State Space Methods for Coloured Petri Nets

Lars Michael Kristensen

Ph.D. Dissertation

Department of Computer Science
University of Aarhus
Denmark

State Space Methods for Coloured Petri Nets

A Dissertation
Presented to the Faculty of Science
of the University of Aarhus
in Partial Fulfillment of the Requirements for the
Ph.D. Degree

by
Lars Michael Kristensen
January 31, 2000

Preface

Summary

An increasing number of system development projects are concerned with distributed and concurrent systems. There are numerous examples, ranging from large scale systems, in the areas of telecommunication and applications based on WWW technology, to medium or small scale systems, in the area of embedded systems. A typical distributed or concurrent system consists of a number of independent but communicating processes. This means that the execution of such systems may proceed in many different ways, e.g., depending on whether messages are lost, the speed of the processes involved, and the time at which input is received from the environment. As a result, distributed and concurrent systems are, by nature, complex and difficult to design and test. This has motivated the development of methods which support computer-aided analysis, validation, and verification of the behaviour of concurrent and distributed systems.

State space methods are some of the most prominent approaches in this field. The basic idea underlying state spaces is (in its simplest form) to compute all reachable states and state changes of the system, and represent these as a directed graph. The virtue of a constructed state space is that it makes it possible to algorithmically reason about the behaviour of a system, e.g., verify that the system possesses certain desired properties or locate errors in the system. The main disadvantage of using state spaces is the state explosion problem: even relatively small descriptions/systems may have an astronomically or even infinite number of reachable states, and it is a serious limitation on the use of state space methods in the analysis of real-life systems. The development of reduction methods to alleviate this inherent complexity problem is, therefore, a central topic in the development of state space methods. Reduction methods avoid representing the entire state space of the system or represent the state space in a compact form. The reduction is done in such a way that properties of the system can still be derived from the reduced state space.

In this thesis we study state space methods in the framework of Coloured Petri Nets which is a graphical language for modelling and analysis of concurrent and distributed systems. The thesis consists of two parts. Part I is the mandatory overview paper which summarises the work which has been done. Part II is composed of five *individual papers* and constitutes the core of this thesis. Four of the these papers have been published elsewhere as conference papers, journal papers, or book chapters. The fifth paper is submitted to an international conference.

The overview paper introduces the research field of state space methods for Coloured Petri Nets and summarises the contents and contributions of the five individual papers. A substantial part of the overview paper has also been devoted to putting the results presented in the five individual papers in a broader perspective in the form of a discussion of related work.

The first paper considers state space analysis of 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 paper presents 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 datastructures which support computation and storage of state spaces and which exploit the hierarchical structure of the models.

The second paper presents a computer tool for verification of distributed systems. The tool implements the method of state spaces with symmetries. The basic idea in the approach is to exploit the symmetries inherent in many distributed systems in order to construct a condensed state space. As an example, the correctness of Lamport's Fast Mutual Exclusion Algorithm is established. We demonstrate a significant increase in the number of states which can be analysed. State spaces with symmetries is not our invention. Our contribution is the development of the tool and verification of the example, demonstrating how the method of state spaces with symmetries can be put into practical use.

The third paper demonstrates the potential of verification based on state spaces reduced by equivalence relations. The basic observation is that quite often some states of a system are similar, i.e., they induce similar behaviours. Similarity can be formally expressed by defining an equivalence relation on the set of states and on the set of actions of a system under consideration. A state space can be constructed in which the nodes correspond to equivalence classes of states, and the arcs correspond to equivalence classes of actions. Such a state space is often much smaller than the ordinary, full state space, but it does allow derivation of many verification results. State spaces with equivalence classes is not our invention. The contribution of the paper is the specification of a concrete notion of equivalence, and a demonstration of its application for verification of a communication protocol. Aided by a developed computer tool significant reductions of state spaces are exhibited, representing some first results on the practical use of state spaces with equivalence classes for Coloured Petri Nets. Exploiting the symmetries in systems induce a certain kind of equivalence. The verification of the communication protocol demonstrates the potential provided by more general notions of equivalence.

The fourth paper addresses the issue of using the stubborn set method for Coloured Petri Nets without relying on unfolding to the equivalent Place/Transition Net. The stubborn set method exploits the independence between actions to avoid representing all possible interleavings of the system execution. We give a lower bound result which states that there exist Coloured Petri Nets for which computing good stubborn sets re-

quires time proportional to the size of the equivalent Place/Transition Net. We suggest an approximative method for computing stubborn sets of so-called process-partitioned Coloured Petri Nets which does not rely on unfolding. The underlying idea is to add some structure to the Coloured Petri Net, which can be exploited during the stubborn set construction to avoid the unfolding. The practical applicability of the method is demonstrated with both theoretical and experimental case studies, in which reduction of the state space, as well as savings in time, are obtained.

The fifth paper presents two new question-guided stubborn set methods for state properties. The first method makes it possible to determine whether a state is reachable in which a given state property holds. It generalises earlier results on stubborn sets for state properties in the sense that the earlier methods can be seen as an implementation of our more general method. We also propose alternative, more powerful implementations that have the potential of leading to better reduction results. This potential is demonstrated on some practical case studies. As an extension of the first method, we present a novel method which can be used to determine if it is always possible to reach a state where a given state property holds. Compared to earlier methods the benefit is again in the potential for better reduction results.

Acknowledgements

Time has passed away quickly since I started my studies in computer science. There are many who deserve to be thanked for their part in making my period of study a pleasant time. Here I can only mention a few of them.

First and foremost I would like to thank Kurt Jensen and Søren Christensen for supervision, cooperation, and inspiration throughout my Ph.D. studies. Thanks to them for hiring me as a student programmer prior to my Ph.D. studies, and for later encouraging me to enrol into the Ph.D. programme. Their supervision has been a well-balanced combination of guidance, support, and freedom. Also, they gave me the opportunity to participate in many interesting conferences, workshops, tutorials, and meetings.

During the first part of my Ph.D. studies I enjoyed sharing an office with Jens Bæk Jørgensen. Thanks to him for fruitful cooperation and co-authoring of two of the papers which are part of this thesis. From him I also learned a number of working practices and skills from which I have since benefitted.

Thanks to Antti Valmari for hosting an inspiring and rewarding six month visit to Tampere University of Technology, Finland in the spring of 1997, and for conveying to me significant insight into the research field of state space methods. The visit to Tampere led to continuous cooperation and the co-authoring of two of the papers which are part of this thesis.

Throughout my Ph.D studies I was part of the Coloured Petri Nets Group (CPN group) at the Department of Computer Science (DAIMI). Being part of a group of people working on the same topics and sharing the same research interests has been of great importance professionally as well as socially. From the CPN group I would especially like to thank Kjeld H. Mortensen, Lisa M. Wells, Bo Lindstrøm, Louise Lorentsen, Thomas Mailund, and Jorge C. Abrantes de Figueiredo for cooperation in various projects.

Thanks also to Christian Nørgaard Storm Pedersen and Rune Bang Lyngsø for good friendship and company during our years of study as well as for cooperation in project work during the first four years at DAIMI.

The work done for this thesis has been supported by grants from the University of Aarhus Research Foundation and the Danish National Science Research Council (SNF).

Lars Michael Kristensen
Århus, January 31, 2000.

Contents

Part I

Overview

Chapter 1

Introduction

This chapter introduces the research field of state space methods for Coloured Petri Nets. Section 1.1 gives an introduction to state space methods. Section 1.2 gives a brief introduction to Coloured Petri Nets, and Sect. 1.3 gives an introduction to state spaces of Coloured Petri Nets. Section 1.4 presents the motivation and aims of this thesis. Section. 1.5 gives an overview of the work done and the structure of this thesis, including an outline for the remainder of this overview paper.

1.1 State Space Methods

An increasing number of system development projects are concerned with distributed and concurrent systems. There are numerous examples, ranging from large scale systems, in the areas of telecommunication and applications based on WWW technology, to medium or small scale systems, in the area of embedded systems. The development of concurrent and distributed systems is complex. A major reason is that the execution of a concurrent system consisting of a number of independent but co-operating processes may proceed in many different ways, e.g., depending on whether messages are lost, the speed of the processes involved, and the time at which input is received from the environment. As a result, distributed and concurrent systems are, by nature, complex, and difficult to design and test, and can be subject to subtle errors that can go undetected for a long time. This has motivated the development of methods which support computer-aided analysis, validation, and verification of the behaviour of concurrent and distributed systems, and which can be used during design to obtain more reliable and trustworthy concurrent systems.

One of the most promising ways to increase reliability, and reduce design errors of such systems is the use of *formal methods* [27], which are mathematically-based languages, techniques, and tools for specifying, analysing, and verifying systems. The application of formal methods is typically based on the construction of models which can be manipulated and analysed by a computer. The model describes/represents the behaviour of the concurrent system under consideration, and is typically a simplified and abstract representation of the real-world system and, as such, includes only those aspects of the real-world system relevant to the problem at hand. The use of formal methods does not automatically guarantee correctness of a system. However, the act of constructing a model and analysing its behaviour is a way of gaining greater confidence in the proposed designs and helps reveal inconsistencies, ambiguities, and incompleteness that might otherwise not be detected. Choosing the abstraction level of a model

is one of the arts in applying formal methods, and it is primarily a matter of intuition based on practical experience.

A central point in most formal methods is to conduct formal analysis and verification based on a constructed model, i.e., to determine whether or not a system under development or analysis satisfy certain formally-stated properties. State space methods are some of the most prominent approaches for conducting formal analysis and verification. The basic idea of *state spaces* is to calculate all reachable states and state changes of the system and represent these as a directed graph. State spaces can be constructed fully automatically. From a constructed state space it is possible to answer a large set of analysis and verification questions concerning the behaviour of the system such as absence of deadlocks, the possibility of always being able to reach a given state, and the guaranteed delivery of a given service. The applicability of state space methods is closely tied to the existence of suitable computer tool support – manual calculation and inspection of the state space for more than trivial systems is time-consuming, error-prone, and impossible for practical purposes.

One of the main advantages of state space methods is that they can provide counter examples, i.e., debugging information as to why an expected property does not hold. Furthermore, they are relatively easy to use, and they have a high degree of automation. The ease of use is primarily due to the fact that with state space methods it is possible to hide a large portion of the underlying complex mathematics from the user. This means that quite often the user is only required to formulate the property which is to be investigated and then start a computer tool.

The main disadvantage of using state spaces is the *state explosion problem* [129]: even relatively small descriptions/systems may have an astronomical or even infinite number of reachable states, and this is a serious problem for the use of state space methods in the analysis of real-life systems. The development of *reduction methods* to alleviate this inherent complexity problem is, therefore, a central topic in the development of state space methods. Reduction methods avoid representing the entire state space of the system or represent the state space in a compact form. The reduction is done in such a way that properties of the system can still be derived from the reduced state space. Most reduction methods take advantage of certain aspects or properties of a system. One example of this is to exploit the symmetries present in many systems. Another example is to take advantage of the asynchrony between processes in order to avoid representing all interleaved executions of the system. However, due to the theoretically provable intractable complexity of verification, there is no single reduction method which works well in all situations. It is therefore important to develop a toolbox of different reduction methods.

Another drawback of state space methods is that they require one to fix the parameters of the system. For example, a system is typically verified for some finite number of processes. This drawback is, however, less severe compared to the state explosion problem. One reason is that errors tend to manifest themselves in even small configurations of the system. Another reason is that if one has verified a system for a number of configurations, then it is likely that the system also works correctly in other configurations. In fact, using clever reduction methods it is sometimes possible to establish results which are valid for all configurations of the system.

The advantages offered by state space methods are so attractive that researchers have pursued the development of state space methods rather than giving up due to

the drawbacks mentioned above. The state space methods of today have indeed been shown to give reasonable performance for systems of practical size [27].

1.2 Coloured Petri Nets

In this thesis we study state space methods in the context of *Coloured Petri Nets* (CP-nets or CPN) [66,67,69,70,85] which provide a framework for construction and analysis of distributed and concurrent systems. A CPN model of a system describes the states which the system may be in and the actions causing the system to change its state. CP-nets is a graphical language based on the ideas of *Petri Nets* [104] and *Predicate/Transition Nets* [44]. The development of CP-nets has been driven by the desire to develop an industrial-strength modelling language – at the same time theoretically well-founded and versatile enough to be used in practice – for systems of the size and complexity found in typical industrial projects. To achieve this, CP-nets combine the strength of Petri Nets with the strength of programming languages. Petri Nets provide the primitives for describing synchronisation of concurrent processes, while a programming language provides the primitives for defining data types and manipulating data values. CP-nets have been supported by the computer tool DESIGN/CPN [16,99] since 1989. The DESIGN/CPN tool uses the functional programming language STANDARD ML [95,119] as *inscription language*, i.e., the programming language for defining data types and manipulating data values.

Petri Nets are traditionally divided into *low-level Petri Nets* and *high-level Petri Nets*. CP-nets belong to the class of high-level Petri Nets which are characterised by the combination of Petri Nets and programming languages. Low-level Petri Nets are primarily suited as a theoretical model for concurrency, although certain classes of low-level Petri Nets are often applied for modelling and verification of hardware systems. High-level Petri Nets are aimed at practical use, in particular because they allow for construction of compact and parameterised models.

Petri Nets (and CP-nets) is just one out of many formal methods for specifying the behaviour of concurrent systems. Other prominent examples are *Statecharts* [56], *Calculus of Communicating Systems* [94], *I/O Automatons* [90], and *Communicating Sequential Processes* [57].

Example. Figure 1.1 shows a simple example of a CP-net modelling a two-phase commit protocol [30]. Such a protocol can, for instance, be used to implement atomicity of distributed transactions. We will not go into the details of this CP-net here. The intention is just to give a flavour of CP-nets. The left-hand side models the loop executed by the *Coordinator* which is responsible for determining whether or not the transaction can be committed. The right-hand side models the loop executed by the *Workers*. The commit protocol should ensure that a transaction is committed if and only if all workers are prepared to execute their part of the transaction. The middle part models the network connecting the coordinator and the workers. The syntactical elements of a CP-net consist of *places* (drawn as ellipses), *transitions* (drawn as rectangles), *arcs* connecting the places and transitions, and *inscriptions* associated with the places, transitions, and arcs. Places are used to model the states of the system. Transitions are used to model the actions of the system. A state (in CP-net terminology called a *marking*) is a distribution of so-called *tokens* carrying data values (colours) on

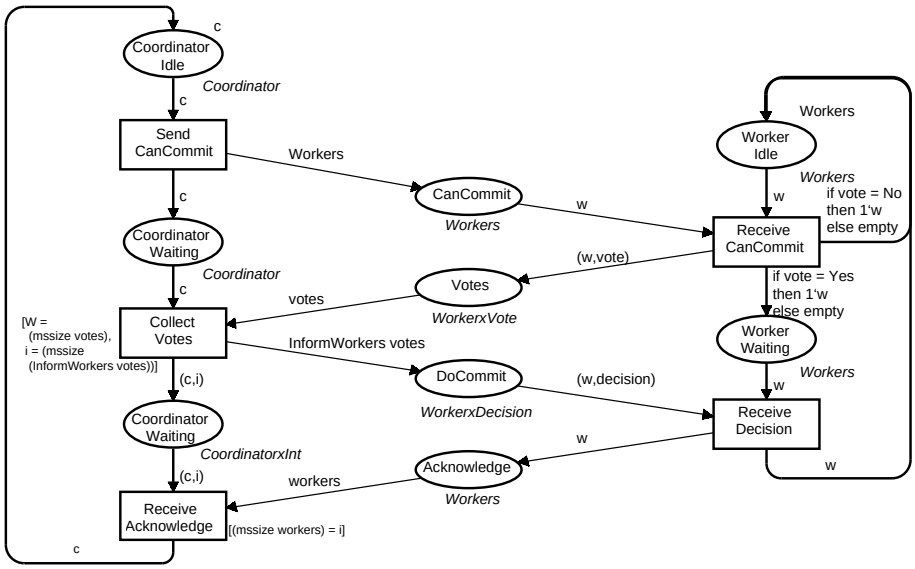

Figure 1.1: CPN model of a simple two-phase commit protocol.

the places of the CP-net. The arcs and arc inscriptions are used to describe the quantity and colours of tokens which have to be present on the input places of a transition in order for the transition to be *enabled*, and to describe the quantity and colours of tokens added to output places of a transition when the transition *occurs*. Initially the coordinator as well as the workers are idle. This is described by the *initial marking* (initial distribution of tokens) of the CP-net which is specified by means of inscriptions associated with the places. The inscriptions of a CP-net are written in STANDARD ML.

1.3 State Spaces of Coloured Petri Nets

The considerable modelling power of CP-nets inherited from Place/Transition Nets [107] and the STANDARD ML language implies that essentially all interesting verification questions concerning CP-nets are undecidable (see, e.g., [37] for a survey). However, many CP-nets arising in practice do have a finite state space making verification possible. In addition to this, it is often possible to prove and disprove properties of a system by relying on a partial state space, i.e., a finite subgraph of the full state space.

Example. Figure 1.2 shows the full state space for the CPN model of the commit protocol with two workers. Each node corresponds to a *reachable marking* of the CP-net, i.e., a marking which can be reached by occurrences of transitions starting from the initial marking. The arcs correspond to occurring *binding elements*. A binding element is a pair consisting of a transition and an assignment of data values to the variables appearing in the inscriptions associated with the transition and on the surrounding arcs of the transition. An arc between two nodes means that the binding element to which the arc corresponds is enabled in the marking represented by the source node, and the occurrence of this binding element in the marking of the source node leads to the

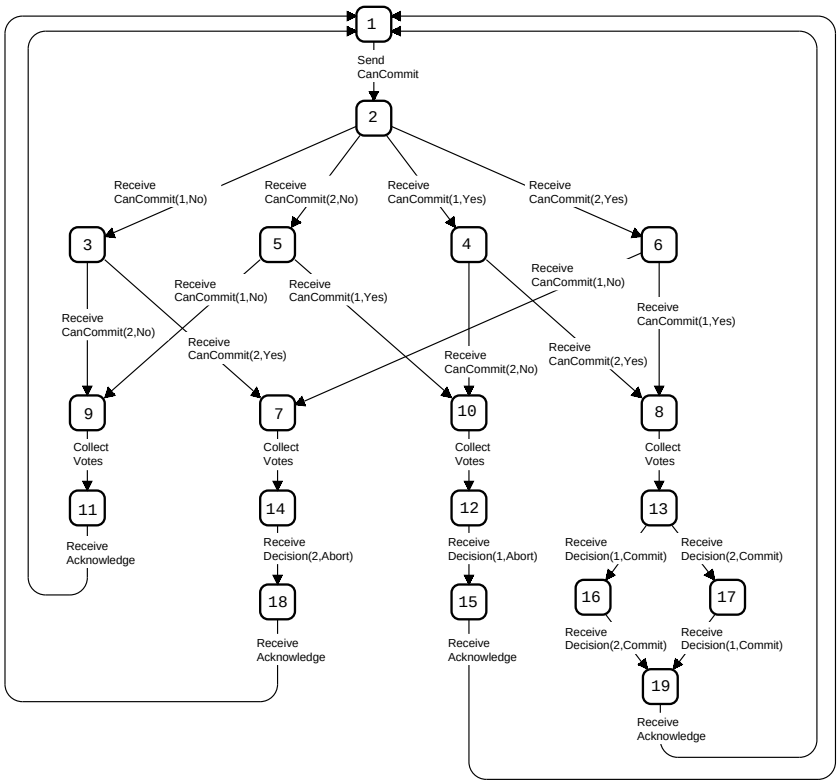

Figure 1.2: Full state space for two-phase commit protocol in Fig. 1.1.

marking represented by the destination node.

Node 1 corresponds to the initial marking of the CPN model. Each arc has an associated label specifying the name of the occurring transition to which it corresponds. For the transitions modelling the actions of the workers the label also specifies in parentheses which worker conducts the corresponding action. In the case of ReceiveCanCommit the label also specifies whether the worker voted Yes or No, and in the case of ReceiveDecision the label also specifies the decision, i.e., whether the worker is to Abort or Commit its part of the transaction. As can be seen from Fig. 1.2, initially only one transition is enabled and can occur corresponding to the coordinator broadcasting a CanCommit message to all workers. After the occurrence of that transition four things can happen. Worker 1 can receive its CanCommit message first and reply with a Yes or a No, or worker 2 can receive its CanCommit message first and reply with a Yes or a No. Hence, nodes 3-6 correspond to the situations in which one worker has received a CanCommit message and has sent a reply. Nodes 7-10 correspond to states of the system where both workers have replied and the coordinator can collect the votes. The leftmost part (nodes 9 and 11) corresponds to an execution of the protocol in which both workers reply with a No. The two middle parts (nodes $7, 14, 18$ and nodes $10, 12, 15$) correspond to an execution of the protocol where one worker replies No and the other one replies Yes. The rightmost part (nodes $8, 13, 16, 17, 19$) corresponds to an execution of the protocol where both workers reply Yes. A directed path in the state space, i.e., a sequence of markings and binding elements starting in

```
1:  Nodes := {M_0}
2:  Arcs := ∅
3:  Unprocessed := {M_0}
4:  while Unprocessed ≠ ∅ do
5:              Select M_1 ∈ Unprocessed
6:              Unprocessed := Unprocessed − {M_1}
7:              {Process all enabled binding elements in M_1}
8:              for all (b, M_2) such that M_1[b⟩M_2 do
9:                      if M_2 ∉ Nodes then
10:                             Nodes := Nodes ∪ {M_2}
11:                             Unprocessed := Unprocessed ∪ {M_2}
12:                     end if
13:                     Arcs := Arcs ∪ {(M_1, b, M_2)}
14:             end for
15: end while
```

Figure 1.3: Basic algorithm for state space construction.

some node, corresponds to a so-called *occurrence sequence* of the CP-net.

From the state space it is, e.g., easy to check that the coordinator does not collect the votes received from workers (occurrence of transition CollectVotes) before all workers have sent their reply. It can be seen that the state space for two workers consists of 19 nodes and 27 arcs. For n workers the state space has $1 + 2 * 3^n = \Theta(3^n)$ nodes. This also demonstrates the state explosion problem – the number of nodes grows exponentially in the number of components in the system (in this case the number of workers).

The basic algorithm for construction of state spaces is listed in Fig. 1.3. The algorithm is essentially a modified version of a standard algorithm for traversal of a directed graph. Initially, a node corresponding to the initial marking (denoted M_0) is created (line 1), and this node is marked as unprocessed by inserting it into the set of unprocessed nodes/markings (denoted Unprocessed) (line 3). The algorithm then loops until no unprocessed nodes exist. In each iteration of the loop (lines 4-15) an unprocessed node/marking (denoted M_1) is selected. For each binding element which is enabled in that marking, the marking resulting from the occurrence (denoted M_2) of this binding element is calculated. The notation $M_1[b\rangle M_2$ used in line 8 means that the binding element b is enabled in M_1, and the occurrence of b in M_1 yields the marking M_2. If the resulting marking M_2 is not already included in the set of nodes, it is added and marked as being unprocessed (lines 9-11). In any event an arc is created (line 13) leading from M_1 to M_2 corresponding to an occurrence of the binding element b.[1]

Although the idea of using state spaces for verification of systems can be dated back to the very early papers on Petri Nets, the use of state space methods is not limited

[1]For the sake of completeness it should be mentioned that a state space, in contrast to a conventional directed graph, may have multiple arcs leading from one node to another node. This happens if two distinct binding elements which are enabled in a given marking have the same effect on the marking of the CP-net.

to Petri Nets nor CP-nets – state space methods have been used in a wide variety of other modelling formalisms. This means that verification algorithms and reduction methods also have been developed in other fields, and in many cases it is possible to (re)use ideas developed for other modelling formalisms within the context of Petri Nets. The state explosion problem is, for the same reason, not specific for CP-nets but present in all formal analysis methods based on the idea of constructing all reachable states and state changes of the system and representing these as a directed graph.

1.4 Motivation and Aims of Thesis

When the work on this thesis began in 1995, there had been only a few reports in literature on projects and case studies which applied the state space method for high-level Petri Nets (and CP-nets in particular). Simulation of CPN models had been applied in a number of projects, e.g., for performance analysis of *Usage Parameter Control* (UPC) algorithms in *ATM Networks* [28], *Transaction Processing and Interconnect Fabrics* [12], and *Document Storage Systems* [108]. Simulation of CPN models had also been used to investigate the logical correctness of systems, e.g., in the areas of *Intelligent Networks* [10], *VLSI Chips* [110], and *Network Management Systems* [15]. Since a simulation of a CPN model is a random execution of the model, it is like testing techniques – it can be used to detect errors but cannot be used to prove the correctness of a design. Limited use of state space methods had been reported in a few papers, e.g., in the area of *Integrated Services Digital Networks* (ISDN) [41], *Arbiter Cascades* [45], and *Distributed Program Execution* [80]. These three projects all applied a very early prototype of a state space tool for CP-nets. All projects applied *full state spaces*, i.e., they did not involve any kind of state space reduction method.

The overall motivation for the work in this thesis has been to advance the applicability of state space methods for CP-nets. The work has taken its origin in an early stand-alone prototype of a state space tool, which was integrated into the DESIGN/CPN tool with the participation of the author, while working as a student programmer in the Coloured Petri Net Group at the University of Aarhus in the beginning of 1995. The theoretical foundation for this early state space tool had been given in [63] and [67].

The theoretical foundation for some of the reduction methods based on exploiting symmetries and equivalence in systems was also given in [63] and [67]. However, no computer tool existed supporting these reduction methods, and, as a consequence, no practical case studies had been conducted which applied these reduction methods. A number of other reduction methods existed, such as the stubborn set method [120,125]. The application to CP-nets was, however, based on unfolding to the equivalent low-level Place/Transition Net [98, 107]. It is, in general, possible to transfer many state space methods developed for low-level Petri Nets, such as Place/Transition Nets, as well as other modelling formalisms to CP-nets by exploiting the fact that a CP-net can be unfolded to a Place/Transition Net with an equivalent behaviour. However, the unfolded form of a CP-net will generally be much bigger than the CP-net itself, and it may even be infinite causing the approach to fail. As a consequence, unfolding should be avoided if possible. Due to the powerful inscription language and data types offered by CP-nets it is, however, a challenge and a non-trivial issue to avoid this unfolding.

A main objective for the work in this thesis has been to develop state space methods and computer tools which work directly at the CP-net level without unfolding to the equivalent Place/Transition Net.

1.5 Overview and Structure of Thesis

The area of state space methods is a field in which theoretical development, computer tools, and practical applications go hand in hand. The theoretical developments are necessary in order to be able to develop sound computer tools, and computer tool support is needed in order to evaluate the theoretical developments in practice on more than just trivial examples. This is also reflected in the work done for this thesis which has been concerned with theoretical development, computer tools, and practical applications.

The work done for this thesis has been documented in five papers [19,74,75,86,87]. Part II of this thesis (Chapters 8 through 12) contains a reprint of these papers. Each of these chapters begins with a short description of the publication history and status of the paper contained in the chapter.

The rest of Part I, constituting the overview paper, is organised as follows:

Chapter 2 summarises the paper *State Space Analysis of Hierarchical Coloured Petri Nets* [19]. The paper presents joint work with Søren Christensen and has been published in *Petri Net Approaches for Modelling and Validation*, LINCOM Studies in Computer Science, No. 1, 1999 (To appear). The paper is contained in full in Chapter 8.

Chapter 3 summarises the paper *Computer Aided Verification of Lamport's Fast Mutual Exclusion Algorithm Using Coloured Petri Nets and Occurrence Graphs with Symmetries* [74]. The paper presents joint work with Jens B. Jørgensen and has been published in *IEEE Transactions on Parallel and Distributed Systems*, Vol.10, No. 7, pages 714-732, July 1999. The paper is contained in full in Chapter 9.

Chapter 4 summarises the paper *Verification of Coloured Petri Nets Using State Spaces with Equivalence Classes* [75]. The paper presents joint work with Jens B. Jørgensen and has been published in *Petri Net Approaches for Modelling and Validation*, LINCOM Studies in Computer Science, No. 1, 1999 (To appear). The paper is contained in full in Chapter 10.

Chapter 5 summarises the paper *Finding Stubborn Sets of Coloured Petri Nets Without Unfolding* [86]. The paper presents joint work with Antti Valmari and has been published in *Proceedings of 19th International Conference on Application and Theory of Petri Nets*, Volume 1420 of Lecture Notes in Computer Science, pages 104-123, Springer-Verlag, 1998. The paper is contained in full in Chapter 11.

Chapter 6 summarises the paper *Improved Question-Guided Stubborn Set Methods for State Properties* [87]. The paper presents joint work with Antti Valmari and

has been published as a technical report at the Department of Computer Science, University of Aarhus, DAIMI-PB 543, 2000. The paper is submitted to the 21st International Conference on Application and Theory of Petri Nets. The paper is contained in full in Chapter 12.

Chapters 2 through 6 are each divided into three sections. The first section gives an introduction and some background knowledge on the specific topic considered in the paper being treated. The second section gives a summary of the paper in question. The third section contains a comparison of the results contained in the paper with related work. Chapter 7 concludes on the work done for this thesis and presents some ideas and directions for future work.

1.5.1 Reading Guide

To read this overview paper detailed knowledge about CP-nets and state spaces is not required. It is, however, beneficial to be familiar with the basic ideas of Petri Nets such as Place/Transition Nets as described in, e.g., [98,107]. Readers without knowledge of Petri Nets who wish to study this thesis in more detail should start by reading the first part of [74] which contains an informal as well as a formal introduction to CP-nets and state spaces. The remaining papers [19, 75, 86, 87] can be read in any order. Readers with basic knowledge of CP-nets and state spaces can read the papers in any order. The papers have, however, been organised in Part II in the order in which it feels most natural to read the papers.

The summaries and discussions in Chapters 2 through 6 of results and related work are kept at an informal level due to the limited space available for this overview paper. This implies that here and there ambiguities and incomplete descriptions are unavoidable. The reader is referred to the specific papers for the precise definitions and explanations.

1.5.2 Terminology

The state space of a CP-net is traditionally called an *occurrence graph* [67] or a *reachability tree* [63]. For low-level Petri Nets, such as Place/Transition Nets [98, 107], they are also sometimes called *reachability graphs*. In this overview paper we will use the more modern term *state spaces*. The papers [19, 75, 86, 87] also use the term state space, whereas the paper [74] uses the term occurrence graphs. This means that the terms state space method, occurrence graph method, and reachability tree/graph method can be considered equivalent terms. The same holds true for state space analysis, occurrence graphs analysis, and reachability tree/graph analysis.

Chapter 2

State Space Analysis of Hierarchical Coloured Petri Nets

This chapter treats the paper *State Space Analysis of Hierarchical Coloured Petri Nets* [19]. Section 2.1 contains a brief introduction to the results in the paper. Section 2.2 gives a summary of the paper. Section 2.3 contains a discussion of related work.

2.1 Introduction and Background

The goals of the project on which the paper is based were to continue the development of an early stand-alone prototype of a state space tool for CP-nets and to integrate it into the DESIGN/CPN tool. This work has been ongoing throughout the work on this thesis, and the early prototype has evolved into the DESIGN/CPN OCCURRENCE GRAPH TOOL (OG tool) [14]. The OG tool is now a fully integrated part of the DESIGN/CPN tool. The OG tool has also served as a basis for the development of the DESIGN/CPN OE/OS GRAPH TOOL to which we will return in Chapters 3 and 4. Moreover, it has served as a basis for the experiments conducted as part of the work on the stubborn set method to which we will return in Chapters 5 and 6.

The title of the paper and this chapter refers to *Hierarchical Coloured Petri Nets*. This reflects that CP-nets support a hierarchical structuring mechanism which makes it possible to split a CPN model into a number of modules (in CPN terminology called *pages*). This hierarchical structuring mechanism makes it possible to work bottom-up as well as top-down when constructing models, and it makes it possible to reuse a module in different parts of a CPN model. This is important when constructing CPN models of large systems. As we will see, it is possible to take advantage of the hierarchical structuring in the storage of state spaces. All results presented in this thesis are also valid for Hierarchical Coloured Petri Nets, and in the remainder of this overview paper we will therefore omit the word *Hierarchical* and use the terms Coloured Petri Nets, CP-nets, and CPN models also for the hierarchical case.

2.2 Summary of Paper

The focus of the paper is twofold: firstly, to present state space analysis of CP-nets as supported by the OG tool and as seen from the users point of view, and secondly to present the core datastructures and algorithms implemented in the OG tool. The

graphical interface of the OG tool is implemented in C [82], whereas the core algorithms and datastructures for state space generation, state space storage, verification, and analysis are implemented in STANDARD ML [95, 119]. The central components in the OG tool are summarised below.

State Space Generation. The OG tool supports two modes in which the user can generate state spaces: *interactive* and *automatic*. In interactive generation the user chooses a marking, and the OG tool then calculates both the enabled binding elements in this marking and the successor markings resulting from the occurrence of each of these binding elements. Interactive generation is typically used in connection with visualisation (see below). In automatic generation the state space is calculated fully automatically without any intervention from the user. The automatic state space generation is based on the algorithm from Sect. 1.3 (Fig. 1.3) and uses a queue to implement a breadth-first generation of the state space. The DESIGN/CPN simulator is used to calculate the set of enabled binding elements in each marking encountered during state space generation.

The generation can be controlled using so-called *stop* and *branching options*. The generation control is particularly important in early phases of state space analysis which are typically concerned with identifying a configuration of the system which is tractable for state space analysis. The stop and branching options also make it possible to obtain a *partial state space*. A partial state space is a subset of the full state space. Partial state spaces are supported since in many situations it is not necessary to generate the full state space in order to check the correctness of a system. Moreover, partial state spaces can be quite effective in locating and detecting errors.

State Space Storage. The storage of the state space (i.e., the nodes and arcs and their relationship) is based on a datastructure which exploits the hierarchical structure of a CP-net and the locality of Petri Nets. It is based on the observation that the occurrence of a transition in one module in a hierarchical CP-net 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. This can be exploited in the representation of markings to avoid duplicate representation of complex values.

Markings are stored on three levels: the *Global level*, the *Module level* and the *Multi-set level*. The Multi-set level is concerned with the storage of the markings of the individual places. The Module level is concerned with the storage of markings for the individual modules. A *module state* (MS) has an entry for each place in the module, which is a pointer into the multi-set storage corresponding to that place. In this way, multi-sets can be *shared* among the MSs. The Global level is concerned with the storage of markings 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 among the GSs. To make insertion in the storages efficient, all storages are implemented as a variant of AVL trees [83]. In the paper it is demonstrated by means of a representative example that this datastructure is capable of reducing the number of multi-sets which have to be stored to around 0.1% of the multi-sets which should have been stored if no sharing was done. Similarly, it is demonstrated that the number of module markings can be reduced to around 2% of the module markings which should have been stored if no sharing was done. In addition,

the datastructure makes it efficient to check whether a marking is already included in the state space. This is important for the implementation of the state space generation.

State Space Report. The OG tool makes it possible to generate a *state space report* which is a textual file containing answers to a set of standard behavioural properties which can be used on any CPN model (i.e., generic properties that can be formulated independently of the system under consideration). The state space report contains information about the *boundedness*, *home*, *liveness*, and *fairness* properties of the CPN model. Practical use has shown that the properties of a system which are investigated first are very often contained in this set of system independent properties. The *state space report* can be produced totally automatically, and by studying the state space report the user gets a rough idea, as to whether the CPN model works as expected. If the system contains errors, they are often reflected in the state space report.

Query Languages. In addition to the state space report, the OG tool also offers a set of *standard query functions* that allow the user to make a more detailed inspection of the standard behavioural properties. Many of these query functions return results which are already included in the state space report. The implementation of the standard query functions are based on the proof rules given in [67]. A proof rule states a relationship between a dynamic property of a CP-net and the state space of the CP-net. The standard query functions for boundedness properties consist essentially of a traversal of the generated state space. The standard query functions for home, liveness, and fairness properties exploit *strongly connected components* (SCCs) which can be computed using TARJAN's algorithm [48] in linear time and space in the size of the state space.

The state space report and the standard queries are good at providing a rough picture of the behaviour of the system. However, they also have some limitations since many interesting properties of systems cannot easily be investigated using the system independent standard queries. Moreover, for debugging of systems more elaborate queries are often needed for locating the source of the problem. Therefore, a more general query language implemented on top of STANDARD ML is provided. It provides primitives for traversing the state space in different ways and, thereby, for writing non-standard and system dependent queries.

Visualisation. Since state spaces often become large, it seldom 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 marking to another. A good and quick way to get detailed information about a small number of nodes and arcs is to draw the corresponding fragment of the state space. This makes visualisation particularly useful when locating errors in a system under consideration. 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. Visualisation of state spaces is often used in conjunction with interactive generation. Figure 1.2 was obtained in this way.

Integration with Simulation. During a modelling and design process, the user quite often switches between state space analysis and simulation. The OG tool is tightly integrated with the simulator to support this. This makes it possible to transfer markings between the simulator and the OG tool. When a marking is transferred from the OG

tool into the simulator, the user can inspect the marking of the individual places, and the enabling of the individual transitions. It is also possible to start a simulation from the transferred marking. Transferring the current marking of the simulator into the OG tool is supported as well. A typical use of this is to investigate all possible markings reachable within a few steps from the current simulator marking. In this case, the user transfers the simulator marking into the OG tool and all successor markings can be found using interactive generation combined with visualisation.

2.3 Related Work

An abundance of verification and analysis frameworks based on state space methods have been developed supporting different modelling languages and classes of systems (see e.g., [39]). Even so, they typically consist of the same basic components: a language for modelling the systems, algorithms and datastructures for generation and storage of state spaces, a language for specifying analysis and verification questions, and algorithms for determining the answers to these questions. In the following we discuss the basic components of the OG tool from this perspective, and compare them to similar components found in other frameworks for computer-aided analysis and verification. To the extent possible the discussion is attempted to be kept at a conceptual level rather than being tool specific.

Temporal Logics. The language for specifying analysis and verification questions in the OG tool is based on the *proof rules* in [67] made available as a collection of query functions corresponding to the standard dynamic properties of CP-nets.

Propositional Temporal Logic [31] is another very widely used way of formulating properties about systems. A temporal logic consists essentially of atomic propositions, propositional operators (such as "$\wedge$" and "$\vee$"), and temporal operators. The role of the atomic propositions is to provide an abstraction mechanism which makes the link to the modelling language. For example, in a state oriented temporal logic for CP-nets, a possible atomic proposition could be $|M(p)| \leq k$ where $|M(p)|$ denotes the number of tokens on a place p in a marking M, and k denotes an integer constant. This atomic proposition is valid in a marking if and only if p contains at most k tokens in the marking M. The role of the temporal operators is to express temporal relationships between the atomic propositions. An example of a typical claim expressed using temporal logic could be that in all reachable markings the atomic proposition $|M(p)| \leq k$ is valid. This can be expressed using a temporal operator "always" (often denoted $\Box$). The claim would then correspond to the temporal logic formula $\Box(|M(p)| \leq k)$. A temporal logic thus provides a query language for expressing temporal properties of the system under consideration. A *model checking algorithm* [25] is an algorithm which takes a temporal logic formula ϕ and a state space SS, and determines whether SS is a so-called *model* of ϕ, i.e., whether ϕ is a true statement about SS. A wide variety of different temporal logics, differing in their expressive power, have been developed, but two logics in particular have been in wide-spread use: *Linear Temporal Logic* (LTL) [47, 132] and *Computation Tree Logic* (CTL) [25]. LTL is used, e.g., as query language in the SPIN [61, 118] and PROD [117, 134] tools. CTL is used, e.g., as query language in the SMV [93, 112] and PEP [51, 62] tools.

Temporal logic can also be used to express standard dynamic properties of a CP-net. Boundedness properties and dead transitions/binding elements (transitions/binding elements which can never become enabled) can be expressed in both CTL and LTL. Home properties and liveness of transitions/binding elements (transitions/binding elements which can always become enabled) can be expressed in CTL. Fairness properties of transitions and binding elements can be expressed in LTL. It is therefore relevant to ask whether the standard query functions are needed in the OG tool since they would be automatically available by supporting CTL and LTL model checking. The answer to this question can be found in the results produced by model checking algorithms. A model checking algorithm outputs either *Yes* or *No* depending on whether the formula provided as input is a true statement about the state space or not. In case the answer is *No*, then very often a subset of the state space, e.g., a path, is given as a counter example. As an example consider *integer bounds*. An integer k is an upper bound of a place p if and only if in all reachable markings there are no more than k tokens on p. However, in practice one is very often interested in the *best upper integer bound* of p which is the minimal k which is an upper integer bound for p. To find the best upper integer bound using temporal logic, the temporal logic formula from above could be used in a sequence of questions on the form $\Box(|M(p)| \leq k)$ in order to find the minimal k for which the formula is valid.

The result returned by a standard query function of the OG tool is typically not a Yes/No answer, but a more complex value such as an integer giving the best upper integer bound of a place, or the list of dead markings (markings without enabled transitions). Moreover, the standard query functions have a direct implementation and are not based on determining the answers to a sequence of temporal logic queries like in the example above.

The above discussion demonstrates that standard query functions are more geared towards *analysis* in the sense of asking what the properties of the system are, and then inspecting and interpreting the results afterwards. Temporal logic on the other hand is more geared towards *verification* in the sense of stating a property and then checking whether the system has this property or not. When experimenting with different design ideas and fixing an incorrect design, one often switches between analysis and verification. Both standard queries and temporal logic therefore have their justification as query languages. In fact, the OG tool supports temporal logic by means of the library ASK-CTL [20]. This library makes it possible to make queries formulated in a state and action oriented variant of CTL [11]. The fact that ASK-CTL is both state and action oriented reflects the fact that Petri Nets are both state and action oriented.

State Space Storage. Because of the state explosion problem, it is essential to provide a succinct representation of the markings of the CP-net. The OG tool exploits the hierarchical structure of CPN models and the locality of Petri Nets as sketched in the previous section to achieve a compact representation of the state space. The basic observation here is that 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. A similar technique has been used in the SPIN tool [61, 118] which uses the specification language PROMELA (Process Meta Language) for construction of models. Here the local states of the processes and channels in the PROMELA description are stored separately from the global states. The

SPIN tool has in general been subject to much research on approaches for obtaining more compact representation of state spaces [58]. Below the more prominent results are surveyed.

Graph Encoded Tuple Sets (GETS) [52] is a datastructure for storing sets of tuples in a compact way. The basic idea is to encode the states as tuples where an element in the tuple represents, e.g., a state of a process in the model. The compact representation of GETS is obtained by sharing common suffixes and prefixes of the tuples which are represented using a graph structure. The experimental results in [52] showed that this reduced memory consumption with a factor of 7 to 8 at the expense of a factor 3 increase in generation time on some selected SPIN models.

The use of minimised *Deterministic Finite Automatons* (DFAs) to store the set of reachable states has been investigated in [60]. The basic idea is to encode states as strings of a fixed length k over an alphabet Σ of bits or bytes. The alphabet Σ of the DFA is determined by the level of encoding, e.g., for bit-level encoding $\Sigma = \{0, 1\}$ and for byte level encoding $\Sigma = \{0, .., 255\}$. The length k of the strings is determined by the number of bits/bytes needed to represent a state. During state space generation a DFA is maintained which accepts exactly the strings corresponding to the reachable states generated so far. The experimental results in [60] showed that a substantial memory reduction can be obtained with only a small overhead in generation time. The DFA representation seems to represent a better time-space trade-off than GETS.

Reduced Ordered Binary Decision Diagrams (ROBDDs) [8] is another datastructure which has been successfully applied for representing state spaces [9, 93]. A ROBDD can provide a canonical representation of a boolean function of n boolean variables, or equivalently a set of bit-vectors of length n. A ROBDD is a directed, acyclic graph where the compact representation of the boolean function is achieved by merging nodes which are roots of isomorphic subgraphs and by removing nodes which are redundant. In the following we will use BDD as an abbreviation of ROBDD.

The basic idea in application of BDDs for representing state spaces is to represent sets of states as a BDD $S(v_1, \ldots, v_n)$ of n boolean variables $v_1, v_2, \ldots, v_n$, and the transition relation of the system as a BDD $R(v_1, v_2, \ldots, v_n, v'_1, v'_2, \ldots, v'_n)$ of $2n$ variables. If we let $v = v_1, v_2, \ldots, v_n$ and $v' = v'_1, v'_2, \ldots, v'_n$, then the BDD R is such that $R(v, v')$ is true iff there is a transition which is enabled in the state represented by v and whose occurrence leads to the state represented by v'. The set of reachable states can be computed starting from a BDD representing the initial state and then continuously repeating the iteration $S(v) := S(v) \vee \exists v' : S(v') \wedge R(v', v)$ until a fixpoint is reached. In the k'th iteration the states reachable in less than or equal to k steps from the initial marking will be contained in the BDD. The canonical form of a BDD is used to detect when the fixpoint is reached, and it is exploited that conjunction (disjunction) between BDDs can be computed efficiently yielding a new BDD representing the conjunction (disjunction) of the two BDDs. The use of BDDs has been particularly successful in *symbolic model checking* where BDDs are combined with the model checking procedures for the temporal logics CTL and LTL. BDDs have proven to be a very compact way to represent state spaces and significant results have been obtained with them – in particular in the area of hardware circuit verification. However, BDDs also have some drawbacks. Firstly, to be efficient they require some currently not well-understood form of regularity in the system. Secondly, they are highly sensitive to the variable ordering chosen. Furthermore, the memory require-

ments are somewhat unpredictable because there is no general correlation between the size of the state space and the size of the BDD representing it. In particular, the size of the intermediate BDDs during generation of the state space is often a problem.

The use of BDDs in the context of Petri Nets has been considered in [100]. The results from [100] consider k-bounded Place/Transition Nets, i.e., Place/Transition Nets which contains at most k tokens on place. These results are not directly applicable to CP-nets, since the elaborate notion of data types and inscription language prohibit compiling a BDD representing the transition relation directly from the CP-net. It is, however, possible to apply BDDs to CP-nets using the approach taken in [135]. Here the transition relation is not represented as a BDD, but instead a conventional state space generation algorithm is applied, and the BDD is only used to store the set of reachable states encountered so far. The experimental results presented in [135] showed that this yielded a reduction in memory but at the expense of a significant increase in running time.

The advantages of using GETS, DFAs, and BDDs, in the context of CP-nets compared to the datastructure currently used in the OG tool has not been investigated. The application of GETS in the context of CP-nets requires an encoding of the markings as tuples. A simple approach would be for a CP-net consisting of the places $p_1, p_2, \ldots, p_n$ to encode a marking M as the tuple $(M(p_1), M(p_2), \ldots, M(p_n))$. However, this would require augmenting GETS with the capability of being able to change the encoding of states on-the-fly, since it is not generally possible to determine in advance how many bits/bytes are needed to store a marking of a place. A similar remark applies to DFAs and BDDs for representing the reachable markings of a CP-net. This is also the reason why the results in [100] are restricted to Petri Nets for which it is possible to determine in advance a k such that no place contains more than k tokens in any reachable marking.

Visualisation. Another central component in the OG tool is the support for visualising selected parts of the state space. The recognition of visualisation of the state space as a means to analyse the behaviour of systems has also been considered in a process algebraic setting in [130]. The approaches taken in the OG tool and in [130] are, however, different. In [130], the view of 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.

Summarising the discussions above, it can be seen that the OG tool contains the same basic components as most other tools for computer-aided analysis and verification. The strengths of the OG tool compared to many other tools are, however, that it is closely integrated with a simulator, that it supports visualisation and interactive state space generation, and that it aims at supporting both analysis and verification.

Chapter 3

State Spaces with Symmetries and Lamport's Algorithm

This chapter treats the paper *Computer Aided Verification of Lamport's Fast Mutual Exclusion Algorithm Using Coloured Petri Nets and Occurrence Graphs with Symmetries* [74]. Section 2.1 contains a brief introduction to state spaces with symmetries, which is the main subject of the paper. Section 2.2 gives a summary of the paper. Section 2.3 contains a discussion of related work.

3.1 Introduction and Background

Many concurrent systems possess a certain degree of symmetry. For example, many concurrent systems are composed of similar components whose identities are immaterial or interchangeable from a verification point of view. This kind of structural symmetry is also reflected in the state spaces of such systems. The basic idea in state spaces with symmetries is to factor out this symmetry, and to obtain a *condensed state space* which is typically orders of magnitude smaller than the ordinary full state space, but from which the same kind of dynamic properties can be directly verified and analysed without unfolding to the full state space.

A concrete example of a system possessing symmetry is the commit protocol introduced in Chap. 1 (see Fig. 1.1). Intuitively, this protocol is symmetric in the sense that it treats the workers in the same way, i.e., all workers behave in the same way – they are "symmetric". The fact that the workers are symmetric is also reflected in the state space (see Fig. 1.2). Consider for instance the two markings[1] M_3 and M_5, which correspond to states in which exactly one worker has received the CanCommit message and sent a No reply back to the Coordinator. These two markings are symmetric in the sense that M_3 can be obtained from M_5 by swapping the identity of the two workers. Similarly, the two markings M_4 and M_6, which correspond to states in which exactly one worker has sent a Yes reply back, can be obtained from each other by interchanging the identity of the workers. Furthermore, it can be observed that two symmetric markings such as M_3 and M_5 have symmetric sets of enabled binding elements, and symmetric sets of successor markings. Using induction, this property can be expanded to finite and infinite occurrence sequences. This justifies that if we have a set of symmetric markings then it is sufficient to explore the possible behaviours of the system for only one of these markings. The CPN model of the commit protocol

[1] M_i denotes the marking corresponding to node i.

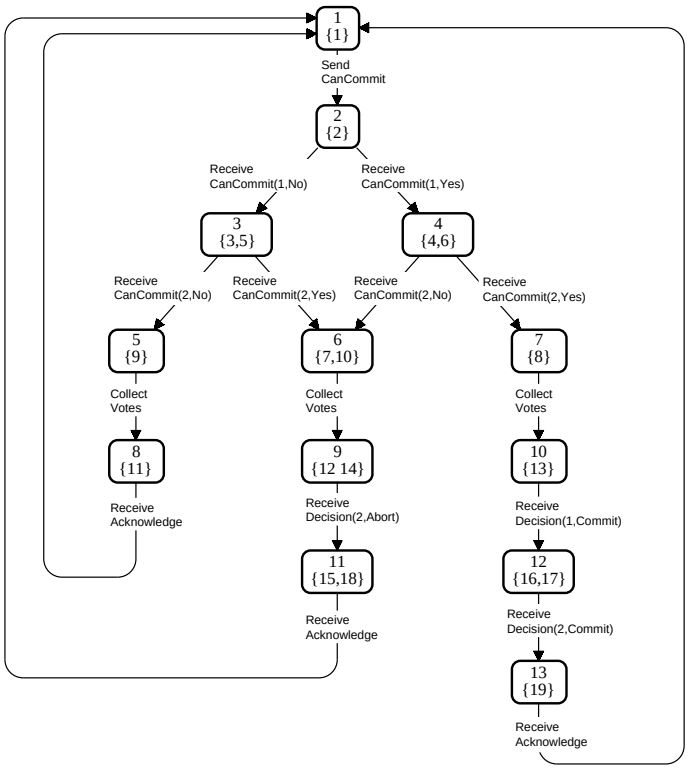

Figure 3.1: Condensed state space for two-phase commit protocol in Fig. 1.1.

contains many markings that are symmetric in this way. The basic idea in condensed state spaces is to lump together such symmetric markings and symmetric binding elements into *equivalence classes.* Figure 3.1 shows the condensed state space for the commit protocol obtained by considering the two workers to be symmetric, i.e., allowing permutations of the identity of workers. The nodes and arcs now represent equivalence classes of markings and binding elements, respectively. The equivalence class of markings represented by a node is listed in brackets in the inscription of the node, e.g., node 3 represents the markings M_3 and M_5 from Fig. 1.2. The condensed state space for the commit protocol has 13 nodes and 16 arcs. For n workers the condensed state space has $(n+1)*(n+2) = \Theta(n^2)$ nodes. This demonstrates that the reduction obtained by exploiting symmetries can indeed be orders of magnitude.

Symmetry is technically expressed by means of a so-called *permutation symmetry specification* which assigns an algebraic *symmetry group* of permutations to each *atomic colour set* of the CP-net[2]. A symmetry group determines how the colours of an atomic colour set are allowed to be permuted. For example, a symmetry group may specify that all colours can be permuted arbitrarily, or that they must all be fixed, i.e., cannot be changed. Many intermediate forms exists, e.g., all rotations of a finite, ordered colour set. A permutation symmetry specification determines a group Φ of

[2]Each place of a CP-net has an associated colour set which is similar to a type in a programming language. The colour set determines the kind of tokens which can reside on the place. An atomic colour set is a colour set (type) defined without reference to other colour sets.

permutation symmetries which in turn induces two equivalence relations — one on the set of markings and one on the set of binding elements. Two markings M_1 and M_2 are considered symmetric (equivalent) if and only if there exists a permutations symmetry $\phi \in \Phi$ such that $M_1 = \phi(M_2)$, and similarly for binding elements. The fact that the symmetry groups are algebraic groups ensures that the relations on markings and binding elements are indeed equivalence relations.

A permutation symmetry specification is required to capture inherent symmetries of the system, i.e., symmetries which are actually present in the system. A permutation symmetry specification in accordance with the system is said to be *consistent*. The consistency requirement ensures that symmetric markings have symmetric sets of enabled binding elements, and symmetric sets of successor markings. This in turn ensures that dynamic properties of the CP-net can actually be derived from the condensed state space.

