Modular State Space Analysis of Coloured Petri Nets

Søren Christensen
Computer Science Department,
Aarhus University
Ny Munkegade, Bldg. 540
DK-8000 AARHUS C, Denmark
Phone: +45 86 12 71 88
Telefax: +45 89 42 32 55
E-mail: schristensen@daimi.aau.dk

Laure Petrucci
Institut d'Informatique d'Entreprise,
CEDRIC—IIE
18, Allée Jean Rostand
F-91025 EVRY Cedex, France
Phone: +33 (1) 69 36 73 71
Telefax: +33 (1) 69 36 73 05
E-mail: petrucci@iie.cnam.fr

Abstract. State Space Analysis is one of the most developed analysis methods for Petri Nets. The main problem of state space analysis is the size of the state spaces. Several ways to reduce it have been proposed but cannot yet handle industrial size systems.
Large models often consist of a set of modules. Local properties of each module can be checked separately, before checking the validity of the entire system. We want to avoid the construction of a single state space of the entire system.
When considering transition sharing, the behaviour of the total system can be captured by the state spaces of modules combined with a Synchronisation Graph. To verify that we do not lose information we show how the full state space can be constructed.
We show how it is possible to determine usual Petri Nets properties, without unfolding to the ordinary state space.

1 Introduction

State spaces, also called Occurrence Graphs or Reachability Graphs, grow exponentially with respect to the number of independent processes, i.e., if a system has n independent processes each of which can be in m states the full state space have m^n states while n*m states would be sufficient to decide all the properties of the system. The stronger the relations between the processes are, the smaller the state space will be. Judging from industrial application of CP-nets, (e.g., [CJ91]) it seems that there is much local behaviour relative to the interaction between modules, i.e., the Occurrence Graphs will be large.

The paper is organised as follows.

In section 2, we summarise definitions related to Modular CP-nets.

It turns out that it is much easier to define Modular State Spaces if we restrict the models to use transition fusion only. In section 3 we do this. We also show that no information is lost; this is done by unfolding the Modular State Spaces into ordinary state spaces.

The usual properties of Petri Nets, i.e. reachability, bounds, home, and liveness can all be checked using the Modular State Spaces without unfolding to ordinary state spaces. In section 4 we specify how this can be done.

Several practical applications and tools use place fusion, and not transition fusion, so to make the Modular State Spaces more usable section 5 discusses how we can translate models using place fusion into models using only transition fusion—without changing the behaviour of the system.

Section 6 reports on a larger example we have used to test the Modular State Space approach.

Finally, section 7 concludes by comparing the results obtained to other work in this area.

2 Modular CP-nets

Before defining modular state space graphs, we recall the definition of Modular Coloured Petri Nets [CP92].

The reasons for using modular CP-nets are twofold. First of all, we want a simplified model which does not include all the concepts which are useful when modelling with CP-nets. Instead, we only include the basic primitives needed in the discussion of CP-nets analysis. Secondly, we want to discuss composition concepts which are based both on the sharing of places and on the sharing of transitions.

Modular CP-nets consist of sets of formally related CP-nets, each CP-net is called a module. Two sorts of relations between modules are considered, which are quite natural and often used. The first construct can be described as a set of places sharing the same tokens. When a transition adds (respectively removes) a token to one of the places in the set, it is added to (respectively removed from) all the places in the set. This is place fusion. The second construct fuses sets of transitions. All transitions of a set occur as one indivisible action sharing the values assigned by a common binding.

In the following, we use the notations of [Jen92] for CP-nets.

Definition 2.1 ([CP92], Definition 4.3)
A **Modular CP-net** is a triple MCPN = (S, PF, TF), satisfying the following requirements:

(i) S is a finite set of **modules** such that:

- Each module, s   S, is a CP-net:
  $M_S = (\Sigma_S, P_S, T_S, A_S, N_S, C_S, G_S, E_S, I_S)$.
- The sets of net elements are pairwise disjoint:
  $s_1, s_2 \quad S: [s_1 \neq s_2 \quad (P_{s_1} \quad T_{s_1} \quad A_{s_1}) \cap (P_{s_2} \quad T_{s_2} \quad A_{s_2}) = \emptyset]$.

(ii) PF 2^P is a finite set of **place fusion sets** such that:

- Members of a place fusion set have identical colour sets and equivalent initialisation expressions:
  p_1, p_2 pf: $[C(p_1) = C(p_2)$ $I(p_1) = I(p_2)]$.

(iii) TF 2^T is a finite set of **transition fusion sets**.

(i) A modular CP-net contains a finite set of modules, each of them being a CP-net. These modules must have disjoint sets of places, transitions and arcs. In general we use the X_S to denote "X of module s" or "X restricted to module s". We recall the notations of the components of a CP-net. Σ is the set of colour sets. P is the set of places and T the set of transitions. A is the set of arcs. Function N associates with each arc the pair (source node, destination node). Function C associates a colour set with each place. Function G associates a guard with each transition. Function E determines the arc expressions which permit to specify the input and output tokens of a transition. Finally, function I gives the initialisation expressions of the places in order to determine the initial marking of the coloured net.

(ii) Each place fusion set is a set of places to be fused together. 2^P denotes the set of all subsets of places. We demand that all elements of a place fusion set have the same colour set and that they have equivalent initial markings. By external places EP P we denote the set of all places which are members of a place fusion set and by internal places, IP = P – EP, we denote all non-fused places. It should be noted that, in contrast to [HJS90], we do not demand the place fusion sets to be disjoint.

(iii) Each transition fusion set is a set of transitions to be fused together. By external transitions ET T we denote the set of all transitions which are members of a transition fusion set (tf TF, t tf: t ET) and by internal transitions, IT = T – ET, we denote all non-fused transitions. The external transitions are those shared by several modules. It should be noted that, in contrast to [HJS90], we do not demand the transition fusion sets to be disjoint.

In Def. 2.2, we introduce place groups and transition groups. The notion of place groups corresponds to the notion of place instance groups for hierarchical CP-nets ([Jen92], Def. 3.5).

Definition 2.2 ([CP92], Definition 4.4)
A **place group** pg P is an equivalence class of the smallest equivalence relation containing all pairs (p_1,p_2) P×P where
pf PF: p_1,p_2 pf.
A **transition group** tg T consists of either a single non-fused transition t or all the members of a transition fusion set tf TF.
The set of place (transition) groups is denoted by PG (TG).

Place groups and transition groups are defined very differently. A place can be a member of at most one place group while a transition can be a member of several transition groups. Place groups form a partition of the set of places. This means that each place p is a member of one and only one place group which shall be denoted [p]. Note that all place groups and transition groups have at least one element. In the following, we use names with a prime to denote place groups and transition groups, e.g., p' and t'. From Def. 2.1 (ii) and 2.2 we know that all places of a place group will have the same colour set and equivalent initial markings; this allows us to talk about C(p') and I(p') without being ambiguous. We define: p'=[p]   PG: C(p') = C(p) and I(p')<> = I(p)<>.

