Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 174 (2007) 83–98
www.elsevier.com/locate/entcs

Coding Mobile Synchronizing Petri Nets into Rewriting Logic*

Fernando Rosa-Velardo

Departamento de Sistemas Informáticos y Programación
Universidad Complutense de Madrid, Spain

e-mail: fernandorosa@sip.ucm.es

Abstract

Mobile Synchronizing Petri Nets (MSPN's) are a model for mobility and coordination based on coloured Petri Nets, in which systems are composed of a collection of (possibly mobile) hardware devices and mobile agents, both modelled homogenously. In this paper we approach their verification, for which we have chosen to code MSPN's into rewriting logic. In order to obtain a representation of MSPN systems by means of a rewrite theory, we develop a class of them, that we call ν-Abstract Petri nets (ν-APN's), which are easily representable in that framework. Moreover, the obtained representation provides a local mechanism for fresh name generation. Then we prove that, even if ν-APN's are a particular class of MSPN systems, they are strong enough to capture the behaviour of any MSPN system. We have chosen Maude to implement ν-APN's, as well as the translation from MSPN's to ν-APN's, for which we make intensive use of its reflective features.

Keywords: Mobility, Petri nets, rewriting, security, specifications

1 Introduction

In several previous papers [14,15] we have presented a model for concurrent, mobile and ubiquitous systems [5,12], based on Petri Nets, that we call *Mobile Synchronizing Petri Nets* (MSPN). Petri nets provide friendly graphical representations of systems, and we can profit from their solid theoretical background. Moreover, since they are not Turing-complete, there are many decidability results for them that do not hold in general for other more expressive models. Several models for mobility based on Petri Nets have been proposed in the literature [1,19,9,3,8]. However, the previous models do not consider security aspects, certainly crucial in this setting, nor any other Petri net based model for mobility, up to our knowledge.

* Work partially supported by the Spanish projects MIDAS TIC 2003-01000, MASTER TIC 2003-07848-C02-01 and PROMESAS-CAM S-0505/TIC/0407.

doi:10.1016/j.entcs.2006.10.021

In [15] we studied the expressiveness of MSPN systems. We proved several interesting decidability results, such as the decidability of coverability, that can be used to specify security properties such as integrity or confidentiality. In order to achieve the verification of these properties, we have chosen rewriting logic [10] as the framework to develop it. This logic supports in a natural way the managing of distributed and concurrent systems, and has been efficiently implemented by the Maude [6] programming language, that has been widely used for the implementation of a number of process algebras and other formalisms for concurrency [20,18].

In order to perform the translation from MSPN systems to rewriting logic, we introduce ν-Abstract Petri nets (ν-APN's), that provide an intermediate step in the procedure. They are essentially a subclass of coloured Petri nets that can produce fresh identifiers. These nets capture the core of MSPN systems. Any time a name is created in a MSPN system, *any* different new name could have been created, so that reachability of a certain marking is equivalent to that of any marking in which we (consistently) rename the set of new names. However, for the sake of homogeneity, we will assume that we can rename any name, even those that were present in the initial marking. To capture this intuition, we assume that ν-APN's work module α-conversion of tokens. We represent them as rewrite theories in such a way that generation of fresh names can be achieved without needing to access the global state.

Since the rewrite theory that results from the translation of an MSPN system to a ν-APN first, and then to rewriting logic, can be quite distant from its original description, we use the reflective properties of Maude in order to automatically perform that translation. Finally, we implement an algorithm that decides the coverability problem for MSPN systems.

The remainder of the paper is structured as follows. Section 2 gives an informal description of Mobile Synchronizing Petri Nets. Section 3 describes in a nutshell the representation of Petri nets in Maude. Section 4 defines Abstract Petri Nets and ν-Abstract Petri Nets, establishes their translation to rewrite theory and proves the equivalence of MSPN systems and ν-APN's. Finally, in Section 5 we briefly describe the implementation in Maude and Section 6 presents the conclusions and directions for further work.

2 Mobile Synchronizing Petri Nets: overview

In this section we briefly describe our Mobile Synchronizing Petri Nets. For more details see [14] or [15]. An MSPN $N = (P, T, F, \lambda, C)$ is a special kind of labelled coloured Petri Net [7], that is, P is a finite set of places, T is a finite set of transitions, and F is a partial function that defines as its domain the set of arcs, and labels those arcs with variables taken from a set $Var = Var_{\mathcal{L}} \cup Var_{Id} \cup \{\varepsilon\}$. We say that some variables in Var_{Id}, those in $Var_{Auth} \subset Var_Id$, are authentication variables. MSPN's only have three different colour types: one for localities, taken from a set $\mathcal{L}$, one for identifiers, taken from a set Id, with both $\mathcal{L}$ and Id infinite, and a singleton colour type $\{\bullet\}$ for ordinary black tokens. Sometimes we just write $\bullet$

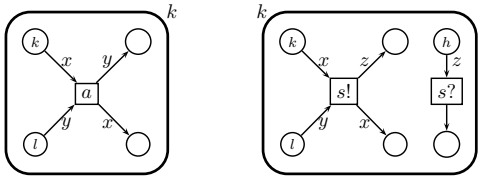

Figure 1. Autonomous (left) and synchronizing (right) transitions

to denote that singleton. We use the symbol $\mathcal{T}$ to range over the set $\{\bullet, \mathcal{L}, Id\}$, and *Tokens* to denote the union $\mathcal{L} \cup Id \cup \{\bullet\}$. The function $C : P \to \{\bullet, \mathcal{L}, Id\}$ establishes a partition in the set of places, so that a place p with $C(p) = \mathcal{T}$ may only contain tokens in $\mathcal{T}$. Finally, according to $\lambda : T \to \mathcal{A} \cup Sync$, MSPN's may have two different kinds of transitions, autonomous (those with $\lambda(t) \in \mathcal{A}$) and synchronizing transitions (those with $\lambda(t) \in Sync$). The set $\mathcal{A}$ of autonomous labels has two distinguished labels *new* and *go*. The set *Sync* of synchronizing labels is the disjoint union of $S? = \{s? \mid s \in S\}$ and $S! = \{s! \mid s \in S\}$, where S is a set of service names. Intuitively, $s!$ is the offer of a service s, while $s?$ is the request of that service, although formally they are just the two symmetric sides of a synchronization.

