

Tutorial: Proving Properties of Concurrent Systems with SPIN

Gerard J. Holzmann
AT&T Bell Laboratories
Murray Hill, New Jersey 07974, USA
gerard@research.att.com

Extended Abstract

SPIN is an on-the-fly model checking system for finite state systems, that is optimized for the verification of linear time temporal logic (LTL) properties.¹ SPIN's input language, PROMELA, can be used to specify concurrent systems with dynamically changing numbers of interacting processes, where process interactions can be either synchronous (rendez-vous) or asynchronous (buffered).

In the tutorial we will examine some of the algorithms that determine SPIN's functionality and performance. After a brief summary of the automata theoretic foundation of SPIN, we consider the methodology for LTL model checking, the recognition of Büchi acceptance conditions, cycle detection, and the handling of very large verification problems.

Automata Theoretic Framework

The semantics of PROMELA is defined in terms of a standard labeled transition system. The language is founded on the notion of *executability*. A first rule determines for each type of statement whether it is executable or blocked in a given system state. A second rule determines how the given statement modifies a given system state if and when it is executed. As one small example: a plain condition such as $(a > b)$ is a self-contained type of primitive statement in PROMELA: it is executable when the condition holds, and it has no effect on the system state when executed (other than the changing of the control-state of the process executing the statement). To avoid the obvious, all PROMELA expressions are required to be *pure* (side-effect free).

The behavior of a concurrent system, that is a dynamically changing ensemble of disjoint or interacting processes, is defined with in terms of interleaving semantics. Processes can interact by manipulating shared data objects: variables, data structures, or message queues. Message passing through shared queues is either synchronous (i.e., executed in an atomically linked combination of a matching send and receive operation on a given rendezvous port), or asynchronous, through finite and typed message buffers.

SPIN can compute the interleaving product of a set of concurrent processes in several different ways. The default algorithm is a straight exhaustive reachability analysis [H91]. For problem sizes that preclude exhaustive verification because of their size, a high-coverage approximation of an exhaustive run can be performed in minimal amounts of memory [H88],[H95]. The user has also the option of applying a partial order reduction algorithm to the search (either exhaustive or bitstate) that provably preserves all safety and liveness properties [HP94]. Finally, a weak fairness constraint can be imposed on the search, based on a direct application of Choueka's flag

1. All source is available via anonymous ftp from the machine netlib.att.com, directory /netlib/spin.

construction method [C74],[CVWY92].

LTL Properties and Büchi Automata

In the currently distributed version of SPIN it is the user's responsibility to convert an LTL formula into a Büchi automaton (in PROMELA: a `never claim`) for the purposes of verification. Two recent improvements have made it possible to allow the user to specify an LTL formula directly, and leave the translation into a Büchi automaton to the software [BCG95],[GPVW95]. We have implemented the algorithm from [GPVW95], and plan to integrate it into the SPIN system in an upcoming release.

The Büchi acceptance conditions require the verifier to reliably detect the presence or absence of acceptance cycles in the reachability graph. Since SPIN performs all verifications on-the-fly, in a minimum amount of memory, the cycle detection method must conform to those constraints.

Memory-Efficient Cycle Detection

SPIN uses a nested depth-first search algorithm to detect both acceptance- and non-progress cycles on-the-fly [H91],[CVWY92]. The algorithm is implemented with just two bits of overhead per reachable state, using the encoding method detailed in [GH93]. The use of Tarjan's classic algorithm [T72] would require 64 bits of overhead per state (for storing two 32 bit numbers: the *dfs*-number and *lowlink*-number).

By using a nested depth-first search algorithm for cycle detection, we give up the ability to detect all possible variations of a cycle, but preserve the ability to detect and reproduce at least one cycle of the type searched for, provided that at least one such cycle exists. Because both non-progress and acceptance cycles in SPIN always constitute a counter-example to a correctness requirement, establishing only the absence or presence of such cycles suffices for the purposes of verification. After all, it takes no more than a single counter-example to disprove any correctness property.

Bitstate Hashing

SPIN also incorporates a competitive implementation of the frugal bitstate hashing or *supertrace* technique [H88],[H95]. We will give an analytical argument for its expected performance and compare those estimates with empirical tests. With this algorithm, again just two bits of memory are needed to store any given reachable state. The two bit addresses are computed with two statistically independent hash functions.

