Modular Analysis of Petri Nets

SØREN CHRISTENSEN[1] AND LAURE PETRUCCI[2]

[1] *Department of Computer Science, University of Aarhus, Ny Munkegade, Building 540, DK-8000 ÅRHUS C, Denmark*
[2] *LSV, CNRS UMR 8643, ENS de Cachan, 61 avenue du Pdt Wilson, F-94235 CACHAN Cedex, France*
Email: petrucci@lsv.ens-cachan.fr

This paper shows how two of the most important analysis methods for Petri nets can be performed in a modular way. We illustrate our techniques by means of modular Place/Transitions nets (modular PT-nets) in which the individual modules interact via shared places and shared transitions. For place invariants we show that it is possible to construct invariants of the total modular PT-net from invariants of the individual modules. For state spaces, we show that it is possible to decide behavioural properties of the modular PT-net from state spaces of the individual modules plus a synchronization graph, without unfolding to the ordinary state space. The generalization of our techniques to high-level Petri nets is rather straightforward.

Received November 6, 1998; revised April 13, 2000

1. INTRODUCTION

The use of high-level Petri net formalisms has made it possible to create Petri net models of large systems. Even though the use of such models allows the modeller to create compact representations of data and action, the size of models has been increasing. A large model can make it difficult to handle the complexity of the modelling as well as the analysis of the total model. It is well known that the use of a modular approach to modelling has many advantages: it allows the modeller to consider different parts of the model independently of one another. A modular approach to analysis is also attractive: it often dramatically decreases the complexity of the analysis task.

Our aim in this paper is to show how two of the most important analysis methods for Petri nets can be performed in a modular way: place invariants and state spaces. We illustrate our techniques by means of modular Place/Transitions nets (modular PT-nets) in which the individual modules (PT-nets) interact via shared places and shared transitions. Sharing is often accomplished using place fusion sets and transition fusion sets. These two sorts of communication are present in a number of models, e.g. see [1, 2] for models using shared transitions and [3, 4] for models using place fusion.

For place invariants we show that it is possible to construct invariants of the total modular PT-net from invariants of the individual modules.

For state spaces, also known as occurrence graphs and reachability graphs, we show that it is possible to decide behavioural properties of the modular PT-net from state spaces of the individual modules plus a synchronization graph, without unfolding to the ordinary state space. The combined size of the occurrence graphs of the modules and synchronization graph is smaller than the size of the ordinary, unstructured state space. Hence, it is possible to handle more complex systems.

From our presentation it will be clear that the techniques also immediately apply to coloured Petri nets [4, 5] and other forms of high-level Petri nets [1, 6]. This is very important since most practical modelling and analysis are performed by means of high-level Petri nets. By presenting the techniques in terms of PT-nets we reduce the level of technical details. The generalization to high-level Petri nets is rather straightforward.

The modular Petri net model and the invariants composition were presented in [7], with coloured Petri nets as modules. A first version of modular occurrence graphs was published in [8], but there was a lack of tools to prove the properties of the system directly on these. It turned out that this first version of modular occurrence graphs had to be revised in order to do so.

This paper is structured as follows. Section 2 presents an example which illustrates the basic ideas behind modular PT-nets, i.e. place fusion and transition fusion. In Sections 3 and 4 we show how place invariants and state spaces can be constructed in a modular way. Then we turn to the formal definitions. Section 5 defines modular PT-nets. Section 6 defines place invariants for modular PT-nets, while Section 7 defines modular state spaces. In Section 8 we state their proof rules, i.e. the rules by which behavioural properties can be decided. Section 9 presents other, larger, examples. Finally, Section 10 is the conclusion.

2. MODULAR PT-NETS

In this section we present two different ways of modelling a problem, one using place sharing, and the other using transition sharing. The example described is a variation of the resource allocation system from [4]. In the next two sections the resource allocation examples are used to show how analysis results of modules can be composed. This

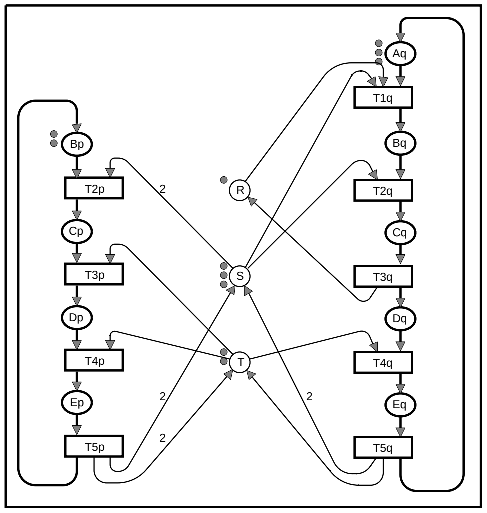

FIGURE 1. Example of the resource allocation system.

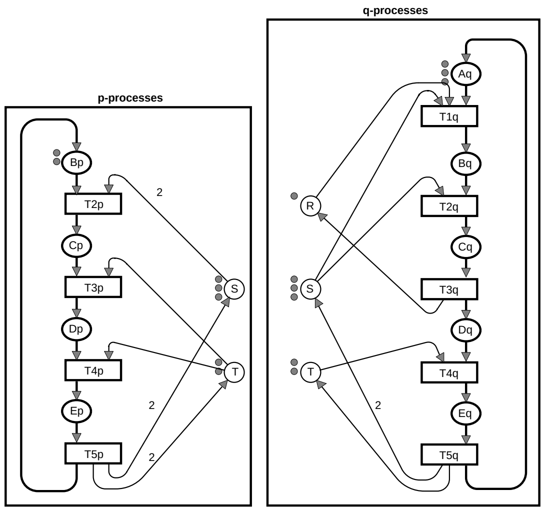

FIGURE 2. Modular PT-net with two modules and two place fusion sets with two members each.

means that properties of a modular PT-net can be proved by means of formal analysis of the individual modules.

The resource allocation example 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. The p-processes can be in four different states, while q-processes can be in five different states. In the initial state there are two p-processes and three q-processes plus one r-resource, three s-resources and two t-resources. The PT-net is presented in Figure 1. It is so small that we would not decompose it in practice, but it can still be used to introduce the basic concepts of modular PT-nets. A first possibility to model this system in a modular way consists in modelling the p-processes and the q-processes separately. This leads to two modules, as shown in Figure 2, each of them describing the interaction between processes of one kind and the resources. The modules are composed by fusion of the two shared resource places, i.e. the two places labelled with S are fused and likewise for the two places labelled with T. In Figure 2 the places to be fused together have the same name. The sets of places to be fused together are called place fusion sets. In a practical modelling tool we would need more elaborated techniques to identify members of a given fusion set, but this is not our purpose here.

You can view all places of a place fusion set as being representatives of the same underlying place. This means that they share the same marking: when a token is added to a place which belongs to a place fusion set, all places of the place fusion set will have the same token added. When a token is removed from a place which belongs to a place fusion set, all places of the place fusion set will have the same token removed. In addition, the fusion of the resource places, of the same kind, ensures that the modular PT-net of Figure 2 has exactly the same behaviour as the PT-net of Figure 1. All places in a place fusion set have identical initial markings.

Another way of modelling the resource allocation system is to separate the cycle of p-processes, that of q-processes and the use of resources, as shown in Figure 3. The three modules share transitions, corresponding to synchronous actions. Transitions having the same names belong to the same fusion set. This means that we have nine transition fusion sets with two members each. Each transition of a transition fusion set describes a part of a more complex action and all parts must occur simultaneously, as one indivisible action. We say that a transition fusion set is enabled if all the transitions in the fusion set are enabled. The change produced by the occurrence of a transition fusion set is the sum of changes produced by all the transitions of the fusion set.

A transition can describe an action which is a basic part of a number of independent actions. This means that a transition can be a member of several transition fusion sets. Since the behaviour, in the resources module, of the three transitions T3p, T4p and T4q is identical, we could include only one of the three transitions and have it as a member of three transition fusion sets. If our main concern was guidelines for modelling we would have done this, but Figure 3 corresponds to a straightforward separation of the original PT-net.

The modular PT-net of Figure 3 has exactly the same behaviour as the PT-net in Figure 1. Each transition fusion set of the modular PT-net corresponds to exactly one transition of the PT-net.

We have presented two examples of modular PT-nets having the same behaviour. The first one was an example of modules related by place fusion while the second used

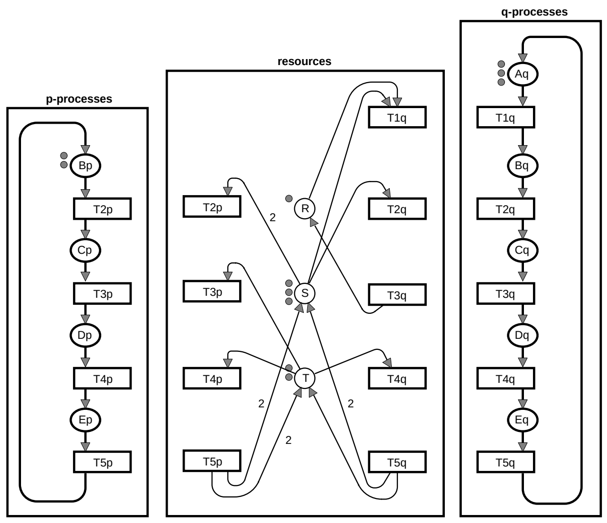

FIGURE 3. Modular PT-net with three modules and nine transition fusion sets with two members each.

transition fusion. In general, both place fusion and transition fusion can coexist within a modular PT-net. In this paper we do not consider the problem of identifying place and transition fusions sets. We suppose that the model is already designed in a modular way. However, one should keep in mind that the choice of these sets has an influence on the performance of modular analysis.

In Section 5 we give formal definitions of a modular PT-net and of its equivalent PT-net. In the next section we present a modular approach of place invariants calculus.

3. PLACE INVARIANTS OF MODULAR PT-NETS

All analysis methods extract information about the properties of a PT-net in a condensed way. Place invariants express invariant relations on the markings of places. A weight (positive or negative integer) is attached to the places. It specifies the information we want to extract from the markings of a place. A weight function (vector of weights associated with places) determines a place invariant if the sum of the weighted markings of places is constant for all reachable markings. It is often the case that some place invariant ignores the marking of some places. This is done by assigning weight zero to the places which should be ignored.

3.1. Place invariants of the PT-net

For the example of Figure 1 we find several place invariants. One of the place invariants shows the cycle of the q-processes: the weight of places Aq, Bq, Cq, Dq, and Eq is one and the weight of all other places is zero. We can show that the sum of the weighted markings is constant for all reachable markings. This means that the set of q-processes does not change, only the location of the q-processes changes during their cycle. Instead of checking all reachable markings, we can also check that the weighted sum of tokens consumed by each transition is equal to the weighted sum of tokens produced. We say that a set of weights having this property defines a place flow. It can be proved that the place flow property is sufficient to ensure that the weight function determines a place invariant.

In our notation of weight functions, places having a zero weight are simply left out and we use the names of the places to refer to their markings, e.g. we write Bp instead of M(Bp). For the example in Figure 1 we have five linearly independent place invariants:

$$\begin{aligned}
W_1&: \quad Bp + Cp + Dp + Ep = 2,\\
W_2&: \quad Aq + Bq + Cq + Dq + Eq = 3,\\
W_3&: \quad R + Bq + Cq = 1,\\
W_4&: \quad S + Bq\\
&\qquad +2 \times (Cp + Dp + Ep + Cq + Dq + Eq) = 3,\\
W_5&: \quad T + Dp + Eq + 2 \times Ep = 2.
\end{aligned}$$

All of the above place invariants can be interpreted in terms of the PT-net: W_1 shows that all the p-processes are in one of the states represented by Bp, Cp, Dp or Ep; and W_3 shows that the r-resources are either free, i.e. in state R, or occupied by a q-process in state Bq or Cq. We can construct other place invariants, but they can also be expressed as linear combinations of the five invariants W_1 to W_5. Using the five place invariants above, it is straightforward to prove the deadlock-freeness and similar behavioural properties of the system.

