State Space Analysis of Hierarchical Coloured Petri Nets

Søren Christensen and Lars Michael Kristensen

Computer Science Department, University of Aarhus
Ny Munkegade, Bldg. 540
DK–8000 Aarhus C, Denmark
E-mail: {schristensen, kris}@daimi.aau.dk

Abstract. In this paper, we consider state space analysis of Hierarchical Coloured Petri Nets. It is well-known that almost all dynamic properties of the considered system can be verified when the state space is finite. However, state space analysis is more than just formulating a set of formal requirements and invoking a corresponding set of queries. State space analysis is also applicable during the design and debugging of a system. An approach towards this is to allow the user to analyse the behaviour of systems by drawing and generating selected parts of the state space.

The contribution of this paper is to present a tool in which formal verification, partial state spaces, and analysis by means of graphical feedback and simulation are integrated entities. The focus of the paper is twofold: the support for graphical feedback and the way it has been integrated with simulation, and the underlying algorithms and data-structures supporting computation and storage of state spaces exploiting the hierarchical structure of the models.

Keywords: State space based approaches, efficient model checking, tools, Coloured Petri Nets, validation and simulation.

1 Introduction

State space analysis is one of the main formal analysis methods of Petri Nets [Mur89], and has proven successful in the verification of concurrent systems like communication protocols, parallel- and distributed algorithms. In this paper, we consider state space analysis of Hierarchical Coloured Petri Nets (CP-nets or CPN) [Jen92]. CP-nets allow large models to be structured as a number of modules with well-defined relations between them. They provide the modeller with a mechanism for structuring and abstraction when constructing CPN models of large systems.

The *state space* of a CP-net is a directed graph with a node for each reachable state and an arc for each possible state change. If the state space is finite, it can be used to verify an abundance of properties about the CP-net, e.g., reachability, boundedness, liveness etc. One of the main drawbacks of state space analysis is the state explosion problem, which, in particular, imposes limitations on the applicability of state spaces for large models. One of the contributions of this paper is to present a data-structure which exploits the locality of CP-nets and the structuring mechanism offered by hierarchical CP-nets to obtain a compact representation of the state space and reduce the limitations imposed by the state explosion problem.

Another aspect of this paper is to show how state space analysis can be integrated with graphical feedback and simulation of CPN models. Simulation has proven to be a successful method for validation of systems. This paper presents an approach, which reconciles state space analysis and simulation, and which allows the user to switch between formal verification by means of state spaces, and validation by means of simulation.

CP-nets are supported by the computer tool Design/CPN [CJK97,Jen], which is a graphical-oriented tool supporting construction and simulation of CPN models. The state space tool presented in this paper is based on the theory presented in [Jen94] and is an integrated part of the Design/CPN environment. The state space tool has been applied in a number of projects documented in the literature, e.g, in the areas of communication protocols [CJ97,FBD96], and embedded systems [RS96].

This paper is organised as follows: Section 2 gives an informal introduction to CP-nets and introduces the example which is used throughout the paper. Section 3 presents the query languages, the support for drawing and generation of state spaces and explains how this has been integrated with simulation. The algorithms and data-structures used for computation and storage of state spaces are considered in Sect. 4. Finally, Sect. 5 contains the conclusions and a discussion of related work. The reader is assumed to be familiar with non-hierarchical CP-nets.

2 Hierarchical Coloured Petri Nets

This section informally introduces the formalism of CP-nets by building a CPN model of a packet-switch network. This model is used as an example throughout this paper. The example is based on Sect. 5 in [Tel94]. In order to make the introduction to CP-nets brief, we only introduce the concepts which are used in the subsequent sections. For a complete description see [Jen92].

Consider a number of packets travelling through a reliable packet-switched communication network. A packet is stored at each node in the network before it is forwarded to the next node on its path to the destination. Each node in the network allocates a finite amount of buffer space for this purpose. Since the amount of buffer space is finite, situations may occur in which a group of packets can never reach their destination. Such situations are called *store-and-forward deadlocks*. As an example, consider a network consisting of two internal *nodes* connected to each other and two endpoints called *sites*. Suppose *Site1* wants to send three packets to *Site2* and vice versa, and the buffer size allocated in the two internal nodes is three. Suppose further, that *Site1* sends its three packets to *Node1*, and at the same time *Site2* sends its three packets to *Node2*. Since *Node1* cannot pass any packet to *Node2* whose buffer is full and vice versa, we have a store-and-forward deadlock. It is the responsibility of *controllers* located at the nodes to avoid such deadlocks by putting restrictions on when a packet can be forwarded.

Figure 1 depicts the *Network* module, which is the most abstract CP-net description of the packet-switch network.