Unlike for ordinary Coloured Petri Nets, where arbitrary expressions over some syntax can label arcs, we only allow variables to specify the flow of tokens from preconditions to postconditions. In particular, this means that only equality of identifiers can be imposed by matching, but not any other relation between them.

For consistency we are assuming in the definition that every arc (every pair (p, t) or (t, p) in the domain of F) is labelled by a variable. However, since we only need variables to distinguish between different locality tokens and identifier tokens, we introduce the special variable ε, that labels every arc that is adjacent to an ordinary black-token place and is not usually depicted. Moreover, variables from $Var_{\mathcal{L}}$ are only used for arcs that are adjacent to places p with $C(p) = \mathcal{L}$ and those from Var_{Id} only for arcs next to places p with $C(p) = Id$. In this way, we guarantee that the different types of tokens are never mixed.

We use $post(t)$ to denote the set of variables in arcs going from t to some place, i.e., going out of t (except for ε). Analogously, we use $pre(t)$ to denote the set of variables in arcs reaching t. We take $Var(t) = post(t) \cup pre(t)$. If t is an autonomous transition with $\lambda(t) \neq new$ then it must be the case that $post(t) \subseteq pre(t)$, so that autonomous transitions can only move or delete locality and identifier tokens, but not create them. As usual in P/T nets, we denote by $t^\bullet$ and ${}^\bullet t$ the set of postconditions and preconditions of t, respectively.

Then, an MSPN system $\mathcal{S}$ is just a pair $(\mathcal{N}, \mathcal{M})$, where $\mathcal{N}$ is a set of disjoint nets and $\mathcal{M}$ is the initial marking of $\mathcal{N}$. A marking of $\mathcal{N}$ is a pair (M, loc), where M is a function that maps each place to a finite multiset of tokens, and $loc : \mathcal{N} \to \mathcal{L}$ maps each net to its current location, taken from the set $\mathcal{L}$. Given two markings $\mathcal{M}_1 = (M_1, loc_1)$ and $\mathcal{M}_2 = (M_2, loc_2)$, we say that $\mathcal{M}_2$ covers $\mathcal{M}_1$ if $M_1(p) \subseteq M_2(p)$, for every $p \in P$, and $loc_1(N) = loc_2(N)$, for all N in $\mathcal{N}$.

Since our nets are a particular class of coloured nets [7], their transitions fire relative to a *mode*, that chooses the particular tokens taken from the precondition

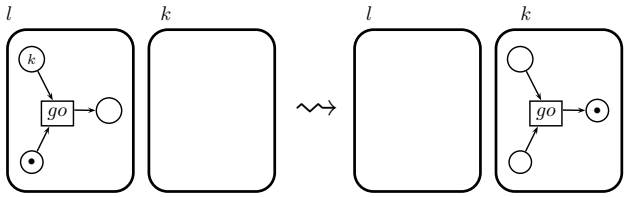

Figure 2. Movement transitions

places. Modes are defined as mappings from $Var(t)$ to $Tokens$, assigning values in $\mathcal{T}$ to variables in $Var_{\mathcal{T}}$. We denote modes by $\sigma, \sigma', \sigma_1, \sigma_2, \ldots$

Autonomous transitions t with $\lambda(t) \notin \{new, go\}$ work as the ordinary transitions in coloured nets (see Fig. 1 left). Movement transitions, those labelled by go, are autonomous transitions that change the location of the net firing it. For that purpose, every movement transition has a single distinguished locality precondition to specify the destination of the net (see Fig. 2). Name-creating transitions, those labelled by new, are autonomous transitions that, when fired, generate a fresh identifier in its identifier postconditions.

Instead, the firing of a synchronizing transition needs the presence of a *compatible* transition in the same location, that will be fired at the same time. For a pair of synchronizing transitions t_1 and t_2 we denote by $post(t_1, t_2) = post(t_1) \cup post(t_2)$, $pre(t_1, t_2) = pre(t_1) \cup pre(t_2)$ and $Var(t_1, t_2) = post(t_1, t_2) \cup pre(t_1, t_2)$. The compatibility conditions are merely syntactical: On the one hand, their labels must be complementary, $s?$ and $s!$ for some $s \in S$; On the other hand, the pair of transitions together must meet the same constraint imposed on autonomous transitions, that is, $post(t_1, t_2) \subseteq pre(t_1, t_2)$ (see Fig. 1 right); Finally, whenever an authentication variable appears in a precondition arc, then it must also appear in a precondition arc of its compatible transitions; This is the way the mechanism for authentication is implemented, by forcing the matching of two identifers.

In order to fire a pair of compatible synchronizing transitions, t_1 and t_2, they must be co-located and separately fireable according to the ordinary firing rule, but relative to a common mode σ, that in the case of synchronizing transitions are mappings from $Var(t_1, t_2)$ to $Tokens$.

In order to have a more compact notation we use $u, u', u_1, u_2, \ldots$ to range both over autonomous transitions and pairs of compatible synchronizing transitions, thus writing $\mathcal{M}[u(\sigma)\rangle\mathcal{M}'$ if $\mathcal{M}'$ is the reached state after the firing of u with mode σ.

3 Maude and Petri Nets

In rewriting logic and Maude the state of a system is formally specified by means of an equational specification in *membership equational logic* [4]. In this logic we can define sorts, subsorts, constructor operators (that can have associated equational attributes such as commutativity or associativity), or equations between terms, to name a few of the elements of this logic.