Next, we define the set of variables associated with a transition group, the binding of a transition group and the guard of a transition group.

Definition 2.3 ([CP92], Definition 4.5)

A **binding** of a transition group t' is a function b defined on the variables of the transition group, $Var(t') = \bigcup_{t \quad t'} Var(t)$, such that:

(i) v Var(t'): b(v) Type(v).

(ii) t t': G(t)<b>.

We denote the conjunction of guards of a transition group t' by G(t'), and the set of all bindings by B(t').

A binding will assign only one value for a variable, i.e. a variable name will refer to the same value for all transitions in a transition group. Such a mechanism allows communication between transitions of a same transition group.

Next, we extend the arc function A to handle place groups and transition groups:

A(x',y') = {a A½ x x' y y': N(a) = (x,y)}.

Then the expression function E is extended from arcs to place groups and transition groups. The summation is well-defined because all the involved expressions have the same type:

(x_1,x_2) (PG×TG   TG×PG): $E(x',y') = \sum_{a \quad A(x',y')} E(a)$.

Now, we define token elements, bindings elements, markings and steps for modular CP-nets. This is done in a similar way as for hierarchical CP-nets.

Definition 2.4 ([CP92], Definition 4.6)

A **token element** is a pair (p',c) where p' PG and c C(p'), while a **binding element** is a pair (t',b) where t' TG and b B(t'). The set of all token elements is denoted by TE while the set of all binding elements is denoted by BE.

A **marking** is a multi-set over TE while a **step** is a *non-empty* and *finite* multi-set over BE. The initial marking M_0 is the marking which is obtained by evaluating the initialisation expressions:

(p',c) TE: $M_0(p',c)$ = I(p')(c).

The set of all markings and steps are denoted by $\mathbb{M}$ and $\mathbb{Y}$, respectively.

The enabling rule of a modular CP-net can now be expressed. The inequality used to compare a value to a marking is the inequality of multi-sets.

Definition 2.5 ([CP92], Definition 4.7)

A step Y is **enabled** in a marking M iff the following property is satisfied:

p' PG: $\sum_{(t',b) \quad Y} E(p',t')<b> \leq M(p')$.

When a step Y is enabled in a marking M_1 it may **occur**, changing the marking M_1 to another marking M_2, defined by:

$$p' \;\; PG\!: M_2(p') = \Big(M_1(p') - \sum_{(t',b)\;\; Y} E(p',t')\langle b\rangle\Big) + \sum_{(t',b)\;\; Y} E(t',p')\langle b\rangle.$$

We say that M_2 is **directly reachable** from M_1 by the occurrence of step Y, which we also denote by: $M_1\,[Y\rangle M_2$.

Let s be a function which returns the source node of an arc. To talk about the elements of the entire Modular CP-net with modules in a set S, we use a notation without indexes, e.g.: $\Sigma = \bigcup_{s\;\; S} \Sigma_s$

A Modular CP-net can be unfolded into an equivalent CP-net:

Definition 2.6 ([CP92], Definition 4.8)
Let a modular CP-net MCPN = (S, PF, TF) be given. Then we define the **equivalent CP-net** to be CPN = (Σ*, P*, T*, A*, N*, C*, G*, E*, I*) where:

(i) $\Sigma^* = \Sigma$.
(ii) $P^* = PG$.
(iii) $T^* = TG$.
(iv) $A^* = \{(a,t') \;\; A\times TG \;ç\; t(a) \;\; t'\}$.
(v) $a^*=(a,t') \;\; A^*$:
$[\; s(a) \;\; P \quad N^*(a^*) = ([p(a)],t')$
$\;\; s(a) \;\; T \quad N^*(a^*) = (t',[p(a)]) \;]$.
(vi) $p^* \;\; P^*\!: C^*(p^*) = C(p^*)$.
(vii) $t^* \;\; T^*\!: G^*(t^*) = G(t^*)$.
(viii) $a^*=(a,t') \;\; A^*\!: E^*(a^*) = E(a)$.
(ix) $p^* \;\; P^*\!: I^*(p^*) = I(p^*)$.

A Modular CP-net and its equivalent CP-net are behaviourally equivalent:

Theorem 2.7 ([CP92], Theorem 4.9)
Let MCPN be a modular CP-net and let CPN* be the equivalent CP-net. Then we have the following properties:

(i) $\mathbb{M} = \mathbb{M}^* \qquad M_0 = M_0^*$.
(ii) $\mathbb{Y} = \mathbb{Y}^*$.
(iii) $M_1,M_2 \;\; \mathbb{M}, \quad Y \;\; \mathbb{Y}\!: M_1\,[Y\rangle_{MCPN} M_2 \Leftrightarrow M_1\,[Y\rangle_{CPN^*} M_2$.

The definitions given in this section are sufficient as a formal base for defining Modular State Spaces.

3 Modular State Spaces - Transitions Sharing Only

In this section, we concentrate on CP-nets composed by transition fusion only. It is theoretically easy to generate the state space of the individual modules and to compose these into the state space of the entire system, as we will see in section 3.3.

But practical use is harder: a module can have an infinite state space while the full state space is finite, e.g., for the following modules composed by fusion of the two grey transitions t.

To avoid handling infinite state spaces, we would like to obtain an efficient construction of state spaces of modules knowing that they will be composed later on. Only the reachable parts of the state space should be constructed.

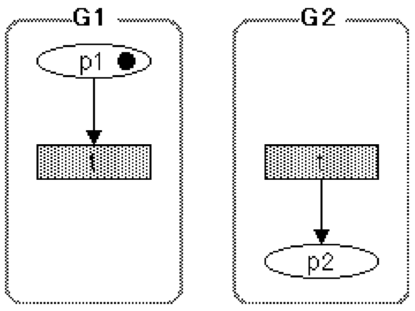

Fig. 3.1: Two modules sharing a transition

The construction of a Modular State Space is similar to the one of the standard state space except that:

- the construction of state spaces using only transitions local to modules can be performed in parallel;
- the construction of a Modular State Space requires to keep track of the occurrence of transition groups and synchronise the modules using this information.

As the method is valid as well for P/T-nets as for CP-nets (replace transitions by binding elements), we indifferently use P/T-nets or CP-nets as examples.

In the following of this section, we first present examples illustrating our method, and then we formalise it. In fact, the synchronisation graphs are a bit simplified in the sense that they do not compact nodes as explained later in the formalisation.

3.1 First Example: Basic Explanation of the Method

Here, we consider a simple P/T-net. One module is on the left, the second one on the right of figure 3.1.1. They share a transition fusion set containing the grey transitions named t.