3.2. Place invariants of the modular PT-net with place sharing

In Figure 2, we have two modules, one for the p-processes and one for the q-processes. The two modules are related through two place fusion sets, i.e. through the s-resources and t-resources. For the p-processes, we have three place invariants:

$$\begin{aligned}
W_{\text{p1}}&: \quad Bp + Cp + Dp + Ep = 2,\\
W_{\text{p2}}&: \quad S + 2 \times (Cp + Dp + Ep) = 3,\\
W_{\text{p3}}&: \quad T + Dp + 2 \times Ep = 2.
\end{aligned}$$

For the q-processes, we have four place invariants:

$$\begin{aligned}
W_{\text{q1}}&: \quad Aq + Bq + Cq + Dq + Eq = 3,\\
W_{\text{q2}}&: \quad R + Bq + Cq = 1,\\
W_{\text{q3}}&: \quad S + Bq + 2 \times (Cq + Dq + Eq) = 3,\\
W_{\text{q4}}&: \quad T + Eq = 2.
\end{aligned}$$

We ask the question: is it possible to construct place invariants of the total system from the place invariants of the individual modules?

The place invariants W_{p1}, W_{q1} and W_{q2} do not include any places which are shared, so W_{p1}, W_{q1} and W_{q2} are place invariants of the total system, independent of the other modules in the modular PT-net. The rest of the place invariants have non-zero weights for some of the shared places. In this situation we can only combine the place invariants if the weight functions assign the same weight to the shared places. This means that we can combine W_{p2} and W_{q3}, because they both have weight one for S and weight zero for T. Analogously, we can combine W_{p3} and W_{q4} because they both have weight one for T and weight zero for S. From the place invariants of the individual modules we deduce the following place invariants of the modular PT-net:

$$\begin{array}{ll}
W_{p1}: & Bp + Cp + Dp + Ep = 2 \\
W_{q1}: & Aq + Bq + Cq + Dq + Eq = 3 \\
W_{q2}: & R + Bq + Cq = 1 \\
W_{p2} + W_{q3}: & 2 \times (Cp + Dp + Ep + Cq + Dq + Eq) \\
 & +S + Bq = 3 \\
W_{p3} + W_{q4}: & T + Dp + Eq + 2 \times Ep = 2.
\end{array}$$

These five place invariants correspond to linearly independent place invariants of the equivalent PT-net.

From the example above we see that a set of place invariants, one for each module, can be combined into a place invariant of the full system if they have the same weights for places which are shared. This is not true in the general case, but if we restrict ourselves to combining place flows it is a valid statement. This will be detailed in the formal definition of invariants, see Section 6. If the sets of weights do not match, in the sense described above, we may sometimes obtain a matching by using a linear combination of the weights in the invariants.

3.3. Place invariants of the modular PT-net with transition sharing

For the example shown in Figure 3 we have three modules, one for the p-processes, one for the q-processes and one for the resources. The three modules are related through transition fusion for each pair composed of a transition in the p-processes or q-processes and the corresponding transition in the resource module. If we view the modules independently of their context we can find a set of place invariants of the individual modules. For the p-processes, we have one place invariant:

$$W_p : \quad Bp + Cp + Dp + Ep = 2,$$

for the q-processes, we have one place invariant:

$$W_q : \quad Aq + Bq + Cq + Dq + Eq = 3,$$

and the resource sharing module has only the trivial null place invariant. We ask: is it possible to construct place invariants of the total system from the place invariants of the modules?

In this case we have no shared places and this means that W_p and W_q are both invariants of the entire system. However, it should be obvious that we cannot construct all the place invariants from those of the individual modules. The problem is that we demand too much from the weight functions. We demand that each transition leaves the invariant unchanged, while it would be sufficient to require that each transition fusion set does this. In order to relax the conditions of the weight functions for the modules, we introduce the notion of flow preservation. We say that a transition is flow preserving if and only if it preserves the invariant. Then we are able to check that the non-fused transitions are flow preserving for each module; and, also, that transition fusion sets are flow preserving across modules, i.e. that the transitions *together* preserve the invariant.

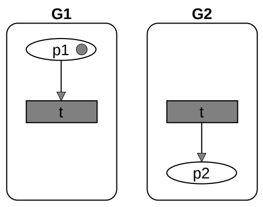

FIGURE 4. Two modules sharing a transition.

The above examples allowed us to show the intuition behind modular PT-nets and their analysis by means of place invariants. In Section 6 we formalize the notion of place invariants and place flows for modular PT-nets.

In the next section we present a modular approach to state spaces construction, which is the second main analysis method for PT-nets.

4. MODULAR STATE SPACES

In this section we introduce the basic ideas of modular state spaces. The main reason for working with modular state spaces is to alleviate the state space explosion problem. The idea is to generate a state space for each module and the information necessary to capture the interaction between modules, and in this way avoid the construction of the full state space. It is important that modular state spaces allow us to prove PT-net properties directly, i.e. without unfolding to the equivalent ordinary state space. Here, we will first consider modular PT-nets with fused transitions only. For that purpose we use another example which contains non-shared transitions, thus being more relevant. Then we will show how the problem can be solved for a modular PT-net with place fusion only.

4.1. Modular state spaces with transitions sharing

In this section, we concentrate on PT-nets composed by transition fusion only. As we will see later, it is theoretically easy to generate the state spaces of the individual modules and to compose these into the state space of the entire system. However, practical use is harder: a module can have an infinite state space while the full state space is finite, e.g. for the modules of Figure 4 composed by the 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,

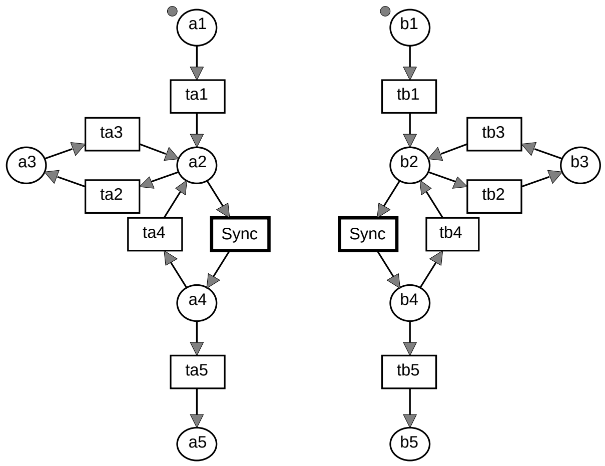

FIGURE 5. An example of two communicating processes.

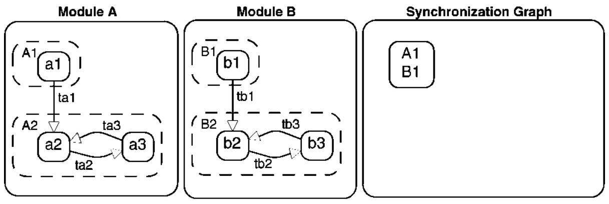

FIGURE 6. After the first step of the modular state space generation.

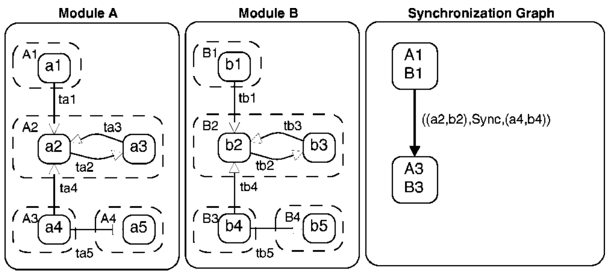

FIGURE 7. After the second step of the modular state space generation.

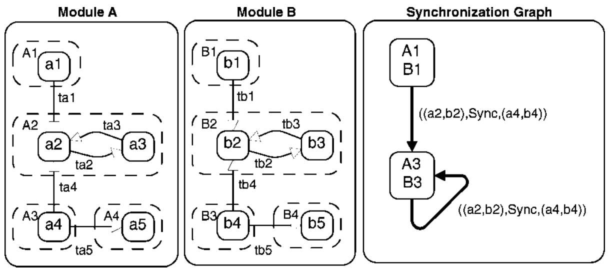

FIGURE 8. After the third step of the modular state space generation.

knowing that their behaviour is restricted by the behaviour of the other modules. Only the reachable parts of the state space should be constructed. Hence, our method is a balance between full reachability graph generation and on-the-fly verification.

A modular state space is composed of one local state space per module and a synchronization graph which captures the communications. A local state space only contains local information, i.e. the sub-graphs representing all reachable markings obtained by occurrences of local transitions only. The markings are restricted to the places of the module. The synchronization graph provides information on the state space of the overall system and the occurrences of fused transitions.

The construction of a modular state space is similar to that of the standard state space except that:

- the construction of local state spaces using only transitions local to modules can be performed in parallel;
- the construction of a modular state space requires one to keep track of the occurrence of shared transitions and to synchronize the modules using this information.

We use the example of Figure 5 to introduce the algorithm for constructing modular state spaces. This is a toy example, but it models a behaviour which resembles that of a typical real system as concerns the ratio between local and shared actions. The system consists of two modules, called module A and module B, each having a behaviour consisting of a start-up phase, a main loop and a termination phase. The main loop can perform local actions and it can synchronize with the other module, the synchronization is achieved by a transition fusion set consisting of the two transitions labelled *Sync*.

In the rest of this section we construct the corresponding modular state space. In Figures 6–9, the sets of nodes in dashed boxes are strongly connected components (SCCs), e.g. in Figure 6, A2 = {a2, a3}. We conclude the example with a comparison to the ordinary state space.

In the first step, we construct graphs containing all locally reachable states of each module. In the example the inscription a1 specifies that the place a1 is marked by a token and all other places of module A are unmarked. We also construct the first node of the synchronization graph, corresponding to the initial state. It is labelled by the set of SCCs of the modules to which the restriction of the initial state belongs; but it also represents the set of all reachable markings from the initial marking by occurrences of local transitions only. The result of this first step is shown in Figure 6.

The result of the second step is presented in Figure 7. From the state spaces of the modules we find reachable states which enable fused transitions. The combination of the states a2 and b2 enables the fused transition *Sync* leading to new states a4 and b4. To the state space of module A we need to add a4, and all markings reachable from a4 by occurrences of local transitions; and a similar construction

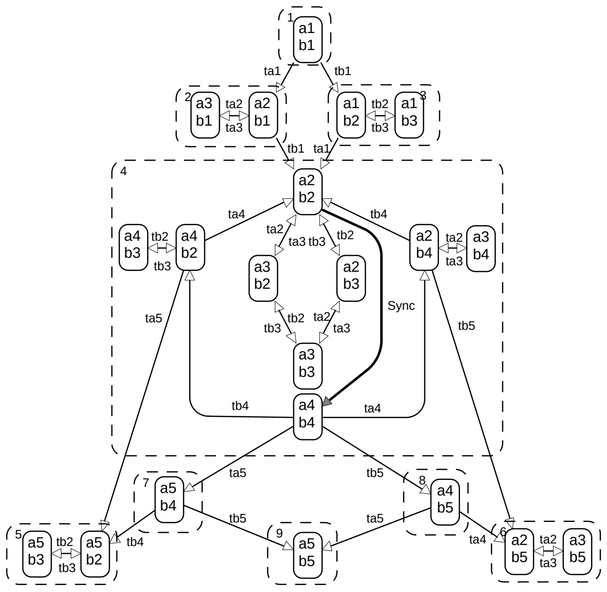

FIGURE 9. The ordinary state space.

