

The Model Checker SPIN

by Gerard J. Holzmann

2/25/2002

1

Review of SPIN

- Support design and verification of asynchronous process systems .
- Rendezvous primitives, asynchronous message passing and shared variables .
- The verification language PROMELA (a Process Meta Language).
- It accepts correctness claims specified in the syntax of standard Linear Temporal Logic (LTL) .
- Models that can be specified in PROMELA are always required to be bounded .

2/25/2002

2

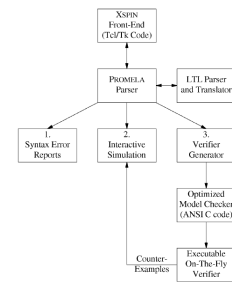
Domain of Application

- Asynchronous software systems .
- The design of the specification language .
- The logic .
- The verification procedure .
- The reduction techniques .
- The state encoding methods.

2/25/2002

3

Structure of SPIN



2/25/2002

4

Foundation of SPIN

- User-defined process templates, i.e. **proctype** definitions.
- At least one process instantiation.
- The templates define the behavior of different types of processes.
- SPIN translates each process template into a finite automaton.
- The global behavior of the concurrent system is obtained by computing an asynchronous interleaving product of automata, one automaton per asynchronous process behavior.

2/25/2002

5

Perform Verification

1. Convert a temporal logic formula into a Büchi automaton .
2. Compute the *synchronous* product of this claim and the automaton representing the global state space. The result is again a Büchi automaton.
3. If the language accepted by this automaton is empty, the original claim is not satisfied .
4. Otherwise, it contains precisely those behaviors that satisfy the original temporal logic formula.

* The verification process *

2/25/2002

6

Linear Temporal Logic

- Use LTL to express correctness.
- Büchi automaton
- Positive claims and negative claims
- SPIN negative correctness claims
- Nested depth-first search algorithm

2/25/2002

7

Nested Depth-First Search

- Cycle detection method (central importance)
- Acceptance cycles can be detected
- Counterexample of a user-defined correctness claim
- Compatible with all modes of verification
 - > exhaustive search
 - > bit-state hashing
 - > partial order reduction techniques

2/25/2002

8

Principle of Nested Depth-First Search

For an accepting cycle to exist in the reachability graph:

1. at least one accepting state must be reachable from the initial system state and
2. it must be reachable from itself.

Therefore, two searches

2/25/2002

9

LTL Grammar

```

f ::= p
    | true
    | false
    | ( f )
    | f binop f
    | unop f

unop ::= []      (always)
        | <->   (eventually)
        | !      (logical negation)

binop ::= U      (strong until)
        | &&     (logical and)
        | ||     (logical or)
        | ->    (implication)
        | <->   (equivalence)
    
```

2/25/2002

10

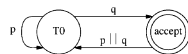
LTLs to Büchi automata

Example1:

$[(p \cup q)]$

```

$ spin -f "[!(p U q)]"
never {
  T0:
    if
    :: (p) -> goto T0
    :: (q) -> goto accept
    fi;
  accept:
    if
    :: ((p) || (q)) -> goto T0
    fi
}
    
```



2/25/2002

11

LTLs to Büchi automata

Example2:

$[(\langle\langle q \rangle\rangle)]$

```

$ spin -f "[!<< q]"
never {
  T0:
    if
    :: (true) -> goto T0
    :: (p) -> goto accept
    fi;
  accept:
    if
    :: (true) -> goto T0
    fi
}
    
```



2/25/2002

12

PartialOrderReduction

- To reduce the number of reachable states that must be explored to complete a verification.
- No noticeable increase of the memory requirements.

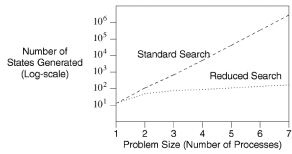


Fig. 5. Effect of partial order reduction.

2/25/2002

13

StateCompression

- Comparable with classical compression techniques
- Small runtime penalty (10 to 20 percent)

GL: *Descriptor for Global Variables*
P1: *Descriptor for Process 1*
C1: *Descriptor for Channel 1*
P2: *Descriptor for Process 2*
SV: **GL:P1:C1:P2** *Global State Descriptor (state vector)*

Fig. 6. State compression algorithm—indexing method.

TABLE 1
EFFECT OF COMPRESSION

Type of Run	No. States	Memory (Mb)	Time (sec.)
Standard	2,435,220	156.59	107.56
Compressed	2,435,220	59.57	123.46

2/25/2002

14

Bit-StateHashing

- high-coverage approximation of the results of an exhaustive run
- a few bits of memory are used to store a reachable state
- bit addresses are computed with two statistically independent hash functions

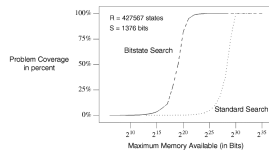


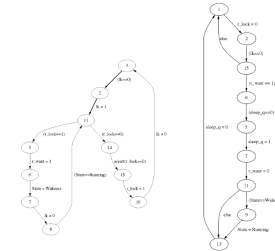
Fig. 7. Measured problem coverage [34], [42] effect of the optional bit-state hashing technique in SPIN.

