SPIN Tutorial: How to become a SPIN Doctor

(Extended Abstract)

Theo C. Ruys

Faculty of Computer Science, University of Twente.
P.O. Box 217, 7500 AE Enschede, The Netherlands.
http://www.cs.utwente.nl/~ruys/

Abstract. Spin is a model checker for the verification of software systems. Spin uses a high level language called Promela to specify systems descriptions. The goal of this tutorial is to introduce novice users to both Promela and Spin. The tutorial itself is divided into two parts. The Basic Spin part is targeted towards novice users of Spin. The Advanced Spin part of the tutorial could also be of considerable interest to intermediate Spin users.

1 Introduction

Spin [2,3] is a model checker for the verification of software systems. During the last decade, Spin has been successfully applied to trace logical design errors in distributed systems, such as operating systems, data communications protocols, switching systems, concurrent algorithms, railway signaling protocols, etc. [7]. Spin checks the logical consistency of a specification; it reports on deadlocks, unspecified receptions, flags incompleteness, race conditions, and unwarranted assumptions about the relative speeds of processes [2]. Spin is considered to be one of the most powerful and advanced model checkers (freely) available today. Spin is widely distributed and has a large user base.

SPIN Beginner's Tutorial. Spin uses a high level language called Promela (PROcess MEta LAnguage) to specify systems descriptions. The purpose of this tutorial at the Spin 2002 Workshop is to introduce novice users to both Promela and Spin. The first part of the tutorial (Basic Spin) gives an introduction to Promela and presents an overview of the validation and verification features of Spin. The material will be illustrated by several demo's using Xspin, the graphical user interface to Spin. The second part of the tutorial (Advanced Spin) discusses guidelines to construct efficient Promela models and shows how to use Spin in the most effective way. Topics to be discussed include: Spin's optimisation algorithms, directives and options to tune verification runs with Spin, guidelines for effecitve Promela modelling, using Spin as a debugger, validation management, etc.

Although the "SPIN Beginner's Tutorial" at the SPIN 2002 Workshop will be targeted towards novice users of SPIN, this 'extended abstract' focuses more on some advanced SPIN topics. The reason for not including 'beginner's material' is twofold. First of all, an abstract is clearly too short to present a thorough introduction to SPIN and PROMELA. But more importantly, there is already a wealth of excellent introductory material available on SPIN; either online or in print (see below).

The organisation of this extended abstract is as follows. To guide the beginning SPIN user through all the available SPIN material, Section 2 provides some pointers to SPIN resources. Section 2 also presents a general procedure that can be followed when verifying a property with XSPIN. Section 3 presents several guidelines with respect to the effective use of PROMELA and SPIN. Some of these guidelines may be too concise to be fully understood. Most of the topics in this extended abstract, however, are discussed in much greater depth in the author's PhD Thesis [10].

2 Basic SPIN

SPIN Material. As said, this paper is not meant as a tutorial for PROMELA or SPIN. Users not yet familiar with the basic operations of SPIN have to turn to other sources of introductory information on SPIN. The usual first piece of advice for beginning users is as always: RTFM – *Read The Fine Manual*. And this time, the documentation is really fine. Apart from the book on the first version of SPIN by Gerard Holzmann [3], the recent versions of SPIN come with extensive online documentation in accessible `.html` format on both the tool and the PROMELA language. For beginning users of SPIN, the following documents are highly recommended:

- *(online)* The Basic SPIN Manual [11] is a general introduction to the language PROMELA and the tool SPIN. This document only discusses the basic use of SPIN. It does not discuss extensions to the language that have been introduced in the later versions of SPIN, i.e. 2.x and 3.x, which are documented elsewhere [15].
- *(online)* The document Guidelines for Verification with XSPIN [12] explains how to use SPIN using the graphical interface XSPIN, which runs independently from SPIN itself and helps by generating the proper SPIN commands based on menu selections.
- And albeit slightly older, [4] is still a good tutorial to get started with SPIN. Naturally, the newer language and tool additions are not covered, but the core of the system – which has not changed over the years – is nicely introduced.

After browsing these documents, one is advised to plunge into the comprehensive set of examples and exercises:

- SPIN Verification Examples and Exercises – a sample set of exercises with SPIN [13].