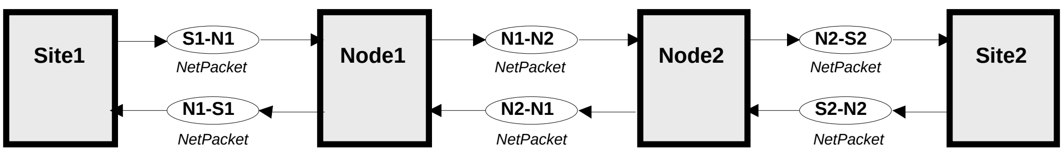

Fig. 1. Network module of the packet-switch model

The communication links of the network are modelled by the six ellipse-shaped places. The place *S1-N1* models the communication link from *Site1* to *Node1* and the place *N1-S1* models the communication link from *Node1* to *Site1*. The remaining communication links are modelled in a similar way by the other places named accordingly. All the six places in Fig. 1 have the type (colour set) *NetPacket* which describes *network packets*. A network packet is either a *data packet* or an *acknowledgement*. An acknowledgement is sent back to a sender to indicate whether the transmitted data packet was accepted or not. If the transmitted data packet is accepted, the sender can delete the data packet from the buffer.

The actions of the *sites* and the *nodes* in the system are described by means of the box-shaped transitions. *Site1* and *Site2* describe the behaviour of the two sites and, similarly, the two transitions *Node1* and *Node2* describe the behaviour of the two internal nodes in the network. The transitions in Fig. 1 are *substitution transitions*, which means that their behaviour is described in more detail in a *sub-module*. The sub-module corresponding to substitution transition *Site1* is shown in Fig. 2

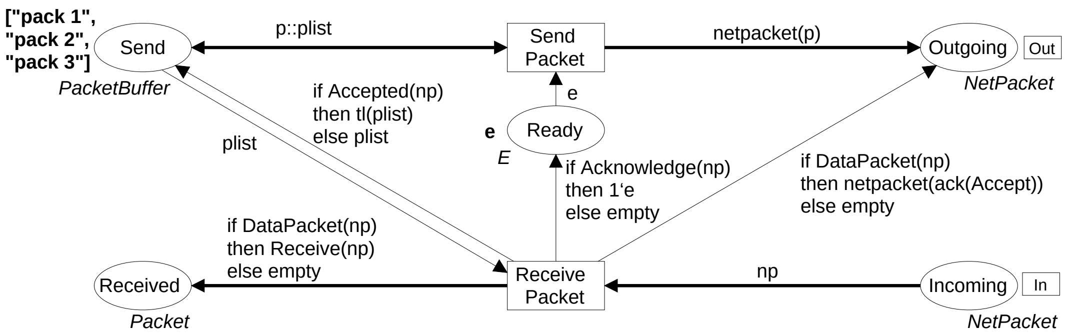

Fig. 2. Site module for the packet-switch model

The state of a site is modelled by the three places *Send*, *Received*, and *Ready*. *Send* contains the data packets to be sent and *Received* contains the data packets received so far. The place *Ready* is used to model that only one data packet is sent at a time from a site. The actions of a site are represented by means of the two transitions *Send Packet* and *Receive Packet*. *Send Packet* models the sending of packets and *Receive Packet* models the receipt of packets.

The sub-module in Fig. 2 is also the sub-module of the substitution transition *Site2*. The places *Incoming* and *Outgoing* are used as *interface* places to the *Network* module in Fig. 1. For the substitution transition *Site1*, the places *Incoming* and *Outgoing* are bound to the places *S1-N1* and *N1-S2*, respectively. Similarly, for *S2-N2* and *N2-S2* and *Site2*. This means that two different *instances* of the sub-module *Site* exist. In a similar fashion as described above, sub-modules exist describing the behaviour of *Node1* and *Node2*. Each *Node* module is decomposed into two *Controller* modules describing the controllers. These modules have a complexity similar to the *Site* module in Fig. 2.

3 State Space Analysis

In this section, we analyse the CPN model of the packet-switch network presented in the previous section. We aim at illustrating the support for state space analysis provided by the tool and the steps involved in conducting the analysis, rather than giving an exhaustive analysis of the packet-switch model.

3.1 State Space Generation

The first step towards state space analysis of a CP-net is the generation of the state space. This step is fully automatic. The generation statistics for the packet-switch model are shown in Table 1. The table shows the size of the state space (number of nodes and arcs) and the time used to generate the state space for different numbers of data packets present on the place *Send* of the *Site* module (see Fig. 2) in the initial state of the CP-net. In all six configurations of the model which was considered, the size of the buffers of the two internal nodes in the network was three. The state space generation was conducted on a Sun Sparc Workstation.

Packets	Nodes	Arcs	CPU Secs
1	516	1,672	2
2	7,144	29,164	38
3	27,100	119,754	192

Packets	Nodes	Arcs	CPU Secs
4	59,748	270,530	506
5	104,716	479,866	1,102
6	162,004	747,762	1,996

Table 1. State space generation statistics.