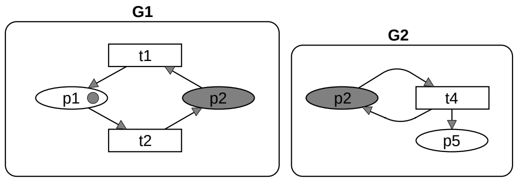

FIGURE 10. Two modules each with finite graphs, but a modular PT-net with infinite graph.

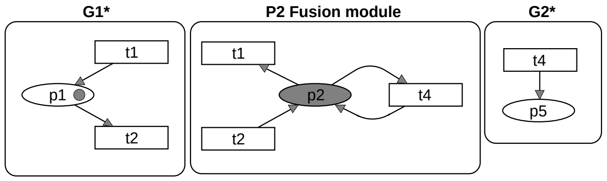

FIGURE 11. A modular PT-net with transition fusion only.

is applied to module B and state b4. Note that the state spaces of the modules, obtained by this construction, are not necessarily connected graphs. Finally, we add the arc corresponding to *Sync* to the synchronization graph. The arc is labelled by the start marking, the occurring transition and the marking obtained. It connects two nodes which represent the SCCs of the two involved markings.

The third step (Figure 8) is similar to the second step. We inspect the state space of the local modules starting from A3 and B3 to find reachable states which enable fused transitions, and since a2 and b2 are locally reachable from A3 and B3, we have to add the arc corresponding to *Sync*. As no new nodes can be added to the local state spaces, we know that they are now complete.

If we compare the ordinary state space of the whole system, illustrated in Figure 9, with the modular state space, we observe that the modular state space is smaller than the ordinary one. The modular state space contains a total of 12 nodes and 12 arcs, while the ordinary state space contains 21 nodes and 37 arcs. Hence, the memory necessary to store the state space is smaller. In Section 9.3, we discuss the size of the modular state space in general.

To construct a modular state space, it is necessary to calculate the SCCs. This is not the case when building an ordinary state space. However the SCCs are needed in order to determine net properties, thus computing them on-the-fly when building the modular state space is not a waste of time.

4.2. Modular state spaces—places sharing

The composition of state space graphs is more complex when sharing places rather than transitions. This can be seen in the example of Figure 10, where the grey place p2, initially empty, is the shared one. In this case, it is ensured that if at least one of the modules has an infinite state space graph, the modular PT-net also has an infinite state space graph. However, it is impossible to tell anything about the state space graph of the modular PT-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 transitions, in another module, to be enabled; and then this second module can provide some more tokens for the first one and so on.

From a practical point of view it is important to be able to handle systems which use place fusion, since the kinds of Petri nets used in practical applications often rely on this.

We define a transformation from a modular PT-net using place fusion to a modular PT-net using only transition fusion. Informally this is done by collecting each place fusion set in a new module and then splitting the input and output transitions, in order to obtain a transition fusion set for each input and output transition of the place fusion set. Figure 11 shows how the system of Figure 10 can be translated into a behaviourally equivalent modular PT-net using 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 here.

In this section we have presented a construction of modular state spaces. It is quite compact compared to the ordinary state space. It has independent parts which can be computed in parallel. In sections 7 and 8 we formalize the concepts presented here, and show the details of how to prove PT-net properties directly from the modular state space, i.e. without unfolding to the equivalent ordinary state space.

5. FORMAL DEFINITION OF MODULAR PT-NETS

We will now give the definition of a modular PT-net. We start by giving the definition of PT-nets, and introducing the notations used in the rest of the paper. We use the following definition of PT-nets.

DEFINITION 5.1. *A* PT-net *is a tuple* $PN = (P, T, W, M_0)$*, satisfying:*

(i) P *is a finite set of* places*;*
(ii) T *is a finite set of* transitions*. The sets of net elements are disjoint:* $T \cap P = \emptyset$*;*
(iii) W *is the arc* weight function *mapping from* $(P \times T) \cup (T \times P)$ *into* $\mathbb{N}$*;*
(iv) M_0 *is the* initial marking. M_0 *is a function mapping from* P *into* $\mathbb{N}$.

Now, we define markings and steps for PT-nets.

DEFINITION 5.2. *A* marking *is a function* M *mapping from* P *into* $\mathbb{N}$ *while a* step *is a* non-empty *and* finite *multi-set over* T*. The sets of all markings and steps are denoted by* $\mathbb{M}$ *and* $\mathbb{Y}$*, respectively.*

We denote the set of multi-sets over a set A by A_{MS}.
The enabling and occurrence rules of a PT-net can now be defined.

DEFINITION 5.3. *A step* Y *is* enabled *in a marking* M*, denoted by* $M[Y\rangle$*, iff the following property is satisfied:*

$$\forall p \in P : \sum_{t \in Y} W(p, t) \leq M(p).$$

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*

$$\forall p \in P : M_2(p) = \left(M_1(p) - \sum_{t \in Y} W(p, t)\right) + \sum_{t \in Y} W(t, p).$$

Note the summations above are over a multi-set Y. This means that $W(p, t)$ and $W(t, p)$ appear as many times as t appears in Y.

We say that M_2 is directly reachable from M_1 by the occurrence of step Y, which is denoted by: $M_1[Y\rangle M_2$. $[M\rangle$ denotes the set of markings reachable from M.

Now we are ready to define modular nets. Some motivation and explanation of the individual parts of the definition are given immediately below it, and it is recommended to read both in parallel.

DEFINITION 5.4. *A* modular PT-net *is a triple* $MN = (S, PF, TF)$*, satisfying the following requirements:*

(i) S *is a finite set of* modules *such that:*
  (a) *each module,* $s \in S$*, is a PT-net:* $s = (P_s, T_s, W_s, M_{0_s})$*;*
  (b) *the sets of nodes corresponding to different modules are pair-wise disjoint*[1]*:* $\forall s_1, s_2 \in S : [s_1 \neq s_2 \Rightarrow (P_{s_1} \cup T_{s_1}) \cap (P_{s_2} \cup T_{s_2}) = \emptyset]$.

(ii) $PF \subseteq 2^P$ *is a finite set of* place fusion sets *such that:*
  (a) $P = \bigcup_{s \in S} P_s$ *is the set of all places of all modules;*
  (b) *for nodes* $x \in P \cup T$ *we use* $S(x)$ *to denote the module to which* x *belongs. For all* p *in* P *we define* $M_0(p) = M_{0_{S(p)}}(p)$*;*
  (c) *members of a place fusion set have identical initial markings:*

$$\forall pf \in PF : \forall p_1, p_2 \in pf : [M_0(p_1) = M_0(p_2)].$$

(iii) $TF \subseteq 2^T$ *is a finite set of* transition fusion sets *where:* $T = \bigcup_{s \in S} T_s$ *is the set of all transitions of all modules.*

[1]For the sake of simplicity, we use, in the examples, the same names for objects (places or transitions) belonging to different modules, but which have to be fused together.

EXPLANATION. (i) A modular PT-net contains a finite set of modules, each of them being a PT-net. These modules must have disjoint sets of nodes.

(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 initial marking. In the following we will denote by $EP \subseteq P$ ('external places') the set of all places which are members of a place fusion set and by $IP = P \setminus EP$ ('internal places'), all non-fused places. Note that 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. In the following, we will denote by $ET \subseteq T$ ('external transitions') the set of all transitions which are members of a transition fusion set and by $IT = T \setminus ET$ ('internal transitions'), all non-fused transitions. Note that we do not demand the transition fusion sets to be disjoint.

In the following definition, we introduce place groups and transition groups.

DEFINITION 5.5. *A* place group $pg \subseteq P$ *is an equivalence class of the smallest equivalence relation containing all pairs* $(p_1, p_2) \in P \times P$ *where:*

$$\exists pf \in PF : p_1, p_2 \in pf.$$

A transition group $tg \subseteq T$ *consists of either a single non-fused transition* $t \in IT$ *or all the members of a transition fusion set* $tf \in TF$.

The set of place groups is denoted by PG and the set of transition groups by TG.

Place groups and transition groups are defined very differently since a place can be a member of at most one place group while a transition can be a member of several transition groups. The reason is that a place group represents a shared resource. A transition can be a member of several transition groups as it can be synchronized with different transitions (sub-action of several more complex actions). Place groups form a partition of the set of places. This means that each place p is a member of exactly one place group which will be denoted $[p]$. Note that all place groups and transition groups have at least one element. From Definitions 5.4(ii) and 5.5, we know that all places of a place group will have the same initial markings, this allows us to define $M_0([p]) = M_0(p)$ without ambiguity.

Next, we extend the arc weight function W to place groups and transition groups:
$\forall (g_1, g_2) \in (PG \times TG) \cup (TG \times PG)$:

$$W(g_1, g_2) = \sum_{x \in g_1, y \in g_2} W(x, y).$$

Now, we extend the definitions of markings and steps to modular PT-nets.

DEFINITION 5.6. *A* marking *is a function* $M : PG \to \mathbb{N}$ *while a* step *is a* non-empty *and* finite *multi-set over TG. The sets of all markings and steps are denoted by* $\mathbb{M}$ *and* $\mathbb{Y}$*, respectively.*

The restriction of a marking M to a module s is denoted by M_s. The enabling and occurrence rules of a modular PT-net can now be expressed.

DEFINITION 5.7. *A step Y is* enabled *in a marking M, denoted by* $M[Y\rangle$*, iff the following property is satisfied:*

$$\forall pg \in PG : \sum_{tg \in Y} W(pg, tg) \leq M(pg).$$

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:*
$\forall pg \in PG$:

$$M_2(pg) = (M_1(pg) - \sum_{tg \in Y} W(pg, tg)) + \sum_{tg \in Y} W(tg, pg).$$

We say that M_2 is directly reachable from M_1 by the occurrence of step Y, which we also denote by: $M_1[Y\rangle M_2$. $[M\rangle$ denotes the set of markings reachable from M, and we generalize this notation to cover the following cases which are all useful in relation to modular PT-nets.

(i) M_2 is reachable from M_1 by the occurrence of an internal transition t: $M_1[[t\rangle M_2$, and $[[M\rangle$ denotes the set of all markings reachable from M by internal transitions only or no transition (i.e. $M \in [[M\rangle$).
(ii) M_2 is reachable from M_1 by the occurrence of a fused transition tf: $M_1[tf\rangle\rangle M_2$, and $[M\rangle\rangle$ denotes the set of all markings reachable from M by the occurrence of a fused transition. This implies that in general $M \notin [M\rangle\rangle$.
(iii) M_2 is reachable from M_1 by a sequence of internal transitions followed by a fused transition $\sigma = t_1 \ldots t_n tf$: $M_1[[\sigma\rangle\rangle M_2$, and $[[M\rangle\rangle$ denotes the set of all markings reachable by a sequence of internal transitions (or none) followed by a fused transition, i.e. the closure of $M[[\rangle\rangle$. As case $n = 0$ is allowed, $[M\rangle\rangle \subseteq [[M\rangle\rangle$.

For a local marking M_s in a module s, $[M_s\rangle_s$ denotes the set of markings reachable from M_s by occurrences of local transitions of module s only.

Next, we show that each modular PT-net has a behavioural equivalent PT-net. Some motivation and explanation of individual parts of the definition of the equivalent PT-net is given immediately below it, and it is recommended to read both in parallel. All names that refer to the equivalent PT-net are marked by an asterisk, e.g. M_0 refers to the initial marking of the modular PT-net and M_0^* to the initial marking of its equivalent PT-net.