The following functional module of Maude defines the syntax of a consumer-producer system where producers can be either idle or ready to send an item,

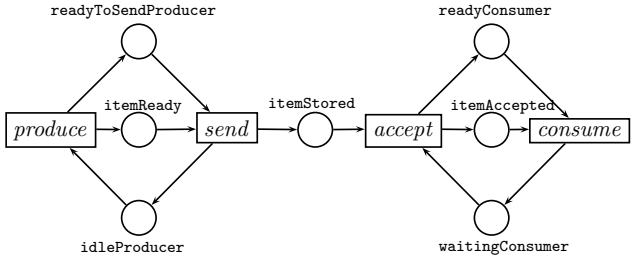

Figure 3. Petri Net modelling a producer-consumer system

consumers can be waiting for a product or ready for their consumption and items can be ready for departure, stored or accepted for consumption.

```
fmod CONSUMER-PRODUCER-SIGNATURE is
  sorts ProducerState ConsumerState ItemState State .
  subsorts ProducerState ConsumerState ItemState < State .
  op __ : State -> State [assoc comm id: null] .
  op null : -> State .
  ops idleProducer readyToSendProducer : -> ProducerState .
  ops waitingConsumer readyToConsume : -> ConsumerState .
  ops itemReady itemStored itemAccepted : -> ItemState .
endfm
```

Equations are assumed to produce confluent and terminating rewriting systems, so that they can be used from left to right to obtain unique (modulo the operational attributes) normal forms representing terms. In our case, we will see that our equational specifications will not need equations, since our set of terms can be obtained as the term algebra generated by the constructors, modulo some equational attributes.

The dynamic part of a system is specified by rewrite rules. In general these rules can be conditional, but in our case they will just have the form $t \to t'$, meaning that whenever a part of the system matches t then that part can be replaced by the corresponding instance of t'.

The next module specifies the behaviour of the producer-consumer system.

```
mod PRODUCER-CONSUMER is
  inc PRODUCER-CONSUMER-SIGNATURE .
  rl [produce] : idleProducer => itemReady readyToSendProducer .
  rl [send] : itemReady readyToSendProducer => idleProducer itemStored .
  rl [accept] : itemStored waitingConsumer => itemAccepted readyConsumer .
  rl [consume] : itemAccepted readyConsumer => waitingConsumer .
endm
```

In fact, the previous module can be seen as the rewriting semantics, written in Maude, of the Petri Net shown in Fig. 3, as stated in [17] or in [16], where the authors continued the work started in [11]. The basic idea is that each marking of a P/T net can be represented as the multiset of places where their tokens lie, so that the firing of a transition corresponds to a rewriting over those multisets.

4 Abstract Petri Nets

In order to be able to apply a similar approach to the one just discussed for plain Petri Nets in the previous section to the representation of our nets in Maude, we introduce Abstract Petri Nets (APN's). An APN is a labelled coloured Petri Net

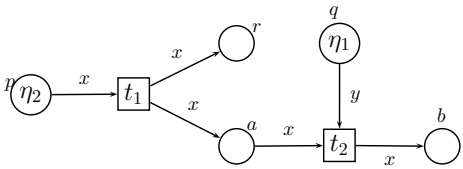

Figure 4. Example of APN

with only one colour type for identifiers, taken from an arbitrary set Id. As in MSPN systems, we only allow variables as labels of arcs (not arbitrary expressions over some syntax, as in general CPN's), that in the case of APN's, are all taken from the same set Var.

We denote by $\mathcal{MS}(A)$ the set of multisets of elements in A, that is, the set of mappings $\mathcal{A} : A \to \mathbb{N}$.

Later, we will allow also the use of a special variable that will mimic the behaviour of the *new* transitions in MSPN systems, but for now we prefer to omit it, in order to focus on the abstract nature of identifier tokens in APN's. The syntactic definition of APN's is the following.

Definition 4.1 An Abstract Petri Net (APN) is a tuple $N = (P, T, F)$ where P is the set of places, T is the set of transitions, and $F : (P \times T) \cup (T \times P) \to Var$ is a partial function such that for every $t \in T$ it holds $post(t) \subseteq pre(t)$, where ${}^\bullet t = \{p \in P \mid (p,t) \in Dom(F)\}$, $t^\bullet = \{p \in P \mid (t,p) \in Dom(F)\}$, $pre(t) = \{F(p,t) \mid p \in {}^\bullet t\}$ and $post(t) = \{F(t,p) \mid p \in t^\bullet\}$.

We also write $Vars(t) = pre(t) \cup post(t)$, by ${}^\bullet t_x$ we denote the set $\{p \in P \mid F(p,t) = x\}$ and, analogously, $t^\bullet_x = \{p \in P \mid F(t,p) = x\}$. Next we define markings of APN's and equivalent markings.

Definition 4.2 A marking M of an APN $N = (P, T, F)$ is a mapping $M : P \to \mathcal{MS}(Id)$. We define $Ids(M) = \{a \in Id \mid a \in M(p) \text{ for some } p \in P\}$. We take $\equiv_\alpha$ as the least equivalence relation on markings such that $M \equiv_\alpha M[b/a]$ with $b \notin Ids(M)$, where $M[b/a](p)(c) = M(p)(c)$ if $c \neq a, b$ and $M[b/a](p)(b) = M(p)(a)$.

We denote by $Markings(N)$, or just $Markings$ when there is no confusion, the set of markings of N. In Fig. 4 we show a simple APN. We are assuming that there are two identifiers in Id, η_1 and η_2, and two variables in Var, x and y. That net can fire transition t_1, and then t_2, producing a marking composed only by identifier η_2 in r and b.

Let us now define the operational behaviour of APN's. We denote by $+$ and $-$ the multiset union and difference, to distinguish them from $\cup$ and $\setminus$, the corresponding operations over sets. As they are a particular class of Coloured Petri Nets, their transitions are fired relative to a *mode*, that we denote by $\sigma, \sigma', \ldots$

Definition 4.3 Let t be a transition of an APN N. We say that $\sigma : Vars(t) \to Id$ is a mode for t, and denote by $Modes(t)$ the set of modes of t. We say that t is enabled with mode σ in marking M if $\sigma(F(p,t)) \in M(p)$ for all $p \in {}^\bullet t$. In that case the transition can be fired, thus producing a marking M', defined by

$M'(p) = M(p) - \{\sigma(F(p,t))\} + \{\sigma(F(t,p))\}$ for every $p \in P$ (taking $\sigma(F(a)) = \emptyset$ if $a \notin Dom(F)$). As usual, we write $M[t(\sigma)\rangle M'$.