For large models the state space is often so huge that it cannot be fully generated. Thus, the user is forced to focus on certain aspects of the model by generating only a subgraph of the state space. For this purpose, the tool provides *stop options* and *branching options*. Stop options are used to terminate generation, e.g., when a certain number of nodes have been generated. Branching options allow the user to specify that, for certain states, only a subset of the successors should be generated.

It is also possible to generate parts of a state space interactively. In this case, the user specifies a state, and the tool then calculates all direct successors. This is typically used in connection with drawing parts of the state space; we will return to this topic in Sect. 3.4. Interactive generation of the state space has many similarities with single step debugging as known from conventional programming environments.

3.2 Query Languages

The aim of generating a state space is to analyse the overall behaviour of the model, as well as to verify whether the considered model has certain specific properties or not.

Some *standard queries* are relevant for many models, e.g., to give generation statistics (number of nodes and arcs), bounds of places, the list of dead states (states with no enabled transitions), and information on liveness of transitions. The tool saves the results of the standard queries in a text report. Negative answers to standard queries are constructive, i.e., they help the user investigate why an expected property does not hold. E.g., if an unexpected dead state is found, a shortest path from the initial state to the dead state is helpful information.

Other queries depend on the model being investigated. A general query language implemented in Standard ML [MHT90] is provided. It provides the user with primitives for traversing the state space in different ways and thereby writing non-standard queries. On top of this general query language, a variant of CTL [CCM96] is provided. In this variant of CTL, it is possible to formulate queries about states as well as state changes.

The most interesting property we would like to verify for the packet-switch model is the absence of store-and-forward deadlocks. The packet-switch system has this property if and only if, from any reachable state, it is always possible to reach the state in which: all data packets have been sent from *Site1* to *Site2* and vice versa, and the network as well as the two buffers in the internal nodes are empty. In CP-net terminology, this property can be formulated as the state described above being a home state. Furthermore, we would expect this state to be a dead state, i.e., a state in which the system has terminated.

By invoking one of the standard queries of the tool, we can list the set of dead states of the packet-switch model. In all six configurations of the packet-switch model considered in Table 1, there was one dead state. Using a second standard query, it was verified that this dead state was also a home state.

3.3 Integration with Simulation

During a modelling and design process, the user often switches between state space analysis and simulation. To support this, the state space tool is tightly integrated with the simulator, making it possible to transfer states between the simulator and the state space tool.

When a state is transferred from the state space into the simulator, the new state of the simulator is displayed graphically (as usual) on the CPN diagram. In this way, it is possible to get a detailed view of the transferred state on the CPN diagram. Moreover, the user may start a simulation from the transferred state.

In all six configurations of the packet-switch model, the state obtained from the queries outlined in the previous section was the expected state. This was checked by transferring the state into the simulator and viewing it on the CPN model.

Transferring the simulator state into the state space is supported as well. A typical use of this is to investigate all possible states reachable within a few steps from the current simulator state. In this case, the user transfers the simulator state into the state space tool and all successor markings can be found and drawn as explained below.

3.4 Support for Drawing

Since state spaces often become large, it rarely makes sense to draw them in full. However, the result of queries will often be a set of nodes and/or arcs possessing certain interesting properties, e.g., a path in the state space leading from one state to another. A good and quick way in which to get detailed information on a small number of nodes and arcs is to draw the necessary fragment of the state space. As an example, we will analyse what happens if the size of the buffers in the two internal nodes of the network is reduced from three to two. The result is a strongly connected state space consisting of 9 nodes and 18 arcs. Using the state space tool, this state space has been drawn in Fig. 3. Node 1 represents the initial state of the CP-net. The figure is explained in more detail below.

To obtain detailed information about the drawn fraction of the state space, it is possible to attach *descriptors* to the nodes and arcs. For the nodes, the descriptors typically show the marking of certain places. For the arcs, the descriptors typically show the occurring transition and the binding of some of its variables. The descriptors have sensible defaults but the user may customise the descriptors by writing a script which specifies the contents and layout of the descriptor. The descriptors thus offer an abstraction mechanism by which the user can define a view on the state space. In Fig. 3, selected descriptors of the arcs are shown, illustrating the actions in the packet-switch model. The descriptor is positioned next to the corresponding arc. For the packet-switch model, the descriptors have been customised to give information about the occurring transition and the network packet manipulated by the transition, if any. By inspection, it can be seen that no data packets sent from *Site1* and *Site2* to *Node1* and

Node2, respectively, will be accepted by the *Controller* residing at that node. Hence, we have a *store-and-forward deadlock* if the size of the internal buffers is decreased to two, since the set of data packets can never reach their destination.