DEFINITION 5.8. *Let a modular PT-net* $MN = (S, PF, TF)$ *be given. Then we define the* equivalent PT-net *to be* $PN^* = (P^*, T^*, W^*, M_0^*)$ *where:*

(i) $P^* = PG$.
(ii) $T^* = TG$.
(iii) $\forall (x^*, y^*) \in (P^* \times T^*) \cup (T^* \times P^*)$:

$$W^*(x^*, y^*) = W(x^*, y^*).$$

(iv) $\forall p^* \in P^* : M_0^*(p^*) = M_0(p^*)$.

EXPLANATION. (i) The equivalent PT-net has one place for each place group.

(ii) The equivalent PT-net has one transition for each transition group.

(iii) The weight associated with a pair (place group, transition group) is unchanged.

(iv) From Definitions 5.4(ii) and 5.5, we know that all places of a place group have the same initial marking and we know that all place groups have at least one member. The initial marking of a place group is determined by one of the members of the place group.

The following theorem shows that a modular PT-net and its equivalent PT-net have the same behaviour.

PROPOSITION 5.1. *Let MN be a modular PT-net and let* PN^* *be the equivalent PT-net. Then we have the following properties:*

1. $\mathbb{M} = \mathbb{M}^* \wedge M_0 = M_0^*$.
2. $\mathbb{Y} = \mathbb{Y}^*$.
3. $\forall M_1, M_2 \in \mathbb{M}, \forall Y \in \mathbb{Y}$:

$$M_1[Y\rangle_{MN} M_2 \Leftrightarrow M_1[Y\rangle_{PN^*} M_2.$$

Proof. (i) $\mathbb{M} = \mathbb{M}^*$, follows from Definitions 5.2, 5.6 and 5.8(i). From Definition 5.8(iv), the two initial markings are identical. Thus, $M_0 = M_0^*$.

(ii) From Definition 5.2, $\mathbb{Y}^*$ consists of all non-empty and finite multi-sets in T^*_{MS}. From Definition 5.6, $\mathbb{Y}$ consists of all non-empty and finite multi-sets in TG_{MS}. From Definition 5.8(ii), $T^* = TG$. Thus $\mathbb{Y} = \mathbb{Y}^*$.

(iii) First, we prove that the enabling rules coincide, i.e.

$$M_1[Y\rangle_{MN} \Leftrightarrow M_1[Y\rangle_{PN^*}.$$

From Definition 5.3 it follows that $M_1[Y\rangle_{PN^*}$ iff:

$$\forall p^* \in P^* : \sum_{t^* \in Y} W^*(p^*, t^*) \leq M_1(p^*),$$

which by Definition 5.8(i) and (ii) is equivalent to:

$$\forall pg \in PG : \sum_{tg \in Y} W^*(pg, tg) \leq M_1(pg).$$

which by Definition 5.8(iii) is equivalent to:

$$\forall pg \in PG : \sum_{tg \in Y} W(pg, tg) \leq M_1(pg).$$

From Definition 5.7, it follows that this is exactly the enabling condition for $M_1[Y\rangle_{MN}$.

Next we must prove that the occurrence rules coincide, i.e.

$$M_1[Y\rangle_{MN} M_2 \Leftrightarrow M_1[Y\rangle_{PN^*} M_2.$$

This part of the proof can be structured like the part regarding enabling rules and we will not include it. □

In Section 2, we claimed that the presented modular PT-nets and the PT-net, given as examples, were equivalent according to the behaviour. This can be checked using Definition 5.8 and Proposition 5.1.

6. PLACE INVARIANT ANALYSIS

In this section we show how the concepts of place invariants and place flows can be extended to modular PT-nets. Place invariants can be used in the proofs of properties of a PT-net, e.g. to show that there is no dead marking. In this paper we focus on the concepts of place invariants and place flows, more than on the use of invariants in the proof of properties of PT-nets.

6.1. Place flows and invariants of PT-nets

In this subsection we recall the concepts of place flow and place invariant.

DEFINITION 6.1. *For a PT-net $PN = (P, T, W, M_0)$, a* weight function *is a function F mapping from P into $\mathbb{Z}$:*

(i) *F is a* place flow *iff:*

$$\forall t \in T : \sum_{p \in P} F(p) * W(p, t) = \sum_{p \in P} F(p) * W(t, p);$$

(ii) *F determines a* place invariant *iff:*

$$\forall M \in [M_0\rangle : \sum_{p \in P} F(p) * M(p) = \sum_{p \in P} F(p) * M_0(p).$$

A weight function F maps each place p to an integer $F(p)$.

(i) We say that a transition t *is flow preserving* with respect to a weight function F iff t possesses the property described in (i). For a subset of transitions $T'' \subseteq T$, we say that F *is T''-flow preserving* iff all $t \in T''$ are flow preserving, with respect to F.
(ii) The intuition of a transition t being flow preserving is that t removes—when the weights of F are taken into account—the same number of tokens as it adds.

For a given marking M, we calculate the weighted sum as the sum of the weights multiplied by the marking of the individual places. The weight function F determines an invariant iff all reachable markings have the same weighted sum.

Note that any linear combination of two place flows is a place flow, i.e. if F_1 and F_2 are place flows, and $z_1, z_2 \in \mathbb{Z}$ then $z_1 * F_1 + z_2 * F_2$ is a place flow. The weight function which assigns zero weights to all places is always a place flow. We say that a place p is included in F if $F(p) \neq 0$. The main reason for introducing place flows is the difficulty checking place invariants on the total set of reachable states. Place flows can be checked on the structure of the PT-net. In practice we do not need to sum through all the places, it is sufficient to sum through the input places of t for the first sum, and through the output places of t for the second sum.

The following theorem describes the relationship between place invariants and place flows.

THEOREM 6.1. *Let a PT-net with no dead transition be given and let F be a weight function.*

F is a place flow $\Leftrightarrow$ F determines a place invariant.

Proof. The theorem is part of the classical theory for invariant analysis (see e.g. [9]).

$\Leftarrow$ is satisfied for all PT-nets with no dead transition.

$\Rightarrow$ is satisfied for all PT-nets. □

6.2. Place invariants of modular PT-nets

In this section we show how the formal definitions of place invariants and place flows can be given for modular PT-nets. This subsection is structured like the previous one and it should be easy to compare the definitions given for PT-nets and modular PT-nets.

DEFINITION 6.2. *For a modular PT-net $MN = (S, PF, TF)$, a* weight function *is a function F mapping from PG into $\mathbb{Z}$.*

(i) *F is a* place flow *iff:*
$\forall tg \in TG$:

$$\sum_{pg \in PG} F(pg) * W(pg, tg) = \sum_{pg \in PG} F(pg) * W(tg, pg).$$

(ii) *F determines a* place invariant *iff:*
$\forall M \in [M_0\rangle$:

$$\sum_{pg \in PG} F(pg) * M(pg) = \sum_{pg \in PG} F(pg) * M_0(pg).$$

A weight function F of a modular PT-net maps each place group pg into an integer $F(pg)$.

(i) We say that a transition group tg *is flow preserving* with respect to a weight function F iff tg possesses the property described in Definition 6.2(i). For a subset of transitions groups $TG'' \subseteq TG$, we say that F *is TG''-flow preserving* iff all $tg \in TG''$ are flow preserving, with respect to F.

The intuition of a transition group tg being flow preserving is that tg removes—when the weights of F are taken into account—the same number of tokens as it adds.

(ii) For a given marking M we calculate the weighted sum as the sum of the weights multiplied by the marking of the individual place groups. The weight function F determines an invariant iff all reachable markings have the same weighted sum.

THEOREM 6.2. *Let a modular PT-net with no dead transition be given and let F be a weight function.*

$$F \textit{ is a place flow} \Leftrightarrow F \textit{ determines a place invariant.}$$

Proof. The theorem can be proved similarly to Theorem 6.1, we just need to consider place groups and transition groups instead of places and transitions.

$\Rightarrow$ is satisfied for all modular PT-nets.
$\Leftarrow$ is satisfied for all modular PT-nets with no dead transition.

6.3. How to find place invariants of modular PT-nets

In the examples presented in sections 2 and 3, we have shown some compositions of place invariants and place flows, using either place fusion only or transition fusion only.

We use the term *global* weight function for a weight function of the entire modular PT-net, while we use the term *local* weight function for a weight function of a single module. In the present section, we state and prove a number of theorems specifying how local place flows and local place invariants are related to global place flows and global place invariants.

DEFINITION 6.3. *Let $MN = (S, PF, TF)$ be a modular PT-net.*

(i) *A set of local weight functions $\{F_s\}_{s\in S}$ of the modules is* consistent *iff they assign the same weight to all members of each place group:*

$$\forall pg \in PG : \forall p_1, p_2 \in pg : F_{S(p_1)}(p_1) = F_{S(p_2)}(p_2).$$

(ii) *A global weight function F of MN* determines a consistent set of local weight functions $\{F_s\}_{s\in S}$ *of the modules:*

$$\forall p \in P : F_{S(p)}(p) = F([p]).$$

(iii) *A consistent set of local weight functions $\{F_s\}_{s\in S}$ of the modules of MN* determines a global weight function *F:*

$$\forall pg \in PG, \forall p \in P : pg = [p] \Rightarrow F(pg) = F_{S(p)}(p).$$

Note that the construction fulfils: if F_1 determines $\{F_s\}_{s\in S}$ and $\{F_s\}_{s\in S}$ determines F_2 then $F_1 = F_2$.

THEOREM 6.3. *Let $MN = (S, PF, TF)$ be a modular PT-net and let $\{F_s\}_{s\in S}$ be a consistent set of local weight functions of the modules which determine the global weight function F. Then we have*

$$[\forall s \in S : F_s \textit{ is a place flow of } T_s] \\ \Rightarrow F \textit{ is a place flow of } MN.$$

Proof. The theorem follows directly from the observation that all transitions of the individual modules are flow preserving, i.e. we know that each member of a transition group is flow preserving. This is a much stronger demand than the transition group being flow preserving as a group. □

In the next theorem we show how Theorem 6.3 can be extended to place invariants if we consider modular PT-nets without place fusion.

THEOREM 6.4. *Let $MN = (S, \emptyset, TF)$ be a modular PT-net without place fusion, and let $\{F_s\}_{s\in S}$ be a consistent set of local weight functions of the modules which determine the global weight function F. Then we have*

$$[\forall s \in S : F_s \textit{ determines a place invariant of module } s] \\ \Rightarrow F \textit{ determines a place invariant of } MN.$$

Proof. From the definition of enabling for modular PT-nets, we know that a transition group will only be enabled if all transitions of the group are enabled. This means that the set of reachable states for the modular PT-net is covered by the set of reachable states of the local modules. □

Note that we can find place invariants of the total system which cannot be expressed as place invariants of the individual modules.

THEOREM 6.5. *Let $MN = (S, PF, \emptyset)$ be a modular PT-net* without transition fusion, *and let F be a global weight function of MN which determines a set of local weight functions $\{F_s\}_{s\in S}$. Then we have*

$$F \textit{ is a place flow of } MN \\ \Rightarrow [\forall s \in S : F_s \textit{ is a place flow of } T_s].$$

Proof. Since transition fusion is not used, we know that each transition group consists of exactly one member. Thus each individual transition is flow preserving. □

From Definition 6.3 and Theorem 6.5 we know that we can find all place flows of the total system from the place flows of the individual modules.

THEOREM 6.6. *Let $MN = (S, PF, TF)$ be a modular PT-net and let F be a global weight function of MN which determines a set of local weight functions $\{F_s\}_{s\in S}$. Then we have*

$$F \textit{ is a place flow of the modular PT-net} \\ \Leftrightarrow [\forall s \in S : F_s \textit{ is a place flow of } IT_s] \\ \wedge [TF \textit{ is flow preserving for } F].$$

Proof. In this proof, it is sufficient to establish a correspondence between the transition groups and the union of the set of internal transitions and the set of fusion sets. If a transition group contains exactly one transition it corresponds to an internal transition and otherwise it corresponds to a transition fusion set. □

Theorem 6.6 is a key result. All place flows of a modular PT-net can be determined from consistent sets of weight functions which are place flows for the internal transitions *IT* and are flow preserved by all transition fusion sets. A corresponding theorem for coloured Petri nets (CP-nets) has been shown for place fusion only in [10].