We want to represent APN's in Maude, similar to ordinary Petri nets. For that purpose we define a *normal form* for sets of equivalent markings. Now that we have distinguishable tokens, it is not sufficient to consider the set of places occupied by tokens, but we need to do that for every different token, thus getting not a multiset of places, but a multiset of multisets of places, one multiset per identifier. However, the ordinary structure in multisets (of multisets) is not enough for our purposes. This is because we do not want to consider multisets containing the empty multiset since, intuitively, the empty multiset would correspond to a non existing identifier, which could then be removed. Instead of restricting the set of states to those multisets not containing the empty multiset, for the sake of homogeneity we prefer to proceed as follows.

Definition 4.4 Let $\backsim$ be the least equivalence relation over $\mathcal{MS}(\mathcal{MS}(P))$ such that $\mathcal{A} + \{\emptyset\} \backsim \mathcal{A}$ for $\mathcal{A} \in \mathcal{MS}(\mathcal{MS}(P))$.

In the following, we will work in $\mathcal{MS}(\mathcal{MS}(P))$ modulo $\backsim$. However, for the sake of clarity of notations, we write $\mathcal{A}$ instead of $[\mathcal{A}]_{\backsim}$ for elements in $\mathcal{MS}(\mathcal{MS}(P))/_{\backsim}$. In order to avoid confusion, sometimes we use the empty operator as union in $\mathcal{MS}(P)$ and the symbol $+$ as the union in $\mathcal{MS}(\mathcal{MS}(P))$, together with the corresponding extended operators $\prod$ and $\sum$. Moreover, we denote by $\emptyset$ the empty multiset of places and by $\{\}$ the empty multiset of multisets.

Before the definition of the mapping from markings to multisets, let us see an example. The marking shown in Figure 4 has two different identifiers, η_1 and η_2, so that we use two multisets to represent it, one for each token. The identifier η_1 appears in q, so that we represent it by $\{q\}$ and η_2 only in p, so that we represent it by $\{p\}$. Therefore, we map that marking to the multiset $\{\{p\},\{q\}\}$, that we will denote by $p + q$. After the firing of t_1 first and then t_2 a marking with only η_2 in places r and b is reached. The corresponding multiset will be simply denoted by $r\ b$.

Normal forms are defined as follows:

Definition 4.5 Let $N = (P, T, F)$ be an APN. We define

$$NF : Markings(N)/_{\equiv_\alpha} \to \mathcal{MS}(\mathcal{MS}(P))/_{\backsim}$$

as $NF([M]_{\equiv_\alpha}) = \mathcal{A}_M = \{M_a \mid a \in Ids(M)\}$, where $M_a \in \mathcal{MS}(P)$ is defined by $M_a(p) = M(p)(a)$.

In principle, the previous definition is formalized using a particular representative of the equivalence class. Let us see that any representative behaves in the same way.

Proposition 4.6 *NF is well defined and is an injection.*

Of course, the union in $\mathcal{MS}(\mathcal{MS}(P))$ is commutative and associative, so that

$(\mathcal{MS}(\mathcal{MS}(P)),+,\{\})$ is a commutative monoid. These properties are preserved in the quotient.

Proposition 4.7 *$(\mathcal{MS}(\mathcal{MS}(P))/_{\backsim},+,[\{\}]_{\backsim})$ is a commutative monoid, where $+$ is (well) defined by $[\mathcal{A}]_{\backsim}+[\mathcal{B}]_{\backsim}=[\mathcal{A}+\mathcal{B}]_{\backsim}$.*

We want to use the previous domain to represent the states of our nets, so let us see that the previous result also holds for markings. We denote by 0 the marking given by $0(p)=\emptyset$ for every p. Moreover, if M and M' are markings of an APN N, we write $M+M'$ to denote the marking defined by $(M+M')(p)=M(p)+M'(p)$. Just taking care of the details, we can export this definition to classes of markings.

Definition 4.8 Let $[M_1]_{\equiv_\alpha},[M_2]_{\equiv_\alpha}\in \mathit{Markings}/_{\equiv_\alpha}$. We define $[M_1]_{\equiv_\alpha}+[M_2]_{\equiv_\alpha}$ as $[M_1'+M_2']_{\equiv_\alpha}$ with $M_1\equiv_\alpha M_1'$, $M_1\equiv_\alpha M_1'$ and $Ids(M_1')\cap Ids(M_2')=\emptyset$.

The previous definition does not depend on any of the representatives chosen, and provides a monoid structure also for markings modulo $\equiv_\alpha$.

Proposition 4.9 *Given an APN N, $(\mathit{Markings}(N)/_{\equiv_\alpha},+,[0]_{\equiv_\alpha})$ is a commutative monoid.*

Furthermore, our translation function preserves those properties.

Proposition 4.10 *NF is an isomorphism of commutative monoids.*

Therefore, we can represent a whole class of markings, $[M]_{\equiv_\alpha}$ as the multiset (of multisets) $NF([M]_{\equiv_\alpha})$. Now let us see how to move the operational behaviour of an APN to that domain. Transitions have the form $M_1[t(\sigma_1)\rangle M_1'$. However, in MSPN systems (and ν-APN's, as we will see), tokens with new fresh names can be created. In particular, their particular names are not relevant, since any new name can be created. We have reproduced that situation in APN's by introducing abstract markings, which are induced by the considered α-equivalence. In fact, it can be seen that if $M_1\equiv_\alpha M_2$ then there is some σ_2 such that $M_2[t(\sigma_2)\rangle M_2'$, with $M_1'\equiv_\alpha M_2'$. Moreover, when working modulo α-conversion, a mode only identifies the relations between the different tokens involved, not the individual values. We formalize all that by means of the following definitions.

Definition 4.11 Let σ be a mode for t. We define the relation on variables $\sim_\sigma$ by $x\sim_\sigma y\Leftrightarrow\sigma(x)=\sigma(y)$.

