	sequence	$N_1 \mathbf{rs} A$	restriction
$N_1 \square N_2$	choice	$[A : N_1]$	scoping
$[N_1 * N_2 * N_3]$	iteration	$N_1 \mathbf{tie} b$	asynchronous links

In the following the considered M-nets are ex-good, except if specified explicitly.

### 3 Preemption

In the following, we make a difference between *programs* (nets) and *processes* (actually, step sequences) which are their dynamic behaviors; each step sequence being a possible *execution* of its *supporting net*.

#### 3.1 Abortion versus suspension

When dealing with preemption, two notions are usually separated: *suspension* which “freezes” an execution but keeps it alive for a possible restart, and *abortion* which kills an execution definitively. Our approach deals with both of them, but focusing on abortion. More precisely, we treat abortion as a suspension followed by some processing in order to remove all the tokens from the net which supports the execution, making it unable to evolve anymore. This solution is based on priorities which make the problem of suspension quite straightforward to solve: in order to suspend a given net, it is enough to enable one of its transitions which has the priority over all the others. Actually this is what we propose: during all the abortion stage, there is always such a transition which is enabled and thus freezes the rest of the net when it is being emptied.

#### 3.2 Preemption and time

Preemption is often associated to time, at least intuitively, because it is expected to have an immediate effect on a system. As far as Petri nets are concerned, “immediate” means that no program transition may fire, from the beginning until the end of the preemption. Of course, in some other contexts, introducing time together with preemption makes sense, but it is not the case for our purpose. This is the reason why this paper never deals with time or time related concepts.

#### 3.3 Internal versus external abortion

From the point of view of an execution, there is a difference between an *internal abortion* (when the execution “decides” to give-up its current work) and an *external abortion* (when the execution is killed by its environment). In both cases, the execution is suspended (*i.e.*, no further transition in the corresponding net is allowed to fire, except, possibly, some well identified transitions involved into the abortion itself), and its supporting net must be “emptied” (*i.e.*, tokens must be removed from the net). All the tokens have to be removed: first, because they make alive the execution being killed, and moreover, because the net must be cleaned up for a possible future re-usage, as the support of another execution.

In the case of an internal abortion, however, the M-net *must not* be completely emptied because its environment is not aware of the abortion and so waits for its completion. Actually, the environment is expecting the exit marking of the net which supports the aborted execution. As a consequence, the case of internal abortion corresponds to an anticipated *termination*, which reflects the abortion of the execution, followed by its completion through the production of the exit

marking. In a way, internal abortion is a simple mean by which an execution can terminate “cleanly”, regardless of its current state.

In the case of an external abortion, various behaviors may be acceptable since the environment is aware of the situation (by definition of external abortion, it is initiated by the environment). In our approach, we choose a solution where the aborted net fires a particular transition which warns its environment that the abortion stage is over, but the exit marking is not reached. This way allows our construction to manage the abortion of nested executions quite elegantly: abortion is transmitted from the top to the bottom, from external executions to nested ones.

### 3.4 Modeling preemption in Petri nets

A preemptible Petri net should be able to run under two mutually exclusive modes: in “standard mode”, it processes its program; in “abortion mode”, it must stop its normal activity, empty and complete (reaching or not its exit marking). Abortion mode can interrupt standard mode, but not the reverse.

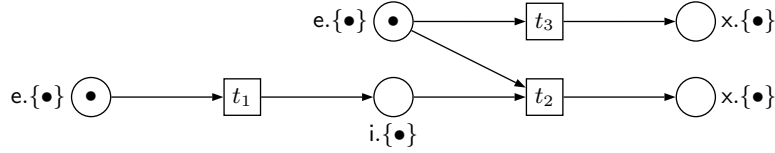
On the one hand, we can consider that, before any move, standard mode checks that abortion mode is disabled. The execution checks the absence of token in the net part which supports abortion mode. This is a zero test, which can be modeled by introducing inhibitor arcs or complementary places. In this point of view, standard mode is responsible for freezing itself when necessary. On the other hand, we can consider that abortion mode has the priority over normal mode: if, in the net, transitions  $t_n$  for normal mode and  $t_a$  for abortion mode are both enabled,  $t_a$  should be always preferred. This point of view naturally leads to consider priorities between transitions. Here, standard mode is completely passive with respect to its freezing.

In this paper, we prefer the second point of view since it allows us to bring to the theory of *priority systems* as presented in [6]. Actually, M-nets extended with priorities and with a suitable definition of unfolding, lead directly to priority systems. This allows us to apply the main result from [6], which consists in transforming a priority system  $(\Sigma, \rho)$  in a Petri net  $\Sigma_\rho$  which is derived from  $\Sigma$  in such a way that it preserves as much as possible of the concurrency in  $\Sigma$  and does not violate the priority constraints specified by  $\rho$ .

## 4 Preemptible M-nets

The purpose of this section is a definition of *preemptible M-nets* (*P/M-nets* for short), a class of composable M-nets provided with some additional information about priority between transitions and having some structural properties which ensure the soundness of their step semantics with respect to the semantics of preemption. For the definition of P/M-nets we proceed as follows: first, we consider an auxiliary and very powerful class of nets called *priority M-nets*, analogous to *priority systems* from [6], as M-nets equipped with a pairwise priority relation between their transitions. The transition rule of these nets takes into account





**Fig. 1.** A marked priority M-net with  $\rho = \{(t_3, t_2)\}$  (with simplified annotations).

the information about priority and so does the step semantics. Then, we extend M-net operations to priority M-nets, giving to them an algebraic structure, and we define a new operation for priority M-nets, called  $\pi$ , which serves to make preemptible any priority M-net. Finally, as the main definition of this section, we introduce P/M-nets as a sub-class of priority M-nets having interesting structural properties with respect to preemption.