To illustrate how we envision the use of the results from this section, we describe how a place flow could be found for a modular PT-net. We will assume that a modular PT-net has mainly internal, i.e. non-fused, nodes, and the main work is to check that the internal transitions are flow preserving. When performing place invariant analysis one can be interested either in all possible flows of the system or in a few, but descriptive, place flows. Theorem 6.6 allows one to calculate all possible place flows in a modular way: calculate all flows for the internal transitions and combine those which are consistent and, finally, check the transition fusion sets.

The result of this theorem is also attractive when using an interactive process where the weights of places are gradually added. It consists in starting with a single module and specifying weights that are flow preserved by all internal transitions. After this, the rest of the weight functions can be restricted, using the weights of the module. Both place fusion sets and transition fusion sets can be used. We know that the weight functions must be consistent and this means that all places of a place fusion set must have the same weight. We also know that the transition fusion sets must be flow preserving and if only one of the surrounding places needs a weight, it can be calculated directly from the known weights. After these restrictions have been applied, the internal transitions of the next module can be checked. In the end it is necessary to check that all transition fusion sets are flow preserving.

In the next section, we present another main analysis method, i.e. modular state spaces.

7. STATE SPACES

In the following, we formally define modular state spaces; but first we introduce some notations and recall the definition of state spaces of PT-nets.

7.1. State spaces of PT-nets

We want to represent state spaces as directed graphs. Since we want to be able to have multiple arcs between pairs of nodes we use the following definition from [9].

DEFINITION 7.1. *A* directed graph *is a tuple* $DG = (V, A, N)$ *such that:*

(i) V *is a set of* nodes *(or* vertices*);*

(ii) A *is a set of* arcs *(or* edges*) such that*

$$V \cap A = \emptyset;$$

(iii) N *is a* node *function. It is defined from* A *into* $V \times V$.

The node function N maps an arc a into a pair of nodes (x_1, x_2), where x_1 is the source and x_2 the destination of arc a.

DEFINITION 7.2. *Let* $PN = (P, T, W, M_0)$*, be a PT-net. The* state space *of PN is the directed graph* $SS = (V, A, N)$*, where:*

(i) $V = [M_0\rangle$;
(ii) $A = \{(M_1, t, M_2) \in V \times T \times V \mid M_1[t\rangle M_2\}$;
(iii) $\forall a = (M_1, t, M_2) \in A : N(a) = (M_1, M_2)$.

The state space contains a node for each reachable marking and an arc for each possible transition occurrence.

For two nodes v_s and v_e we use $DPF(v_s, v_e)$ to denote the set of all directed finite paths connecting v_s to v_e, i.e. all finite sequences of nodes and arcs $v_1, a_1, v_2, a_2, \ldots, v_n$ where $v_1 = v_s$, $v_n = v_e$, and for all i in $1, \ldots n$, $N(a_i) = (v_i, v_{i+1})$.

Next we consider SCCs. Two nodes $v_1, v_2 \in V$ are *strongly connected* iff there exists a finite directed path which starts in v_1 and ends in v_2 and a finite directed path which starts in v_2 and ends in v_1. It is easy to see that strong-connectedness is an equivalence relation and hence determines a set of disjoint equivalence classes. An SCC is a directed graph $DG' = (V', A', N')$, where $V' \subseteq V$ is an equivalence class of strongly connected nodes, $A' \subseteq A$ are all those arcs where both the source and destination belong to V', and N' is the restriction of N to A'.

The set of all SCCs is denoted by *SCC*. For a node $v \in V$ and a component $c \in SCC$ we use the notation $v \in c$ to denote that v is one of the nodes in c. A similar notation is used for arcs. We use v^c to denote the component to which $v \in V$ belongs. We also denote the source (respectively destination) of an arc a by $srce(a)$ (respectively $dest(a)$).

DEFINITION 7.3. *The directed graph* $SS^* = (V^*, A^*, N^*)$ *is the* SCC-graph *of state space SS iff the following properties are satisfied:*

1. $V^* = SCC$;
2. $A^* = \{a \in A \mid srce(a)^c \neq dest(a)^c\}$;
3. $\forall a \in A^* : N^*(a) = (srce(a)^c, dest(a)^c)$.

Definition 7.3 allows us to talk about the SCC graph of a state space and this will turn out to be a very useful tool when we have to specify efficient proof rules for deciding properties of systems. This will be discussed in Section 8.

7.2. Modular state spaces

In this section we consider modular PT-nets with transition fusion only, i.e. no place fusion. To conclude that this is not a severe restriction please recall the arguments of Section 4.

In the definition of modular state spaces we need a compact notation to capture the states reachable from M in all the individual modules, i.e. $[[M\rangle$. It turns out that we can use a product of SCCs of the individual modules to express this representative node: for any reachable marking M, we use $M^{\not{q}}$ to denote the product of SCCs M_s^c of the individual modules:

$$\forall M \in [M_0\rangle : M^{\not{q}} = \prod_{s \in S} M_s^c.$$

This notation is also extended to a set X of markings:

$$X^{\not{q}} = \bigcup_{M \in X} \{M^{\not{q}}\}.$$

In the definition of a modular state space, we have two parts: the synchronization graph and the state spaces of the individual modules. The definition is followed by an explanation and both should be read in parallel.

DEFINITION 7.4. *Let $MN = (S, \emptyset, TF)$ be a modular PT-net* without place fusion *and with the initial marking M_0. The* modular state space *of MN is a pair $MSS = ((SS_s)_{s \in S}, SG)$, where:*

(i) $SS_s = (V_s, A_s, N_s)$ *is the* local state space *of module s:*

  (a) $V_s = \bigcup_{v \in (V_{SG})_s} [v\rangle_s$,
  (b) $A_s = \{(M_1, t, M_2) \in V_s \times IT_s \times V_s \mid M_1[t\rangle M_2\}$,
  (c) $\forall a = (v_1, t, v_2) \in A_s : N_s(a) = (v_1, v_2)$;

(ii) $SG = (V_{SG}, A_{SG}, N_{SG})$ *is the* synchronization graph *of MN:*

  (a) $V_{SG} = [[M_0\rangle\rangle^{\not{q}} \cup \{M_0^{\not{q}}\}$,
  (b) $A_{SG} = \{(M_1^{\not{q}}, (M_1', tf, M_2), M_2^{\not{q}}) \in V_{SG} \times ([M_0\rangle \times TF \times [M_0\rangle) \times V_{SG} \mid M_1' \in [[M_1\rangle \wedge M_1'[tf\rangle M_2\}$,
  (c) $\forall a = (v_1, X, v_2) \in A_{SG} : N_{SG}(a) = (v_1, v_2)$.

EXPLANATION. (i) The definition of the state space graphs of the modules is a generalization of the usual definition of state spaces:

(a) the set of nodes of the state space graph of a module contains all states locally reachable from any node of the synchronization graph;
(b) likewise the arcs of the state space graph of a module correspond to all enabled internal transitions of the module;
(c) an arc (v_1, t, v_2) starts from node v_1 and ends in node v_2.

(ii) Each node of the synchronization graph is labelled by an $M^{\not{q}}$ and represents all the nodes reachable from M by occurrences of local transitions only, i.e. $[[M\rangle$. The definition of $M^{\not{q}}$ ensures that any marking having the same set of markings reachable by internal transitions will be represented by the same node in V_{SG}. The synchronization graph contains the information on the nodes reachable by occurrences of fused transitions:

(a) the nodes of the synchronization graph represent all markings reachable from another marking by a sequence of internal transitions followed by a fused transition. The initial node is also represented;
(b) the arcs of the synchronization graph represent all occurrences of fused transitions;
(c) an arc (v_1, X, v_2) starts from node v_1 and ends in node v_2.

The state space graphs of the modules only contain local information, i.e. the markings of the module and the arcs corresponding to local transitions, but not the arcs corresponding to fused transitions. All the information concerning the occurrences of fused transitions is stored in the synchronization graph. This structure is designed in order to efficiently check properties directly in modular state spaces, as will be shown in Section 8.

The modular state space can be unfolded into an ordinary state space. Let $\mathbb{M}$ denote the set of markings of the full system and $\mathbb{M}_s$ the set of markings of module s. The explanation is given just below the definition and both should be read in parallel.

For a marking m_s of module s, we use m_s^* to denote the marking of the full system where all places of all other modules are empty.

DEFINITION 7.5. *Let $MN = (S, PF, TF)$ be a modular PT-net and $MSS = ((SS_s)_{s \in S}, SG)$ its modular state space. The* unfolded state space *of MSS is $SS = (V, A, N)$, where:*

(i) $V = \bigcup_{v \in V_{SG}} [[v\rangle$;
(ii) $A = \bigcup_{(v,(m,[t],m'),v') \in A_{SG}} \{(m, [t], m')\} \cup \bigcup_{m \in V, s \in S, (m_s, t, m_s') \in A_s} \{(m, t, (m + {m_s'}^*) - m_s^*)\}$;
(iii) $\forall a = (v_1, X, v_2) \in A : N(a) = (v_1, v_2)$.

EXPLANATION. (i) The set of nodes of the equivalent state space is the set of markings represented by nodes in V_{SG}, i.e. the set of markings reachable by internal transitions from any of the nodes of V_{SG}.

(ii) An arc label of the synchronization graph is an arc of the equivalent state space. For each marking m, if a local transition is enabled, there is a corresponding arc in the equivalent state space. The marking obtained is changed for the module concerned as specified in its state space graph.

(iii) An arc (v_1, X, v_2) starts from node v_1 and ends in node v_2.

The following theorem states that the equivalent ordinary state space of a MSS and the state space of the equivalent PT-net of MN are the same.

THEOREM 7.1. *Let MN be a modular PT-net, MSS its modular state space and PN its equivalent PT-net. Let SS_{MSS} be the unfolded state space of MSS and SS_{PN} the state space of PN:*

SS_{MSS} is isomorphic to SS_{PN}.

Proof. The main idea of the proof is to follow the constructive definitions given in Definitions 5.8, 7.2, 7.4, and 7.5. Definition 5.8 shows how we can construct an equivalent PT-net PN from a modular PT-net MN; note that we have a transition in PN for each transition group in MN. Using Definition 7.2, we can then construct the state space SS_{PN} corresponding to PN. Using Definition 7.4, we can construct the modular state space MSS from MN and finally we must show that Definition 7.5 maps *MSS* to an unfolded state space SS_{MSS} which by construction is isomorphic to SS_{PN}. □

We now define an abstract algorithm to construct modular state spaces. We use primitive functions similar to those of [9] (Proposition 1.4). $Waiting_{SG}$ is a set of nodes of the synchronization graph. $Node_{SG}(v)$ is a procedure which creates a new node v in the synchronization graph and adds v to $Waiting_{SG}$. If v is already a node, $Node_{SG}(v)$ has no effect. Analogously, $Arc_{SG}(v_1, (M_1, t, M_2), v_2)$ creates an arc with source v_1, destination v_2, and inscription (M_1, t, M_2) in the synchronization graph, if it does not exist yet. $Next_{SG}(v)$ is used to denote the set of possible *next moves* using fused transitions. These moves can be performed from any marking in $[[v\rangle$, i.e. reachable from v by occurrences of local transitions only. In addition to this, we use the function *AddInternalSuccessors*(M) which takes a marking and develops the corresponding state space of each module, i.e. all local markings reachable from M_s using transitions from IT_s only. This procedure is similar to the state spaces construction algorithm for all modules, with initial marking M_s. *AddInternalSuccessors*(M) is able to determine the SCCs of M_s for each of the local state spaces and from these construct $M^{\mathcal{C}}$ as the returned value.