General procedure to verify a (general) property ϕ on a PROMELA model M using the model checker SPIN:

1. *Sanity check.* Perform some interactive and random simulation runs on the model M and the property ϕ either using XSPIN or SPIN.
2. *Partial check.* Use SPIN's bitstate hashing mode to quickly sweep over the state space. SPIN's bitstate hashing mode is fast and if there are some silly mistakes in the model, chances are high, that SPIN will find them quickly. This 'partial check' is especially useful if the model M is big and it is estimated that the verification will take considerable time.
3. *Exhaustive check.* Run an exhaustive check on the model M and the property ϕ. If the exhaustive verification fails because there is not enough memory to hold the complete state space, there are several ways to proceed:
   - *Compression.* Try one of SPIN's memory compression options, to reduce the size of the state space.
     - The compile-time option `-DCOLLAPSE` collapses state vectors sizes by up to 80% to 90% [6].
     - The compile-time option `-DMA=N` makes `pan` use a minimized DFA encoding [8] for the state space assuming a maximum of `N` bytes in the state vector. This option is very effective, but will also increase the running time of the verification run considerably.

     Both options can be combined.
   - *Optimisations.* Make sure that the model is optimised in terms of the number of states and the size of the state vector. Follow the guidelines in Section 3 to optimise the model M as aggressively as possible.
   - *Abstractions.* If the memory compression options do not work (or are not really an option due the implications on the time of the verification run), one should try to make the model M smaller by making *abstractions* of the model. Go back to step 1 and try to verify the abstracted model.
   - *Bitstate hashing:* If the other methods do not work to get M verified, one might use SPIN's bitstate hashing or hash compaction verification modes to *partially* verify the model.

Fig. 1. Verification of a property ϕ for a PROMELA model M using the model checker SPIN.

The investment into the exercises will be well spent in the sense that one will get a good feeling of the systems and PROMELA models that can be analysed with SPIN.

For the intermediate to advanced user, the online documentation contains reference information on all language constructs [14] and a concise language reference by Rob Gerth [1]. The SPIN community is quite active in testimony whereof the (at least) yearly SPIN Workshops, which are being organised since 1995. The proceedings of these workshops – which are publicly available online via the SPIN home-page [2] – contain a wealth of information on, among others:

- discussions on new and significant algorithms within SPIN;
- contributions and proposals to improve or extend SPIN;
- reports on (successful) industrial applications with SPIN;
- proven best practices when applying SPIN.

Validation Procedure. XSPIN is a so-called *Integrated Validation Environment* (IVE) on top of SPIN; it allows the user to edit, simulate and verify PROMELA models. Most users start using SPIN through XSPIN. For casual use and small to moderate verification projects, XSPIN suffices. And even for the more advanced user of SPIN, XSPIN is very convenient as it releases the user of remembering all options and directives to tune the verification runs with SPIN: most of these options can be set via dialog boxes within XSPIN.

Although XSPIN is user-friendly and straightforward to use, most beginning SPIN users do not know where to start to effectively apply XSPIN (or SPIN) to check a property. Fig. 1 presents a validation procedure to verify a model M against a property ϕ using XSPIN. Step 1. and 2. of Fig. 1 require some user guidance and inspection but are generally quite fast. Step 3. may take much longer but after pressing the Run button does not need any additional user input.

3 Advanced SPIN

In this section we discuss some more advanced topics with respect to application of SPIN. We focus on the effective use of the modelling language PROMELA. In the tutorial at the SPIN 2002 Workshop other more pragmatic issues will be discussed as well.

Extreme Modelling. Now that model checking tools in general and SPIN in particular are becoming more widespread in use [5], these tools are starting to be applied by people that only want to press the button and that do not know precisely what is 'under the hood' of such verification tools. Press-the-button verification is only feasible for small to medium-sized applications. Industrial-size applications need aggressive use of the modelling language, the properties to be checked and the verification tool itself. There is generally a big difference in efficiency in the models developed by a 'casual' user and the models developed by an 'expert' user. Moreover, the 'expert' user knows how to exploit the directives and options of the model checker to optimise the verification runs. Efficient use of model checking tools seems to require an 'assembler programming' approach to model building: use all tricks of the model checker to minimise the state space of the model and make the verification process as efficient as possible. The 'expert' verification engineer resembles the seasoned programmer, who not only has a deep knowledge and understanding of data structures and algorithms but also knows the options and directives to tune the programming tools that he or she is using.

From XSPIN*'s* Help*, Reducing Complexity dialog box:*
When a verification cannot be completed because of computational complexity; here are some strategies that can be applied to combat this problem.