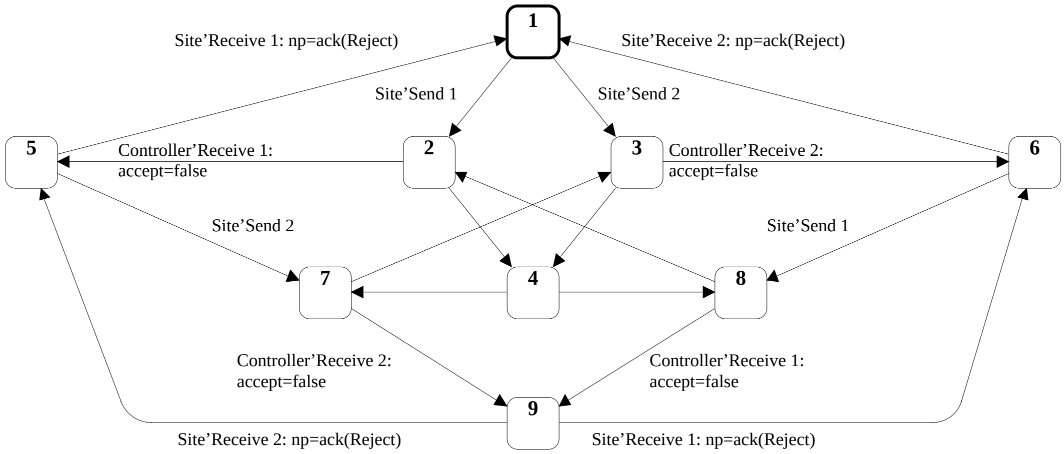

Fig. 3. Drawing of the state space (3 data packets and buffer size 2).

The state space can be drawn either in small steps e.g., node by node or arc by arc, or in fragments using results from, e.g., queries as input to a number of built-in drawing functions. As an example of this, we consider the configuration of the packet-switch model in which a single packet is to be sent from site 1 to site 2 and vice versa, and the size of the internal buffers in the two nodes are 3. We have already seen in Sect. 3.1 that this state space has 516 nodes and 1,672 arcs. If we are interested in detailed information on a shortest path in the state space, leading from the initial state to the dead state, we can get this by using the input from one of the query functions in the tool as input to one of the built-in drawing functions, which is able to display a path in the state space graphically. The result after manual adjustment can be seen in Fig. 4.

Node 1 represents the initial state, and node 516 represent the dead state. From the descriptor of Node 1, it can be seen that, initially, the two sites have received no packets and both buffers in the internal nodes of the network are empty. The descriptor of the arc leading from node 1 to node 2 shows that first site 1 sends a packet with contents “pack 1”. This results in the state represented by node 2. Next site 2 sends a packet (also with contents “pack 1”) leading to node 4. In the next two steps, the controllers residing at node 1 and node 2, respectively, decide to accept the packets leading to the state represented by node 15. The descriptor of node 15 shows that node 1 and node 2 have stored the packet in their internal buffers. Continuing this way, it is possible to trace the communication between the sites and the internal nodes, and we eventually reach state 516 in 18 steps.

4 Algorithms and Data-structures

The fundamental issues in the implementation of the state space tool are: the computation of the state space, the representation of the state space, and, finally, the algorithms supporting the verification of CP-nets, i.e., how dynamic properties like dead states, liveness etc. of a CP-net can be verified using the state space. Below, we focus on the computation and

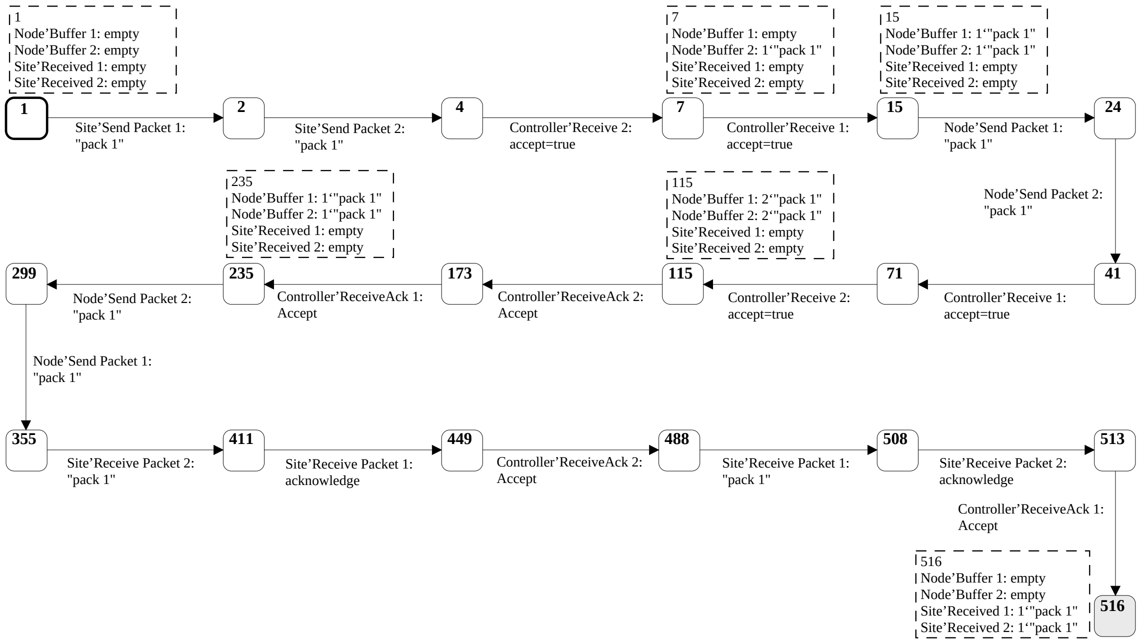

Fig. 4. Execution path leading to the dead state (1 data packet and buffer size 3).