#### 4.1 Pairwise priorities

Let  $N = (S, T, \iota)$  be an M-net. A binary relation  $\rho \subseteq T \times T$  is called a pairwise priority relation. Intuitively,  $(t_1, t_2) \in \rho$  means that during an execution of  $N$ , the firing of transition  $t_2$  is always preferred to  $t_1$  when both are enabled; in other words,  $t_1$  has a lower priority than  $t_2$ . We use standard mathematical notations, in particular, for  $\rho \subseteq T \times T$ , we denote:

$$\begin{aligned} \text{dom}(\rho) &= \{t_1 \in T \mid \exists t_2 \in T \text{ such that } (t_1, t_2) \in \rho\}, \\ \text{cod}(\rho) &= \{t_2 \in T \mid \exists t_1 \in T \text{ such that } (t_1, t_2) \in \rho\}. \end{aligned}$$

#### 4.2 Priority M-nets

A *priority M-net* is a pair  $P = (N, \rho)$  where  $N = (S, T, \iota)$  is an M-net (possibly having some non T-restricted communication transitions) and  $\rho \subseteq T \times T$  is a pairwise priority relation over  $T$ . We call  $N$  the *net part* of  $P$ .

**Definition 1.** Let  $P = (N, \rho)$  be a priority M-net,  $M$  a marking of  $N = (S, T, \iota)$  and  $t$  a transition of  $N$  such that  $M[t]$ ; then  $t$  is  $\rho$ -enabled in  $P$  at  $M$ , denoted  $M[t]_\rho$ , if  $\nexists t' \in T$  such that  $M[t']$  and  $(t, t') \in \rho$ .

Notice that  $\rho$  allows to disable a transition which would have been enabled with usual M-nets transition rule, but not the reverse. In other words, we have  $M[t]_\rho \Rightarrow M[t]$ .

The notion of step and step sequence defined for M-nets could be directly reused for priority M-nets. But, this way, they would lead to inconsistencies in the semantics. Consider for example the priority M-net  $P = (N, \rho)$  shown in figure 1 (taken from [6]), if we do not take  $\rho$  into account, we have the step semantics:

$$\text{steps}(N) = \{\emptyset, \{t_1\}, \{t_3\}, \{t_1, t_3\}, \{t_1\}\{t_3\}, \{t_3\}\{t_1\}, \{t_1\}\{t_2\}\},$$

where  $\emptyset$  is the empty step sequence.

We can see that it contains the sequence  $\{t_1\}\{t_3\}$  which violates  $\rho$ . Removing this sequence is necessary but not enough since it introduces inconsistency. Actually, the semantics cannot contain  $\{t_1, t_3\}$  because  $\{t_1\}\{t_3\}$  is one of its linearisations. The *consistent step semantics* of  $P$ , denoted  $steps(P)$ , is thus the biggest sub-set of  $steps(N)$  such that each step sequence  $D \in steps(P)$  and each of its linearisations respect  $\rho$ . So, we have:

$$steps(P) = \{\emptyset, \{t_1\}, \{t_3\}, \{t_3\}\{t_1\}, \{t_1\}\{t_2\}\}.$$

According to [6], this consistent step semantics is one of the most concurrent semantics one can expect for priority systems.

The unfolding of priority M-nets is a natural extension of the unfolding of M-nets.

**Definition 2.** Let  $P = (N, \rho)$  be a priority M-net. The unfolding of  $P$ ,  $\mathcal{U}(P)$ , is a pair  $(\mathcal{U}(N), \mathcal{U}(\rho))$  where  $\mathcal{U}(N)$  is the usual M-net unfolding and  $\mathcal{U}(\rho)$  is defined as the smallest set such that: for each pair  $(t, t') \in \rho$  such that  $t$  is unfolded into a set of low-level transitions  $\{(t, \sigma_1), \dots, (t, \sigma_n)\}$  and  $t'$  is unfolded into  $\{(t', \sigma'_1), \dots, (t', \sigma'_k)\}$ , we have  $\{(t_i, \sigma_i), (t'_j, \sigma'_j) \mid 1 \leq i \leq n \wedge 1 \leq j \leq k\} \subseteq \mathcal{U}(\rho)$ . If  $M$  is a marking of  $N$ , then  $\mathcal{U}(M)$  is defined as it is for M-nets.

As for M-nets, an extension of the unfolding of priority M-nets to consistent steps and consistent step sequences is straightforward and we still have:

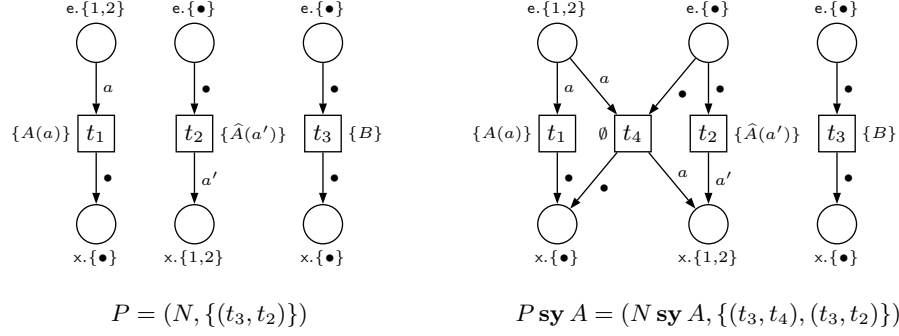
$$\mathcal{U}(steps(P)) = steps(\mathcal{U}(P)).$$

### 4.3 Algebra of priority M-nets

The extension of usual M-net operations to priority M-nets is immediate for most of them. However, in the case of synchronization or refinement, several possible definitions of priority relation can be considered. Our choice is *not* the most general possible, we already have in mind the definition of operation  $\pi$  and of P/M-nets. Priority M-nets are just an intermediate step which avoids circular definitions.