In the following we will use the terminology from [67], and let *OS-graphs* denote condensed state spaces obtained using permutation symmetry specifications. In Chap. 4 we will consider a generalisation of OS-graphs based on allowing general equivalence relations on markings and binding elements to be used. Such condensed state spaces will be denoted *OE-graphs*.

3.2 Summary of Paper

The goals of the project on which the paper is based were to develop computer tool support for OE- and OS-graphs and to conduct some first practical case studies on the use of such state spaces for CP-nets. The two main contributions of the paper are the presentation of the developed DESIGN/CPN OE/OS GRAPH TOOL (OE/OS tool) and the use of OS-graphs to establish the correctness of LAMPORT'S FAST MUTUAL EXCLUSION ALGORITHM [88]. The paper is written to be self-contained and does not assume prior knowledge of any kind of Petri Nets or state spaces.

Lamport's Algorithm is a mutual exclusion algorithm for shared-memory multiprocessors. A shared-memory multiprocessor is an architecture consisting of a number of CPUs connected to a common bus and with a single shared memory. It is assumed that the memory supports atomic read and write operations and that each process has a unique identifier, which is a positive integer. Lamport's Algorithm is symmetric in the sense that it treats all processes in the same way.

The CPN model of Lamport's Algorithm was created by translating each of the statements of Lamport's Algorithm (written in [88] as pseudo-code) into CP-net constructs. The creation of the complete model consisted of putting all of the pieces together. This systematic strategy reduces the probability of accidental errors, and justifies that the constructed CPN model is a proper model of the algorithm. One exception to the above scheme should, however, be mentioned. It is related to the application of symmetries and the modelling of for-statements. Lamport's Algorithm contains a for-statement in which the entries in a boolean array with an entry for each processor is tested. If the entries are tested in turn starting from the processor with identity 1, then an ordering is imposed on the processes in Lamport's Algorithm. Hence, all processes are not treated in the same way from a symmetric point of view. Therefore, two variants of Lamport's Algorithm were considered – one in which no ordering was

imposed, and one in which all entries were tested in one atomic test.

The OE/OS tool is based on the OG tool previously discussed in Chap. 2 and supports verification and analysis by means of OE- and OS-graphs. The tool supports visualisation, analysis, and verification in the same way as the OG tool, except that it deals with OE- and OS-graphs instead of full state spaces. The OE/OS tool supports user-supplied permutation symmetry specifications. The user supplies the information by implementing two predicates (functions). The first predicate expresses when two markings are symmetric/equivalent, and the second predicate expresses when two binding elements are symmetric/equivalent. The two predicates must reflect the permutation symmetry specification, i.e., the symmetry groups assigned to the atomic colour sets. It is the responsibility of the user to ensure that the predicates implement a consistent permutation symmetry specification. This amounts to checking that the arc expressions, guards, and initial markings commute with the permutation symmetry specification in the way precisely defined in [67]. For Lamport's Algorithm this consistency proof consists of a number of cases all of which were essentially trivial. The risk of making mistakes in the manual consistency proof and implementation of the predicates was alleviated using an algorithm developed as part of the case study of Lamport's Algorithm [79]. The algorithm is based on algebraic group theory and makes it possible to compute the size of the full state space from the OS-graph (without unfolding). This makes it possible to compare the size of a generated full state space with the size computed based on the corresponding OS-graph. Comparing these sizes has in practice turned out to express a quite strong necessary condition for consistency and correct implementation of the predicates.

The OS-graph (OE-graph) is computed on-the-fly, i.e., *without* first constructing the full state space and then grouping markings and binding elements into equivalence classes. This is done using a modified version of the algorithm for generating an ordinary state space (see Fig. 1.3). Instead of testing whether the marking is already included in the state space, it is tested whether a symmetric/equivalent marking is already included. Similarly, there is a test of whether a symmetric/equivalent binding element is already inserted between two nodes, before a new arc is created. The OE/OS tool stores equivalence classes using representatives: each node in the OS-graph (OE-graph) is represented by a marking from its equivalence class. Analogously for arcs and binding elements. Markings and binding elements are stored using the same data structures as in the OG tool. The query language consists of a number of standard query functions corresponding to the standard dynamic properties of CP-nets. These query functions are complemented by a set of more general functions for traversing the condensed state space and for writing non-standard and model dependent queries. The standard query functions contain a full implementation of the proof rules for OE- and OS-graph from [67]. The virtue of these standard query functions is that they work directly on the condensed state space without unfolding the equivalence classes.

The verification of Lamport's Algorithm established a number of crucial properties such as *mutual exclusion, persistent reachability* of the critical section, and *absence of deadlocks*. The case study demonstrated significant reductions in the number of nodes and arcs. For example, for four processes the number of nodes is reduced from around 2,000,000 to around 90,000. Another important finding from these practical experiments was that in addition to memory, time was also saved, i.e., the generation of the OS-graph was faster than the generation of the full state space for the same number

of processes. This is surprising since one would expect that the more expensive test related to checking whether a symmetric marking is already in the state space has a negative impact on time performance. The results however suggest that what is lost on a more expensive test on symmetry of markings and binding elements, is accounted for by having much fewer nodes and arcs to generate; and also to compare with before a new node or arc can be inserted in the OS-graph. Lamport's Algorithm was verified for up to four processes when the entries in the for-statement were tested in an arbitrary order, and up to seven processes when the entries were tested in one atomic test.

3.3 Related Work

The idea of exploiting symmetry was proposed for CP-nets in [63]. The OE/OS tool is based on the formulation of state spaces with symmetries given in [67,68]. Meanwhile, the idea of exploiting symmetry has also been applied and further developed in other modelling languages and frameworks.

Symmetry and Model Checking. Exploiting symmetry in model checking of the temporal logic CTL* has been investigated in [26] and [34]. The temporal logic CTL* contains the temporal logics CTL and LTL as subsets. In both cases, the underlying model of computation is not Petri Nets but labelled transition systems and shared variable programs. Exploiting symmetries when investigating safety properties, e.g., deadlocks and invariant violations, has been investigated in [65]

The symmetry reduced state spaces preserving the truth value of a temporal logic formula in [26] and [34] are obtained using an algebraic group of permutation symmetries which is contained in the intersection of the symmetries present in the system and the symmetries present in the temporal logic formula to be checked. The combination of symmetries and symbolic model checking, i.e., when BDDs (Binary Decision Diagram) are used for computing and representing the state space, is also investigated in [26]. These results show that the size of the BDDs used to represent the equivalence classes grows at least exponentially in the minimum of the number of processes of the system and the number of states of one process for frequently appearing symmetry groups (such as those consisting of all permutation or all rotations). Hence, exploiting symmetry in symbolic model checking is restricted to either systems with a small number of processes or processes which have a small number of states. The intuitive reason for BDDs not working well with symmetry reduction is the lack of correlation between the size of the BDD and the size of the state space it represents. Symmetries reduce the number of nodes and arcs which have to be stored – but with BDDs it may take a larger BDD to represent this smaller number of nodes and arcs.

Exploiting symmetry when model checking under fairness assumptions has been investigated in [33] and [53]. Model checking under fairness assumption consists of restricting the model checking to consider only those executions of the system which are fair according to some fairness criteria. An example of a fairness assumptions is *strong fairness* which requires that if a binding element is infinitely often enabled then it also occurs infinitely often. Fairness assumptions are typically needed to establish *progress properties* such as "eventually place p contains a token". The conventional way of taking fairness into account when checking a property ϕ in, e.g., LTL model checking, is to express the fairness assumption as a formula ϕ_{Fair} in LTL and then

perform model checking using the formula $\phi_{Fair} \Rightarrow \phi$. The approaches to symmetries and model checking in [26] and [34] fail to handle this effectively since they are based on intersecting the symmetry in the temporal logic formula with the symmetry in the system, and the fairness assumptions expressed as ϕ_{Fair} in most cases restrict the symmetry group of the formula to consist of the identity permutation only, thereby prohibiting any reduction to be obtained.

Verification of Lamport's Algorithm. In our search for a good example to demonstrate the use of the OE/OS-tool for verification, the inspiration to consider Lamport's Algorithm came from [3]. Here, the authors verify Lamport's Algorithm using Coloured Stochastic Petri Nets [91] and *place invariants.* Place invariants is one of the other main analysis methods of Petri Nets. The basic idea resembles that of invariants or assertions known from establishing correctness of conventional computer programs. A place invariant determines an equation which holds for all reachable markings of a Petri Net. Verification of Lamport's Algorithm in [3] is conducted on a model in which the for-statement is modelled in the coarse fashion where all entries in the boolean array are tested atomically. An advantage of the approach in [3] is that Lamport's Algorithm is verified for an arbitrary number of processes. The capability of verifying Lamport's Algorithm for an arbitrary number of processes is obtained by the combination of place invariants and manual reasoning.

In the original presentation of Lamport's Algorithm in [88], Lamport himself establishes correctness. Here an axiomatic method decorating the algorithm with assertions is applied. Lamport concentrates on establishing deadlock freedom and mutual exclusion. As in [3], the properties are proved for an arbitrary number of processes. Both [3] and [88] conduct complex and lengthy mathematical proofs. For the mutual exclusion property, the former only sketches the proof, while the latter more generally relies on a number of proof sketches. With respect to the logical behaviour of the algorithm, we establish similar properties to [3] and [88], plus other important properties. The main virtue of our proof is that it is almost automatic and, hence, much less error-prone. We do not need to engage in detailed or complex mathematical arguments. Based on this, we claim that our results are quite reliable. However, the disadvantage of our method is that it is necessary to fix the system parameter – in this case the number of processes. Moreover, the number of processes which could be handled was restricted to 4 (or 7 with a coarser modelling of the for-statement).

Computer Tool Support. Developing tool support for OS-graphs involved making a number of design decisions as to how the tool should support the user in conducting verification of systems.

A key design choice is whether the symmetries should be automatically detected by the tool or be provided by the user based on knowledge of the system under consideration. We have chosen the latter approach for several reasons. Firstly, computing the symmetries is expensive, and it is our experience that the user always has some knowledge/intuition about the potential symmetries of the system. For example, in Lamport's Algorithm, it is obvious that the symmetry is in the processes. Secondly, if the tool detects the symmetries in the system, it might result in symmetries which are difficult to interpret. Furthermore, the symmetries which the user has in mind are often symmetries which should be present in the system, and if they are not, this might indicate a design error. Therefore, an automatic calculation of symmetries does not in the

same way allow design errors to be caught as with our approach. An example of this is the for-statement in Lamport's Algorithm. If the tool had automatically calculated the symmetries of the system, then the result would have been that no symmetry was present in the system, and symmetry reduction could not have been applied. But with our approach, we could identify the problem and then verify a more abstract version of Lamport's Algorithm in which the for-statement causing the problem was modelled in a different way.

With respect to detection of symmetries, our approach coincides with the approaches to exploit symmetry described in [22, 23, 26] and implemented in the SYMM tool. This is in contrast to the MURPHI tool [65] which relies on automatic detection of symmetries directly from the syntax of the modelling language. The same is the case for the SMC tool [34, 111]. The symmetry groups in [65] are, however, restricted to algebraic groups consisting of all permutations of a given set or groups consisting of the identity permutation only.

An alternative to computing the symmetries is to put narrow, syntactical restrictions on the modelling language in such a way that only symmetric constructs are expressible. Such ideas have been pursued for Well-formed Coloured Nets (WNs) [13]. Detection of symmetries in WNs can be fully automated, thus effectively eliminating the need of conducting a consistency proof. For flexibility reasons, we have chosen not to base the OE/OS tool on putting syntactical restrictions on CP-nets. This implies that a proof of consistency has to be conducted. Proving the consistency of the permutation symmetry specification is tedious, because of the many cases in the proof, which need to be considered. Therefore, it would be preferable, if the tool could check most of these cases automatically. The tool may not be capable of conducting a full proof of consistency, but it may significantly reduce the number of cases that the user needs to consider.

Practical experiments with the OE/OS tool have clearly identified two points at which the OE/OS tool has to be improved in order to make it more applicable to larger systems. One point is support for automatically generating the symmetry predicates from the permutation symmetry specification, the other point is support for an automatic (or almost automatic) consistency check. These improvements will in addition have the effect of hiding more of the mathematics underlying the OE/OS tool, and less programming skills would be required from the user.

Checking Equivalence. A fundamental aspect of putting OS-graphs into practice is to be able to determine whether two markings (binding elements) are symmetric or not, i.e., whether for two markings M_1 and M_2 (binding elements b_1 and b_2) there exists a permutation symmetry ϕ such that $M_1 = \phi(M_2)$ ($b_1 = \phi(b_2)$). This problem is also referred to as the *orbit problem*, and its computational complexity has been investigated in a several papers. It was shown in [26] that the orbit problem is at least as hard as the *graph isomorphism problem* for which no polynomial time algorithm is known. These results were later extended in [22] showing that the orbit problem is equivalent to important problems in computational group theory which are harder than the graph isomorphism problem. It is not known whether these problems are NP-complete. The *constructive orbit problem* which is related to obtaining a canonical representative for each equivalence class is also investigated in [22]. The constructive orbit problem is known to be NP-hard.

The above complexity results may indicate that the use of OS-graphs is impractical. However, it is our experience that what is lost on an expensive test of equivalence of markings and binding elements, is made up for by having fewer nodes and arcs to generate; and also to compare with before a new node or arc can be inserted in the OS-graph. The experiments with tools supporting reduction by means of symmetry that were reported on in [65] and [26] confirm the above observations. However, practical experiments have subsequently shown that for certain CP-nets with highly structured colour sets, the generation of the OS-graph is faster only when the number of permutation symmetries is small – typically less than 120. This suggests that the use of algebraic theory and algorithms to speed up the equivalence check is an area which deserves more attention in the future. Some work on this has already been done in the context of CP-nets in [2, 43, 76].

Chapter 4

State Spaces with Equivalence Classes and the Transport Protocol

This chapter treats the paper *Verification of Coloured Petri Nets Using State Spaces with Equivalence Classes* [75]. Section 4.1 contains a brief introduction to state spaces with equivalence classes which is the main subject of the paper. Section 4.2 contains a summary of the paper. Section 4.3 contains a discussion of related work.

4.1 Introduction and Background

Like in the definition of OS-graphs, the definition of an OE-graph for a CP-net requires the presence of two equivalence relations. An *equivalence specification* consists of two equivalence relations – one on the set of markings ($\approx_M$) and one on the set of binding elements ($\approx_{BE}$). However, unlike the definition of OS-graphs, where the equivalence relations are derived from the algebraic groups of permutations assigned to the atomic colour sets, there is no requirement on the origin of the two equivalence relations. They can, roughly speaking, be arbitrary equivalence relations on the set of markings and binding elements, respectively. In this way, OS-graphs are a special case of OE-graphs.

To be able to use an OE-graph to verify dynamic properties, the equivalence relations must capture an equivalence actually present in the considered system. Similar to OS-graphs this is referred to as *consistency*. It requires that for all markings M_1, M_1', and binding elements b we have that $M_1 \approx_M M_1'$ and $M_1[b\rangle M_2$[1] implies the existence of a binding element b' and a marking M_2' such that $M_2 \approx_M M_2'$, $b \approx_{BE} b'$, and $M_1'[b'\rangle M_2'$. This property is illustrated in Fig. 4.1. The consistency requirement ensures that equivalent markings have equivalent sets of enabled binding elements, and equivalent sets of directly reachable markings. Like with OS-graphs this justifies that it is sufficient to explore the possible behaviours of the system for one marking of each equivalence class. Construction of the OE-graphs can be done using the same algorithm as for construction of the OS-graphs.

OE-graphs are presented in [67] as a theoretical generalisation of state spaces based on symmetries. In [67] it is noted that the experiences with practical use of OE-graphs are rather limited. Moreover, in the examples of OE-graphs given in [67], the equivalence relations are defined using only the structure of the systems under consideration.

[1] Recall from Sect. 1.3 that the notation $M_1[b\rangle M_2$ means that the binding element b is enabled in M_1, and the occurrence of b in M_1 yields the marking M_2.

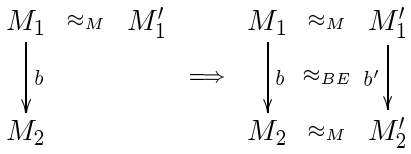

Figure 4.1: Consistency requirement for equivalence specifications.

4.2 Summary of Paper

The results presented in the paper came from our work with developing the OE/OS tool and doing various practical experiments with OE-graphs. The generality of OE-graphs allowed us to experiment with different kinds of user-supplied equivalence specifications. During these experiments we realised a new perspective on OE-graphs: OE-graphs allow equivalences that are dynamic, in the sense that they express that some information become irrelevant or similar as the execution of a system progresses. As noted above, then OS-graphs is a special case of OE-graphs. In particular, symmetry is a structural, static notion based on permutation of similar components. The main contribution of the paper is to recognise that sometimes, a more dynamic kind of equivalence is beneficial, and to demonstrate by means of an example that OE-graphs are applicable for this purpose.

The paper describes the application of OE-graphs and the OE/OS tool for verification of a protocol from the transport layer of the ISO reference model (see e.g., [29]). The protocol studied in the paper is in the following referred to as the *transport protocol*. The transport layer is concerned with protocols ensuring reliable transmission between sites. The transport protocol consists of a *sender*, which wants to transfer some data to a *receiver*. Communication takes place on an unreliable *network*, with risk of loss and overtaking. The transport protocol uses sequence numbers, acknowledgements, timeouts, and retransmissions for ensuring that the data packets are delivered once and only once and in the correct order to the receiver. The protocol deploys a stop-and-wait strategy, i.e., the same data packet is transmitted until an acknowledgement is received.

The *Message Sequence Chart* (MSC) in Fig. 4.2 illustrates the basic operation of the transport protocol. The MSC has columns representing the Sender, Network, and Receiver. The MSC visualises a possible execution of the transport protocol. The sender first sends a data packet with sequence number 1 (Data(1)) to the receiver which replies with an acknowledgement with sequence number 2 (Ack(2)) which is the data packet the receiver is expecting next. After reception of this acknowledgement, the sender then sends the data packet with sequence number 2. The MSC also illustrates a timeout which causes the data packet with sequence number 2 to be retransmitted.

The equivalence specification for the transport protocol is dynamic and is based on the observation that certain packets on the network may become similar as the protocol executes. As an example, consider Fig. 4.2 and the arrival of the retransmitted data packet with sequence number 2 at the receiver. The arrival of this data packet with a sequence number less than what the receiver expects next does not change the state of the receiver. Such a data packet on the network will be called *old*. Arrival of this old data

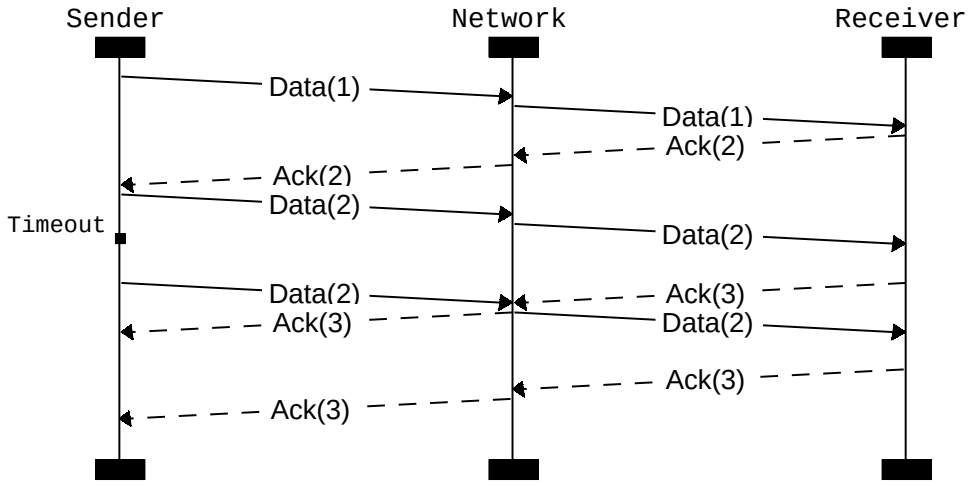

Figure 4.2: Basic operation of the transport protocol.

packet has the effect that an acknowledgement asking for data packet number three is sent. Generalising this, the arrival of any data packet with a sequence number less than the one expected has the effect that an acknowledgement is sent indicating which packet is actually expected. Thus, two old data packets arriving at the receiver have exactly the same effect. Similar observations and terminology apply to acknowledgements arriving at the sender, e.g, when the second acknowledgement with sequence number 3 arrives at the sender. The intuition behind the equivalence specification for the transport protocol is to capture that old data packets and old acknowledgements, respectively, are equivalent. This intuition is formalised in the paper, and it is proven that the equivalence specification is indeed consistent, i.e., it fulfils the requirement in Fig. 4.1.

Using the proof rules for OE-graphs in [67] and their implementation as query functions in the OE/OS tool, the following three crucial properties were established for the transport protocol. *No improper termination*, meaning that if the protocol terminates, all data packets have been received exactly once, in the same order as they were sent, and the network is empty. *Possibility of termination*, meaning that in any reachable state of the protocol, it is always possible within a finite number of steps to terminate the protocol. *Eventual termination* meaning that if the network is fair, i.e., looses only finitely many packets, then the protocol does eventually terminate.

The verification was done for different values of the system parameters, which are the capacity of the network and the number of data packets to be transmitted. Significant reduction in the number of nodes and arcs was demonstrated, making it possible to analyse configurations of the protocol that we could not handle using full state spaces. The experiments showed that the size of the OE-graphs grows linearly in the number of packets for a fixed capacity of the network. The experiments also showed that the time for construction of the OE-graphs and the time for determining the answers to the queries were reduced. The latter is caused by the fact that the

answers to the queries are determined directly on the OE-graphs without unfolding the equivalence classes.

The paper also establishes a general link between consistent equivalence specifications and *bisimulation* [94] (we will return to bisimulation in the next section). This result gives raise to a technique for supporting the user in ensuring the consistency of a proposed equivalence specification. The results states that the full state space and the corresponding OE-graph viewed as labelled transition systems are bisimilar modulo a relabelling of the arcs which maps each binding element into a unique representative for its equivalence class. This means that if the system under consideration is such that the full state space can be generated for small system parameters then an algorithm for checking bisimilarity can be applied as a necessary condition on the equivalence specification to be consistent.

4.3 Related Work

Practical Applications of OE-graphs. Verification by means of state spaces is often touted as being automatic and thus quite reliable. For verification based on OE-graphs, a qualification must be made: proving the consistency of a proposed equivalence specification may be a non-trivial task and is considerably more involved than proving the consistency of a proposed permutation symmetry specification for OS-graphs. For OS-graphs no ingenuity is required, and the proof can be highly computer-aided since it (in most cases) amounts to a more or less trivial case analysis of all static inscriptions of the CP-net. This suggests that the main potential of verification by means of OE-graphs is that it allows verification of larger configurations of the system under consideration. This is particularly important in systems which have several parameters. Due to state explosion it may only be possible to verify the system by means of full state spaces when almost all parameters have small values. By applying OE-graphs it is possible with some extra effort to verify configurations of the system where several of the parameters have large values.

Generality of the Equivalence Specification. Another question is, whether the kind of equivalence specification presented generalise, i.e., apply to other systems. We believe it does. We believe that the notion of being *old* can be found in various disguises in many systems – in particular protocols on the transport layer of the ISO reference model. Such protocols are often designed using standard techniques like timers, retransmissions, and sequence numbers, thereby inherently introducing the concept of old packets. Moreover, OE-graphs are described and defined for the formalism of CP-nets, but the idea generalises immediately to formalisms with an explicit representation of both states and actions of systems. The same is the case for the suggested equivalence specification based on the concept of being *old.*

The contribution of the paper was to recognise that sometimes, a more dynamic kind of equivalence is beneficial than can be provided by OS-graphs, and to demonstrate that OE-graphs are applicable for this purpose. OS-graphs fail to handle such cases since it is a structural, static notion of equivalence, based on permutation of similar components. Some work on adapting symmetries to a more dynamic setting has been presented in [55]. The results makes it possible to handle systems which have both symmetric and asymmetric parts. The CPN model of the transport protocol also

consists of symmetric and asymmetric parts: if we consider permutation of sequence numbers then the part modelling the network is symmetric since it does not look at the contents of the packets. The sender and the receiver part is asymmetric since they compare sequence numbers in an asymmetric manner. The results in [55] are, however, incapable of capturing that packets *become* symmetric as the protocol executes. The reason is that the approach in [55] is still based on permutations. As an example, if we have a set of old data packets: $\{\mathsf{Data(1)}, \mathsf{Data(2)}\}$ then this set can never be considered symmetric to $\{\mathsf{Data(1)}, \mathsf{Data(1)}\}$ since only permutations of the sequence numbers are allowed. This means that the equivalence specification based on old packets cannot be captured with the partial symmetries in [55].

Process-Algebraic Equivalence. The ideas behind OE-graphs are inspired and related to bisimulation as indicated by the link to bisimulation established in the paper. This means that OE-graphs are related to process-algebraic state space verification [57,94,128]. Process-algebraic state space verification is typically based around a set of compositional operators making it possible to compose systems in a hierarchical manner, complemented by a notion of equivalence between systems. The presence of the equivalence implies that for two equivalent systems S_1 and S_2, it does not matter whether S_1 or S_2 is used for the verification task – as long as the notion of equivalence used preserves the properties which are to be verified. In particular, a system can always be substituted with an equivalent smaller system (or sometimes even the smallest equivalent system). Process-algebras are typically action oriented, in contrast to Petri Nets which are both state and action oriented.

One of the strengths of process-algebraic verification methods is that they support a black-box view on systems. This is accomplished by the capability of declaring certain actions of the system invisible and then using a notion of equivalence which allows these invisible actions to be abstracted away. This means that it is possible to abstract the internal operation of a system away and view only the *externally observable behaviour* of the system. The externally observable behaviour is often what is of interest from a verification point of view. With CP-nets this is typically accomplished by manually constructing a more compact and abstract CPN model.

Process-Algebraic Compositionality. Strong and weak bisimulation [94] are two examples of equivalence between systems. To our knowledge the algorithms for reducing the system to an equivalent smaller one, such as those for bisimulation [40] rely on the entire state space of the system to be present before reduction can be applied. This means that process-algebraic equivalence is more geared towards *compositional state space verification* (see, e.g. [128] for a survey). The idea in compositional state space methods is to exploit that many systems are composed of smaller subsystems each with a state space significantly smaller than the state space of the full system.

As an example, suppose that we have an equivalence between systems (denoted $\cong$), an operator (denoted $\|$) for constructing the parallel composition of subsystems, and that we construct the system $S = S_1 \parallel S_2 \parallel \cdots \parallel S_n$ from the subsystems S_i. If each S_i is replaced with an equivalent system $\bar{S}_i$ (i.e., $\bar{S}_i \cong S_i$) and we consider the resulting system $\bar{S} = \bar{S}_1 \parallel \bar{S}_2 \parallel \cdots \parallel \bar{S}_n$, then $\bar{S} \cong S$ provided that $\cong$ is a *congruence* with respect to the composition operator $\|$. This means that it is possible to apply reduction to the state spaces of the components before composing them to obtain the

state space of the full system. The above approach can (of course) be applied in a hierarchical manner to systems with more than one intermediate level.

Compositional state space verification has the drawback that sometimes the intermediate state spaces, which have to be considered, are larger than the full state space of the system. Such situations occur since the context which the subsystems are put into may significantly restrict the behaviour of the subsystem due to synchronisation with its environment. Compositional state space generation may consider executions of the subsystems at the intermediate levels which are not possible in the complete system. At the extreme case an intermediate state space may even be infinite.

A vast amount of process-algebraic equivalences, compositional operators, and algorithms for manipulating them have been suggested in the literature. The best choice of equivalence for a concrete verification task depends on what properties of the system are to be verified. The equivalence should obviously preserve the properties which are to be verified, but on the other hand, it should also be as weak as possible to obtain a good reduction. The same is true with OE-graphs. As an example, the equivalence specification for the transport protocol does not consider binding elements corresponding to loss of packets and successful transmission to be equivalent. The reason for this is that we wanted to prove that if the network looses only finitely many packets then the protocol terminates. Had we considered them equivalent, then this property could not be established from the constructed OE-graph.

Compositional state space verification has been applied in [81] and [131] for verification of variants of the transport protocol. A sliding window variant was considered in [81], and a version based on alternating bits was considered in [131]. Both case studies establish similar properties as we do for the transport protocol. A major difference is, however, that the use of process-algebraic equivalence makes it possible to establish the properties for all values of certain system parameters. The results in [81] are valid independently of the capacity of the communication channel, and the results in [131] are valid independently of the number of retransmissions. Both case studies do use ingenuity to establish these results, but they are interesting since they show that it is sometimes possible to overcome one of the drawbacks of verification based on state spaces – that of having to fix the system parameters.

Compositionality and Petri Nets. The hierarchy concept of CP-nets is as such not suited for compositional state space verification. The constructs for putting modules together are purely a modelling convenience offering abstraction at the syntactical level rather than at the semantical level. To support compositional reasoning modifications to the basic definitions of CP-nets are needed, including the semantics for putting modules together. Such modifications for Place/Transitions Nets has been suggested in an event-oriented approach with parallel composition based on synchronisation of transitions in [122], and for a state-oriented asynchronous approach with parallel composition based on merging of place in [126]. Combining the ideas of process-algebra and Petri Nets has also been considered in [4].

Chapter 5

Stubborn Sets of Coloured Petri Nets

This chapter treats the paper *Finding Stubborn Sets of Coloured Petri Nets Without Unfolding* [86]. Section 5.1 contains an introduction to the stubborn set method which is the main subject of the paper. Section 5.2 contains a summary of the paper. Section 5.3 contains a discussion of related work.

5.1 Introduction and Background

Many concurrent and distributed systems are asynchronous and consist of a number of relatively independent processes that synchronise or communicate only occasionally. In the extreme (over simplified) case, consider a system consisting of n non-interacting processes each executing k actions sequentially before stopping. Since full state spaces represent all possible interleavings of the processes in a system, this system has a state space with $(k+1)^n$ states. This clearly shows that representing all possible interleavings of concurrent or independent actions is a major source of state explosion. Moreover, it seems intuitively clear that in order to reason about certain properties of this system not all of these states are a priori needed. As an example, if for the above system we are interested in verifying that a state is reachable in which all processes have terminated, then only one of the interleaved executions of the system is really needed, i.e., only $nk+1$ states are needed. The basic idea behind the stubborn set method is to obtain a reduced state space by avoid representing all possible interleavings of independent/concurrent actions. The stubborn set method has been developed in a series of papers (see [129] for a survey).

State space construction with the stubborn set method follows the same procedure as the construction of the full state space, with one exception. When processing a marking, a set of binding elements, the so-called *stubborn set*, is constructed. Only the enabled binding elements in the stubborn set are used to construct successor markings. The remaining binding elements are either taken into account in some subsequent marking, or the situation is such that they can be ignored altogether without affecting the analysis results. The fact that only a subset of the enabled binding elements is considered in each marking reduces the number of new markings, and this may lead to a significant reduction in the size of the state space.

The construction of the stubborn sets depends on two factors: dependencies between binding elements such as conflict (the occurrence of each binding element removes a certain token), and the properties that are to be checked of the system. In each marking encountered during the state space construction there are in general several

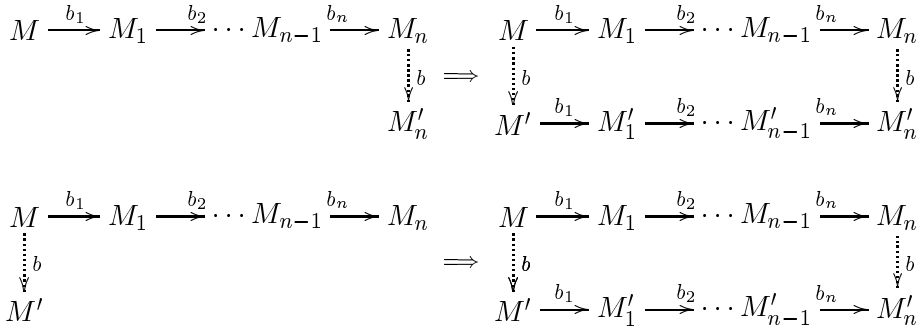

Figure 5.1: Properties of binding elements in stubborn sets.

sets of binding elements which satisfy the properties of a stubborn set. The art of stubborn sets is to choose the stubborn sets such that they contain enough binding elements to preserve the properties which are to be verified of the system, and at the same time choose them as small as possible to obtain a significant reduction.

A wide variety of stubborn set methods preserving different properties have been developed. We will discuss some of them in more detail in Chap. 6. It is, however, common to most of them that the stubborn sets should satisfy the two conditions depicted in Fig. 5.1. The binding elements $b_1, b_2, \ldots, b_n$ are binding elements outside the stubborn set in the marking M, whereas b (labelling the dashed arc) is a binding element in the stubborn set in M. The two figures should be read such that the existence of the occurrence sequence depicted on the left hand side of the double arrow, implies the existence of the occurrence sequences depicted on the right hand side of the double arrow. The first condition requires that it should be possible to reorganise occurrence sequences consisting of an initial prefix of binding elements $(b_1, \ldots, b_n)$ outside the stubborn set followed by a binding element (b) in the stubborn set, such that the reorganised occurrence sequence starts with the binding element in the stubborn set. The second condition requires that it should always be possible to move an enabled binding element in the stubborn set to the end of occurrence sequences consisting entirely of binding elements outside the stubborn set.

The conditions in Fig. 5.1 are not suited for constructing stubborn sets since they are semantical in nature in that they refer to occurrence sequences (which are not known at the time at which the stubborn set in M is to be constructed). This semantical formulation is suited for proving that certain properties are preserved by the stubborn set method, but for implementation some syntactical conditions are needed. Construction of stubborn sets is, therefore, in practice implemented by relying on rules that refer only to the structure of the CP-net and the marking for which a stubborn set is to be computed. The rules typically express sufficient conditions to make the conditions in Fig. 5.1 hold. Such rules can typically be thought of as spanning a *dependency graph*: the nodes of the graph are the binding elements, and there is an edge from a binding element b_1 to a binding element b_2 if and only if the rules demand that if b_1 is in the stubborn set, then also b_2 must be. A stubborn set then corresponds to a set of binding elements that contains an enabled binding element and which is closed under

reachability in the dependency graph. Therefore, to construct a stubborn set, it suffices to know the dependency graph and the set of enabled binding elements.

5.2 Summary of Paper

The stubborn set method has mostly been applied to CP-nets based on an *unfolding* of the CP-net to the *equivalent Place/Transition Net* [66, Sect. 2.4]. Since the equivalent Place/Transition Net has a transition for each binding element of the CP-net, and a place for each possible combination of place and colour in the colour set associated with the place (which might even be infinite), this approach is intractable in practice if implemented in a computer tool. The reason for doing this unfolding is that the dependency analysis needed by the stubborn set method for construction of the stubborn sets is difficult with CP-nets. This is because a CP-net transition may, e.g., simultaneously have a binding element that is concurrent and another binding element that is in conflict with a binding element of some other CP-net transition. An alternative stubborn set construction for CP-nets would be to treat each CP-net transition as a unit and consider a CP-net transition t_2 as dependent on another CP-net transition t_1, unless it is certain that no binding element of t_2 depends on any binding element of t_1. The reduction results obtained with this coarse strategy have usually been very bad. The motivation behind the work presented in the paper has been to try to find a better trade-off between a costly detailed dependency analysis at the level of the equivalent Place/Transition Net versus obtaining a reasonable reduction.

The first contribution of the paper is a lower bound result on the complexity of computing stubborn sets for CP-nets. This result shows that the time complexity of any algorithm which computes non-trivial stubborn sets (if such ones exists) in all markings encountered during state space construction is in worst-case at least proportional to the size of the equivalent Place/Transition Net. A non-trivial stubborn set is a stubborn set which does not contain all enabled binding elements. This result shows that if one wants to avoid making an unfolding (or doing something equally expensive) in worst-case, then some approximation is necessary in terms of not always computing the smallest possible stubborn sets, and hence one is forced to make a compromise with respect to the amount of reduction obtained.

The second contribution of the paper is to suggest an approximative stubborn set method for CP-nets which does not rely on unfolding to the equivalent Place/Transition Net. The underlying idea is to add to the CP-net some structure, which can be exploited during the stubborn set construction to avoid the unfolding, and which at the same time prevent the stubborn sets from becoming too big. The structure added consists of separating the parts of the CP-net which model control flow of processes, data manipulation, and communication. This is done by dividing the CP-net into one or more disjoint *process subnets*, such that each subnet corresponds either to a set of parallel processes executing the same code or to a variable through which two or more processes communicate (a fifo queue, for instance). These process subnets are then connected by different kinds of *border places* which are typically modelling asynchronous communication between processes, including communication between processes in the same process subnet. A CP-net with this structure added is called a *process-partitioned CP-net*. As an example, the CP-net for the commit protocol in Fig. 1.1 can be divided into

two process subnets: one process subnet corresponding to the loop on the left-hand side modelling the coordinator, and one process subnet corresponding to the loop on the right-hand side modelling the workers. The border places would consist of the four places in the middle, modelling the communication between the coordinator and the workers.

The key property of process-partitioned CP-nets is that they make it possible to lift the dependency analysis from the level of individual binding elements to the level of equivalence classes of binding elements. This in turn makes it possible to formulate rules which work at the level of sets of binding elements and which guarantee the properties depicted in Fig. 5.1. This means that dependency graphs can be defined for process-partitioned CP-nets such that the nodes now represent equivalence classes of binding elements instead of individual binding elements. As such the approach can be thought of as a kind of partial unfolding which facilitates approximation, and which at the same time does not fail to work when colour sets with an infinite domain are used as types (colours) of variables of transitions. The size of a dependency graph (number of nodes and arcs) for a process-partitioned CP-nets is bounded by $T \times |M_0|^2$, where T is the number of transitions of the CP-net, and $|M_0|$ is the number of tokens in the initial marking M_0 of the CP-net. This makes it possible to construct stubborn sets for process-partitioned CP-nets in $O(T \times |M_0|^2)$ worst-case time and space.

The stubborn sets computed with the algorithm presented in the paper correspond to the *basic stubborn set method* which means that all dead markings (markings without enabled binding elements) as well as the existence of an infinite occurrence sequence is preserved in the reduced state space.

The third contribution of the paper is to demonstrate the practical applicability of the suggested approximative stubborn set method with some experimental case studies, in which reduction of the state space as well as savings in time are obtained. A common denominator for the experiments was that the reduction obtained more than cancelled out the overhead involved in constructing the stubborn sets. Hence, judging from the experiments, the suggested method seems to give reasonably good stubborn sets in practice, at a very low cost with respect to time. This indicates that the method is a good compromise in the trade-off between not making too detailed an analysis of dependencies and at the same time getting a reasonable reduction.

5.3 Related Work

The stubborn set method is one of a group of rather similar methods also suggested under the names of *persistent sets*, *sleep sets* [49, 50, 136], and *ample sets* [101, 102]. All of these methods are based on the fact that the total effect of a set of concurrent actions is independent of the order in which the actions are executed. Therefore, it often suffices to investigate only one or some orderings in order to reason about the behaviour of the system. All of these methods contain similar ideas but differ with respect to the details of how the algorithms for state space reduction works. These reduction methods are also often referred to as *partial order reduction methods* [24, 103]. This name reflects that their underlying idea is that some actions of a concurrent system are dependent on each other (related), e.g., one must occur before the other, whereas other actions are independent (not related), e.g., they can occur in any order.

Many of these methods were not developed for Petri Nets. However, in the discussions below we will formulate their basic ideas in CP-net terminology.

Persistent and Ample Set Methods. State space construction with ample and persistent sets is similar to state space construction with stubborn sets in that the reduced state space is obtained by exploring only a subset of the enabled binding elements in each marking encountered. The stubborn set method was originally developed to preserve dead markings and the existence of an infinite occurrence sequence, whereas the ample set method was developed to preserve properties which can be expressed in the temporal logic LTL with the next-time operator omitted. This subset of LTL is also referred to as LTL_{-X}. The next-time operator is disallowed since it makes it possible to express properties which are sensitive to the ordering of independent actions. This has the effect that the properties which must be satisfied for a set of binding elements to qualify as an ample set is different compared to a basic stubborn set. Another difference is that an ample set consists only of enabled binding elements, whereas a stubborn set contains both enabled and disabled binding elements. It can however be proven that the set of enabled binding elements in a basic stubborn set constitutes an ample set provided that the additional conditions put on ample sets for preserving LTL_{-X} properties are not required. This implies that the approach we have developed for constructing stubborn sets of process-partitioned CP-nets can also be used to compute ample sets of CP-nets. Moreover, most algorithms for constructing ample sets are such that it is also possible to add disabled binding elements to the ample set and obtain a stubborn set. The persistent set method of [136] is similar to the ample set method and the set of enabled binding elements in a stubborn set constitute a persistent set as proven in [133].

The ample set method has primarily been implemented in the SPIN tool. The construction of ample sets in the SPIN tool [24, 59] is based on an approach similar to the one for constructing stubborn sets of process-partitioned CP-nets, in that similar structural properties of the system description are exploited to produce the ample set. However, the nature of the PROMELA language, which is the modelling language used in the SPIN tool, makes it simpler to distinguish processes, local variables, and communication channels than is the case with CP-nets.

The Sleep Set Method. State space construction with sleep sets follows a different pattern than state space construction with persistent, ample, and stubborn sets. The basic strategy is to compute, in each marking M encountered, a set of binding elements (called the sleep set). The sleep set contains the binding elements which will *not* be explored from M.

The application of the sleep method requires the identification of an independence relation between binding elements. Two binding elements are said to be *independent* if the property shown in Fig. 5.2 holds in all markings M. The requirement here is that two independent binding elements cannot disable each other. Binding elements which are not independent are said to be *dependent*. The sleep set in a given marking is calculated from the sleep set of its predecessor markings, the independence relation, and the order in which enabled binding elements are explored in the predecessor markings.

When used alone the sleep set method reduces only the number of arcs in the state space, whereas the number of nodes remains unchanged. This means that the reduced state space still contains all the reachable markings, and from a reduction point of

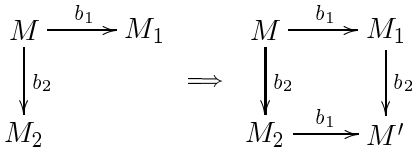

Figure 5.2: Independence between binding elements.

view nothing is really gained by using the sleep set method alone. The persistent set method and the sleep set method can be combined as shown in [136]. The combination has the potential of leading to better reduction results than when the persistent set method is used alone. The combination of the sleep set method and the stubborn set method has been investigated in [133]. In both cases it is shown that the combined reduction preserves the dead markings. It should be possible to adapt the approach for constructing stubborn sets for process-partitioned CP-nets in order to compute an approximation of the independence relation between binding elements required by the sleep set method. This would make it possible also to compute sleep sets of CP-nets without relying on unfolding to the equivalent Place/Transition Net. So far the combination between the persistent set method and the sleep set method has only been explored for the basic methods preserving dead markings. Theoretical considerations are needed to determine whether the sleep set method can be used, e.g., in combination with some of the more advanced stubborn set methods to which we will return in Chap. 6.

Modular Coloured Petri Nets. Alleviating state explosion by avoid representing all possible interleavings of independent actions has also been pursued in the context of CP-nets in [21] for so-called *Modular Coloured Petri Nets*. A modular CP-net is similar to a process-partitioned CP-net in that it consists of a number of subnets called *modules*, typically modelling the program executed by one or more processes. The main difference is that the communication between modules in [21] is by means of transition synchronisation, i.e., synchronous communication, whereas for process-partitioned CP-nets it is by means of places, i.e., asynchronous communication. From a modelling point of view it is less important whether communication is asynchronous or synchronous since in most cases synchronous (asynchronous) communication can be expressed by means of the asynchronous (synchronous) constructs, e.g., by adding an extra module in the synchronous case and by modelling a hand-shake in the asynchronous case.

The state space generation for modular and process-partitioned CP-nets is also different. Our approach relies on generating a single (reduced) state space for the entire system, whereas the approach for modular CP-nets is based on generating a *modular state space*. A modular state space consists of a state space for each module of the modular CP-net complemented by a so-called *synchronisation graph*. The role of the synchronisation graph is to ensure that the state spaces of the modules contain only those markings which are actually reachable in the full system. In effect, the use of this synchronisation graph avoids the problems previously discussed for compositional state space generation in Chap. 4, where the state spaces of the individual components may be infinite even though the state space of the full system is finite.

Another difference between the approach in [21] and our approach is that the approach in [21] does not attempt to reduce away interleaving between processes in same module. For example, in the commit protocol (see Fig. 1.1) the modular approach will only reduce away interleaving between the coordinator and the workers, but it would still represent all possible interleaved executions between the workers. This suggests that it might be worthwhile attempting to combine the two approaches since in practice when modelling with CP-nets (and Modular CP-nets), it is very often the case that one module/process subnet models a set of identical but relatively independent processes. Currently, there exists no tool support for modular CP-nets, and hence the method has only been tested manually on a few smaller examples.

The results in [21] contain proof rules which show how to derive reachability properties, boundedness properties, dead markings and dead binding elements (binding elements which never become enabled) directly from the modular state space. It also gives proof rules expressing necessary conditions for home and liveness properties. This means that the properties which can be verified from the modular state space are restricted compared to those which can be verified with the more advanced stubborn set methods to which we will return in Chap. 6.

Stubborn Sets for High-level Petri Nets. Alleviating the impact of unfolding when constructing stubborn sets for high-level Petri Nets has also been investigated in [7] for Well-Formed Petri Nets. The approach taken there is somewhat different from our approach in that it is based on solving constraint systems. The basic idea is to compile a constraint system from the Well-Formed Petri Net prior to state space generation. This constraint system expresses the dependencies between binding elements needed by the stubborn set method, and it is repeatedly solved during the state space generation to find the stubborn sets. The size (number of equations) of this constraint system is polynomially bounded by the size (number of transition and places) of the Well-Formed Petri Net in question. However, when solving the constraint system during state space generation, the approach still works at the level of individual colours and binding elements, i.e., at the level of unfolding. The fact that the arc expressions of Well-Formed Petri Nets are restricted compared to CP-nets allows these constraint systems to be computed and solved. The paper [7] does not analyse the running time of the suggested algorithm and no reports are given on practical experiments which could provide evidence about the practicality of computing the stubborn sets based on solving such constraint systems.

Chapter 6

Stubborn Sets for State Properties

This chapter treats the paper *Improved Question-Guided Stubborn Set Methods for State Properties* [87]. Section 6.1 contains a survey of some of the more advanced stubborn set methods in order to motivate the results of the paper. Section 6.2 contains a summary of the paper. Section 6.3 contains a discussion of related work.

6.1 Introduction and Background

The results presented in the previous chapter were concerned with finding stubborn sets of CP-nets satisfying the requirements of the *basic stubborn set method* [120]. The basic stubborn set method preserves all dead markings of the CP-net and the existence of an infinite occurrence sequence. More elaborate variants of the stubborn set method preserving additional properties have been developed in a series of papers. Below we give a brief survey of these advanced stubborn set methods in order to motivate the results presented in the paper. The general pattern of the advanced stubborn set methods is that more properties are preserved by putting additional requirements on the stubborn sets.

The *safety preserving stubborn set method* [124] consists of ensuring the conditions of the basic stubborn set method, and in addition, ensuring that each enabled binding element in a marking is eventually taken into the stubborn set in some subsequent marking of the reduced state space. This is also referred to as *elimination of ignoring*. This stubborn set method makes it possible to check liveness properties of transitions and binding elements. It also makes it possible to determine home spaces (a set of markings such that at least one of them can always be reached).

The *trace preserving stubborn set method* [124] can be used to preserve the language of a CP-net. The idea is to assign a label to each binding element. Binding elements assigned the empty label are called *invisible*, the remaining ones are called *visible*. The language of the CP-net is the set of all sequences of labels determined by the finite occurrence sequences of the CP-net. The method is based on the safety preserving stubborn set method with the additional requirement that if the stubborn set contains an enabled visible binding element, then it contains all the visible binding elements. This method makes it possible to check properties such as "the binding element b_1 always occurs before the binding element b_2".

The *LTL preserving stubborn set method* [123] preserves the validity of formulas

which can be expressed in LTL without the next-time operator[1]. It relies on declaring as visible all those binding elements which can change the value of some atomic proposition in the LTL$_{-X}$ formula to be checked. The method is based on the trace preserving stubborn set method by requiring that if the stubborn set in a marking M contains an enabled visible binding element, then it contains all the visible binding elements. In addition to this, it is required that if M has an enabled invisible transition, then such a transition is in the stubborn set used, and for all infinite occurrence sequences of the reduced state space starting in M and all visible binding elements, there is at least one marking in the occurrence sequence in which the visible binding element is in the stubborn set used. This method makes it possible, among other things, to check the fairness properties of transitions and binding elements.

The *CTL preserving stubborn set method* [46] preserves the validity of formulas which can be expressed in CTL* without the next-time operator. This subset of CTL* is denoted CTL$^*_{-X}$. It is based on the trace preserving stubborn set method and relies, like the LTL$_{-X}$ preserving method, on declaring those binding elements visible which can change the truth value of some atomic proposition of the formula to be checked. The method requires that the stubborn sets used are such that they either contain only one invisible enabled binding element or all the enabled binding elements.

It follows from the above that virtually all standard dynamic properties of CP-nets can be preserved using different variants of the stubborn set method. It is, however, a common denominator that they are all very bad at handling reachability properties, e.g., determining whether a certain marking M^* of the CP-net is reachable from the initial marking. Moreover, none of the above methods gives an effective way of determining whether a certain marking is a *home marking* (a marking which can always be reached). The CTL and LTL preserving methods can, in principle, be used for reachability properties, but the formula to be checked will contain an atomic proposition for each pair of place and colour in its colour set which effectively means that all binding elements of the CP-net have to be declared visible. This implies that only a very limited reduction of the state space can be obtained. Reachability of a certain marking can also be transformed into the existence of a dead marking or the occurrence of certain binding elements after a transformation of the CP-net. These transformations make it possible to handle reachability with the basic stubborn set method or the safety preserving stubborn set method. This approach is, however, problematic in that the transformation induces dependencies between the binding elements in such a way that, in practice, all enabled binding elements will be in the stubborn set and no reduction is obtained.

An effective way of overcoming the problem with reachability properties was given in [109] based on the notion of *attractor sets*. The basic idea in the method is the following. Suppose that we want to establish the reachability of a marking M^*. Formulated in terms of a CP-net, an attractor set in a marking M is a set of binding elements with the property that if M^* is reachable from M then any occurrence sequence leading from M to M^* contains at least one of the binding elements in the attractor set. Figure 6.1 illustrates the property of attractor sets: any occurrence sequence leading from M to M^* is required to contain a binding element b_i (labelling the dashed arc) which is in the attractor set. The stubborn set method of [109] requires that the attractor

[1] Recall from Chap. 5 that this subset of LTL is denoted LTL$_{-X}$.

$$M \xrightarrow{b_1} M_1 \cdots \xrightarrow{b_2} M_{i-1} \overset{b_i}{\dashrightarrow} M_i \cdots M_{n-1} \xrightarrow{b_n} M_n = M^*$$

Figure 6.1: Principle of attractor sets.

set is always contained in the stubborn set used. This requirement ensures that if M^* is reachable from a marking M contained in the reduced state space, then one of the successors of M in the reduced state space is closer to M^* than M. In the following we will refer to the stubborn set method of [109] as the *attractor set method*.

6.2 Summary of Paper

The motivation behind the work presented in the paper has been to improve and extend the stubborn set method presented in [109]. The improvement is achieved by generalising the theory of [109] in such a way that the requirements to the stubborn sets can be relaxed. This allows one to compute better (smaller) stubborn sets which can potentially lead to better reduction results.

The first contribution of the paper is to present two new stubborn set methods which make it possible to reason about *state properties.* A state property is a property that refers to only one marking, i.e., a property whose truth value in a given marking can be determined by considering only that given marking. This means that a state property is essentially a mapping ϕ from the set of markings into Booleans. The state properties are composed of so-called *atomic state propositions*, the logical operators "$\wedge$" and "$\vee$", and parentheses "(" and ")". The atomic state propositions considered allow one to compare the marking of a place with a constant and compare the marking of two places using relational operators such as "$=$", "$\leq$", and "$\neq$".

The choice of concrete atomic state propositions is, however, flexible. As a matter of fact, the theory in the paper has been developed in such a way that the stubborn set methods are correct for any set of atomic state propositions which allow for proper definition of so-called *up sets* and *down sets*, and which are such that the resulting state properties which can be composed with them are growing boolean functions. An up set is a set of transitions chosen such that at least one transition in it has to occur in order to make the state property hold. Hence up sets are similar to attractor sets. A down set is a set of transitions chosen such that a transition in the down set has to occur in order to make the property not hold. The up and down sets play a central role in the formulation of the two stubborn set methods.

The first stubborn set method presented makes it possible to answer the following question: "Is it possible to reach a marking where a given state property ϕ holds?" The method is *question-guided*, i.e., it takes a state property as input and generates a reduced state space. This reduced state space will contain a marking where the state property holds if and only if there exists a reachable marking in which the state property holds. This method makes it possible to, e.g., determine whether a certain marking M^* is reachable using a state property which is true in a marking M if and only if M is equal to M^*. In the following we will refer to this method as the *reachability of state property preserving (RSPP) stubborn set method*.