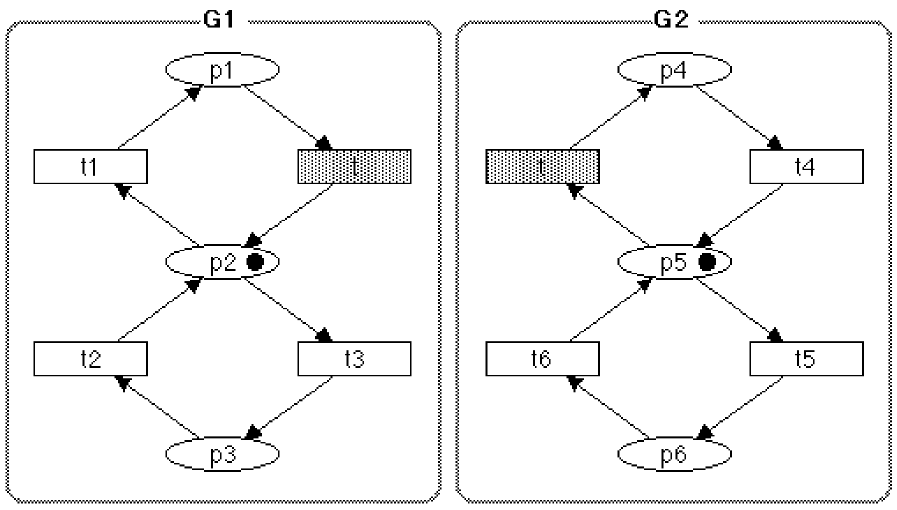

Fig. 3.1.1: Two modules sharing the t transition.

In figure 3.1.2, *state space G1* and *state space G2* are the state space graphs for the first and the second module respectively, while *synchronisation graph* indicates how G1 and G2 synchronise on common transitions (with the numbers referring to the markings of the two other graphs). In the figures, we represent the internal transitions by arrows with hollow heads, while the external transitions are pictured by arrows with solid heads.

To construct these graphs, the following method is used:

Start with the initial markings of the modules (number 1 in G1 and G2).

Then, let us start with G1: from marking 1, t1 and t3 are the only enabled transitions, and they are local, so construct the new markings 2 and 3 and arcs t1 and t3. From marking 2, the only enabled transition is t2, which is local and leads to marking 1, so construct arc t2. From marking 3, the only enabled transition is t which is shared. There, we do not try to calculate the new marking: perhaps this transition is not enabled because of the other module, and so we avoid constructing an unnecessary sub-graph. We just record that t is enabled in this module by drawing the arc labelled by t leading to nothing (for the moment). All the markings of G1 have been examined, so we stop and wait for G2 to be finished in order to synchronise the shared transition t.

In G2, from marking 1, t is enabled, so we draw the arc labelled by t and leading to nothing. Local transition t5 is also enabled so we construct marking 2 and arc t5. From marking 2, t6 is enabled and leads to marking 1, so construct arc t6. All the markings of G2 have been examined, so this part is finished.

The two previous partial graphs construction can be performed in parallel as they are independent of each other.

Now we can synchronise the transitions t. Although t is enabled in marking 1 in G2, it is not from the initial marking in the equivalent (flat) CP-net because it cannot occur from marking 1 in G1. But after t1 occurs, the new marking from G1 (3) allows it to occur, and G2 is still in state one, thus, t is enabled. Now, we can construct the successor of marking 3 in G1 by t, i.e. marking 1 of G1, and of marking 1 in G2 by t, i.e. marking 3 of G2. Then, we can continue the constructions of G1 and G2. We stop when all the nodes have been examined in the graphs. In the synchronisation graph, the node labels indicate states in G1 and G2. The arc label says that shared transition t is enabled from a state composed of number 3 of G1 and number 1 of G2, these two nodes being obtained by occurrence of local transitions only from the previous state 1 in the synchronisation graph. The arc leads to a new node 2.

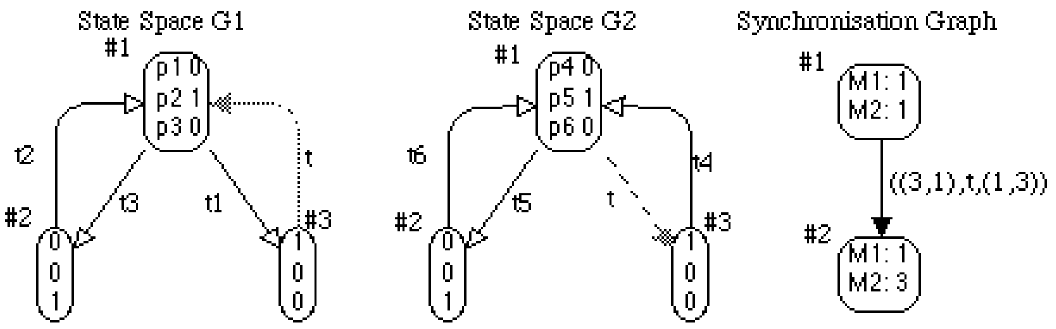

Fig. 3.1.2: State Spaces and Synchronisation Graph.

It may happen that, at the end, there are arcs labelled with shared transitions that lead to nothing. They show branches that could have been constructed in a straightforward algorithm but could not occur in the context of the full model. Hence, we suppress them.

3.2 Second Example: Advanced Explanation of the Method

When a node in the graph of a module has been examined, there may be shared transitions left out because they cannot be synchronised with a corresponding shared transition of another module, but they must be recorded in the module graph because if a path leads back to the marking, maybe the shared transition can become enabled. It is the case in the resource allocation system.

The resource allocation example ([Jen92]) has a set of processes which share a common pool of resources. There are two different kinds of processes, called p-processes and q-processes, and three different kinds of resources: r-resources, s-resources and t-resources. Each process is cyclic and during the individual parts of its cycle, the process needs to have exclusive access to a varying amount of the resources. We use the following definition of colours: U = {p,q} and E = {e}. We use a variable x of type U. The p-processes can be in four different states, while q-processes can be in five different states. In the initial state, there are 2 p-processes and 3 q-processes, plus 1 r-resource, 3 s-resources and 2 t-resources. The Modular CP-net is presented in Fig. 3.2.1. It is decomposed into three modules sharing only transitions. Two modules represent the cycles of p-processes and those of q-processes, while the other one represents the use of resources.