**Definition 3.** Let  $P_i = (N_i, \rho_i)$ , for  $i \in \{1, 2, 3\}$ , be priority M-nets, where  $N_i = (S_i, T_i, \iota_i)$ , and let  $\mathcal{X}$  be a hierarchical symbol,  $A$  a synchronous communication symbol,  $b$  an asynchronous link symbol, and  $f$  a renaming function on communication symbols. The usual M-net operations are extended as follows for priority M-nets:

- $P_1[\mathcal{X} \leftarrow P_2] = (N_1[\mathcal{X} \leftarrow N_2], \rho)$  where
  - $\rho = \{(t, t') \in \rho_1 \mid \lambda_1(t) \neq \mathcal{X} \neq \lambda_1(t')\}$
  - $\uplus \{(t_{\mathcal{X}}.t, t_{\mathcal{X}}.t') \mid (t, t') \in \rho_2 \wedge t_{\mathcal{X}} \in T_1 \wedge \lambda_1(t_{\mathcal{X}}) = \mathcal{X}\}$
  - $\uplus \{(t_{\mathcal{X}}.t, t') \mid t \notin \text{cod}(\rho_2) \wedge (t_{\mathcal{X}}, t') \in \rho_1 \wedge t_{\mathcal{X}} \in T_1 \wedge \lambda_1(t_{\mathcal{X}}) = \mathcal{X}\};$
- $P_1 \text{ tie } b = (N_1 \text{ tie } b, \rho_1);$



**Fig. 2.** Example of synchronization of priority M-nets. (Only synchronous labels are represented.) Restricting on  $A$  would remove from the net  $t_1$  and  $t_2$  (with their surrounding arcs) and  $(t_3, t_2)$  from its priority relation.

- $P_1[f] = (N_1[f], \rho_1)$ ;
- $P_1 \text{ sy } A = (N_1 \text{ sy } A, \rho)$  where  $N_1 \text{ sy } A = (S, T, \iota)$  and  $\rho$  is the smallest set including  $\rho_1$  such that if  $t' \in T$  results from a basic synchronization of  $t_1$  with  $t_2$ , and
  - if  $\exists t''$  such that  $(t_1, t'') \in \rho$  or  $(t_2, t'') \in \rho$ , then  $(t', t'') \in \rho$ ,
  - if  $\exists t''$  such that  $(t'', t_1) \in \rho$  or  $(t'', t_2) \in \rho$ , then  $(t'', t') \in \rho$ .
- $P_1 \text{ rs } A = (N_1 \text{ rs } A, \rho)$ , where  $N_1 \text{ rs } A = (S, T, \iota)$  and  $\rho = \rho_1 \cap (T \times T)$ .

Control flow operators (sequential composition ( $;$ ), iteration ( $[* * *]$ ), parallel composition ( $\parallel$ ) and choice ( $\square$ )) are based on refinement and so defined canonically. Scoping is defined as a synchronisation followed by a restriction:  $[A : P] = (P \text{ sy } A) \text{ rs } A$ .

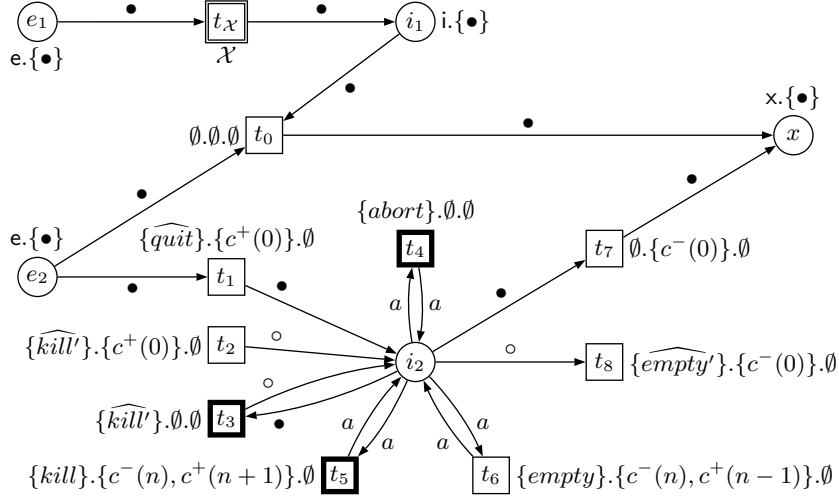
#### 4.4 A new operation for preemption

In order to define operation  $\pi$ , we use the priority M-net  $P_\pi = (N_\pi, \rho_\pi)$  where  $N_\pi$  is represented in figure 3 and the priority relation is

$$\rho_\pi = \{(t_7, t_4), (t_7, t_5), (t_8, t_4), (t_8, t_5), (t_{\mathcal{X}}, t_5), (t_2, t_3)\}.$$

The usage we make of  $N_\pi$  is rather simple, even if the net may look quite complex:

- the top part ( $e_1$ ,  $t_{\mathcal{X}}$  and  $i_1$ ) embeds the net  $N$  from  $(N, \rho)$  which should be made abortable. When  $N$  terminates normally (with no preemption), transition  $t_0$  fires, consuming the token in place  $e_2$  and producing the exit marking in  $x$ ;
- all the rest is used for preemption: internal abortion starts with a firing of  $t_1$  which produces a token  $\bullet$  in place  $i_2$ ; external abortion starts when  $t_2$  or  $t_3$  fires, producing a token  $\circ$ . These two tokens allow to start the emptying of



**Fig. 3.**  $N_\pi$ , net part of  $P_\pi$  where  $type(c) = \mathbb{N}$  and  $\iota(i_2) = i.\{\bullet, \circ\}$ . Large black border of some transitions just indicates that they belong to  $cod(\rho_\pi)$ . Inscriptions on the arcs are simplified: singletons are replaced by their unique element.

- $N$ ; they are different in order to know, after the emptying is done, if  $N_\pi$  may terminate producing the exit marking (for  $\bullet$ , with transition  $t_7$ ) or should be completely emptied (for  $\circ$ , with  $t_8$ );
- transition  $t_1$  is for internal preemption: it will be synchronized with all transitions in  $N$  having a synchronous action *quit* in their label. This way,  $N$  may abort itself by firing such a transition. After  $t_1$  has fired and  $N$  has been completely emptied,  $t_7$  can fire and terminate  $N_\pi$ ;
  - transition  $t_2$  is for external abortion: it may be synchronized later on with a transition such as  $t_5$ , coming from another  $N_\pi$  in which the present one is nested. This synchronization is not yet possible on the net of figure 3, but a further renaming  $\widehat{kill}' \mapsto \widehat{kill}$  will allow it. After the emptying was completed, transition  $t_8$  can fire and fully empty  $N_\pi$ ;  $t_8$  is intended to be synchronized with a transition such as  $t_6$  and, here again coming from an external  $N_\pi$ ; it will be made possible thanks to a further renaming  $\widehat{empty}' \mapsto \widehat{empty}$ ;
  - transition  $t_3$  is similar to  $t_2$  but is used when an external abortion occurs while an internal one is already in progress. It just replaces the token  $\bullet$  in  $i_2$  with a token  $\circ$ . This corresponds to a switch from internal to external abortion mode. The priority  $(t_2, t_3)$  in  $\rho_\pi$  ensures that  $t_3$  is always preferred to  $t_2$  when both are enabled;
  - emptying is performed by transitions  $t_4$ ,  $t_5$  and  $t_6$ . We already had an intuition about the rôle of  $t_5$  and  $t_6$ : the former triggers an abortion into a  $N_\pi$  nested in  $N$ , doing this, it increments a counter handled through the links on  $c$  (this counter was initialized to zero by either  $t_1$  or  $t_2$ ); the latter