The second stubborn set method is based upon the first stubborn set method and makes it possible to answer the question: "Is it possible from all reachable markings to reach a marking where a given state property ϕ holds?". Like the first stubborn set method this method is also question-guided. This method makes it possible to determine, e.g., whether a certain marking is a home marking. In the following we will refer to this method as the *home state property preserving (HSPP) stubborn set method*.

The second contribution of the paper is to present three different implementations of the two suggested stubborn set methods. The first implementation consists of always including the up set in the stubborn set used for RSPP method, and always including both the up set and the down set for the HSPP method. For the RSPP method this is exactly what the attractor set method in [109] does, and this shows that the RSPP method presented is a generalisation of the results in [109] in the sense that the attractor set method can be seen as an implementation of our more general method. The virtue of this implementation of the RSPP method is that it preserves a shortest path in the state space to a marking where the state property holds (if such a marking exists).

The two more powerful implementations are both based on generating the reduced state space in a depth-first order, and they have the potential of leading to better reduction results than the implementation sketched above. The first implementation is based on checking whether the requirements on the stubborn sets are satisfied whenever a cycle in the state space is detected, i.e., whenever a marking is encountered during state space generation which is already on the depth-first search stack. The stubborn sets are augmented on-the-fly with additional binding elements if the requirements are not satisfied.

The most powerful implementation is based on the simultaneous generation of the reduced state space and strongly connected components using TARJAN's algorithm. This implementation augments the stubborn sets such that they satisfy the requirements imposed whenever a terminal strongly connected component is found. The implementation exploits the fact that TARJAN's algorithm relies on depth-first traversal and that it finds the strongly connected components in a depth-first order.

The third contribution of the paper is to present some experimental results on the reduction obtained with the RSPP stubborn set method. These case studies showed that the RSPP stubborn set method is significantly better than the attractor set method when the state property does not hold in any reachable marking. When a reachable marking exists in which the state property does hold, then the method represents a good compromise to the trade-off between preserving shortest paths to a marking where the property holds, and considering potentially large state spaces. The experimental case studies also compare different heuristics for the implementation of the RSPP stubborn set method.

6.3 Related Work

Stubborn set methods are most conveniently developed and formulated at the level of state spaces, i.e., occurrence sequences. This has the advantage of separating the stubborn set method from the concrete syntax of the modelling language, and it also makes the proof of correctness simpler. Figure 5.1 which expresses the requirements

of the basic stubborn set method is an example of such requirements formulated at the level of occurrence sequences. Hence, when developing stubborn set methods, it is advantageous to separate the specification of *what* properties the stubborn set should satisfy from *how* this is implemented. The stubborn set method can then be transferred to other modelling formalisms by showing how one can compute stubborn sets which satisfy the semantical properties from the syntax of the concrete modelling language.

The two stubborn set methods in the paper are defined and formulated in the context of Place/Transition Nets. It can, however, be observed that in the paper we only directly refer to Place/Transition Nets when showing how to implement up and down sets. The suggested stubborn set methods can, therefore, be transferred to other modelling formalisms – provided that they allow for the implementation of sets of transitions satisfying the properties of up and down sets. Since CP-nets and Place/Transition Nets viewed at the level of state spaces are the same, this means that the RSPP and HSPP stubborn set methods can be transferred to the process-partitioned CP-nets discussed in Chap. 5 by showing how to compute up and down sets for such CP-nets.

The Attractor Set Method. The main difference between the RSPP method and the attractor set method of [109] is in how progress towards a marking where the state property holds (if such one exists) is ensured. The attractor set method ensures progress towards a marking where the property holds by requiring that the attractor set is *always* included in the stubborn set used. Our method ensures progress by requiring the weaker condition that *eventually* the up set has been taken into the stubborn sets used. In effect we have replaced the *always progress* condition of [109] with the weaker *eventual progress* condition, which has the potential of leading to better reduction results, and which contains the always progress condition as a special case.

The HSPP stubborn set method is more effective than the method for home properties suggested in [109] which is based on the safety property preserving stubborn set method. The reason is that it does not impose all the requirements of the safety property preserving stubborn set method. Another novelty of the method is that it makes it possible to check whether a certain transition (binding element) is live (i.e., can always be made enabled) more effectively than the safety preserving stubborn set method which preserves the liveness of all transitions (binding elements) simultaneously.

Relaxed Visibility. The main motivation behind the work in [109] (and thereby our work) was to find a better stubborn set method for checking reachability properties. The problem was that the CTL_{-X} and LTL_{-X} preserving stubborn set methods failed to do this effectively since checking a reachability property required making virtually all binding elements visible. Practical experiments have shown that the reduction decreases rapidly with the number of visible binding elements. An approach to alleviate this problem for the LTL_{-X} preserving stubborn set method has been given in [84]. It is based on the observation that for certain properties it is possible to let the number of visible binding elements dynamically decrease while the property is being checked. As demonstrated by some experimental case studies in [84], this approach can substantially alleviate the problems with visibility present in the original LTL_{-X} method.

The RSPP stubborn set method preserves the truth value of the LTL_{-X} formula $\Box\neg\phi$ ("ϕ never holds"). The visibility requirements for LTL_{-X} formulas of the form $\Box\neg\phi$ can, however, not be relaxed with the approach in [84]. The RSPP stubborn

set method considers, therefore, a property which is not handled effectively with the approach in [84].

Combining Symmetry and Stubborn Sets. An interesting aspect of the stubborn set method is the combined use with the symmetry method. The combination was first investigated in [125] showing that the combined reduction preserves the equivalence classes of markings containing the dead markings as well as an infinite occurrence sequence. The combined use of the stubborn set method and the symmetry method in general yield better reduction results than when each method is used alone. The reason is that the two methods are orthogonal in that they focuses on different aspects of the system. The stubborn set method focus on the interleaving of the processes, whereas the symmetry method focuses on the similarity of processes in the system. The combination of symmetry and stubborn set was also investigated in [114] with the additional aspect that BDDs were used for computation and storage of the state space. The results concerning the properties preserved by the combined use of symmetry and stubborn sets were later extended in [32] in the framework of ample sets. The paper [32] shows how the combined reduction can be made to preserve the properties which can be expressed in LTL_{-X} and CTL^{*}_{-X}.

It would be worthwhile to revisit two new question-guided stubborn set methods for state properties and investigate their combined use with the symmetry method also. If a state property ϕ under consideration is such that for any two symmetric markings M_1 and M_2 it is the case that $\phi(M_1) = \phi(M_2)$, i.e., the truth value of the state property is invariant under symmetry, then the symmetry method is known to preserve the reachability of a marking where the state property holds. Intuitively, one would also expect the combined reduction to preserve the reachability of a marking where ϕ holds.

The Unfolding Method. The RSPP stubborn set method preserves the truth value of the CTL temporal logic formula $EF\phi$ ("there exist a future in which ϕ holds"). The HSPP stubborn set method preserves the truth value of the CTL temporal logic formula $EFAG\neg\phi$ ("there exist a future in which ϕ cannot be made to hold"). A more general subset of CTL, allowing arbitrary combinations of the temporal operators EF and AG has been considered in [36]. The model checking algorithm of [36] is based on the so-called *unfolding method* [92] and is valid for 1-safe Petri Nets, i.e., Place/Transition Nets where each place contains at most one token. It is important to notice that the term *unfolding* in this context has nothing to do with the process of unfolding a CP-net to a Place/Transition Net.

The idea underlying the unfolding method is the same as for the stubborn set method – avoid representing all interleaved executions of the system. However, the two methods address this in fundamentally different ways. The stubborn set method constructs a subset of the full state space, whereas the unfolding method constructs a different structure – the so-called unfolding. The unfolding is constructed by essentially viewing the Petri Net as a directed graph and performing a graph unwinding. This unwinding results in a Petri Net with a certain simple structure – a so-called *occurrence net*. The unfolding is, therefore, not a state space-like structure, although it represents the same amount of information, i.e., all reachable markings. As such it can be considered a kind of intermediate representation between a Petri Net and its state space. The unfolding of a Place/Transition Net is not necessarily finite, but it was shown in [92] how to obtain a finite prefix of this unfolding which represents all

reachable markings. It was also shown in [92] how this finite prefix can be used to determine the dead markings. The prefix construction was later improved in [38] for 1-safe Petri Nets such that it is guaranteed that the size of the unfolding is bounded by the size of the state space.

From an application point of view, a main difference between the unfolding method and most stubborn set methods is that the unfolding method is not question-guided, i.e., the unfolding of the Petri Net is constructed once and for all, and it can then be used to answer all queries which can be expressed in, e.g., the given subset of CTL considered in [36]. Compared to the many different variants of the stubborn set method, the unfolding method has so far been developed for a rather restricted set of properties and has shown to be effective for a rather restricted class of Petri Nets which are mainly used for modelling hardware. One of the strengths of the unfolding method compared to the stubborn set method is that it avoids doing any kind of complicated analysis to figure out dependencies between binding elements. The variant of the unfolding method [92] can be applied to CP-nets with finite colour sets by simply applying the method on the equivalent Place/Transition Net. To our knowledge this approach has not yet been explored in practice, but it is likely that it will suffer from the same performance problems as when the stubborn set method is applied based on unfolding to the equivalent Place/Transition Net.

Chapter 7

Conclusions and Future Work

This chapter concludes on the work done and presents some directions for future work. Section 7.1 contains a brief summary of the main contributions of this thesis. Section 7.2 contains a brief survey of some projects in which other researchers and engineers have applied the tools and methods developed as part of this thesis. Section 7.3 presents some directions for future work.

7.1 Summary of Contributions

The research has focused on the development of theoretical foundations allowing novel reduction methods to be applied in the framework of CP-nets, and the continued development of computer tool support for full state spaces and existing reduction methods. The main contributions of the work done are summed up below.

- The development of the DESIGN/CPN OCCURRENCE GRAPH TOOL (OG tool) [14, 19]. This tool has been developed starting from an early stand-alone prototype of a tool supporting state spaces to being a fully integrated part of the DESIGN/CPN tool. The tool supports full state spaces for CP-nets and combines analysis and verification using state spaces with simulation and visual debugging of systems.

- The development of the DESIGN/CPN OE/OS GRAPH TOOL (OE/OS tool) [74,78]. The development of this tool has taken state spaces reduced by means of symmetry and equivalence from being theoretically promising reduction methods to methods which can be explored in practice. It has allowed some first practical case studies to be conducted.

- The verification of LAMPORT'S FAST MUTUAL EXCLUSION ALGORITHM [74] using state spaces with symmetries. The main contribution of this case study has been to verify Lamport's Algorithm and to demonstrate that a significant amount of space and time can be saved by exploiting symmetry. This case study also demonstrated the use and capabilities of the developed OE/OS tool.

- Verification of a transport protocol using state spaces with equivalence classes [75]. One of the important contributions of this case study has been to identify a new way in which to exploit the general notion of equivalence provided by such state spaces. It has been established that it can be used to capture the fact that

as the execution of a system progresses some components/information in the system become equivalent/symmetric. This kind of equivalence is likely to be present in many systems, in particular in the area of communication protocols.

- Stubborn sets for Coloured Petri Nets [86]. The main contributions of this work has been a theoretical result on the complexity of computing stubborn sets for CP-nets and the development of an approximative stubborn set method for CP-nets. This approximative method avoids an unfolding to the equivalent low-level Place/Transition Net. The results from practical experiments with a first prototype implementation have been quite promising.

- Stubborn sets for state properties [87]. The main contributions of this work have been the generalisation of an existing stubborn set method and the development of a novel stubborn set method for state properties. The potential of these methods is that they have more powerful implementations than existing methods which in turn leads to better reduction results. This potential has been demonstrated on some practical case studies by means of a prototype implementation.

7.2 Practical Applications

When the work for this thesis started there had been only very few applications and case studies on the use of state space methods for CP-nets. Here we briefly survey a number of larger projects where some of the tools and methods developed have been put into practical use by other researchers and engineers. The purpose is to give an impression of the kind and complexity of systems which can be handled with the current tools and methods.

Security Systems at DALCOTECH A/S. The paper [105] is based on a project in which CP-nets and the DESIGN/CPN tool were used for the specification, design, and implementation of software to be used in a new version of a security system at the Danish engineering company DALCOTECH A/S.

Simulation and later full state spaces were used to validate and verify the designed security system. Due to the state explosion problem the state space analysis had to be based on small configurations and reduced scenarios of the security system. For the configurations which could be handled with full state spaces, crucial properties of the system were verified such as reversibility (the system can always return to its initial state), and if a detector is triggered, an alarm is generated and sent to the alarm control centre. Despite the restriction to small configurations of the system, the state space analysis led to the identification of approximately 15 non-trivial errors in the design, some of which would very likely also have existed in the final implementation of the system. The state space analysis of the system applied the standard as well as the more elaborate query languages of the OG tool. The drawing and visualisation capabilities of the OG tool were used extensively for identification of errors. The state spaces had up to 150,000 nodes and 250,000 arcs.

Communication Gateways at AUSTRALIAN DEFENCE FORCES. The paper [42] is based on a project in which CP-nets and the DESIGN/CPN tool were used to spec-

ify and design gateways between Radio Networks and Broadband Integrated Services Digital Networks (B-ISDN) to be used in the Australian Defence Forces Tactical Packet Radio Network (TPRN). A TPRN consists of a number of mobile radio nodes, and the role of the gateway is to make it possible to use a B-ISDN infrastructure as a backbone for connecting different TPRNs.

Starting from an initial behavioural specification of the gateway the design was gradually refined. The OG tool was used to verify the basic behaviour of the gateway such as testing for deadlocks and the correct establishment of calls. The project also involved building an integration between the OG tool and the PROTEAN tool [6]. The purpose of this integration was to be able use the process-algebraic equivalence techniques supported by the PROTEAN tool in order to be able to check whether the design of the gateway conformed with the specification. This analysis revealed that first attempts to refine the specification did not meet the specification. State space analysis of the gateway only required quite small state spaces containing up to 365 nodes and 860 arcs.

Audio/Video Systems at BANG&OLUFSEN (B&O). The paper [17] is based on a project in which CP-nets and the DESIGN/CPN tool were used to validate vital parts of the B&O BeoLink system. The BeoLink makes it possible to distribute sound and vision throughout a home via a network.

The state space analysis part of the project focused on the lock management protocol of the BeoLink. This protocol is used to grant devices in the system exclusive access to various services in the system. More specifically, the state space analysis established that in the initialisation phase of the system, a single *key* is eventually generated. This key is used by the devices to obtain exclusive access to the services. An interesting aspect of the project was that state spaces were generated for a timed CPN model, i.e., a CPN model which in addition to specifying the logical behaviour of the system also specifies the time taken by different activities. The model was timed since timing is crucial in the communication protocols of the BeoLink system. Another interesting aspect was that the verification results were obtained using partial state spaces. The state spaces needed to verify the initialisation phase of the lock management protocol for four devices had up to 13,420 nodes and 41,962 arcs.

Mobile Phone Software at NOKIA RESEARCH CENTER. The paper [137] is based on a project in which CP-nets and the DESIGN/CPN tool were used for modelling and analysis of a new software architecture for a mobile phone family. The purpose of the modelling was to be able to analyse the time and space performance of the software system and for configuring different product family members. The purpose of applying state spaces was primarily to ensure the correctness of the models.

State space analysis was conducted on several sub-models and for different scenarios. The standard query language of the OG tool and the close integration with the DESIGN/CPN simulator was used to investigate the properties of the communication protocols including the interaction protocols for the call control system. It was, e.g., established that in a call collision case (in which a user makes a call at the same time as a call comes from the network) the protocols behave correctly in that they have no deadlocks and lead to termination in the desired state. Several non-trivial errors were detected showing that state space analysis is an effective way of debugging a model and a system. The state spaces had on the order of 5,000 nodes and 16,000 arcs.

Flowmeter Systems at DANFOSS A/S. The paper [89] is based on a project in which CP-nets and the DESIGN/CPN tool were used for the modelling and analysis of a flowmeter system. A modern flowmeter system consists of a number of communicating processes cooperating to make various measurements on, e.g., the flow of water through a pipe. The purpose of the project was to investigate the use of CP-nets for validating the communication protocols in the flowmeter system.

The state space analysis part of the project aimed at applying state spaces for verifying crucial properties of the flowmeter system. The state space analysis identified deadlocks and data consistency problems in the proposed communication protocols. The OE/OS tool was also applied in the project for verification using state spaces with symmetries of a modified design proposal which avoided the identified deadlock and data consistency problems. The symmetries were based on the observation that the processes in a flowmeter system behave in a similar way. The use of symmetries allowed configurations of the system approaching those found in typical installations to be verified. The state spaces with symmetries had up to 120,000 nodes and 500,000 arcs corresponding to full state spaces with up to 600,000 nodes and 2,000,000 arcs.

Interworking Traders at UNIVERSITY OF SOUTH AUSTRALIA. The paper [115] is based on a project in which CP-nets and the DESIGN/CPN tool were used to develop an approach for analysing Interworking Traders. Trading is an information infrastructure service which allows software to advertise or export a service or resource to a trusted party, known as a Trader. Trading is an important part of the realization of Open Object-based Distributed Systems and has been a topic of standardisation by the International Organisation for Standardisation (ISO), the International Electrotechnical Commission, and the International Telecommunication Union as part of their work on the Reference Model for Open Distributed Processing.

In the project a CPN model reflecting the specification of the Trading standard was constructed. The process of constructing the model led to the identification of several ambiguities in the proposed standard. These ambiguities were resolved and a number of improvements suggested. The state space analysis showed that the interworking traders operate correctly in a number of scenarios such as stand-alone trader, stand-alone trader with multiple concurrent requests, as well as more complex configurations involving multiple traders. A number of errors were identified which had not been revealed by simulations of the CPN model. The project applied full state spaces as well as state spaces with equivalence classes as supported by the OE/OS tool. The full state spaces had in the order of 5,000-10,000 nodes and 6,000-40,000 nodes. The state spaces with equivalence classes were typically a factor 2-3 smaller. The experimental results showed the same pattern with respect to time as observed in the two case studies on condensed state spaces conducted as part of this thesis – the generation of the condensed state space was faster than generation of the corresponding full state space.

Mechatronic Systems at PEUGEOT-CITROËN. The paper [97] is based on a project in which CP-nets and the DESIGN/CPN tool were used for the modelling and analysis of a so-called *mechatronic system*. Active suspension, automatic gear boxes, and engine control in modern cars are all examples of mechatronic systems which can be characterised as a hybrid system consisting of a continuous physical system, a discrete control system, and some actuators and sensors allowing the discrete control system to communicate with the physical system. A central activity in designing a mechatronic

system is dependability evaluation which is concerned with identifying sequences of events which can lead to so-called feared events or faults.

In the project a quite simple mechatronic system consisting of a pressure tank containing oil, a pump supplying oil to the tank, and a consumer of oil was considered. State spaces were used to perform qualitative dependability analysis of the system in the presence of re-configurations. The use of state spaces established some of the most important properties of the system including the identification of which sequences of events can lead to the feared events. The state space analysis conducted was heavily based on the structure of the strongly connected component graph of the full state space. The state space of the considered system was relatively small – 25 nodes and 59 arcs.

7.3 Future Work

As stated in Sect. 1.4, the overall goal and motivation for the work has been to advance the applicability of state space methods for CP-nets. Based on the contributions listed in Sect. 7.1 and the summary of projects given in Sect. 7.2, it is reasonable to conclude that this has been achieved. Altogether the toolbox of available state space methods has been extended both from a theoretical and from a practical point of view. Even so, there are still many interesting ideas and directions to pursue in the future. Below we discuss a number of ideas and directions for future work based upon the work done for this thesis.

State Space Storage. If one considers the sizes of state spaces which have been reported on in the case studies which were briefly surveyed in the previous section, then the sizes might not impress compared to the sizes which are reported elsewhere in the literature for other modelling languages and tools. There are several reasons for this.

One reason is that the storage of the state space in the developed tools are implemented in STANDARD ML which is not likely to be particularly space efficient both in terms of storing values but also due to garbage collection. The reasons for using STANDARD ML for the storage of the state space was that the simulator of DESIGN/CPN is implemented in STANDARD ML, and that the inference mechanism provided by the pattern matching facilities of the STANDARD ML language were needed for computing the set of binding elements enabled in a given marking – which is one of the core components of the state space tools. It may very well turn out to be advantageous to keep the computation of the enabled binding elements in STANDARD ML but implement the storage of the state space in another language such as C or C++ which are likely to be more space efficient. Moreover, CP-nets have a quite complex notion of state which means that a state/-marking of a CP-net in general takes up more space than a typical state found in many other modelling languages. The main reason for this is the rather rich set of data types inherited from the STANDARD ML language.

Experiments and comparison with other datastructures such as BDDs for storage of state spaces is an important topic for future work. In particular, it would be interesting to see whether they work well with the elaborate notion of state provided by CP-nets.

Another issue is that the size of a state space tells only one part of the truth – what matters is the complexity of the systems which can be handled (albeit complexity of

a system is also difficult to measure). CP-nets makes it possible to perform rather complex manipulations in a single step in the execution of the model, e.g., mapping through a list of elements and finding the elements which satisfy a certain predicate. Therefore, when considering a system of a certain complexity, a CPN model typically has fewer states than a model of the system in a formalism with less powerful primitives, where operations like the one described above have to be encoded as a sequence of steps. The projects surveyed in the previous section confirm this observation. Even though they are rather complex systems, they still have a moderate sized state space. The above also sometimes have the effect that some reduction methods such as the stubborn set method exhibit less reduction when applied to CP-nets than when applied to other modelling formalisms.

State Spaces for Timed CP-nets. The support in the OG tool for state space analysis of timed CP-nets is still rather rudimentary and has only been put into practical use in one larger project [17]. One of the main obstacles is that the time concept of CP-nets and the current definition of state spaces for timed CP-nets imply that the state spaces become infinite in practice. The reason for the state spaces becoming infinite is that the absolute notion of time in a timed CP-net is carried over into the state space and becomes part of the state/marking information. This means that quite often only partial results can be obtained for timed CP-nets. It would be interesting to revisit the theory of state spaces for timed CP-nets and develop a relative notion of time which would be more suited for state space analysis. One possible way to proceed is to use state spaces with equivalence classes to factor out the absolute notion of time. It seems intuitively clear that an equivalence specification can be developed, proved consistent for all timed CP-nets, and be supported fully automatically by a computer tool.

State Spaces with Symmetries. The practical experiments and case studies conducted with the current version of the OE/OS tool have indicated several areas where the support for the symmetry method can be improved. The main obstacles in using the OE/OS tool are the manual implementation of the symmetry predicates and the proof of consistency. The next generation of the OE/OS tool should support automatic compilation of the symmetry predicates directly from symmetry specifications, and it should support a (semi)automatic consistency check. The practical experiments with the current version of the OE/OS tool have given ideas on how this can be done, and have hence represented an important intermediate step towards more elaborate tool support for state spaces with symmetries. The case studies have also indicated that as the number of permutation symmetries becomes large, the generation time for state spaces with symmetries starts to become a problem. Taking advantage of algebraic theory and algorithms to speed up the generation is a subject which deserves to be explored as part of future work.

Stubborn Sets for CP-nets. The stubborn set method developed for CP-nets currently covers the basic stubborn set method. As part of future work it would be of interest to extend the results on finding stubborn sets for process-partitioned CP-nets beyond the basic stubborn set method, i.e., extend the approach to the different advanced stubborn set methods which preserve additional properties of the system. A suitable computer tool supporting the approach still has to be developed. In fact, the current state of the stubborn set method in the context of process-partitioned CP-nets

is much like the status of condensed state spaces when the work for this thesis started: the theory exists and some small experiments have been conducted, but a suitable computer tool is lacking. This prevents the method from being explored on some larger examples.

Stubborn Sets for State Properties. The stubborn set methods for state properties can be used to determine whether a marking is reachable in which a given state property holds, and whether a marking can always be reached in which a given state property holds. It would be interesting to investigate whether the concepts of up and down sets, which are central to the two stubborn set methods, can be used to develop new stubborn set methods for other kinds of properties.

The practical experiments with the state property preserving stubborn set methods are still rather limited. Only the simplest of the more powerful algorithms has been implemented, and the impact of different implementation heuristics still needs to be explored. Transferring the state property preserving stubborn set methods into the framework of process-partitioned CP-nets is also a subject of future work.

The combination of the state property preserving stubborn set methods and the symmetry and equivalence methods is another aspect which has potential for further development. The combination of partial order reduction methods and symmetry is something which until now has only been explored to a very limited extent in practice.

The combination of the two methods is also interesting from the perspective that for some systems, the reduction obtained with the approximative stubborn set method for process-partitioned CP-nets combined with the symmetry method is the same as the reduction obtained with the stubborn set method based on unfolding combined with the symmetry method. This means that sometimes the symmetry method is capable of accounting for the reduction which is lost by using the approximative stubborn set method.

Part II

Papers

Chapter 8

State Space Analysis of Hierarchical Coloured Petri Nets

The paper *State Space Analysis of Hierarchical Coloured Petri Nets* constituting this chapter has been published as a workshop paper [18] and as a book chapter [19].

[18] S. Christensen and L.M. Kristensen. State Space Analysis of Hierarchical Coloured Petri Nets. In: B. Farwer, D. Moldt and M-O. Stehr (Eds): Proceedings of Workshop on Petri Nets in System Engineering (PNSE'97) Modelling, Verification, and Validation, Hamburg, Germany, Publication No. 205, Universität Hamburg, Fachberich Informatik, pp. 32-43, 1997.

[19] S. Christensen and L.M. Kristensen. State Space Analysis of Hierarchical Coloured Petri Nets. In: W. v. d. Aalst, J.-M. Colom, F. Kordon, G. Kotsis, and D. Moldt. Petri Net Approaches for Modelling and Validation. LINCOM Studies in Computer Science, No. 1, 1999. To appear.

The content of this chapter is equal to the book chapter [19] except for minor typographical changes.

State Space Analysis of Hierarchical Coloured Petri Nets

S. Christensen* L. M. Kristensen*

Abstract

In this paper, we consider state space analysis of 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 datastructures supporting computation and storage of state spaces which exploit the hierarchical structure of the models.

Keywords: State Space Based Approaches, Efficient Model Checking, Tools, Coloured Petri Nets, Verification and Simulation, Validation.

8.1 Introduction

State space analysis is one of the main formal analysis methods of Petri Nets [98], 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 Coloured Petri Nets (CP-nets or CPN) [66]. 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, and fairness. One of the main drawbacks of state space analysis is the state explosion problem, which imposes limitations on the applicability of state spaces for large models. One of the contributions of this paper is to present a datastructure which exploits the locality of CP-nets and the hierarchical structuring concepts of CP-nets to

*Department of Computer Science, University of Aarhus, DK-8000 Aarhus C., DENMARK.
E-mail: {schristensen,lmkristensen}@daimi.au.dk.

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 [16, 99], 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 [67] 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 [17, 42], and embedded systems [105].

This paper is organised as follows: Section 8.2 gives an informal introduction to the hierarchical structuring concepts of CP-nets and it introduces the example which is used throughout the paper. Section 8.3 presents the query languages, the support for drawing and generation of state spaces and explains how this have been integrated with simulation. The algorithms and datastructures used for computation and storage of state spaces are considered in Sect. 8.4. Section 8.5 contains the conclusions and a discussion of related work. The reader is assumed to be familiar with the basic concepts of CP-nets [67].

8.2 Hierarchical Coloured Petri Nets

This section informally introduces the hierarchical constructs of CP-nets by building a CPN model of a packet-switch network. This model will be used as an example throughout this paper. The example is based on Sect. 5 in [113]. In order to keep the introduction brief, we only introduce the concepts which are used in the subsequent sections. For a complete description see [67].

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 site 1 wants to send three packets to site 2 and vice versa, and the buffer size allocated in the two internal nodes is three. Suppose further, that site 1 sends its three packets to node 1, and at the same time site 2 sends its three packets to node 2. Since node 1 cannot pass any packet to node 2 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 8.1 depicts the Network module, which is the most abstract CP-net description of the packet-switch network. The communication links of the network are

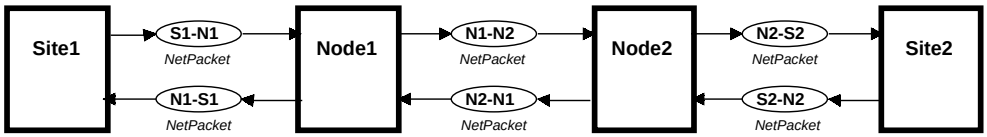

Figure 8.1: Network module of the packet-switch model.

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 accordingly named places. All six places in Fig. 8.1 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 sending network node to indicate whether the transmitted data packet was accepted or not. If the transmitted data packet was accepted, the sending node can delete the data packet from its 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. The two transitions Node1 and Node2 describe the behaviour of the two internal nodes in the network. The transitions in Fig. 8.1 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. 8.2

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. Place Received has the type Packet. A data packet is a pair consisting of a string (the contents) and an integer (denoting the number of hops in the network the data packet has travelled). The number of hops in a data packet is used by the controllers at the nodes to determine whether a data packet can be accepted. Place Send has the type PacketBuffer which denotes lists of Packets. The place Ready is used to model that only one data packet is sent at a time. The place Ready has the type E which contains a single element e. The actions of a site are represented by means of the two transitions SendPacket and ReceivePacket. SendPacket models the sending of packets and ReceivePacket models the receipt of packets.

The sub-module in Fig. 8.2 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. 8.1. For the substitution transition Site1, the places Incoming and Outgoing are bound to the places S1-N1 and N1-S1, respectively. Similarly, for S2-N2 and N2-S2 and Site2. This means that two different *instances* of the sub-module Site exist.

The two internal nodes of the network are modelled by the sub-module in Fig. 8.3. A node consists of two controllers represented by the two substitution transitions Controller1 and Controller2. The two controllers communicate using a shared variable modelled by the place Capacity. This variable indicates the amount of free buffer space at the node. A node has two controllers since a controller is only capable of handling traffic going in one direction. The interfaces places OutgoingLeft, IncomingLeft are in Node1 bound to the places S1-N1 and N1-S1 respectively. The interface places IncomingRight and OutgoingRight are in Node1 bound to the places N1-N2 and N2-N1.

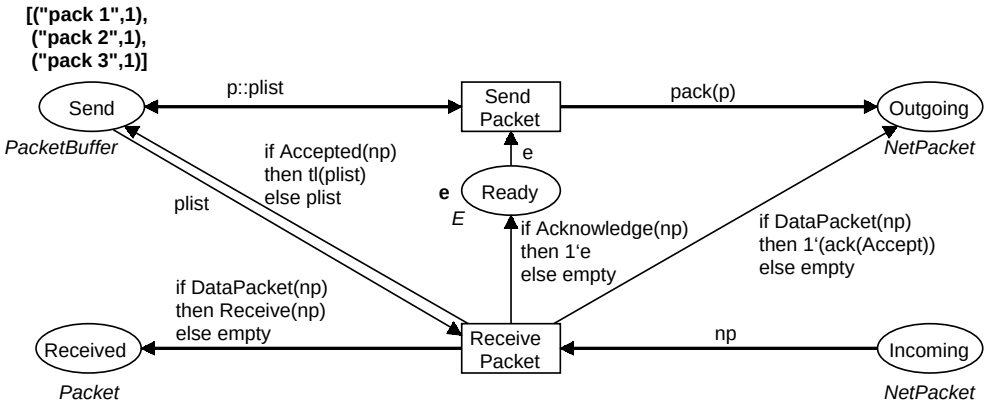

Figure 8.2: Site module for the packet-switch model.

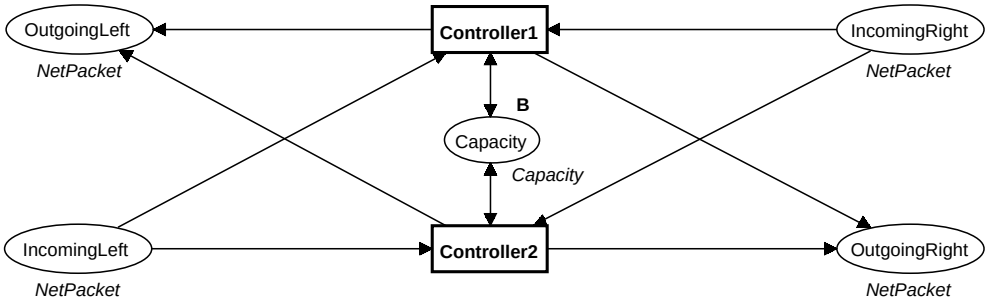

Figure 8.3: Node module for the packet-switch model.

Analogous for Node2.

The controller sub-module is shown in Fig. 8.4. The state of a controller is modelled by the places Capacity, Ready, and Buffer. The interface place Capacity is bound to the place Capacity in the Node module (see Fig. 8.3). It has the type Capacity denoting the set of integers. The initial marking of place Capacity is a token with colour B. B is a constant denoting the size of the buffers at the internal nodes. The place Buffer is used to hold the packets currently stored by the controller at the node. The place Ready is used to ensure that only one data packet is sent at a time to the next node.

The actions of the controller is modelled by the three transitions ReceivePacket, SendPacket, and ReceiveAck. ReceivePacket models reception of IncomingPackets. When a data packet is received, an acknowledgement is sent back using the place OutgoingAcks indicating whether the data packet could be accepted (Accept) or not (Reject). If the data packet was accepted, the free capacity of buffer is decremented. Transition SendPacket models the sending (forwarding) of a data packet to the next node. When a packet is sent, it is put on place OutgoingPackets, and at the same time the number of hops in the packet is incremented. Transition ReceiveAck models the reception of an acknowledgement for a data packet which has been sent. If the data packet was accepted by the next node, the packet is removed from the buffer and the free buffer capacity is incremented.

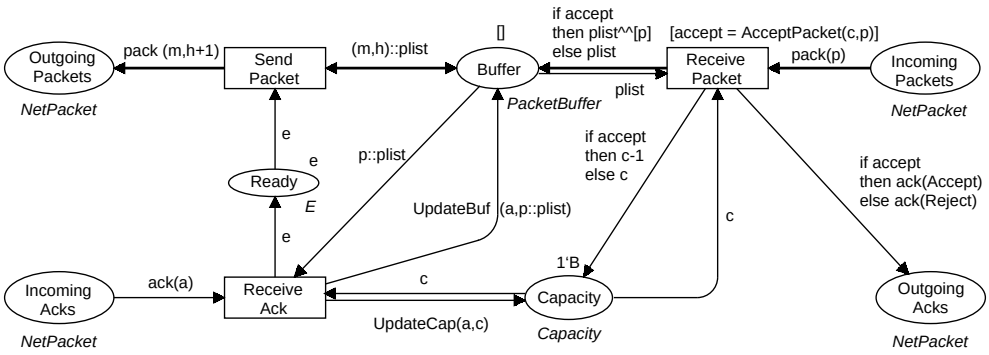

Figure 8.4: Controller module for the packet-switch model.

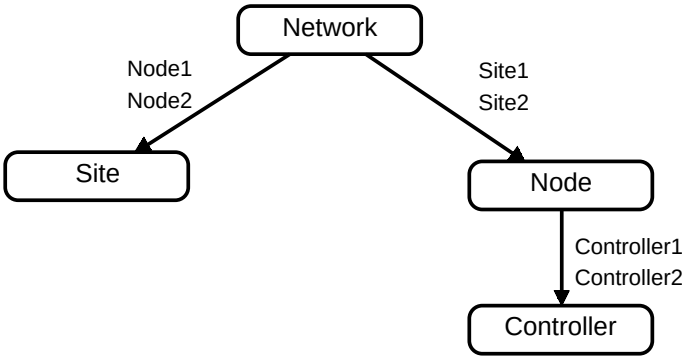

Figure 8.5: Hierarchy for the packet-switch model.

Figure 8.5 depicts the hierarchical structure of the packet-switch model. Each node in the figure represents a module. Node Network corresponds to the most abstract view of the model shown in Fig. 8.1. This node has two outgoing arcs – one leading to the Site module (see Fig. 8.2) and one leading to the Node module (see Fig. 8.3). The arcs are annotated with the name of the corresponding substitution transitions. The reuse of sub-modules means that we will have one instance of the Network module, two instances of the Site module, two instances of the Node module, and four instances of the Controller module.

8.3 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.

8.3.1 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 8.1. 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. 8.2) 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 Ultra Sparc Workstation with 512 MB RAM.

Packets	Nodes	Arcs	CPU Secs
1	516	1,672	2
2	7,144	29,164	38
3	27,100	119,754	192
4	59,748	270,530	506
5	104,716	479,866	1,102
6	162,004	747,762	1,996

Table 8.1: 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 is 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. 8.3.4. Interactive generation of the state space has many similarities with single step debugging as known from conventional programming environments.

8.3.2 Query Languages

The aim of generating a state space is to analyse the overall behaviour of the model and to investigate 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, generation time), 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 textual report. Negative answers to standard queries are constructive, i.e., they help the user investigate why an expected property does not hold. For example, 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 [95] 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 [11, 20] is provided. In this variant of CTL, it is possible to formulate queries about states as well as state changes. This corresponds to the fact that Petri Nets are both state and action oriented at the same time.

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 8.1, there was one dead state. Using a second standard query, it was verified that this dead state was also a home state.

8.3.3 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 diagram.

Transferring the simulator state into the state space is supported as well. A typical use of this is to investigate all 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.

8.3.4 Support for Drawing

Since state spaces often become large, it rarely makes sense to draw them in full. However, the result of queries is often 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 to get detailed information on a small number of nodes and arcs is to draw the necessary fragment of the state space. To illustrate this, 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. 8.6. Node 1 represents the initial state of the CP-net. Figure 8.6 is explained in more detail below.

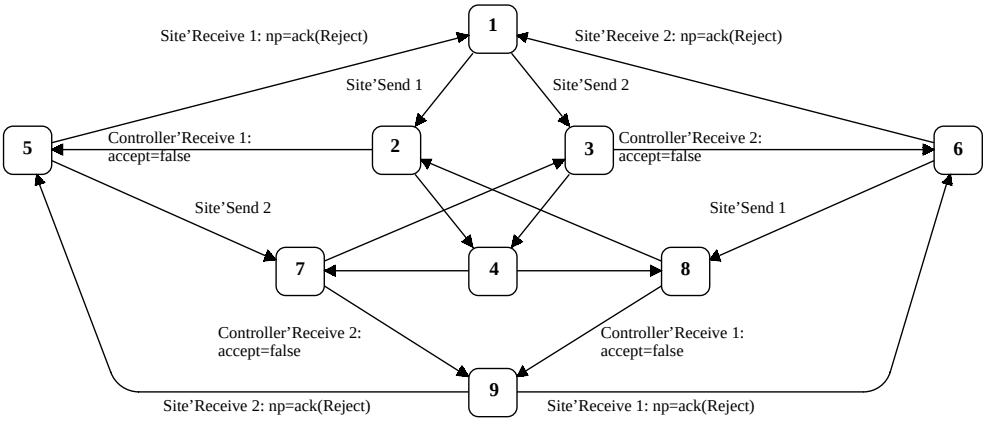

Figure 8.6: Drawing of the state space (3 data packets and buffer size 2).

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. 8.6, 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 site 1 and site 2 to node 1 and node 2, 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.

Parts of a state space can be drawn either manually in small steps, e.g., node by node or arc by arc, or automatically 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. 8.3.1 that this state space has 516 nodes and 1,672 arcs. We are interested in detailed information on a shortest path in the state space, leading from the initial state to the dead state. This can be done using the output 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. 8.7.

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

(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.

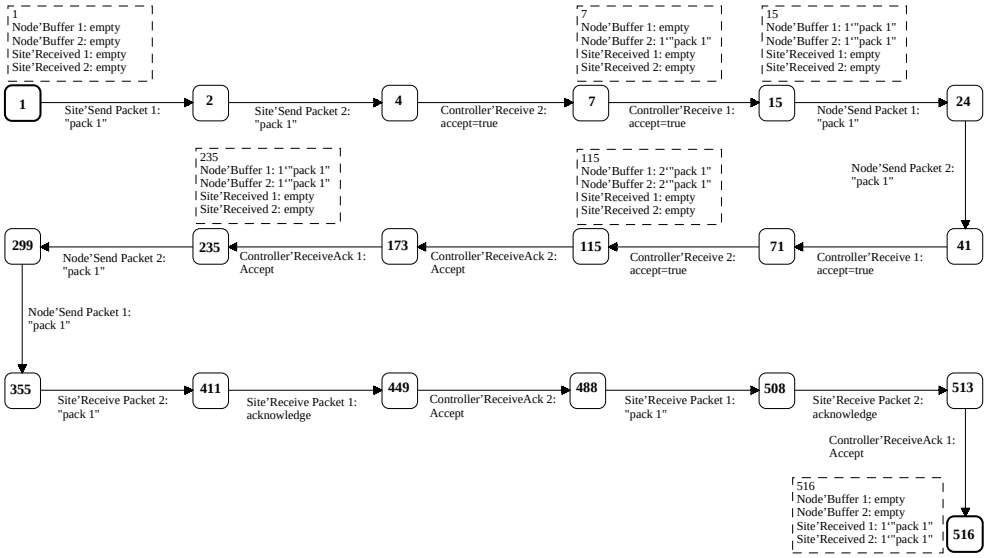

Figure 8.7: Execution path leading to the dead state.

8.4 Algorithms and Datastructures

The fundamental issues in the implementation of the state space tool are: the computation of the state space, the storage of the state space, and 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 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 [67] and [11].

8.4.1 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 a state is already represented in the state space. We will return to the latter issue in the following section.

8.4.2 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 unnecessary duplication of complex values. As an example of this, consider the packet-switch model. When the transition SendPacket (see Fig. 8.2) in the instance of the Site module corresponding to *Site1* occurs, only the markings of the places Outgoing and Ready in this module are affected. Because Outgoing is an interface place to the Network module (see Fig. 8.1), 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 importantly, the state of the instance of the Node and Site module corresponding to Node2 and Site2 is left unchanged.

Figure 8.8 depicts how states are stored, exemplified by the two modules of the packet-switch model shown in Figs 8.1 and 8.2.

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

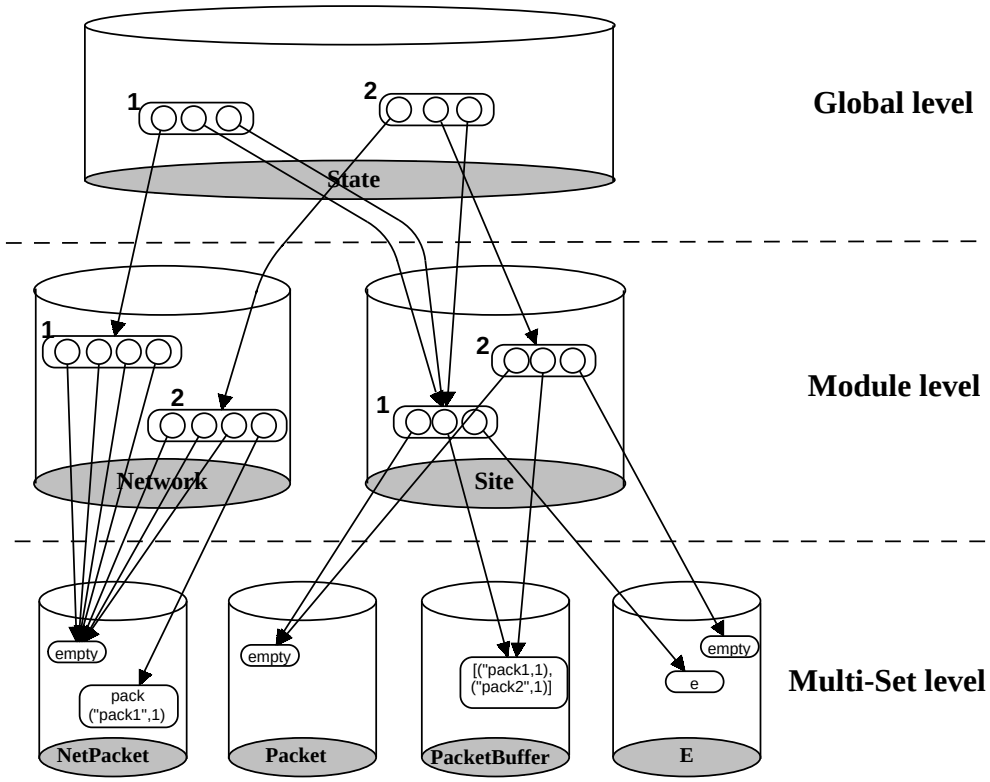

Figure 8.8: Representation of states.

states of the CP-net. Figure 8.8 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 SendPacket 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 of SendPacket 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 Tables 8.2 and 8.3. 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}

Table 8.2: Statistics for the Module level - 3 packets.

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 8.3: Statistics for the Module level - 6 packets.

The *Total* row shows that the saving in the number of MSs is significant. For example, 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 done, is compared to the number of multi-sets which is stored, when sharing is done. The figures are shown in Tables 8.4 and 8.5. Again, significant savings are obtained. For example, 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.

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}

Table 8.4: Statistics for the Multi-set level - 3 packets.

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 8.5: Statistics for the Multi-set level - 6 packets.

8.5 Conclusions

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 datastructure for a compact representation of state spaces of hierarchical CP-nets was presented and statistics given, demonstrating the applicability of the datastructure.

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 [130]. The approaches presented in this paper and in [130] are, however, different. In [130], 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 extent, the idea of avoiding duplication of complex values resembles BDDs as, e.g., used in the SMV system [93]. 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 avoids, however, 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 is no general cor-

relation 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 datastructure does not suffer from such "spurious" behaviour. However, the compression obtained by the suggested datastructure is sensitive to the hierarchical structure of the CP-net.

The use of BDDs in the context of Petri Nets has been considered in [100]. The results from [100], 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 inscription language 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 [61] for verification of asynchronous process systems. The datastructure used to represent the state space in the SPIN tool has some similarities with the datastructure 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 [26, 34, 65, 67] are present in many distributed systems and can be exploited to obtain a reduced state space. A prototype integrated in Design/CPN supporting this approach now exists [78].

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 [124, 136] attempt to avoid representing unnecessary 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 datastructure 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. For practical use of partial order reduction on CP-nets it is important to avoid relying on unfolding to the underlying Place/Transition net. Some results on the use of stubborn sets for CP-nets can be found in [86, 125].

Another approach to obtain a reduced state space is based on compositionality [128] and modularity [21]. 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.

Chapter 9

Computer Aided Verification of Lamport's Fast Mutual Exclusion Algorithm Using Coloured Petri Nets and Occurrence Graphs with Symmetries

The paper *Computer Aided Verification of Lamport's Fast Mutual Exclusion Algorithm Using Coloured Petri Nets and Occurrence Graphs with Symmetries* constituting this chapter has been published as a technical report [71] and as a journal paper [74].

[71] J. B. Jørgensen and L. M. Kristensen. Computer Aided Verification of Lamport's Fast Mutual Exclusion Algorithm Using Coloured Petri Nets and Occurrence Graphs with Symmetries. Technical report, Department of Computer Science, University of Aarhus, Denmark. February 1997. DAIMI PB-512.

[74] J. B. Jørgensen and L. M. Kristensen. Computer Aided Verification of Lamport's Fast Mutual Exclusion Algorithm Using Coloured Petri Nets Occurrence Graphs with Symmetries. IEEE Transactions on Parallel and Distributed Systems. Vol.10, No. 7, pp. 714-732, July 1999.

The content of this chapter is equal to the journal paper [74] except for minor typographical changes.

Computer Aided Verification of Lamport's Fast Mutual Exclusion Algorithm Using Coloured Petri Nets and Occurrence Graphs with Symmetries

J. B. Jørgensen* L. M. Kristensen†

Abstract

In this paper, we present a computer tool for verification of distributed systems. As an example, we establish the correctness of Lamport's Fast Mutual Exclusion Algorithm. The tool implements the method of occurrence graphs with symmetries (OS-graphs) for Coloured Petri Nets (CP-nets). The basic idea in the approach is to exploit the symmetries inherent in many distributed systems to construct a condensed state space. We demonstrate a significant increase in the number of states which can be analysed. The paper is to a large extent self-contained and does not assume any prior knowledge of CP-nets (or any other kinds of Petri Nets) or OS-graphs. CP-nets and OS-graphs are not our invention. Our contribution is the development of the tool and verification of the example, demonstrating how the method of occurrence graphs with symmetries can be put into practice.

Index Terms: Modelling and Analysis of Distributed Systems, Formal Verification, Coloured Petri Nets, High-Level Petri Nets, Occurrence Graphs, State Spaces, Symmetries, Mutual Exclusion.

9.1 Introduction

Coloured Petri Nets (CP-nets) [66] is a language for modelling and analysis of distributed systems. The ideas behind CP-nets build upon those of ordinary Petri Nets [98] and those of Predicate/Transition Nets [44]. CP-nets is at the same time theoretically well-founded and capable of modelling large distributed systems. A number of formal verification methods are available, by which the behaviour of a CP-net can be analysed. One of these methods is occurrence graphs (O-graphs) [67], also referred to as state spaces and reachability trees/graphs. The basic idea is to construct a directed graph with a node for each reachable state and an arc for each possible state change. An abundance of verification results can be derived from an O-graph. The method unfortunately suffers from the state explosion problem, which severely limits its practical usability. An approach to alleviate this problem is occurrence graphs with symmetries (OS-graphs) [67, 68], which are much more compact, but still enable us to obtain the

*Systematic Software Engineering A/S, DK-8230 Aabyhøj, DENMARK.
E-mail: jbj@systematic.dk.
†Department of Computer Science, University of Aarhus, DK-8000 Aarhus C., DENMARK.
E-mail: lmkristensen@daimi.au.dk.

same verification results as with O-graphs. Consequently, it is possible to investigate larger distributed systems, provided that they possess some kind of symmetry.

The applicability of OS-graphs is highly dependent on the existence of computer tools supporting the approach. Manual calculations of OS-graphs even for small systems are impossible. One contribution of this paper is to present a new computer tool supporting OS-graphs, and thereby developing the method from being theoretically promising to something which can be exploited in practice. Another contribution is the use of OS-graphs to establish the correctness of Lamport's Fast Mutual Exclusion Algorithm [88], in this paper referred to as *Lamport's Algorithm.*

Lamport's Algorithm is a mutual exclusion algorithm for shared-memory multiprocessors. A shared-memory multiprocessor is an architecture consisting of a number of CPUs connected to a common bus and with a single shared memory. It is assumed that the memory supports atomic read and write operations and that each process has a unique identifier, which is a positive integer. Figure 9.1 depicts the code that process i executes in Lamport's Algorithm, when attempting to enter the critical section. The algorithm uses three global variables: x and y which are integers, and an array $b[1..N]$ of booleans, where N is the number of processes. The statement await $cond$ represents a busy loop and can be seen as an abbreviation for while $\neg cond$ do $skip$. Angle brackets are used to enclose the atomic statements, which are the reads and writes of x, y, and the entries of b. In this paper, we will not explain how Lamport's Algorithm works, because it is not important for our purpose. The curious reader is encouraged to consult [88].

This paper is organised as follows. In Sect. 9.2, we present Coloured Petri Nets and create the model of Lamport's Algorithm to be used throughout the paper. In Sect. 9.3, we introduce OS-graphs, and in Sect. 9.4 we describe the tool supporting OS-graphs. In Sect. 9.5, we formulate correctness criteria for Lamport's Algorithm, and in Sect. 9.6, we report on the use of the tool for the actual verification. Finally, in Sect. 9.7, we draw some conclusions and discuss related and future work.

9.2 Coloured Petri Nets

In this section, we introduce *Coloured Petri Nets* (CP-nets or CPNs). As we go along with the explanation of the basic concepts, we show how these can be used to model Lamport's Algorithm. Section 9.2.1 provides an informal introduction to CP-nets. Section 9.2.2 contains the formal definitions and may be skipped by readers already familiar with CP-nets. The complete CPN model of Lamport's Algorithm can be seen in Fig. 9.2.

9.2.1 Informal Introduction to CP-nets

In contrast to many modelling languages, CP-nets is both state and action oriented. A state of a CP-net is represented by means of *places.* By convention, places are drawn as ellipses or circles with a name positioned inside. The basic idea in our CPN model is to describe the value of the program counters of the processes during the execution of Lamport's Algorithm. Therefore, Fig. 9.2 has a place for each line in Lamport's Algorithm. A place is named according to the statement in that line. As an example,

```
1:  start:
2:  ⟨ b[i] := true ⟩;
3:  ⟨ x := i ⟩;
4:  if ⟨ y ≠ 0 ⟩ then
5:                  ⟨ b[i] := false ⟩;
6:                  await ⟨ y = 0 ⟩;
7:                  goto start;
8:  end if;
9:  ⟨ y := i ⟩;
10: if ⟨ x ≠ i ⟩ then
11:                 ⟨ b[i] := false ⟩;
12:                 for j :=1 to N do
13:                                 await ⟨ not b[j] ⟩;
14:                 end for;
15:                 if ⟨ y ≠i ⟩ then
16:                                 await ⟨ y = 0 ⟩;
17:                                 goto start;
18:                 end if;
19: end if;
20:
21: {critical section}
22:
23: ⟨ y := 0 ⟩;
24: ⟨ b[i] := false ⟩;
```

Figure 9.1: Lamport's Algorithm.

the place setx_3 near the upper left corner of the drawing of the model corresponds to the program counter being in a position, where the statement $< x := i >$ in line 3 is ready to be executed.