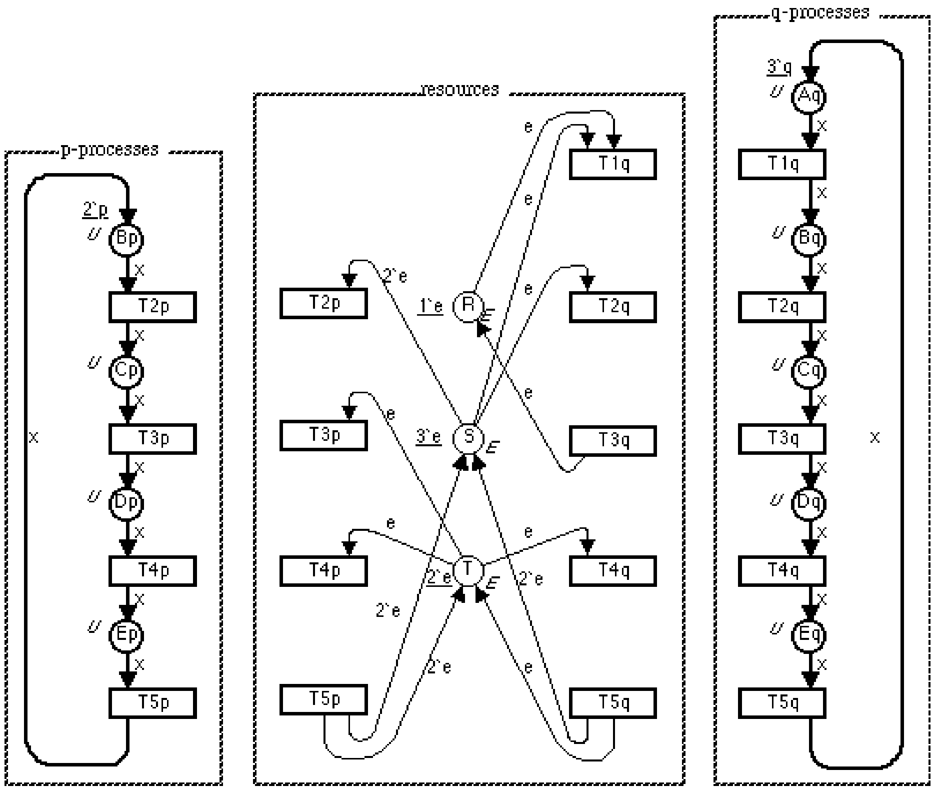

Fig. 3.2.1: The three modules of the resource allocation system

We construct the state spaces of the modules and synchronise them as explained in the previous section. The nodes in the synchronisation graph contain the number of the nodes in the state spaces of p-processes, q-processes and resources.

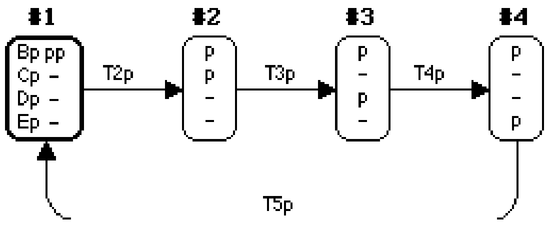

Fig. 3.2.2: The graph for the p-processes module

We can compare the size of the state spaces constructed to the size of the complete state space for each module. For the p-processes module, the full state space has 10 nodes, but only 4 of them are reachable. For the q-processes module, the full state space has 34 nodes, but only 7 of them are reachable. The full state space for the resource module is infinite, its covering graph contains 55 nodes but only 9 nodes are reachable.

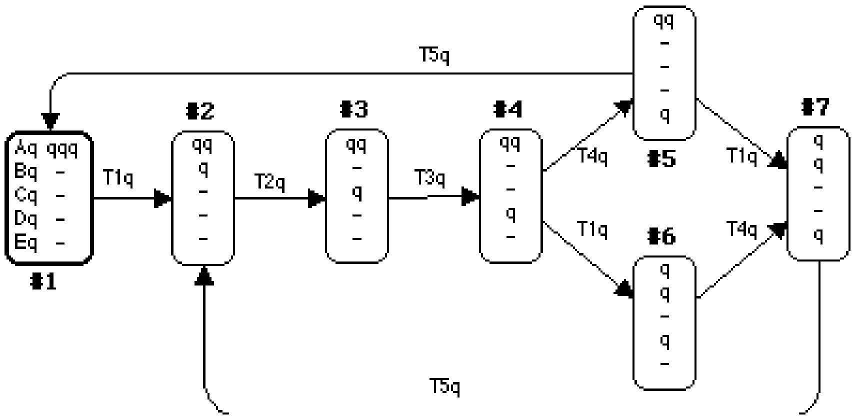

Fig. 3.2.3: The graph for the q-processes module

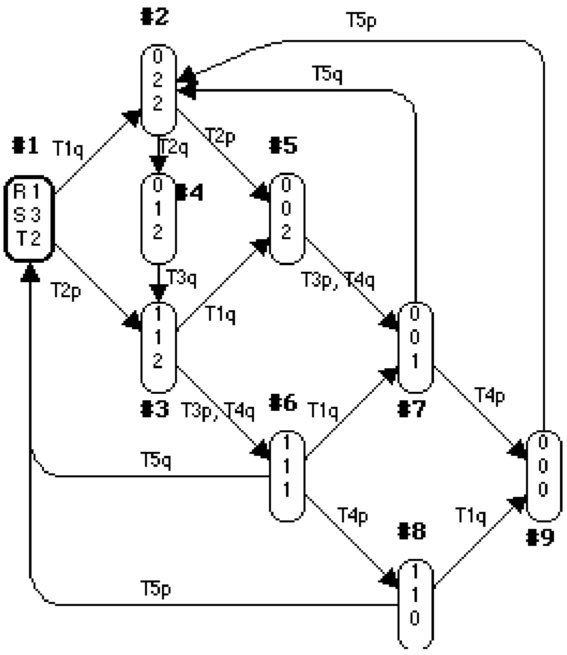

Fig. 3.2.4: The graph for the resources module

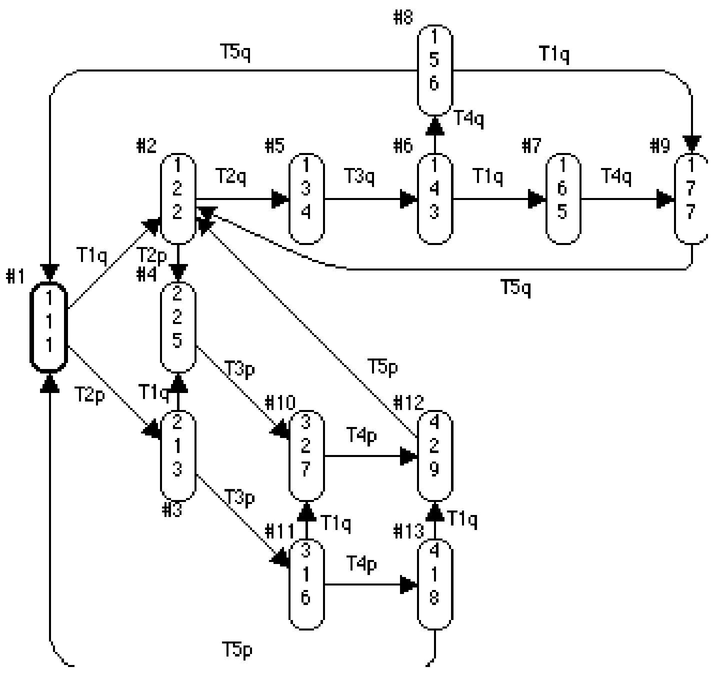

Fig. 3.2.5: The synchronisation graph

A local analysis on the state spaces of modules can be performed to examine, e.g., boundedness. Looking only at the p-processes graph, we can see that 2 p-processes cannot be active at the same time.

We note that in both examples, only the nodes used in the full state space were constructed, the state spaces of the modules are thus smaller than what we would have obtained using a straightforward method. The resource allocation system example is an extreme one in the sense that all the transitions are shared, so no local occurrences can be performed and all we gain is an automatic two-level encoding of states.