representation of state spaces. The reader interested in the algorithms used to determine dynamic properties of CP-nets by means of state spaces is encouraged to consult [Jen94] and [CCM96].

4.1 Computation of State Spaces

The computation of the state space of a CP-net is straightforward, using a standard graph traversal algorithm. In each state encountered, starting from the initial state of the CP-net, the set of enabled binding elements (pairs consisting of a transition and a binding of its variables) is computed and from this set, the set of immediate successor states. One proceeds in this way, until all encountered states have been processed. The three key issues in this algorithm are: given a state how to calculate the enabled binding elements in this state, given an enabled binding element how to calculate the successor state, and given a state how to determine whether this state is already included in the state space.

The capability of computing the set of enabled binding elements in a given state is already part of the simulator. Moreover, given a state and an enabled binding element, the simulator is able to calculate the successor state resulting from an occurrence of this binding element. Hence, the core in the implementation is the bidirectional mapping between state representation in the simulator and in the state space tool, and determining whether two states are equal or not. We will return to the latter issue in the following section.

4.2 State Space Storage

We now consider how the state space, i.e., the nodes and arcs constituting the state space, is stored. At first, we consider the states/nodes. Because of the well-known state explosion problem, it is essential to provide a succinct representation of the states. At the same time, in order to make the computation of the state space time efficient, it must also be fast to determine whether a given state/node is already included in the state space.

To obtain a succinct representation of the states, the locality of Petri Nets is exploited. A large hierarchical CP-net often consists of several modules. The occurrence of a transition in a module changes only the marking of the immediate surrounding places. Therefore a large fraction of the modules will be left unchanged by the occurrence of a transition.

The basic idea in the representation of states is to avoid an unnecessary duplication of complex values. As an example of this, consider the packet-switch model. When the transition *Send Packet* (see Fig. 2) in the instance of the *Site* module corresponding to *Site1* occurs, only the marking of the places *Outgoing* and *Ready* in this module is affected. Because *Outgoing* is an interface place to the *Network* module (see Fig. 1), the marking of *S1-N1* also changes. Similarly, the marking of the interface place of *S1-N1* in the module *Node1* changes. The markings of all other places of the CP-net remain unchanged, and more important, the state of the instance of the *Node* and *Site* module corresponding to *Node2* and *Site2* is left unchanged.

Figure 5 depicts how states are stored, exemplified by the two modules of the packet-switch model shown in Fig. 1 and Fig. 2. The figure is explained below.

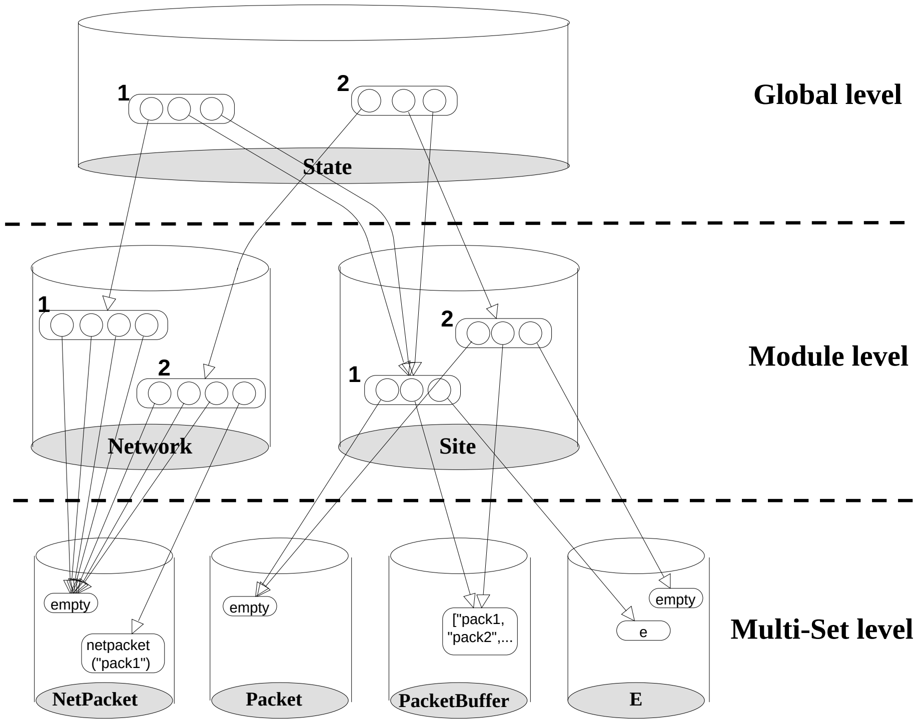

Fig. 5. Representation of states

States are stored on three levels: The *Global level*, the *Module level* and the *Multi-set level*. The Multi-set level is concerned with storage of multi-sets, i.e., the marking of the individual places. There is one multi-set storage for each type (colour set) in the CP-net.