Industrial Applications

The bitstate hashing technique has made it possible to apply formal verification to problem sizes that would ordinarily have remained well beyond the scope of automated verification tools. We will summarize one of the recent experiences with formal verification based on this technique in a two-year case study performed in cooperation with AT&T International Switching [C91],[H94].

In this study, named the *NewCoRe* project, a routine implementation of an ISDN protocol was pursued by a team of five people, with a methodology based on formal verification. A total of 10,000 mechanical verification runs were performed, 145 LTL properties were formalized and proven to be satisfied for the final design, and a total of 112 design errors were detected. Many of the design errors could be traced back to the original design requirements.

Design for Verifiability

The number of both academic and industrial applications of SPIN is steadily growing. At the start of 1995, we estimated there to be well over 1,000 installations of the tool. Though this is encouraging, a sobering thought is that the number of systems that is designed without any benefit from formal verification tools, of this type or any other, still completely dominates the industry. How long will it take for the wider availability of verification tools to have a more measurable impact?

When Charles Babbage proposed the use of his analytical engine in 1825 to improve the accuracy of printing astronomical and logarithmic tables, he noted:

“The great importance of having accurate tables is admitted by all who understand their uses; but the multitude of errors really occurring is comparatively little known. . . .”
“. . . [it] is really extraordinary that, when it was demonstrated that all tables are capable of being computed by machinery [. . .] the Astronomer Royal did not become the most enthusiastic supporter of an instrument which could render such invaluable service to his own science.”

Which only goes to say: in the long run, things do tend to change...

References

- [C91] J. Chaves, ‘Formal methods at AT&T, an industrial usage report,’ *Proc. 4th FORTE Conference*, Sydney, Australia, 1991, pp. 83–90.
- [C74] Y. Choueka, ‘Theories of automata on ω -tapes: a simplified approach,’ *Journal of Computer and System Science*, Vol. 8, 1974, pp. 117–141.
- [BCG95] G. Bhat, R. Cleaveland, and O. Grumberg, ‘Efficient on-the-fly model checking for CTL*,’ *Proc. 10th Symp. on Logic in Computer Science*, San Diego, CA, 1995.
- [CVWY92] C. Courcoubetis, M. Vardi, P. Wolper, M. Yannakakis, ‘Memory efficient algorithms for the verification of temporal properties,’ *Formal Methods in Systems Design*, Vol I, 1992, pp. 275–288.
- [GPVW95] R. Gerth, D. Peled, M. Y. Vardi, P. Wolper, ‘Simple on-the-fly automatic verification of linear temporal logic,’ *Proc. IFIP/WG6.1 Symp. on Protocol Specification, Testing, and Verification*, PSTV95, Warsaw, Poland, June 1995.
- [H88] G. J. Holzmann, ‘An improved protocol reachability analysis technique,’ *Software, Practice and Experience*, 18(2):137-161, 1988.
- [GH93] P. Godefroid and G. J. Holzmann, ‘On the verification of temporal properties,’ *Proc. IFIP/WG6.1 Symp. on Protocol Specification, Testing, and Verification*, PSTV93, Liege, Belgium, June 1993.
- [H91] G. J. Holzmann, *Design and validation of computer protocols*. Prentice Hall, Englewood Cliffs, NJ, 1991.
- [H94] G. J. Holzmann, ‘The theory and practice of a formal method: NewCoRe,’ *Proc. 13th IFIP World Computer Congress*, Hamburg, Germany, 1994, pp. 35–44.
- [HP94] G. J. Holzmann and D. Peled, ‘An improvement in formal verification,’ *Proc. 7th FORTE Conference*, Bern, Switzerland, 1994.
- [H95] G. J. Holzmann, ‘An analysis of bitstate hashing,’ *Proc. IFIP/WG6.1 Symp. on Protocol Specification, Testing, and Verification*, PSTV95, Warsaw, Poland, June 1995.
- [T72] R. E. Tarjan, ‘Depth first search and linear graph algorithms,’ *SIAM J. Computing*, 1:2, pp. 146-160, 1972.