The relation $\sim_\sigma$ identifies those variables that are instantiated to the same value by the mode σ. That is the information about modes in which we are interested, so that we identify modes up to that information, as formalized as follows:

Definition 4.12 Let t be a transition. We define the relation $\approx_t$ over modes of t by $\sigma_1\approx_t\sigma_2\Leftrightarrow{\sim_{\sigma_1}}={\sim_{\sigma_2}}$.

Proposition 4.13 *Let us suppose that t is fireable in M_1 with mode σ_1 and $M_1\equiv_\alpha M_2$. Then there is a mode σ_2 with $\sigma_1\approx_t\sigma_2$ such that t is fireable in M_2 with mode σ_2. Moreover, the markings obtained by firing $t(\sigma_1)$ and $t(\sigma_2)$ in M_1 and M_2, respectively, are also α-equivalent.*

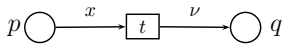

Figure 5. Simple ν-APN

Therefore, when we are working modulo α-conversion, the firings can be represented in the corresponding quotient space as $[M_1]_{\equiv_\alpha}[t([\sigma]_{\approx_t})\rangle[M_2]_{\equiv_\alpha}$. Our goal is to simulate these firings by means of rewrites $NF([M_1]_{\equiv_\alpha}) \to NF([M_2]_{\equiv_\alpha})$. Let us see which rules generate those rewrites. In fact, we will see that each pair consisting of t and $[\sigma]_{\approx_t}$ gives rise to a different rule.

First, we introduce the following auxiliary notation. Given a transition t of an APN and a mode σ for t we denote by $\mathcal{P}(t(\sigma))$ the quotient $Vars(t)/_{\sim_\sigma}$. We will treat elements in $\mathcal{P}(t(\sigma))$ as the sets of their representatives, so that we can write $Var \supseteq X \in \mathcal{P}(t(\sigma))$.

Lemma 4.14 *If $\sigma_1 \approx_t \sigma_2$ then $\mathcal{P}(t(\sigma_1)) = \mathcal{P}(t(\sigma_2))$.*

We have already settled all the machinery we need for the translation of the operational semantics of APN's into rewriting logic. In the following we denote by V the set of variables in the logic, to distinguish them from APN variables. Moreover, we will use injections $FV : \mathcal{P}(t(\sigma)) \to V$.

Definition 4.15 Let σ be a mode for t. We define $rl(t(\sigma))$ as the unconditional rule $l_{t(\sigma)} \to r_{t(\sigma)}$, where

$$l_{t(\sigma)} = \sum_{X \in \mathcal{P}(t(\sigma))} \big(\prod_{x \in X} {}^\bullet t_x\big) FV(X), \quad r_{t(\sigma)} = \sum_{X \in \mathcal{P}(t(\sigma))} \big(\prod_{x \in X} t_x^\bullet\big) FV(X)$$

Intuitively, each of the multisets appearing in $rl(t(\sigma))$ represent an identifier that is involved in the firing of $t(\sigma)$. Therefore, given $X \in \mathcal{P}(t(\sigma))$, if $\sigma(X) = \{a\}$ for some $a \in Id$, then $FV(X)$ represents the multiset of places containing a that are not affected by the firing.

Lemma 4.16 *If $\sigma_1 \approx_t \sigma_2$ then $rl(t(\sigma_1))$ and $rl(t(\sigma_2))$ are equal up to a consistent renaming of its free variables.*

If rl is a rule of a rewrite theory we write $t_1 \xrightarrow{rl} t_2$ if $t_1 \to t_2$ can be proved using the *replacement* rule of deduction exactly once, with rule rl.

Theorem 4.17 $[M_1]_{\equiv_\alpha}[t([\sigma]_{\approx_t})\rangle[M_2]_{\equiv_\alpha} \Leftrightarrow NF([M_1]_{\equiv_\alpha}) \xrightarrow{rl(t(\sigma))} NF([M_2]_{\equiv_\alpha})$.

Example 4.18 Let us see what are the rules that we obtain by applying Def. 4.15 to the net in Fig. 4. That net has two transitions, t_1 and t_2. The quotient $Modes(t_1)/_{\approx_{t_1}}$ has only one element, that can be represented as $\{\{x\}\}$. Since ${}^\bullet(t_1)_x = \{p\}$ and $(t_1)^\bullet_x = \{r, a\}$ the rule generated is $p\ M \to r\ a\ M$. However, $Modes(t_2)/_{\approx_{t_2}}$ has two elements, that we represent as $\{\{x, y\}\}$ (identifying x and y) and $\{\{x\}, \{y\}\}$ (not identifying them). Therefore, that transition produces two different rules. Since ${}^\bullet(t_2)_x = \{a\}$, ${}^\bullet(t_2)_y = \{q\}$, $(t_2)^\bullet_x = \{b\}$ and $(t_2)^\bullet_y = \emptyset$, the resulting rules are $a\ q\ M \to b\ M$ and $a\ M_1 + q\ M_2 \to b\ M_1 + M_2$, respectively.

Notice that the resulting rules are only valid because we are working module $\backsim$. Indeed, if we want to be able to use the last rule to rewrite the state $a + q$, we must

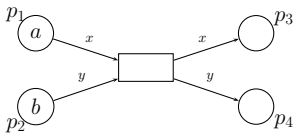

Figure 6. A simple example

instantiate both M_1 and M_2 with $\emptyset$, thus producing $b\ \emptyset + \emptyset$, which we identify via $\backsim$ with b.

So far we have dealt with APN's, which are just a particular class of coloured Petri nets that work under α-conversion. Our goal is to use APN's to simulate MSPN systems. However, for that purpose, APN's are not sufficient, due to the *new* transitions in MSPN's that produce fresh identifiers. Therefore, we need to add that primitive to APN's, thus getting ν-APN's. For that purpose, we assume that there is a special variable $\nu \in Var$, that we will only use to label arcs going from transitions to places. If we only allow modes to instantiate this variable by values that do not appear in the current marking, then we get a safe by construction mechanism of fresh name creation. Moreover, when moving to the representation of markings in $\mathcal{MS}(\mathcal{MS}(P))$, the described mechanism of name creation can be implemented in a local way, without having to explore the whole marking to guarantee the freshness of the new name or without the need of a global variable, because the new name simply corresponds to a new multiset, that at the time of the creation will only contain the place where the identifier is created.