3.3 Modular State Spaces for Transition Fusion

We now present a formal description of the state space of a modular CP-net with transition fusion only, consistent with the definition of these nets, i.e. also defined in a modular way. Explanation is given just after the definitions and they should be read in parallel. We first give definitions of notations that is used to formalise Modular State Spaces.

Definition 3.3.1
Let MCPN = (S, PF, TF) be a modular CP-net.
(i) $m \quad \mathbb{M}: \prod(m) = \{m' | \quad \sigma \quad IT^*, m[\sigma>m'\}$.
(ii) $m_1,m_2 \quad \mathbb{M}^2: m_1 \, R \, m_2 \Leftrightarrow [\prod(m_1) \cap \prod(m_2) \neq \emptyset]$.
(iii) $m \quad \mathbb{M}: \nu(m) = \{m' \quad \mathbb{M}: m \, R_{TC} \, m'\}$.

(i) For all markings m of the full system, $\prod(m)$ denotes the set of states of the full system which are reachable from m using only internal transitions of the modules. Thus $\prod(m)$ is the cross product of the reachable states of the modules. Checking whether a given state x is in $\prod(m)$ does not require to generate $\prod(m)$, it is sufficient to check that x_s is locally reachable from m_s for all modules s.

(ii) Two markings are related by relation *R* iff they have common internally reachable markings.

(iii) We define ν(m) as a set of related markings by taking the transitive closure of relation *R*. Note that for a given marking m, ν(m) is unique.

These definitions are illustrated in figure 3.3.1. A triangle below m represents ∏(m).

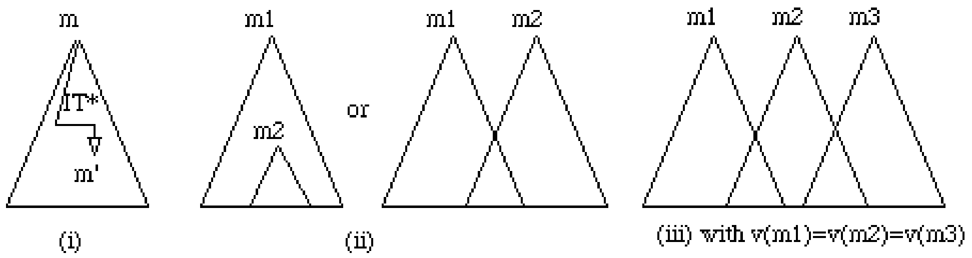

Fig. 3.3.1: Illustration of Def. 3.3.1

In definition 3.3.2 we give an algorithm in order to define Modular State Spaces.

Definition 3.3.2
Let MCPN = (S, PF, TF) be a modular CP-net where:
(i) PF=Ø.
The **Modular State Space** of MCPN is MOG = (SG, $(OG_s)_{s\ \ S}$, TF), where:
(ii) SG=(V_{sg}, A_{sg}, N_{sg}) is the Synchronisation Graph of MCPN, constructed as follows:

(ii.i) ν(M_0) V_{sg}.

(ii.ii) v V_{sg}:   m   𝕄: ν(m) = v: t   ET,   b   B([t]):
m[([t],b)>m'   (v,(m,([t],b),m'),ν(m')) A_{sg}.

(ii.iii) a=(v,(m,([t],b),m'),v') A_{sg}: N_{sg}(a)=(v,v').

(iii) s   S, OG_s=(V_s, A_s, N_s) is a State Space Graph of module s. A_s is partitioned into two sets IA_s and EA_s. IA_s is the set of arcs labelled by internal transitions. EA_s is the set of arcs labelled by fused (external) transitions. OG_s is constructed as follows:

(iii.i) M_{0s} V_s.

(iii.ii) m_s V_s: t IT_s,   b   B(t):
m_s[(t,b)>m'_s (m'_s V_s (m_s,(t,b),m'_s) IA_s).
t ET_s,   (v,(m,([t],b),m'),v') A_{sg}:
m_s[(t,b)>m'_s (m'_s V_s (m_s,(t,b),m'_s) EA_s).

(iii.iii) a=(m,(t,b),m') A_s: N_s(a)=(m,m').

(i) There is no place fusion.

(ii.i) The initial marking is represented by a node of the synchronisation graph.

(ii.ii) Let us consider a node v of the synchronisation graph and a marking m represented by this node. If an external transition t is enabled from m with a binding b, leading to a new marking m', then (v,(m,([t],b),m'),ν(m')) is an arc of the synchronisation graph.

(ii.iii) An arc (v,(m,([t],b),m'),v') starts from node v and ends in node v'.

(iii.i) The initial marking restricted to a module s is a node of the state space graph of s.

(iii.ii) Let us consider a node m_s of the state space graph of a module s. If an internal transition t is enabled, with a binding b from m_s, leading to a marking m'_s, then m'_s is also a node of OG_s and (m_s,(t,b),m'_s) is an arc of OG_s. If an external transition t is enabled from a marking m such that there exists an arc (v,(m,([t],b),m'),v') in the synchronisation graph, then m'_s is also a node of OG_s and (m_s,(t,b),m'_s) is an arc of OG_s.

(iii.iii) An arc (m,(t,b),m') starts from node m and ends in node m'.

The modular state space can be unfolded into a flat state space. The explanation is given just below the definition and both should be read in parallel.

Definition 3.3.3
Let MCPN be a modular CP-net and MOG = (SG, $(OG_s)_{s\ \ S}$, TF) its modular state space. The **equivalent state space** of MOG is OG=(V,A,N), defined as follows:

(i) $V = \{m\ \ \mathbb{M} \mid \nu(m)\ \ V_{sg}\}$.

(ii) $A = \bigcup_{\substack{t\ \ T,\ b\ \ B(t),\ m\ \ V: \\ s\ \ S \text{ with } t\ \ T_s: \\ (m_s,([t],b),m'_s)\ \ A_s}} \{(m,([t],b),m + \sum_{s\ \ S,(m_s,([t],b),m'_s)\ \ A_s} (m'_s - m_s)^*\ \ )\}$

where $m\ \ \mathbb{M}_s$: $m^*\ \ \mathbb{M}\ \ \ m^*_s = m_s\ \ [\ \ s'\ \ S, s' \neq s: m^*_{s'} = 0]$.

(iii) $a=(m,(t,b),m')\ \ A$: $N(a)=(m,m')$.

(i) The set of nodes of the equivalent state space is the set of markings represented by a node in V_{sg}.

(ii) If a transition was enabled in all the modules in which it appeared (a single module for an internal transition), there is a corresponding arc in the equivalent state space. The marking obtained is changed for all the modules concerned as specified in their state space.

(iii) Function N gives the source and destination of an arc.

The following theorem states that the equivalent flat state space of MOG and the state space of the equivalent non-hierarchical CP-net of MCPN are the same.

