# Chapter 4

# Summary

This chapter sums up the work done as part of this thesis as well as applications of the work conducted and directions for future work. The contributions of the work is summarised in Sect. 4.1, applications by the thesis author and others of the tools and methods developed as part of the thesis are summarised in Sect. 4.2, and interesting directions for future research are summarised in Sect. 4.3. For more detailed outlines of applications and future work, please refer to Sects. 2.5 and 3.5.

## 4.1 Contributions

This section summarises the contributions made as part of this thesis. As indicated by the structure of the thesis, work has been done within two areas: behavioural verification of formal models by means of reachability graphs and behavioural visualisation of formal models. The main contribution within the field of reachability graph analysis of formal models consists of improving algorithms for efficient storage of reachability graphs [T2, T1]. The main contributions within the field of visualisation of formal models consist of the development of a tool for visualisation of the behaviour of formal models, the BRITNeY Suite [T3, C2], an application of the BRITNeY Suite visualisation tool to build a model-based prototype of a protocol facilitating communication between nodes in a mobile ad-hoc network, and a formal framework for visualisations [T5]. In the following we provide a more detailed perspective on the main contributions.

### Extension of the sweep-line method to handle liveness properties

Prior to our work conducted in [T1], the sweep-line method could only check invariant properties [25, 104] and even then it was not possible in general to provide a trace from the initial state to a violating state using internal memory only, as parts of the reachability graph have been removed from memory (though work by Kristensen and Mailund existed which use external memory to provide error traces [105]).

Using our work in [T1], it becomes possible to check liveness properties, e.g., formulated using Linear Temporal Logic (LTL) [74], as well as providing error traces, using internal memory only, by storing a very compact representation of the reachability graph in internal memory. The method is shown to use significantly less memory on models with a clear notion of progress, while using only a small overhead for methods with little or no notion of progress.

### Making the hash-compaction reduction technique complete

The hash-compaction technique stores the reachability graph in a highly compact manner by compressing state descriptors using a hash function. The drawback is that the hash function may not be injective, causing hash collisions, where two or more states have the same compressed state descriptor. As only one state with each compressed state descriptor is explored, this leads to parts of the reachability graph remaining unexplored. Using more than one hash function [155] the number of hash collisions can be reduced, but the basic problem, namely that the method is incomplete, persists.

The ComBack method [T2] extends the hash compaction reduction reduction technique by maintaining a spanning tree of the reachability graph rooted in the initial state. This makes it possible to resolve hash collisions on-the-fly during exploration, thereby making the method complete. The method is shown to perform reasonably well on both academic and real-life examples, trading execution time for memory usage compared to ordinary reachability graph exploration.

### Development of the BRITNeY Suite visualisation tool

Prior to the development of the BRITNeY Suite [T3, C2], a lot of visualisation tools existed. Most of these tools were closed source, used a closed architecture, were tied to a single tool for formal modelling, or had more than one of these problems. Furthermore, CPN Tools [C1, 33] had no means of behavioural visualisation except using Gallash and Kristensen's COMMS/CPN library [53] for communication with external programs. This required implementing a Remote Procedure Call mechanism in each case as well as writing visualisations from scratch in a standard programming language such as Java or C++.

The BRITNeY Suite is open source and has an open architecture, which allows extension of the tool by means of plug-ins or scripts. Furthermore, it is independent of the modelling tool. This makes it possible to use the BRITNeY Suite with CPN Tools, which is also it main application. The BRITNeY Suite can also be used with other modelling tools and even other applications as well. While the BRITNeY Suite offers more than 20 plug-ins out of the box, it is easy to extend the tool to provide custom visualisations as required, due to the pluggable architecture and open source license.

### Development of a model-based prototype of a protocol facilitating communication between nodes in a mobile ad-hoc network

During the B2NCW project [101] at Ericsson Denmark A/S, Telebit [47], with the resources and time available, it was deemed impossible to implement a prototype using real hardware of the protocol facilitating communication between nodes in a mobile ad-hoc network. Instead a prototype of a different, simpler, protocol was developed using real hardware, but as the extended protocol was deemed a better choice, a prototype based on a formal model was developed of this protocol.

The model-based prototype of the extended protocol was developed as a coloured Petri net model and a visualisation was developed to allow people who are not formal methods-experts to experiment with it. During the project, the BRITNeY Suite was developed and tested in a real-life setting.