The global variables are also modelled by means of places. We have an accordingly named place for each of the variables x, y, and b. All places modelling variables are circular and have a thick border in order to distinguish them from the places modelling the program counters. The shape and line thickness have no formal meaning. It should be noted that there are three places named y in Fig. 9.2. These are conceptually the same place, but have been drawn as three copies in order to reduce the number of crossing arcs and thereby improve the legibility of the CPN model. A similar remark applies to the four places named b.

Each place in a CP-net has a *colour set* (a type[1]), which determines the kind of data the place may contain. An element of a colour set is called a *colour*. By convention, the colour set is written in italics to the lower right of the place. From Fig. 9.2, it can be seen that the place b has the colour set PID×BOOL, and that the places x and y have the colour set PID_0N. The places wait and done have colour set PID×PID. All other

[1] An alternative and perhaps better name for Coloured Petri Nets might be "Typed Petri Nets". However, the term "coloured" has a historical explanation, and it has stuck. "Colour set" and "type" are used as synonyms in this paper.

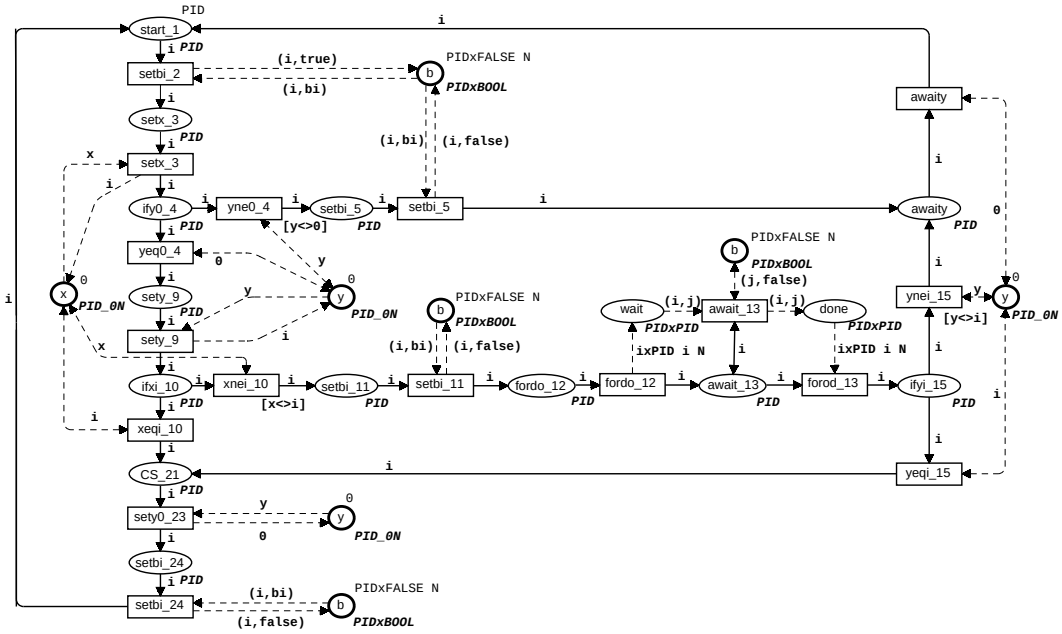

Figure 9.2: The CPN model of Lamport's Algorithm.

places have colour set PID. PID stands for Process IDentifier. The definition of the colour sets are as follows:

$$\begin{aligned} PID\_0N &= \{0, 1, \ldots, N\} \\ BOOL &= \{true, false\} \\ PID &= PID\_0N \setminus \{0\} \\ PID \times BOOL &= \{(i, bi) \mid i \in PID \wedge bi \in BOOL\} \\ PID \times PID &= \{(i, j) \mid i, j \in PID\} \end{aligned}$$

Thus, the place b can contain pairs consisting of an integer and a boolean. The places x and y can contain integers from 0 to N, and the places wait and done can contain pairs of integers from 1 to N. All other places can contain integers from 1 to N. The value 0 is special. It is used to signal when the values of the shared variables x and y do not correspond to any of the processes.

A state of a CP-net is called a *marking*. A marking describes how *tokens* are distributed on the individual places. A token is a value, which is a member of the colour set of the corresponding place. The initial marking of a place is specified in the CPN model, by convention, to the upper right of the place. The initial marking of the place start_1 is PID, i.e., the tokens from 1 to N. This models that to begin with, the program counters of all processes are positioned at the start label. For each of the places x, y, and b, the initial marking describes the start value of the corresponding variable. Both x and y are equal to 0 initially. The initial marking of the b-place is determined by the expression $PID \times FALSE\ N$, which evaluates to a set of tokens modelling that all entries $b[i]$ are $false$ for $1 \leq i \leq N$. Initially, all other places are empty,

Aside from having different tokens on a place, it is also possible to have several

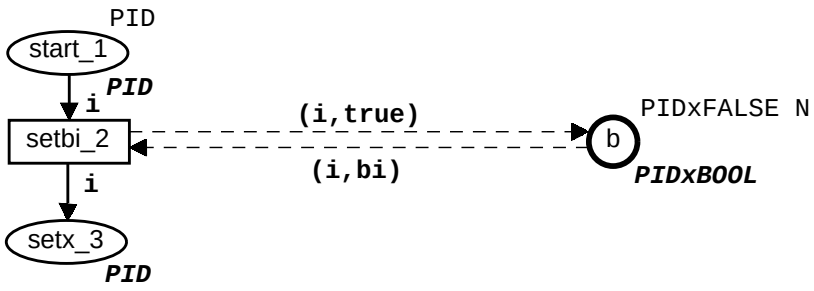

Figure 9.3: Modelling of an assignment.

tokens with the same colour. Therefore, the marking of a place is in general a *multi-set*[2]. A number of operations, such as addition and scalar-multiplication, are defined for multi-sets, and we will apply them freely is this paper. For details, see [66].

The actions of a CP-net are represented by *transitions*, which, by convention, are drawn as rectangles. Transitions and places are connected by *arcs*. In Fig. 9.2, solid arcs are used for control flow and dashed arcs are used for data manipulation. The graphical appearance of an arc has no formal meaning. The two kinds of arcs are only used to make a clearer presentation.

A transition removes tokens from the places connected to incoming arcs (input places) and adds tokens to the places connected to outgoing arcs (output places). The tokens to be removed from input places and added to output places are determined by the *arc expressions*, which are positioned next to the arcs.

In Lamport's Algorithm, the actions are execution of statements. Therefore, we have associated an accordingly named transition with each statement. For example, the transition setbi_2 (see Fig. 9.3) models the execution of the statement $b[i] := true$ in line 2 of Fig. 9.1.

The transition has two incoming arcs and two outgoing arcs. The arc expressions of the incoming arcs are i and (i,bi), where i and bi are variables of type PID and BOOL, respectively. To talk about an *occurrence* of the transition setbi_2, the variable i has to be bound to a value from PID, and bi has to be bound to a value from BOOL, in order to evaluate the arc expressions. A pair consisting of a transition and a binding of the variables of its surrounding arcs is called a *binding element*. A binding element may occur, iff the tokens to be removed exist on the respective input places.

Assume now that we bind the variable i to 1 and bi to $false$. Then, the expression on the incoming arc from start_1 will evaluate to 1, and the expression on the incoming arc from b will evaluate to $(1, false)$. Since in the initial marking, denoted M_0, a 1-token is on start_1, and a $(1, false)$-token is on b, the described binding element, denoted $(\text{setbi\_2}, \langle i = 1, bi = false \rangle)$, may occur. The binding element is said to be *enabled* in M_0. Several binding elements may be enabled in the same marking. For example, the binding element $(\text{setbi\_2}, \langle i = 2, bi = false \rangle)$ is also enabled in M_0. The two binding elements may occur in the same *step*, since in M_0, they do not share any of the tokens on the input places. The two binding elements are said to be

[2] A multi-set is often referred to as a bag. Sets can be considered a special kind of multi-sets, and therefore, in this paper, we sometimes use a set-like notation for multi-sets.

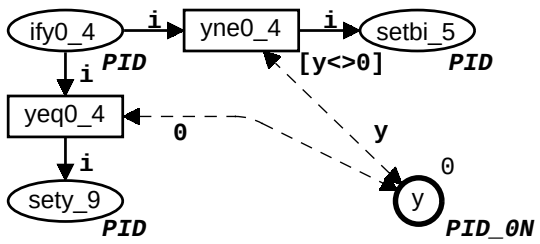

Figure 9.4: Modelling of an if-statement.

concurrently enabled. This corresponds to processes 1 and 2 being able to do this assignment independently of each other.

An occurrence of the binding element $(\text{setbi\_2}, \langle i = 1, bi = false \rangle)$ will remove the 1-token from start_1 and, similarly, remove the $(1, false)$-token from b. As determined by the arc expressions of the outgoing arcs, a 1-token will be added to setx_3, and a $(1, true)$-token will be added to b. An occurrence of this binding element corresponds to process 1 executing the statement $< b[i] := true >$ in line 2 of Fig. 9.1. In this way, an occurrence of a transition models the execution of an atomic statement in Lamport's Algorithm. All other assignments in Lamport's Algorithm are modelled in a similar fashion.

We will now describe how to model the other statements in Lamport's Algorithm, i.e., the if-, await-, for-, and goto-statements. Consider the if-statement starting in line 4 of Lamport's Algorithm. This statement is modelled by the part of the CPN model shown in Fig. 9.4.

The condition $y \neq 0$ evaluates to true or false, and depending on this, one of the two branches in Lamport's Algorithm is chosen. The case where the condition is false is modelled by the transition yeq0_4. It has two incoming arcs, one from the place ify0_4 and one from the place[3] y. The arc expression on the arc from y is 0 and will evaluate to 0, independent of the binding of the variable i, i.e., the process executing the if-statement. Thus, the transition will only be enabled when y contains a 0-token, corresponding to y being 0 in Lamport's Algorithm. When the transition occurs, it puts the 0-token back on y and puts an i-token on the place sety_9. The transition yne0_4 models the case in which the condition $y \neq 0$ is true. The transition has two incoming arcs with arc expressions i and y, respectively. Associated with the transition is also a *guard*. Guards are, by convention, put in brackets and located next to the lower right corner of the transition. A guard is a boolean expression, which imposes an additional condition on enabling. The variables must be bound so that the guard evaluates to true. In this case, the boolean expression is y<>0. The transition is, therefore, only enabled when y is not bound to 0. The two if-statements starting in lines 10 and 15 are modelled in a similar fashion.

We now turn to the modelling of the await-statement in line 6. The await-statement is modelled by the part of the CPN model shown in Fig. 9.5. The transition has an incoming arc from awaity and from y. The arc expression from y evaluates to 0 independent of the binding of i on the arc from awaity. Thus, the transition is only enabled

[3] A double arc is a shorthand for two arcs with the same arc expression, one arc in each direction.

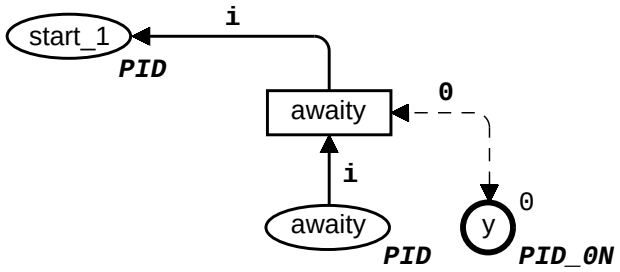

Figure 9.5: Modelling of an await-statement.

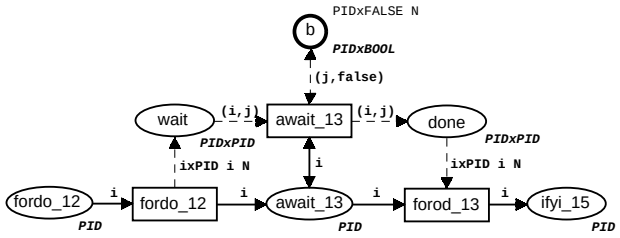

Figure 9.6: Modelling of a for-statement.

when y contains a 0-token, which corresponds to y being 0 in Lamport's Algorithm.

The goto-statements are modelled implicitly. Consider, e.g., the goto-statement immediately after the await-statement in line 7. In the model, we have drawn an arc from the transition modelling the execution of the await-statement to the place start_1.

Finally, we consider the for-statement starting in line 12. It is modelled by the part of the CPN model shown in Fig. 9.6. For reasons to become clear later (in Sect. 9.6), we model a more general form of the for-statement. In Lamport's Algorithm, the for-statement is used to test each of the entries in the b-array in turn starting from $b[1]$. In the model, we do not impose an order in which the entries are tested.

When process i enters the for-statement by occurrence of the transition fordo_12, the multi-set denoted $i \times PID\ i\ N = \{(i,j) \mid j \in PID\}$ is put on the place wait, which contains the entries in the b-array that process i still needs to test. The transition await_13 models the execution of the await-statement inside the for-statement, and is only enabled when a $(j, false)$-token is present on the b-place. An occurrence of that transition will remove an (i,j)-token from wait and add it to the place done, which contains the entries in the b-array that process i has already tested. Process i leaves the for-statement when the transition forod_13 occurs. As it can be seen, this transition is only enabled when place done contains the multi-set $i \times PID\ i\ N$, i.e., when all the entries in the b-array have been tested.

We have now explained how to model all the basic constructs of Lamport's Algorithm. The creation of the complete model just consists of putting all the pieces together. The process might even be automated. No ingenuity is required — nor desired. This systematic strategy reduces the probability of accidental errors, and thus makes it unlikely that the constructed CP-net is not a proper model of the algorithm.

Lamport's Algorithm is modelled in a similar way in [3].

9.2.2 Formal Definition of CP-nets

We now give a formal definition of CP-nets and their behaviour. The purpose of this section is twofold. First of all, to clear out any ambiguity that might be in the informal introduction to CP-nets in the previous section, and second, to fix the notation to be used in this paper. The definitions and notation closely follow [66], and readers familiar with that reference may skip this section.

Structure of CP-nets

Before giving the formal definition of a CP-net, we fix some notation and terminology. The term *net expressions* refers to the expressions describing colour sets, initial markings, arc expressions, and guards. Related to net expressions, we introduce the following notation:

- $Type(expr)$ denotes the type of an expression $expr$.
- $Var(expr)$ denotes the set of variables in an expression $expr$.
- $Type(v)$ denotes the type of a variable v.
- $Type(vars)$, where $vars$ is a set of variables, denotes the set of types $\{Type(v) \mid v \in vars\}$.
- S_{MS} denotes the set of multi-sets over a set S.
- $Bool$ denotes the set of booleans, i.e., $Bool = \{true, false\}$.

We now formally define CP-nets. Explanation follows the definition.

Definition 1 *A **CP-net** is a tuple $CPN = (\Sigma, P, T, A, N, C, G, E, I)$ satisfying the requirements below:*

1. *Σ is a finite set of non-empty types, called* colour sets.
2. *P is a finite set of* places.
3. *T is a finite set of* transitions.
4. *A is a finite set of* arcs *such that $P \cap T = P \cap A = T \cap A = \emptyset$.*
5. *N is a* node *function. It is defined from A into $P \times T \cup T \times P$.*
6. *C is a* colour *function. It is defined from P into Σ.*
7. *G is a* guard *function. It is defined from T into expressions such that:*
   $\forall t \in T : [Type(G(t)) = Bool \wedge Type(Var(G(t))) \subseteq \Sigma]$.

8. E *is an* arc expression *function. It is defined from* A *into expressions such that:*
   $\forall a \in A : [Type(E(a)) = C(p(a))_{MS} \wedge Type(Var(E(a))) \subseteq \Sigma]$,
   where $p(a)$ *is the place of* $N(a)$.

9. I *is an* initialisation *function. It is defined from* P *into expressions without free variables such that:* $\forall p \in P : [Type(I(p)) = C(p)_{MS}]$. □

Item 1 determines the set of colour sets and hence the colours which can be referred to in the net expressions. In the CPN model of Lamport's Algorithm we have the following set of colour sets: $\Sigma = \{PID\_0N, PID, BOOL, PID \times BOOL, PID \times PID\}$.

Items 2, 3, and 4 specify the places, transitions, and arcs. Item 5, the node function, determines the source and destination of arcs. Note that an arc always connects a place and a transition. Item 6, the colour function, associates a colour set with each place. In the CPN model of Lamport's Algorithm, the colour function maps the place b into $PID \times BOOL$, the places x and y into PID_0N, the places wait and done into $PID \times PID$, and all other places into PID. Item 7, the guard function, ensures that guards are expressions which evaluate to a boolean, and that the types of the variables in the guards are in Σ. Likewise, items 8 and 9, the arc expression function and the initialisation function, ensure similarly, appropriate type constraints. In the rest of this paper, we will assume that a CP-net CPN is given, $CPN = (\Sigma, P, T, A, N, C, G, E, I)$.

Normally, a CP-net is created in terms of a *CPN diagram*, i.e., a graphical representation as in Fig. 9.2, and not by specifying a 9-tuple as in Def. 1. Figure 9.2 is created using the Design/CPN tool [16, 99], which supports construction and analysis of CP-nets. For declarations of colour sets, variables, and functions; and for net expressions, this tool uses CPN ML, which is an extension of the functional programming language Standard ML (SML) (see, e.g., [119]). The declarations for the CPN model of Lamport's Algorithm can be seen in Fig. 9.7.

In line 2, the number of processes N is specified. In this case, $N = 3$. Lines 8-12 declare the colour sets. Lines 15-17 declare the variables and their type. Finally, the function $PID \times FALSE$, used to specify the initial marking on the place b, and the function $i \times PID$, used in the modelling of the for-statement, are declared. Both are typical SML-style recursive functions.

Behaviour of CP-nets

We now turn to the formal definition of behaviour of CP-nets. First, we fix some more notation.

- $Var(t)$, for a transition $t \in T$, denotes the set of variables of t present in either the guard $G(t)$ or in an arc expression of one of the surrounding arcs denoted $A(t)$. Formally:
  $Var(t) = \{v \mid v \in Var(G(t)) \vee \exists a \in A(t) : v \in Var(E(a))\}$.

- $A(x_1, x_2)$ for $(x_1, x_2) \in P \times T \cup T \times P$ denotes the set of connecting arcs. Formally:
  $A(x_1, x_2) = \{a \in A \mid N(a) = (x_1, x_2)\}$.

```
1:  (* Number of processes – in this case 3 *)
2:  val N = 3;
3:
4:  (* non-zero predicate *)
5:  fun nonzero i = (i<>0);
6:
7:  (* Declarations of the colour sets *)
8:  color PID_0N = int with 0..N declare ms;
9:  color BOOL = bool;
10: color PID = subset PID_0N by nonzero declare ms;
11: color PIDxPID = product PID * PID;
12: color PIDxBOOL = product PID * BOOL;
13:
14: (* Declaration of the variables *)
15: var x,y : PID_0N;
16: var i,j : PID;
17: var bi : BOOL;
18:
19: (* Function used to specify the initial marking on b *)
20: fun PIDxFALSE 0 = empty
21:   | PIDxFALSE i = 1‘(i,false) + (PIDxFALSE (i-1));
22:
23: (* Function used in the for-statement *)
24: fun ixPID i 0 = empty
25:   | ixPID i j = 1‘(i,j)+(ixPID i (j-1));
```

Figure 9.7: Declarations for the CPN model of Lamport's Algorithm.

As a consequence, if x_1 and x_2 are not connected, $A(x_1, x_2) = \emptyset$.

- $E(x_1, x_2)$ for $(x_1, x_2) \in P \times T \cup T \times P$ denotes the expression of (x_1, x_2). Formally:

  $E(x_1, x_2) = \sum_{a \in A(x_1, x_2)} E(a)$.

It should be noted that the sum in the definition of $E(x_1, x_2)$ is well-defined because of item 8 in Def. 1, which ensures that all terms in the sum are of the same multi-set type. Having fixed the notation, we define the concept of a binding. $expr\langle b \rangle$ denotes the result of evaluating an expression $expr$, whose variables are bound to values as determined by b.

Definition 2 *A **binding** of a transition $t \in T$ is a function b defined on $Var(t)$ such that:*

1. $\forall v \in Var(t) : b(v) \in Type(v)$.
2. $G(t)\langle b \rangle$.

B(t) denotes the set of all bindings for t. □

Item 1 ensures that only values of the correct type can be bound to a variable. Item 2 expresses that in order for b to be a binding of t, the guard must evaluate to true in b. In the following, bindings will be written on the form $\langle v_1 = c_1, v_2 = c_2, \ldots v_n = c_n \rangle$, when $Var(t) = \{v_1, v_2, \ldots v_n\}$. Now, we formally define markings, binding elements, and steps.

Definition 3 *A **marking** M is a function defined on P such that $M(p) \in C(p)_{MS}$ for all $p \in P$. The set of all markings is denoted $\mathbb{M}$. The initial marking is denoted M_0.*

*A **binding element** is a pair (t, b), where $t \in T$ and $b \in B(t)$. The set of all binding elements is denoted BE, while the set of binding elements for a specific transition $t \in T$ is denoted $BE(t)$.*

*A **step** is a non-empty and finite multi-set over BE. The set of all steps is denoted $\mathbb{Y}$.* □

By defining a step as a multi-set of binding elements, we allow multiple occurrences of a binding element in a given step. We now give the formal definition of enabling.

Definition 4 *A step $Y \in \mathbb{Y}$ is **enabled** in a marking $M \in \mathbb{M}$, iff the following property is satisfied:*

$$\forall p \in P : \sum_{(t,b) \in Y} E(p,t)\langle b \rangle \leq M(p).$$

$M[Y\rangle$ denotes that Y is enabled in M. □

The definition states that each binding element $(t, b) \in Y$ must be able to get the tokens specified by $E(p, t)\langle b \rangle$ — which is the multi-set of tokens removed from p, when t occurs with the binding b — without having to share these with other binding elements in Y. The summation is a multi-set sum, i.e., if (t, b) appears in Y multiple times, this multiplicity is taken into account in the sum. If a binding element for a transition t is included in an enabled step in a marking M, we will say that t is enabled in M.

When a step Y is enabled, it may occur. When Y occurs, it removes tokens from the input places and adds tokens to the output places of the included transitions, according to the following definition, which also introduces the concepts of occurrence sequences and reachability.

Definition 5 *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:*

$$\forall p \in P : M_2(p) = \Big(M_1(p) - \sum_{(t,b) \in Y} E(p,t)\langle b \rangle\Big) + \sum_{(t,b) \in Y} E(t,p)\langle b \rangle.$$

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

*A **finite occurrence sequence** is a sequence of markings and steps:*

$M_1[Y_1\rangle M_2[Y_2\rangle M_3 \ldots M_n[Y_n\rangle M_{n+1}$ *such that*[4] $n \in \mathbb{N}$ *and* $M_i[Y_i\rangle M_{i+1}$ *for* $i = 1, \ldots, n$.

[4] $\mathbb{N} = \{0, 1, 2, \ldots\}$ denotes the set of non-negative integers.

*Analogously, an **infinite occurrence sequence** is a sequence of markings and steps:*

$M_1[Y_1\rangle M_2[Y_2\rangle M_3 \ldots$ *such that* $M_i[Y_i\rangle M_{i+1}$ *for* $i = 1, 2, \ldots$.

*A marking M' is **reachable** from a marking M, iff there exists a sequence of steps $Y_1, Y_2, \ldots Y_n$ such that:*

$M[Y_1\rangle M_2[Y_2\rangle M_3 \ldots M_n[Y_n\rangle M'$.

The set of markings which are reachable from M is denoted $[M\rangle$. □

If a binding element for a transition t is included in a step Y, which occurs in a marking M, we will say that t occurs in M.

Quite often, the purpose of creating a CP-net is to investigate whether certain dynamic properties hold. An example of such a property is the existence of dead markings, corresponding to deadlocks of a considered system. In Sect. 9.5.2, we formally define a number of dynamic properties for CP-nets and use them to verify Lamport's Algorithm.

9.3 Occurrence Graphs with Symmetries

This section introduces the verification method of occurrence graphs with symmetries, which we are going to use to establish correctness of Lamport's Algorithm. The section is structured as follows. Section 9.3.1 briefly sums up the concept of full occurrence graphs (O-graphs). In Sect. 9.3.2, occurrence graphs with symmetries (OS-graphs) are described in an informal way. OS-graphs are formally defined in Sect. 9.3.3, which may be skipped by readers familiar with [67].

9.3.1 O-Graphs

One of the classical verification methods for CP-nets employs occurrence graphs. In its simplest form, an occurrence graph for a CP-net is a directed graph with a node for each reachable marking and an arc for each occurring binding element. This kind of graphs are called full occurrence graphs or *O-graphs*. Except for concurrency properties[5], all dynamic properties for a CP-net[6] can be derived from its O-graph — in particular, the properties to be used for the verification of Lamport's Algorithm.

As mentioned in Sect. 9.1, a serious drawback of the occurrence graph method is that it suffers from the state explosion problem: Even for relatively small CP-nets, the occurrence graphs are often so large that they cannot be constructed in practice given the computer technology presently available. Alleviation of this inherent complexity problem is a major challenge of research. Several theoretical methods have been proposed. Among them are OS-graphs. They are defined in [67]. The main ideas will be repeated here.

9.3.2 Informal Introduction to OS-graphs

Lamport's Algorithm treats all processes in the same way. The processes are symmetric in a sense to be illustrated in the following. In the CPN model for $N = 3$, consider

[5] When working with O-graphs, we only consider steps consisting of one single binding element.

[6] Only CP-nets with a finite number of reachable markings are considered.

the two markings M_1 and M_2 shown below. Multi-sets are written in the notation from [66]: As a sum using the symbol "+", where the number of appearances of each element is the coefficient preceeding the symbol ‘ (pronounced "back quote" or "of").

$$\begin{array}{lcl}
M_1(\mathsf{setx_3}) & = & 1‘1 \\
M_1(\mathsf{start_1}) & = & 1‘2 + 1‘3 \\
M_1(\mathsf{b}) & = & 1‘(1, true) + 1‘(2, false) + 1‘(3, false) \\
M_1(\mathsf{x}) & = & 1‘0 \\
M_1(\mathsf{y}) & = & 1‘0
\end{array}$$

$$\begin{array}{lcl}
M_2(\mathsf{setx_3}) & = & 1‘2 \\
M_2(\mathsf{start_1}) & = & 1‘1 + 1‘3 \\
M_2(\mathsf{b}) & = & 1‘(1, false) + 1‘(2, true) + 1‘(3, false) \\
M_2(\mathsf{x}) & = & 1‘0 \\
M_2(\mathsf{y}) & = & 1‘0
\end{array}$$

For all other places p, $M_1(p) = M_2(p) = empty$, where $empty$ denotes the empty multi-set. In both markings, all processes but one are on the place start_1. The remaining one is on the place setx_3. The two markings differ by which process is on setx_3. In M_k, the marking of setx_3 is k for $k = 1, 2$.

M_1 and M_2 are symmetric, in the sense that one can be obtained from the other by interchanging the colours 1 and 2. The crucial observation about symmetric markings is that they describe states of the system that are similar: If we know the possible behaviours of the system starting from M_1, then we do not need to explore the possible behaviours from M_2. An indication of this is to consider the set of binding elements BE_k, which are enabled in M_k, for $k = 1, 2$:

$$\begin{array}{lcl}
BE_1 & = & \{(\mathsf{setbi_2}, \langle i = 2, bi = false\rangle), \\
 & & (\mathsf{setbi_2}, \langle i = 3, bi = false\rangle), (\mathsf{setx_3}, \langle i = 1, x = 0\rangle)\}
\end{array}$$

$$\begin{array}{lcl}
BE_2 & = & \{(\mathsf{setbi_2}, \langle i = 1, bi = false\rangle), \\
 & & (\mathsf{setbi_2}, \langle i = 3, bi = false\rangle), (\mathsf{setx_3}, \langle i = 2, x = 0\rangle)\}.
\end{array}$$

BE_1 is symmetric to BE_2, i.e., BE_2 can be obtained from BE_1 by interchanging 1 and 2. Now, consider the marking $M_1^{'}$ reached when, e.g., the binding element $(\mathsf{setx\_3}, \langle i = 1, x = 0\rangle)$ occurs in M_1; and the marking $M_2^{'}$ reached when the binding element $(\mathsf{setx\_3}, \langle i = 2, x = 0\rangle)$ occurs in M_2. $M_1^{'}$ is identical to M_1, and $M_2^{'}$ is identical to M_2, except for the places listed below:

$$\begin{array}{lcl}
M_1^{'}(\mathsf{x}) & = & 1‘1 \\
M_1^{'}(\mathsf{setx_3}) & = & empty \\
M_1^{'}(\mathsf{ify0_4}) & = & 1‘1
\end{array}$$

$$\begin{array}{lcl}
M_2^{'}(\mathsf{x}) & = & 1‘2 \\
M_2^{'}(\mathsf{setx_3}) & = & empty \\
M_2^{'}(\mathsf{ify0_4}) & = & 1‘2.
\end{array}$$

It is easy to see that $M_1^{'}$ and $M_2^{'}$ are symmetric, i.e., that $M_1^{'}$ can be obtained from

M_2' by interchanging 1 and 2.

The property illustrated above is that symmetric markings have symmetric sets of enabled binding elements, and symmetric sets of directly reachable markings. Using induction, this property can be expanded to finite and infinite occurrence sequences. The CPN model of Lamport's Algorithm contains many markings that are symmetric in this way. The basic idea in OS-graphs is to lump together symmetric markings and symmetric binding elements.

The definition of an *OS-graph* for a CP-net requires the presence of two equivalence relations — one on the set of markings and one on the set of binding elements. The OS-graph has a node for each reachable equivalence class of markings[7]. The OS-graph has an arc between two nodes, iff there is a marking in the equivalence class of the source node in which a binding element is enabled, and whose occurrence leads to a marking in the equivalence class of the destination node. There is exactly one arc for each equivalence class of binding elements with this property. Typically an OS-graph is much smaller than the corresponding O-graph.

The two equivalence relations are induced by an algebraic group of functions called *permutation symmetries*. A permutation symmetry maps markings to markings and binding elements to binding elements. Two markings are *equivalent* (or *symmetric*), iff there exists a permutation symmetry mapping one of the markings to the other. Similarly for binding elements[8].

The user defines the group of permutation symmetries by writing a *permutation symmetry specification*. A permutation symmetry specification assigns a *symmetry group* to each *atomic* colour set appearing in the CP-net. A colour set defined without reference to other colour sets is atomic. In the CPN model of Lamport's Algorithm, there are two atomic colour sets: PID_0N and $BOOL$. A symmetry group determines how the colours of an atomic colour set are allowed to be permuted. For example, a symmetry group may specify that all colours can be permuted arbitrarily, or that they must all be fixed, i.e., cannot be changed. Many intermediate forms exists, e.g., all rotations of a finite, ordered colour set.

A permutation symmetry specification for the CPN model of Lamport's Algorithm capturing that processes corresponding to the integers in the set $\{1, ..., N\}$ behave in a symmetric way, and that the integer 0 is a special value used for initialisation purposes, can be described as follows: We assign the symmetry group to PID_0N, that allows arbitrary permutations in the set $\{1, .., N\}$, and insists that 0 is fixed. This symmetry group has $N!$ elements. $BOOL$ is assigned the singleton symmetry group consisting of the identity function id only. Thus, the values $true$ and $false$ cannot be swapped. They are (of course) fundamentally different.

A *structured* colour set is one, which is not atomic. The symmetry group for a structured colour set is inherited from the symmetry groups of its *base colour sets*, i.e., the colour sets that it is built from using the CPN ML type constructors. In the CPN model of Lamport's Algorithm, there are three structured colour sets: PID, $PID \times BOOL$, and $PID \times PID$ (see Fig. 9.7). PID inherits its symmetry group from its base colour set PID_0N (by the subset type constructor). An element of

[7] A reachable equivalence class is one which contains a reachable marking. As we shall see, for two equivalent markings, either both of them are reachable or none of them are reachable.

[8] A permutation symmetry can also be used to map colours to colours. We will speak about two colours being equivalent (or symmetric), iff there exists a permutation symmetry mapping one to the other.

the symmetry group for PID_0N induces a permutation on PID. Likewise, $PID \times BOOL$ inherits its symmetry group from the symmetry groups of PID and $BOOL$ (by the product type constructor). An element of the symmetry group of $PID \times BOOL$ is a pair, where the first element is a member of the symmetry group of PID, and the second element is a member of the symmetry group of $BOOL$. $PID \times PID$ inherits its symmetry group from the symmetry group of PID (by the product type constructor). An element of the symmetry group of $PID \times PID$ is a pair, where the first and the second element are identical members of the symmetry group of PID.

The purpose of a permutation symmetry specification is to capture inherent symmetries of the model. A permutation symmetry specification in accordance with the model, in a way to be defined precisely in Sect. 9.3.3, is said to be *consistent.* As we will see, the permutation symmetry specification described above for the CPN model of Lamport's Algorithm is consistent. But if we, e.g., assigned a symmetry group to PID_0N that allowed arbitrary permutations in the set $\{0, 1, \ldots, N\}$, and, hence, had not insisted that 0 should stay fixed, the resulting permutation symmetry specification would not be consistent. To see this, consider, e.g., the transition awaity in Fig. 9.5. A necessary requirement for this transition to be enabled, is that the place y contains a 0-token. Thus, if we allowed to swap 0 with another colour, we could obtain two symmetric markings, where awaity was enabled in one of them, but not in the other. These two marking would not contain the same information, and it would be wrong to consider them symmetric. Consequently, a consistency requirement is crucial.

9.3.3 Formal Definition of OS-graphs

In this section, we introduce the concepts necessary to formally define OS-graphs. All definitions and propositions are taken from [67] and are included here to make this paper self-contained. Readers familiar with [67] may skip this section. First the basics.

Definition 6 *A **permutation symmetry specification** is a function SG that maps each atomic colour set $S \in \Sigma$ into a subgroup $SG(S)$ of the set of permutations of S. $SG(S)$ is called the **symmetry group** of S.*

*A **permutation symmetry** for SG is a function ϕ that maps each atomic colour set $S \in \Sigma$ into a permutation $\phi_s \in SG(S)$. The set of all permutation symmetries for SG is denoted Φ_{SG}.* □

The permutation symmetry specification SG_L for the CPN model of Lamport's Algorithm, informally described in Sect. 9.3.2, is formally defined below. $PERM(I)$ is the set of all permutations of a finite set I.

$$SG_L(PID\_0N) = \{\phi \in PERM\{0, ..., N\} \mid \phi(0) = 0\}$$
$$SG_L(BOOL) = \{id\}.$$

An example of a permutation symmetry $\phi \in \Phi_{SG_L}$ is the following, where the function $(l\ k)_I$ swaps the values k and l in the set I:

$$\phi : PID\_0N \mapsto (1\ 2)_{\{0,\ldots,N\}}$$
$$\phi : BOOL \mapsto \{id\}.$$

ϕ induces the following mappings on the structured colour sets:

$\phi : PID \mapsto (1\ 2)_{\{1,\ldots,N\}}$
$\phi : PID \times BOOL \mapsto ((1\ 2)_{\{1,\ldots,N\}}, id)$
$\phi : PID \times PID \mapsto ((1\ 2)_{\{1,\ldots,N\}}, (1\ 2)_{\{1,\ldots,N\}})$.

As mentioned in Sect. 9.3.2, each permutation symmetry $\phi \in \Phi_{SG}$ induces a function which maps markings into markings. $\phi(M)$ is simply a substitution of each colour (value) $v \in S$, where S is some colour set, by $\phi(S)(v)$. A function mapping binding elements to binding elements is induced similarly. For example, consider the markings and binding elements used in the example from Sect. 9.3.2. $\phi \in \Phi_{SG_L}$ defined above maps M_1 to M_2. Moreover, ϕ maps the binding element $(\mathsf{setx\_3}, \langle i = 1, x = 0\rangle)$ to the binding element $(\mathsf{setx\_3}, \langle i = 2, x = 0\rangle)$, and ϕ also maps M_1' to M_2'.

Definition 7 formally defines consistency of a permutation symmetry specification. The transition of a given arc a is denoted $t(a)$.

Definition 7 *A permutation symmetry specification SG is **consistent**, iff the following properties are satisfied for all $\phi \in \Phi_{SG}$, all $t \in T$, and all $a \in A$:*

1. $\phi(M_0) = M_0$.
2. $\forall b \in B(t) : G(t)\langle\phi(b)\rangle = G(t)\langle b\rangle$.
3. $\forall b \in B(t(a)) : E(a)\langle\phi(b)\rangle = \phi(E(a)\langle b\rangle)$. □

Item 1 ensures that each permutation symmetry maps the initial marking to itself. Item 2 ensures that no transition has an asymmetric guard, i.e., a guard that treats two symmetric colours differently. Item 3 states that arc expressions and permutation symmetries must commute. Thus, asymmetric arc expressions are ruled out. It is important to notice that all three properties are local and structural. They can be checked without considering occurrence sequences.

When a consistent permutation symmetry specification is given, the important dynamic property proved in [67] and stated in the next proposition holds. It formalises that symmetric markings have symmetric sets of enabled binding elements, and symmetric sets of directly reachable markings, as illustrated in Sect. 9.3.2. Thus, the proposition justifies that it is sufficient to explore the possible behaviours of the system for one marking of each equivalence class.

Proposition 1 *A consistent permutation symmetry specification SG satisfies the following property:*

$\forall M_1, M_2 \in \mathbb{M}\ \forall b \in BE\ \forall \phi \in \Phi_{SG} : M_1[b\rangle M_2 \Leftrightarrow \phi(M_1)[\phi(b)\rangle\phi(M_2)$. □

We now formally define the two equivalence relations that are derived from the group of permutation symmetries, determined by a permutation symmetry specification SG.

Definition 8 *The relation* $\approx_{\mathbb{M}} \subseteq \mathbb{M} \times \mathbb{M}$ *is defined by:*

$$M \approx_{\mathbb{M}} M^* \Leftrightarrow \exists \phi \in \Phi_{SG} : M = \phi(M^*).$$

The relation $\approx_{BE} \subseteq BE \times BE$ *is defined by:*

$$b \approx_{BE} b^* \Leftrightarrow \exists \phi \in \Phi_{SG} : b = \phi(b^*). \quad \square$$

The fact that $\Phi_{SG} \in [\mathbb{M} \to \mathbb{M}]$ and $\Phi_{SG} \in [BE \to BE]$ both constitute algebraic groups ensures that the two relations $\approx_{\mathbb{M}}$ and $\approx_{BE}$ are indeed equivalence relations. The set of all equivalence classes for $\approx_{\mathbb{M}}$ is denoted $\mathbb{M}_{\approx}$. Similarly with $\approx_{BE}$ and $BE_{\approx}$. The equivalence class of an element x is denoted $[x]$. This notation is naturally extended to sets: $[X] = \bigcup_{x \in X} [x]$. Now OS-graphs are formally defined.

Definition 9 *Let a consistent permutation symmetry specification* SG *for* CPN *be given. The* ***OS-graph*** *is the directed graph* $OSG = (V, A, N)$ *where:*

1. $V = \{C \in \mathbb{M}_{\approx} \mid C \subseteq [M_0\rangle\}$.
2. $A = \{(C_1, B, C_2) \in V \times BE_{\approx} \times V \mid \exists (M_1, b, M_2) \in C_1 \times B \times C_2 : M_1[b\rangle M_2\}$.
3. $\forall a = (C_1, B, C_2) \in A : N(a) = (C_1, C_2)$. $\square$

Item 1 defines the set of nodes — one node for each reachable equivalence class of markings. Item 2 similarly defines the set of arcs. Item 3 is necessary, because we utilise a definition of directed graphs, which is slightly different from what normally appears in classical literature on graph theory. Apart from the set of nodes and the separately defined set of arcs, we have a function mapping each arc to a pair of nodes — the first component being the source and the second the destination. In this way, multiple arcs between two nodes are allowed, and this may appear in OS-graphs.

9.4 A Computer Tool Supporting OS-graphs

This section describes the *Design/CPN OS Graph Tool (OS-tool)* [78], which supports generation, analysis, and drawing of OS-graphs. The OS-tool is an integrated part of Design/CPN [16], the general tool for CP-nets mentioned in Sect. 9.2, which supports editing, simulation, and occurrence graph analysis of CP-nets. The existing support for O-graphs in Design/CPN (O-tool) [14] has served as a basis for the implementation of the OS-tool. Section 9.4.1 provides an overview of the OS-tool, while Sect. 9.4.2 uses the drawing facilities of the tool to compare O- and OS-graphs.

9.4.1 Overview of the OS-tool

Figure 9.8 gives an overview of the various parts of the OS-tool. The dashed boxes in the figure represent parts which are either modified or new compared to the O-tool. The other boxes correspond to parts which are identical to parts in the O-tool. The OS-tool consist of three major parts: a *Graphical User Interface (GUI)*, a *CPN ML* part, and an *Interface* between these two parts.

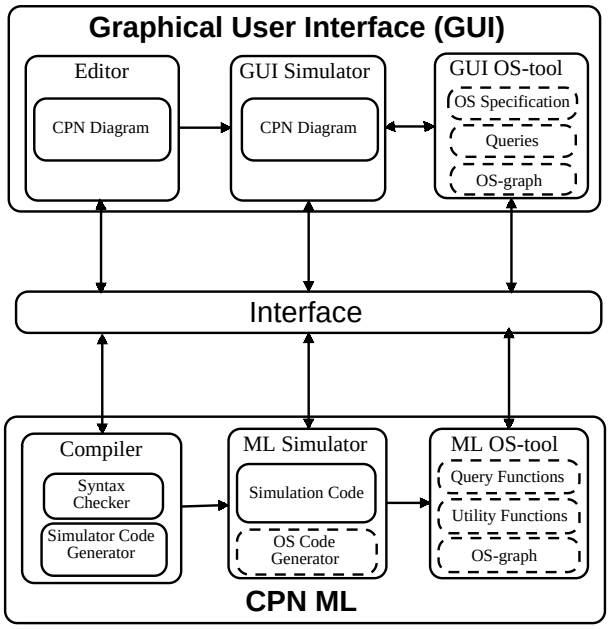

Figure 9.8: Architecture of the OS-tool.

The Graphical User Interface is the front-end of the application. When the user has created a *CPN Diagram* in the *Editor*, the *Compiler* in the *CPN ML* part can be invoked. The *Compiler* has two parts: First, the CPN diagram is syntax checked by the *Syntax Checker*. If the CPN diagram represents a legal CP-net, then the *Simulation Code Generator* is invoked to generate the *Simulation Code* for the *ML Simulator*. Once this code has been generated, the CPN model can be simulated — the user can examine markings and execute steps directly on the CPN Diagram in the *GUI Simulator*. In the *ML Simulator*, we have implemented an *OS Code Generator*. This code generator uses the *Simulation Code* and the user-written *OS Specification* (a permutation symmetry specification), provided through the *GUI OS-tool*, to generate the necessary code for the *ML OS-tool*. The *OS Specification* is written using the *Utility Functions*. When the code for the ML OS-tool has been generated, the user can start generate, and draw (parts of) an *OS-graph*, and make *Queries* using the *Query Functions* to investigate properties of the considered system.

The OS-tool stores equivalence classes using representatives: Each node in the OS-graph is represented by a marking from its equivalence class. Analogously for arcs and binding elements. Before an OS-graph can be generated, the user is required to implement a permutation symmetry specification. In the current version of the OS-tool, this consists of writing two CPN ML functions: A predicate $EquivMark$ defining when two markings are equivalent, and a predicate $EquivBE$ defining when two binding elements are equivalent. These two predicates must reflect the symmetry groups that the user has assigned to the atomic colour sets, and they must implement the rules saying how structured colour sets inherit their symmetry groups from their base colour sets. Moreover, the user must make sure that the predicates implement a consistent permutation symmetry specification. In the current version of the tool, this is not

```
1: Waiting := empty;
2: Node([M0]);
3: repeat
4:              Select(M1,[Waiting]);
5:              for all (b,M2) such that M1[b⟩M2 do
6:
7:                      if not (Represented(M2)) then
8:                              Node([M2]);
9:                      end if
10:                     if not (Represented([M1],[b],[M2])) then
11:                             Arc([M1],[b],[M2]);
12:                     end if
13:             end for
14:             Waiting := Waiting - {[M1]};
15: until Waiting = empty
```

Figure 9.9: Algorithm to generate an OS-graph.

checked automatically. In a future version, the user will only have to assign a symmetry group to each of the atomic colour sets. The tool will then automatically generate $EquivMark$ and $EquivBE$.

When the predicates $EquivMark$ and $EquivBE$ have been written, a predefined function that generates the OS-graph can be invoked. When the generation has finished, the user is ready to analyse the OS-graph to get information about the considered CP-net. The function that generates the OS-graph implements an algorithm from [67]. This algorithm is a natural modification of the algorithm to construct a normal state space, i.e., an O-graph: The test of equality before a new node is inserted, is replaced by a test for equivalence. Similarly, the algorithm to construct OS-graphs precedes insertion of an arc with a test for equivalence.

The algorithm is shown in Fig. 9.9. It uses a number of auxiliary functions: $Node/Arc$ creates a node/arc in the OS-graph for the given equivalence class, and $Node$ moreover adds its argument to the set $Waiting$ of unprocessed nodes. $Select$ picks a node from a given set. $Represented$ uses the predicates $EquivMark$ and $EquivBE$, provided by the user, to determine whether the equivalence class of the given node/arc is already in the OS-graph.

9.4.2 A First use of the OS-tool

In this section, we will illustrate the drawing facilities of the OS-tool. With respect to verification, drawing is of minor importance. Generation of the OS-graph followed by suitable queries is the way to verify systems. However, drawings are very adequate for presentation purposes. Here, we will use them to compare the O- and OS-graph for the CPN model of Lamport's Algorithm, for $N = 3$.

Part of the O-graph is shown in Fig. 9.10. To enhance readability, we have only shown some of the markings and some of the binding elements. Node 1 is the initial marking. The text placed right above the node describes the marking. Empty places are not listed. In the initial marking, three binding elements are enabled. They correspond

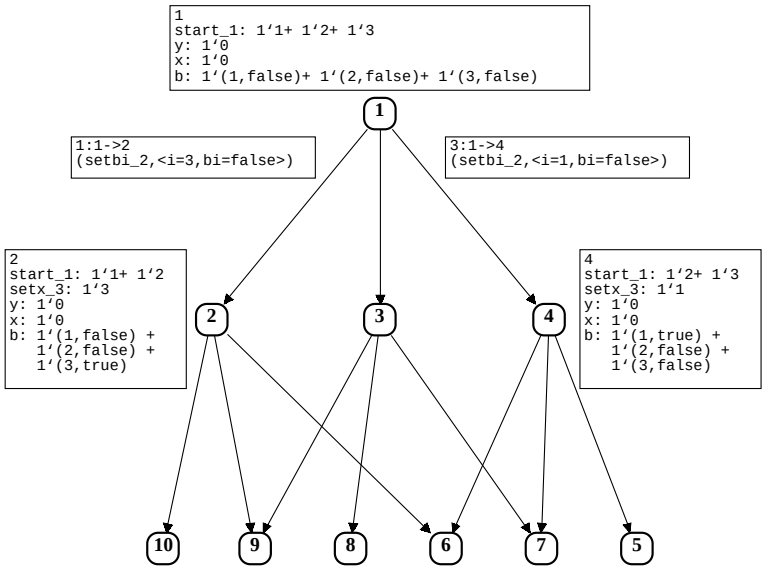

Figure 9.10: Part of O-graph for the CPN model of Lamport's Algorithm.

to the three output arcs from node 1. Consider the arc leading from node 1 to node 2. From the text placed next to this arc, it can be seen that an occurrence of the binding element $(\mathsf{setbi\_2}, \langle i = 3, bi = false \rangle)$, in the initial marking, leads to the marking of node 2. This marking is described by the text in the box positioned next to node 2.

When the permutation symmetry specification SG_L for the CPN model of Lamport's algorithm is implemented, the OS-graph can be generated and drawn. Part of it is shown in Fig. 9.11. As in Fig. 9.10, we have associated texts with the nodes and arcs, which describe the corresponding marking or binding element, chosen as representatives for the equivalence classes.

Let us compare in detail the partial O-graph in Fig. 9.10 with the partial OS-graph in Fig. 9.11. We will argue that they contain the same information, namely all occurrence sequences of the CPN model with at most two single steps. Consider node 1 in the OS-graph. This node represents the set of markings, which are equivalent to the initial marking. Because the permutation symmetry specification is consistent, we know from item 1 of Def. 7 that the size of this equivalence class is 1. Hence, node 1 in the OS-graph represents the equivalence class consisting exactly of node 1 in the O-graph. Nothing is saved yet.

Things improve, however, when we consider the immediate successors of node 1 in the two graphs. In the O-graph, node 1 has three successors; in the OS-graph, only one successor. This is because nodes 2, 3, and 4 in the O-graph are symmetric, i.e., belong to the same equivalence class. For example, node 2 can be mapped into node 4 by swapping the processes 1 and 3. The occurring binding elements, which lead from node 1 to the nodes 2, 3, and 4, are also symmetric, and therefore, the OS-graph has only one arc from node 1 to node 2.

In a similar fashion, nodes 5, 8, and 10 of the O-graph are symmetric. They are all markings in which two different processes have executed one statement each, and they are represented by node 4 in the OS-graph. The same goes for the nodes 6, 7, and

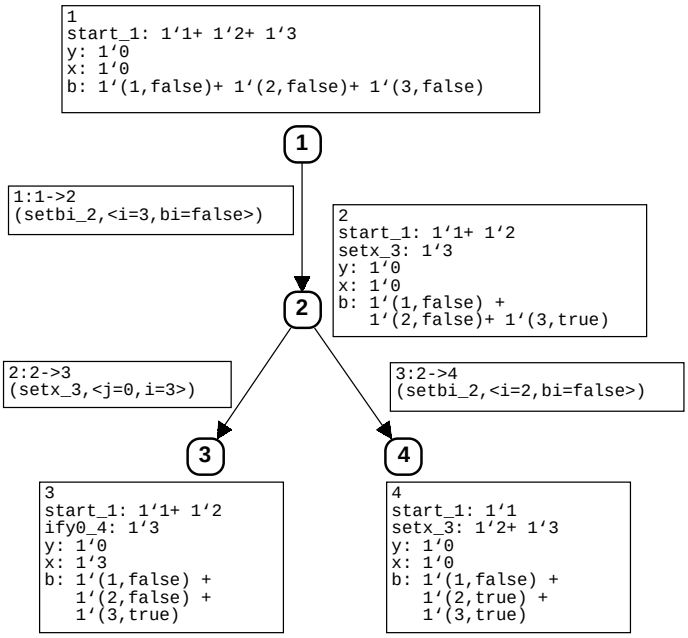

Figure 9.11: Part of OS-graph for the CPN model of Lamport's Algorithm.

9. They are all markings in which one process has executed two statements, and are represented by node 3 in the OS-graph.

9.5 Correctness of Lamport's Algorithm

In this section, we describe how to verify Lamport's Algorithm by means of OS-graphs. In Sect. 9.5.1, some properties expressing the correctness of Lamport's Algorithm are listed. In Sect. 9.5.2, these are translated into dynamic properties of the CPN model. Finally, in Sect. 9.5.3, we consider how to verify dynamic properties for CP-nets using OS-graphs.

9.5.1 Properties of Lamport's Algorithm

In [106], a number of properties that mutual exclusion algorithms must posses in order to be correct are discussed. These properties are 1 to 4 listed below:

1. *Mutual exclusion*: At any time, no more than one process is in the critical section.

2. *Persistent reachability of the critical section*: When several processes attempt to enter the critical section, eventually one will do so. It is not possible to have a situation in which all processes are starved.

3. *No deadlocks*: No execution of the mutual exclusion protocol can lead to a situation in which there is no activity among the processes, i.e., a situation in which all processes are blocked.

4. *Independence*: The behaviour of a process outside the mutual exclusion protocol does not influence the protocol.

In addition to these minimal requirements, there are some additional properties, which we would like to verify. They are:

5. *Return to start*: In any execution, it is always possible to return to a state in which all processes are positioned at the start label.

6. *No dead code*: Any statement always has the possibility of being executed by some process in the future.

Obviously, there are logical relations between some of these properties. For example, *No dead code* implies *No deadlocks*.

9.5.2 Translation into CPN Dynamic Properties

Now, we explain how the properties formulated for Lamport's Algorithm in the previous section can be verified by means of the CPN model. Each property of Lamport's Algorithm is translated into a dynamic property of the CPN model. The necessary formal definitions are given as we proceed. For a more complete description of dynamic properties for CP-nets, the reader is encouraged to consult [66].

Mutual exclusion. An *integer bound* for a place p is a limit on the number of tokens on p in all reachable markings. The *best integer bound* for p is the maximal number of tokens on p in any reachable marking. Formally:

Definition 10 *$n \in \mathbb{N}$ is an **integer bound** for $p \in P$, iff*

$$\forall M \in [M_0\rangle : |M(p)| \leq n.$$

*If an integer bound exists, p is said to be **bounded**. For a bounded place p, the **best integer bound** is the minimal $n \in \mathbb{N}$ such that n is an integer bound.* □

The *Mutual exclusion* property can be verified by considering the place CS_21 in the CPN model (see Fig. 9.2): When CS_21 contains a token with colour i, it corresponds to process i being in the critical section. If 1 is an integer bound for CS_21, then at any time at most one process will be in the critical section.