Theorem 3.3.4
Let MCPN be a modular CP-net, MOG its modular state space and CPN its equivalent non-hierarchical CP-net. Let OG_{MOG} be the equivalent state space of MOG and OG_{CPN} the state space of CPN. We have:

$OG_{MOG} = OG_{CPN}$.

Proof: follows from definitions 3.3.1, 3.3.2, 3.3.3, 2.6, theorem 2.7 and the definitions of enabling and occurrence rules for a non-hierarchical CP-net. ♦

We have defined, in the case of transitions fusion, a modular state space, consisting of a synchronisation graph, the state spaces of the modules and the transition fusion sets. Modular state spaces can be flattened to lead to a normal state space. We have shown that the flat state space of a modular CP-net is the same as the state space of its equivalent non-hierarchical CP-net.

We wanted to obtain a structure which is as small as possible but still allows to check net properties without constructing the equivalent flat state space. We show, in the next section, how properties can be checked directly on the modular state space.

As concerns the compactness of the modular state space, we can easily find the best and worst cases. The best case we can obtain is when there is no fused transition: the state spaces of modules are the full state spaces of the modules, and the synchronisation graph contains only one node and no arc. The equivalent state space would be the cross product of the modules and would thus be much larger. The worst case would be when all the transitions are shared. Then the synchronisation graph and the equivalent state space have the same size. But these cases are extreme ones and of little interest when modelling real systems.

4 Proof Rules for Modular State Spaces

In this section we show how the usual Petri Nets properties can be decided from the Modular State Spaces.

In the rest of this section we assume that MCPN is a modular CP-net, and MOG its modular state space. We use DPF(m_1,m_2) to denote the set of all directed finite paths from node m_1 to node m_2. For each proposition, we indicate the corresponding proposition for CP-nets in [Jen94]).

Proposition 4.1 ([Jen94], proposition 1.12)
For the **reachability properties**, we have the following proof rules, valid for all M_1, M_2 $[M_0>$:

(i.i) $[M_0> = \{M \;\; \mathbb{M} \; / \;\; v \;\; V_{sg}: \nu(M) = v\}$.

(i.ii) $M \;\; [M_0> \Leftrightarrow [\;\; v \;\; V_{sg}: \nu(M) = v]$.

(ii) $M_2 \;\; [M_1> \Leftrightarrow [M_2 \;\; \prod(M_1)]$
$[\;\; a_1 \ldots a_n \;\; DPF_{sg}(\nu(M_1),\nu(M_2))$:
$s(a_1) \;\; \prod(M_1)$, $s(a_i) \;\; \prod(d(a_{i-1}))$ for $1<i\leq n$, $M_2 \;\; \prod(d(a_n))]$.

Explanation:

(i.i) and (i.ii) The set of reachable markings is the set of markings represented by a node in V_{sg}.

(ii) Functions s and d return the source and destination of an arc. These markings are inscribed in the arc expression. Sequence $a_1 \ldots a_n$ is the projection of the occurrence sequence from M_1 to M_2 on the set of external transitions. Note that there can be arcs, a, starting and ending in the same node of V_{sg}.

Proof: (i.i) and (i.ii) follow from the definition of ν.

(ii) We can distinguish two cases:
Case 1: Assume that $M_2 \;\; \prod(M_1)$.
From the definition of $\prod$, we have: $M_2 \;\; \prod(M_1) \Leftrightarrow [\;\; \sigma \;\; IT^*:M_1[\sigma>M_2]$.
Thus $M_2 \;\; [M_1>$.
Case 2: Assume that:
(0) $a_1 \ldots a_n \;\; DPF_{sg}(\nu(M_1),\nu(M_2))$ such that:
(1) $s(a_1) \;\; \prod(M_1)$,
(2) $s(a_i) \;\; \prod(d(a_{i-1}))$ for $1<i\leq n$,
(3) $M_2 \;\; \prod(d(a_n))$.
From (0) and (1) we deduce that: $\sigma_1 \;\; IT^*:M_1[\sigma_1>s(a_1)$.
From (2) we have: $\sigma_i \;\; IT^*:d(a_{i-1})[\sigma_i>s(a_i)$.
From (3) we have: $\sigma_{n+1} \;\; IT^*:d(a_n)[\sigma_{n+1}>M_2$.
Thus, $M_1[\sigma_1 a_1 \sigma_2 a_2 \ldots a_n \sigma_{n+1}>M_2$ as required.

Either M_2 is reachable from M_1 by occurrences of local transitions only and then we are in case 1, or we have to prove that we are in case 2. The sequence from M_1 to M_2 can be written $M_1[\sigma_1 a_1 \sigma_2 a_2 \ldots a_n \sigma_{n+1}>M_2$ where $\sigma_i \;\; IT^*$ and $a_i \;\; ET^*$. Using this and the definition of $\prod$, it is straightforward to see that $s(a_1) \;\; \prod(M_1)$, $s(a_i) \;\; \prod(d(a_{i-1}))$ for

$1<i\leq n$, M_2 $\prod(d(a_n))$. From the definition of ν, arc a_1 starts from node $\nu(M_1)$ and arc a_n ends in node $\nu(M_2)$. ♦

Proposition 4.2 ([Jen94], proposition 1.13)
For the **boundedness properties**, we have the following proof rules, valid for all X TE, all s S, all p P_s, all functions F [$\mathbb{M}\rightarrow$A], and all functions F_s [$\mathbb{M}|P_s\rightarrow$A] where (A,≤) is an arbitrary set with a linear ordering relation:

(i.i) $\text{BestUpperBound}(X_s) = \max_{M_s \;\; OG_s} |(M_s|X_s)|$.

(i.ii) $\text{UpperBound}(X) = \sum_{s \;\; S} \text{BestUpperBound}(X_s)$

(i.iii) $\text{BestUpperBound}(X) = \max_{(v \;\; V_{sg}, M:\nu(M) = v)} |(M|X)|$.

(ii) $\text{BestUpperIntegerBound}(p) = \max_{M_s \;\; OG_s} |M_s(p)|$.

(iii) $\text{BestUpperMulti-setBound}(p) = \max_{M_s \;\; OG_s} M_s(p)$.

(iv.i) $\text{BestUpperBound}(F_s) = \max_{M_s \;\; OG_s} F_s(M_s)$.

(iv.ii) $\text{UpperBound}(F) = \sum_{s \;\; S} \text{BestUpperBound}(F_s)$

(iv.iii) $\text{BestUpperBound}(F) = \max_{(v \;\; V_{sg}, M:\nu(M) = v)} F(M)$.

Explanation:

(i.i) and (iv.i) We define the BestUpperBound for a module s.

(i.ii) and (iv.ii) The sum of BestUpperBounds of modules is an upper bound but maybe not the best one.

(i.iii) and (iv.iii) The best one is the one obtained by looking only at the markings represented by an element of V_{sg}.

Proof: Follows from definition 3.3.2, 3.3.3 and [Jen94] Prop. 1.13. ♦

Proposition 4.3 ([Jen94], proposition 1.14)
For the **home properties**, we have the following proof rules, valid for all X [M_0> and all M [M_0>:

(i) X HS [s S: SCC_{sT} $X_s{}^c$] SCC_{sgT} $\nu(X)^c$.