The syntactic definition is almost the same as that of APN's, but for the consideration of the new variable.

Definition 4.19 A ν-Abstract Petri Net (ν-APN) is a tuple $N = (P, T, F)$ where P is the set of places, T is the set of transitions, and $F : (P \times T) \cup (T \times P) \to Var$ is a partial function such that for all $t \in T$ it holds $post(t) \subseteq pre(t)$, where ${}^\bullet t = \{p \in P \mid (p,t) \in Dom(F)\}$, $t^\bullet = \{p \in P \mid (t,p) \in Dom(F)\}$, $pre(t) = \{F(p,t) \mid p \in {}^\bullet t\} \setminus \{\nu\}$ and $post(t) = \{F(t,p) \mid p \in t^\bullet\} \setminus \{\nu\}$.

However, the behaviour changes as follows.

Definition 4.20 Let t be a transition of a ν-APN N. We say that $\sigma : Vars(t) \to Id$ is a mode for t if whenever $\nu \in Vars(t)$ then $\sigma(\nu) \neq \sigma(x)$ for every $x \in Vars(t)$ with $x \neq \nu$. We say that t is enabled in mode σ in marking M if:

- If $\nu \in Vars(t)$ then $\sigma(\nu) \notin Ids(M)$
- $\sigma(F(p,t)) \in M(p)$ for all $p \in {}^\bullet t$.

Then the transition can be fired, producing a marking M', defined by $M'(p) = M(p) - \{\sigma(F(p,t))\} + \{\sigma(F(t,p))\}$. Once again, we write $M[t(\sigma)\rangle M'$.

Therefore, the only difference is that now we restrict modes so that the variable ν can only be instantiated by identifiers not appearing in the marking. In particular, modes must not instantiate ν and any other variable by the same value, so that the only element in $\mathcal{P}(t(\sigma))$ to which ν can belong is $\{\nu\}$.

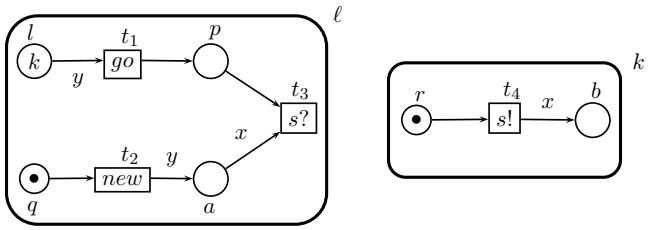

Figure 7. MSPN system

All the previous considerations about the representation of an APN as a rewrite theory are still valid if we extend our mappings FV to $FV : \mathcal{P}(t(\sigma)) \to V \cup \{\emptyset\}$, so that only $FV(\{\nu\}) = \emptyset$. Thus, the multiset appearing in $rl(t(\sigma))$ that represents ν is empty in the left handside of the rule (because ${}^\bullet t_\nu = \emptyset$) and is only $t_\nu^\bullet$ in the right hand side, so that we obtain a fresh identifier. Consider for instance the net in Fig. 5, that takes a token from p and produces a fresh identifier in q. Since modes cannot instantiate x and ν by the same value, the only equivalence class in $Modes(t)/_{\approx_t}$ is $\mathcal{P} = \{\{x\}, \{\nu\}\}$, that is, that in which x and ν are not related. Thus, according to the definition in the previous section, the only rule produced by that net is

$${}^\bullet t_x FV(\{x\}) + {}^\bullet t_\nu FV(\{\nu\}) \longrightarrow t_x^\bullet FV(\{x\}) + t_\nu^\bullet FV(\{\nu\})$$

If we take $FV(\{x\}) = M$, since it must be the case that $FV(\{\nu\}) = \emptyset$, the rule is just $p\ M \to M + q$, so that it is guaranteed that the name appearing in q is indeed fresh.

Notice that the reachability property in ν-APN's differs from ordinary reachability, since we are allowing the renaming of every identifier appearing in markings, and not only those newly created. For instance, let us consider the net in Fig. 6, in which no token is created. If we ask whether the marking M given by $M(p_1) = M(p_2) = \emptyset$, $M(p_3) = \{b\}$ and $M(p_4) = \{a\}$, can be reached, the result would be affirmative, since $M[a/b, b/a]$ is reachable in one step, and $M \equiv_\alpha M[a/b, b/a]$. Instead, if we do not consider α-equivalence over markings, then the answer should be negative, since the only reachable marking is that with an a in p_3 and a b in p_4, which is different from M. This situation, as proved in 4.17, is mimicked in our representation of multisets, in which the multiset $p_1 + p_2$ (p_1 representing a and p_2 representing b) is rewritten to $p_3 + p_4$.

This can be avoided with the introduction for each name in the initial marking of an isolated place containing that name. Let us see how this construction works for the net in Fig. 6. Since there are only two names in the initial marking, we add two places, one for a and one for b, containing tokens a and b, respectively. Now, the reached marking M' is that given by $M'(p_0) = M'(p_1) = \emptyset$, $M'(p_3) = M'(a) = a$ and $M'(p_4) = M'(b) = b$, but now it is not the case that $M' = M[a/b, b/a]$, because of the introduced places, nor any other marking that is α-equivalent to M. That marking would be represented in the rewrite theory as $a\ p_1 + b\ p_2$, which can be rewritten to the multiset $a\ p_3 + b\ p_4$.

On the one hand, ν-APN's are a particular class of MSPN systems, with only one net component, without locality places or movement transitions and without

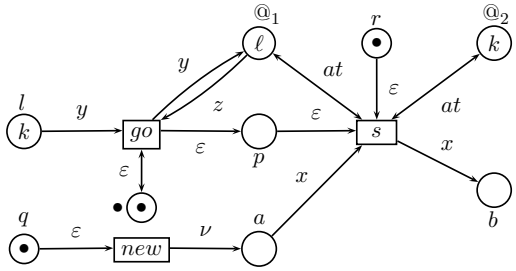

Figure 8. APN that simulates the MSPN system in Fig. 7

synchronizing transitions. However, on the other hand, they are powerful enough to simulate any arbitrary MSPN system stepwise.