0. *Slicing.* Run the Slicing Algorithm (in the Run Menu) to find potential redundancy in your model for the stated properties.
1. *Abstraction.* Try to make the model more general, more abstract. Remember that you are constructing a verification model and not an implementation. SPIN's strength is in proving properties of *interactions* in a distributed system (the implicit assumptions that processes make about each other) – its strength is *not* in proving things about local *computations*, data dependencies, etc.
2. *Redundancy.* Remove everything that is not directly related to the property you are trying to prove: redundant computations, redundant data. *Avoid counters*; avoid incrementing variables that are used for only book-keeping purposes. The Syntax Check in the Run Menu option will warn about the gravest offenses.
3. *Channels.* Asynchronous channels are a significant source of complexity in verification. Use a synchronous (rendez-vous) channel where possible. Reduce the number of slots in asynchronous channels to a minimum (use 2, or 3 slots to get started).
4. *Intermediate processes.* Look for processes that merely transfer messages. Consider if you can remove processes that only copy incoming messages from one channel into another, by letting the sender generate the final message right away. If the intermediate process makes choices (e.g. to delete or duplicate, etc.), let the sender make that choice, rather than the intermediate process.
5. *Local computations.* Combine local computations into `atomic` or `d_step` sequences.
6. *Temporary data.* Avoid leaving scratch data around in variables. You can reduce the number of states by, for instance, resetting local variables that are used inside `atomic` sequences to zero at the end of those sequences; so that the scratch values aren't visible outside the sequence. Alternatively: introduce some extra global 'hidden' variables for these purposes (see the WhatsNew.html document [15]). Use the predefined variable "_" as a write-only scratch variable wherever possible.
7. *Combine behaviour.* If possible to do so: combine the behaviour of two processes into a single one. Generalise behaviour; focus on coordination aspects (i.e. the interfaces between processes), rather than the local computation inside processes.
8. *Exploit PO.* Try to exploit the partial order reduction strategies. Use the `xr` and `xs` assertions (see WhatsNew.html [15]); avoid sharing channels between multiple receivers or multiple senders. Avoid merging independent data-streams into a single shared channel.

Fig. 2. The Reducing Complexity guidelines of the XSPIN 3.4.x Help.

Guidelines to construct PROMELA models for efficient verification with SPIN:

- The macro processor `m4` is more powerful than `cpp` and can be used to generalise PROMELA models.
- Beware of infinite loops in the presence of `atomic` and `d_step` clauses.
- A non-deterministic random construct should be constructed using an `if`-clause.
- Do not use an array of `bits` in Promela; use a (self-defined) `bitvector` instead.
- Variables whose values are always smaller than `16` should be declared as `unsigned` variables.
- User defined types can be efficiently implemented in PROMELA using (`inline`) macros and `d_step` constructs.
- Lossy channels can best be modelled by letting the sending process 'lose' messages or by a 'stealing daemon' process.
- A multicast or broadcast application is best modelled on top of a multicast or broadcast *service*.
- Use *local* variables for variables that are only used within a single process. If a local variable is to be used in a `never` claim, one should define it as a `local` *global* variable.
- PROMELA processes that may terminate should not be created last in a PROMELA model, unless you want the terminating processes to be replaced by new processes.
- Do not use an unguarded monitor process with an `assert` statement to check for invariance if the original model contains a `timeout` statement.
- Changing the layout of the state vector can have (positive) effects on the effectiveness of SPIN's bitstate hashing mode.
- When checking an invariant property with SPIN, use a monitor process with a guarded `assert` statement.

Fig. 3. Summary of the PROMELA and SPIN 'recipes' presented in [9,10].

Fortunately, it is not necessary to become an 'expert' verification engineer to use SPIN *effectively*. Several pragmatic guidelines and rules of thumb have been identified over the last few years which can be applied by novice and intermediate users to develop verification-effective PROMELA models (see below).

Optimisation Order. With model checking tools there is – just as with programming – a trade-off between time and space requirements. For the model checking process, however, the space requirements are much more important than the time requirements. With respect to effective model checking with SPIN, the following optimisation order should be adopted:

1. *Number of states.* Because of the state space explosion, it is crucial to reduce the number of states as much as possible. So reduction of the number of states is the first consideration.

2. *State vector size.* The minimization of the size of the state vector (i.e. the amount of memory which is needed to encode a single state) is the next concern.
3. *Size of search stack.* Our next priority lies with the minimisation of SPIN's depth-first search stack of states.
4. *Verification time.* Only in the last case, reduction of the verification time should be taken into account.