Persistent reachability of the critical section. A transition t is **impartial**, iff in any infinite occurrence sequence starting in the initial marking, t has infinitely many occurrences. Formally:

Definition 11 *Let IOS be the set of infinite occurrence sequences starting in M_0 and $OC_t(\sigma)$ be the number of occurrences of a transition $t \in T$ in an infinite occurrence sequence $\sigma \in IOS$.*

*A transition $t \in T$ is **impartial**, iff $\forall \sigma \in IOS : OC_t(\sigma) = \infty$.* □

The *Persistent reachability of the critical section* property can be verified by considering the transition sety0_23: When it occurs, process i is leaving the critical section. If sety0_23 is impartial, then we cannot have an infinite occurrence sequence in which the critical section is not left and, hence, not entered by some process an infinite number of times. Thus, the critical section always remains reachable.

However, if no infinite occurrence sequence exists, the impartiality property is trivially fulfilled. We therefore also have to establish the existence of an infinite occurrence sequence.

No deadlocks. A marking M is *dead*, iff no binding element is enabled in M. Formally:

Definition 12 *A marking $M \in \mathbb{M}$ is **dead** iff $\forall x \in BE : \neg M[x\rangle$.* □

The *No deadlocks* property can be proved directly by proving that the CPN model has no dead markings: Then, at any time during execution, at least one transition will be enabled and, hence, at least one process will be able to execute a statement.

Independence. For this property, we only need the basic concepts of markings and enabling already defined in Defs. 3 and 4. The *Independence* property is established, if we can verify that a process cannot be forced to enter the mutual exclusion protocol in order to unblock processes, which are executing the mutual exclusion protocol. Entering the mutual exclusion protocol corresponds to occurrence of the transition setbi_2. All other transitions of the CPN model are internal to the protocol. What we want to show, is that if setbi_2 is the only enabled transition, then all processes are outside the protocol, i.e., on the place start_1.

Return to start. A set of markings X is a *home space*, iff it is possible from any reachable marking to reach one of the markings in X. Formally:

Definition 13 *A set of markings $X \subseteq \mathbb{M}$ is a **home space**, iff*

$$\forall M \in [M_0\rangle : X \cap [M\rangle \neq \emptyset.$$ □

The *Return to start* property holds, if the set of markings X described next constitutes a home space: A marking M belongs to X, iff it is identical to the initial marking for all places but x, which is allowed to contain any single PID-token — in contrast to y, x will never be equal to 0, except from at the very beginning.

No dead code. A transition t is *live*, iff from any reachable marking, we can reach a marking in which t is enabled. Formally:

Definition 14 *A transition $t \in T$ is **live**, iff*

$$\forall M' \in [M_0\rangle \; \exists M'' \in [M'\rangle \; \exists x \in BE(t) : M''[x\rangle.$$ □

The *No dead code* property holds, if all transitions are live: Liveness of a transition means that the corresponding statement always has the possibility of being executed.

No fairness. In addition to the properties listed in Sect. 9.5.1, yet another property of Lamport's Algorithm is easy to derive from the CPN model. The algorithm is *not* fair: Any process wanting to enter the critical section may be starved forever. In the CPN model in Fig. 9.2, an infinite occurrence sequence starving any given process can easily be constructed.

9.5.3 Verification by Means of OS-graphs

It can be formally proved that with a consistent permutation symmetry specification all standard dynamic properties of a CP-net are preserved up to symmetry in the OS-graph. This means that we can verify such dynamic properties by considering the OS-graph instead of the O-graph. The standard dynamic properties of a CP-net are traditionally divided into *reachability*, *boundedness*, *home*, *live*, and *fairness* properties. These five sets of dynamic properties include the properties introduced in Sect. 9.5.2.

To verify the dynamic properties of a CP-net from the OS-graph, it is also worthwhile to construct the strongly connected components (SCCs) of the OS-graph and consider the *SCC-graph* [67]. Investigating the SCC-graph instead of the OS-graph may significantly speed up the check of a dynamic property. Using Tarjan's algorithm (see, e.g., [48]) or a similar algorithm, the construction of the SCC-graph is an inexpensive operation. Its time complexity is linear in the size of the OS-graph.

Before we show how to verify dynamic properties of a CP-net using the OS- and SCC-graph, we will fix some terminology and notation. The set of nodes in the SCC-graph is denoted SCC. A node in the SCC-graph or the OS-graph is called *terminal*, iff it has no outgoing arcs. The set of terminal nodes in an SCC-graph is denoted SCC_T. A node in the SCC-graph is called *trivial*, iff it corresponds to one single node in the OS-graph and has no arcs. The set of trivial nodes in the SCC-graph is denoted SCC_{TR}.

Proposition 2 below contains *proof rules* specifying how to investigate the OS- and the SCC-graph. A proof rule states a relationship between a dynamic property of a CP-net and the corresponding OS- or SCC-graph. The proof rules are explained after the proposition. The proofs of the items of the proposition can all be found in [67].

Proposition 2 *Let a consistent permutation symmetry SG for CPN be given. Then the following holds.*

1. ***Best integer bound**:*

   Let V be a set of representatives for the nodes of the OS-graph, and let $p \in P$. Then the best integer bound for p is: $max_{M \in V}|M(p)|$.

2. ***Impartiality**:*

   Let $t \in T$, and let $SCC_{OS\setminus t}$ denote the SCC-graph derived from the OS-graph after removing all arcs containing $t \in T$. Then:

   t is impartial $\Leftrightarrow \forall c \in SCC_{OS\setminus t} : c$ is trivial.

3. ***Existence of an infinite path starting in M_0:***

   $IOS \neq \emptyset \Leftrightarrow SCC_{TR} \neq SCC$.

4. ***Dead marking**:*

   Let $M \in [M_0\rangle$. Then:

   M is dead $\Leftrightarrow$ $[M]$ is terminal.

5. ***Home space**:*

   Let $X \subseteq \mathbb{M}$ and let X^c denote the set of strongly connected components to which some member of X belongs. Then:

   $[X]$ is a home space $\Leftrightarrow$ $SCC_T \subseteq X^c$.

6. ***Enabling**:*

   Let $t \in T$. Then:

   t is enabled in M $\Leftrightarrow$ t appears on an output arc from $[M]$.

7. ***Liveness**:*

   Let $T(c)$ denote the set of transitions appearing on arcs in a strongly connected component c, and let $t \in T$. Then:

   t is live $\Leftrightarrow$ $\forall c \in SCC_T : t \in T(c)$. □

The upper integer bound of a place may be found by visiting all nodes of the OS-graph. For each node, the number of tokens in the considered place is found. Finally, the maximum of all the computed numbers is returned. This proof rule is valid because any permutation symmetry preserves the number of tokens on each place, and all reachable markings have a representative in the OS-graph.

A transition t is impartial, iff it appears on an arc in all cycles of the OS-graph. This is the same as saying that if all arcs containing t are removed from the OS-graph, then there are no cycles, i.e., all strongly connected components are trivial. This proof rule is valid because any permutation symmetry maps a binding element to a binding element of the same transition.

Obviously, an infinite occurrence sequence starting in the initial marking M_0 exists, iff there is a strongly connected component which is not trivial.

A reachable marking M is dead iff the corresponding equivalence class in the OS-graph is terminal, i.e., it has no outgoing arcs. This is the same as saying that there are no enabled binding elements in any marking of the equivalence class of M.

With OS-graphs, it is not possible to distinguish equivalent markings. Thus, we cannot show that an arbitrary set X of markings is a home space. However, when all markings that are equivalent with markings in X are considered, the following holds: This set of markings is a home space iff all terminal nodes in the SCC-graph contain a node from X. This proof rule is valid because from any reachable node in the SCC-graph, it is possible to reach a terminal node in the SCC-graph and hence an element of $[X]$.

A transition t is enabled in a marking M, iff the equivalence class of M in the OS-graph has an output arc containing t. This follows immediately from the consistency requirement and the observation that permutation symmetries preserves transitions.

A transition is live, iff it appears on an arc in each of the terminal strongly connected components. This proof rule is valid because any permutation symmetry maps a binding element to a binding element of the same transition.

The reader interested in a complete treatment of how the standard dynamic properties are verified using the OS- and the SCC-graph is referred to [67]. The crucial observation to make here is that to use the OS-tool, it is not necessary to know these details. The OS-tool contains a full implementation of all the proof rules from [67], and the user simply has to invoke the appropriate query function and inspect the returned result. Examples of this will be given in the next section.

9.6 Carrying out the Verification

In this section, we consider the actual verification of Lamport's Algorithm using the OS-tool. Section 9.6.1 describes necessary preparations. Section. 9.6.2 reports on the application of the OS-tool, and includes statistics gathered to compare O- and OS-graphs. Finally, in Sect. 9.6.3, the obtained verification results are discussed.

9.6.1 Preparation of the Verification

In order to use the OS-tool for verification of Lamport's Algorithm, we have to prove that the permutation symmetry specification SG_L is consistent, i.e., prove that the three requirements in Def. 7 are fulfilled. The proof, which is included in full detail in [77], consists of a large number of cases, all of which are truly trivial. We will not present the proof in this paper. One thing related to the proof should, however, be noted at this point. In Sect. 9.2.1, we modelled a more general form of the for-statement in Lamport's Algorithm. We did not specify the order in which the entries in the b-array were to be tested. Had we done so, the permutation symmetry specification would not have been consistent. The reason is that if the entries are to be tested in turn staring from $b[1]$, then an ordering is imposed on the processes in Lamport's Algorithm. Hence, all processes are not treated in the same way from a symmetric point of view.

Once the permutation symmetry specification is proved consistent, the OS-tool can be applied. Verification of Lamport's Algorithm amounts to the following steps, which will be discussed below.

1. Implementation of the permutation symmetry specification.
2. Generation of the OS-graph.
3. Generation of the SCC-graph for the OS-graph.
4. Invocation of suitable query functions.

Item 1 consists of implementing the predicates $EquivMark$ and $EquivBE$ previously discussed in Sect. 9.4.1. The utility functions provided by the OS-tool to support the implementation of the predicates are described, together with the underlying data structures, in [2]. In this paper, we will not describe how to implement the two predicates. They are included in full detail in [77]. For a CPN model like the one for Lamport's Algorithm, it is very easy to program a naive version of $EquivMark$ and $EquivBE$. One way to implement, e.g., $EquivMark$ is just to let it test all permutation symmetries in turn. If one is found that maps the first marking given as argument to the second, true is returned, otherwise false is returned. However, for efficiency

reasons, it is important – and indeed possible – to write the predicates in a more clever way.

When the permutation symmetry specification has been implemented, the OS-graph and the SCC-graph can be generated (items 2 and 3). This is fully automatic — two generation functions are available via menus. Finally, suitable query functions (item 4) can be invoked to produce the desired verification results.

Figure 9.6.1 shows how the correctness of Lamport's Algorithm, as formulated in Sect. 9.5, is expressed as query functions in the OS-tool. The mutual exclusion property (lines 1-2) is checked using the standard query function UpperInteger. This function takes a place as argument and returns the best upper integer bound of the place. The place CS_21 is referred to by Mark.Lamport'CS_21 1. The impartiality of the transition sety0_23 is checked using the query function IsImpartial (lines 4-5) which takes a transition as argument, and checks whether the transition is impartial. The transition sety0_23 is referred to by TI.Lamport'sety0_23 1. The existence of an infinite occurrence sequence is easily checked by inspecting the number of nodes in the SCC-graph. If there are fewer nodes in the SCC-graph than the OS-graph, a non-trivial SCC node exists.

The absence of deadlocks property is checked using the query function ListDeadMarkings (lines 7-8). This function lists the equivalence classes containing the dead markings. If this list is empty, we know that the CP-net has no dead markings.

The independence property is verified using the query function PredAllNodes (lines 10-12), which takes a predicate as argument, and returns the equivalence classes satisfying the predicate. The predicate expresses that we are searching for the equivalence classes in which the transition setbi_2 is the only enabled transition, and where some process is not positioned at the start label. If no equivalence classes satisfy this property, we have verified independence.

To establish the return to start property, we use the query function HomeSpace (lines 14-16), which takes a list of equivalence classes, and checks whether the corresponding set of markings constitutes a home space. We use the query function PredAllNodes to obtain the list of equivalence classes where all processes are positioned at the start label and y is 0. Finally, we verify the no dead code property using the query function ListLiveTIs (lines 18-19), which lists the live transitions. We check that the live transitions equals the set of all transition (TI.All).

9.6.2 Application of the OS-tool

An inherent property of the occurrence graph method is that any graph is generated for a fixed value of the system parameters — in this case the number of processes N. Thus, Lamport's Algorithm was verified for a set of fixed values. The computing power available determines the possible values of N. The results presented here were obtained on a SUN Ultra Sparc Workstation with 512 MB of RAM.

In addition to generating and analysing the OS-graphs, we also considered O-graphs. This is a main point, because the overall goal of using OS-graphs is to save space, and we want to demonstrate that this was actually accomplished. Table 9.6.2 contains the sizes of the O- and OS-graphs. The columns with headline Ratio shows the reduction factor for the OS-graph compared with the O-graph. It holds the number of nodes and arcs, respectively, for the O-graph divided with the corresponding num-

```
1: (* — Mutual Exclusion — *)
2: UpperInteger (Mark.Lamport'CS_21 1);
3:
4: (* — Persistent Reachability of the critical section — *)
5: IsImpartial(TI.Lamport'sety0_23 1);
6:
7: (* — No deadlocks — *)
8: ListDeadMarkings ();
9:
10: (* — independence — *)
11: PredAllNodes (fn n => (OnlyEnabled(TI.Lamport'setbi_2 1 n)) andalso
12:                          ((Mark.Lamport'start_1 1 n) <><> PID));
13:
14: (* — return to start — *)
15: HomeSpace (PredAllNodes (fn n => ((Mark.Lamport'start_1 1 n) == PID)
16:                          andalso ((Mark.Lamport'y 1 n) == 1`0)));
17:
18: (* — No dead code — *)
19: ListLiveTIs () = TI.All;
```

Figure 9.12: Queries for the correctness of Lamport's Algorithm.

	Nodes			Arcs			
N	O-graph	OS-graph	Ratio	O-graph	OS-graph	Ratio	N!
2	380	191	2.0	716	358	2.0	2
3	19,742	3,367	5.9	58,272	9,788	6.0	6
4	1,914,784	83,235	23.0	9,046,048	383,030	23.6	24

Table 9.1: Sizes of O- and OS-graphs.

ber for the OS-graph. The outermost right column lists the factorial $N!$ of N, i.e., the size of the group of permutation symmetries.

Due to the state explosion problem, O-graphs could only be generated for values of N up to 3 with the available computing resources. In spite of this, for $N = 4$, we actually do know the size of the O-graph. It is calculated from the OS-graph. Using algebraic group theory, we have designed an efficient algorithm to do so without unfolding. The details of the method are described in [79]. This algorithm is interesting, because it enables us to compare the sizes of the O- and OS-graph, even when generation of the O-graph is impossible. The algorithm also turned out to be a significant test to justify that the implementation of the permutation symmetry specification, i.e., the predicates $EquivMark$ and $EquivBE$ was correct, in the sense that it captured the intended assignment of symmetry groups to the atomic colour sets, and the inheritance rules for the structured colour sets. Moreover, the algorithm was suitable to increase our confidence in the consistency of the chosen permutation symmetry specification. For $N \leq 3$, if a discrepancy between the size of a generated O-graph and the size calculated from the OS-graph appeared, then we know that something is wrong. Using

	Seconds of CPU Time		
N	O-graph	OS-graph	Ratio
2	1	1	1
3	66	16	4
4	-	3,637	-

Table 9.2: Generation times for O- and OS-graphs.

this test, we corrected two non-trivial errors (see [77]) in our initial implementation of $EquivMark$. When an accordance between the sizes obtained by generation and calculation was recorded, it was very strong evidence that the CPN model and the permutation symmetry specification were as intended. In this way, the algorithm was used to narrow the gap between the abstract permutation symmetry specification, i.e., the assignment of algebraic groups to the atomic colour sets, and its implementation.

Now, consider the time used for the verification. Generation of SCC-graphs and evaluation of query functions take a relatively short time. The dominant time-consuming task is to generate the OS-graphs, or the O-graphs when we want to compare. These generation times are contained in Table 9.6.2. An empty entry (-) signals that the measure could not be obtained.

9.6.3 Discussion of the Verification

With OS-graphs, we could verify Lamport's Algorithm for all $N \leq 4$. Results from queries in the OS-tool showed that the correctness properties listed in Sect. 9.5 were true. From Table 9.6.2, it can be seen that for a given N, the O-graph is almost $N!$ bigger than the OS-graph. This is remarkable. Because no more than $N!$ permutation symmetries are available, an equivalence class cannot be bigger than $N!$. Therefore, $N!$ is a theoretical limit on the size of the O-graph divided by the size of the OS-graph, i.e., the reduction obtained is almost maximal. From Table 9.6.2, it can be seen that for a given N, generation of the OS-graph was faster than generation of the O-graph. Even though we only have two observations, they indicate what seems to be a general fact: What it lost on a more expensive test on equivalence of markings and binding elements, is accounted for by having fewer nodes and arcs to generate; and also to compare with before a new node or arc can be inserted in the OS-graph.

As explained in the beginning of this section, a slightly generalised version of Lamport's Algorithm was the subject for our verification, because of a problem caused by the for-statement with respect to applying OS-graphs. The model of the generalised algorithm has a larger O-graph than the model of the original algorithm. Thus, even though OS-graphs yield big savings, in some cases, the starting point for using them is worse than the starting point for using O-graphs. However, it is still worthwhile to use OS-graphs: For $N = 3$, the O-graph for the CPN model of the original algorithm has $11,978$ nodes and $32,226$ arcs. The OS-graph for the CPN model of the generalised algorithm has only $3,367$ nodes and $9,788$ arcs. As an aside, after our own verification of Lamport's Algorithm, we discovered that for-statements have also been identified as causing problems with respect to exploiting symmetries in verification in [65].

At a first glance, the values of N, for which Lamport's Algorithm can be verified,

	Nodes			Arcs			
N	O-graph	OS-graph	Ratio	O-graph	OS-graph	Ratio	N!
2	268	135	1.9	484	247	1.9	2
3	6,134	1,071	5.7	16,296	2,765	5.9	6
4	118,176	5,755	20.5	410,244	18,600	22.1	24
5	2,071,872	24,035	86.2	8,892,460	91,383	97.3	120
6	34,258,216	83,895	408.4	175,300,026	361,151	485.4	720
7	543,954,112	255,394	2129.9	3,233,579,902	1,213,953	2663.7	5040

Table 9.3: Sizes of O- and OS-graphs – atomic for-statement.

might not impress. We would of course like as large values as possible. Can anything be done with respect to creating a model more suitable for occurrence graph analysis? The answer is yes, but we pay a price with respect to the credibility of the verification. If we model the for-statement in a coarser fashion, we are be able to do the verification for all $N \leq 7$. The way to modify the modelling of the for-statement is to have one transition, which is enabled when all $b[i]$'s are $false$, instead of testing all the entries of the b-array individually. The size of the OS-graphs and O-graphs (for comparison) using this coarser modelling are listed in Table 9.6.3.

The coarser modelling of the for-statement can be informally justified by the observation that when a process starts the execution of the for-statement, it has to read all the entries in the b-array. In between the reading of the individual entries, there is no writing to any of the shared variables, and hence the process cannot convey any information to the other processes during the execution of the for-statement thereby possibly influencing their execution. Therefore, the process might as well "wait" until all entries are $false$ and then continue execution. The coarser modelling is a bit dangerous though, because it violates the assumption about atomicity in Lamport's Algorithm. A non-atomic statement is modelled as if it was atomic, jeopardising the correctness of the model.

9.7 Conclusions

The main contributions of this paper are the presentation of the developed OS-tool supporting verification of CP-nets by means of OS-graphs, and the demonstration of the OS-graph method on a non-trivial example. Using OS-graphs, it was possible to verify the crucial properties of Lamport's Algorithm. Once the permutation symmetry specification was proved consistent and implemented in terms of the predicates $EquivMark$ and $EquivBE$, the verification was very easy and almost automatic: Generate an OS-graph and an SCC-graph, and invoke suitable query functions in the OS-tool.

9.7.1 Verification of Lamport's Algorithm

In our search for a good example to demonstrate the OS-tool for verification, the inspiration to consider Lamport's Algorithm came from Balbo et al. [3]. Here, the authors

verify Lamport's Algorithm using Coloured Stochastic Petri Nets [91] and place invariants. Balbo et al. verify Lamport's Algorithm on a model in which the for-statement is modelled in the coarse fashion described at the end of Sect. 9.6. An advantage of the approach of Balbo et al. is that Lamport's Algorithm is verified for an arbitrary value of N.

In the original presentation of Lamport's Algorithm in [88], Lamport himself establishes correctness. Here an axiomatic method decorating the algorithm text with assertions is applied. Lamport concentrates on establishing deadlock freedom and mutual exclusion. As in [3], the properties are proved for an arbitrary value of N. Both Balbo et al. and Lamport conduct complex and lengthy mathematical proofs. For the mutual exclusion property, the former only sketch the proof, while the latter more generally relies on a number of proof sketches.

Balbo et al. also study the performance of Lamport's Algorithm. It is an important subject, but outside the scope of the work we present in this paper. With respect to the logical behaviour of the algorithm, we establish similar properties to Balbo et al. and Lamport, plus other important properties. The main virtue of our proof is that it is almost automatic and, hence, much less error-prone. We do not need to engage in detailed or complex mathematical arguments. Based on this, we claim that our results are quite reliable.

Verification based on OS-graphs also has some drawbacks. First of all, it is necessary to fix the system parameter — in this case the number of processes. Secondly, the number of processes, which can be handled presently, is restricted to $N \leq 4$ (or $N \leq 7$ with a coarser modelling). Therefore, it is relevant to ask if we could have done better with respect to the chosen method of verification, e.g., if we had combined symmetries with other methods for condensing occurrence graphs. One idea is to consider Haddad's structural reductions [54]. However, by inspecting of the CPN model in Fig. 9.2, it can be seen that the conditions which are required in order to use structural reductions are not present. Yet another idea is to apply Valmari's stubborn sets [86, 125]. It is generally recognised [32] that stubborn sets and symmetries can be applied simultaneously, thus yielding an even smaller occurrence graph.

9.7.2 Tool Support for OS-graphs

Developing tool support for OS-graphs involved making a number of design decisions as to how the tool should support the user in conducting verification of systems. A key design choice is whether the symmetries should be automatically detected by the tool or be provided by the user based on knowledge of the system being considered. We have chosen the latter approach for two main reasons. Firstly, computing the symmetries is expensive, and it is our experience that the user always has some knowledge/intuition about the potential symmetries of the system. For example, in Lamport's Algorithm, it is obvious that the symmetry is in the processes. Secondly, if the tool detects the symmetries in the system, it might result in symmetries which are difficult to interpret, and the tool may even fail to detect the symmetries due to, e.g., an error in the design causing the system to behave asymmetrically.

An alternative to computing the symmetries is to put narrow syntactically restrictions on the modelling language in such a way that only symmetric constructs are expressible. Such ideas have been pursued for Well-formed Coloured Nets (WNs) [13].

Detection of symmetries in WNs can be fully automated, thus effectively eliminating the need of conducting a consistency proof. For flexibility reason, we have chosen not to base the OS-tool on putting syntactical restrictions on CP-nets. This implies that a proof of consistency has to be conducted. Proving the consistency of the permutation symmetry specification is tedious, because of the many cases in the proof, which need to be considered. Therefore, it would be preferable, if the tool could check most of these cases automatically. This can be done in a way similar to the checking of a proposed place invariant as described in [67]. The tool may not be capable of conducting a full proof of consistency but may significantly reduce the number of cases that the user needs to consider.

With respect to detection of symmetries, our approach is similar to the approach to exploiting symmetry described in [22, 23, 26] and implemented in the SYMM tool. In [26], symmetries are also combined with binary decision diagrams (BDDs) to design an efficient model checking algorithm. In contrast, the Murphi programming language/environment [65], relies on automatic detection of symmetries directly from the syntax of the description language. The same is the case for the SMC tool [34,111].

A fundamental aspect of putting OS-graphs into practice is to be able to determine whether two markings/binding elements are symmetric or not. The computational complexity of this problem has been studied in a number of papers, e.g., [22, 26], showing that the problem is equivalent to the graph isomorphism problem for which no polynomial time algorithm is known. This may indicate that the use of OS-graphs is impractical. However, it is our experience that what is lost on a expensive test on equivalence of markings and binding elements, is accounted for by having fewer nodes and arcs to generate; and also to compare with before a new node or arc can be inserted in the OS-graph.

In the OS-tool the equivalence test is handled by the two functions $EquivMark$ and $EquivBE$ provided by the user. Case studies with the OS-tool have shown that writing these two predicates is error-prone. Therefore, we plan to add an improved interface for permutation symmetry specifications: The user is only asked to assign the chosen symmetry groups to the atomic colour sets. The OS-tool then automatically generates $EquivMark$ and $EquivBE$.

Chapter 10

Verification of Coloured Petri Nets Using State Spaces with Equivalence Classes

The paper *Verification of Coloured Petri Nets Using State Space with Equivalence Classes* constituting this chapter has been published as a technical report [72], as a workshop paper [73], and as a book chapter [75].

[72] J. B. Jørgensen and L. M. Kristensen. Verification by State Spaces with Equivalence Classes. Technical report, Department of Computer Science, University of Aarhus, Denmark, February 1997. DAIMI PB-515.

[73] J. B. Jørgensen and L. M. Kristensen. Verification of Coloured Petri Nets Using State Spaces with Equivalence Classes. In: B. Farwer, D. Moldt and M-O. Stehr (Eds): Proceedings of Workshop on Petri Nets in System Engineering (PNSE'97) Modelling, Verification, and Validation, Hamburg, Germany, Publication No. 205, Universität Hamburg, Fachberich Informatik, pp. 20-31, 1997.

[75] J. B. Jørgensen and L. M. Kristensen. Verification of Coloured Petri Nets Using State Spaces with Equivalence Classes. In: W. v. d. Aalst, J.-M. Colom, F. Kordon, G. Kotsis, and D. Moldt. Petri Net Approaches for Modelling and Validation. LINCOM Studies in Computer Science, No. 1, 1999. To appear.

The content of this chapter is equal to the book chapter [75] except for minor typographical changes.

Verification of Coloured Petri Nets Using State Spaces with Equivalence Classes

J. B. Jørgensen* L. M. Kristensen†

Abstract

This paper demonstrates the potential of verification based on state spaces reduced by equivalence relations. The basic observation is that quite often some states of a system are similar, i.e., they induce similar behaviours. Similarity can be formally expressed by defining an equivalence relation on the set of states and on the set of actions of a system under consideration. A state space can be constructed in which the nodes correspond to equivalence classes of states, and the arcs correspond to equivalence classes of actions. Such a state space is often much smaller than the ordinary full state space, but it does allow derivation of many verification results.

Other researchers have taken advantage of the symmetries in systems, which induce a certain kind of equivalence. The contribution of this paper is to show that a more general notion of equivalence is useful. As a representative example a communication protocol is verified. Aided by a developed computer tool significant reductions of state spaces are exhibited, representing some first results on the practical use of state spaces with equivalence classes for Coloured Petri Nets.

Keywords: State Space Based Approaches, Efficient Model Checking, Tools, Coloured Petri Nets, Communication Protocols, Reduction by Equivalence and Symmetry.

10.1 Introduction

In the research on verification of parallel and distributed systems, attention has been given to take advantage of *symmetry* [35] to alleviate the state explosion problem. Symmetry appears when a system is composed of similar components, whose identities are immaterial with respect to state space verification. As an example, consider the well-known dining philosophers system. A state of this system in which philosophers 1 and 3 are eating, is symmetric to a state in which philosophers 3 and 5 are eating. The first state can be mapped to the second by the permutation which rotates philosopher i into philosopher $i + 2$ (modulo the number of philosophers). Symmetry is also present in many real-world systems.

*Systematic Software Engineering A/S, DK-8230 Aabyhøj, DENMARK.
E-mail: jbj@systematic.dk.
†Department of Computer Science, University of Aarhus, DK-8000 Aarhus C., DENMARK.
E-mail: lmkristensen@daimi.au.dk.

State spaces with equivalence classes (SSEs) are presented in [67] under the name *occurrence graphs with equivalence classes* (OE-graphs) as a theoretical generalisation of state spaces based on symmetries. In [67] it is noted that the experiences with practical use of SSEs are rather limited. Moreover, in the examples of SSEs given in [67] the equivalence relations are defined using only the structure of the systems under consideration. In particular, symmetry is a structural, static notion, based on permutation of similar components. The contribution of this paper is to recognise that sometimes, a more dynamic kind of equivalence is beneficial, and to demonstrate that SSEs are applicable for this purpose. SSEs are described and defined for the formalism of Coloured Petri Nets (CP-nets or CPN) [66], but the idea generalises immediately to formalisms allowing an explicit representation of both states and actions of systems.

This paper is organised as follows: Section 10.2 introduces the communication protocol to be used as example and the properties that we are going to verify. Section 10.3 presents the concept of SSEs and shows how to define appropriate equivalence relations on the states and actions of the considered example. In Sect. 10.4, the example is verified: It is first proved that the equivalence relations are consistent (well-defined), ensuring that the SSEs can actually be used to derive the desired verification results. Then it is described how the verification was carried out using the developed computer tool. Finally, statistics are presented to compare verification based on SSEs and verification based on ordinary full state spaces. Section 10.5 presents some techniques to support the proof of consistency by a computer tool. Section 10.6 draws the conclusions and discuss related work. The reader is assumed to be familiar with CP-nets as defined in [66].

10.2 An Example - The Transport Protocol

In this section we present a CPN model of a protocol from the transport layer of the ISO reference model (see e.g., [29]). In the following this protocol will be referred to as the *transport protocol*. This section also introduces the basic notation related to CP-nets used in subsequent sections. The CPN model has been created with the Design/CPN tool [16, 99] supporting CP-nets. The computer tool developed and applied in this paper for verification based on SSEs [78] is an integrated part of the Design/CPN tool, and is referred to as the Design/CPN Condensed State Space Tool.

The transport layer is concerned with protocols ensuring reliable transmission between sites. The CPN model of the transport protocol is shown in Fig. 10.1. The system consists of a *sender* (left), which wants to transfer some data to a *receiver* (right). Communication takes place on an unreliable *network* (middle), with risk of loss and overtaking. The data is a text string, split into substrings of length eight, and each assigned a *sequence number*. A pair consisting of a sequence number and a string is called a *data packet*. Data packets must be received in the right order. Whenever a data packet is received, an *acknowledgement* is sent. The protocol is a stop-and-wait protocol: The sender keeps sending copies of the data packet that the receiver expects next, until the sender gets a proper acknowledgement from the receiver. Then, the sender starts sending the next data packet. The term *packet* is used generically for both data packets and acknowledgements.

The state of the sender is modelled by the two places Send and NextSend. Send

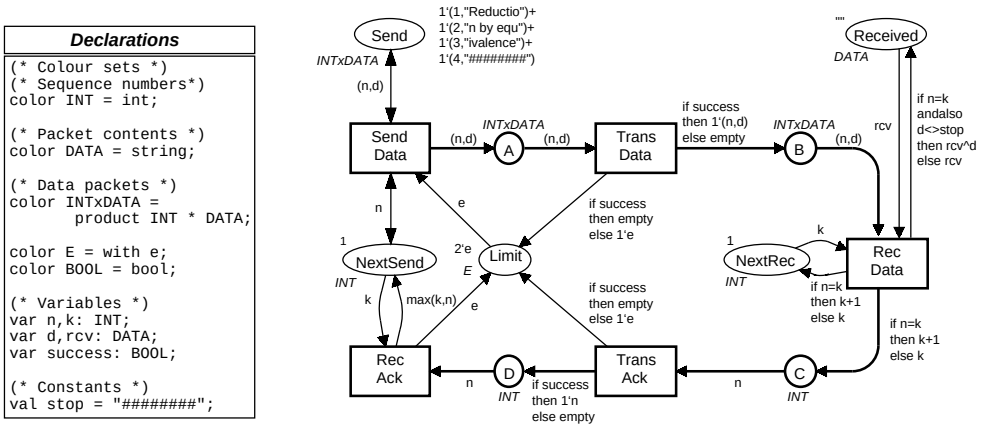

Figure 10.1: CPN model of the Transport Protocol.

contains all data packets, and NextSend contains the sequence number of the next data packet to be sent. The state of the receiver is, in a similar way, modelled by the two places Received and NextRec. Received contains the data received until now, and NextRec contains the sequence number of the data packet expected next. The state of the network is modelled by the circular *network places*, A and B which may contain data packets, and C and D which may contain acknowledgements. The place Limit is used to model that the network has a certain capacity, i.e., that the network can maximally contain a certain number of packets at a time. This also ensures that the system has a finite state space.

The actions of the sender correspond to the two transitions SendData and RecAck. SendData models sending of data packets and RecAck models reception of acknowledgements. The receiver has only one action, corresponding to the transition RecData which models reception of data packets and sending of acknowledgements. The actions of the network correspond to the two transitions TransData and TransAck modelling transfer of packets. The possibility of losing packets on the network is modelled using the boolean variable success. In occurrences of the transitions TransData and TransAck, success can be bound to either true or false. The former case corresponds to successful transmission, the latter to loss of a packet.

The initial markings of the places are written next to the places (and omitted when empty). For example, Send initially contains all data packets, all four network places are empty, and Received contains the empty string (“”).
The properties which we want to verify for the transport protocol are:

- *No improper termination*: If the protocol terminates, all data packets have been received exactly once, in the same order as they were sent, and the network is empty.
- *Possibility of termination*: In any reachable state of the protocol, it is always possible within a finite number of steps to terminate the protocol.
- *Eventual termination*: If the network looses only finitely many packets, then the protocol does eventually terminate.

Multi-sets are functions from their domain into the set of natural numbers. A multi-set ms over a domain X is written as a formal sum like $\sum_{x \in X} ms(x)'x$. empty denotes the empty multi-set.

In a given marking M of a CP-net, the marking of a place p is denoted $M(p)$. $\mathbb{M}$ denotes the set of all markings. M_0 denotes the initial marking. When a *binding element* b (a pair consisting of a transition and a binding of data values to its variables) is enabled in a marking M_1 and the occurrence yields the marking M_2, we write $M_1[b\rangle M_2$. The notation $M_1[b\rangle$ means that b is enabled in M_1. A binding element of a transition t will be written on the form $(t, \langle v_1 = c_1, v_2 = c_2, \ldots v_n = c_n\rangle)$, where $v_1 \ldots v_n$ are the variables of t and $c_1 \ldots c_n$ are data values. The set of all binding elements is denoted BE.

A *reachable marking* is a marking which can be obtained from M_0 by a sequence of occurrences of binding elements. $[M_0\rangle$ denotes the set of all reachable markings.

10.3 State Spaces with Equivalence Classes

An *ordinary state space (SSO)* for a CP-net is a directed graph with a node for each reachable marking and arcs corresponding to occurring binding elements. The definition of a *state space with equivalence classes* (SSE) for a CP-net requires that an *equivalence specification* is given. An equivalence specification consists of two equivalence relations — one on the set of markings ($\approx_M$) and one on the set of binding elements ($\approx_{BE}$). The equivalence relations must capture an equivalence actually present in the considered system. This means that two equivalent markings must induce similar behaviours. This requirement is referred to as *consistency* and is related to bisimulation [94], an issue to which we will return in Sect. 10.5. Consistency is formalised in Def. 15 below, which is equivalent to Def. 2.2. in [67].

For two markings (or two binding elements) x and y, if x is equivalent to y, we write $x \approx y$, and the equivalence class of x is written $[x]$. For a set X, $[X]$ denotes the set of elements equivalent with some element in X. For example, $[[M_0\rangle]$ denotes the set of markings equivalent with a reachable marking.

Definition 15 *Let $ES = (\approx_M, \approx_{BE})$ be an equivalence specification for a CP-net. ES is **consistent** if and only if for all markings $M_1, M_1^{'} \in [[M_0\rangle]$ and all binding elements $b \in BE$:*

$$M_1^{'} \in [M_1] \wedge M_1[b\rangle M_2 \Rightarrow \exists b^{'} \in [b] \wedge \exists M_2^{'} \in [M_2] : M_1^{'}[b^{'}\rangle M_2^{'}$$

□

The definition expresses that equivalent markings are required to have equivalent sets of enabled binding elements, and equivalent sets of directly reachable markings. Intuitively, this justifies that it is sufficient to explore the possible behaviours of the system for one marking of each equivalence class.

Given a consistent equivalence specification, the SSE has a node for each equivalence class containing a reachable marking. Moreover, the SSE has an arc between two nodes if and only if there is a marking in the equivalence class of the source node

in which a binding element is enabled and leads to a marking in the equivalence class of the destination node. There is exactly one arc for each equivalence class of binding elements with this property. This is formalised in Def. 16 below, which, disregarding differences in terminology, is identical to Def. 2.3 in [67]. The set of all equivalence classes of markings is denoted $\mathbb{M}_{\approx}$. The set of all equivalence classes of binding elements is denoted $BE_{\approx}$.

Definition 16 *A **state space with equivalence classes** (SSE) for a CP-net is a triple* (V, A, N) *satisfying the requirements below:*

1. $V = \{C \in \mathbb{M}_{\approx} | C \cap [M_0\rangle \neq \emptyset\}$
2. $A = \{(C_1, B, C_2) \in V \times BE_{\approx} \times V \mid \exists (M_1, b, M_2) \in C_1 \times B \times C_2 : M_1[b\rangle M_2\}$
3. $\forall a = (C_1, B, C_2) \in A : N(a) = (C_1, C_2)$ □

Items 1 and 2 define the sets of nodes and arcs, respectively. Item 3 defines a function which for each arc designates its source and destination. Item 3 is necessary to allow multiple arcs between two nodes which may appear in SSEs. An SSE is often much smaller than the corresponding SSO, of course depending on the equivalence specification. The weaker the equivalence specification, i.e., the more markings and binding elements are considered equivalent, the more reduction is obtained. The SSE can be computed on-the-fly, i.e., *without* first constructing the SSO and then merging markings and binding elements into equivalence classes. This is done using a modified version of the standard algorithm for generating an SSO. Instead of testing whether the marking is already included in the SSE, it is tested whether an equivalent marking is already included. Similarly, there is a test of whether an equivalent binding element is already inserted between the two nodes before a new arc is created.

The user of the condensed state space tool supplies the equivalence specification by writing two predicates: one predicate expressing when two markings are equivalent and one predicate expressing when two binding elements are equivalent. Using the supplied equivalence specification the tool then computes the corresponding SSE. When the SSE is finite, i.e., it has a finite set of nodes and arcs, it can be used directly to prove many dynamic properties of the CP-net. The user of the tool verifies/proves properties about the considered system by invoking suitable query functions. The query functions are based on proof fules which relates dynamic properties of CP-nets with properties of the SSE. The weaker the equivalence specification the weaker are the proof fules, and hence the equivalence specification should to be chosen such that the desired properties can still be verified. In Sect. 10.4 we describe how this is done for the three properties listed at the end of Sect. 10.2 which we want to verify for the transport protocol. The reader interested in an exhaustive treatment of the proof rules is encouraged to consult Sect. 2.3 in [67]. The condensed state space tool contains a full implementation of the proof rules for SSEs from Sect. 2.3 in [67].

The equivalence specification for the transport protocol is dynamic, and based on the observation that certain packets on the network become similar as the system executes. Suppose that the receiver expects data packet number three next. Arrival of any data packet with a sequence number less than three does not change the state of the receiver. Such a data packet on the network will be called *old*. Arrival of any old data packet has the effect that an acknowledgement asking for data packet number three

is sent. Thus two old data packets arriving at the receiver have exactly the same effect. Similar observations and terminology apply to acknowledgements arriving at the sender. The purpose of the equivalence specification is to capture that old data packets and old acknowledgements, respectively, are equivalent. Below we formalise the above observation by formally defining the equivalence specification for the transport protocol.

First we consider the equivalence relation on the set of markings. Let $M_1, M_2 \in \mathbb{M}$ be two markings. $M_1 \approx_M M_2$ requires that the markings of all places but the network places A, B, C, and D are identical in M_1 and M_2. For each of the network places, the marking of the place is partitioned into two multi-sets, one containing the old packets, and one containing the other packets. In order for $M_1 \approx_M M_2$ we require that the number of old packets are the same in M_1 and M_2, and that the multi-sets of the other packets are identical in M_1 and M_2. Below, $|ms|$ denotes the size of the multi-set ms, i.e., the number of elements with their multiplicity taken into account. For a multi-set with only one element, ms also denotes that element. The definition uses a function old which takes a marking M in which $|M(\mathsf{NextRec})| = 1$ and one of the network places $p \in \{\mathsf{A}, \mathsf{B}\}$ as arguments, and yields the multi-set of old packets on that place:

$$old(M, p) = \sum_{\{(n,d) \in M(p) \mid n < M(\mathsf{NextRec})\}} ((M(p))(n, d))'(n, d) \tag{10.1}$$

A similar function, also called old, coping with old acknowledgements on the places $p \in \{\mathsf{C}, \mathsf{D}\}$ is used, where the condition $n < M(\mathsf{NextRec})$ in (1) is replaced by $n \leq M(\mathsf{NextSend})$. Now the definition of $\approx_M$:

$$\begin{aligned}
&M_1 \approx_M M_2 \Leftrightarrow \\
&\quad (\forall p \in \{\mathsf{NextSend}, \mathsf{NextRec}\} : |M_1(p)| = |M_2(p)| = 1) \wedge \\
&\quad (\forall p \in \{\mathsf{Send}, \mathsf{NextSend}, \mathsf{Limit}, \mathsf{Received}, \mathsf{NextRec}\} : M_1(p) = M_2(p)) \wedge \\
&\quad (\forall p \in \{\mathsf{A}, \mathsf{B}, \mathsf{C}, \mathsf{D}\} : |old(M_1, p)| = |old(M_2, p)| \wedge \\
&\qquad\qquad M_1(p) - old(M_1, p) = M_2(p) - old(M_2, p)
\end{aligned}$$

We now consider the equivalence relation on the binding elements. Let $b_1, b_2 \in BE$ be two binding elements. In order for $b_1 \approx_{BE} b_2$ we require that b_1 and b_2 are binding elements for the same transition. For the transitions SendData, RecData, and RecAck we consider all binding elements equivalent. For TransData and TransAck there is the additional requirement that a loss of packet in either b_1 or b_2 must be matched by the other. We do not consider all binding elements of TransData and TransAck equivalent, since we want to be able to make a distinction between loss and successful transmission of packets in connection with the eventual termination property.

In the definition below, we use a predicate $is\_success$ which given a binding element for the transition TransData yields true if the binding element corresponds to successful transmission of a data packet. We use a similar predicate $is\_success$ which is defined for TransAck.

$$is\_success(\mathsf{TransData}, \langle \mathsf{n} = n', \mathsf{d} = d', \mathsf{success} = success' \rangle) = success'$$

Below follows the definition of $\approx_{BE}$. The transition of a binding element $b \in BE$ is denoted $t(b)$.

$$
\begin{array}{l}
b_1 \approx_{BE} b_2 \Leftrightarrow \\
\quad t(b_1) = t(b_2) = \mathsf{SendData} \vee t(b_1) = t(b_2) = \mathsf{RecData} \vee \\
\quad t(b_1) = t(b_2) = \mathsf{RecAck} \vee \\
\quad (t(b_1) = t(b_2) = \mathsf{TransData} \wedge (is\_success(b_1) \Leftrightarrow is\_success(b_2))) \vee \\
\quad (t(b_1) = t(b_2) = \mathsf{TransAck} \wedge (is\_success(b_1) \Leftrightarrow is\_success(b_2)))
\end{array}
$$

10.4 Verification of the Transport Protocol

In this section, we describe verification of the transport protocol. First, we prove that the equivalence specification defined in the previous section is consistent. Then, we translate the properties listed at the end of Sect. 10.2 into dynamic properties of the CP-net, and outline how these properties can be proved from the SSE. Finally, we present statistics to compare the verification based on SSEs and SSOs for different values of the system parameters L and N. L is the capacity of the network as determined by the number of e tokens on the Limit place in the initial marking, and N is the number of data packets as determined by the initial marking of the Send place.

We first prove that the equivalence specification defined in Sect. 10.3 is consistent according to Def. 15. Let $M_1, M_1^{'} \in [[M_0\rangle]$. Let $b \in BE$. Assume that $M_1 \approx_M M_1^{'}$ and that $M_1[b\rangle M_2$. We prove the existence of $b^{'}$ and $M_2^{'}$ such that:

$$b^{'} \approx_{BE} b \wedge M_2^{'} \approx_M M_2 \wedge M_1^{'}[b^{'}\rangle M_2^{'}.$$

We do so by a case analysis on the transition of b. In this paper, we sketch the proof by considering the transition TransData. The other cases are similar. We split the proof into four cases according to whether b corresponds to an old data packet or not, and whether b corresponds to a loss or not.

We use a predicate is_old defined on the binding elements of TransData and TransAck which yields true if the binding element corresponds to transmission of an old data packet or acknowledgement. For $p \in \{\mathsf{A}, \mathsf{B}\}$ and a marking M, let $young(M, p)$ be the marking of p in M in which all old data packets are removed from p.

Case 1: Assume $\neg is\_old(b) \wedge is\_success(b)$, i.e., b corresponds to a successful transmission of a data packet $(n', d') \in young(M_1, \mathsf{A})$. Since $M_1 \approx_M M_1^{'}$ we have $(n', d') \in young(M_1^{'}, \mathsf{A})$. Hence b is also enabled in $M_1^{'}$, and we choose $b^{'} = b$. Let $M_2^{'}$ be such that $M_1^{'}[b\rangle M_2^{'}$. Since an occurrence of b cannot change the marking of NextRec, an occurrence of b cannot convert a data packet which is not old to an old, and vice versa. Since $M_1 \approx_M M_1^{'}$, we therefore have: $young(M_2, \mathsf{A}) = young(M_1, \mathsf{A}) - \{(n', d')\} = young(M_1^{'}, \mathsf{A}) - \{(n', d')\} = young(M_2^{'}, \mathsf{A})$. For old data packets:

$|old(M_2, \mathsf{A})| = |old(M_1, \mathsf{A})| = |old(M_1^{'}, \mathsf{A})| = |old(M_2^{'}, \mathsf{A})|$. A similar argument can be made for the place B. Since all places but A and B are left unchanged by an occurrence of b, we conclude that $M_2^{'} \approx_M M_2$.

Case 2: Assume $\neg is\_old(b) \wedge \neg is\_success(b)$, i.e., b corresponds to a loss of a data packet which is not old. This case is similar to case 1 above.

Case 3: Assume $is\_old(b) \wedge is\_success(b)$, i.e., b corresponds to a successful transmission of an old data packet. Since $M_1 \approx_M M_1^{'}$ there is also an old data packet

on A in M_1'. Choose b' corresponding to a successful transmission of this old data packet. Clearly, $b' \approx_{BE} b$ and b' is enabled in M_1'. Let M_2' be such that $M_1'[b\rangle M_2'$. The proof that $M_2' \approx_M M_2$ is similar to the corresponding part of case 1.

Case 4: Assume $is\_old(b) \wedge \neg is\_success(b)$, i.e., b corresponds to a loss of an old data packet. This case is similar to case 3 above.

Let us now consider the three properties listed at the end of Sect. 10.2 which we want to verify for the transport protocol. We want to translate them into appropriate dynamic properties of the CP-net.

For *No improper termination*, we need the concept of a *dead marking*, which is a marking in which no binding elements are enabled. The *No improper termination* property is established if we can verify that in all reachable dead markings of the CP-net, all data packets have been properly received and the network is empty. The reachable dead markings can be derived from the SSE based on the proof rule below. The proof rule states that a reachable marking M is dead if and only if the node in the SSE representing the equivalence class to which M belong is terminal, i.e., has no outgoing arcs.

$$M \in [M_0\rangle \text{ is a dead marking} \Leftrightarrow [M] \text{ is terminal}$$

For *Possibility of termination*, we need the concept of a *home space*, which is a set of markings with the property that from any reachable marking, it is possible to reach a marking of the home space. Home spaces can be derived from the SSE by exploiting the strongly connected component graph (SCC-graph) (see e.g., [48]). The nodes in the SCC-graph are referred to as *strongly connected components*. The set of nodes in the SCC-graph without outgoing arcs are referred to as the terminal strongly components and is denoted SCC_T. The strongly connected components to which a set of nodes X belongs are denoted X^c. Whether the set of markings equivalent with a set of markings $X \subseteq [[M_0\rangle]$ constitutes a home space can be investigated based on the following proof rule.

$$[X] \text{ is a home space} \Leftrightarrow SCC_T \subseteq X^c$$

The possibility of termination property is established if we can verify that there exists a home space containing only dead markings.

For *Eventual termination*, we need the concept of a set of binding elements being *impartial*. This means that elements from the set occur infinitely often in any infinite occurrence sequence starting in the initial marking. We denote by DCS the set of simple directed cycles in the SSE. For a simple directed cycle dc we denote by $BE(dc)$ the set of binding elements determined by the arcs of dc. Impartiality of a set of binding elements can be investigated based on the following proof rule.

$$[X] \subseteq BE \text{ is impartial} \Leftrightarrow \forall dc \in DCS : X \cap BE(dc) \neq \emptyset$$

If the binding elements in the equivalence classes of binding elements corresponding to loss of packets is impartial, then the protocol terminates in all executions where

only finitely many packets are lost. Hence, the *Eventual termination* property is established if the set of binding elements corresponding to loss of packets is impartial.

The three proof rules for SSEs (OE-graphs) listed above and their implementation as query functions in the tool, allowed us to verify the protocol. It turned out that for each investigated value of the system parameters L and N, the generated SSE had exactly one terminal node (a node with no outgoing arcs). The corresponding equivalence class had only one member, namely the marking in which all data packets had been properly received, and all four network places were empty. This equivalence class was a home space, thus the only dead marking was a *home marking*.

Table 10.1 contains the sizes of the SSOs and SSEs for different values of L and a fixed number of data packets $N = 4$. The Ratio columns hold the savings factors, i.e., the figure for the SSO divided by the figure for the SSE. The measures were obtained on a Sun Ultra Sparc Enterprise 3000 computer with 512 MB RAM. An empty entry (-) signals that it was not possible to obtain that measure. It can be seen that SSEs yielded remarkable reductions in the number of nodes and arcs, and that SSEs enabled us to analyse capacities of the network that we could not handle using SSOs.

	Number of Nodes			Number of Arcs		
L	SSO	SSE	Ratio	SSO	SSE	Ratio
1	33	33	1.0	44	44	1.0
2	293	155	1.9	764	383	2.0
3	1,829	492	3.7	6,860	1,632	4.2
4	9,025	1,260	7.1	43,124	5,019	8.6
5	37,477	2,803	11.2	213,902	12,685	16.9
6	136,107	5,635	24.2	891,830	28,044	31.8
7	-	10,488	-	-	56,203	-
8	-	18,366	-	-	104,442	-
9	-	30,605	-	-	182,754	-
10	-	48,939	-	-	304,445	-

Table 10.1: Verification statistics (4 data packets).

The generation and query times (in CPU seconds) are listed in Table 10.2. It can be seen, that from a certain point, generation of the SSE was faster than generation of the SSO. For example, for $L = 6$ the generation time is reduced from approximately 2 hours and 5 minutes to 3 minutes. The query time, i.e., the actual verification of the three considered properties of the transport protocol, was, again from a certain point, significantly faster on the SSE than on the SSO. For example, for $L = 6$ the query time is reduced from approximately 28 minutes to 39 seconds. This is because the queries are made directly on the SSE, and the size of the graph is the critical factor in the time complexity. The sizes of the SSOs and SSEs for different values of N (the number of data packets) is listed in Table 10.3 for a fixed value of $L = 4$. Again, it can be seen that the use of SSEs yielded remarkable reductions in the number of nodes and arcs, and that SSEs enabled us to analyse configurations of the protocol that we could not handle using SSOs. In particular, it is worth noting that nodes and arcs in the SSE grow linearly in the number of packets.

L	Generation Time			Query Time		
	SSO	SSE	Ratio	SSO	SSE	Ratio
1	1	1	1.0	1	1	1.0
2	1	1	1.0	1	1	1.0
3	6	3	2.0	8	2	4.0
4	56	15	3.7	63	6	10.5
5	642	63	10.2	358	17	21.1
6	7,507	190	39.5	1,666	39	42.7
7	-	559	-	-	84	-
8	-	1,546	-	-	167	-
9	-	3,712	-	-	318	-
10	-	8,394	-	-	567	-

Table 10.2: Verification statistics (4 data packets).

N	Number of Nodes			Number of Arcs		
	SSO	SSE	Ratio	SSO	SSE	Ratio
1	120	120	1.0	375	375	1.0
2	885	500	1.7	3,570	1,923	1.8
3	3,336	880	3.8	15,009	3,471	4.3
4	9,025	1,260	7.1	43,124	5,019	8.6
5	20,016	1,640	12.2	99,355	6,567	15.1
6	38,885	2,020	19.3	198,150	8,115	24.4
7	68,720	2,400	28.6	356,965	9,663	36.9
8	-	2,780	-	-	11,211	-
9	-	3,160	-	-	12,759	-
10	-	3,540	-	-	14,307	-

Table 10.3: Verification statistics (network capacity 4).