- allows such an aborted net to empty completely (by firing its transition  $t_8$ ), decrementing the counter;
- the counter in  $c$  of killed sub-nets is used to ensure that each aborted sub-net is also emptied: as stated in definition 4,  $t_4$  and  $t_5$  have the priority over  $t_7$  and  $t_8$ , but  $t_6$  has not. So, any possible firing of  $t_4$  and  $t_5$  will be done before  $t_7$  or  $t_8$  have a chance to fire, then they must wait for  $t_6$  to perform all the needed (and possible) emptying and thus to decrease the counter until zero. Finally, and not before,  $t_7$  or  $t_8$  is allowed to perform communication  $c^-(0)$ ;
- the way  $t_4$  works is not directly visible in  $N_\pi$ . Roughly speaking, for every place  $s$  in  $N$ , we will add an *emptying transition*  $t_s$  with a synchronous action *abort* in its label and an arc  $s \rightarrow t_s$  labelled with a singleton  $\{a\}$ , where  $a$  is an arbitrary variable. These transitions will be synchronized with  $t_4$  and so, the loop on  $t_4$  will be used to empty  $N$ , taking tokens one by one;
- for convenience, the type of  $c$  has been set to  $\mathbb{N}$  which leads, after unfolding, to an infinite number of places in the obtained low-level net. Fortunately, this type can easily be bounded in practice.

In order to have things working properly, we assume that actions symbols *quit*, *kill*, *kill'*, *empty*, *empty'* and *abort*, their conjugated symbols and link symbol  $c$  are reserved for operation  $\pi$ , and so are never used somewhere else.

Operation  $\pi$  relies on  $P_\pi$ : it first refines the net which should be made preemptible into  $P_\pi$  and then adds the emptying transitions (such as  $t_s$ ) as described above (and formalized below). Scoping on *quit* allows internal preemption, scoping on *abort* allows to control the emptying transitions and scoping on  $\{\widehat{kill}, \widehat{empty}\}$  allows the transmission of abortion to the potentially nested priority M-nets. A tie on  $c$  is also made so the counter can work properly. Finally, actions *kill'* and *empty'* are renamed in order to allow the result to be nested in another  $\pi$ .

**Definition 4.** *Let  $P$  be a priority M-net. Then,*

$$\pi(P) = \left[ \left\{ \widehat{abort}, \widehat{quit}, \widehat{kill}, \widehat{empty} \right\} : Ab(P_\pi[\mathcal{X} \leftarrow P]) \text{ tie } c \right] \\ \left[ \widehat{kill}' \mapsto \widehat{kill}, \widehat{empty}' \mapsto \widehat{empty} \right]$$

where  $P_\pi$  is the priority M-net defined above and  $Ab$  is an auxiliary operation which includes the additional emptying transitions; if  $P_\pi[\mathcal{X} \leftarrow P] = P' = ((S', T', \iota'), \rho')$ , then  $Ab(P') = ((S'', T'', \iota''), \rho'')$  with:

- $S'' = S'$ , and  $\forall s \in S'' : \iota''(s) = \iota'(s)$ ;
- $T'' = T' \uplus T_s$  where  $T_s = \{t_s \mid s \in S' \setminus \{x\} \wedge s^\bullet \cap \text{cod}(\rho') = \emptyset\}$
- and  $\forall t \in T'' : \iota''(t) = \begin{cases} \iota'(t) & \text{if } t \in T', \\ \{\widehat{abort}\} \cdot \emptyset \cdot \emptyset & \text{if } t \in T_s; \end{cases}$
- $\forall (t, s) \in T'' \times S'' : \iota''(t, s) = \begin{cases} \iota'(t, s) & \text{if } t \in T', \\ \emptyset & \text{if } t \in T_s; \end{cases}$

$$\begin{aligned}
& - \forall (s, t) \in S'' \times T'' : \iota''(s, t) = \begin{cases} \iota'(s, t) & \text{if } t \in T', \\ \{a\} \subset \text{Var} & \text{if } t = t_s \in T_s, \\ \emptyset & \text{if } t \in T_s \setminus \{t_s\}; \end{cases} \\
& - \rho'' = \rho' \uplus \{(t, t_s) \mid t_s \in T_s \wedge t \in (\bullet t_s)^\bullet\}.
\end{aligned}$$

Let us make two remarks about this definition:

- places which are input places for transitions belonging to  $\text{cod}(\rho)$  (if  $P = (N, \rho)$ ) are left untouched; the reason is that such places are already “under control”, *i.e.*, belong to a nested emptying mechanism. We do not need additional emptying transitions for them;
- the added emptying transitions *are always* synchronized with transition  $t_4$  in  $N_\pi$ , whose enabling is well controlled. So, net  $\pi(P)$  never empties in an uncontrolled way. Another consequence is that the only non T-restricted transitions are those such as  $t_2$ ,  $t_3$  and  $t_8$ .

#### 4.5 P/M-nets

We are now in position to define *preemptible M-nets* (*P/M-nets*). They are defined as a sub-class of priority M-nets with some structural constraints. This sub-class happens to be reasonably wide (see section 6) and sound with respect to the semantics of preemption (see section 4.6).