PROPOSITION 7.1. *The following algorithm constructs the modular state space:*

```
$Waiting_{SG} := \emptyset$
$v_0 := AddInternalSuccessors(M_0)$
$Node_{SG}(v_0)$
repeat
    select a node $v_1 \in Waiting_{SG}$
    for all $(M_1, t, M_2) \in Next_{SG}(v_1)$ do
    begin
        $v_2 := AddInternalSuccessors(M_2)$
        $Node_{SG}(v_2)$
        $Arc_{SG}(v_1, (M_1, t, M_2), v_2)$
    end
    $Waiting_{SG} := Waiting_{SG} \setminus \{v_1\}$
until $Waiting_{SG} = \emptyset$
```

This algorithm is a generalization of the one for constructing state spaces. The generalizations follow directly from Definition 7.4.

In the next section, we will show how to prove properties directly on the modular state space, i.e. without unfolding to the ordinary state space.

8. PROVING PROPERTIES

Here, we will prove properties for modular PT-nets. It has been our main concern to ensure that all properties are defined in such a way that they are consistent with the definitions known for PT-nets. For each of the PT-net properties we present proof rules which take advantage of the characteristics of modular state spaces. We also sketch algorithms showing how these proof rules can be implemented in order to obtain efficient analysis of modular PT-nets and avoid unfolding to ordinary state spaces.

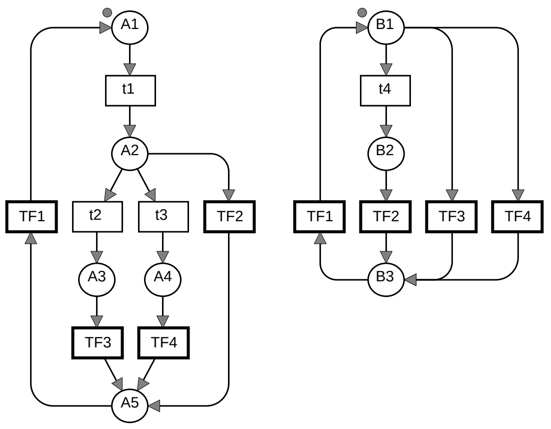

FIGURE 12. Modular PT-net with modules A and B.

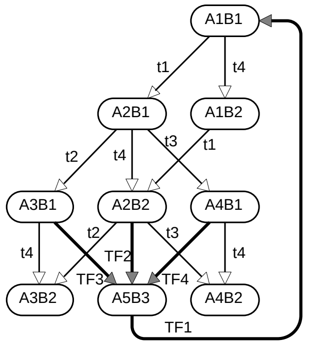

FIGURE 13. The full state space of the system.

We first introduce the example that will be used throughout this section. In Figure 12 we present a modular PT-net consisting of two modules A and B which share four common transitions TF1, TF2, TF3, and TF4. Module A is presented on the left-hand side while module B is on the right-hand side. This system will be used to illustrate the properties in the following subsections.

The modular PT-net is equivalent to a PT-net where the transitions shared by both modules are fused, i.e. TF1 of module A is fused with TF1 of module B, TF2 of module A is fused with TF2 of module B, and so on. The occurrence graph of this PT-net is presented in Figure 13. The modular state space of the system is presented in Figure 14. Note that we do not distinguish between nodes and SCCs, since they all contain exactly one node.

8.1. Reachability

Here, our purpose is to find whether a given marking M is reachable or not. The *set of ancestors* of a local marking M_s in the state space graph of module s is the set of SCCs from which M_s can be reached, i.e. for all s in S and for all local

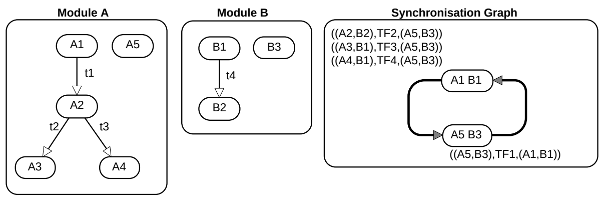

FIGURE 14. The modular state space.

markings M_s in V_s, we define:
$anc_s(M_s) = \{M_i^c \mid M_i \in V_s \wedge M_s \in [M_i\rangle_s\}$.

We now express the reachability properties.

PROPOSITION 8.1.

(i) $M \in [M_0\rangle \Leftrightarrow \exists v \in V_{SG} : M \in [[v\rangle$.
(ii) $M \in [M_0\rangle \Leftrightarrow [(\forall s \in S : M_s \in V_s)$
$\wedge((\prod_{s\in S} anc_s(M_s) \cap V_{SG}) \neq \emptyset)]$.

EXPLANATION. (i) A global marking is reachable iff it is internally reachable from one of the nodes in V_{SG}.

(ii) The first part of the conjunction ensures that all local markings are reachable. This is a necessary but not sufficient condition, and the second part of the conjunction ensures that there exists a node in V_{SG} from where the required combination of local states is reachable.

Proof. (i) Follows directly from Definition 7.4.

(ii) The first part of the conjunction ensures that the ancestors of all M_s are well defined. Reachability then follows from Definition 7.4 and the definition of function anc_s. □

SKETCH OF ALGORITHM. The process to check that a marking M is reachable can easily be implemented by first looking at the restrictions of M to the modules. If for one module s, M_s is not in SS_s, then M is not reachable. Otherwise, we check if there exists a node v in the synchronization graph from which M is locally reachable. This can be done efficiently, using the information of the SCCs of the modules.

EXAMPLE. Let us apply Proposition 8.1(ii) to the example presented in Figures 12–14, to check whether A4B2 is reachable or not. Node A4 is in SS_A. Node B2 is in SS_B. Node A4B2 is locally reachable from node A1B1: the ancestors of A4 in SS_A are A4, A2 and A1, and the ancestors of B2 in SS_B are B2, B1. Thus all the conditions are satisfied and A4B2 is reachable. We also check if A5B2 is reachable. Node A5 is in SS_A. Node B2 is in SS_B. The only ancestor of A5 in SS_A is itself, the ancestors of B2 in SS_B are B2, B1. Thus the cross-products of ancestors are {A5B2, A5B1} which are not in SG. Hence, the last condition of the proposition is not satisfied and A5B2 is not reachable.

8.2. Dead markings

We will now give properties to find dead markings.

PROPOSITION 8.2.

(i) $M \in [M_0\rangle$ *is dead* $\Leftrightarrow$
$[(\forall s \in S : \forall(m_1, t, m_2) \in A_s : m_1 \neq M_s)$
$\wedge(\forall(v_1, (M_1, tf, M_2), v_2) \in A_{SG} : M_1 \neq M)]$.
(ii) $M \in [M_0\rangle$ *is dead* $\Leftrightarrow$
$[(\forall s \in S : (M_s)^c \in Term(SCC_s) \cap Trivial(SCC_s))$
$\wedge(\forall(v_1, (M_1, tf, M_2), v_2) \in A_{SG} : M_1 \neq M)]$.
(iii) $M \in [M_0\rangle$ *is not dead* $\Leftarrow$
$\exists s \in S : \forall m \in V_s : \exists(m, t, m') \in A_s$.
(iv) $M \in [M_0\rangle$ *is not dead* $\Leftarrow$
$\exists s \in S : Term(SCC_s) \cap Trivial(SCC_s) = \emptyset$.

EXPLANATION. (i) A marking is dead iff there is no enabled transition, neither local nor fused.

(ii) A marking is dead iff it belongs to terminal and trivial components of all local state spaces and does not enable any fused transitions. We use function *Term* which returns the set of terminal SCCs and function *Trivial* which returns the set of trivial SCCs. An SCC is terminal if it has no outgoing arcs, and it is trivial if it has exactly one node and no arc.

(iii) We know that there is no reachable dead marking if there exists a local module which for any marking has an enabled transition.

(iv) We know that there is no reachable dead marking if there exists a local module without strongly connected components being both terminal and trivial.

Proof. (i) $\Rightarrow$ Let M be a reachable dead marking. No local transition is enabled. Thus, for all s in S, M_s cannot be the source of an arc. No fused transition is enabled. Hence there is no arc labelled by M in SG.

$\Leftarrow$ The proof is similar to that of $\Rightarrow$.

(ii) The right-hand sides of (i) and (ii) are equivalent.

(iii) Let us suppose that there exists a module s in S such that all nodes have at least one outgoing arc. Then from each reachable marking a local transition of s is enabled. Thus M is not dead.

(iv) The right-hand sides of (iii) and (iv) are equivalent. □

SKETCH OF ALGORITHM. Here we want to find all reachable dead markings of the system using Proposition 8.2(ii). We will consider all the nodes of SG. For each node m, we will only consider for all m_s the leaves of $[m_s\rangle_s$, in the state space graph of module s. These are the terminal and trivial M_s.

We construct the sets of $\prod_{s\in S} M_s^c$ for all the SG nodes. We remove from this set all the markings labelling SG arcs. Thus the remaining markings satisfy the conditions of the proposition above and form the set of all reachable dead markings.

EXAMPLE. We apply this algorithm to the example presented in Figures 12–14, to find all reachable dead markings.

There are two nodes in SG labelled by the SCCs' cross-products A1B1 and A5B3. The set of trivial leaves of $[\text{A1}\rangle_A$ is {A3, A4}, the one of $[\text{B1}\rangle_B$ is {B2}. The set of trivial leaves of $[\text{A5}\rangle_A$ is {A5}, the one of $[\text{B3}\rangle_B$ is {B3}. Thus the set of cross-products to check is {A3B2, A4B2, A5B3}. The set of arc labels in *SG* is {A3B1, A4B1, A2B2, A5B3}. Hence, the set of reachable dead markings is {A3B2, A4B2}.

8.3. Liveness

We now want to determine whether a given set of transitions is live or not. We first express the liveness properties for a fused transition, then for a local one and finally for a set of transitions in general.

PROPOSITION 8.3.

(i) *$tf \in TF$ is live* $\Leftrightarrow$
$[\forall scc \in Term(SCC_{SG}) : tf \in Trans(scc)]$
$\wedge[\forall v \in V_{SG} : \forall M \in [[v\rangle :$
$(\forall s \in S : M_s^c \in Term(SCC_s)) \Rightarrow$
$\exists(v, (M_1, tf', M_2), v_2) \in A_{SG} : M_1 \in [[M\rangle\ ].$

(ii) *$t \in IT_s$ is live* $\Leftrightarrow$
$[\forall scc \in Term(SCC_{SG}) :$
$\exists v \in scc : t \in Trans([v_s\rangle_s)\ ]$
$\wedge[\forall v \in V_{SG} : \forall M \in [[v\rangle :$
$(M_s^c \in Term(SCC_s)) \Rightarrow$
$(t \in Trans(M_s^c)$
$\vee\exists(v, (M_1, tf, M_2), v_2) \in A_{SG} : M_1 \in [[M\rangle\ )].$

(iii) *$X \subseteq T$ is live* $\Leftrightarrow$
$[\forall scc \in Term(SCC_{SG}) :$
$(X \cap Trans(scc) \neq \emptyset$
$\vee\exists v \in scc : X \cap Trans([[v\rangle) \neq \emptyset]$
$\wedge[\forall v \in V_{SG} : \forall M \in [[v\rangle :$
$(\forall s \in S : M_s^c \in Term(SCC_s)) \Rightarrow$
$(\exists s \in S : X \cap Trans(M_s^c) \neq \emptyset)$
$\vee(\exists(v, (M_1, tf, M_2), v_2) \in A_{SG} : M_1 \in [[M\rangle\ )].$

EXPLANATION. We use SCC_{SG} to denote the set of SCCs of the synchronization graph. *Trans* maps an SCC into the set of transitions which occur in the labels of the arcs in the component. Similarly, we use *Trans* to map a set of reachable markings to the set of transitions which occur in the labels of the arcs between two nodes of the set.