10.5 Computer-Aided Consistency Proofs

Verification methods based on state spaces are often touted as being automatic and thus quite reliable. For verification based on SSEs, a qualification must be made: Proving the consistency of a proposed equivalence specification may be a tedious task, as demonstrated in Sect. 10.3. Verification based on SSEs is semi-automatic since a manual mathematical proof of consistency has to be conducted, with the risk of making mistakes. In this section we present a technique for alleviating this problem, which can be implemented in a computer tool to support the modeller in ensuring the consistency of a proposed equivalence specification.

The technique is based on a close relationship between a consistent equivalence specification and bisimulation [94]. The underlying idea is to map the SSO and SSE into labelled transition systems between which a bisimulation exists if the equivalence specification is consistent. Below we first develop the necessary theory, and subse-

quent to this we discuss how the result can be supported by a computer tool and applied in practice.

First we give the formal definition of the ordinary full state space of a CP-net. The definition is similar to that of a state space with equivalence classes as given in Def. 16.

Definition 17 *An **ordinary state space** (SSO) for a CP-net is a triple (V, A, N) satisfying the requirements below:*

1. $V = [M_0\rangle$.
2. $A = \{(M_1, B, M_2) \in V \times BE \times V \mid M_1[b\rangle M_2 \}$.
3. $\forall a = (M_1, b, M_2) \in A : N(a) = (M_1, M_2)$. □

Below we define labelled transition systems and bismulations between labelled transition systems.

Definition 18 *A **labelled transition system** (LTS) is a tuple $L = (S, \Sigma, \Delta, s_0)$ where S is a set of **states**, Σ is a set of symbols called the **alphabet**, $\Delta \subseteq S \times \Sigma \times S$ is the **transition relation**, and $s_0 \in S$ is the **initial state**. In the following $(s, \alpha, s') \in \Delta$ is also written as $s \xrightarrow{\alpha} s' \in \Delta$.* □

Definition 19 *Let $L_1 = (S_1, \Sigma, \Delta_1, s_{01})$ and $L_2 = (S_2, \Sigma, \Delta_2, s_{02})$ be two LTSs such that $S_1 \cap S_2 = \emptyset$. A binary relation $\sim \subseteq S_1 \times S_2$ is a **bisimulation**, if and only if the following conditions hold:*

- $s_{01} \sim s_{02}$
- *For every $s \in S_1$ and $s' \in S_2$ such that $s \sim s'$:*
  1. *If $s \xrightarrow{\alpha} s_1 \in \Delta_1$, there exists $s_2 \in S_2$ such that $s' \xrightarrow{\alpha} s_2 \in \Delta_2$ and $s_1 \sim s_2$.*
  2. *If $s' \xrightarrow{\alpha} s_2 \in \Delta_2$, there exists $s_1 \in S_1$ such that $s \xrightarrow{\alpha} s_1 \in \Delta_1$ and $s_1 \sim s_2$.*

*L_1 and L_2 are said to be **bisimilar** (denoted $L_1 \cong_{BS} L_2$) if and only if there exists a bisimulation between L_1 and L_2.* □

In order to map the SSO and SSE into labelled transition systems we introduce two mappings. Given an equivalence specification ES, we denote by $\psi_{BE} \in [BE \mapsto BE]$ a mapping which maps a binding element $b \in BE$ into a unique representative of the equivalence class to which b belongs. Similarly, we denote by $\psi_{[BE]} \in [BE_{\approx} \mapsto BE]$ a mapping which maps an equivalence class $B \in BE_{\approx}$ of binding elements into a unique representative binding element for the equivalence class. The two mappings are required to "agree" on the chosen unique representative by satisfying for a binding element b belonging to an equivalence class $B : \psi_{BE}(b) = \psi_{[BE]}(B)$.

We are now in a position to define the desired mappings of the SSO and SSE into labelled transitions systems according to an equivalence specification ES.

Definition 20 *Let $ES = (\approx_M, \approx_{BE})$ be an equivalence specification for a CP-net. Let $SSO = (V_{SSO}, A_{SSO}, N_{SSO})$ be the ordinary state space, and let $SSE = (V_{SSE}, A_{SSE}, N_{SSE})$ the state space with equivalence classes.*

*The **LTS determined by SSO** is the LTS $L_{SSO} = (V_{SSO}, BE, \Delta_{SSO}, M_0)$ where:*

$$(M_1, b, M_2) \in \Delta_{SSO} \Leftrightarrow (M_1, b', M_2) \in A_{SSO} \wedge b = \psi_{BE}(b')$$

*The **LTS determined by SSE** is the LTS $L_{SSE} = (V_{SSE}, BE, \Delta_{SSE}, [M_0])$ where:*

$$(C_1, b, C_2) \in \Delta_{SSE} \Leftrightarrow (C_1, B, C_2) \in A_{SSE} \wedge b = \psi_{[BE]}(B)$$

□

Both the determined LTSs have the set of binding elements BE as alphabet. The states of the LTS determined by SSO corresponds to the nodes of the SSO, whereas the states of the LTS determined by SSE correspond to the nodes of the SSE. The transition relations of the two LTSs are in a one-to-one correspondence with the arcs of the SSO and the arcs of the SSE, respectively, except for a renaming of the labels according to the functions ψ_{BE} and $\psi_{[BE]}$. The initial states of the LTSs correspond to the node representing the initial marking and the equivalence class containing the initial marking, respectively.

Below we present the result underlying the proposed technique: The LTS determined by SSO and the LTS determined SSE are bisimilar if the equivalence specification is consistent. The result thus gives a necessary condition on a proposed equivalence specification to be consistent.

Theorem 1 *Let $L_{SSO} = (V_{SSO}, BE, \Delta_{SSO}, M_0)$ be the LTS determined by SSO and $L_{SSE} = (V_{SSE}, BE, \Delta_{SSE}, [M_0])$ the LTS determined by SSE according to some equivalence specification ES. Then the following holds:*

$$\textit{ES is a consistent equivalence specification} \Rightarrow L_{SSO} \cong_{BS} L_{SSE}$$

Proof. Assume that ES is a consistent equivalence specification according to Def. 15 and consider the following binary relation $\sim$ on $V_{SSO} \times V_{SSE}$: $M_1 \sim [M_2] \Leftrightarrow M_1 \in [M_2]$. We prove that $\sim$ is a bisimulation between L_{SSO} and L_{SSE} according to Def. 19.

Clearly $M_0 \sim [M_0]$ since $M_0 \in [M_0]$. Let $M \in V_{SSO}$, $C \in V_{SSE}$ be such that $M \sim C$, and assume that $M \xrightarrow{b} M_1 \in \Delta_{SSO}$. By the definition of L_{SSO} there exists $(M, b', M_1) \in A_{SSO}$ such that $b = \psi_{BE}(b')$. By the definition of an SSE and the consistency of the equivalence specification there exists $(C_1, B, C_2) \in A_{SSE}$ such that $M \in C_1$, $M_2 \in C_2$ and $b' \in B$. Since $M \in C_1$ and a marking belongs to only one equivalence class, $C = C_1$. Hence $(C, B, C_2) \in A_{SSE}$. The definition of L_{SSE} now implies that $C \xrightarrow{b} C_2 \in \Delta_{SSE}$, since $\psi_{[BE]}(B) = b$ by the requirement on ψ_{BE} and $\psi_{[BE]}$. Now $M_1 \in C_2$ implies that $M_1 \sim C_2$. Hence condition 1 in the definition of bisimulation is satisfied by $\sim$. The proof that $\sim$ satisfies condition 2 is similar. □

At first the results may not seem particularly applicable in practice because it relies on the SSO to be generated in order to apply the result – what we essentially want to avoid by generating a reduced state space such as an SSE. However, many systems are such that it is possible for small configurations of the system to generate the SSO. This is for instance the case with the transport protocol studied in this paper. Any algorithm for checking bisimilarity of LTSs can then be applied as a necessary condition on the equivalence specification to be consistent for small configurations of the system. If the LTSs determined by the SSO and the SSE are *not* bisimilar, then the equivalence specification *cannot* be consistent.

In order to apply the theorem on the transport protocol, functionality has been added to the condensed state space tool which makes it possible to automatically generate L_{SSO} and L_{SSE} from the SSO and the SSE. We have then used the bisimulation tool of the Edinburgh Concurrency Workbench (ECW) [96] for checking bisimilarity of the two generated LTSs. This was done by generating the two LTSs as CCS expressions which is the input language of the ECW tool. For all configurations of the transport protocol in which the SSO could be generated it turned out, that the determined L_{SSO} and L_{SSE} were bisimilar. A future version of the tool might include the implementation of a bisimulation algorithm such that it is not necessary to rely on other tools as a back-end for checking bisimilarity.

Alternatively, the theorem above can be used in a slightly different way by observing that if the equivalence specification is consistent then the binary relation $\sim$ defined in the proof above is a bisimulation. Hence, instead of testing whether a bismulation exists between the determined LTSs, another possibility is to explicitly test whether $\sim$ defines a bisimulation. This approach gives a stronger requirement but has the disadvantage of relying on the two predicates provided by the user implementing the equivalence specification. In practice one would therefore apply the theorem in both ways, since the two applications in a sense complement each other. In conclusion the result above provides the modeller with a fast and automatic necessary check on the consistency of a proposed equivalence specification. This can be of good help to locate possible mistakes.

A question is of course how strong the check above is. During the case study of the transport protocol we experimented with getting a weaker equivalence specification than the one presented in the previous sections. This would have resulted in an even better reduction of the state space. One idea which came up was to weaken the requirements with respect to old packets such that instead of requiring the same number of old packets on the network places it is only required that if there are some old packets on a network place, say place A, in M_1 then there should also be some old packets on place A in M_2 in order for the markings M_1 and M_2 to be equivalent. This equivalence specification is however *not* consistent, and the necessary check above is able to reveal this, since it turns out that the determined LTSs are not bisimilar.

Below we demonstrate that the check based on bismulation can only be used as a necessary condition on the equivalence specification to be consistent. We do so by means of an example showing that there exist equivalence specifications which are not consistent but for which the determined LTSs are bisimilar. The left-hand side of Fig. 10.2 shows a simple CP-net consisting of a single transition T and two places A and B. The colour set E consists of the single element e. For this CP-net we consider the equivalence specification in which all binding elements are considered equivalent

and two markings are considered equivalent if they are the same when the marking of place A is ignored.

The determined LTSs are shown in the middle and on the right hand side of Fig. 10.2. As it can be seen these two LTSs are clearly bisimilar. However the equivalence specification is not consistent. To see this consider the two markings $M_0, M' \in [[M_0\rangle]$ where $M'(A) = M'(B) = empty$. M_0 is equivalent to M' and $M_0[(T, \langle\rangle)\rangle$, but M' is a dead marking and therefore the equivalence specification is not consistent.

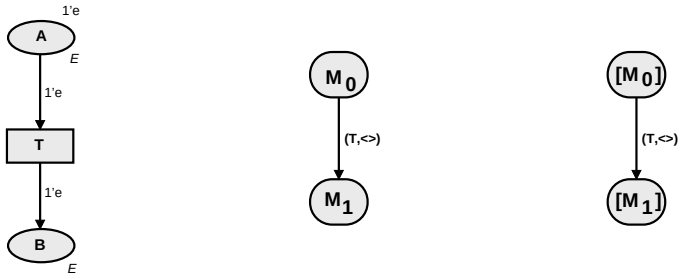

Figure 10.2: CP-net (left) and LTSs determined by SSO (middle) and SSE (right).

10.6 Conclusions

The motivation to write this paper came from our work with developing tool support for SSEs [78]. Our prime focus was on state spaces with symmetries (OS-graphs with permutations for CP-nets as defined in [67]), because their usefulness was already recognised. The journal [35] contains four papers [23, 34, 65, 68] that all demonstrate the potential of using symmetry in state space verification. A common denominator for the four papers is that symmetry is conceived as a structural property, described by permutations of similar components.

The generality of SSEs allowed us to experiment with different kinds of user-supplied equivalence relations. During these experiments we realised the new perspective of SSEs. In this paper, we saw that SSEs allow equivalences that are dynamic, in the sense that they express that some information becomes irrelevant as the execution of a system progresses. Furthermore, the use of SSEs allowed us to verify configurations of the considered protocol which could not be handled using ordinary full state spaces. One of the main potentials of verification by means of SSEs is that it allows verification of larger configurations of the system under consideration. This is in particular important in systems which have several parameters. Due to state explosion it may only be possible to verify the system by means of full state spaces when almost all parameters have small values. By applying SSEs it is possible to verify configurations of the system where several of the parameters have large values.

An interesting question is of course, whether the kind of equivalence specifications presented in this paper generalises, i.e., apply to other systems. We believe they do. We believe that the notion of being *old* of the example can be found in various disguises in many systems in particular for protocols on the transport layer of the ISO reference

model. Such communication protocols are often designed using standard techniques like timers, retransmissions, and sequence numbers, thereby inherently introducing the concept of an old packet. We are currently conducting additional case studies on the use of SSEs within this domain, which includes sliding window protocols.

The complexity of the consistency proof is a drawback of verification based on SSEs. However, the confidence in the consistency of an equivalence specification can be highly increased with the aid of the condensed state space tool using the techniques discussed in this paper. The techniques presented relied on the possibility of generating the ordinary state space for small configurations of the system and thus comply with the observation that the main potential of verification based on SSEs is the verification of large configurations of systems. Future work includes investigating how the proof of consistency can be further computer-aided.

In state spaces with symmetries (OS-graphs with permutations of [67]), proving consistency of a proposed specification can be done by a trivial analysis of all static inscriptions of the CP-net. No ingenuity is required, and the proof can be highly computer-aided. In the approaches to symmetry of [23, 34], it is the responsibility of the user to define the symmetries of the system and ensure their consistency. In contrast, [65] presents a procedure which automatically detects the symmetries of a system. The basic idea is to impose narrow syntactical restrictions on the modelling language ensuring that only symmetric constructions are expressible. Similar ideas have been used for Well-formed Petri Nets in [13]. The work [13] has been further developed in [55] making it possible to handle systems which have both symmetric and asymmetric parts. It would be of interest to investigate whether the partial symmetries suggested in [55] can be used to capture the notion of old packets of the transport protocol.

In the condensed state space tool the user makes queries using a number of built-in query functions implementing proof rules for SSEs. These query functions cover the standard dynamic properties of CP-nets such as bounds on places, liveness and fairness of transitions etc. However, SSEs also makes it possible to use temporal logics such as CTL or LTL [1, 31, 64] as query language. The Design/CPN ASK-CTL Library [20] which can be used with the condensed state space tool makes it possible for the user to write queries in an action- and state oriented variant of CTL.

Another issue which deserves attention in future work is the combination of state spaces with equivalence classes and other reduction methods, such as partial order reduction methods. It was shown in [32] that partial order and symmetry reduction can be combined in such a way that the truth value of CTL and LTL formulas are preserved. It is obvious to pursue similar ideas in the framework of state spaces with equivalences classes. In conclusion, we do believe that the observations made and the reductions exhibited in this paper are very encouraging for verification based on SSEs.

Chapter 11

Finding Stubborn Sets of Coloured Petri Nets Without Unfolding

The paper *Finding Stubborn Sets of Coloured Petri Nets Without Unfolding* constituting this chapter has been published as a conference paper [86].

[86] L. M. Kristensen and A. Valmari. Finding Stubborn Sets of Coloured Petri Nets Without Unfolding. In Proceedings of 19th International Conference on Application and Theory of Petri Nets (ICATPN'98). J. Desel and M. Silva (Eds). Volume 1420 of Lecture Notes in Computer Science, pp. 104-123, Springer-Verlag, 1998.

The content of this chapter is equal to the conference paper [86] except for minor typographical changes.

Finding Stubborn Sets of Coloured Petri Nets Without Unfolding

L. M. Kristensen* A. Valmari†

Abstract

In this paper, we address the issue of using the stubborn set method for Coloured Petri Nets (CP-nets) without relying on unfolding to the equivalent Place/Transition Net (PT-net). We give a lower bound result stating that there exist CP-nets for which computing "good" stubborn sets requires time proportional to the size of the equivalent PT-net. We suggest an approximative method for computing stubborn set of process-partitioned CP-nets which does not rely on unfolding. The underlying idea is to add some structure to the CP-net, which can be exploited during the stubborn set construction to avoid the unfolding. We demonstrate the practical applicability of the method with both theoretical and experimental case studies, in which reduction of the state space as well as savings in time are obtained.

Topics: System design and verification using nets, Analysis and synthesis, Higher-level net models, Computer tools for nets.

11.1 Introduction

State space methods have proven powerful in the analysis and verification of the behaviour of concurrent systems. Unfortunately, the sizes of state spaces of systems tend to grow very rapidly when systems become bigger. This well-known phenomenon is often referred to as *state explosion*, and it is a serious problem for the use of state space methods in the analysis of real-life systems.

Many techniques for alleviating the state explosion problem have been suggested, such as the *stubborn set method* [120, 127]. It is one of a group of rather similar methods first suggested in the late 80's and early 90's [49, 50, 101, 102]. It is based on the fact that the total effect of a set of concurrent transitions is independent of the order in which the transitions are executed. Therefore, it often suffices to investigate only one or some orderings in order to reason about the behaviour of the system.

In stubborn set state space generation an analysis of the dependencies between transitions is made at each state, and only certain transitions are used to generate immediate successor states. The "stubborn set" is the set of these transitions, together

*Department of Computer Science, University of Aarhus, DK-8000 Aarhus C., DENMARK. E-mail: lmkristensen@daimi.au.dk.

†Tampere University of Technology, Software Systems Laboratory, PO Box 553, FIN-33101 Tampere, FINLAND. E-mail: ava@cs.tut.fi.

with some disabled transitions. The disabled transitions have no significance, but are included in the stubborn set for technical reasons. The remaining transitions are either taken into account in some subsequent states, or the situation is such that they can be ignored altogether without affecting analysis results. The set of transitions that is investigated in a given state depends on two factors: dependencies between transitions such as conflict (both transitions want to consume the same token), and the properties that are to be checked of the system. In this paper we concentrate on the first factor.

In the field of Petri nets, stubborn sets have been applied mostly to elementary and Place/Transition Nets (PT-nets). This is because a transition of a high-level Petri net such as a *Coloured Petri Net* [66] (*CP-net* or *CPN*), is really a packed representation of several low-level transitions, in CP-net terminology referred to as *binding elements*. The dependency analysis needed by the stubborn set method is difficult with high-level nets, because, for instance, a high-level transition may simultaneously have a binding element that is concurrent and another binding element that is in conflict with a binding element of some other high-level transition. In [125] this problem was avoided by effectively unfolding the CP-net during the construction of stubborn sets. However, the unfolded form of a high-level net may be much bigger than the high-level net itself and may even be infinite. As a consequence, unfolding may be very time-consuming and should be avoided. An algorithm based on constraint systems for alleviating the impact of unfolding has been given for Well Formed Coloured Petri Nets in [7].

An alternative stubborn set construction for high-level net would be to treat each high-level transition as a unit and consider a high-level transition t_2 as dependent on another high-level transition t_1, unless it is certain that no binding element of t_2 depends on any binding element of t_1. In essence, this strategy replaces the detailed low-level dependencies by high-level dependencies that approximate the low-level dependencies from above. Such approximations do not affect the correctness of the results obtained with stubborn sets, but they tend to make the stubborn sets bigger and weaken the reduction results. In our experience, the reduction results obtained with this coarse strategy have usually been very bad.

Efficient construction of "good" stubborn sets of high-level nets seems thus to require more information than can be obtained from the structure of the high-level net without unfolding, but some approximation from above has to be made in order to avoid unfolding too much. In this paper we suggest such a strategy, and demonstrate its power with a couple of examples. The new method is based on adding some structure to the high-level net. The high-level net is divided into disjoint subnets, such that each subnet corresponds either to a set of parallel processes executing the same code or to a variable through which two or more processes communicate (a fifo queue, for instance). Stubborn set construction uses knowledge of this structure in order to prevent the stubborn sets from becoming too big. When dependencies between binding elements have to be analysed, the method approximates from above to avoid unfolding. We will present our method in the framework of CP-nets, but the same ideas should also be applicable to most other high-level net formalisms.

The paper is organised as follows. Section 11.2 recalls the basic facts of CP-nets and stubborn sets that are needed to understand the rest of this paper. In Sect. 11.3 we will prove a theorem that, in essence, says that sometimes "good" stubborn sets cannot be constructed without the cost of unfolding. The structure we add to CP-nets is described in Sect. 11.4. Our new method is given in Sect. 11.5 and is illustrated

with an annotated example in Sect. 11.6. Section 11.7 gives some numerical data on the performance of the new method on some case studies. Section 11.8 contains the conclusions and some directions for future work.

11.2 Background

This section summarises the basic facts of CP-nets and stubborn sets needed to understand the rest of the paper. The definitions and notation we will use for CP-nets are given in Section 11.2.1, and they follow closely [66] and [67]. Section 11.2.1 is not much more than a list of notation, so we assume that the reader is familiar with PT- and CP-nets, their dynamic behaviour, and the unfolding of a CP-net to a PT-net. Section 11.2.2 introduces the necessary background on stubborn sets.

11.2.1 Coloured Petri Nets

A **multi-set** ms over a domain X is a function from X into the set of natural numbers. A multi-set ms is written as a formal sum like $\sum_{x \in X} ms(x)'x$, where $ms(x)$ is the number of occurrences of the element x in ms. We assume that addition ($+$), subtraction ($-$), multiplication by a scalar, equality ($=$), and comparison ($\leq$) are defined on multi-sets in the usual way. $|ms|$ denotes the size of the multi-set ms, i.e., the total number of elements with their multiplicities taken into account. S_{MS} denotes the set of multi-sets over a domain S.

A **CP-net** [66] is a tuple $CPN = (\Sigma, P, T, A, N, C, G, E, I)$ where Σ is a set of **colour sets**, P is a set of **places**, T is a set of **transitions**, and A is a set of **arcs**. N is a **node function** designating for each arc a **source** and **destination**. C is a **colour function** mapping each place p to a colour set $C(p)$ specifying the type of **tokens** which can reside on p. G is a **guard function** mapping each transition t to a boolean expression $G(t)$. E is an **arc expression function** mapping each arc a into an expression $E(a)$. Finally, I is an **initialisation function** mapping each place p to a multi-set $I(p)$ of type $C(p)_{MS}$ specifying the initial marking of the place p.

A **token element** is a pair (p, c) such that $p \in P$ and $c \in C(p)$. For a colour set $S \in \Sigma$, the **base colour sets** of S are the colour sets from which S was constructed using some structuring mechanism such as cartesian product, record, or union.

For $x \in P \cup T$ the **postset** of x, denoted $Out(x)$, is the set: $\{x' \in P \cup T \mid \exists a \in A : N(a) = (x, x')\}$. Similarly, the **preset** of x denoted $In(x)$ is the set: $\{x' \in P \cup T \mid \exists a \in A : N(a) = (x', x)\}$.

Since it is possible to have several arcs between a place and a transition and vice versa, we denote by $A(x_1, x_2)$ for $(x_1, x_2) \in (P \times T) \cup (T \times P)$ the set of arcs from x_1 to x_2, and define the **expression of** (x_1, x_2) as: $E(x_1, x_2) = \sum_{a \in A(x_1, x_2)} E(a)$.

The set of **variables** of a transition $t \in T$ is denoted $Var(t)$. For a variable $v \in Var(t)$, $Type(v) \in \Sigma$ denotes the **type** of v. A **binding element** (t, b) is a pair consisting of a transition t and a **binding** b of data values to its variables such that $G(t)\langle b \rangle$ evaluates to true. For an expression $expr$, $expr\langle b \rangle$ denotes the value obtained by evaluating the expression $expr$ in the binding b. A binding element is written in the form $(t, \langle v_1 = c_1, v_2 = c_2, \ldots, v_n = c_n \rangle)$, where $v_1, \ldots, v_n \in Var(t)$ are the variables of t and $c_1, \ldots, c_n$ are data values such that $c_i \in Type(v_i)$ for $1 \leq i \leq n$.

For a binding element (t, b) and a variable v of t, $b(v)$ denotes the value assigned to v in the binding b. $B(t)$ denotes the set of all bindings for t. The set of all binding elements is denoted BE.

In a given **marking** M of a CP-net, the marking of a place p is denoted $M(p)$. M_0 denotes the **initial marking**. If a binding element (t, b) is **enabled** in a marking M_1 (denoted $M_1[(t, b)\rangle$), then (t, b) may **occur** in M_1 yielding some marking M_2. This is written $M_1[(t, b)\rangle M_2$. Extending this notion, an **occurrence sequence** is a sequence consisting of markings M_i and binding elements (t_i, b_i) denoted $M_1[(t_1, b_1)\rangle M_2 \ldots M_{n-1}[(t_{n-1}, b_{n-1})\rangle M_n$ and satisfying $M_i[(t_i, b_i)\rangle M_{i+1}$ for $1 \leq i < n$. A **reachable marking** is a marking which can be obtained (reached) by an occurrence sequence starting in the initial marking. $[M_0\rangle$ denotes the set of reachable markings.

Below we define place weights, place flows and place invariants. The definition is identical to Def. 4.6 in [67] except that we define the weights to map only between multi-sets. This is done for simplicity reasons, since we do not need the more general notion of weighted-sets. For two sets A and B the set of linear functions from A to B is denoted $[A \rightarrow B]_L$.

Definition 21 *([67], Def. 4.6) For a CP-net CPN a **set of place weights** with range $A \in \Sigma$ is a set of functions $W = \{W_p\}_{p \in P}$ such that $W_p \in [C(p)_{MS} \rightarrow A_{MS}]_L$ for all $p \in P$.*

1. *W is a **place flow** iff:*

$$\forall (t, b) \in BE : \sum_{p \in P} W_p(E(p, t)\langle b\rangle) = \sum_{p \in P} W_p(E(t, p)\langle b\rangle)$$

2. *W determines a **place invariant** iff:*

$$\forall M \in [M_0\rangle : \sum_{p \in P} W_p(M(p)) = \sum_{p \in P} W_p(M_0(p))$$ □

The following theorem is central to place invariant analysis of CP-nets. It states that the static property of Def. 21 (1) is sufficient to guarantee the dynamic property of Def. 21 (2).

Theorem 2 *([67], Theorem 4.7) W is a place flow ⇒ W determines a place invariant.* □

11.2.2 Stubborn Sets

State space construction with stubborn sets follows the same procedure as the construction of the full state space of a Petri net, with one exception. When processing a marking, a set of transitions (or binding elements in the case of a CP-net), the so-called **stubborn set**, is constructed. Only the enabled transitions (binding elements) in it are used to construct new markings. This reduces the number of new markings, and may lead to significant reduction in the size of the state space. To get correct analysis results, stubborn sets should be chosen such that the state space obtained with them (from now on called **SS state space**) preserves certain properties of the full state space.

The choice of stubborn sets thus depends on the properties that are being analysed or verified of the system. This has led to the development of several versions of the stubborn set method. However, it is common to almost all of them that the following theorem should hold:

Theorem 3 *Let M be any marking of the net, $Stub$ a stubborn set in M, $n \geq 0$, $t \in Stub$, and $t_1, t_2, \ldots, t_n \notin Stub$.*

1. *If $M\ [t_1\rangle\ M_1\ [t_2\rangle\ \ldots\ [t_{n-1}\rangle\ M_{n-1}\ [t_n\rangle\ M_n\ [t\rangle\ M'_n$, then $M\ [t\rangle$.*
2. *If $M\ [t_1\rangle\ M_1\ [t_2\rangle\ \ldots\ [t_{n-1}\rangle\ M_{n-1}\ [t_n\rangle\ M_n$ and $M\ [t\rangle\ M'$, then there are M'_1, $M'_2, \ldots, M'_n$ such that $M'\ [t_1\rangle\ M'_1\ [t_2\rangle\ \ldots\ [t_n\rangle\ M'_n$, and $M_n\ [t\rangle\ M'_n$.* □

It is also required that if M_0 is not a dead marking (a marking without enabled transitions), then $Stub$ contains at least one enabled transition (binding element).

From this theorem it is possible to prove that the SS state space contains all the dead markings of the full state space. Furthermore, if the full state space contains an infinite occurrence sequence, then so does the SS state space. By adding extra restrictions to the construction of stubborn sets, the stubborn set method can be made to preserve more properties, but that topic is beyond our present interest. With PT-nets, Theorem 3 holds if stubborn sets are defined as follows:

Definition 22 *Let (P, T, A, W, I) be a PT-net. The set $Stub \subseteq T$ is* stubborn *in marking M, if the following hold for every $t \in Stub$:*

1. *If $\exists t_1 \in T : M\ [t_1\rangle$, then $\exists t_2 \in Stub : M\ [t_2\rangle$.*
2. *If $\neg M\ [t\rangle$, then $\exists p \in {\bullet}t : M(p) < W(p,t)\ \wedge\ {\bullet}p \subseteq Stub$.*
3. *If $M\ [t\rangle$, then $({\bullet}t){\bullet} \subseteq Stub$.* □

Because this definition analyses the dependencies between transitions at a rather coarse level, it is not an "optimal" definition in the sense of yielding smallest possible stubborn sets and smallest SS state spaces, but we will use it in the following because of its simplicity. Once the basic ideas of our new CP-net stubborn set construction method are understood, they can be applied to more detailed dependency analysis if required.

Definition 22 gives a condition with which one can check whether a given set of transitions is a stubborn set in a given marking. Part (1) says that unless the marking is a dead marking, the stubborn set should contain at least one enabled transition. Parts (2) and (3) can be thought of as rules that, given a transition t that is intended to be in the stubborn set, produce a set of other transitions that must be included. In the case of (3), the set is just $({\bullet}t){\bullet} \subseteq Stub$. Part (2) requires the selection of some place $p \in {\bullet}t$, such that p contains fewer tokens than t wants to consume, and then produces the set ${\bullet}p$. If there are several such places, most stubborn set algorithms just make an arbitrary choice between them. A somewhat expensive algorithm that investigates all choices for p is explained in [121].

Important for the rest of this paper is that the rules can be thought of as spanning a *dependency graph*: the nodes of the graph are the transitions, and there is an edge from t_1 to t_2 if and only if the above rules (with a fixed arbitrary choice of the p in

(2)) demand that if t_1 is in the stubborn set, then also t_2 must be. A stubborn set then corresponds to a set of transitions that contains an enabled transition and that is closed under reachability in the dependency graph. Therefore, to construct a stubborn set, it suffices to know the dependency graph and the set of enabled transitions.

In this paper we will not actually give any concrete algorithm for finding stubborn sets of CP-nets. Instead, we describe a method for obtaining a "good" dependency graph, from which one can construct "good" stubborn sets with the old algorithms that rely on dependency graphs.

11.3 The Necessity of Unfolding

Because every CP-net can be unfolded to an equivalent PT-net, and because good dependency graphs for PT-nets are known, one can always construct a stubborn set of a CP-net by first unfolding it to a PT-net. Unfolding is, however, often expensive, so one wants to avoid it. We will demonstrate in this section that, unfortunately, there are situations where good stubborn sets cannot be constructed — not even named, as a matter of fact — without unfolding or doing something equally expensive. We will do that by analysing the behaviour of the CP-net in Fig. 11.1. The CP-net has 9 places, 8 transitions, and all but two of its places have colour set $N = \{1, 2, \ldots, n\}$. The remaining places p_5 and p_8 have a colour set containing only one element (colour) denoted `()`. The variable `x` is of type N. In the initial marking place p_1 contains the tokens with colour $1 \ldots n$. The remaining places are initially empty.

Let H be any subset of `N`, and let M_H be the marking where $M(p_1) = M(p_5) = M(p_8) = M(p_9) = \emptyset$, $M(p_2) = M(p_6) = H$, $M(p_3) = M(p_7) = N - H$, and $M(p_4) = N$. This marking can be reached from the initial marking by letting t_1 and t_2 occur with suitable bindings followed by the occurrence of t_7. We will consider the stubborn sets in M_H obtained by unfolding the CP-net to a PT-net, and then using Def. 22.

In M_H, all binding elements of t_5 are enabled, and they are the only enabled binding elements in M_H. Assume that a binding element $(t_5, \langle x = h \rangle)$ where $h \in H$ is in a stubborn set $Stub$. Rule (3) of Def. 22 forces us to include the binding elements $(t_4, \langle x = h \rangle)$ and $(t_6, \langle x = h \rangle)$ into $Stub$. The binding element $(t_6, \langle x = h \rangle)$ is disabled exactly because there is no token of colour h in p_7. So rule (2) forces the inclusion of $(t_2, \langle x = h \rangle)$ into the stubborn set. Rule (2) should then be applied to $(t_2, \langle x = h \rangle)$, but this does not make the stubborn set grow any more, because the only input place of t_2 has no input transitions.

The binding element $(t_4\langle x = h \rangle)$ is disabled because there is no token on p_8. Rule (2) of Def. 22 forces us to include the binding elements $(t_3, \langle x = k \rangle)$ into $Stub$, where $k \in N$. The binding elements $(t_3, \langle x = k \rangle)$ are disabled because p_2 and p_3 do not contain tokens with the same colour. For those values of k that are not in H, rule (2) takes the analysis through the token element (p_2, k) to t_1 but not to anywhere else. But when the value of k is in H, the analysis proceeds through (p_3, k) to $(t_5, \langle x = k \rangle)$. So we see that $Stub$ must contain all the binding elements $(t_5, \langle x = k \rangle)$ where $k \in H$. On the other hand, the set consisting of those binding elements together with certain disabled binding elements satisfies Def. 22, and is thus stubborn in M_H.

Assume now that $Stub$ contains a binding element $(t_5, \langle x = h \rangle)$ where $h \notin H$.

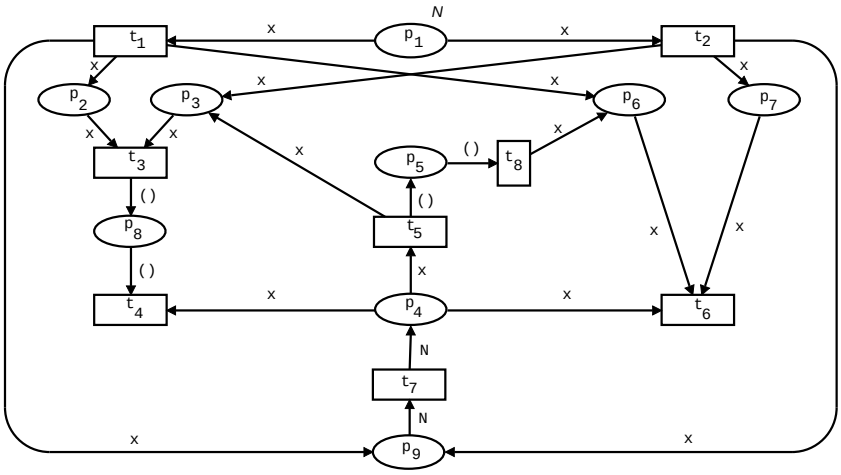

Figure 11.1: CP-net demonstrating the necessity of unfolding.

Rule (3) leads to $(t_6, \langle x = h \rangle)$, from which rule (2) takes us through (p_6, h) and further through p_5 to $(t_5, \langle x = k \rangle)$ for every $k \in N$. As a conclusion, $Stub$ must contain all enabled binding elements.

There are thus only two possibilities for the stubborn set in M_H: either the stubborn set consists of the binding elements $(t_5, \langle x = k \rangle)$ where $k \in H$ plus some disabled binding elements, or the stubborn set contains all enabled binding elements. The existence of the above CP-net implies the following lower bound result.

Theorem 4 *The size of the equivalent PT-net PTN is a lower bound on the worst-case time complexity of any algorithm that computes non-trivial stubborn sets (if they exist) according to Def. 22, in all markings encountered during the SS state space construction of a CP-net CPN.*

Proof. The argument preceeding the theorem demonstrated the existence of a CP-net and a marking M_H with two possible stubborn sets: either the stubborn set consists of the binding elements $(t_5, \langle x = k \rangle)$ where $k \in H$ plus some disabled binding elements, or the stubborn set contains all enabled binding elements. The latter is the trivial stubborn set, so the stubborn set construction algorithm should find the former set. But, depending on the history of the CP-net, H may be just any subset of N. Since $|N| = n$, the algorithm has to deliver at least n bits to be able to unambiguously specify its answer. To do that it needs $\Omega(n)$ time. However, the CP-net is of constant size (or of size $\Theta(\log n)$, if you want to charge the bits that are needed to specify n). Since the size of the equivalent PT-net obtained by unfolding the CP-net in Fig. 11.1 is $\Theta(n)$, constructing a non-trivial stubborn set requires at least time proportional to the unfolding.

We are left with proving that any such algorithm for SS state space construction has to consider the markings M_H for all possible choices of $H \subseteq N$. It suffices to prove that M_H is contained in the SS state space when choosing the stubborn sets with the fewest possible enabled binding elements, since choosing larger stubborn sets will only add markings to the SS state space. Because $(t_1, \langle x = k \rangle)$ and $(t_2, \langle x = k \rangle)$ are

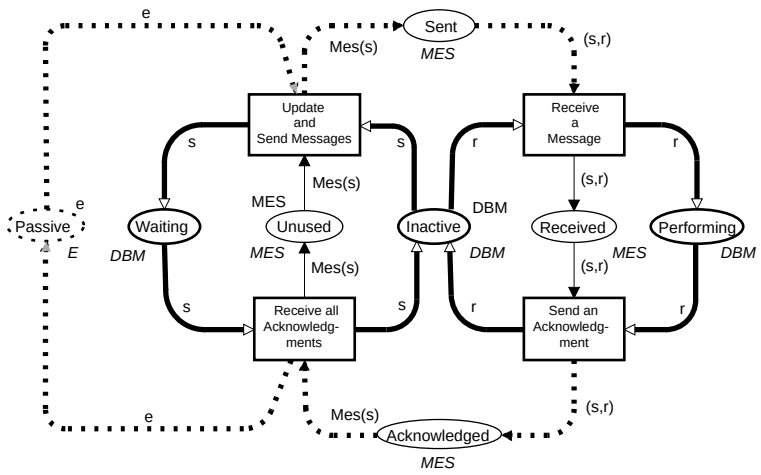

Figure 11.2: CPN model of the data base system

in conflict for every $k \in N$ it is relatively straightforward to check that every SS state space of the CP-net relying on Def. 22 contains the markings M_H for all $H \subseteq N$. □

It is worth observing that in the above construction it already takes n bits to describe M_H, so the cost of unfolding is not a major factor of the total cost of state space construction for the CP-net in Fig. 11.1. Even so, the example demonstrates that the construction of non-trivial stubborn sets sometimes requires analysis at the level of unfolding.

11.4 Process-Partitioned CP-nets

In this section we explain our new method for computing stubborn sets of CP-nets. The method is first explained in an informal way and then followed by the formal definitions. Before that, we introduce an example system used to clarify the definitions.

11.4.1 The Data Base Example System

The distributed data base system from [66], depicted in Fig. 11.2, is used as a running example throughout this and subsequent sections.

The CP-net describes the communication between a set of data base managers maintaining consistent copies of a data base in a distributed system. The states of the managers are modelled by the three places Waiting (for acknowledgements), Inactive, and Performing (an update requested by another manager). The managers are modelled by the colour set DBM $= \{d_1, \ldots, d_n\}$ where n is the number of managers. The messages in the system are modelled by the colour set MES. A message is a pair consisting of a sender and a receiver. In Fig. 11.2, the names *DBM*, *E*, and MES in italics positioned next to the places denote the colour sets of places. E denotes the colour set consisting of a single element e.

The actions of the managers are modelled by the four transitions. Update and Send Messages (SM) models a manager updating its copy of the data base and sending a message to every other manager, so that it can perform the same update on its copy. Receive a Message (RM) models a manager receiving a request for updating its copy of the data base, and Send an Acknowledgement (SA) models the sending of an acknowledgement message after a requested update has been performed. Receive all Acknowledgements models the manager receiving the acknowledgements sent back by the other managers. To maintain consistency between the copies of the data base, the place Passive ensures mutual exclusion for updating the data base. Initially, all managers are on Inactive and all messages are on Unused. This is shown by the initial markings MES and DBM positioned next to the places Unused and Inactive. The initial marking of place Passive is the multi-set $1'e$. The initial markings of initially empty places are omitted in the figure.

11.4.2 Informal Explanation

For the construction of stubborn sets, we will distinguish one or more subnets of the CP-net which we will call **process subnets**. The process subnets may be connected to each other by sharing common **border places**, but are otherwise disjoint. Together the process subnets contain all the transitions and places of the CP-net. A process subnet models the states and actions of one or more processes that run the same program code. The data base system has only one process subnet.

Each transition in the CP-net belongs to some unique process subnet. We require that each transition has a distinct variable, which, when bound in an occurrence of a binding element of that transition, identifies the process executing the action modelled by the transition. We will call this variable the **process variable**. In the data base system, SM and RA have the process variable s, whereas RM and SA have the process variable r. This will allow us to make a disjoint partitioning of the binding elements of a transition according to the following definition.

Definition 23 *Let pv_t be the* **p***rocess* **v***ariable of a transition $t \in T$, and let $c \in Type(pv_t)$. The c-**binding-class of t** denoted $t[pv_t = c]$ is the following set of binding elements: $\{(t,b) \in BE \mid b(pv_t) = c\}$.* □

The term **binding class** will be used when the particular choice of c is not important.

There are three types of places in process subnets: process places, local places and border places.

Process places are used to model the control flow of the processes. In the data base system the places Waiting, Inactive, and Performing are process places. Each token residing on such a place is assumed to have a colour which identifies the corresponding process, and is referred to as a **process token**. When we have a specific process in mind, identified by the colour c, we will talk about the c-**process-token**.

We assume that in any reachable marking there is exactly one c-process-token present in a given process subnet for a given c. This corresponds to a process having only one point of control. Therefore, each transition has at least one input and at least one output process place (process place connected to an incoming / outgoing arc). The arc expressions should ensure that an occurrence of a binding element in

the c-binding-class of a transition removes exactly one c-process-token from its input process places, adds exactly one c-process-token to its output process places, and does not affect c'-process-tokens where $c' \neq c$. Because of this, a process token residing on a process place determines one binding class of each of its output transitions, namely the c-binding-class which can remove the process token. We will therefore talk about the **corresponding binding classes** of a process token residing on a process place. For instance, in the initial marking of the data base system, the corresponding binding classes of the d_1-process token on Inactive are: the d_1-binding-class of SM, that is, $SM[s = d_1]$; and the d_1-binding-class of RM, that is, $RM[r = d_1]$.

Local places are used to model state information local to a process. Intuitively, a token residing on such a place can only be removed by a specific process, and a token added by one process cannot be removed by another process. In the data base system the local places in the process subnet are: Unused (which a data base manager uses to store unused messages) and Received (which a data base manager uses to temporarily store a received message).

The **border places** connect the process subnets, and model asynchronous communication between processes, including communication between processes in the same process subnet. There are two kinds of border places: **shared places** and **buffer places**. A token residing on a shared place may be removed by several processes, whereas a token residing on a buffer place may only be removed by a specific process. In the data base system, there are two buffer places: Sent and Acknowledged, and one shared place: Passive.

11.4.3 Formal Definitions

We now present the formal definitions of the concepts informally introduced in the previous section. First we give the definition of a process subnet of a CP-net. An explanation of the individual parts of the definition is given below.

Definition 24 *A **process subnet** is a tuple $(CPN, P_{pr}, P_{loc}, P_{bor}, P_{buf}, \Xi, PV, PrId)$, where*

1. *$CPN = (\Sigma, P, T, A, N, C, G, E, I)$ is a CP-net.*
2. *$P_{pr} \subseteq P$ is a set of **process places**, $P_{loc} \subseteq P$ is a set of **local places**, and $P_{bor} \subseteq P$ is a set of **border places** such that:*

$$P_{pr} \cap P_{loc} = P_{pr} \cap P_{bor} = P_{loc} \cap P_{bor} = \emptyset \text{ and } P = P_{pr} \cup P_{loc} \cup P_{bor}.$$

3. *$P_{buf} \subseteq P_{bor}$ is a set of **buffer places**.*
4. *$\Xi \in \Sigma$ is a common base colour set of $C(p)$ for all $p \in P_{pr} \cup P_{loc} \cup P_{buf}$.*
5. *PV is a function associating with each transition $t \in T$ a **process variable** $PV(t) = pv_t \in Var(t)$ such that $Type(pv_t) = \Xi$.*
6. *$PrId = \{PrId_p\}_{p \in P}$ is a set of place weights with range Ξ such that for $p \in P_{pr} \cup P_{loc} \cup P_{buf}$, $PrId_p$ projects a multi-set over $C(p)$ into a multi-set over the common base colour set Ξ (cf. item 4) and maps any multi-set into the empty*

*multi-set on the remaining places in P. The colour $PrId_p(c)$ is the **process identity** of the token element (p, c).*

7. *In the initial marking there is exactly one token with a given colour in Ξ on the process places of the process subnet:*

$$\sum_{p \in P_{pr}} PrId_p(M_0(p)) = \Xi \tag{11.1}$$

8. *The following equations hold for all transitions $t \in T$ and $b \in B(t)$:*

$$\sum_{p \in P_{pr}} PrId_p(E(p,t)\langle b \rangle) = \sum_{p \in P_{pr}} PrId_p(E(t,p)\langle b \rangle) = 1'(b(pv_t)) \tag{11.2}$$

$$\forall p \in P_{loc} \cup P_{buf} : PrId_p(E(p,t)\langle b \rangle)(b(pv_t)) = |PrId_p(E(p,t)\langle b \rangle)| \tag{11.3}$$

$$\forall p \in P_{loc} : PrId_p(E(t,p)\langle b \rangle)(b(pv_t)) = |PrId_p(E(t,p)\langle b \rangle)| \tag{11.4}$$

□

In the definition above, item 1 to item 5 are rather straightforward. Item 6 defines the weights which are used to project out the process identity of tokens on the process, local, and buffer places of the process subnet. In the data base example, the common base colour set used to model the identity of the processes is DBM. The weight on the process places Waiting, Inactive, and Performing is the identity function on multi-sets. On the local place Received it is the projection into the second component. On the local place Unused it is the projection into the first component. This is also the weight on the buffer place Acknowledged, because we required in Equation (11.3) of item 8 that each token in a buffer place has a unique process that may *consume* it, and that process is identified by the first component. On the buffer place Sent the weight is the projection into the second component.

Item 7 expresses that a given process has only a single point of control. Notice that the colour set Ξ is interpreted as a multi-set in the equation.

Equation (11.2) in item 8 expresses that the occurrence of a binding element of a transition in the subnet removes exactly one token from the input process places of the transition, and adds exactly one token to the output process places of the transition. Furthermore, the colour of the tokens removed and added matches the binding of the process variable of the transition. This equation ensures that in any reachable marking, the process places contain exactly one token of each process identity in Ξ. Equation (11.3) in item 8 expresses that a token residing on a local or buffer place of the subnet can only be removed by the occurrence of binding elements belonging to c-binding-classes of transitions in the subnet, where c is the process identity of the token. Similarly, Equation (11.4) expresses that tokens added to a local place by the occurrence of a binding element get the process identity of the process that added them. Together these imply that tokens in a local place are processed and tokens in a buffer place are consumed by one process only.

We now continue with the definition of corresponding binding classes. By Equation (11.2) in Def. 24, for a token residing on a process place, they are those binding classes that contain binding elements which can potentially remove the token from the process place.

Definition 25 *Let $(CPN, P_{pr}, P_{loc}, P_{bor}, P_{buf}, \Xi, PV, PrId)$ be a process subnet. Let $p \in P_{pr}$. The **corresponding binding classes** of a token element (p, c) denoted $CB(p, c)$ are $CB(p, c) = \{t[pv_t = PrId_p(c)] \mid t \in Out(p)\}$.* □

We now define process partitioning of a CP-net, which divides a CP-net into a number of process subnets and ensures that these subnets are only allowed to share border places and are otherwise disjoint.

Definition 26 *A **process partitioning** of a CP-net $CPN = (\Sigma, P, T, A, N, C, G, E, I)$ is a set of n process subnets of the CPN: $\{(CPN^i, P^i_{pr}, P^i_{loc}, P^i_{bor}, P^i_{buf}, \Xi^i, PV^i, PrId^i)\}_{i \in I = \{1,2,\ldots,n\}}$, satisfying:*

1. *The set of places of the CP-net is the union of the places in the process subnets: $P = \bigcup_{i \in I} P^i$.*
2. *The set of transitions of the CP-net is a disjoint union of the transitions in the process subnets: $T = \bigcup_{i \in I} T^i$ and $\forall i, j \in I : [i \neq j \Rightarrow T^i \cap T^j = \emptyset]$.*
3. *The set of arcs in the CP-net is a disjoint union of the arcs of the process subnets: $A = \bigcup_{i \in I} A^i$ and $\forall i, j \in I : [i \neq j \Rightarrow A^i \cap A^j = \emptyset]$.*
4. *If two process subnets have common places, then they are border places: $\forall i, j \in I : [i \neq j \Rightarrow P^i \cap P^j \subseteq P^i_{bor}]$.*
5. *If a place is a buffer place of some process subnet, then only that subnet can consume tokens from it: $\forall i \in I : \forall p \in P^i_{buf} : Out(p) \subseteq T^i$.* □

If a border place is not a buffer place of any process subnet, then it is called a **shared place** of the process partitioning.

We can now formulate a proposition stating that the process places of the individual process subnets are related by a place invariant.

Proposition 3 *Let $\{(CPN^i, P^i_{pr}, P^i_{loc}, P^i_{bor}, P^i_{buf}, \Xi^i, PV^i, PrId^i)\}_{i \in I = \{1,2,\ldots,n\}}$ be a process partitioning of a CP-net CPN. For $i \in I$ define the set of place weights $\overline{PrId}^i = \{\overline{PrId}^i_p\}_{p \in P}$ by:*

$$\overline{PrId}^i_p = \begin{cases} PrId^i_p & : \quad p \in P^i_{pr} \\ 0_{MS} & : \quad otherwise \end{cases} \tag{11.5}$$

where 0_{MS} denotes the function mapping any multi-set into the empty multi-set. Then the following holds:

$$\forall M \in [M_0\rangle : \sum_{p \in P} \overline{PrId}^i_p(M(p)) = \Xi^i \tag{11.6}$$

Proof. First we prove that $\overline{PrId}^i$ is a place flow. For $t \notin T^i$ the place flow condition in Def. 21 is clearly satisfied since all input and output places of t then have 0_{MS} as weight. For $t \in T^i$ the place flow condition is guaranteed by Equation (11.2) of Def. 24. Hence, by Theorem 2, $\overline{PrId}^i$ determines a place invariant and the proposition now follows from Equation (1) of Def. 24. □

11.5 Stubborn Sets of Process-Partitioned CP-nets

In Section 11.2.2 we pointed out that most stubborn set construction algorithms rely on the notion of *dependency graphs.* In the case of PT-nets, the vertices of a dependency graph are the transitions, and each edge (t_1, t_2) represents a rule of the form "if t_1 is in the stubborn set, then also t_2 must be." To construct a stubborn set it suffices to know the dependency graph and the set of enabled transitions. Several different algorithms for this task have been suggested.

The goal of this section is to define dependency graphs for process-partitioned CP-nets such that their size is proportional to the number of transitions of the CP-net times the number of tokens on process places in the initial marking rather than the size of the equivalent PT-net. To achieve this, vertices of the new dependency graphs will be corresponding binding classes instead of binding elements. Although stubborn sets will eventually be defined as sets of binding elements, the discussion is simplified if we also talk about stubborn sets of binding classes:

Definition 27 *A set $Stub$ of binding classes is stubborn in a marking $M \in [M_0\rangle$, if and only if the following hold for every $t[pv_t = c] \in Stub$:*

1. *If $\exists (t_1, b_1) \in BE : M[(t_1, b_1)\rangle$, then $\exists\ t_2[pv_{t_2} = c'] \in Stub$ and $(t_2, b_2) \in t_2[pv_{t_2} = c'] : M[(t_2, b_2)\rangle$*

2. **Disabled Rule (D-rule):** *assume that $t[pv_t = c]$ may contain disabled binding elements (either it is not known whether $t[pv_t = c]$ contains disabled binding elements, or it is known that it does). For each input border place of t, consider the process subnets containing a transition with this place as an output place. The corresponding binding classes of the process token elements in these process subnets must be in $Stub$.*

3. **Enabled Rule (E-rule):** *assume that $t[pv_t = c]$ does contain enabled binding elements. For each input shared place of t, consider the process subnets containing a transition with this place as an input place. The corresponding binding classes of the process token elements in these process subnets must be in $Stub$.*

4. *A process token with process identity c is located on one of the input process places, p, of t in M and $CB(p, c)$ is in $Stub$.*

A set of binding elements is stubborn, if and only if it is the union of a stubborn set of binding classes. □

The edges of the new dependency graphs are determined according to D- and E-rules in item 2 and 3. Item 4 ensures that only binding classes resulting from the

use of the D- and E-rule are included in the stubborn set. The reason for the word "may" in D-rule is that often it is impossible or impractical to decide without unfolding whether a binding class contains disabled binding elements. "Unnecessary" use of D-rule makes the stubborn set larger, but does not endanger correctness, so we may allow it. This is an instance of approximating from above where a precise analysis requires unfolding. On the other hand, any algorithm that constructs the full state space of a CP-net must find all enabled binding elements. Therefore, when formulating E-rule, we assumed that it can be decided whether any given binding class contains enabled binding elements. This is not important, though; also E-rule can be used unnecessarily without affecting correctness. Note that if $t[pv_t = c]$ contains both enabled and disabled binding elements, then both rules must be applied.