(ii) X HS [s S: $|SCC_{sT}|\leq|X_s|$] $|SCC_{sgT}|\leq|\nu(X)|$.

(iii) M HM $SCC_{sT}=\{M_s{}^c\}$ $SCC_{sgT}=\{\nu(M)^c\}$.

(iv) HM≠Ø $|SCC_{sT}| = 1$ $|SCC_{sgT}| = 1$.

(v) M_0 HM $|SCC_s| = 1$ $|SCC_{sg}| = 1$.

Explanation:

(i) If X is a home space, in each module, the terminal strongly connected components are included in the components of X (restricted to this module) and each terminal strongly connected component of the synchronisation graph is in the components of the representative node of X.

(ii) If X is a home space, in each module, the number of terminal strongly connected components is smaller than the size of X (restricted to this module) and the number of terminal strongly connected components of the synchronisation graph is smaller than the number of representative nodes of X.

(iii) If M is a home marking, its component in a module s is the only terminal strongly connected component of OG_s and the component of its representative in the synchronisation graph is the only strongly connected component of SG.

(iv) If there is a home marking, the state spaces of the modules and the Synchronisation Graph have only one terminal strongly connected component.

(v) If the initial marking is a home marking, there is only one strongly connected component in the state spaces of the modules and in the synchronisation graph.

Proof:

(i) Assume that X is a home space. Then:

M [M_0>: M' X, σ T*: M[σ>M'.

In any module s, we have: M_s[$σ_s$>M'_s, i.e. M'_s is a home space of module s.

From [Jen94], Prop. 1.14 (i), we conclude that s S: SCC_{sT} X_s^c.

As M[σ>M', the projection of σ on the set of external transitions leads from the node corresponding to M in SG to the one corresponding to M', i.e. ν(M)[$σ_{ET}$>ν(M'). Thus SCC_{sgT} ν(X)^c.

(ii) The proof is the same as for (i), but we use [Jen94] Prop. 1.14 (ii) instead of [Jen94] Prop. 1.14 (i).

(iii) Same proof as (i) with X={M'} and using [Jen94] Prop. 1.14 (iii).

(iv) and (v) follow from (iii). ♦

Proposition 4.4 ([Jen94], proposition 1.15)

For the **liveness properties**, we have the following proof rules, valid for all M [M_0>, all X BE, all s S and all t T:

(i) M is dead ⇔ [t T: s S: [t' [t] ∩ T_s: a A_s: M' $\mathbb{M}_s$:
b B(t'): a≠(M_s,(t',b),M')]].

(ii) X is dead in M ⇔ [(t,b) X: s S: [t' [t] ∩ T_s: a A_s: M' $\mathbb{M}_s$:
a≠(M_s,(t',b),M')]].

(iii) X is live s S: [c SCC_{sT}: T_s ∩ X = Ø BE(c) ∩ X ≠ Ø].

(iv) t is live s S: [c SCC_{sT}: t T_s t BE(c)].

(v) t is strictly live s S: [c SCC_{sT}: t T_s [b B(t): (t,b) BE(c)]].

Explanation:

(i) and (ii) A marking M is dead iff no internal transition is enabled and if no external transition is enabled. A set X of binding elements is dead in a marking M iff none of its elements is enabled in M. For each binding element we can distinguish two possibilities: either the transition is local and it exits in only one module were it must not be enabled, or it is shared and there must exist one module containing a transition member of the same group which is not enabled.

(iii), (iv) and (v) Let s be a module. If transition t (or a—set of—binding element) is live, either it is not a transition of s, or it appears in all strongly connected components of the graph of module s.

Proof:

(i) A marking M is dead iff all the transitions are not enabled in this marking. We can distinguish two sorts of transitions. First, the internal transitions only belong to one module s and constitute a whole transition group. Such a transition is not enabled; it is equivalent to no arc labelled with this transition group in OG_s. Second, the shared transition is not enabled iff one member of its group is not enabled in its module. Hence we get the property.

(iii) The proof is a slight modification of the proof of (i).

(iv) The proof is similar to the one of (v) given just below.

(v) Assume that transition t is live. Let s be a module. Either t does not belong to T_S—and then t does not have to be looked up in module s—or t does—and we now consider this case. Let c be a terminal strongly connected component of module s and M a marking in c. As t is live and c is terminal, there exists an occurrence sequence containing t starting in M. Thus t is in BE(c).

(vi) The proof is a slight modification of the proof of (v). ♦

In this section we have shown how the standard Petri Nets properties—as defined in [Jen94]—can be determined from the Modular State Spaces.

5 Modular State Spaces – Places Sharing

The composition of state space graphs is more complex when sharing places rather than transitions. In this case, we are ensured that if at least one of the modules has an infinite state space graph, the modular CP-net also have an infinite state space graph. But it is impossible to tell anything about the state space graph of the modular CP-net if those of the modules are finite. This is due to the fact that a module can provide enough tokens in a place fusion set to allow some new bindings, in another module, to be enabled. And then this second module can provide some more tokens for the first one and so on.

This can be seen on the following example, where the grey place p2, initially empty, is the shared one:

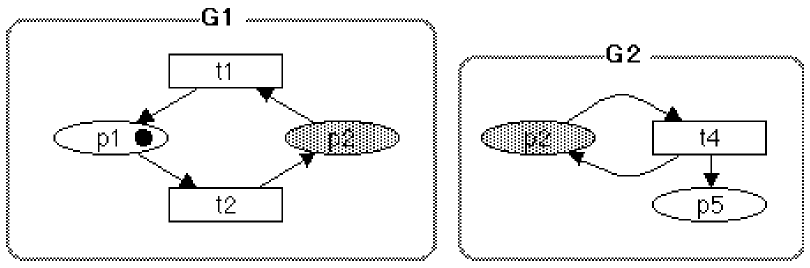

Fig. 5.1: Two modules with finite graphs, but a net with infinite graph

Thus, it is impossible to deduce the modular CP-net state space graph from those of the modules.

From a practical point of view it is important to be able to handle systems which use place fusion, since several kinds of CP-nets use this.

We define a transformation from a Modular CP-net using place fusion to a Modular CP-net using only transition fusion. Informally this is done by collecting each place fusion in a new module and then splitting the input and output transitions of each place fusion set, such that we get a transition fusion set for each input and output transition of the fusion set. In figure 5.2 it is shown how the system of figure 5.1 can be translated into a behaviourally equivalent modular net using transition fusion only.