**Definition 5.** *Let  $P = (N, \rho)$  be a priority M-net.  $P$  is a P/M-net iff either:*

- $N$  is an *ex-good* M-net and  $\rho = \emptyset$ , or;
- $P$  is defined as  $\pi(P_1)$ ,  $P_1[\mathcal{X} \leftarrow P_2]$ ,  $P_1 \parallel P_2$ ,  $P_1; P_2$ ,  $P_1 \square P_2$ ,  $[P_1 * P_2 * P_3]$ ,  $P_1 \text{ sy } A$ ,  $P_1 \text{ rs } A$ ,  $[A : P_1]$ ,  $P_1 \text{ tie } b$  or  $P_1[f]$ , where  $P_i$ , for  $i \in \{1, 2, 3\}$ , are P/M-nets,  $\mathcal{X}$  is a hierarchical symbol,  $A$  is a synchronous communication symbol,  $b$  is an asynchronous link symbol, and  $f$  is a renaming function on communication symbols.

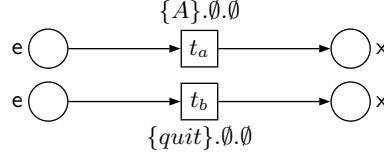
**Definition 6.** *A P/M-net  $(N, \rho)$  is said: valid if  $N$  is an ex-good M-net; finite if  $\mathcal{U}(N)$  is finite (in number of places and transitions); 1-safe if  $\mathcal{U}(N)$  is 1-safe.*

#### 4.6 Soundness

In order to define the soundness of a P/M-net  $P = (N, \rho)$  with respect to the semantics of preemption, we split the set  $T$  of transitions of  $N$  in four disjoint parts:  $T = T_r \uplus T_t \uplus T_i \uplus T_p$ .

If  $P = \pi(P')$ , then,  $T_r$  ( $r$  is for *root*) is the set of transitions involved in an abortion of  $P$  at the top level. In other words,  $T_r$  contains all the transitions coming from a synchronization with  $t_4$ ,  $t_5$  or  $t_6$  in  $N_\pi$  used in  $\pi(P')$ .  $T_i$  (*internal*) contains the same kind of transitions, but issued from a possible  $\pi$  nested in  $P'$ .  $T_t$  (*termination*) contains only transition  $t_7$  coming from  $P_\pi$  in  $\pi(P')$ . All the remaining transitions are in  $T_p$  (*program*). If  $P \neq \pi(P')$ , we have  $T_r = T_t = \emptyset$ .

Definition 7 below formalizes the intuition of the expected behavior of a preemptible M-net  $P$ . Actually,  $P$  can evolve in one of the following ways:



**Fig. 4.** Net part of P/M-net  $P_1 = (N_1, \rho_1)$  with  $\rho_1 = \emptyset$ . All the places of  $N_1$  have type  $\{\bullet\}$ , and arcs should be annotated  $\{\bullet\}$ .

- $P$  runs without any abortion. In this case, it fires only transitions from  $T_p$ ;
- $P$  runs as before by firing transitions in  $T_p$  until some abortions occur in some nested  $\pi$ 's. In this case, all the next fired transitions are in  $T_i \uplus T_p$ ;
- $P = \pi(P')$  and it completely aborts at a point of its execution. In this case, it starts like in the previous case, until it begins aborting and thus only fires transitions from  $T_r$  or from  $T_i$  because of some possible abortions transmitted to nested  $\pi$ 's. After the end of the abortion, it may fire  $t_7$  from  $T_t$  to produce its exit marking.

**Definition 7.** Let  $P$  be a P/M-net.  $steps(P)$  is said sound with respect to the semantics of preemption iff each  $D \in steps(P)$  is of the form  $\delta_1 \dots \delta_n \delta'_1 \dots \delta'_m \delta''$  or  $\delta_1 \dots \delta_n \delta'_1 \dots \delta'_m$  if  $P = \pi(P')$ , otherwise  $D = \delta_1 \dots \delta_n$ , where for all  $j$ ,  $\delta_j \in \mathcal{M}_f(T_i \uplus T_p)$ , for all  $k$ ,  $\delta'_k \in \mathcal{M}_f(T_i \uplus T_r)$ , and  $\delta'' = \{t_7 \in T_t\}$ .

By induction on the algebraic structure, we get that what we structurally defined through P/M-nets is what we expected as a model preserving the semantics of preemption.

**Theorem 2.** Let  $P$  be a P/M-net. Its step semantics,  $steps(P)$ , is sound with respect to the semantics of preemption.

## 5 A detailed example

This section gives an example of P/M-net. We show the construction of the P/M-net

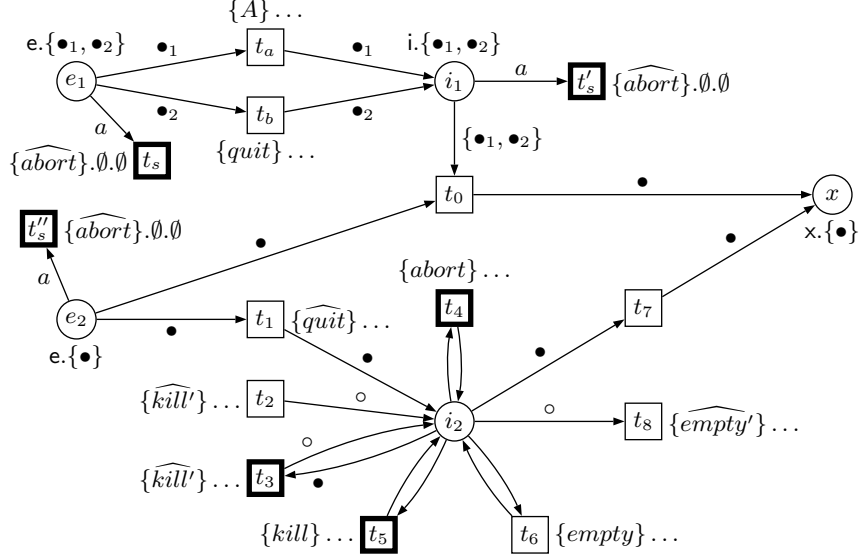
$$P = \pi(\pi(P_1) \parallel P_2)$$

where  $P_1$  is a P/M-net with two parallel transitions  $t_a$ , labelled  $\{A\}.\emptyset.\emptyset$ , and  $t_b$  labelled  $\{quit\}.\emptyset.\emptyset$  (see figure 4).  $P_2$  is a net from which we will just consider one transition  $t_c$  having one internal input place  $i_c$ ; we assume that this net is valid, well defined and does not use  $\pi$  in its construction (this would demonstrate transmission of abortion but, for the sake of simplicity, we prefer to show this feature with  $P_1$  only).  $P_2$  is schematized in figure 5. This example allows us to illustrate internal and external abortion as well as the propagation of abortion to nested  $\pi$  operations. In the following, most annotations are omitted from figures in order to keep them as readable as possible.