**Development of a formal framework for describing visualisations of the behaviour of formal models**

We have devised a framework [T5], which regards both visualisations and formal models as game transitions systems, which are labelled transition systems where the transitions are separated into controllable and uncontrollable transitions. The two are executed simultaneously in a manner so that controllable transitions of the formal model are synchronised with uncontrollable transitions of the visualisation and vice versa. We require that whenever a controllable transition can be executed in the formal model or the visualisation, a corresponding uncontrollable transition can be executed in the other. The intuition is that actions initiated by the formal model (controllable transitions in the formal model) are shown to the user, and stimulation of the visualisation (controllable transitions in the visualisation) are reflected in the formal model. This approach has many advantages over previous visualisation tools. Firstly, this approach does not require changes to most formalisms, as their dynamic behaviours are usually stated using labelled transition systems as semantical domain. Secondly, it is easy to extend tools supporting visualisations in this manner as it is possible to provide a uniform interface for visualisations. Furthermore, it is difficult to forget visualisation elements as we require that the visualisation is able to accommodate any transition allowed in the formal model, so the only way to ignore a transition in the model is to do so explicitly and therefore deliberately.

## 4.2 Applications

This section sums up applications of the tools and methods described in this thesis. While the work of this thesis, as mentioned earlier, falls into two categories, applications of the methods are only available within the field of behavioural visualisation of formal models. The reason is that one of the verification papers, the one describing the ComBack method [T2], has only recently been published at the time of writing. The other verification method, the extended version of the sweep-line method [T1], is mainly useful for checking more complex properties, such as liveness using Linear Temporal Logic, and this does not have easy accessible tool support in tools supporting the algorithm, making real-life applications difficult. The lack of real-life applications has diminished the requirement for improvements of the algorithm. Applications of the BRITNeY Suite fall into three categories: use of the BRITNeY Suite for visualisation, use of the BRITNeY Suite for meta-visualisation, and other uses of the BRITNeY Suite. Each of these categories will be explained in the following.

### Visualisation

The BRITNeY Suite has of course been used for visualisation of formal models in numerous cases. One such example is of course Kristensen, the author of this thesis, and Nørgaard's model-based prototyping of a protocol facilitating communication between nodes in mobile ad-hoc networks [T4]. Another application performed by Jørgensen and Lassen is visualisation of a formal model of blanc loan applications [94] for requirements engineering. The BRITNeY Suite has also been used by Jørgensen, Lassen, and Aalst to verify that a formal model of requirements for an electronic patient record [144] corresponds to the intended system.

**Meta-visualisation**

The BRITNeY Suite has also been used to implement visualisation of other formalisms by translating them into CP-nets. One such example is visualisation of UML [131] sequence diagrams, which is done independently by Machade et al. in [114] and Ribeiro and Fernandes in [145]. Both of these papers also present an industrial example of this. The BRITNeY Suite is also used to implement a workflow simulator based on coloured workflow nets [162] by Kristian Bisgaard Lassen from the University of Aarhus, Denmark. This work uses the idea of visualisations as games to develop a single visualisation, which can be used for any coloured workflow net model.

**Other Applications of the BRITNeY Suite**

The BRITNeY Suite has also been used in other ways. Riahi Bilel from Faculté des Sciences de Tunis (FST) uses the BRITNeY Suite to integrate algorithms written in Java with a CPN model by writing a visualisation that does not actually show anything, but only performs the required calculations. The author of this thesis has used the BRITNeY Suite to load CPN models from the command-line, which is not possible or feasible to implement using CPN Tools. György Balogh from Vanderbilt University, USA, integrate the CPN simulator into Morse et al.'s HLA (High Level Architecture) [84,128], by writing glue code as an extension of the BRITNeY Suite.

## 4.3 Future Work

As can be seen in Sects. 4.1 and 4.2, the goal of this thesis, namely to construct and improve methods for locating errors in computer systems, has been reached. We have focused on a tool for visualisation of the behaviour of formal models, and shown, via our own and other people's case studies, that this tool and method is indeed very useful for improving formal models. We have improved the state-of-the-art of methods for reachability graph, thereby making it possible to analyse even larger systems using this analysis approach. Still a lot of work remains, though. In Sects. 2.5.1 and 3.5.3 we provide several interesting directions for future work. The rest of this thesis is dedicated to briefly summarise this.