(i) A fused transition tf is live iff it occurs in all terminal strongly connected components of the synchronization graph, and furthermore it is always possible to get to a combination of local states which enables a fused transition.

(ii) A transition local to module s is live iff, for all terminal SCCs of the synchronization graph, there exists a node which enables t, and furthermore it is always possible to get to a combination of local states which enables a fused transition, or to a state enabling t.

(iii) A set of transitions X is live iff for each terminal SCC of the synchronization graph there is an occurrence of some element of X, or some node enabling an internal transition in X. Furthermore it is always possible to get to a combination of local states which enables a fused transition, or to a state enabling an element in X.

Proof. (i) and (ii) are particular cases of (iii). Thus, we will only prove (iii).

$\Rightarrow$ Let X be a live set of transitions. This means that it is possible from any reachable marking to reach another marking in which one of the transitions of X is enabled. For each terminal SCC of the synchronization graph either there exists a fused transition in X belonging to the component ($X \cap Trans(scc) \neq \emptyset$) or there exists a node v in *scc* for which a local transition of X is enabled ($\exists v \in scc : X \cap Trans([[v\rangle) \neq \emptyset$). Moreover, from each node in the synchronization graph it is possible to reach a set of terminal SCCs either containing a local transition of X ($X \cap Trans(M_s^c) \neq \emptyset$) or enabling a fused transition ($\exists(v, (M_1, t, M_2), v_2) \in A_{SG} : M_1 \in [[M\rangle$). This ensures that it is possible from each node of the synchronization graph either to locally reach a marking enabling a transition of X or to attain another node in the synchronization graph. If we apply this last case several times, we will reach a node in a terminal SCC of the synchronization graph. From the first condition, one of its successors enables a transition of X.

$\Leftarrow$ The proof is similar to that of $\Rightarrow$. □

SKETCH OF ALGORITHM. Here, we want to check the liveness of a given set X of transitions. We mark the nodes that satisfy the first part of the condition on the right-hand side of (iii). To do that we mark the nodes in the SCC graphs of modules which enable or contain a local transition of X, as well as their ancestors. Then we mark the nodes in the terminal SCCs of the synchronization graph that are built from at least one marked SCC. We also mark the nodes of the terminal SCCs of the synchronization graph which enable a fused transition of X. Then, if there exists a terminal SCC of the synchronization graph without any marked node, the first part of the condition is not satisfied, thus X is not live. Otherwise, we have to check the second part of the condition. For each node in the synchronization graph we take the local successors which are built from terminal SCCs only. For each of these we check that it either contains a local transition of X or labels an arc of the synchronization graph. If one node does not satisfy this requirement X is not live, otherwise it is.

EXAMPLE. We apply this algorithm to the example presented in Figures 12–14, to check that $X = T$ is not live. We could of course deduce this property from the fact that the system has dead markings, but this is just an illustration of the algorithm to check liveness. In the SS_s of modules A and B we mark nodes A1, A2 and B1. The synchronization graph contains only one SCC. Node A1B1 is built from a marked local node (e.g. A1), so we mark it. Hence, the terminal SCC has a marked node, and the first part of the condition is satisfied. The terminal nodes we have to check now are A3B2, A4B2 and A5B3. Node A3B2 does not contain a transition of X nor labels an arc of the synchronization graph. Thus X is not live. Note that the same problem arises with A4B2, and that A5B3 satisfies the condition.

8.4. Home properties

Here, we want to check whether a given set of reachable markings is a home space or not. We first express the home properties for a marking and then for a set of markings.

PROPOSITION 8.4.

(i) *$M_H \in [M_0\rangle$ is a home marking* $\Leftrightarrow$
$[\forall scc \in Term(SCC_{SG}) : \exists v \in scc : M_H \in [[v\rangle\ ]$
$\wedge[\forall v \in V_{SG} : \forall M \in [[v\rangle :$
$(\forall s \in S : M_s^c \in Term(SCC_s)) \Rightarrow$
$(M_H \in [[M\rangle$
$\vee\exists(v, (M_1, tf, M_2), v_2) \in A_{SG} : M_1 \in [[M\rangle\ )].$

(ii) *$X \subseteq [M_0\rangle$ is a home space* $\Leftrightarrow$
$[\forall scc \in Term(SCC_{SG}) : \exists v \in scc : X \cap [[v\rangle \neq \emptyset]$
$\wedge[\forall v \in V_{SG} : \forall M \in [[v\rangle :$
$(\forall s \in S : M_s^c \in Term(SCC_s)) \Rightarrow$
$(X \cap [[M\rangle \neq \emptyset$
$\vee\exists(v, (M_1, tf, M_2), v_2) \in A_{SG} : M_1 \in [[M\rangle\ )].$

EXPLANATION. (i) A marking M_H is a home marking if it is always possible to reach this marking. To prove this we must show that all terminal SCCs of the synchronization graph have a node which contains M_H in its local successors; and we must show that if we cannot enable any transition fusion we must be able to reach M_H using internal transitions only.

(ii) Checking a home space is similar to (i), we just check for a non-empty intersection instead of membership.

Proof. The proof is similar to the proof of liveness. (i) is a particular case of (ii) where X contains only one marking. Thus, we will only prove (ii).

$\Rightarrow$ Let X be a home space. This means that it is possible, from any reachable marking, to reach a marking of X. Let *scc* be a terminal SCC of the synchronization graph. From a node v of *scc*, it is possible to reach any node in $[[v\rangle$. Then, from the hypothesis, $X \cap [[v\rangle \neq \emptyset$. Now, let v be a node of the synchronization graph, M be a marking in $[[v\rangle$ which is in a terminal SCC of all modules, then either $[[M\rangle$ contains a node of X or it is possible to leave v in order to reach another node in the synchronization graph.

$\Leftarrow$ The proof is similar to that of $\Rightarrow$. □

SKETCH OF ALGORITHM. Here we want to check whether a given set X of markings is a home space or not. We mark the nodes that satisfy the first part of the condition. To do that we mark the nodes of X in the SCC graphs of the modules as well as their ancestors. Then we look at the nodes in the terminal SCCs of the synchronization graph which are built only from marked SCCs. We mark those which contain at least one marking of X in their local successors. We have to do that because it might be the case that the component in one module is marked due to the restriction of a marking M_1 in X and the component in another module is marked due to the restriction of another marking M_2 in X. Then, if there exists a terminal SCC of the synchronization graph without any marked node, the first part of the condition is not satisfied, thus X is not a home space. Otherwise, we have to check the second part of the condition. For each node in the synchronization graph we take the local successors which are built from terminal SCCs only. For each of these, we check that it either contains a marking of X or labels an arc of the synchronization graph. If one node does not satisfy this requirement, X is not a home space, otherwise it is.

EXAMPLE. We apply this algorithm to the example presented in Figures 12–14, to check that X1 = {A2B2} is not a home space and that X2 = {A3B2, A4B2} is. Let us start with X1. In the SS_s of modules A and B we mark nodes A1, A2 and B1, B2. The synchronization graph contains only one SCC. Node A1B1 is built from only marked local nodes and has A2B2 as a local successor, so we mark it. Hence, the terminal SCC has a marked node and the first part of the condition is satisfied. The terminal nodes we have to check now are A3B2, A4B2 and A5B3. Node A3B2 has no outgoing arc. Thus X1 is not a home space. Note that the same problem arises with A4B2 and that A5B3 satisfies the condition.

Now let us check that X2 is a home space. In the state space graphs of modules A and B we mark nodes A1, A2, A3, A4 and B1, B2. The synchronization graph contains only one SCC. Node A1B1 is built only from marked local nodes, and has A3B2 as a local successor, so we mark it. Hence, the terminal SCC has a marked node and the first part of the condition is satisfied. The terminal nodes we have to check now are A3B2, A4B2 and A5B3. Both A3B2, A4B2 are in X2. Node A5B3 labels an arc of the synchronization graph. Thus X2 is a home space.

8.5. Boundedness

We explain how boundedness properties can be checked. Here we use the fact that we have no place fusion, i.e. that all places are members of one local module.

PROPOSITION 8.5. *For the boundedness properties, we have the following proof rules, valid for all $s \in S$, and all $p \in P_s$:*

(i) *$BestUpperBound(p) = \max_{M_s \in V_s} M_s(p)$;*
(ii) *$BestLowerBound(p) = \min_{M_s \in V_s} M_s(p)$.*

EXPLANATION. (i) The best upper bound for a place p in P_s can be found directly in the local state space of module s.

(ii) States the same result for lower bounds.

Proof. (i) The *BestUpperBound* for a given place is local to the module s of this place. As the state space graph of module s contains exactly the reachable markings of this module, we just have to find the *BestUpperBound* in SS_s.

(ii) The arguments in the proof of (i) also apply to (ii). □

REMARKS. Bounds of places can be generalized in several ways. Bounds can be defined for a set of places instead of a single place and even a general function can be applied to the reachable markings. Similar generalizations could be specified for modular state spaces. One important observation is that these general bounds can be checked very efficiently if they only depend on a single module, or if they can be expressed as a positive linear combination of bounds of the modules.

SKETCH OF ALGORITHM. The bounds of a single place are local properties. Thus they are very easy to check on the state space of the module containing place p. This allows us to investigate the state space graph of one module instead of the state space of the entire system.

EXAMPLE. We apply this to the example presented in Figures 12–14.

If we want to check that A1 is bounded by one, we traverse the nodes of module A, this means that it is necessary to check five nodes. Similarly, it is necessary to traverse three nodes to check the bound of a place of module B. In the ordinary state space it is always necessary to traverse the nine nodes.

To illustrate the remark on the extension to general bounds above, we want to check the mutual exclusion between A5 and B2. For that purpose, we use a function F so that $F(M) = M(\text{A5}) + M(\text{B2})$. The projections of this function on both modules are F_A and F_B such that: $F_A(M_A) = M_A(\text{A5})$ and $F_B(M_B) = M_B(\text{B2})$.

Now we compute the maxima of these functions for each node in *SG*. There are only two nodes A1B1 and A5B3. For the first node we get $\max(F_A) = 0$ and $\max(F_B) = 1$, thus $\max(F) = 1$ for this node. For the second node, we obtain $\max(F_A) = 1$ and $\max(F_B) = 0$, thus $\max(F) = 1$ for this node. Thus the maximum for function F in *SG* is one and the mutual exclusion between places A5 and B2 is proved.

If we investigate the complexity of the algorithms required in order to decide the properties described in this section we find that all of them are linear in the size of the modular state space. In particular, it should be noted that a number of properties of local modules can be checked more efficiently when using the modular state spaces than the ordinary state spaces.

In the next section we apply the results presented up to now to larger models.

9. LARGER EXAMPLES

There exists a large number of industrial applications of Petri nets, in particular for high-level Petri nets such as CP-nets. This is the reason why parts of this section discuss models which are not PT-nets. As explained in the introduction, the results shown in the previous sections can also be generalized to CP-nets.