First, let us look carefully at the P/M-net produced by  $\pi(P_1)$  whose construction is detailed in figures 6 to 8. The first step for building  $\pi(P_1)$  is to refine



**Fig. 5.** The scheme for  $P_2$ , its priority relation is  $\rho_2 = \emptyset$ .



**Fig. 6.** Net part of  $P_\pi[\mathcal{X} \leftarrow P_1]$  with the added emptying transitions. See figure 3 for the detailed annotations.

$P_1$  into  $P_\pi$  and to add the emptying transitions; the net part of the result is shown in figure 6 and the corresponding priority relation is

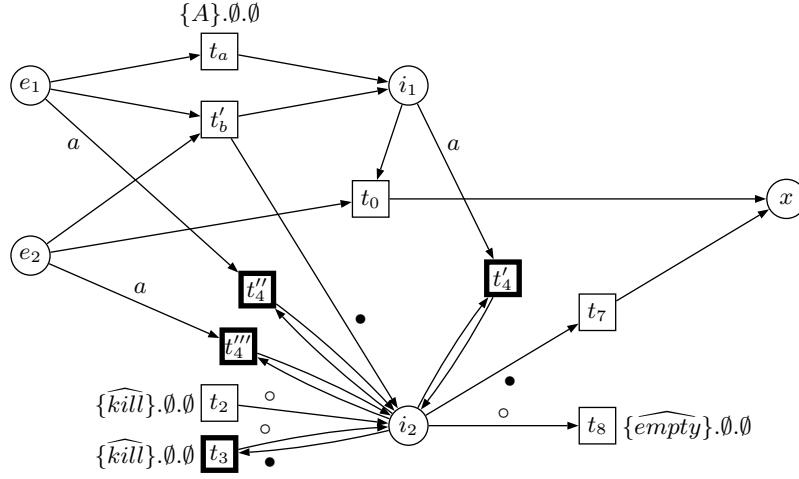
$$\rho'_1 = \{(t_7, t_4), (t_8, t_4), (t_7, t_5), (t_8, t_5), (t_a, t_s), (t_b, t_s), (t_0, t'_s), (t_a, t_5), (t_b, t_5), (t_2, t_3)\}.$$

The next step consists in performing the scoping over  $\widehat{kill}$  and  $\widehat{empty}$ , making asynchronous links over  $c$ , and renaming  $\widehat{kill}'$  into  $\widehat{kill}$  and  $\widehat{empty}'$  into  $\widehat{empty}$ , as stated in definition 4. The result is sketched in figure 7 (most annotations are omitted as well as place  $s_c$  for asynchronous links on  $c$ ). Transitions  $t_b$  and  $t_1$  yield a new transition  $t'_b$  whose firing starts internal abortion stage. The synchronization of  $t_4$  with the added emptying transitions yield  $t'_4$  and  $t''_4$ . Transitions  $t_5$  and  $t_6$  are removed because they are nothing to synchronize with (it will not be the case later in our example, when we will consider the nesting of  $\pi(P_1)$  into another  $\pi$ ). The priority relation of  $\pi(P_1)$  is

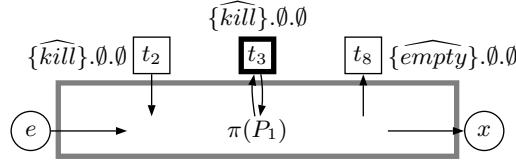
$$\rho''_1 = \{(t_a, t''_4), (t'_b, t''_4), (t_7, t''_4), (t_8, t''_4), (t_0, t'_4), (t_7, t'_4), (t_8, t'_4), (t_2, t_3)\}.$$

One can observe that everything works right when this net is started from its entry marking: if  $t'_b$  (internal abortion) or  $t_2$  (external abortion) fires, then the





**Fig. 7.** Simplified  $\pi(P_1)$ . Transitions labels are  $\emptyset.\emptyset.\emptyset$ , except on  $t_a$ ,  $t_2$ ,  $t_3$  and  $t_8$ .



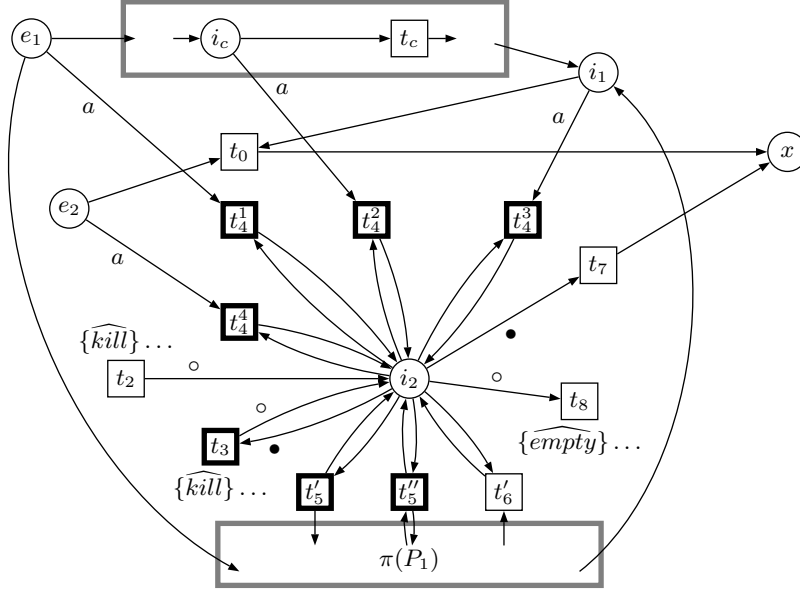
**Fig. 8.** A *really* simplified representations of  $\pi(P_1)$ .

net is correctly emptied, whenever  $t_a$  did fire or not. Similarly, if  $t_3$  fires after  $t'_b$  did. The difference between external and internal abortion appears at the very end: for internal abortion,  $t_7$  fires, producing the exit marking; for external abortion, only  $t_8$  can fire. We will see later in the example how  $t_2$ ,  $t_3$  and  $t_8$  are synchronized in order to achieve a correct transmission of abortion when  $\pi(P_1)$  is nested into another  $\pi$ .