Stubborn sets of process-partitioned CP-nets can be constructed from the dependency graphs just as in the case of PT-nets, with the exception that a vertex now represents binding classes that may consist of several binding elements. To start the construction, one can pick a process token in some process subnet such that at least one of the corresponding binding classes contains an enabled binding element. We will illustrate the new dependency graphs and their use in Sect. 11.6.

To show the correctness of the new method for constructing stubborn sets, we will need an auxiliary notion of **process-closure** $PrCl(t,b)$ of a binding element (t,b). It is defined as the set of binding elements (t',b') such that t' is a transition of the same process subnet as t and $b'(pv_{t'}) = b(pv_t)$. In other words, $PrCl(t,b)$ is the set of those binding elements modelling the actions of the process identified by the binding element (t,b). This notion is extended to sets of binding elements by defining $PrCl(B) = \bigcup_{(t,b)\in B} PrCl(t,b)$.

Theorem 5 *Let PPC be a process-partitioned CP-net, and PTN the PT-net that is obtained by unfolding PPC. Let $Stub_{PPC}$ be a stubborn set of binding elements of PPC in the sense of Def. 27. Then $PrCl(Stub_{PPC})$ is a stubborn set of PTN in the sense of Def. 22.* □

The proof of the theorem is omitted because of lack of space. The theorem says, in essence, that the process closure of the unfolding of any stubborn set obtained with the new dependency graphs is a stubborn set of the unfolded PT-net (albeit not necessarily an optimal one). Therefore, and because a CP-net has exactly the same behaviour as the equivalent PT-net [66], the analysis results obtained with a process-partitioned CP-net and its stubborn sets are the same as what would be obtained with ordinary stubborn sets and the unfolded PT-net. Because PT-net stubborn sets are guaranteed to preserve dead markings and possibility of non-termination, our process-partitioned CP-net stubborn sets also preserve these properties.

11.6 Stubborn Sets of the Data Base System

We now illustrate the use of D- and E-rule on the data base system for $n = 3$ data base managers in the initial marking M_0, and in two subsequent markings.

Stubborn set in M_0. Assume that we select the d_1-process-token in the only process subnet. Since the d_1-process-token is on Inactive we initiate the construction by

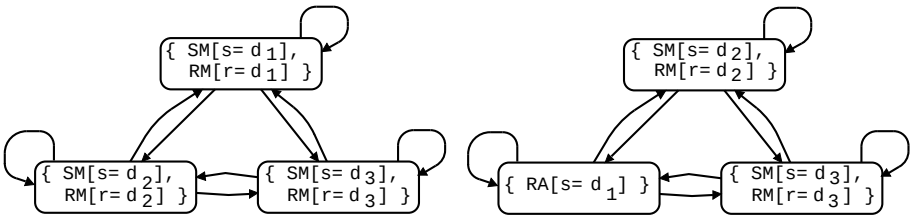

Figure 11.3: Computation of the stubborn sets in M_0 (left) and M_1 (right).

including $\{SM[s = d_1], RM[r = d_1]\}$ into the stubborn set. We now apply the D- and E-rule recursively.

First we consider $SM[s = d_1]$. Since the transition SM has only one variable, and that variable is the process variable, in this case it is possible to determine that the binding class contains no disabled binding elements. Thus, it suffices to apply only E-rule. SM has the shared place Passive as an input place. There is only one process subnet in the whole system, thus there is only one process subnet with a transition having Passive as an input place. All process tokens of this process subnet are located on Inactive and hence we include the following corresponding binding classes to our stubborn set: $\{SM[s = d_1], RM[r = d_1]\}$ and $\{SM[s = d_2], RM[r = d_2]\}$ and $\{SM[s = d_3], RM[r = d_3]\}$.

We now consider $RM[r = d_1]$. In the initial marking this binding class contains only disabled binding elements, hence (only) D-rule is applied. The transition has one input buffer place: Sent. We locate the process tokens in process subnets containing a transition with Sent as an output place, which leads us to include $\{SM[s = d_1], RM[r = d_1]\}$ and $\{SM[s = d_2], RM[r = d_2]\}$ and $\{SM[s = d_3], RM[r = d_3]\}$.

We have now processed the binding classes $SM[s = d_1]$ and $RM[r = d_1]$, and have found out that we also have to investigate the binding classes $\{SM[s = d_2], RM[r = d_2]\}$ and $\{SM[s = d_3], RM[r = d_3]\}$. Because they are symmetric to the first case, their analysis reveals that the inclusion to the stubborn set of $SM[s = d_i]$ and $RM[r = d_i]$ for any $i \in \{1, 2, 3\}$ will force the inclusion of $SM[s = d_i]$ and $RM[r = d_i]$ also with the other two possible values of i. These dependencies between binding classes can be illustrated with the dependency graph depicted on the left hand side of Fig. 11.3. The dependency graph contains all enabled binding elements and has only one strongly connected component. Hence, any stubborn set must contain all the enabled binding elements in the initial marking.

Stubborn set in M_1. Consider now the marking M_1 reached by the occurrence of the binding element $(SM, \langle s = d_1 \rangle)$ in M_0 (the two other cases corresponding to d_2 and d_3 are similar by symmetry, and we will skip them). Assume that we choose the d_2-process-token located on Inactive. Thus we initiate the construction by including $\{SM[s = d_2], RM[r = d_2]\}$ into the stubborn set.

Continuing with the application of the rules until all binding classes have been handled yields the dependency graph on the right hand side of Fig. 11.3. Again, all enabled binding elements must be included into the stubborn set. As a matter of fact,

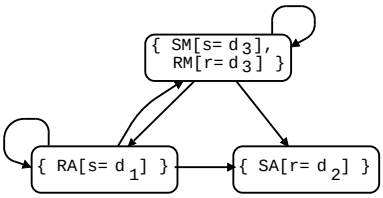

Figure 11.4: Computation of the stubborn set in M_2.

an analysis performed at the unfolded level shows that it is not necessary to take any other enabled binding elements than $(RM, \langle s = d_1, r = d_2 \rangle)$ into the stubborn set, but our method fails to see that this is the case. As was mentioned in the introduction, making the stubborn set analysis at too detailed a level would cause the analysis to collapse to the unfolding of the CP-net, which we want to avoid. It is better to keep the analysis simple and every now and then include more binding classes than absolutely necessary.

Stubborn set in M_2. Consider now the marking M_2 reached by the occurrence of the binding element $(RM, \langle s = d_1, r = d_2 \rangle)$ in M_1. Assume that we pick the d_3-process-token on Inactive. Thus, we initiate the construction of the stubborn set by including $\{SM[s = d_3], RM[r = d_3]\}$ into the stubborn set. Continuing with the application of the rules yields the dependency graph in Fig. 11.4.

An important aspect of the dependency graph is that there are no edges out of $SA[r = d_2]$. The reason is that both D-rule and E-rule look at input border places, but SA has none of them: Performing is a process place and Received is a local place. Hence we can choose $\{SA[r = d_2]\}$ as the stubborn set in M_2. It contains only one enabled binding element: $(SA, \langle s = d_1, r = d_2 \rangle)$.

It is worth noticing that this result generalises to all data base managers and remains valid even if the total number of the data base managers is not three. That is, independent of the number of the data base managers, the set $\{SA[r = d_j]\}$ is stubborn whenever $(SA, \langle s = d_i, r = d_j \rangle)$ is enabled for some i and j. In the markings reached from now on, there is only a single enabled binding element until the initial marking is reached again.

The number of markings in the full state space for the data base system is $1 + n3^{n-1} = \Theta(n3^n)$. Observing that the number of tokens on the place Received is always at most one with the new method for computing stubborn sets, the number of markings in the SS state space is $1 + n(2^{n-1} + (n-1)2^{n-2}) = \Theta(n^2 2^n)$. With unfolding it is possible to get a reduced state space with as few as $1+n(1+2(n-1)) = \Theta(n^2)$ markings. The reduction given by our new method is thus not as good as what may be obtained if one is willing to do the expensive unfolding.

11.7 Experiments

To obtain evidence on the practical use and performance with respect to reduction obtained and time used to generate the SS state space, an experimental prototype con-

$\|DBM\|$	Full state space			SS state space		
	Nodes	Arcs	Time	Nodes	Arcs	Time
3	28	42	1	25	30	1
4	109	224	1	81	104	1
5	406	1,090	2	241	330	2
6	1,459	4,872	16	673	972	9
7	5,104	20,426	142	1,793	2,702	40
8	17,497	81,664	1,139	4,609	7,184	157

Table 11.1: Verification statistics for the data base system.

taining the new method has been implemented on top of the state space tool of Design/CPN [16].

In this prototype, the user supplies the information on process subnets, and specifies which places are process places, local places etc. Once the information has been supplied, the SS state space can be generated fully automatically. The prototype uses a simple heuristic for choosing between the possible stubborn sets. In each marking, one of the stubborn sets containing a minimum number of enabled binding elements is selected as the stubborn set. It is worth noting that in general, this may fail to lead to the best possible reduction of the state space.

Below the prototype is applied to two case studies: the data base system from the previous sections, and to a stop-and-wait protocol. All measures presented in this section were obtained on a Sun Ultra Sparc Enterprise 3000 workstation with 512 MB RAM.

Distributed data base system. First we consider the data base system from the previous sections. Table 11.1 contains the sizes (nodes and arcs) of the full state space and the SS state space for varying number of data base managers. In addition, the generation times for the state spaces (in CPU seconds) are shown. A careful inspection of Table 11.1 shows that the experimental sizes fit the theoretical sizes obtained in Sect. 11.6.

Stop-and-wait protocol. We now consider a larger example in the form of a stop-and-wait protocol from the datalink control layer of the OSI network architecture. The protocol is taken from [5].

The CP-net of this stop-and-wait protocol is a hierarchical CP-net consisting of five pages. The CP-net has four process subnets modelling the threads in the receiver and sender parts of the protocol. It has six border places. Two border places are used to model the communication between the threads in the receiver and the sender, respectively, and two border places model the communication channels between the sender and the receiver.

Table 11.2 shows the verification statistics for the stop-and-wait protocol for varying capacities of the data channel (ChanD) and the acknowledgement channel (ChanA), and varying number of packets (Packets) sent from the sender to the receiver. The CP-net of the stop-and-wait protocol uses lists, strings and integers as types of the vari-

			Full state space			SS state space		
$ChanD$	$ChanA$	$Packets$	Nodes	Arcs	Time	Nodes	Arcs	Time
1	1	2	7,929	27,708	44	5,065	9,469	42
1	1	3	12,163	42,652	83	7,775	14,580	76
2	1	2	19,421	70,847	259	12,428	24,268	186
1	2	2	20,303	74,936	291	13,157	25,825	199
2	2	2	49,515	190,383	947	32,145	65,792	579
3	2	2	110,963	433,409	5,618	72,169	150,006	3,812
2	3	2	115,751	453,995	6,157	75,721	157,528	3,991

Table 11.2: Verification statistics for the stop-and-wait protocol.

ables of the transitions, and is therefore an example of a CP-net where the unfolding approach fails to work. As a consequence, we cannot compare the reductions obtained with the new method and the algorithm based on unfolding.

11.8 Conclusions and Future Work

We addressed the issue of computing stubborn sets of CP-nets without relying on unfolding to PT-nets. It was shown that the problem is computationally hard in the sense that there are CP-nets for which computing a non-trivial stubborn set requires time proportional to the size of the unfolded CP-net. A method for process-partitioned CP-nets was given which avoids the unfolding by exploiting additional structure on top of the CP-net. The method approximates the unfolded stubborn sets from above, thereby not necessarily yielding the best possible stubborn sets with respect to the reduction obtained.

The practical applicability of the suggested method was assessed by some case studies. A common denominator for the experiments was that the reduction obtained more than cancelled out the overhead involved in computing the stubborn sets. Hence, judging from the experiments, the suggested method seems in practice to give reasonably good stubborn sets, at a very low cost with respect to time. This indicates that the method seems to be a good compromise in the trade-off between not making too detailed an analysis of dependencies and at the same time getting a reasonable reduction. Equally important, unlike the method based on unfolding, the new method does not fail to work when colour sets with an infinite domain are used as types of variables of transitions.

Another interesting aspect arises when combining the stubborn set method with reduction by means of symmetry as suggested in [125]. If the method for computing stubborn sets in this paper is combined with symmetry reduction, then it may result in the same reduction as when the stubborn sets obtained with unfolding is combined with symmetry reduction. This is, for instance, the case with the data base system studied in this paper. Therefore, although the stubborn sets are not as good as the stubborn sets obtained with unfolding, they may still yield equally good results when symmetry is applied on top. This suggests using the symmetry method as a way of further improving the results. Future work will include work in this direction, as well

as the application of the new method to more elaborate versions of the stubborn set method that preserve more properties.

Our method requires the user to supply some information regarding the process subnets, process places, local places, border places, etc. It is reasonable to assume that the developer of a CPN model is able to supply such information, as it is similar to declaring types in a programming language. Also, the kind of information which must be supplied seems natural from the point of view of concurrent systems. However, in order to use the method on large examples, the validity of the supplied information must be checked automatically. One possible approach to this would be to exploit the techniques developed in [116] for place invariant analysis of CP-nets.

Chapter 12

Improved Question-Guided Stubborn Set Methods for State Properties

The paper *Improved Question-Guided Stubborn Set Methods for State Properties* constituting this chapter has been published as a technical report [87] and is submitted to the 21st International Conference on Application and Theory of Petri Nets (ICATPN'00).

[87] L. M. Kristensen and A. Valmari. Improved Question-Guided Stubborn Set Methods for State Properties. Technical report, Department of Computer Science, University of Aarhus, Denmark, 1999. DAIMI PB-543.

The content of this chapter is equal to the technical report [87] except for minor typographical changes.

Improved Question-Guided Stubborn Set Methods for State Properties

L. M. Kristensen* A. Valmari†

Abstract

We present two new question-guided stubborn set methods for state properties. The first method makes it possible to determine whether a marking is reachable in which a given state property holds. It generalises the results on stubborn sets for state properties recently suggested by Schmidt in the sense that that stubborn set method can be seen as an implementation of our more general method. We propose also alternative, more powerful implementations that have the potential of leading to better reduction results. This potential is demonstrated on some practical case studies.

As an extension of the first method, we present a second method which makes it possible to determine if from all reachable markings it is possible to reach a marking where a given state property holds. The novelty of this method is that it does not rely on ensuring that no transition is ignored in the reduced state space. Again, the benefit is in the potential for better reduction results.

Topics: System design and verification using nets, Analysis and synthesis of nets, Computer tools for nets.

12.1 Introduction

State space methods have proven powerful in the analysis and verification of concurrent systems. Unfortunately, the state spaces of systems tend to grow very rapidly when systems become bigger. This well-known phenomenon is referred to as *state explosion*, and it is a serious problem for the use of state space methods in the analysis of real-life systems.

Many techniques for alleviating the state explosion problem have been suggested, such as the *stubborn set methods* [120, 129]. They comprise a subgroup of rather similar methods first suggested in the late 80's and early 90's [49, 50, 101]. These methods are based on the fact that the total effect of a set of concurrent transitions is independent of the order in which the transitions occurs. Therefore, it often suffices to investigate only one or some orderings in order to reason about the behaviour of the system.

*Department of Computer Science, University of Aarhus, DK-8000 Aarhus C., DENMARK. E-mail: lmkristensen@daimi.au.dk.

†Tampere University of Technology, Software Systems Laboratory, PO Box 553, FIN-33101 Tampere, FINLAND. E-mail: ava@cs.tut.fi.

This paper presents two new stubborn set methods which make it possible to reason about *state properties.* A state property is a property that talks about only one marking. For instance, $M(p) \leq 10$ is a state property, whereas $\exists M' \in [M\rangle : M'(p) > M(p)$ is not.

The first stubborn set method makes it possible to answer the following question: "is it possible to reach a marking where a given state property holds?" The method is *question-guided*, i.e., it takes a state property as input and generates a reduced state space. This reduced state space will contain a marking where the property holds if and only if there exists a reachable marking in which the state property holds. This method is important, because with it one can, e.g., find place bounds, and check reachability of a (perhaps incompletely specified) marking, more efficiently than with existing stubborn set methods [109, 123, 124]. The method presented is based on the ideas in [109], but tries to compute better stubborn sets. This can potentially lead to better reduction results.

The second question-guided method makes it possible to answer the question: "is it possible from all reachable markings to reach a marking where a given state property holds?" This method can for instance be used to check liveness of a single transition with better reduction results than an earlier method [124] that check liveness of all transitions simultaneously. It can also be used to check whether a given (perhaps incompletely specified) marking is a home marking more efficiently than with the technique described in [109].

The paper is organised as follows. Section 12.2 recalls the basic facts of Place/-Transition Nets (PT-nets), state spaces, and stubborn sets used in the rest of this paper. Section 12.3 gives an informal introduction to the first stubborn set method by means of a small example. Sections 12.4-12.7 formally develop the new stubborn set methods, and Sect. 12.8 considers their implementation. Section 12.9 discusses applications of the first method to boundedness properties of PT-nets. Section 12.10 gives some numerical data on the performance of the first method on some case studies. Finally, we sum up the conclusions in Sect. 12.11.

12.2 Background

This section briefly summarises the basic facts and notation of PT-nets, state spaces, and stubborn sets used in the rest of the paper. We assume that the reader is familiar with the dynamic behaviour of PT-nets and the basic ideas of state spaces (also called occurrence graphs or reachability graphs/trees).

Definition 28 *A **Place/Transition Net** is tuple $PTN = (P, T, A, W, M_I)$, where P is a finite set of places, T is a finite set of transitions such that $P \cap T = \emptyset$, $A \subseteq (P \times T) \cup (T \times P)$ is a set of arcs, $W : A \to \mathbb{N}_+$ is an arc weight function, and $M_I : P \to \mathbb{N}_0$ is the initial marking.* □

We use M_I as the initial marking instead of the more conventional M_0. This allows us to use M_0 as the first marking of *occurrence sequences* which do not necessarily start in the initial marking. If a transition t is *enabled* in a marking M_1 (denoted $M_1[t\rangle$), then t may *occur* in M_1 yielding some marking M_2. This is written $M_1[t\rangle M_2$. Extending this notation, an occurrence sequence is denoted $M_0[t_1\rangle M_1 \cdots M_{n-1}[t_n\rangle M_n$ and

satisfies $M_{i-1}[t_i\rangle M_i$ for $1 \leq i \leq n$. When the intermediate markings in an occurrence sequence are not important we will write it as $M_0[t_1 t_2 \cdots t_n\rangle M_n$. A *reachable marking* is a marking which can be obtained (reached) by an occurrence sequence starting in the initial marking. By $[M\rangle$ we denote the set of markings reachable from a marking M. For a place (transition) x, $\bullet x$ denotes the set of input transitions (places) of x, and $x\bullet$ is a similar notation for output transitions (places). The notation is extended to sets by taking the union of $\bullet x$ ($x\bullet$) over each member x of the set. In a marking M, the marking of a place p is denoted $M(p)$.

Definition 29 *The **Full State Space** of a PT-net is a directed graph $SG = (V, E)$, where $V = [M_I\rangle$ and $E = \{\ (M_1, t, M_2) \in V \times T \times V \mid M_1[t\rangle M_2\ \}$.* □

In the rest of this paper we assume that a PT-net (P, T, A, W, M_I) with a *finite* full state space $SG = (V, E)$ is given. For some of the stubborn set algorithms presented in this paper we will exploit the *strongly connected components*. A strongly connected component (SCC) is a non-empty set C of reachable markings such that if $M \in C$ then $C = \{\ M' \mid M' \in [M\rangle \wedge M \in [M'\rangle\ \}$. An SCC is said to be a *terminal strongly connected component* iff $M \in C$ implies $[M\rangle \subseteq C$.

State space construction with stubborn sets follows the same procedure as the construction of the full state space of a PT-net, with one exception. When processing a marking, a set of transitions, the so-called *stubborn set*, is constructed. Only the enabled transitions in the stubborn set are used to construct successor markings. This means that only a subset of the relation $M[t\rangle M'$ is used for the construction of the reduced state space. We denote this subset by $M[t\rangle_{SSG} M'$, and define $M[t_1 \cdots t_n\rangle_{SSG} M'$ and $[M\rangle_{SSG}$ as for the full state space but now based on the relation $M[t\rangle_{SSG} M'$. The stubborn set reduced state space (from now on called the *SS state space*) can be defined as a directed graph $SSG = (V_{SSG}, E_{SSG})$ based on the relation $M[t\rangle_{SSG} M'$ in a similar way as the full state space. We define the (terminal) SCCs for the SS state space analogously to the case for the full state space.

The choice of stubborn sets depends on the properties that are being analysed or verified of the system. Many stubborn set algorithms are surveyed in [129]. They all assume that the stubborn sets used in each marking satisfy certain conditions, and stubborn set methods for different properties are obtained by using different conditions. However, it is common to almost all of them that the conditions listed below should hold. Below $T_{\mathsf{s}}(M)$ denotes the stubborn set used in the marking M.

D1 If $t \in T_{\mathsf{s}}(M_0)$, $t_1, \ldots, t_n \notin T_{\mathsf{s}}(M_0)$, $M_0[t_1 t_2 \cdots t_n\rangle M_n$, and $M_n[t\rangle M'_n$, then there is M'_0 such that $M_0[t\rangle M'_0$ and $M'_0[t_1 t_2 \cdots t_n\rangle M'_n$.

D2 If M_0 has an enabled transition, then there is at least one transition $t_{\mathsf{k}} \in T_{\mathsf{s}}(M_0)$ such that if $t_1, \ldots, t_n \notin T_{\mathsf{s}}(M_0)$ and $M_0[t_1 t_2 \cdots t_n\rangle M_n$, then $M_n\ [t_{\mathsf{k}}\rangle$. Any transition with this property is called a *key transition* of $T_{\mathsf{s}}(M_0)$.

The conditions D1 and D2 as such are not suited for constructing stubborn sets since they refer to occurrence sequences. Therefore, the construction of stubborn sets is in practice implemented by relying on rules that refer only to the structure of the PT-net and the current marking, and which express sufficient conditions to make D1 and D2 hold. The tutorial [129] lists a number of such. Below we give a simple proposition

which guarantees that D1 and D2 hold. The proposition analyses the dependencies between transitions at a rather coarse level, and it is not optimal in the sense of yielding smallest possible stubborn sets and smallest SS state spaces. We will use it only for illustration purposes.

Proposition 4 *The conditions D1 and D2 hold if the following hold for every* $t \in T_{\mathsf{s}}(M)$*:*

1. *If* $\exists t_1 \in T : M\,[t_1\rangle$*, then* $\exists t_2 \in T_{\mathsf{s}}(M) : M\,[t_2\rangle$.
2. *If* $\neg M\,[t\rangle$*, then* $\exists p \in {\bullet t} : M(p) < W(p,t) \ \wedge \ {\bullet p} \subseteq T_{\mathsf{s}}(M)$.
3. *If* $M\,[t\rangle$*, then* $({\bullet t}){\bullet} \subseteq T_{\mathsf{s}}(M)$. □

The important aspect of Prop. 4 is that the three items can be read as rules. Item 1 specifies that if there is an enabled transition, then an enabled transition has to be in the stubborn set. Item 2 specifies that if a disabled transition t has been included in the stubborn set, some place p in the preset of t which does not contain enough tokens for t to be enabled must be chosen and its preset included. Finally, item 3 specifies that if an enabled transition t has been included then the postset of the preset of t must be included. A number of algorithms for constructing stubborn sets based on propositions like Prop. 4 are given in [129].

12.3 An Example

In this section we introduce the first of our improved stubborn set methods in an informal way using the simple PT-net shown in Fig. 12.1. Figure 12.2 shows the full state space of this PT-net. Node 1 corresponds to the initial marking. Each arc has an associated label giving the name of the transition to which it corresponds. For a node n we denote the corresponding marking by M_n.

Suppose that we want to check that there exists a reachable marking in which the place p_{10} contains at least two tokens. This can be expressed as the *state property* $\phi \equiv M(p_{10}) \geq 2$. M_9 is the only such marking.

The stubborn set method in [109], in the following referred to as the *attractor set method*, would define an *attractor set* in M_1, denoted $A_\phi(M_1)$, for the *atomic state proposition* $M(p_{10}) \geq 2$. The role of the attractor set is to ensure that in each step of the SS state space construction, progress is made towards a marking where the property holds. The attractor set in M_1 would consist of the transitions which can add tokens to p_{10}. Hence $A_\phi(M_1) = \{t_5, t_6\}$. The attractor set method requires the attractor set to be a subset of the stubborn set in each marking. If we apply Prop. 4, then the stubborn set in M_1 will be $\{t_1, t_2, t_3, t_4, t_5, t_6\}$. Hence both enabled transitions (t_1 and t_2) are in the stubborn set in M_1.

If we consider the marking M_2 then the attractor set remains the same as in M_1 and Prop. 4 gives us $\{t_2, t_4, t_5, t_6\}$ as the stubborn set. Again, all enabled transitions are included in the stubborn set. The situation in M_3 is symmetric to M_2. In M_4, the transition t_2 will be in the stubborn set. The situation in M_6 is symmetric to M_4, and in M_5, the transitions t_5 and t_6 will be in the stubborn set. In M_7 and M_8, the transition

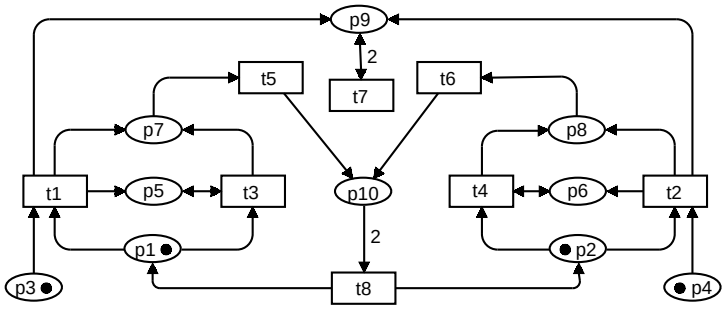

Figure 12.1: Example PT-net.

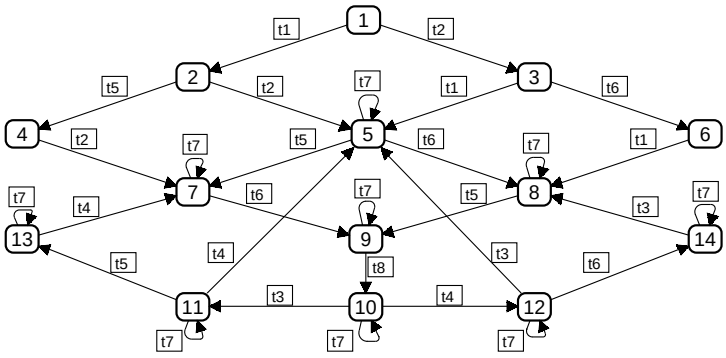

Figure 12.2: Full state space for the PT-net in Fig. 12.1.

t_6 and t_5, respectively, will be in the stubborn set. In conclusion this means that the attractor set method yields an SS state space consisting of markings M_1 to M_9.

It can however be observed that it is possible to select stubborn sets during the construction of an SS state space with fewer enabled transitions than those required by the attractor set method. This could potentially lead to more reduction. The basic idea in our new method is to relax the requirement that the attractor set must *always* be contained in the stubborn set.

Suppose that the requirement imposed by the attractor set were totally removed. From Prop. 4 it follows that in M_1 we can select $\{t_2, t_4\}$ or $\{t_1, t_3\}$ as the stubborn set. Suppose that we select $\{t_1, t_3\}$. Proposition 4 implies that it is possible to select $\{t_5\}$ or $\{t_2, t_4\}$ as the stubborn set in M_2. If in M_2 we select the latter, then in M_5 we can select $\{t_5\}$, $\{t_6\}$, or $\{t_7\}$ as the stubborn set.

If in M_5 we select the stubborn set consisting of $\{t_7\}$ only, then the construction of the SS state space will terminate at this point, since M_5 is already included in it. This means that we would wrongly conclude that there does not exist a marking in which ϕ holds. The problem is that we have not ensured *progress* towards such a marking. The attractor set method ensures progress in each marking of the SS state space by *always* including the attractor set in the stubborn set. Instead of this strong requirement we will ensure that from each marking in the SS state space *eventually* progress can be made, i.e., a marking is reachable in the SS state space in which progress is made.

For this purpose we introduce the notion of *up sets*. An up set is a set of transitions

chosen such that at least one transition in it has to occur in order to make the state property hold. Hence the up sets are similar to attractor sets. However, unlike the attractor set method, we will not require that the up set is always contained in the stubborn set. Moreover, we will additionally exploit that the state properties which we consider are growing Boolean functions. This makes it possible to ensure progress towards the property by either reducing the length of an occurrence sequence leading to a marking where the state property holds, or by increasing the number of atomic state propositions which are satisfied. This requirement will ensure that t_5 or t_6 is in the stubborn set in M_5. Similarly, it will ensure that t_6 is in the stubborn set in M_7, and that t_5 is in the stubborn set in M_8.

Ensuring eventual progress is however not sufficient for preserving state properties. As a simple example, suppose that we want to show that a marking is reachable in which $M(p_3) = 0$ and $M(p_4) = 1$. This corresponds to showing that M_2 or M_4 is reachable. If $\{t_2, t_4\}$ is selected as the stubborn set in M_1, then neither of M_2 and M_4 will be in the SS state space. The problem is that in M_1 the only enabled transition in the stubborn set is t_2, and an occurrence of this transition can change the value of the state property from True to False. To account for this we introduce the notion of *down sets*. A down set is a set of transitions chosen such that a transition in the down set has to occur in order to make the property not hold. We will ensure that if an enabled transition which is in the down set is in the stubborn set, then the transitions in the up set are also in the stubborn set. This will ensure that if t_2 is in the stubborn set in M_1 then also t_1 is.

12.4 State Properties

We consider state properties expressed as formulas that are composed of so-called *atomic state propositions* using only the logical operators "$\wedge$" and "$\vee$" and parentheses "(" and ")". For a state property ϕ we denote its atomic state propositions by $\varphi_1, \varphi_2, \ldots, \varphi_n$, and let $I = \{1, 2, \ldots, n\}$ denote the set of indices of the atomic state propositions. The atomic state propositions and state properties are interpreted on the markings of the PT-net, and the resulting truth values are denoted by $\varphi_i(M)$ and $\phi(M)$. The atomic state propositions are defined according to the following syntax, where p, p_1, and p_2 denote arbitrary places and k is an integer constant.

$$\begin{aligned} \varphi_i \quad ::= \quad & M(p) \geq k \mid M(p_1) \geq M(p_2) \mid M(p) = k \mid M(p_1) = M(p_2) \mid \\ & M(p) \leq k \mid M(p_1) > M(p_2) \mid M(p) \neq k \mid M(p_1) \neq M(p_2) \end{aligned}$$

We have not included $M(p) > k$ and $M(p) < k$ as atomic state propositions since they can be expressed as $M(p) \geq k + 1$ and $M(p) \leq k - 1$, respectively. The set of atomic state propositions could be extended provided that the corresponding up and down sets to be defined in Sect. 12.5 are implemented properly.

Above only conjunction and disjunction were allowed as the Boolean operators. However, the atomic state propositions are closed under negation (p_1 and p_2 may be swapped when needed), so formulas which use negation can always be re-written to a form allowed by the above syntax using *De Morgan's equivalences* (i.e., $\neg(\phi_1 \vee \phi_2) = \neg\phi_1 \wedge \neg\phi_2$ and $\neg(\phi_1 \wedge \phi_2) = \neg\phi_1 \vee \neg\phi_2$). Therefore, the syntax does not restrict generality. It is however important for the correctness of the later algorithms

that formulas are given in a negation-free form, i.e., that they have been preprocessed before being provided as input to the algorithms.

Definition 30 *Let M be a marking and ϕ a state property constructed from the atomic state propositions $\{ \varphi_i \mid i \in I \}$. The set of the indices of the atomic state propositions which are satisfied in M is denoted $\mathsf{on}_\phi(M)$. The set of the indices of the atomic state propositions which are not satisfied in M is denoted $\mathsf{off}_\phi(M)$. Formally:*

$$\mathsf{on}_\phi(M) = \{ i \in I \mid \varphi_i(M) \} \text{ and } \mathsf{off}_\phi(M) = \{ i \in I \mid \neg\varphi_i(M) \}$$ □

If we let $\mathbb{B} = \{\mathsf{True}, \mathsf{False}\}$, treat the φ_i's as argument symbols, and define $\mathsf{False} \leq \mathsf{True}$, then a state property formula ϕ determines a monotonically increasing Boolean function from $\mathbb{B}^n$ to $\mathbb{B}$.

The following proposition lists important properties of the state property formulas which will be exploited later.

Proposition 5 *Let M and M' be markings and ϕ a state property constructed from the atomic state propositions $\{ \varphi_i \mid i \in I \}$. Then the following holds:*

1. $\forall i \in I : \varphi_i(M) \leq \varphi_i(M') \Rightarrow \phi(M) \leq \phi(M')$.
2. $\phi(M) \wedge \neg\phi(M') \Rightarrow \exists i \in I : \varphi_i(M) \wedge \neg\varphi_i(M')$. □

Item 1 states that ϕ is a monotonically increasing Boolean function. Item 2 states that if ϕ is satisfied in M but not in M' then there exists at least one atomic state proposition which is satisfied in M but not satisfied in M'. Item 2 is a consequence of item 1.

12.5 Up/Down and Satisfiability Sets

To describe the required properties of the stubborn sets, we define two sets of transitions related to a state property ϕ: an *up set* and a *down set*. The *up set* of ϕ in a marking M is a set of transitions chosen such that if ϕ does not hold in M then at least one transition in the up set must occur before ϕ can start to hold. The *down set* of ϕ is a set of transitions chosen such that it contains at least all transitions whose occurrence can change the value of some atomic state proposition φ_i of ϕ from True to False. In addition to these two sets we define the *satisfiability set* of ϕ in M as a set of indices of the atomic state propositions such that at least one atomic state proposition that has its index in the set has to change its value from False to True in order to make the state property hold.

The implementation of concrete up and down sets will be determined from the atomic state propositions and Boolean combinators. However, the properties of up and down sets are general concepts and not tied to the specific set of state properties considered in this paper. Therefore, we define up and down sets as properties of a set of transitions. A similar remark applies to satisfiability sets.

Definition 31 *Let ϕ be a state property constructed from the atomic state propositions $\{ \varphi_i \mid i \in I \}$ and let $M_0 \in [M_I\rangle$. A set of transitions $T' \subseteq T$ has the **up set property** in M_0 with respect to ϕ iff the following holds for all occurrence sequences $M_0[t_1 t_2 \cdots t_n\rangle M_n$ starting in M_0:*

$$\neg\phi(M_0) \wedge \phi(M_n) \Rightarrow \exists j : \ 1 \leq j \leq n \wedge t_j \in T'$$

*A set of transitions $T' \subseteq T$ has the **down set property** with respect to ϕ iff the following holds for all markings $M, M' \in [M_I\rangle$, all $t \in T$, and all $i \in I$:*

$$M\ [t\rangle\ M' \wedge \varphi_i(M) \wedge \neg\varphi_i(M') \Rightarrow t \in T'$$

*A set of indices $J \subseteq I$ has the **satisfiability set property** in M_0 with respect to ϕ iff the following holds for all occurrence sequences $M_0[t_1 t_2 \cdots t_n\rangle M_n$:*

$$\neg\phi(M_0) \wedge \phi(M_n) \Rightarrow \exists i \in J : \neg\varphi_i(M_0) \wedge \varphi_i(M_n) \qquad \square$$

The properties of up set and satisfiability set are relative to the current marking whereas the down set property is not. This is deliberate and due to the way our methods will later use these sets. It is worth observing that the definition of up (down) set property allows approximations of the up (down) sets to be used: if $T' \subseteq T''$ and T' has the up (down) set property then also T'' has the up (down) set property. A similar remark applies to indices and the satisfiability set property. This will be exploited later once we show how to construct such sets. Moreover, if a marking in which ϕ holds is reachable from a marking M then a satisfiability set in M exists and it is non-empty because of Prop. 5. From now on we will assume that we have an algorithm that given a state property ϕ produces some down set down_ϕ, and additionally given a marking produces some up set $\mathsf{up}_\phi(M)$ and some satisfiability set $\mathsf{sat}_\phi(M)$. We will give such an algorithm in Sect. 12.8.

12.6 Preserving Reachability of State Properties

This section presents the new stubborn set method for determining whether a reachable marking exists in which a given state property holds. The method consists of obeying the D1 condition from Sect. 12.2 and two additional conditions formulated in the following definition. An explanation of the definition will be given below.

Definition 32 *Let M be a marking and ϕ a state property constructed from the atomic state propositions $\{\ \varphi_i \mid i \in I\ \}$. A set $T_\mathsf{s}(M) \subseteq T$ is **Reachability of a State Property Preserving (RSPP) stubborn** in M, iff the following hold:*

D1 *If $t_1, \ldots, t_n \notin T_\mathsf{s}(M)$, $t \in T_\mathsf{s}(M)$, $M[t_1 t_2 \cdots t_n\rangle M_n$, and $M_n[t\rangle M'_n$, then there is M' such that $M[t\rangle M'$ and $M'[t_1 t_2 \cdots t_n\rangle M'_n$.*

SPP1 *If $\neg\phi(M)$ and $\exists t : M\ [t\rangle \wedge t \in \mathsf{down}_\phi \wedge t \in T_\mathsf{s}(M)$ then $\mathsf{up}_\phi(M) \subseteq T_\mathsf{s}(M)$.*

SPP2 *For every $i \in \mathsf{sat}_\phi(M)$ there is an occurrence sequence $M_0[t_1\rangle M_1[t_2\rangle \cdots [t_n\rangle M_n$ such that $M = M_0$, t_j is a key transition of $T_\mathsf{s}(M_{j-1})$ for $1 \leq j \leq n$, and $\phi(M_n) \vee \mathsf{up}_{\varphi_i}(M_n) \subseteq \bigcup_{j=0}^{n} T_\mathsf{s}(M_j)$.* $\square$

The intuitive purpose of SPP1 is to ensure that a next step in the SS state space can be taken in such a way that we do not get further away from a marking where ϕ holds. SPP1 requires that if we have taken an enabled transition in the down set, then we

have also included the transitions in the up set. The latter transitions represent a step towards a marking where ϕ holds, since we know that a transition in the up set has to occur in order to make ϕ hold. Therefore if one transition makes regress then there is another transition that makes progress. It is also possible that no enabled transition makes regress or progress.

SPP2, on the other hand, is there to *ensure* progress – to ensure that we will eventually get closer to a marking where ϕ holds. If ϕ holds in M then SPP2 holds trivially since we can then choose $n = 0$. If SPP2 does not take us directly to a marking where ϕ holds, then it ensures that there is a path in the SS state space where we eventually try every transition in the up set of some atomic state proposition which has to change its value. This represents progress, since such an additional atomic state proposition has to be satisfied in order to make the state property hold. SPP2 states its requirement to every element $i \in \mathsf{sat}_\phi(M)$ because $\mathsf{sat}_\phi(M)$ is an upper approximation and we do not necessarily know which member is important.

We now turn to the correctness of the RSPP-stubborn set method. The key to establishing correctness is the following lemma.

Lemma 1 *Let ϕ be a state property, $SG = (V, E)$ the full state space, and $SSG = (V_{SSG}, E_{SSG})$ an SS state space constructed using RSPP-stubborn sets. Let $M_0 \in V_{SSG}$ be a marking such that $\neg\phi(M_0)$ and for which there exists an occurrence sequence $M_0[t_1\rangle M_1[t_2\rangle \cdots [t_n\rangle M_n$ such that $\phi(M_n)$ holds. Then there is a marking $M'_0 \in [M_0\rangle_{SSG}$ such that the following holds.*

1. *There are transitions $t'_1, t'_2, \ldots, t'_m$ and markings $M'_1, M'_2, \ldots, M'_m$ such that $M'_0[t'_1\rangle M'_1[t'_2\rangle \cdots [t'_m\rangle M'_m$ and $\phi(M'_m)$ holds.*

2. *The occurrence sequence in item 1 leading to a marking where ϕ holds is no longer than the original occurrence sequence, i.e., $m \leq n$.*

3. *The length of the occurrence sequence in item 1 has decreased, i.e., $m < n$, or the set of the atomic state propositions of ϕ which are satisfied has grown, i.e., $\mathsf{on}_\phi(M_0) \subset \mathsf{on}_\phi(M'_0)$.* □

Proof of Lemma 1. Since $\neg\phi(M_0)$ and $\phi(M_n)$ then $\mathsf{sat}_\phi(M_0)$ contains an i such that $\neg\varphi_i(M_0)$ and $\varphi_i(M_n)$. Since $\neg\phi(M_0)$ and $T_\mathsf{s}(M_0)$ (the RSPP-stubborn set in M_0) satisfies the condition SPP2 there exist key transitions $\bar{t}_1, \ldots, \bar{t}_k$ and markings $\bar{M}_0, \ldots, \bar{M}_k \in V_{SSG}$ such that $M_0 = \bar{M}_0[\bar{t}_1\rangle_{SSG}\bar{M}_1[\bar{t}_2\rangle_{SSG}\cdots[\bar{t}_k\rangle_{SSG}\bar{M}_k$, and $\phi(\bar{M}_k) \vee \mathsf{up}_{\varphi_i}(\bar{M}_k) \subseteq \bigcup_{j=0}^{k} T_\mathsf{s}(\bar{M}_j)$.

If $\phi(\bar{M}_j)$ holds for some $0 \leq j \leq k$ then the claim holds by choosing $M'_0 = \bar{M}_j$ and $m = 0$. In this case $m < n$ since $\neg\phi(M_0)$ and $\phi(M_n)$. From now on we may therefore assume that $\forall j, 0 \leq j \leq k : \neg\phi(\bar{M}_j)$. In the rest of the proof the following fact is needed:

(1) If $\{t_1, \ldots, t_n\} \cap T_\mathsf{s}(\bar{M}_h) = \emptyset$ for every $0 \leq h < l \leq k$, then, due to the key transition property of $\bar{t}_1, \ldots, \bar{t}_k$ and D1, there are markings $\hat{M}_0, \ldots, \hat{M}_l$ such that $M_n = \hat{M}_0[\bar{t}_1 \cdots \bar{t}_l\rangle\hat{M}_l$ and $\bar{M}_l[t_1 \cdots t_n\rangle\hat{M}_l$. Furthermore, if we assume that there exists a smallest index h such that $\bar{t}_{h+1} \in \mathsf{down}_\phi$ then $\phi(\hat{M}_0), \ldots, \phi(\hat{M}_h)$

hold due to the down set property. Thus, $\neg\phi(\bar{M}_h)$ and $\phi(\hat{M}_h)$ and SPP1 implies that $\emptyset \neq \{t_1, \ldots, t_n\} \cap \mathsf{up}_\phi(\bar{M}_h) \subseteq T_\mathsf{s}(\bar{M}_h)$ contrary to our assumption. Consequently, $\bar{t}_1, \ldots, \bar{t}_l \notin \mathsf{down}_\phi$ and we therefore have $\mathsf{on}_\phi(\bar{M}_0) \subseteq \mathsf{on}_\phi(\bar{M}_1) \subseteq \cdots \subseteq \mathsf{on}_\phi(\bar{M}_l)$, $\mathsf{on}_\phi(\hat{M}_0) \subseteq \mathsf{on}_\phi(\hat{M}_1) \subseteq \cdots \subseteq \mathsf{on}_\phi(\hat{M}_l)$, and $\phi(\hat{M}_0), \ldots, \phi(\hat{M}_l)$ hold.

We now split the proof in two cases.

Case A: $\{t_1, \ldots, t_n\} \cap T_\mathsf{s}(\bar{M}_j) \neq \emptyset$ for some $0 \leq j \leq k$. In this case we can pick the smallest such j and apply (1) for $l = j$. Since $T_\mathsf{s}(\bar{M}_j)$ contains at least one of the transitions $t_1, t_2, \ldots, t_n$, then we can pick the first such transition t_h and apply D1 on $\bar{M}_j[t_1 \cdots t_{h-1} t_h t_{h+1} \cdots t_n\rangle \hat{M}_j$ to obtain a marking M'' such that $\bar{M}_j[t_h\rangle M''[t_1 \cdots t_{h-1} t_{h+1} \cdots t_n\rangle \hat{M}_j$. The claim now holds with $M'_0 = M''$ and $m = n - 1$.

Case B: $\{t_1, \ldots, t_n\} \cap T_\mathsf{s}(\bar{M}_j) = \emptyset$ for every $0 \leq j \leq k$. In this case (1) gives us that $\mathsf{on}_\phi(\bar{M}_0) \subseteq \mathsf{on}_\phi(\bar{M}_k)$ and $\mathsf{on}_\phi(\hat{M}_0) \subseteq \mathsf{on}_\phi(\hat{M}_k)$. If $\mathsf{on}_\phi(\bar{M}_0) = \mathsf{on}_\phi(\bar{M}_k)$ then we must have one of the $t_1, t_2, \ldots, t_n$ in $\mathsf{up}_{\varphi_i}(\bar{M}_k)$ and by SPP2 also in $\bigcup_{j=0}^{k} T_\mathsf{s}(\bar{M}_j)$ which contradicts the assumption that $\{t_1, \ldots, t_n\} \cap T_\mathsf{s}(M_j) = \emptyset$ for every $0 \leq j \leq k$. Therefore $\mathsf{on}_\phi(\bar{M}_0) \subset \mathsf{on}_\phi(\bar{M}_k)$ and the claim holds with $M'_0 = \bar{M}_k$ and $m = n$. □

The following theorem states that if there exists a marking in the SS state space from which it is possible to reach a marking where the state property holds then the SS state space also contains a marking in which the state property holds. The correctness of the RSPP stubborn set method follows immediately from the theorem by letting $M_0 = M_I$.

Theorem 6 *Let ϕ be a state property, $SG = (V, E)$ be the full state space, $SSG = (V_{SSG}, E_{SSG})$ an SS state space constructed using RSPP-stubborn sets, and let $M_0 \in V_{SSG}$. Then:*

$$\exists M \in [M_0\rangle : \phi(M) \Leftrightarrow \exists M' \in [M_0\rangle_{SSG} : \phi(M') \qquad \square$$

Proof of Thm. 6. The $\Leftarrow$ direction follows from the fact that $V_{SSG} \subseteq V$ and $E_{SSG} \subseteq E$. For establishing the $\Rightarrow$ direction we apply Lemma 1 inductively to obtain $M_1 \in [M_0\rangle_{SSG}, M_2 \in [M_0\rangle_{SSG}, \ldots$, until we find an $M_n \in [M_0\rangle_{SSG}$ such that $\phi(M_n)$ holds. The induction hypothesis is that there is a marking M^i_ϕ and an occurrence sequence σ_i such that $M_i[\sigma_i\rangle M^i_\phi$ and $\phi(M^i_\phi)$ holds. When $i = 0$ this holds with $M^0_\phi = M$.

Define the *distance* $\Delta(M', \sigma, \phi)$ between a marking $M' \in V_{SSG}$ and a marking $M_\phi \in V$ which satisfies ϕ and which can be reached from M' by the occurrence sequence σ as follows ($|\sigma|$ denotes the length of the occurrence sequence σ):

$$\Delta(M', \sigma, \phi) = (|I| + 1) \cdot |\sigma| + |\mathsf{off}_\phi(M')|$$

The reason for the rather complicated definition of distance is that if at a marking we choose a transition which decreases the length of the occurrence sequence leading to

a marking where ϕ holds we may at the same time switch some of the atomic state propositions off.

If ϕ does not hold in M_{i-1}, then Lemma 1 gives a marking $M_i \in [M_{i-1}\rangle_{SSG}$, an occurrence sequence σ_i, and a marking M^i_ϕ such that $M_i[\sigma_i\rangle M^i_\phi$ and ϕ holds in M^i_ϕ. Items 2 and 3 of Lemma 1 ensure that $\Delta(M_i, \sigma_i, \phi) < \Delta(M_{i-1}, \sigma_{i-1}, \phi)$. Clearly $0 \leq \Delta(M_i, \sigma_i, \phi) < \infty$, so eventually this process terminates in a marking $M_n \in [M_0\rangle_{SSG}$ in which ϕ holds. □

12.7 Preserving Home State Properties

This section presents a new stubborn set method for determining whether from all reachable markings it is possible to reach a marking where a given state property holds. This can be formally expressed as determining whether $\forall M \in [M_I\rangle : \exists M' \in [M\rangle : \phi(M')$. The method presented is based on the observation that by negation this is the same as determining whether a reachable marking exists from which it is *not* possible to make the given state property hold. This can be expressed as $\exists M \in [M_I\rangle : \forall M' \in [M\rangle : \neg\phi(M')$. We will use "$\phi \in [M\rangle$" as an abbreviation of $\exists M' \in [M\rangle : \phi(M')$ (from M a marking M' can be reached where ϕ holds), and "$\phi \in [M\rangle_{SSG}$" as an abbreviation of $\exists M' \in [M\rangle_{SSG} : \phi(M')$.

The method consists of obeying the conditions from the RSPP-stubborn set method from Sect. 12.6 and two additional conditions formulated in the following definition.

Definition 33 *Let M be a marking and ϕ a state property. A set $T_{\mathsf{s}}(M) \subseteq T$ is **Home State Property Preserving (HSPP) Stubborn** in M, iff $T_{\mathsf{s}}(M)$ is RSPP stubborn in M and the following hold:*

D2′ *If $t_1, \ldots, t_n \notin T_{\mathsf{s}}(M)$, $t \in T_{\mathsf{s}}(M)$, $M[t_1 t_2 \cdots t_n\rangle M_n$, and $M\ [t\rangle$, then $M_n\ [t\rangle$.*

SPP3 *For every $t \in \mathsf{down}_\phi$ there is an occurrence sequence $M_0\,[t_1\rangle \cdots [t_n\rangle\, M_n$ such that $M = M_0$, $t_j \in T_{\mathsf{s}}(M_{j-1})$ for $1 \leq j \leq n$, and $t \in T_{\mathsf{s}}(M_n)$.* □

The intuitive purpose of $T_{\mathsf{s}}(M)$ being RSPP stubborn in the context of this method is to ensure that we from any marking in the SS state space always attempt to make ϕ hold (recall that we are trying to show that there exists a reachable marking from which ϕ cannot be made to hold). The D2′ condition is like the D2 condition from Sect. 12.2, except that it requires all enabled transitions in the stubborn set to be key transitions and allows $T_{\mathsf{s}}(M) = \emptyset$. Together with the D1 condition inherited from RSPP this implies that HSPP stubborn sets are *strong stubborn sets* [129]. SPP3 is there to *ensure* progress, to ensure that we eventually get closer to a marking from which ϕ cannot be made to hold. This is formulated in terms of transitions in the down set since such a transition has to occur in order to make ϕ not hold.

The correctness of the HSPP stubborn set method follows immediately from the following theorem by letting $M_0 = M_I$.

Theorem 7 *Let ϕ be a state property, $SG = (V, E)$ be the full state space, $SSG = (V_{SSG}, E_{SSG})$ an SS state space constructed using HSPP stubborn sets, and let $M_0 \in V_{SSG}$. Then:*

$$\exists M \in [M_0\rangle : \phi \notin [M\rangle \Leftrightarrow \exists M_{SSG} \in [M_0\rangle_{SSG} : \phi \notin [M_{SSG}\rangle_{SSG} \qquad \square$$

Proof of Thm. 7. The $\Leftarrow$ direction follows immediately from Thm. 6. We prove the $\Rightarrow$ direction by showing the following for a strictly decreasing sequence of values of n:

(1) There are M_0^n, $t_1^n, \ldots, t_n^n$ and M^n such that $M_0^n \in [M_0\rangle_{SSG}$, $M_0^n\ [t_1^n t_2^n \cdots t_n^n\rangle$ M^n and $\phi \notin [M^n\rangle$.

Initially we get (1) for some finite non-negative value of n from the left hand side of the theorem if we choose $M_0^n = M_0$. If $\phi \notin [M_0^n\rangle$, then $\phi \notin [M_0^n\rangle_{SSG}$, so M_0^n can be chosen as the M_{SSG} and the right hand side of the theorem holds. This happens at the latest when $n = 0$. Therefore, we get our result by showing that if (1) holds for some n and $\phi \in [M_0^n\rangle$, then (1) holds also for some m such that $0 \leq m < n$.

So we assume that $\phi \in [M_0^n\rangle$ holds. Thm. 6 asserts the existence of $t_1'', \ldots, t_h''$ and $M_0'', \ldots, M_h''$ such that $M_0^n = M_0''$, $M_0''[t_1''\rangle_{SSG} M_1''[t_2''\rangle_{SSG} \cdots [t_h''\rangle_{SSG} M_h''$ and $\phi(M_h'')$. We first establish the existence of an occurrence sequence $\bar{M}_0[\bar{t}_1\rangle_{SSG}\bar{M}_1$ $[\bar{t}_2\rangle_{SSG} \cdots [\bar{t}_k\rangle_{SSG}\bar{M}_k$ such that $M_0^n = \bar{M}_0$ and $\{t_1^n, \ldots, t_n^n\} \cap T_{\mathsf{s}}(\bar{M}_j) \neq \emptyset$ for some $1 \leq j \leq k$. We split the proof in two cases.

Case A: If at least one $t_1^n, \ldots, t_n^n$ belongs to $T_{\mathsf{s}}(M_0'') \cup \cdots \cup T_{\mathsf{s}}(M_h'')$, then we just choose $k = h$ and $\bar{t}_i = t_i''$ and $\bar{M}_i = M_i''$ for $1 \leq i \leq h$.