SPIN has several optimisation algorithms to make verification runs more effective, for instance: partial order reduction, minimised automaton encoding of states, state vector compression and bitstate hashing. SPIN supports several command-line options and directives to tune these optimisation algorithms. Not surprisingly, many of these options are related to the trade-off between space and time requirements. Within XSPIN, these options and directives can be accessed and modified via: Run → Set Verification Parameters → Set Advanced Options. These options and directives are concisely explained in XSPIN's Help.

Reducing Complexity. In Fig. 1 we mentioned that one should *optimise* the PROMELA model to make the verification with SPIN feasible. Users that are new to SPIN, however, might not know what is meant by an 'optimised' PROMELA model.

The best advice to reduce the complexity of a PROMELA model stems from the help system of the XSPIN program itself. Under Help, Reducing Complexity, Gerard Holzmann has listed several rules of thumb that should be applied first to reduce the complexity of the PROMELA model under verification. For reference, we have included this list of guidelines in Fig. 2. The SPIN user who already lives by all these rules-of-thumb, is on the right track.

Additional Guidelines. For his PhD Thesis [10], the author has investigated several 'expert' techniques to optimise both the modelling and verification process when using SPIN. These techniques are mostly concerned with the minimisation of the number of states or the reduction of the state vector. The proposed techniques are verified and checked using numerous controlled experiments with SPIN itself. Fig. 3 summarises most lessons learned from [9,10]. In the tutorial at the SPIN 2002 Workshop a few of these guidelines will be discussed in greater depth.

4 Conclusions

SPIN is considered to be one of the most powerful and advanced model checkers (freely) available today. When provided with a model M and a property ϕ to be verified, *in principle*, SPIN comes up with a result fully automatically. Problems arise when the state space of the PROMELA model is too large to be checked exhaustively. Although users of SPIN do not have to know what is happening 'under-the-hood' of SPIN, one should obey certain 'rules-of-thumb' to reduce the complexity of PROMELA models as much as possible. SPIN users should be aware of these guidelines when constructing their verification models.

Acknowledgements. Gerard Holzmann, the Spin master, is thanked for his approval to reprint the Reducing Complexity guidelines of Xspin in this paper.

References

1. R. Gerth. Concise Promela Reference. *Accessible from [2].*
2. G. J. Holzmann. Spin homepage: http://netlib.bell-labs.com/netlib/spin/.
3. G. J. Holzmann. *Design and Validation of Computer Protocols.* Prentice Hall, Englewood Cliffs, New Jersey, USA, 1991.
4. G. J. Holzmann. Tutorial: Design and Validation of Protocols. *Computer Networks and ISDN Systems*, 25(9):981–1017, 1993.
5. G. J. Holzmann. Spin Model Checking - Reliable Design of Concurrent Software. *Dr. Dobb's Journal*, pages 92–97, October 1997.
6. G. J. Holzmann. State Compression in Spin: Recursive Indexing and Compression Training Runs. In *Proceedings of SPIN97, the Third International Workshop on SPIN*, University of Twente, Enschede, The Netherlands, April 1997. Also available from URL: http://netlib.bell-labs.com/netlib/spin/ws97/gerard.ps.Z.
7. G. J. Holzmann. The Model Checker Spin. *IEEE Transactions on Software Engineering*, 23(5):279–295, May 1997.
8. G. J. Holzmann and A. Puri. A Minimized Automaton Representation of Reachable States. *Software Tools for Technology Transfer (STTT)*, 3(1), 1999.
9. T. C. Ruys. Low-Fat Recipes for Spin. In K. Havelund, J. Penix, and W. Visser, editors, *SPIN Model Checking and Software Verification, Proceedings of the 7th International SPIN Workshop (SPIN'2000)*, volume 1885, pages 287–321, Stanford, California, USA, August 2000.
10. T. C. Ruys. *Towards Effective Model Checking.* PhD thesis, University of Twente, Enschede, The Netherlands, March 2001. *Available from the author's homepage.*
11. Spin Online Documentation. Basic Spin Manual. *Accessible from [2].*
12. Spin Online Documentation. Guidelines for Verification with Xspin – Spin Verifier's Roadmap: using Xspin. *Accessible from [2].*
13. Spin Online Documentation. Spin Verification Examples and Exercises. *Accessible from [2].*
14. Spin Online Documentation. Spin Version 3.3: Language Reference – Man-Pages and Semantics Definition. *Accessible from [2].*
15. Spin Online Documentation. What's New in Spin Versions 2.0 and 3.0 – Summary of changes since Version 1.0. *Accessible from [2].*