2/25/2002

15

Example – ProcessScheduling

- A Client – Server model



2/25/2002

16

Example – ProcessScheduling

Client Process	Server Process	Line No.	Local States	Comment
[r_lock = 1]		22	16,15	Consume Resource
[lk = 0]		23	3,15	Release the lock
[lk=0]		10	2,15	Test the lock
[lk = 1]		10	11,15	Set the lock
[r_lock=1]		12	5,15	Resource not available
[r_want = 1]		13	6,15	Set want flag
	[r_want=1]	33	6,6	Server checks flag
	[sleep_q=0]	35	6,5	'Wait lock on sleep_q
	[sleep_q = 1]	35	6,7	Set it
	[r_want = 0]	37	6,11	Resets the flag
	[lck=1]	44	6,13	'No process sleeping'
	[sleep_q = 0]	46	6,1	Release the lock
[State = Wakeme]	[r_lock = 0]	30	6,2	Provide new resource
[lk = 0]		14	7,2	Client goes to sleep!
	[lk=0]	15	8,2	Releases the lock
	[lck=1]	31	8,15	Server checks the lock
	[lck=1]	47	8,1	No flag is set
Non-Progress Cycle:				
	[r_lock = 0]	30	8,2	The server repeats this
	[lk=0]	31	8,15	forever, while the client
	[lck=1]	47	8,1	remains suspended.

Fig. 9. An error scenario generated by SPIN (annotated).

2/25/2002

17

Automatic Verification of Finite -State Concurrent Systems Using Temporal Logic Specifications

E.M. CLARKE
 Carnegie Mellon University
 E.A. EMERSON
 University of Texas, Austin
 and
 A.P. SISTLA
 GTE Laboratories, Inc.

2/25/2002

18

Introduction

- Manual proof construction is unnecessary
- A model-theoretic approach to perform automatic verification
- Wide applicability
- The specification language -- *computation tree logic* (CTL)

2/25/2002

19

CTL Syntax

- (1) Every atomic proposition $p \in AP$ is a CTL formula.
 - (2) If f_1 and f_2 are CTL formulas, then so are $\neg f_1$, $f_1 \wedge f_2$, $AX f_2$, $EX f_1$, $A[f_1 U f_2]$, and $E[f_1 U f_2]$.
- AP is the underlying set of atomic propositions.
 - X is the *nexttime* operator.

2/25/2002

20

Semantics of CTL

We define the semantics of CTL formulas with respect to a labeled state-transition graph. Formally, a CTL structure is a triple $M = (S, R, P)$ where

- (1) S is a finite set of states.
- (2) R is a binary relation on S ($R \subseteq S \times S$) which gives the possible transitions between states and must be total; that is, $\forall x \in S \exists y \in S (x, y) \in R$.
- (3) $P: S \rightarrow 2^{AP}$ assigns to each state the set of atomic propositions true in that state.

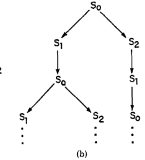
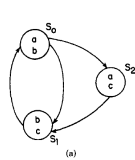


Fig. 1. (a) A structure. (b) The corresponding tree for start state s_0 .

2/25/2002

21

Semantics of Example Formulas

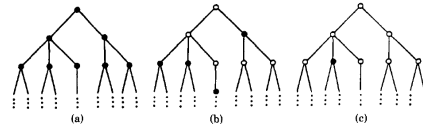


Fig. 2. (a) AGp : p is invariant. (b) AFp : p is inevitable. (c) Efp : p potentially holds. $\bullet = p$, $\circ = \neg p$.

2/25/2002

22

Example of Global State Transition Graph

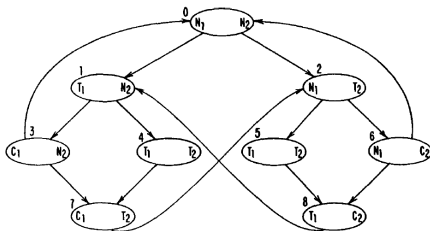


Fig. 3. Global state transition graph for the two-process mutual exclusion problem.

2/25/2002

23

Model Checker

- State labeling algorithm :
 - Stage 1: process all subformulas of length 1
 - Stage 2: process all subformulas of length 2
 - ...
 - Stage i : process all subformulas of length i

```

procedure label_graph(f)
begin
  ...
  [main operator is AU]
  begin
    ST := empty_stack;
    for all s in S do marked(s) := false;
    L: for all s in S do
      if ¬marked(s) then au(f, s, b)
    end
    ...
  end
end
Algorithm for the case  $f = A[f_1 U f_2]$ 
    
```

2/25/2002

24

ModelChecker

- Therecursiveprocedure $au(f,s,b)$ performs the search for formula f starting from states s .
- When au terminates, the boolean result parameter b will be set to true iff $sl=f$.

2/25/2002

25

AfterModelChecking

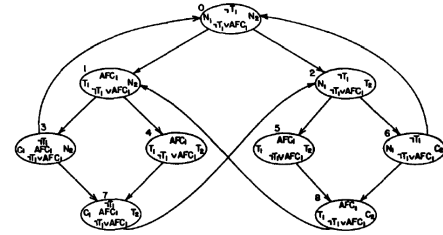


Fig. 4. Global state transition graph after termination of the model checking algorithm.

2/25/2002

26