Case B: If none of $t_1^n, \ldots, t_n^n$ is in $T_{\mathsf{s}}(M_0'') \cup \cdots \cup T_{\mathsf{s}}(M_h'')$, then due to D1 and D2′ there is M_h' such that $M_h''[t_1^n \cdots t_n^n\rangle M_h'$ and $M^n[t_1'' \cdots t_h''\rangle M_h'$. Because of $\phi \notin [M^n\rangle$ we know that $\neg\phi(M_h')$. Because $\phi(M_h'')$ holds, there is $t_i^n \in \{t_1^n, \ldots, t_n^n\}$ such that $t_i^n \in \mathsf{down}_\phi$. Therefore, SPP3 gives the desired occurrence sequence.

We can now choose the smallest j such that $\{t_1^n, \ldots, t_n^n\} \cap T_{\mathsf{s}}(\bar{M}_j) \neq \emptyset$. Let i be the smallest number such that $t_i^n \in T_{\mathsf{s}}(\bar{M}_j)$. Due to D1 and D2′ there is M^{n-1} such that $\bar{M}_j[t_1^n \cdots t_n^n\rangle M^{n-1}$ and $M^n[\bar{t}_1 \cdots \bar{t}_j\rangle M^{n-1}$. We have $\phi \notin [M^{n-1}\rangle$, because otherwise $\phi \notin [M^n\rangle$ would not hold. Due to D1 there is M_0^{n-1} such that $\bar{M}_j[t_i^n\rangle_{SSG} M_0^{n-1}[t_1^n \cdots t_{i-1}^n t_{i+1}^n \cdots t_n^n\rangle M^{n-1}$. We thus have (1) for $n-1$. $\square$

12.8 Implementation

We now consider the implementation of the RSPP and HSPP stubborn set methods presented in the previous sections. In Sect. 12.8.1 we show how to construct up, down, and satisfiability sets. In Sect. 12.8.2 we discuss different ways of implementing the conditions D1, D2′, and SPP1-3.

12.8.1 Implementation of Up/Down and Satisfiability Sets

In this section we show how to define up, down and satisfiability sets for the state properties considered in this paper. The construction is in all three cases specified inductively using the syntactical structure of the state properties. We end the section with a proposition which states that the defined up, down, and satisfiability sets posses the up, down, and satisfiability set properties as defined in Def. 31. First we give the definition of up sets.

Definition 34 *Let M be a marking and ϕ a state property. The up set $\mathsf{up}_\phi(M)$ in M is defined as follows. If ϕ holds in M we define $\mathsf{up}_\phi(M) = \emptyset$. If ϕ does not hold in M we define $\mathsf{up}_\phi(M)$ according to the following cases:*

Case $\phi \equiv M(p) \leq k$ *: $\mathsf{up}_\phi(M)$ consists of the transitions which can remove tokens from p and which add at most k tokens to p:*

$$\mathsf{up}_\phi(M) = \{\, t \in T \mid W(p,t) > W(t,p) \wedge W(t,p) \leq k \,\}$$

Case $\phi \equiv M(p) = k$ *: If p contains too few tokens then $\mathsf{up}_\phi(M)$ consists of the transitions which can add tokens and do not require additional tokens to be present on p, and if p contains too many tokens then $\mathsf{up}_\phi(M)$ consists of the transitions which can remove tokens from p and which add at most k tokens:*

$$\mathsf{up}_\phi(M) = \begin{cases} \{\, t \in T \mid W(p,t) < W(t,p) \wedge W(p,t) \leq M(p) \,\} & \text{if } M(p) < k \\ \{\, t \in T \mid W(p,t) > W(t,p) \wedge W(t,p) \leq k \,\} & \text{if } M(p) > k \end{cases}$$

Case $\phi \equiv M(p) \geq k$ *: $\mathsf{up}_\phi(M)$ consists of the transitions which can add tokens and which do not require additional tokens to be present on p:*

$$\mathsf{up}_\phi(M) = \{\, t \in T \mid W(p,t) < W(t,p) \wedge W(p,t) \leq M(p) \,\}$$

Case $\phi \equiv M(p) \neq k$ *: $\mathsf{up}_\phi(M)$ consists of the transitions which can change the marking of p and which do not require additional tokens to be present on p:*

$$\mathsf{up}_\phi(M) = \{\, t \in T \mid W(p,t) \neq W(t,p) \wedge W(p,t) \leq k \,\}$$

Case $\phi \equiv M(p_1) > M(p_2)$ **or** $\phi \equiv M(p_1) \geq M(p_2)$ *:*

$$\mathsf{up}_\phi(M) = \{\, t \in T \mid W(t,p_1) - W(p_1,t) > W(t,p_2) - W(p_2,t) \,\}$$

Case $\phi \equiv M(p_1) = M(p_2)$ *: $\mathsf{up}_\phi(M) =$*

$$\begin{cases} \{\, t \in T \mid W(t,p_1) - W(p_1,t) > W(t,p_2) - W(p_2,t) \,\} & \text{if } M(p_1) < M(p_2) \\ \{\, t \in T \mid W(t,p_1) - W(p_1,t) < W(t,p_2) - W(p_2,t) \,\} & \text{if } M(p_1) > M(p_2) \end{cases}$$

Case $\phi \equiv M(p_1) \neq M(p_2)$ *:*

$$\mathsf{up}_\phi(M) = \{\, t \in T \mid W(t,p_1) - W(p_1,t) \neq W(t,p_2) - W(p_2,t) \,\}$$

Case $\phi \equiv \phi_1 \wedge \phi_2$ *: $\mathsf{up}_\phi(M)$ is the up set of one ϕ_i which does not hold in M:*

$$\big(\neg\phi_1(M) \wedge \mathsf{up}_\phi(M) = \mathsf{up}_{\phi_1}(M)\big) \;\vee\; \big(\neg\phi_2(M) \wedge \mathsf{up}_\phi(M) = \mathsf{up}_{\phi_2}(M)\big)$$

Case $\phi \equiv \phi_1 \vee \phi_2$: $\mathsf{up}_\phi(M) = \mathsf{up}_{\phi_1}(M) \cup \mathsf{up}_{\phi_2}(M)$ □

Next we give the definition of down sets.

Definition 35 *Let M be a marking and ϕ a state property. The down set down_ϕ is defined as follows:*

Case $\phi \equiv M(p) \leq k$: $\mathsf{down}_\phi = \{\, t \in T \mid W(p,t) < W(t,p) \wedge W(p,t) \leq k \,\}$

Case $\phi \equiv M(p) = k$: $\mathsf{down}_\phi = \{\, t \in T \mid W(p,t) \neq W(t,p) \wedge W(p,t) \leq k \,\}$

Case $\phi \equiv M(p) \neq k$: $\mathsf{down}_\phi = \{\, t \in T \mid W(p,t) \neq W(t,p) \wedge W(t,p) \leq k \,\}$

Case $\phi \equiv M(p) \geq k$: $\mathsf{down}_\phi = \{\, t \in T \mid W(p,t) > W(t,p) \wedge W(t,p) < k \,\}$

Case $\phi \equiv M(p_1) > M(p_2)$ **or** $\phi \equiv M(p_1) \geq M(p_2)$:
$\mathsf{down}_\phi = \{\, t \in T \mid W(p_1,t) - W(t,p_1) > W(p_2,t) - W(t,p_2) \,\}$

Case $\phi \equiv M(p_1) = M(p_2)$ **or** $\phi \equiv M(p_1) \neq M(p_2)$:
$\mathsf{down}_\phi = \{\, t \in T \mid W(t,p_1) - W(p_1,t) \neq W(t,p_2) - W(p_2,t) \,\}$

Case $\phi \equiv \phi_1 \wedge \phi_2$ **or** $\phi \equiv \phi_1 \vee \phi_2$: $\mathsf{down}_\phi = \mathsf{down}_{\phi_1} \cup \mathsf{down}_{\phi_2}$ □

Finally, we give the definition satisfiability sets.

Definition 36 *Let M be a marking and ϕ a state property constructed from the atomic state propositions $\{\, \varphi_i \mid i \in I \,\}$. The satisfiability set $\mathsf{sat}_\phi(M)$ in M is defined as follows. If $\phi(M)$ holds then $\mathsf{sat}_\phi(M) = \emptyset$. Otherwise it is defined as follows.*

Case $\phi \equiv \varphi_i$: $\mathsf{sat}_\phi(M) = \{\, i \,\}$

Case $\phi \equiv \phi_1 \vee \phi_2$: $\mathsf{sat}_\phi(M) = \mathsf{sat}_{\phi_1}(M) \cup \mathsf{sat}_{\phi_2}(M)$

Case $\phi \equiv \phi_1 \wedge \phi_2$: $\mathsf{sat}_\phi(M)$ *is the satisfiability set of one ϕ_i which does not hold in M:*
$(\neg\phi_1(M) \wedge \mathsf{sat}_\phi(M) = \mathsf{sat}_{\phi_1}(M)) \vee (\neg\phi_2(M) \wedge \mathsf{sat}_\phi(M) = \mathsf{sat}_{\phi_2}(M))$ □

The following proposition states that the up, down, and satisfiability sets defined above have the required properties. The proof of the proposition is based on structural induction on the state properties and is not contained in this paper.

Proposition 6 *Let M be a marking and assume that $\mathsf{up}_\phi(M) \subseteq T$, $\mathsf{down}_\phi \subseteq T$, and $\mathsf{sat}_\phi(M) \subseteq I$ are constructed according to Def. 34, Def. 35, and Def. 36, respectively. Then the following hold:*

1. *$\mathsf{up}_\phi(M)$ has the up set property in M with respect to ϕ.*
2. *down_ϕ has the down set property with respect to ϕ.*
3. *$\mathsf{sat}_\phi(M)$ has the satisfiability set property in M with respect to ϕ.* □

12.8.2 Implementation of RSPP and HSPP Stubborn

We now consider the implementation of D1, D2′, and SPP1-3. The implementation of D1, D2′ and SPP1 is rather straightforward. Techniques for ensuring D1 and D2′ are well-established (see, e.g., [129] for a survey), and SPP1 can be handled with similar techniques. Below we suggest three implementations of SPP2 and SPP3. The more complex implementations have the potential of leading to better reductions of the state space.

Attractor Set. A simple way to implement SPP2 is to ensure that in each marking M encountered during the SS state space generation we have $\mathsf{up}_{\varphi_i}(M) \subseteq T_{\mathsf{s}}(M)$ for every $i \in \mathsf{sat}_{\phi}(M)$. In the case of SPP3 we also ensure that $\mathsf{down}_{\phi} \subseteq T_{\mathsf{s}}(M)$. This guarantees that the n in the formulation of SPP2 and SPP3 can be chosen to be zero. SPP1 is automatically guaranteed since $\bigcup_{\{\ i \in \mathsf{sat}_{\phi}(M)\ \}} \mathsf{up}_{\varphi_i(M)} \subseteq T_{\mathsf{s}}(M)$ has the up set property in M. This implementation of SPP2 coincides with the attractor set method suggested in [109].

Terminal SCC Detection. A more powerful implementation of SPP2 can be obtained by exploiting strongly connected components (SCCs) and the fact that for a directed graph it is always possible to reach the nodes belonging to some terminal SCC. This fact implies that if all enabled transitions in the stubborn sets used are key transitions, then a sufficient condition for SPP2 and SPP3 to hold is that for every i there exists an occurrence sequence satisfying SPP2 and SPP3 in each of the terminal SCCs of the SS state space. Stubborn sets in which all enabled transitions are key transitions are also referred to as *strong stubborn sets*. Strong stubborn sets are already guaranteed in case of HSPP due to D2′. In case of RSPP, we can obtain strong stubborn sets by simply ensuring also D2′ in addition to D1, SPP1, and SPP2.

Checking that the terminal SCCs satisfy the requirement formulated above can be done on-the-fly when combining a depth-first generation of the SS state space with generation of SCCs by means of TARJAN's algorithm [48]. If a terminal SCC C is about to be completed and the construction of the SS state space is about to backtrack from the marking M_0 then we check that either $\phi(M)$ holds in some $M \in C$ or for each atomic state proposition φ_i we have that $\mathsf{up}_{\varphi_i}(M_0) \subseteq \bigcup_{\{\ M \in C\ \}} T_{\mathsf{s}}(M)$. If we find an atomic state proposition i such that 1) $\mathsf{up}_{\varphi_i}(M_0) \not\subseteq \bigcup_{\{\ M \in C\ \}} T_{\mathsf{s}}(M)$ and 2) the stubborn set in M_0, $T_{\mathsf{s}}^{\mathsf{up}_i}(M_0)$ containing $\mathsf{up}_{\varphi_i}(M_0) - \bigcup_{\{\ M \in C\ \}} T_{\mathsf{s}}(M)$ contains enabled transitions which are not in $T_{\mathsf{s}}(M_0)$, then we extend the stubborn set used in M_0 with $T_{\mathsf{s}}^{\mathsf{up}_i}(M_0)$. The extension of the stubborn set is simple to implement since the union of two stubborn sets are again a stubborn set. SPP3 requires also the check that for every $t \in \mathsf{down}_{\phi}$, there is a $M \in C$ such that $t \in T_{\mathsf{s}}(M)$.

The use of terminal SCCs was first suggested in [124] for a condition which from an implementation point of view resembles SPP2 and SPP3. The condition was later called "S" in [129]. We refer to [124] for additional details about its implementation.

Cycle Detection. An approximation to ensuring that an occurrence sequence exists satisfying SPP2 and SPP3 in each of the terminal SCCs is to ensure the stronger requirement that such an occurrence sequence exists in each of the SCCs. This can

implemented without the use of TARJAN's algorithm, and we can rely on depth-first generation and strong stubborn sets only. The algorithm operates as follows.

Whenever we reach a marking M_1 during the SS state space generation which is on the depth-first search stack, we search backwards in the stack through markings $M_n, M_{n-1}, \ldots, M_1$ and check whether for all i we have that $\mathsf{up}_{\varphi_i}(M_1) \subseteq \bigcup_{j=1}^{n} T_{\mathsf{s}}(M_j)$. For all atomic state proposition i such that $\mathsf{up}_{\varphi_i}(M_1) \not\subseteq \bigcup_{j=1}^{n} T_{\mathsf{s}}(M_j)$ we compute a new stubborn set in M_1, $T_{\mathsf{s}}^{\mathsf{up}_i}(M_1)$ containing $\mathsf{up}_{\varphi_i}(M_1) - \bigcup_{j=1}^{n} T_{\mathsf{s}}(M_j)$, and extend the stubborn set used in M_1 with $T_{\mathsf{s}}^{\mathsf{up}_i}(M)$. Again, SPP3 requires taking also down_{ϕ} into account in the check.

12.9 Applications

In this section we develop stubborn set methods for *boundedness properties* based on the RSPP stubborn set method. The considered boundedness properties are inspired from how boundedness properties of High-level Petri Nets are interpreted at the level of the equivalent PT-net. The purpose of this section is twofold. Firstly, to develop methods for a general set of boundedness properties as such, and secondly to illustrate how the results of this paper can be applied as a tool for developing stubborn set methods for state properties composed of atomic state propositions beyond those considered in this paper.

Best Upper Bounds. An integer k is an *upper bound* for a set of places $P' \subseteq P$ iff $\forall M \in [M_I\rangle : \sum_{p \in P'} M(p) \leq k$. We are interested in finding the minimal such k, denoted the *best upper bound* of P'. One approach is to check the state properties $\phi^k(M) \equiv \sum_{p \in P'} M(p) \geq k$ starting with $k = 0$ and incrementing k until a k_0 is found for which a marking with $\sum_{p \in P'} M(p) \geq k_0$ is not reachable. $k_0 - 1$ is then the best upper bound. A problem which has to be solved before this approach works is that we have not allowed $\sum_{p \in P'} M(p) \geq k$ as an atomic state proposition. However, all that is needed to make our stubborn set algorithm work in this case is to define proper up and down sets for this "new" atomic state proposition. The up set for ϕ^k can be defined as the set of transitions which adds tokens to P' and which does not require additional tokens to be present on P'. The down set can be defined as the transitions which can remove tokens from P' and which produces less than k token on P'. Formally:

$$\begin{aligned}
\mathsf{up}_{\phi^k}(M) &= \{\, t \in T \mid \sum_{p \in P'} W(p,t) < \sum_{p \in P'} W(t,p) \wedge \sum_{p \in P'} W(p,t) \leq \sum_{p \in P'} M(p) \,\} \\
\mathsf{down}_{\phi^k} &= \{\, t \in T \mid \sum_{p \in P'} W(p,t) > \sum_{p \in P'} W(t,p) \wedge \sum_{p \in P'} W(t,p) < k \,\}
\end{aligned}$$

An alternative is to observe that up_{ϕ^k} is independent of k and down_{ϕ^k} can be approximated from above by removing "$\sum_{p \in P'} W(t,p) < k$" from its definition. This means that the stubborn sets used are then independent of k. It therefore suffices to generate just a single SS state space.

Best Lower Bounds. An integer k is a *lower bound* for a set of places $P' \subseteq P$ iff $\forall M \in [M_I\rangle : \sum_{p\in P'} M(p) \geq k$. We are interested in finding the maximal such k, referred to as the *best lower bound* of P'. Similarly to the upper bound case we consider state properties of the form : $\phi^k(M) \equiv \sum_{p\in P'} M(p) \leq k$ starting with $k = 0$ and continuing until the first k_0 is found for which a marking with $\sum_{p\in P'} M(p) \leq k_0$ is reachable. k_0 is then the best lower bound. The up and down sets for the atomic state proposition $\sum_{p\in P'} M(p) \leq k$ can be defined as shown below. If one is interested in generating only a single SS state space when finding the best lower bound of a set of places, then the dependency of k can be eliminated like for the best upper bound case by approximating the up and down sets to become independent of k.

$$\begin{aligned} \mathsf{up}_{\phi^k}(M) &= \{\, t \mid \sum_{p\in P'} W(p,t) > \sum_{p\in P'} W(t,p) \wedge \sum_{p\in P'} W(t,p) \leq k \,\} \\ \mathsf{down}_{\phi^k} &= \{\, t \mid \sum_{p\in P'} W(p,t) < \sum_{p\in P'} W(t,p) \wedge \forall p \in P' : W(p,t) \leq k \,\} \end{aligned}$$

12.10 Experimental Results

We have implemented the RSPP stubborn set method on top of the state space tool of Design/CPN [16]. The prototype implements the *Attractor Set* and *Cycle Detection* algorithms given in Sect. 12.8.2. The construction of stubborn sets is based on the *strong component algorithm* described in [129] adapted to take the condition SPP1 into account.

Tables 12.1 and 12.2 gives numerical data on the reduction obtained with the two implemented algorithms on some examples. For PETERSON's and HYMAN's mutual exclusion algorithms we consider the two state properties corresponding to mutual exclusion (Mutual Excl.), and that each of the two processes can reach the critical section (Reach. of CS). For the Reader/Writer protocol we consider three state properties: the writers can get write access (Reach. of Write), the three readers can get read access (Reach. of Read), and the protocol guarantees exclusive write (Excl. Write). For the Reader/Write protocol we consider a configuration with 2 writers and 3 readers. For the Master/Slave protocol we consider two properties: a marking is reachable in which the master has received a response from all slaves which in turn have returned to their idle state (DoneIdle), and the master never continues before having received a response from all slaves (DoneWIdle). For the Master/Slave protocol we consider configurations with 3,5 and 6 slaves.

Table 12.1 gives information about the performance of the *Cycle Detection* algorithm. The table contains two main parts. In the Up set Driven part the construction of the stubborn set is initiated from the transitions in the up set and it favours stubborn sets containing transitions in the up set. In the Up set/Enabling Driven part the construction of the stubborn set is initiated from the transitions in the up set but it does not favour stubborn sets containing transitions in the up set. The DFG columns represent a depth-first generation of the state space with *early termination*, i.e., as soon as a marking has been found where the state property holds the generation stops. The CG columns represent a complete generation, i.e., the generation continues even though a

Model/ Property	Up set Driven					Up set/Enabling Driven				
	DFG		CG		Min	DFG		CG		Min
	Nodes	Arcs	Nodes	Arcs	Length	Nodes	Arcs	Nodes	Arcs	Length
Peterson										
Reach. of CS	9	8	36	59	9/6	6	5	34	51	6/6
Mutual Excl.	-	-	48	84	-	-	-	47	79	-
Hyman										
Reach. of CS	5	4	60	95	5/5	8	7	38	46	8/8
Mutual Excl.	19	19	64	106	17/12	18	20	45	57	12/12
Reader/Writer										
Reach. of Write	3	2	77	221	3/3	7	6	14	17	7/7
Reach. of Read	3	2	85	197	3/3	14	17	14	17	7/7
Excl. Write	-	-	46	111	-	-	-	24	37	-
Master/Slave										
DoneIdle-3	60	67	229	548	60/15	30	30	130	152	30/15
DoneIdle-5	230	277	7,837	32,412	230/23	691	790	1,654	2,172	483/23
DoneIdle-6	516	622	46,781	233,276	513/27	1,744	2,084	5,600	7,658	1053/27
DoneWIdle-3	-	-	231	562	-	-	-	185	272	-
DoneWIdle-5	-	-	7,839	32,494	-	-	-	3,745	6,592	-
DoneWIdle-6	-	-	46,783	233,470	-	-	-	16,769	31,168	-

Table 12.1: Experimental results – Cycle detection algorithm.

marking has been found where the state property holds. This gives information about how large a state space the corresponding algorithm considers in the worst-case. For those properties where no marking is reachable where the property holds, depth-first and complete generation coincide, and only the numbers for the complete generation is given. The entries in the Min Length columns are of the form x/y, where x gives the number of nodes in a shortest path leading to a marking where the property holds for the depth-first generation (if such one exists), and y gives the corresponding number for the complete generation. This gives information about how good the algorithm is at providing short witness paths.

Table 12.2 gives information about the performance of the *Attractor Set* algorithm and the full state space. The table contains two main parts. The Full State Space part lists the size of the full state space. In the Attractor Set Method part, the DFG and CG columns represent depth-first generation with early termination, and complete generation, respectively. The BFG column represents a breadth-first generation with early termination. The entries in the Min Length are of the form x/y, where x gives the number of nodes in a shortest path leading to a marking where the property holds for the depth-first generation (if such one exists), and y gives the corresponding number for the BFG generation. It was proved in [109] that the latter equals the number of nodes in one of the shortest paths of the full state space. All state spaces reported on in this section were generated in less than 2 minutes on a 166 Mhz PII PC.

If we first compare the numbers for the complete generation (CG) in Tables 12.1 and 12.2 then in all cases the Up set/Enabling Driven implementation gives much better reduction than the Attractor Set Method. The Up set Driven implementation gives approximately the same reduction as the Attractor Set Method. As a consequence of this the Up set/Enabling Driven implementation outperforms the Attractor Set Method in the cases where the state property does not hold. If we consider the set of state properties which holds then for the first three examples the Attractor Set Method seems slightly better than the cycle detection algorithm in terms of yielding small state spaces

Model/ Property	Full State Space		Attractor Set Method						
			DFG		BFG		CG		Min.
	Nodes	Arcs	Nodes	Arcs	Nodes	Arcs	Nodes	Arcs	Length
Peterson	58	116							
Reach. of CS			9	8	10	11	39	67	9/5
Mutual Excl.			-	-	-	-	50	90	-
Hyman	80	160							
Reach. of CS			7	6	14	17	60	95	7/5
Mutual Excl.			36	42	49	76	64	106	30/12
Reader/Writer	136	532							
Reach. of Write			3	2	3	2	77	221	3/3
Reach. of Read			3	2	3	2	85	197	3/3
Excl. Write			-	-	-	-	118	419	-
Master/Slave									
DoneIdle-3	232	588	61	79	212	520	229	548	45/15
DoneIdle-5	7,840	32,656	1,623	4,144	7,700	31,966	7,837	32,412	575/23
DoneIdle-6	46,784	233,856	9,819	37,155	33,092	156,317	46,701	233,276	1566/27
DoneWIdle-3	232	588	-	-	-	-	231	562	-
DoneWIdle-5	7,840	32,656	-	-	-	-	7,839	32,494	-
DoneWIdle-6	46,784	233,856	-	-	-	-	46,783	233,470	-

Table 12.2: Experimental results – Full state space and attractor set algorithm.

and generating short witness paths. However, when we turn to the larger Master/Slave example, then the Up set/Enabling Driven implementation again outperforms the Attractor Set Method in terms of reduction and it is still able to generate a short witness path. The intuitive reason for the Up set/Enabling Driven implementation to be better in these cases is that if the state property is located "far" from the initial marking (as is the case for the Master/Slave example), then the Attractor Set Method and to some extent also the Up Set Driven implementation have a high risk of investigating wrong branches of the state space first. For the cases where the state property holds the Up set/Enabling Driven implementation therefore seems to represent a good solution to the trade-off between generating short witness paths and considering large state spaces.

12.11 Conclusions

We have presented two new stubborn set methods for reasoning about state properties. The method for determining whether a reachable marking exists in which a given state property holds was based on ideas first presented in [109]. The main difference between our new method and [109] is in how progress towards the state property is ensured. We have replaced the *always progress* condition of [109] with the weaker *eventual progress* condition, which have the potential of leading to better reduction results, and which contains the always progress condition as a special case. We have demonstrated the potential on some practical case studies by means of an implementation of the new method. The case studies showed that the new stubborn set method is significantly better when the state property does not holds in any reachable marking. When a reachable marking exists in which the state property does hold, then it represents good solution to the trade-off between short witness paths and large state spaces. From an implementation point of view the more powerful implementations which we have suggested for the eventual progress condition requires *strong stubborn sets*, whereas for the always progress implementation it suffices to use *weak stubborn sets*.

We have extended the first stubborn set method to obtain a second stubborn set method representing a novel technique for determining, e.g., whether a marking is a home marking, and for checking liveness of only a single transition. Like existing methods for checking liveness of transitions it relies on strong stubborn sets, but it does not require *ignoring* to be eliminated.

As an application to boundedness properties we have illustrated the use of the results presented in this paper as a tool for developing stubborn set methods for state properties beyond those considered in the paper. In fact, it can be observed that we only directly referred to PT-nets in the implementation of up and down sets, and hence the suggested methods can be transferred to other modelling formalisms – provided that they allow for the definition of sets of transitions satisfying the properties of up and down sets.

List of Figures

List of Tables

Index

Bibliography

[1] K. Ajami, S. Haddad, and J. M. Ilié. Exploiting Symmetry in Linear Time Temporal Logic Model Checking: One Step Beyond. In B. Steffen, editor, *Proceedings of TACAS'98*, volume 1384 of *Lecture Notes in Computer Science*, pages 52–67. Springer-Verlag, 1998.

[2] R.D. Andersen, J.B. Jørgensen, and M. Pedersen. Occurrence Graphs with Equivalent Markings and Self-symmetries. Master's thesis, Department of Computer Science, University of Aarhus, Denmark, 1991. Only available in Danish: Tilstandsgrafer med ækvivalente mærkninger og selvsymmetrier.

[3] G. Balbo, S. Bruell, O. Chen, and G. Chiola. An Example of Modeling and Evaluation of a Concurrent Program Using Colored Stochastic Petri Nets: Lamport's Fast Mutual Exclusion Algorithm. In T.-Y. Feng, editor, *IEEE Transactions on Parallel and Distributed Systems*, pages 221–240. IEEE Computer Society, 1992.

[4] T. Basten. *In Terms of Nets – System Design with Petri Nets and Process Algebra*. PhD thesis, Eindhoven University of Technology, 1998.

[5] D. Bertsekas and R. Gallager. *Data Networks.* Prentice-Hall, 1992.

[6] J. Billington, G. R. Wheeler, and M. C. Wilbur-Ham. PROTEAN: A High-level Petri Net Tool for the Specification and Verification of Communication Protocols. In K. Jensen and G. Rozenberg, editors, *High-level Petri Nets*, pages 560–575. Springer-Verlag, 1991.

[7] R. Brgan and D. Poitrenuad. An Efficient Algorithm for the Computation of Stubborn Sets of Well-Formed Petri Nets. In G. De Michelis and M. Diaz, editors, *Proceedings of ICATPN'95*, volume 935 of *Lecture Notes in Computer Science*, pages 121–140. Springer-Verlag, 1995.

[8] R. E. Bryant. Graph Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, 1986.

[9] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic Model Checking: 10^{20} States and Beyond. *Information and Computation*, 98(2):142–170, 1992.

[10] C. Capellmann and H. Dibold. Formal Specificaions of Services in an Intelligent Network Using High-Level Petri Nets. In *Case study Proceedings of the 15th In-*

ternational Conference on Application and Theory of Petri Nets (ICATPN'94), 1994.

[11] 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.

[12] L. Cherkasova, V. Kotov, and T. Rokicki. On Net Modelling of Industrial Size Concurrent Systems. In *Case study Proceedings of the 15th International Conference on Application and Theory of Petri Nets (ICATPN'94)*, 1994.

[13] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. On Well-Formed Coloured Nets and Their Symbolic Reachability Graph. In K. Jensen and G. Rozenberg, editors, *High-level Petri Nets*, pages 373–396. Springer-Verlag, 1991.

[14] S. Christensen, K. Jensen, and L.M. Kristensen. *Design/CPN Occurrence Graph Manual*. Department of Computer Science, University of Aarhus, Denmark. On-line version: `http://www.daimi.au.dk/designCPN/`.

[15] S. Christensen and L. O. Jepsen. Modelling and Simulation of a Network Management System Using Hierarhical Coloured Petri Nets. In E. Mosekilde, editor, *Proceedings of the 1991 European Simulation Multiconference*, pages 47–52. Society for Computer Simulation, 1991.

[16] S. Christensen, J. B. Jørgensen, and L. M. Kristensen. Design/CPN - A Computer Tool for Coloured Petri Nets. In E. Brinksma, editor, *Proceedings of TACAS'97*, volume 1217 of *Lecture Notes in Computer Science*, pages 209–223. Springer-Verlag, 1997.

[17] S. Christensen and J.B. Jørgensen. Analysis of Bang and Olufsen's BeoLink Audio/Video System Using Coloured Petri Nets. In P. Azéma and G. Balbo, editors, *Proceedings of ICATPN'97*, volume 1248 of *Lecture Notes in Computer Science*, pages 387–406. Springer-Verlag, 1997.

[18] S. Christensen and L. M. Kristensen. State Space Analysis of Hierarchical Coloured Petri Nets. In B. Farwer, D. Moldt, and M-O. Stehr, editors, *Proceedings of Workshop on Petri Nets in System Engineering (PNSE'97) Modelling, Verification, and Validation*, pages 32–43. Universität Hamburg, Germany, Fachberich Informatik, 1997. Publication No. 205.

[19] S. Christensen and L. M. Kristensen. State Space Analysis of Hierarchical Coloured Petri Nets. In W. v. d. Aalst, J.-M. Colom, F. Kordon, G. Kotsis, and D. Moldt, editors, *Petri Net Approaches for Modelling and Validation*, number 1 in LINCOM Studies in Computer Science. LINCOM Europe, 1999. To appear.

[20] S. Christensen and K. H. Mortensen. *Design/CPN ASK-CTL Manual*. Department of Computer Science, University of Aarhus, Denmark, 1996. Available via `http://www.daimi.au.dk/designCPN/libs`.

[21] S. Christensen and L. Petrucci. Modular State Space Analysis of Coloured Petri Nets. In G. De Michelis and M. Diaz, editors, *Proceedings of ICATPN'95*, volume 935 of *Lecture Notes in Computer Science*, pages 201–217. Springer-Verlag, 1995.

[22] E. Clake, E.A. Emerson, S. Jha, and A.P. Sistla. Symmetry Reductions in Model Checking. In A.J. Hu and M.Y. Vardi, editors, *Proceedings of CAV'98*, volume 1427 of *Lecture Notes in Computer Science*, pages 147–159. Springer-Verlag, 1998.

[23] E. M. Clarke, R. Enders, T. Filkorn, and S. Jha. Exploiting Symmetries in Temporal Model Logic Model Checking. *Formal Methods in System Design*, 9, 1996.

[24] E. M. Clarke, O. Grumberg, M. Minna, and D. Peled. State Space Reduction using Partial Order Techniques. *International Journal on Software Tools for Technology Transfer*, 2(3):279–287, 1999.

[25] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic Verification of Finite State Concurrent Systems using Temporal Logic. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[26] E.M. Clarke, T. Filkorn, and S. Jha. Exploiting Symmetries in Temporal Model Logic Model Checking. In C. Courcoubetis, editor, *Proceedings of CAV'93*, pages 450–462. Springer-Verlag, 1993.

[27] E.M. Clarke and J. M. Wing. Formal Methods: State of the Art and Future Directions. *ACM Computing Surveys*, 28(4):626–643, December 1996. Report by the Working Group on Formal Methods for the ACM Workshop on Strategic Directions in Computing Research.

[28] H. Clausen and P. Ryberg. Validation and Performance Analysis of Network Algoritms by Coloured Petri Nets. In *Proceedings of 5th International Workshop on Petri Nets and Performacnce Models PNPM'93*, pages 280–289. IEEE Computer Society Press, 1993.

[29] D. E. Comer. *Computer Networks and Internets.* Prentice-Hall International, Inc., 1997.

[30] G. Coulouris, J. Dollimore, and T. Kindberg. *Distributed Systems - Concepts and Design.* Addison-Wesley, 2nd edition, 1998.

[31] E. A. Emerson. *Temporal and Modal Logic*, volume B of *Handbook of Theoretical Computer Science*, chapter 16, pages 995–1072. Elsevier, 1990.

[32] E. A. Emerson, S. Jha, and D. Peled. Combining Partial Order and Symmetry Reduction. In E. Brinksma, editor, *Proceedings of TACAS'97*, volume 1217 of *Lecture Notes in Computer Science*, pages 35–49. Springer-Verlag, 1997.

[33] E. A. Emerson and A. P. Sistla. Utilizing Symmetry when Model Checking under Fairness Assumptions: An Automata-theoretic Approach. In *Proceedings*

of CAV'95, volume 939 of *Lecture Notes in Computer Science*, pages 309–324. Springer-Verlag, 1995.

[34] E. A. Emerson and A. Prasad Sistla. Symmetry and Model Checking. *Formal Methods in System Design*, 9, 1996.

[35] E.A. Emerson. *Formal Methods in System Design, Volume 9, Numbers 1/2 — Special Issue on Symmetry in Automatic Verification.* Kluwer Academic Publishers, 1996.

[36] J. Esparza. Model Checking using Net Unfoldings. *Science of Computer Programming*, 23:151–195, 1994.

[37] J. Esparza. Decidability and Complexity of Petri Net Problems – An Introduction. In W. Reisig and G. Rozenberg, editors, *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science*, pages 374–428. Springer-Verlag, 1998.

[38] J. Esparza, S. Römer, and W. Vogler. An Improvement of McMillan's Unfolding Algorithm. In *Proceedings of TACAS'97*, volume 1055 of *Lecture Notes in Computer Science*, pages 87–106. Springer-Verlag, 1996.

[39] Formal Methods Europe. FME Tools Database. `http://www.csr.ncl.ac.uk/projects/FME/InfRes/tools/`.

[40] J. C. Fernandez. An Implementation of an Efficient Algorithm for Bisimulation Equivalence. *Science of Computer Programming*, 13:219–236, 1989/90.

[41] G.A. Findlow, G.S. Gerrand, J. Billington, and R.J. Fone. Modelling ISDN Supplementary Services Using Coloured Petri Nets. In *Proceedings of Communications '92*, pages 37–41, Sydney, Australia, 1992.

[42] D.J. Floreani, J. Billington, and A. Dadej. Designing and Verifying a Communications Gateway Using Colored Petri Nets and Design/CPN. In J. Billington and W. Reisig, editors, *Proceedings of ICATPN'96*, volume 1091 of *Lecture Notes in Computer Science*, pages 153–171. Springer-Verlag, 1996.

[43] A. Foroughipour. Construction of OS-graphs with Permutation Symmetries of a Coloured Petri Net. Master's thesis, Department of Computer Science, University of Aarhus, Denmark, 1994.

[44] H. Genrich. Predicate/Transition Nets. In K. Jensen and G. Rozenberg, editors, *High-level Petri Nets*, pages 3–43. Springer-Verlag, 1991.

[45] H.J. Genrich and R.M. Shapiro. Formal Verification of an Arbiter Cascade. In K. Jensen, editor, *Proceedings of ICATPN'92*, volume 616 of *Lecture Notes in Computer Science*, pages 205–223. Springer-Verlag, 1992.

[46] R. Gerth, R. Kuiper, D. Peled, and W. Penczek. A Partial Order Approach to Branching Time Logic Model checking. In *Proc. of 3rd Israel Symposium on the Theory of Computing and Systems*, pages 130–140. IEEE, 1995.

[47] R. Gerth, D. Peled, M. Vardi, and P. Wolper. Simple On-the-fly Automatic Verification of Linear Temporal Logic. In *Proc. Protocol Specification, Testing, and Verification*, pages 3–18. Chapman & Hall, 1995.

[48] A. Gibbons. *Algorithmic Graph Theory*. Cambridge University Press, 1985.

[49] P. Godefroid. Using Partial Orders to Improve Automatic Verification Methods. In *Proceedings of Computer-Aided Verification '90*, volume 531 of *Lecture Notes in Computer Science*, pages 175–186. Springer-Verlag, 1990.

[50] P. Godefroid. *Partial-Order Methods for the Verification of Concurrent Systems, An Approach to the State-Explosion Problem*, volume 1032 of *Lecture Notes in Computer Science*. Springer-Verlag, 1996.

[51] Bernd Grahlmann. The PEP Tool. In Orna Grumberg, editor, *Proceedings of CAV'97*, volume 1254 of *Lecture Notes in Computer Science*, pages 440–443. Springer-Verlag, 1997.

[52] J. C. Grégoire. State Space Compression in SPIN with GETSs. In *Proc. Second Spin Workshop*. American Mathematical Society, DIMACS/32, 1996. `http://netlib.bell-labs.com/netlib/spin/whatispin.html`.

[53] V. Gyuris and A. P. Sistla. On-the-Fly Model Checking Under Fairness That Exploits Symmetry. In *Proceeding of CAV'97*, volume 1254 of *Lecture Notes in Computer Science*, pages 232–243. Springer-Verlag, 1997.

[54] S. Haddad. A Reduction Theory for Coloured Nets. In K. Jensen and G. Rozenberg, editors, *High-level Petri Nets*, pages 399–425. Springer-Verlag, 1991.

[55] S. Haddad and J. M. Ilié. Symbolic Reachability Graph and Partial Symmetries. In G. De Michelis and M. Diaz, editors, *Proceedings of ICATPN'95*, volume 935 of *Lecture Notes in Computer Science*, pages 238–257. Springer-Verlag, 1995.

[56] D. Harel. Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming*, 8:231–274, 1987.

[57] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall International, 1985.

[58] G. J. Holzmann. State Compression in SPIN: Recursive Indexing and Compression Training Runs. In *Proc. Third Spin Workshop*, 1997. `http://netlib.bell-labs.com/netlib/spin/whatispin.html`.

[59] G. J. Holzmann and D. Peled. Partial Order Reduction of the State Space. In *Proceedings of first SPIN Workshop*, 1995. `http://netlib.bell-labs.com/netlib/spin/whatispin.html`.

[60] G. J. Holzmann and A. Puri. A Minimized Automaton Representation of Reachable States. *International Journal on Software Tools for Technology Transfer*, 2(3):270–278, 1999.

[61] G.J. Holzmann. *Design and Validation of Computer Protocols.* Prentice-Hall International Editions, 1991.

[62] PEP Homepage.
`http://theoretica.informatik.uni-oldenburg.de/~pep/`.

[63] P. Huber, A.M. Jepsen, L.O. Jepsen, and K. Jensen. Reachability Trees for High-level Petri Nets. *Theoretical Computer Science*, 45:261–292, 1986. Also in K. Jensen and G. Rozenberg (eds.): High-level Petri Nets. Theory and Application, 319-350, Springer-Verlag, 1991.

[64] J. M. Ilié and K. Ajami. Model Checking Through Symbolic Reachability Graph. In M. Bidoit and M. Dauchet, editors, *Proceedings of TAPSOFT'97*, volume 1214 of *Lecture Notes in Computer Science*, pages 213–224. Springer-Verlag, 1997.

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

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

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

[68] K. Jensen. Condensed State Spaces for Symmetrical Coloured Petri Nets. *Formal Methods in System Design*, 9, 1996.

[69] K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 3, Practical Use*. Monographs in Theoretical Computer Science. Springer-Verlag, 1997.

[70] K. Jensen. An Introduction to the Practical Use of Coloured Petri Nets. In W. Reisig and G. Rozenberg, editors, *Lectures on Petri Nets II*, volume 1492 of *Lecture Notes in Computer Science*, pages 237–292. Springer Verlag, 1998.

[71] J. B. Jørgensen and L. M. Kristensen. Computer Aided Verification of Lamport Fast Mutual Exclusion Algorithm Using Coloured Petri Nets and Occurrence Graphs with Symmetries. Technical report, Department of Computer Science, University of Aarhus, Denmark, February 1997. DAIMI PB-512.

[72] J. B. Jørgensen and L. M. Kristensen. Verification by State Spaces with Equivalence Classes. Technical report, Department of Computer Science, University of Aarhus, Denmark, February 1997. DAIMI PB-515.

[73] J. B Jørgensen and L. M. Kristensen. Verification of Coloured Petri Nets Using State Spaces with Equivalence Classes. In D. Moldt B. Farwer and M-O. Stehr, editors, *Proceedings of Workshop on Petri Nets in System Engineering (PNSE'97) Modelling, Verification, and Validation*, pages 20–31. Universität Hamburg, Germany, Fachberich Informatik, 1997. Publication No. 205.

[74] J. B. Jørgensen and L. M. Kristensen. Computer Aided Verification of Lamport's Fast Mutual Exclusion Algorithm Using Coloured Petri Nets and Occurrence Graphs with Symmetries. *IEEE Transactions on Parallel and Distributed Systems*, 10(7):714–732, July 1999.

[75] J. B. Jørgensen and L. M. Kristensen. Verification of Coloured Petri Nets Using State Spaces with Equivalence Classes. In W. v. d. Aalst, J.-M. Colom, F. Kordon, G. Kotsis, and D. Moldt, editors, *Petri Net Approaches for Modelling and Validation*, number 1 in LINCOM Studies in Computer Science. LINCOM Europe, 1999. To appear.

[76] J.B. Jørgensen. Construction of Occurrence Graphs with Permutation Symmetries Aided by the Backtrack Method. Technical report, Department of Computer Science, University of Aarhus, Denmark, 1997. DAIMI PB-516, February 1997.

[77] J.B. Jørgensen and L.M. Kristensen. Computer Aided Verification of Lamport Fast Mutual Exclusion Algorithm Using Coloured Petri Nets and Occurrence Graphs with Symmetries. Technical report, Department of Computer Science, University of Aarhus, Denmark, 1996. On-line version: `http://www.daimi.au.dk/designCPN/`.

[78] J.B. Jørgensen and L.M. Kristensen. *Design/CPN Condensed State Space Tool Manual*. Department of Computer Science, University of Aarhus, Denmark, 1996.
Online: `http://www.daimi.au.dk/designCPN/`.

[79] J.B. Jørgensen and L.M. Kristensen. Efficient Calculation of the Size of the O-graph from the OS-graph. Technical report, Department of Computer Science, University of Aarhus, Denmark, 1996. On-line version: `http://www.daimi.au.dk/designCPN/`.

[80] J.B. Jørgensen and K.H. Mortensen. Modelling and Analysis of Distributed Program Execution in BETA Using Coloured Petri Nets. In J. Billington and W. Reisig, editors, *Proceedings of ICATPN'96*, volume 1091 of *Lecture Notes in Computer Science*, pages 249–268. Springer-Verlag, 1996.

[81] R. Kaivola. Using Compositional Predorders in the Verification of a Sliding Window Protocol. In *Proceedings of CAV'97*, volume 1254 of *Lecture Notes in Computer Science*, pages 48–57. Springer-Verlag, 1997.

[82] B. W. Kernighan and D. M. Ritchie. *The C Programming Language*. 2nd edition. Prentice Hall, 1988.

[83] D. E. Knuth. *The Art of Computer Programming – Sorting and Searching*, volume 3. Addison Wesley Longman, second edition, 1998.

[84] I. Kokkarinen, A. Valmari, and D. Peled. Relaxed Visibility Enhances Partial Order Reduction. In *Proceedings of CAV'97*, volume 1254 of *Lecture Notes in Computer Science*, pages 328–339. Springer-Verlag, 1997.

[85] L. M. Kristensen, S. Christensen, and K. Jensen. The Practitioner's Guide to Coloured Petri Nets. *International Journal on Software Tools for Technology Transfer*, 2(2):98–132, December 1998.

[86] L. M. Kristensen and A. Valmari. Finding Stubborn Sets of Coloured Petri Nets Without Unfolding. In J. Desel and M. Silva, editors, *Proceedings of ICATPN'98*, volume 1420 of *Lecture Notes in Computer Science*, pages 104–123. Springer-Verlag, 1998.

[87] L. M. Kristensen and A. Valmari. Improved Question-Guided Stubborn Set Methods for State Properties. Technical report, Department of Computer Science, University of Aarhus, Denmark, 1999. DAIMI PB-543. Submitted to the 21st International Conference on Application and Theory of Petri Nets (ICATPN'00).

[88] L. Lamport. A Fast Mutual Exclusion Algorithm. In *ACM Transactions on Computer Systems, Vol. 5, No. 1*, pages 1–11. Association for Computing Machinery, 1987.

[89] L. Lorentsen and L. M. Kristensen. Modelling and Analysis of a Danfoss Flowmeter System using Coloured Petri Nets. Submitted to the 21st International Conference on Application and Theory of Petri Nets.

[90] N. A. Lynch. *Distributed Algorithms.* Morgan Kaufmann, 1997.

[91] M. A. Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets.* Series in Parallel Computing. Wiley, 1995.

[92] K. L. McMillan. A Technique of State Space Search Based on Unfolding. *Formal Methods in System Design*, 6(1):45–65, 1995.

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

[94] R. Milner. *Communication and Concurrency.* Prentice-Hall International Series in Computer Science. Prentice-Hall, 1989.

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

[96] F. Moller. *The Edinburgh Concurency Workbench.* Department of Computer Science, University of Edinburgh, version 6.1 edition, October 1992.

[97] G. Monchelet, S. Christensen, H. Demmou, M. Paludetto, and J. Porras. Analysing a Mechatronic System with Coloured Petri Nets. *Int. Journal on Software Tools for Technology Transfer*, 2(2):160–167, 1998.

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

[99] Design/CPN Online. `http://www.daimi.au.dk/designCPN/`.

[100] E. Pastor, O. Roig, J. Cortadella, and R. Badia. Petri Net Analysis Using Boolean Manipulation. In R. Valette, editor, *Proceedings of ICATPN'94*, volume 815 of *Lecture Notes in Computer Science*, pages 416–435. Springer-Verlag, 1994.

[101] D. Peled. All from One, One for All: On Model Checking Using Representatives. In *Proceedings of CAV'93*, volume 697 of *Lecture Notes in Computer Science*, pages 409–423. Springer-Verlag, 1993.

[102] D. Peled. Combining Partial Order Reductions with On-the-fly Model Checking. *Formal Methods in System Design*, 8(1):39–64, 1996.

[103] D. Peled. Ten Years of Partial Order Reduction. In *Proceedings of CAV'98*, volume 1427 of *Lecture Notes in Computer Science*, pages 17–28. Springer-Verlag, 1998.

[104] C.A. Petri. *Kommunikation mit Automaten.* Bonn: Institut für Instrumentelle Mathematik, Schriften des IIM Nr. 2, 1962.

[105] J.L. Rasmussen and M. Singh. Designing a Security System by Means of Coloured Petri Nets. In J. Billington and W. Reisig, editors, *Proceedings of ICATPN'96*, volume 1091 of *Lecture Notes in Computer Science*, pages 400–419. Springer-Verlag, 1996.

[106] M. Raynal. *Algorithms for Mutual Exclusion.* North Oxford Academic, 1986.

[107] W. Reisig. *Petri Nets*, volume 4 of *EACTS Monographs on Theoretical Computer Science.* Springer-Verlag, 1985.

[108] G. Scheschonk and M. Timpe. Simulation and Analysis of a Document Storage System. In R. Valette, editor, *Proceedings of ICATPN'94*, volume 815 of *Lecture Notes in Computer Science*, pages 454–470. Springer-Verlag, 1994.

[109] K. Schmidt. Stubborn Sets for Standard Properties. In S. Donatelli and J. Kleijn, editors, *Proceedings of ICATPN'99*, volume 1639 of *Lecture Notes in Computer Science*, pages 46–65. Springer-Verlag, 1999.

[110] R.M. Shapiro. Validation of a VLSI Chip Using Hierarchical Coloured Petri Nets. *Journal of Microelectronics and Reliability, Special Issue on Petri Nets*, 1991.

[111] A.P. Sistla, L. Milliades, and V. Gyuris. SMC: A Symmetry based Model Checker for Verification of Liveness Properties. In *Proceedings of CAV'97*, volume 1254 of *Lecture Notes in Computer Science*, pages 464–467. Springer-Verlag, 1998.

[112] The SMV System. `http://www.cs.cmu.edu/~modelcheck/smv.html`.

[113] G. Tel. *Introduction to Distributed Algorithms.* Cambridge University Press, 1994.

[114] M. Tiusanen. Symbolic, Symmetry, and Stubborn Set Searches. In *Proceedings of ICATPN'94*, volume 815 of *Lecture Notes in Computer Science*, pages 511–530. Springer-Verlag, 1994.

[115] A. Tokmakoff and J. Billington. An Approach to the Analysis of Interworking Traders. In *Proc. of ICATPN'99*, volume 1639 of *Lecture Notes in Computer Science*, pages 127–146. Springer-Verlag, 1999.

[116] J. Toksvig. Design and Implementation of a Place Invariant Tool for Coloured Petri Nets. Master's thesis, Department Computer Science, University of Aarhus, Denmark, 1995.

[117] The PROD Tool. `http://www.tcs.hut.fi/prod/`.

[118] The SPIN Tool. `http://netlib.bell-labs.com/netlib/spin/whatispin.html`.

[119] J.D. Ullman. *Elements of ML Programming*. Prentice-Hall, 1997.

[120] A. Valmari. Error Detection by Reduced Reachability Graph Generation. In *Proceedings of the 9th European Workshop on Application and Theory of Petri Nets*, pages 95–112, 1988.

[121] A. Valmari. *State Space Generation: Efficiency and Practicality*. PhD thesis, Tampere University of Technology, Tampere, Finland, 1988. Publication 55.

[122] A. Valmari. Compositional State Space Generation. In *Proceedings of ICATPN'90*, pages 43–63, 1990.

[123] A. Valmari. A Stubborn Attack on State Explosion. In *Proceedings of Computer-Aided Verification '90*, volume 531 of *Lecture Notes in Computer Scienc*, pages 156–165. Springer-Verlag, 1990.

[124] 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*, pages 491–515. Springer-Verlag, 1990.

[125] A. Valmari. Stubborn Sets of Coloured Petri Nets. In G. Rozenberg, editor, *Proceedings of ICATPN'91*, pages 102–121, 1991.

[126] A. Valmari. Compositional Analysis with Place-Bordered Subnets. In *Proceedings of ICATPN'94*, volume 815 of *Lecture Notes in Computer Science*, pages 531–547. Springer-Verlag, 1994.

[127] A. Valmari. State of the Art Report: Stubborn Sets. *Petri Net Newsletter*, 46:6–14, 1994.

[128] A. Valmari. Compositionality in State Space Verification Methods. In J. Billington and W. Reisig, editors, *Proceedings of ICATPN'96*, volume 1091 of *Lecture Notes in Computer Science*, pages 29–56. Springer-Verlag, 1996.

[129] A. Valmari. The State Explosion Problem. In W. Reisig and G. Rozenberg, editors, *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science*, pages 429–528. Springer-Verlag, 1998.

[130] 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*, pages 187–194. IEEE Computer Society Press, 1996.

[131] A. Valmari and I. Kokkarinen. Unbounded Verification Results by Finite-State Compositional Techniques: 10^{any} States and Beyond. In *Proceedings of International Conference on Application of Concurrency to System Design*, pages 75–85. IEEE Computer Society, 1998.

[132] M. Vardi and P. Wolper. An Automata-Theoretic Approach to Automatic Program Verification. In *In Proc. of IEEE Symposium on Logic in Computer Science*, pages 322–331, 1986.

[133] K. Varpaaniemi. On Combining the Stubborn Set Method with the Sleep Set Method. In *Proceedings of ICATPN'94*, volume 815 of *Lecture Notes in Computer Science*, pages 548–567. Springer-Verlag, 1994.

[134] K. Varpaaniemi. *On The Stubborn Set Method in Reduced State Space Generation*. PhD thesis, Helsinki University of Technology, 1998.

[135] W. Visser. Memory Efficient State Storage in SPIN. In *Proc. of Second Spin Workshop*. American Mathematical Society, DIMACS/32, 1996. `http://netlib.bell-labs.com/netlib/spin/whatispin.html`.

[136] 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 *Lecture Notes in Computer Science*. Springer-Verlag, 1993.

[137] J. Xu and J. Kuusela. Analyzing the Execution Architecture of Mobile Phone Software with Coloured Petri Nets. *Int. Journal on Software Tools for Technology Transfer*, 2(2):133–143, 1998.