9.1. An example of a modular approach to place invariants

We have tried to use the results from Section 6.3 to find place flows of the hierarchical CP-net described in [11]. This is a model describing a detailed design of the network management system of the RcPAX X.25 wide area network. It consists of 30 modules (pages), many of these having up to seven instances due to the reuse of pages. The modular approach made it easy to find the place invariants needed in the proof of properties which were local to a few pages. The modular approach also made it possible to compose place flows of the individual page instances into place flows of the total system. An example of a property which could be proved directly by means of a place invariant and which involved many pages was the preservation of packets in the system. The handling of packages was relatively complex and involved grouping packages into larger logical units. By adding extra places which would keep a log of the information passed on the network, it was possible to investigate how messages could be lost and check that the information which was re-established either matched the original message, or the originating sender would be notified. The work on modular place invariants was performed after the design of the model was completed, and not as an integrated part of the modelling process. It should also be noted that the work on the examples was performed by hand. Tool support will be necessary if place invariants should be used as part of the development of large descriptions. A similar approach could easily be applied to other hierarchical CP-net models, e.g. the ISDN Basic Rate Interface described in [12].

We have not investigated how a modular approach can be used for large systems related by means of transition fusion, since we have no models of this nature at our disposal. From our own experiments with small systems it seems to be possible to use a modular approach for larger ones too.

9.2. An example of a modular approach to occurrence graph analysis

To test the ideas of modular state spaces before doing an actual implementation we have investigated a larger example using an existing tool, Design/CPN [13].

Design/CPN supports analysis of CP-nets by means of state spaces. The facilities of Design/CPN allowed us to emulate the algorithm described in Proposition 7.1 by manually calling appropriate routines. The lack of full tool support is the reason for choosing an example which is still of a moderate size.

The aim of this testing is twofold: first of all we want to show how modular state spaces work for concrete examples, second we want to illustrate that the basic functionalities of the Design/CPN tool can be used as a basis for an implementation of a tool supporting analysis by means of modular state spaces.

The example we have chosen is constructed to reflect some of the important properties that we expect to find in industrial applications: the system consists of three modules, each of them has an initialization phase, a cyclic main phase and a termination phase. The modules communicate pairwise, and the communications lead to new behaviour of the modules. Thus this example has a structure similar to that of Figure 5, but using more modules.

The ordinary state space has 1728 nodes and 7368 arcs. The system has a single non-trivial SCC, which implies that the system has no dead markings and every reachable state is a home state.

The modular state space has four nodes and 54 arcs in the synchronization graph, and three modules with altogether 18 nodes and 18 arcs. The generation was performed using the facilities proposed in Design/CPN to specify breakpoints in the construction. Thus it was possible not to fire fused transitions or transitions of another module, but only build local parts of the graph. Moreover, the generation could be continued after determining the nodes obtained by firing a shared transition. If we inspect the SCC graphs of the modules we find that none of them has a trivial terminal component. According to Proposition 8.2(iv) this implies that the system has no dead state. To show that every reachable state is a home state, we use Proposition 8.4(i) and prove that the initial marking is a home state. First we observe that all combinations of local terminal components enable at least one fused transition, i.e. we cannot be trapped in the combination of local components. We conclude by checking that the initial marking is included in the nodes locally reachable in the terminal SCC of the synchronization graph.

In the example we have a system where communication does not involve all modules. This means the number of arcs in the synchronization graph grows with the number of states of the modules which do not take part in the communication. If we handle this case by introducing a special symbol denoting that a module does not participate in the communication the number of arcs of the synchronization graph will decrease from 54 to 10.

9.3. Practical use of modular analysis

The modular method to obtain place invariants is rather satisfactory since it works both for place fusion and transition fusion. Thus a superset of the actual flows that will be relevant for the entire system can be calculated locally to each module. Then their combination can be performed, leading to flows of the entire system. We have to note that from the experience of our groups a modularly designed net consists of different modules with often several instances of the same module. In that case the calculus of the flow would be the same for all instances and obtaining it for a single instance is sufficient. Moreover, modules do not participate in all flows, and that can be seen very easily when performing the combinations. This is an advantage compared to the non-modular calculus, where all the places would be examined. Our method could be seen as a module-guided heuristic for invariants calculus.

As concerns the modular occurrence graphs, the practical use of the method is not as obvious, but we are convinced of its relevance in practical cases.

In the worst case, where all the transitions of the modules are shared, the synchronization graph will be isomorphic to the ordinary occurrence graph, i.e. have the same number of nodes and arcs, while the state spaces of the modules will have no arcs and as many disconnected nodes as the restriction of the occurrence graph nodes to the corresponding module. Thus, in the worst case there will be more nodes and as many arcs.

In the best case, where there is no synchronization at all, the synchronization graph will contain exactly one node (the initial one) and no arc. The individual state spaces of the modules will be the restriction of the usual occurrence graph to each module. They are also the normal occurrence graph obtained for the module. Thus the synchronous product of the graphs will be avoided.

In a real application there would often be a few transitions to be fused, and thus we can expect to be not very far from the best case.

Another parameter to consider that affects the size of the synchronization graph, and thereby of the modular state space, is the number of modules. In the large example of Section 9.2, there were three modules and for each communication one of the modules did not participate. This implies that the local behaviour of this module appears in the initial markings of arcs in the synchronization graph, although it is useless. As suggested for the example, such a situation can be handled in practice by, for example, adding a flag. Hence even if there is a large number of modules, their local behaviour would not be duplicated.

10. CONCLUSION

In this paper, we have presented a framework for modular analysis of PT-nets. We have considered sets of individual PT-nets related by transition fusion and by place fusion. Transition fusion can be used to model synchronous actions, while place fusion can be used to model shared data. Modular PT-nets form a simple, but yet very general, framework to discuss analysis of structured net models.

We also introduced analysis of modular PT-nets by means of place flows. It allows us to determine place flows of the modular PT-net from the individual modules, only transition fusion needs to be checked globally. This means that it is not necessary to recompute all place flows when a single module is modified. Previous works (e.g. [10]) have focused on modules communicating via places only.

Finally, 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 a modular PT-net with place fusion into a modular PT-net using transition fusion only.

If the results of Definition 7.5 and Theorem 7.1 are used to construct the ordinary state space, then the modular state space method is only a faster 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.

Moreover, it is possible to check properties using the modular state space directly, i.e. without unfolding to the ordinary state space. When designing algorithms there is often a trade-off between time and space complexity. For state space analysis it is attractive to have a rather fast way to decide properties, but the state space explosion problem makes it absolutely necessary to minimize memory usage.

A similar approach was presented in [14], but it starts by constructing the full state space of the modules, i.e. states that may not be reachable in the full system, as we point out in Section 4. Several reduction methods have been proposed to avoid the state space explosion problem, but they have drawbacks: they do not allow one to check all properties of the system or lead to a construction of the full ordinary state space (e.g. [15, 16, 9, 17, 18, 19]).

One of the next steps in the development of the modular state space approach is to implement a first version of a tool supporting it. In Section 7, we showed an abstract algorithm for the construction of modular state spaces, and in Section 9, we argued that the capabilities of an existing tool could be used to emulate this algorithm. In Section 8, the discussion of each property included an abstract algorithm which can be used directly in the implementation of a tool.

Having access to tool support will allow us to test the ideas for industrial size models. As described in Section 9, many industrial models seem to have a structure which makes them suited for this kind of analysis, i.e. they consist of relatively independent parts.

REFERENCES

[1] Battiston, E., De Cindio, F. and Mauri, G. (1991) OBJSA nets systems: a class of high-level nets having objects as domains. In G. Rozenberg (ed.), *Advances in Petri Nets 1988 (Lect. Notes Comput. Sci., vol. 340)*, pp. 20–43. Springer, New York. Also in Jensen, K. and Rozenberg, G. (eds) (1991) *High-level Petri Nets: Theory and Application*, pp. 189–212. Springer.

[2] Christensen, S. and Damgaard Hansen, N. (1994) Coloured Petri nets extended with channels for synchronous communication. In R. Valette (ed.), *Application and Theory of Petri Nets 1994 (Lect. Notes Comput. Sci., vol. 815)*, pp. 159–178. Springer, New York. Also available as: Daimi PB-390, ISSN 0105-8517.

[3] Huber, P., Jensen, K. and Shapiro, R. M. (1990) Hierarchies in coloured Petri nets. In G. Rozenberg (ed.), *Advances in Petri Nets 1990 (Lect. Notes Comput. Sci., vol. 383)*, pp. 342–416. Springer, New York. Also in Jensen, K. and Rozenberg, G. (eds) (1991) *High-level Petri Nets: Theory and Application*, pp. 215–243. Springer.

[4] Jensen, K. (1992) *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 1, Basic Concepts. Monographs in Theoretical Computer Science*. Springer.

[5] Jensen, K. (1986) Coloured Petri nets. In G. Rozenberg (ed.), *Advances in Petri Nets 1986, Part I (Lect. Notes Comput. Sci., vol. 254)*, pp. 248–299. Springer, New York.

[6] Reisig, W. (1991) Petri nets and algebraic specifications. *Theoret. Comput. Sci.*, **80**, 1–34. Also in Jensen, K. and Rozenberg, G. (eds) (1991) *High-level Petri Nets: Theory and Application*, pp. 137–170. Springer.

[7] Christensen, S. and Petrucci, L. (1992) Towards a modular analysis of coloured Petri nets. In K. Jensen (ed.), *Application and Theory of Petri Nets 1992 (Lect. Notes Comput. Sci., 616)*, pp. 113–133. Springer, New York. Also available as: Daimi PB-391, ISSN 0105-8517.

[8] Christensen, S. and Petrucci, L. (1995) Modular state space analysis of coloured Petri nets. In G. de Michelis and M. Diaz (eds), *Application and Theory of Petri Nets 1995 (Lect. Notes Comput. Sci., vol. 935)*, pp. 201–217. Springer, New York.

[9] Jensen, K. (1994) *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 2, Analysis Methods. Monographs in Theoretical Computer Science*. Springer.

[10] Narahari, Y. and Viswanadham, N. (1985) On the invariants of coloured Petri nets. In G. Goos and J. Hartmanis (eds), *Advances in Petri Nets 1985 (Lect. Notes Comput. Sci., vol. 222)*, pp. 330–341. Springer, New York.

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

[12] Huber, P. and Pinci, V. O. (1991) A formal, executable specification of the ISDN basic rate interface. *Proc 12th Int. Conf. on Application and Theory of Petri Nets*, pp. 1–21.

[13] Design/CPN 3.0. META Software and Aarhus University, (1996) Also available as:
http://www.daimi.au.dk/designCPN.

[14] Notomi, M. and Murata, T. (1994) Hierarchical reachability graph of bounded Petri nets for concurrent software analysis. *IEEE Trans. Software Eng.*, **20**, 325–336.

[15] Dimitrovici, C., Hummert, U. and Petrucci, L. (1991) Semantics, composition and net properties of algebraic high-level nets. In G. Rozenberg (ed.), *Advances in Petri Nets 1991 (Lect. Notes Comput. Sci., vol. 524)*, pp. 93–117. Springer, New York.

[16] Finkel, A. and Petrucci, L. (1991) Avoiding state explosion by composition of minimal covering graphs. *Proc. 3rd Computer-Aided Verification Workshop (Lect. Notes Comput. Sci., vol. 575)*, pp. 169–180. Springer, New York.

[17] Valmari, A. (1990) Compositional state space generation. *Proc. 11th Int. Conf. on Application and Theory of Petri Nets*, pp. 43–62.

[18] Valmari, A. (1990) Stubborn sets for reduced state space generation. In Rozenberg, G. (ed.), *Advances in Petri Nets 1990 (Lect. Notes Comput. Sci., vol. 483)*, pp. 491–515. Springer, New York.

[19] Vernadat, F. and Michel F. (1997) Covering step graph preserving failure semantics. In P. Azéma and G. Balbo (eds), *Application and Theory of Petri Nets 1997 (Lect. Notes Comput. Sci., vol. 1248)*, pp. 253–270. Springer, New York.