The Module level is concerned with the storage of the state of a single module. The state of a module equals the marking of each of the places in the module. A *module state* (MS) has an entry for each place in the module, which is a pointer into the multi-set storage, corresponding to the type of the place. In this way, multi-sets can be *shared* among the MSs. On the Module level, there is a storage for each module in the CP-net.

The Global level is concerned with the storage of states of the entire CP-net. A *global state* (GS) has an entry for each module in the CP-net, which is a pointer into the corresponding storage at the Module level. In this way, the MSs can be shared by states of the CP-net. Figure 5 depicts the storage of two states, corresponding to the initial state of the CP-net (GS 1), and the state (GS 2) resulting from an occurrence in the initial state of the transition *Send Packet* in the instance of the *Site* module corresponding to *Site1*. It can be seen that the two GSs share an MS in the site storage (MS 1). This corresponds to the fact that an occurrence *Send Packet* does not affect the instance of the *Site* module corresponding to *Site2*; as explained above. Because the initial state of both instances of the *Site* module is the same, the two corresponding entries in GS 1 refer to MS 1. In a similar way, it can be seen that multi-sets are shared by the MSs.

The representation above avoids unnecessary duplication of complex values and makes it efficient to determine, during the generation of the state space, whether a state resulting from the occurrence of a binding element is already included in the state space. First, the marking of those places which are changed by the occurrence of the binding element is inserted into the Multi-set level. Using the pointers thus obtained, MSs for those modules, whose state is changed by the occurrence of the binding element are created and inserted into the Module level. In this way, pointers into the Module level are obtained for those modules whose state is changed. Pointers into the Module level for the modules whose marking is unchanged are the same as the ones for the marking in which the binding element occurs. Now, given the pointers into the Module level, the marking is already included in the state space if and only if, a GS exists having the same pointers into the Module level and comparing pointers is an efficient operation. To make insertion in the storages efficient, all storages are organised as search trees.

We now study the effect of using the above sharing of MSs and multi-sets. First, we consider the sharing on the module level for two different configurations of the packet-switch model: three and six packets. For each of the four modules in the CP-net, we compare the number of MSs which should have been stored if no sharing of MSs was made with the number of MSs actually stored. The figures can be found in Table 2. The column *Exp* depicts the number of MSs which should have been used if no sharing was made. The column *Act* depicts the actual number of MSs when sharing is used. The column *Ratio* depicts the actual number divided by the expected number of MSs.

Module	Exp	Act	Ratio
Network	27,100	5,609	2.1×10^{-1}
Site	54,200	28	5.2×10^{-4}
Node	54,200	3	5.5×10^{-5}
Controller	108,400	17	1.6×10^{-4}
Total	243,900	5657	2.3×10^{-2}

Module	Exp	Act	Ratio
Network	162,004	25,682	1.6×10^{-1}
Site	324,008	91	2.8×10^{-4}
Node	324,008	3	9.3×10^{-6}
Controller	648,016	35	5.4×10^{-5}
Total	1,458,036	25,811	1.8×10^{-2}

Table 2. Statistics for the Module level - 3 packets (left), 6 packets (right)

The *Total* row shows, that the saving in the number of MSs is significant. E.g., from the *Total* row it can be seen that the number of MSs stored is reduced to 2.3% for three packets and 1.8% for 6 packets.

Let us now consider the sharing of multi-sets. For each type (colour set) in the CP-net, the number of multi-sets which had to be stored if no sharing was made, is compared to the number of multi-sets which is stored, when sharing is used. The figures are shown in Table. 3.

Type	Exp	Act	Ratio
NetPacket	33,654	27	8.0×10^{-4}
Packet	28	4	1.4×10^{-1}
PacketBuffer	45	11	2.4×10^{-1}
E	45	2	4.4×10^{-2}
Capacity	3	3	1.0
Total	33,775	47	1.4×10^{-3}

Type	Exp	Act	Ratio
NetPacket	154,092	51	3.3×10^{-4}
Packet	91	7	7.7×10^{-2}
PacketBuffer	126	23	1.8×10^{-1}
E	126	2	1.6×10^{-2}
Capacity	3	3	1.0
Total	154,428	86	5.6×10^{-4}

Table 3. Statistics for the Multi-set level - 3 packets (left), 6 packets (right)

Again, significant savings are obtained. E.g., for 3 packets the number of multi-sets stored was reduced to 0.14% for 3 packets, and 0.056% for 6 packets.