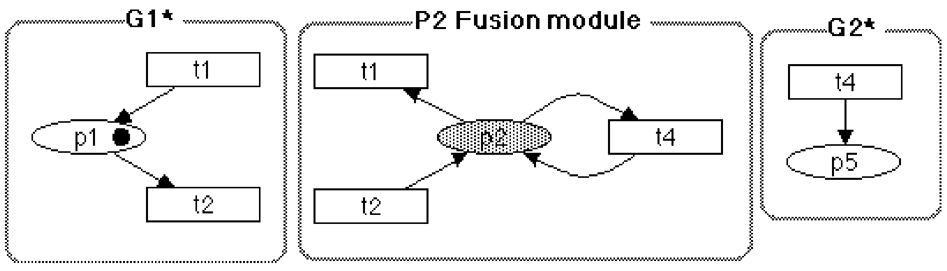

Fig. 5.2: A modular CP-net with transition fusion only

The formal definition of the translation and the proof that the behaviour is preserved are not included in this paper. They contain a number of technical details which are of little importance for the results presented.

6 A Larger Example

To test the ideas of modular state spaces on a larger example we have conducted the following experiment.

Using the Design/CPN Occurrence Graph Analyzer, [OGA95], we have generated the ordinary state space of a system consisting of two different modules communicating through a single shared transition.

The size of the ordinary state space was 16,384 nodes and 126,976 arcs. It took 605 seconds to generate the graph.

We did not implement the algorithm described in 3.3.2, but it was possible for us to simulate the algorithm using the features of the ordinary tool. It was however not possible to measure the duration of the generation of the modular state space. Generating the local state spaces of both the modules took less than 3 seconds. This corresponds to the observation that the time to generate a state space is proportional to the number of arcs in the graphs. We did not generate the synchronisation graph, but from the structure of the state spaces of the modules we have calculated the size of the synchronisation graph.

The state space of module 1 contained 256 nodes and 1,024 arcs, while the one of module 2 contained 64 nodes and 192 arcs. The synchronisation graph contained 1 node and 12,288 arcs.

Hence we have a total of 321 nodes and 13,504 arcs in the modular state space, which means that the modular graph representation needs only 1.95% of the nodes and 10.6% of the arcs for this particular example.

The modular construction of the occurrence graph allows either to gain time or space: the two extreme cases are when all the transitions are shared and when just one of them is. In the first case, all the modules have a very small graph, but the synchronisation process is very heavy and takes time, while in the second case, there is few synchronisations, but each of the modules has large graph and thus takes space. The efficiency of the method depends a lot on the modular structure chosen. We think that most industrial models would not correspond to one of the extreme cases and thus the method would be efficient both in time and space.

7 Conclusion

In this paper, we have presented a way of generating state spaces of systems exploiting their modular structure. We have shown how to construct this for systems without place fusion, and we have shown a translation from Modular CP-nets with place fusion into Modular CP-nets using transition fusion only.

If the results of definition 3.3.3 and theorem 3.3.4 are used to construct the ordinary state space, then the Modular State Space method is only a fast way of generating the ordinary state space. Except for degenerated cases it is faster since the local behaviour is only developed once—not for each global state allowing this particular behaviour.

But it is possible to check properties using the modular state space directly, i.e., without unfolding to the ordinary state space.

In [Val90a], Valmari describes how the state space can be generated by generating/ reducing and combining state spaces of modules until the resulting state space of the full system is generated. The most important differences to our approach is that we preserve all information, do not generate the full graph, and have specified how properties can be checked directly on the Modular State Space.

The work presented in [NM94] first constructs the complete state spaces of modules. We pointed out that we did not want to do that because it is possible to have a module with an infinite state space while the graph for the complete system is finite. Thus we only construct the reachable markings. [NM94] also presents results for reachability and deadlock analyses in P/T-nets. Our results are necessary and sufficient conditions for reachability, deadlock and boundedness analyses in CP-nets. We also give necessary conditions for home and liveness properties.

There are two other approaches to reduce the size of state spaces, called Symmetric Occurrence Graphs (see [Jen94]), and Stubborn Sets as introduced by Valmari in [Val90b, Val91]. The approach of this paper shares an important property with Symmetric Occurrence Graphs—no information is lost. This fact makes it attractive to combine Modular State Spaces with Symmetric Occurrence Graphs, and since they exploit two totally different properties of the systems—symmetry in the states and actions vs. locality of actions—the reduction should be the sum of reductions from each of the methods. It is not clear how a combination of Modular State Spaces and Stubborn sets would work. These two methods are closely related—since both try to minimise the representation of the interleaving of independent actions.

Acknowledgements

We would like to thank Gérard Berthelot, Kurt Jensen, Jens Bæk Jørgensen, Kjeld Høyer Mortensen, Romain Vassallo and the anonymous referees whose comments helped us to improve the paper.

References

[CJ91] S. Christensen, L. O. Jepsen: **Modelling and simulation of a network management system using hierarchical coloured Petri nets**. In Erik Mosekilde (ed.): Proceedings of the 1991 European Simulation Multiconference. ISBN 0–911801–92–8, pp. 47–52. An extended version available as: Daimi PB–349, ISSN 0105–8517, April 1991.

[CP92] S. Christensen, L. Petrucci: **Towards a modular analysis of coloured Petri nets**. In: K. Jensen (ed.): Application and Theory of Petri Nets 1992. Lecture Notes in Computer Science, vol. 616, Springer–Verlag, 1992, 113–133. Also available as: Daimi PB–391, ISSN 0105–8517, April 1992.

[HJS90] P. Huber, K. Jensen and R. M. Shapiro: **Hierarchies in coloured Petri nets**. In: G. Rozenberg (ed.): Advances in Petri Nets 1990. Lecture Notes in Computer Science, vol. 383. Springer-Verlag, 1990, pp. 342-416. Also in [JR91], pp. 215-243.

[Jen92] K. Jensen: **Coloured Petri nets. Basic concepts, analysis methods and practical use. Volume 1: Basic concepts.** EATCS monographs on Theoretical Computer Science, Springer-Verlag 1992.

[Jen94] K. Jensen: **Coloured Petri nets. Basic concepts, analysis methods and practical use. Volume 2: Analysis methods.** EATCS monographs on Theoretical Computer Science, Springer-Verlag 1994.

[JR91] K. Jensen and G. Rozenberg (eds.): **High-level Petri nets**: **theory and application.** Springer-Verlag 1991. ISBN 3-540-54125-X/0-387-54125-X.

[NM94] M. Notomi and T. Murata, **Hierarchical reachability graph of bounded Petri nets for concurrent-software analysis.** IEEE Transactions on Software Engineering, pp. 325 -336, Vol. 20, No. 5 May 1994.

[OGA95] **The Design/CPN occurrence graph analyzer**. Version 0.5, Meta Software Corporation. Daimi version - March, 1995, unpublished.

[Val90a] A. Valmari: **Compositional state space generation**. Proceedings of the 11th International Conference on Application and Theory of Petri Nets, Paris, France, June 1990, pp. 43-62.

[Val90b] A. Valmari: **Stubborn sets for reduced state space generation.** In: Rozenberg, G. (ed.): Advances in Petri Nets 1990. Lecture Notes in Computer Science, Vol. 483; Springer-Verlag 1991, pp. 491-515.

[Val91] A. Valmari: **Stubborn sets of coloured Petri nets.** Proceedings of the 12th International Conference on Application and Theory of Petri Nets, Gjern, Denmark, June 1991, pp. 102-121.