The communication transitions, related to the preemption and visible from the outside of  $\pi(P_1)$ , are  $t_2$ ,  $t_3$  and  $t_8$ . So, in the following,  $\pi(P_1)$  will be schematized as shown in figure 8. (Notice that we show only one entry place while there are actually two, this has no other consequence than simplifying the figures.)

In order to finish our example, we have to put  $\pi(P_1)$  in parallel with  $P_2$  and to apply  $\pi$  on the result. We get the P/M-net depicted (in a simplified version) in figure 9. The net resulting from  $\pi(P_1) \parallel P_2$  is splitted in two parts, the part coming from  $P_2$  corresponds to the gray box in the top and the part which comes from  $\pi(P_1)$  corresponds to the box in the bottom. The middle part, which is not boxed, comes from the last application of  $\pi$ . For this final P/M-net, the priority relation is  $\rho = \{(t_c, t_4^2), (t_7, t_4^2), (t_8, t_4^2), (t_1, t_4^3), (t_7, t_4^3), (t_8, t_4^3), (t_7, t_5'), (t_7, t_5''), \dots\}$ .

We can see that an abortion has three effects: one is the emptying of  $P_2$  with transition  $t_4^2$  (there would be much more transitions like  $t_4^2$  if  $P_2$  would be larger),



**Fig. 9.** A simplified representation of the P/M-net resulting from  $\pi(\pi(P_1) \parallel P_2)$ .

one other is the emptying of entry and exit places for the parallel composition  $\pi(P_1) \parallel P_2$  (with transitions  $t_4^1$  and  $t_4^3$ ), and the last is the propagation of the emptying to  $\pi(P_1)$  (with transition  $t_5'$  or  $t_5''$ ). When this sub-abortion is done and when all  $P_2$  is empty (which can be done in any order, even in parallel),  $t_6'$  can fire, ending the emptying of  $\pi(P_1)$ .

We can also observe that suspension is immediately effective. Thanks to the definition of  $\rho_\pi$ ,  $t_c$  is suspended as soon as  $t_4^2$  is enabled (and it is disabled after  $t_4^2$  fired). Similarly, transitions in  $\pi(P_1)$  are first suspended as long as  $t_5'$  and  $t_5''$  are enabled and, when of them one fires, the suspension is done internally because  $t_4'$ ,  $t_4''$  and  $t_4'''$  in  $\pi(P_1)$  are then enabled (see figure 7).

In this final net, transition  $t_7$  is dead because the net never uses internal abortion; even if sub-net  $\pi(P_1)$  does, it is not the case for  $\pi(\pi(P_1) \parallel P_2)$  as a whole.

Asynchronous links on  $c$  (even if not shown in the figures) ensure that the propagation of abortion to  $\pi(P_1)$  is always followed by its complete emptying, through a firing of transition  $t_6'$  (see figure 9).

Let us conclude this example noticing that the net we obtain is ready to be nested in another  $\pi$ , like  $\pi(P_1)$  was, thanks to its own transitions  $t_2$ ,  $t_3$  and  $t_8$ . If no such nesting is to be made, the net should be restricted over *kill* and *empty*, resulting in a valid P/M-net. This restriction would be necessary, not only for validness, but also in order to avoid a spontaneous firing of transition  $t_2$  or  $t_3$  which would result in an unexpected abortion of the net.

## 6 Properties of P/M-nets and links with existing works

Let us observe first that a P/M-net  $(N, \rho)$  which has been constructed without operation  $\pi$  is always valid and has  $\rho = \emptyset$ . This property states that the introduced extension is conservative and does not disturb the existing model if one makes no usage of operation  $\pi$ . This is the reason why we consider P/M-nets as a “reasonably wide” model: it contains M-nets which already proved being useful.

However, in general, a P/M-net  $P = (N, \rho)$  can have some crippled transitions. It turns out that they can easily be identified by their synchronous label  $\{\widehat{empty}\}$  or  $\{\widehat{kill}\}$ . These transitions belong to the communication interface of  $P$  and are crucial in order to nest  $P$  in another operation  $\pi$ , *e.g.*, in  $P' = \pi(\dots P \dots)$ . Operation  $\pi$ , thanks to the scoping on *kill* and *empty*, allows  $P'$  to abort its preemptible parts such as  $P$ . However, if  $P$  is not to be nested in an operation  $\pi$ , then these crippled transitions should be removed. Actually, in that case, the P/M-net of interest is  $\mathbf{Prs}\{kill, empty\}$ . Moreover,  $\mathbf{Prs}\{kill, empty\}$  is valid; this property is important because it shows that even if our modeling needs to relax the T-restrictness of some transitions, the final result can always be T-restricted.

Valid P/M-net semantics may still appear as somehow unsatisfactory, because of the use of priorities. It turns out that some results in the field of semantics of priority systems may be applied for P/M-nets. In [6], the authors define a transformation of a finite 1-safe Petri net  $\Sigma$ , equipped with a pairwise priority relation  $\rho$ , into a bounded Petri net which retains as much as possible of the concurrency of  $(\Sigma, \rho)$ . In this context, *as much as possible* means that only semantics composed of *consistent steps* are considered (see [6, section 3]). This result can be directly applied to the unfolding of a valid, finite and 1-safe P/M-net. (One can see that if  $P$  is 1-safe, so is  $\pi(P)$ .) Then, applying the result from [7], the obtained bounded Petri net can be transformed into a 1-safe Petri net which has the same pomset semantics (partially ordered multi-sets semantics), and we can state:

**Proposition 1.** *Let  $P$  be a valid, finite and 1-safe P/M-net. Then,  $P$  can be transformed into a low-level 1-safe Petri net having the same consistent step semantics.*

So, P/M-nets can be transformed, in most of reasonable cases (*i.e.*, finite ones), into 1-safe Petri nets having an equivalent concurrent semantics. However, the construction given in [7] leads to really huge nets and so is not intended to be used in practice. Nevertheless, the above proposition is important since it means that 1-safe Petri nets are expressive enough to model preemption with a concurrent semantics.