For the representation of the arcs/binding elements, a single storage is used. For the same reasons as with states, pointers are also used to avoid the duplication of complex values.

5 Conclusion

In this paper, we have presented a state space tool supporting formal verification, generation of partial state spaces, and analysis of CP-nets based on the capability of drawing selected parts of the state space. We also considered the way in which state space analysis is integrated with simulation. A data-structure for a compact representation of state spaces of hierarchical CP-nets was presented and statistics given, demonstrating the applicability of the data-structure.

The recognition of graphical representations of the state space as a means to analyse the behaviour of systems has also been considered in a process algebraic setting in [VKS96]. The approach presented in this paper and in [VKS96] is, however, different. In [VKS96], the view on the system is specified at the syntactical level and the state space is generated and reduced according to this view. In contrast, we generate the state space once, and then define different views on the system at the semantic (state space) level. To our knowledge the integration between simulation, graphical representation and state space analysis has not been considered by others.

To some extend, the idea of avoiding duplication of complex values resembles BDDs as e.g. used in the SMV system [McM93]. BDDs have indeed proven to be a very compact way in which to represent state spaces and significant results have been obtained; in particular when the system under consideration exhibits some form of regularity. The data-structure presented in this paper, however, avoids some of the potential drawbacks of BDDs. First of all, it does not assume any regularity in the system and is independent of any kind of variable ordering. Furthermore, the memory requirements are more predictable than with BDDs, for which there are no general correlation between the size of the state space and the size of the BDD. In particular, the size of the intermediate BDDs during generation of the state space is often a problem. The suggested data-structure does not suffer from such "spurious" behaviour.

The use of BDDs in the context of Petri Nets has been considered in [PRCB96]. The results from [PRCB96], which considers k-bounded Petri Nets are not directly applicable to CP-nets, since they rely on encoding/representing the state of a Petri Net as a vector of bits.

Due to the elaborated notion of data types and hence states provided by CP-nets, this is not a priori possible.

The idea of generating partial state spaces has many similarities with on-the-fly model checking as implemented in the SPIN tool [Hol90] for verification of asynchronous process systems. The data-structure used to represent the state space in the SPIN tool has some similarities with the data-structure presented in this paper.

Even taking the above considerations into account, the size of the state spaces remains a problem. Several methods for construction of smaller state spaces, preserving many properties, have been proposed in the literature. One approach is based on the observation that equivalence and symmetry [Jen94,ES96,ID96,CFJ93] are present in many distributed systems and can be exploited to obtain a reduced state space. A novel prototype integrated in Design/CPN supporting this approach now exists [JK96].

An orthogonal approach is based on the observation that one of the main reasons for state explosion is caused by representing all possible interleavings of independent actions. Partial order reduction techniques [WG93,Val90] are an attempt to avoid representing all such interleavings. Both of the above techniques could be integrated with the ideas of graphical feedback, and the same representation of the state space and the idea of utilising the simulator in the computation of the state space would still work. It is worth observing that the data-structure for representation of the state space presented in this paper already to a certain extent takes partial order reduction into account, since a given state of a module is only stored once. State explosion is rarely caused by the individual modules having many different states, but more commonly caused by the cartesian product of the "small" number of states for the individual modules. Hence, by also applying partial order reduction, the main reduction in memory usage, would appear at the Global level and in the memory used to store arcs.

An obstacle which still remains to be solved, before partial order reduction can be applied is how e.g., stubborn sets can be computed for CP-nets without unfolding to the underlying Place/Transition net. Unfolding is an approach which is inapplicable in practice.

Another approach to obtain a reduced state space is based on compositionality [Val96] and modularity [CP96]. Systems like for instance the packet-switch model considered in this paper are often specified in a number of modules. The state space of the individual modules is often of a size which can easily be handled. By introducing suitable parallel composition operators and introducing black-box semantics, compositionality in the model of the system can be exploited with respect to verification.

Acknowledgements. We thank Jens B. Jørgensen for proof-reading and comments on this paper. The work has been supported by grants from the University of Aarhus Research Foundation and the Danish National Science Research Council.

References

[BR96] J. Billington and W. Reisig, editors. *Proceedings of the 17th International Conference on Application and Theory of Petri Nets, Osaka, Japan*, volume 1091 of *Lecture Notes in Computer Science*. Springer-Verlag, 1996.

[CCM96] A. Cheng, S. Christensen, and K.H. Mortensen. Model Checking Coloured Petri Nets Exploiting Strongly Connected Components. In M.P. Spathopoulos, R. Smedinga, and P. Kozák, editors, *Proceedings of the International Workshop on Discrete Event Systems, WODES96*. Institution of Electrical Engineers, Computing and Control Division, Edinburgh, UK, 1996.