**Improvement of methods and tools for behavioural verification using reachability graphs**

The ComBack method [T2] can be extended and combined with other reduction techniques in different ways. Interesting ways to do that is to combine it with partial order reduction techniques [28, 136], which are known to reduce the in-degrees, as this would minimise the number of reconstructions required by the algorithm. It would also be interesting to combine the method with the sweep-line method, so states in front of the sweep-line are cached, also reducing the number of reconstructions.

In order for the methods described to be really useful, they should be implemented in tools which make it easy to use them on real models. Such a tool would need to implement user-friendly ways to specify properties to check. This includes user-friendly ways to specify properties that hold in a given state and natural ways to combine such properties into more complex properties stating facts about the dynamics of the formal model. We suggest looking at SPIN's never-claims [77], which formulate properties in the same language as the formal models, Petri's Facts [139], which are Petri net transitions which must never be enabled, and Cardelli and Gordon's ambient logic [16], which state properties of the ambient calculus using a syntax that closely resembles the syntax used to specify the ambient calculus models. All of these approaches use the formalism itself or something very similar to specify properties, which is very different from CPN Tools, which uses Standard ML and temporal logics such as LTL and CTL to specify properties.

Furthermore, to support development of even better reduction techniques, a test-suite must be devised to test the methods. Such a test-suite must be able to automatically perform a large number of executions of verifications of various properties using different reduction techniques on a varied selection of models. Furthermore, it should be easy to navigate the results and track improvement over time.

### Improvement of the BRITNeY Suite for behavioural visualisation

The BRITNeY Suite has already been used in numerous applications, but especially for more advanced applications improvements can be made. Firstly, the documentation is not completely satisfactory, and could be improved. Furthermore, some technical choices would be made differently today. The first choice would be to use SOAP web-services [63] instead of the currently used XML-RPC [170]. SOAP, today, enjoys wider acceptance and supports the Web Service Definition Language (WSDL) [20] for describing services, which makes it easier to integrate the BRITNeY Suite with other tools for formal modelling thanks to the wide availability of implementations of both SOAP and WSDL. As no such implementation exist for Standard ML, the implementation language of the CPN simulator used in CPN Tools [C1, 33], a complete replacement of the current implementation is unlikely to happen, however. Another, more promising, change is to use the Eclipse [41] platform rather than the custom plug-in mechanism used today. This would make it easier to develop for the BRITNeY Suite, as the Eclipse IDE could be used to debug and single-step through code, something which is not possible today. Furthermore, such a change would enable the integration of an IDE for developing (Java) code with a tool for specification using formal models and for visualising said specifications. This would make it possible to keep the specification very close to the actual implementation.

The implementation of visualisations as games, as implemented by the BRITNeY Suite at the time of writing, mainly focuses on events, which is not completely satisfactory for formalisms that are both state and event oriented, such as CP-nets. It would be nice to make synchronisation of states easy as well.

### Use of behavioural visualisation to convey results of behavioural verification

The final direction for future work, we present in this thesis is a combination of the two areas dealt with in this thesis, namely behavioural verification of formal models and behavioural visualisation of formal models. The idea is to use a visualisation to present counter-examples to users. Violations of simple properties can often be visualised easily, e.g., to prove that invariant properties do not hold, we can just show an execution sequence leading to a violating state, whereas violations of other kinds of properties are not that easy to present to the user. It is quite difficult to present counter-examples to the existence of winning strategies of games or to the validity of a CTL formula, however, as they are basically annotated reachability graphs.

The idea is to let the user contend against the computer to prove the existence of a winning strategy or the validity of the CTL formula. The user believes that the property holds, while the computer has a counter-example, proving that the property does not hold. The user selects certain transitions to execute and the computer selects other transitions. The user selects transitions using a visualisation, and the transitions selected by the computer are shown using the same visualisation. The idea is that at some point the computer will select an unanticipated transitions (or the user is unable to select an anticipated transition), which convince the user that winning strategy cannot exist/that the CTL formula does not hold. This can be because the model is wrong (if the computer performs an action not permitted by the specification or because the user cannot choose a transition which should be possible according to the specification), in which case the model needs to be changed. It is also possible that the specification is wrong, in which case the specification has to be modified and the model updated accordingly.