In practice, it should be possible to modify the existing model checker of PEP [13, 11] in order to have it dealing with priorities. The model checker relies on finite prefixes computation: it develops the branching process semantics of the analyzed low-level net until it finds a maximal cut. (This is always the case in finite 1-safe Petri nets since the number of states is finite.) Under such

conditions, the influence of priorities should be to prune some branches in the computation. For two enabled transitions  $t_1$  and  $t_2$ , the existing algorithm would build one branch starting with a firing of  $t_1$  and another for  $t_2$ . With priorities, if  $(t_1, t_2) \in \rho$ , the branch starting with  $t_1$  does not have to exist anymore. This does not mean that the computation would be more efficient; actually, the contrary should hold: adding priorities would force the examination of much more cases in order to take into account some possibly disabled transitions, because of priorities. However, the model checking remains decidable.

## 7 Conclusion

We presented what is, to the best of our knowledge, a first attempt to provide a fully compositional model of Petri nets with a preemption operator, with a concurrent semantics. Our construction is based on M-nets which are extended with priorities, structured into an algebra and structurally restricted leading to *preemptibles M-nets (P/M-nets)*. The P/M-net algebra is similar to the M-net algebra, but having an additional operation  $\pi$ , which transforms any P/M-net into its preemptible equivalent.

We show that P/M-nets can be considered as a high-level version of so called *priority systems* (as defined in [6]) by defining an unfolding operation which transforms a P/M-net into a low-level Petri net having priorities between its transitions. Thus, applying results from [6] and [7], any reasonable P/M-net can be transformed into a 1-safe Petri net (without priorities) which retains as much as possible of the concurrency present in the P/M-net. This transformation leads to enormous nets and is not tractable in practice, but it shows that 1-safe Petri nets are powerful enough to model preemption with a concurrent semantics.

The presented work has been already applied for giving the semantics of some preemption-related extensions of the parallel programming language  $B(PN)^2$ , introducing abortable blocks, treatment of exceptions, a generalized timeout construct and a small Unix-like process manager.

## Acknowledgments

We are very grateful to all the participants of the three last PORTA/BAT workshops for their remarks on this work and especially for their thoughts about model checking nets with priorities. Particular thanks go to Raymond Devillers.

## References

- [1] G. Berry. The Foundations of Esterel. Language and Interaction: Essays in Honour of Robin Milner, G. Plotkin, C. Stirling and M. Tofte, editors, MIT Press, 1998.
- [2] E. Best, R. Devillers, and J. Esparza. General refinement and recursion operators for the Petri box calculus. *LNCS 665:130–140*, 1993.
- [3] E. Best, R. Devillers, and J. G. Hall. The box calculus: a new causal algebra with multi-label communication. *LNCS 609:21–69*, 1992.

- [4] E. Best, W. Fraczak, R. Hopkins, H. Klaudel, and E. Pelz. M-nets: An algebra of high-level Petri nets, with an application to the semantics of concurrent programming languages. *Acta Informatica*, 35, 1998.
- [5] E. Best and R. P. Hopkins.  $B(PN)^2$  — A basic Petri net programming notation. PARLE'93, *LNCS 694:379–390*, 1993.
- [6] E. Best and M. Koutny. Petri net semantics of priority systems. *Theoretical Computer Science*, 96(1):175–215, 1992.
- [7] E. Best and H. Wimmel. Reducing  $k$ -safe Petri nets to pomset-equivalent 1-safe Petri nets. ICATPN'2000, to appear.
- [8] R. Devillers, H. Klaudel, and R.-C. Riemann. General refinement for high-level Petri nets. FST&TCS'97, *LNCS 1346:297–311*, 1997.
- [9] H.J. Genrich, K. Lautenbach, and P.S. Thiagarajan. Elements of General Net Theory. In W. Brauer, editor, *Net Theory and Applications*, Proceedings of the Advanced Course on General Net Theory of Processes and Systems, *LNCS 84:21–163*, 1980.
- [10] R. Devillers, H. Klaudel, and R.-C. Riemann. General parameterised refinement and recursion for the M-net calculus. *Theoretical Computer Science*, to appear.
- [11] J. Esparza. Model checking using net unfoldings. *Science of Computer Programming*, 23(2–3):151–195, 1994.
- [12] H. Fleishhack and B. Grahlmann. A Petri net semantics for  $B(PN)^2$  with procedures. PDSE'97, Boston, 1997. IEEE Computer Society.
- [13] B. Grahlmann and E. Best. PEP — more than a Petri net tool. *LNCS 1055*, 1996.
- [14] N. Halbwachs. Synchronous Programming of Reactive Systems. Kluwer Academic Publishers, 1993.
- [15] H. Klaudel. Compositional high-level Petri net semantics of a parallel programming language with procedures. *Science of Computer Programming*, to appear.
- [16] H. Klaudel. Parametrized M-expression semantics of parallel procedures. DAP-SYS'00, to appear.
- [17] H. Klaudel and F. Pommereau. Asynchronous links in the PBC and M-nets. ASIAN'99, *LNCS 1742:190–200*, 1999.
- [18] H. Klaudel and R.-C. Riemann. High level expressions with their SOS semantics. CONCUR'97, *LNCS 1243:288–301*, 1997.
- [19] H. Klaudel and R.-C. Riemann. Refinement-based semantics of parallel procedures. PDPTA'99, volume 4. CSREA Press, 1999.
- [20] J. Lilius and E. Pelz. An M-net semantics of  $B(PN)^2$  with procedures. *ISCIS*, volume 1, 1996.
- [21] R. Milner. A calculus of communicating systems. *LNCS 92*, 1980.
- [22] T. Vesper and M. Weber. Automatisches verteiltes rücksetzen. In K. Spies and B. Schätz, editors, *Proceedings of the workshop “Formale Beschreibungstechniken für verteilte Systeme”*, number 9 in GI/ITG Fachgespräch, Munich, 1999.
- [23] M. Kishinevsky, J. Cortadella, A. Kondratyev, L. Lavagno, A. Taubin and A. Yakovlev. Coupling asynchrony and interrupts: place chart nets and their synthesis. ICATPN'97, *LNCS 1248:328–347*, 1997.