Theorem 4.21 *Every MSPN system can be (strongly) simulated by a ν-APN.*

In Fig. 8 we show the APN that results from the construction described in the proof of the previous theorem, applied to the MSPN system in Fig. 7. Basically, it consists on adding places $\{@_1, \ldots, @_n\}$, used to save the current location of each net, unfold possible synchronizations ad adding test arcs to restrict them according to locations. We assume, in order to simplify notations, that $\bullet \in Id$ and $\mathcal{L} \subseteq Id$.

Summing up, we have proved that MSPN systems can be strongly simulated by rewrite systems, meaning that there is a one-to-one correspondence not only between their states, but also between their computations. Moreover, this isomorphism preserves the monoid structure in MSPN systems, so that typical property in Petri Nets, such as reachability, coverability, boundedness or home space property, can be studied in the resulting rewrite theory.

5 Some implementation details

We have seen a canonical translation from the operational semantics of ν-APN's to rewrite theories. These rewrite theories can be straightforwardly expressed in Maude. We need, not only sorts for places and multisets of places, but also for multisets of multisets of places, which correspond to markings.

```
sorts Place MSPlaces Marking .
subsorts Place < MSPlaces < Marking .
```

The following are the constructors for multisets of places and multisets of multisets of places, respectively.

```
op emptySet : -> MSPlaces .
op  __  : MSPlaces MSPlaces -> MSPlaces [comm assoc id: emptySet] .
op _+_  : Marking  Marking ->  Marking  [comm assoc id: emptySet] .
```

We want to work module $\backsim$, so that $M + \{\emptyset\} = M$ must be a valid identity in the equational theory. The relation $\backsim$ can be proved to be the least congruence for $+$ such that $\{\} \backsim \{\emptyset\}$. Therefore, it is enough to use the same constant as identity both in `MSPlaces` and in `Marking` in order to obtain the desired domain.

Then, a ν-APN is simply a system module of Maude that rewrites terms of sort `Marking`. The construction of the rewrite theory can be rather cumbersome for

systems of medium size. This is because the Maude system module corresponding to the ν-APN translation of the given MSPN system can be much bigger and quite different to that of the original system. In fact, if $Var(t)$ has n different variables then t produces B_n different rules, where B_n is the n-th Bell's number. Therefore, to develop it in an automatic way using the Maude system, we define a signature for MSPN systems so that they can be represented as Maude terms of sort `NetSystem`. We represent nets as sets of arcs and places (in order to allow the existence of isolated places). We need auxiliary sorts for labels, colours, variables and others that we do not mention here. Some of the main constructors are the following:

```
subsorts Place Arc < Net < NetSystem .

op _-_->_ : Place Var Transition -> Arc .
op _,_ : Net Net -> Net [assoc comm] .
op _-_ : NetSystem NetSystem -> NetSystem [assoc comm] .
```

As an example to illustrate the signature, we can define the following constants, that represent the MSPN system in Fig. 7.

```
ops N1 N2 : -> Net .
op S : -> NetSystem .

eq N1 = ((l,locality) - y -> [t1,go]) , ((q,black) - eps -> [t2,new]) ,
        ([t1,go] - eps -> (p,black)),([t2,new] - y -> (a,identifier)),
        ((p,black) - eps -> [t3,s ?]),((a,identifier) - x -> [t3,s ?]).

eq N2 = ((r,black) - eps -> [t4,s !]),([t4,s !] - x -> (b,identifier)) .

eq S = N1 - N2 .
```

We take advantage of the reflective properties of Maude in order to automatically perform the desired translation, by means of a function

```
op moduleOf : NetSystem -> SModule .
```

that returns the metarepresentation of the Maude system module that gives us the translation of the given system into an equivalent ν-APN. We can use the module `moduleOf(S)` to query about the behaviour of `S`. For instance, we can execute `S` from an initial marking

```
red metaRewrite(moduleOf(S),upTerm((rid1,black)+(pid2,locality)),
                                                    unbounded) .
```

or ask whether a marking is reachable from an intitial marking

```
red metaSearch(moduleOf(S),upTerm((rid1,black)+(pid2,locality),
               upTerm((rid2,black)+(sid1,locality)),'*,unbounded,0) .
```

In [15] we proved that coverability is decidable for MSPN systems. We have implemented in Maude the decision procedure used there, by means of

```
op cover : Marking Marking SModule -> Bool .
```

In order to avoid the user the torment of introducing the description of a system by hand, we have implemented an interface with one of the many existing graphical tools that deal with Petri nets. We have chosen *CPN Tools* [13], since it is probably the most widely used tool in the Petri Net community. By using *CPN Tools* we can draw Coloured Petri nets in a friendly way. Those nets are stored in a special format called CPNML, which is just an XML type definition. We have used the XML $->$ Maude translator written by Steven Eker. The obtained file contains a module "TRANSLATION" that declares a constant "translation" defined by a

term over a Maude signature for XML documents. To conclude we just need to define the mapping from these translations into `NetSystem` terms.

```
op translate : Element -> NetSystem .
op translatedNet : -> NetSystem .
eq translatedNet = translate(translation) .
```

Thus, the term `moduleOf(translatedNet)` is the metarepresentation of the system module corresponding to the system drawn in *CPN Tools*. Therefore we can, for instance, use it as an input for the coverability function shown before.

6 Conclusions and future work

In this paper we have introduced a possible approach for the verification of MSPN systems, based on a translation to rewriting logic, that is a natural semantic framework for concurrent systems. We have defined ν-APN's, that capture the essential characteristics of MSPN systems, and formalized the translation from a ν-APNs to an equivalent rewrite theory. The resulting theory can be implemented in Maude, and we can use the reflective properties of Maude in order to automatically perform the translation.

As future work, we plan to improve and enhance our Maude prototype, since it is still at a very early stage. For instance, we have only defined a one-way translation, from MSPN systems to Maude modules. It would be desirable to be able to deal with markings in the same level as MSPN systems, so that we could also use *CPN Tools* to enter initial markings or markings to reach or cover. For that purpose we would need to implement not only the function NF, but also NF^{-1}. Moreover, from the point of view of verification it would be essential to have a link from the translation back to the original model.