[CFJ93] E.M. Clarke, T. Filkorn, and S. Jha. Exploiting Symmetries in Temporal Model Logic Model Checking. In C. Courcoubetis, editor, *Proceedings of the 5th International Conference on Computer Aided Verification, Elounda, Greece*, pages 450–462. Springer-Verlag, 1993.

[CJ97] S. Christensen and J.B. Jørgensen. Analysing Bang & Olufsen's BeoLink® Audio/Video System Using Coloured Petri Nets. In P. Azema and G. Balbo, editors, *Proceedings of the 18th International Conference on Application and Theory of Petri Nets, Toulouse, France*, volume 1248 of *Lecture Notes in Computer Science*. Springer-Verlag, 1997.

[CJK97] S. Christensen, J.B. Jørgensen, and L.M. Kristensen. Design/CPN — A Computer Tool for Coloured Petri Nets. In E. Brinksma, editor, *Proceedings of the Third International Workshop on Tools and Algorithms for the Construction and Analysis of Systems, Twente, The Netherlands*, volume 1217 of *Lecture Notes in Computer Science*, pages 209–223. Springer-Verlag, 1997. Also available as DAIMI PB-511, ISSN 0105-8517, February 1997.

[CP96] S. Christensen and L. Petrucci. Modular State Space Analysis of Coloured Petri Nets. In G. De Michelis and M. Diaz, editors, *Proceedings of the 16th International Conference on Application and Theory of Petri Nets, Torino, Italy*, volume 935 of *Lecture Notes in Computer Science*. Springer-Verlag, 1996.

[ES96] E. Allan Emerson and A. Prasad Sistla. Symmetry and model checking. *Formal Methods in System Design*, 9, 1996.

[FBD96] D.J. Floreani, J. Billington, and A. Dadej. Designing and Verifying a Communications Gateway Using Colored Petri Nets and Design/CPN. In Billington and Reisig [BR96].

[Hol90] G.J. Holzmann. *Design and Validation of Computer Protocols*. Prentice-Hall, 1990.

[ID96] C.N. Ip and D.L. Dill. Better Verification Through Symmetry. *Formal Methods in System Design*, 9, 1996.

[Jen] K. Jensen. Design/CPN Online, Computer Science Department, University of Aarhus, Denmark. Online: `http://www.daimi.aau.dk/designCPN/`.

[Jen92] K. Jensen. *Coloured Petri Nets — Basic Concepts, Analysis Methods and Practical Use. Vol. 1, Basic Concepts*. EATCS Monographs on Theoretical Computer Science. Springer-Verlag, 1992.

[Jen94] K. Jensen. *Coloured Petri Nets — Basic Concepts, Analysis Methods and Practical Use. Vol. 2, Analysis Methods*. EATCS Monographs in Theoretical Computer Science. Springer-Verlag, 1994.

[JK96] J.B. Jørgensen and L.M. Kristensen. *Design/CPN OE/OS Graph Manual*. Computer Science Department, University of Aarhus, Denmark, 1996.
Online: `http://www.daimi.aau.dk/designCPN/`.

[McM93] K.L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.

[MHT90] R. Milner, R. Harper, and M. Tofte. *The Definition of Standard ML*. MIT Press, 1990.

[Mur89] T. Murata. Petri Nets: Properties, Analysis and Application. In *Proceedings of the IEEE, Vol. 77, No. 4*. IEEE Computer Society, 1989.

[PRCB96] E. Pastor, O. Roig, J. Cortadella, and R. Badia. Petri Net Analysis Using Boolean Manipulation. In G. De Michelis and M. Diaz, editors, *Proceedings of the 16th International Conference on Application and Theory of Petri Nets, Torino, Italy*, volume 935 of *Lecture Notes in Computer Science*. Springer-Verlag, 1996.

[RS96] J.L. Rasmussen and M. Singh. Designing a Security Systen by Means of Coloured Petri Nets. In Billington and Reisig [BR96].

[Tel94] Gerard Tel. *Introduction to Distributed Algorithms*. Cambridge University Press, 1994.

[Val90] A. Valmari. Stubborn Sets for Reduced State Space Generation. In G. Rozenberg, editor, *Advances in Petri Nets 90*, volume 483 of *Lecture Notes in Computer Science*. Springer-Verlag, 1990.

[Val96] A. Valmari. Compositionality in State Space Verification Methods. In Billington and Reisig [BR96].

[VKS96] A. Valmari, K. Karsisto, and M. Setala. Visualisation of Reduced Abstracted Behaviour as a Design Tool. In *Proc. Fourth Euromicro Workshop on Parallel and Distributed Processing*. IEEE, 1996.

[WG93] P. Wolper and P. Godefroid. Partial Order Methods for Temporal Verification. In E. Best, editor, *Proc. of 4th international conference on Concurrency Theory (CONCUR)*, volume 715 of *LNCS*. Springer-Verlag, 1993.