In the last section we have mentioned an algorithm to decide the coverability problem of MSPN systems. However, that algorithm is far from efficient. Although some of the causes can be mitigated, the problem lies in the fact that the backward analysis produces an explosion on the set of predecessor markings, even with the ideal-based representation of sets of markings we use. We plan to study in depth the complexity of the algorithm, and we are currently studying the possibility of using a forward analysis instead, that as in [2] should contribute to increase the efficiency of the algorithm. We also plan to combine that approach with the introduction of abstractions or appropriate type systems, that would turn our analysis incomplete, but at the same time more manageable when applicable.

So far, we have only defined the behaviour of our nets by means of firings, so that the implicit semantics that we are considering is in fact a trace semantics. We also plan to extend the existing works on firings of bags of transitions and process semantics of Petri nets to our setting and study to which extent the developed translation into rewriting logic preserves also those concurrent semantics.

It would also be interesting studying how we can use Maude built-in facilities in the analysis of our systems. Finally, since we are using the CPNML markup language to represent MSPN systems, it would be desirable to have a type definition to detect documents that, although syntactically correct according to CPNML, do

not represent any MSPN system.

Acknowledgements

The author would like to thank David de Frutos and Alberto Verdejo for their valuable comments and Steven Eker for his XML→Maude translator.

References

[1] A. Asperti, and N. Busi. *Mobile Petri Nets.* Technical Report UBLCS-96-10, University of Bologna, 1996.

[2] P. A. Abdulla, A. Collomb-Annichini, A. Bouajjani, and B. Jonsson. Using forward reachability analysis for verification of lossy channel systems. *Formal Methods in System Design*, 25(1):39–65, 2004.

[3] M.A. Bednarczyk, L. Bernardinello, W. Pawlowski, and L. Pomello. *Modelling Mobility with Petri Hypernets.* 17th Int. Conf. on Recent Trends in Algebraic Development Techniques, WADT'04. LNCS vol. 3423, Springer-Verlag, 2004.

[4] A. Bouhoula, J. Jouannaud and J. Meseguer. *Specification and proof in membership equational logic.* Theoretical Computer Science, vol. 236(1-2), pp. 35-132, 2000.

[5] L. Cardelli. Abstractions for Mobile Computation. Secure Internet Progr., Security Issues for Mobile and Distributed Objects, LNCS vol. 1603. Springer-Verlag, 1999.

[6] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer and C. Talcott. The Maude 2.0 System. In Proc. Rewriting Techniques and Applications, 2003. LNCS vol. 2706, pp. 76–87. Springer-Verlag, 2003.

[7] K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use.* Volume 1, Basic Concepts. Monographs in Theoretical Computer Science, Springer-Verlag, 2nd corrected printing 1997. ISBN: 3-540-60943-1.

[8] O. Kummer. Referenznetze. Logos-Verlag, 2002.

[9] I.A. Lomazova. *Nested Petri Nets; Multi-level and Recursive Systems.* Fundamenta Informaticae vol.47, pp.283-293. IOS Press, 2002.

[10] N. Martí-Oliet and J. Meseguer. *Rewriting logical as a logical and semantic framework.* ENTCS vol. 4, 2000.

[11] J. Meseguer and U. Montanari. *Petri Nets are Monoids: a New Algebraic Foundation for Net Theory.* In Proc. 3rd Annual Symposium on Logic in Computer Science, pp. 155-164. IEEE Comput. Soc. Press, 1988.

[12] R. Milner. *Theories for the Global Ubiquitous Computer.* Foundations of Software Science and Computation Structures-FoSSaCS 2004, LNCS vol.2987, pp.5-11. Springer-Verlag, 2004.

[13] A.V. Ratzer, L. Wells, H.M. Lassen, M. Laursen, J.F. Qvortrup, M.S. Stissing, M. Westergaard, S. Christensen, K. Jensen. *Tools for Editing, Simulating, and Analysing Coloured Petri Nets.* In Applications and Theory of Petri Nets, ICATPN 2003, LNCS vol. 2679, pp. 450-462. Springer-Verlag, 2003.

[14] F. Rosa Velardo, O. Marroquín Alonso and D. Frutos Escrig. *Mobile Synchronizing Petri Nets: a choreographic approach for coordination in Ubiquitous Systems.* In 1st Int. Workshop on Methods and Tools for Coordinating Concurrent, Distributed and Mobile Systems, MTCoord'05. ENTCS, 150.

[15] F. Rosa Velardo, D. Frutos Escrig and O. Marroquín Alonso. *On the expressiveness of Mobile Synchronizing Petri Nets.* In 3rd International Workshop on Security Issues in Concurrency, SecCo'05. ENTCS (to appear). `http://kimba.mat.ucm.es/~frosa.`

[16] M. Stehr, J. Meseguer and P.C. Ölveczky. *Rewriting Logic as a Unifying Framework for Petri Nets.* In Unifying Petri Nets, Advances in Petri Nets. LNCS, vol. 2128, pp. 250-303. Springer-Verlag, 2001.

[17] M. Stehr, J. Meseguer and P.C. Ölveczky. *Representation and Execution of Petri Nets Using Rewriting Logic as a Uniform Framework.* In Uniform Approaches to Graphical Process Specification Techniques, UNIGRA'01. ENTCS, vol. 44, 2001.

[18] P. Thati, K. Sen, and N. Martí-Oliet. *An Executable Specification of Asynchronous Pi-Calculus Semantics and May Testing in Maude 2.0.* ENTCS, 71, 2002.

[19] R. Valk. *Petri Nets as Token Objects: An Introduction to Elementary Object Nets*. Applications and Theory of Petri Nets 1998, LNCS vol.1420, pp.1-25, Springer-Verlag, 1998.

[20] Alberto Verdejo and Narciso Martí-Oliet. Implementing CCS in Maude 2. *Electr. Notes Theor. Comput. Sci.*, 71, 2002.