The Model Checker SPIN

Author: Gerard J. Holzmann

Presented By: Maulik Patel

Outline

- Introduction
- Structure
- Foundation
- Algorithms
- Memory management
- Example/Demo

SPIN-Introduction

- SPIN (Simple Promela INterpreter): Tool for software model checking.
- Written by Gerard Holzmann and others.
- System to be verified is described in PROMELA (Process Meta Language).
- Properties to be verified are expressed as LTL formulae.
- Also works as simulator.

SPIN-Introduction(cont.)

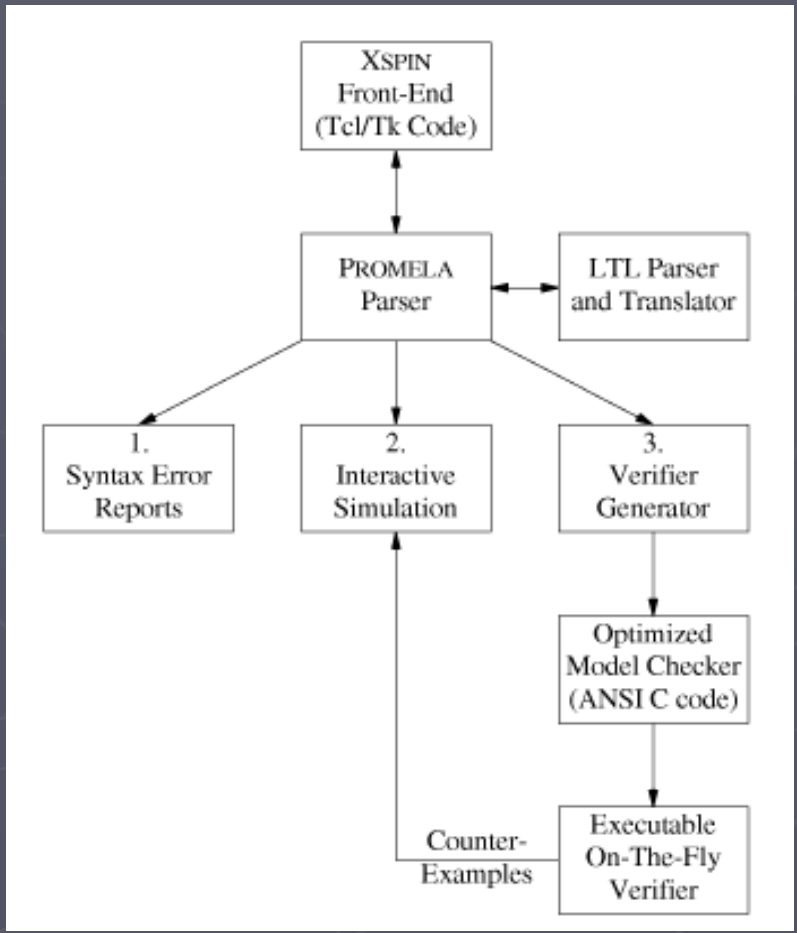
- SPIN does not actually perform model-checking itself, but instead generates C sources for a problem-specific model checker.

- Options to further speed up the process and save memory:
  - Partial order reduction
  - State compression
  - Bit-state hashing

SPIN-Introduction(cont.)

- SPIN aims to provide:
  - Program like notation to specify design choices unambiguously.
  - Concise notation for expressing general correctness requirements.
  - A methodology for establishing the logical consistency of the design choices.

Basic Structure



Foundation

- The predecessors of SPIN were limited to standard safety properties and limited liveness properties.
- Automata theoretic model became the formal basis for temporal logic model checking in SPIN.
- Description of a concurrent system in PROMELA is one or more process templates (*proctype* definitions).
- Each template is translated into a finite automate.

Foundation

- The global system behavior (asynchronous interleaving product) is also an automaton.
- Correctness claim (temporal logic formula) is converted into Buchi automaton.
- Combined behaviour: synchronous product of the system and claim
- Correctness claims are used to specify undesirable behaviours of the system.
- In worst case, reachability graph has the size of the Cartesian product of the components of the system.

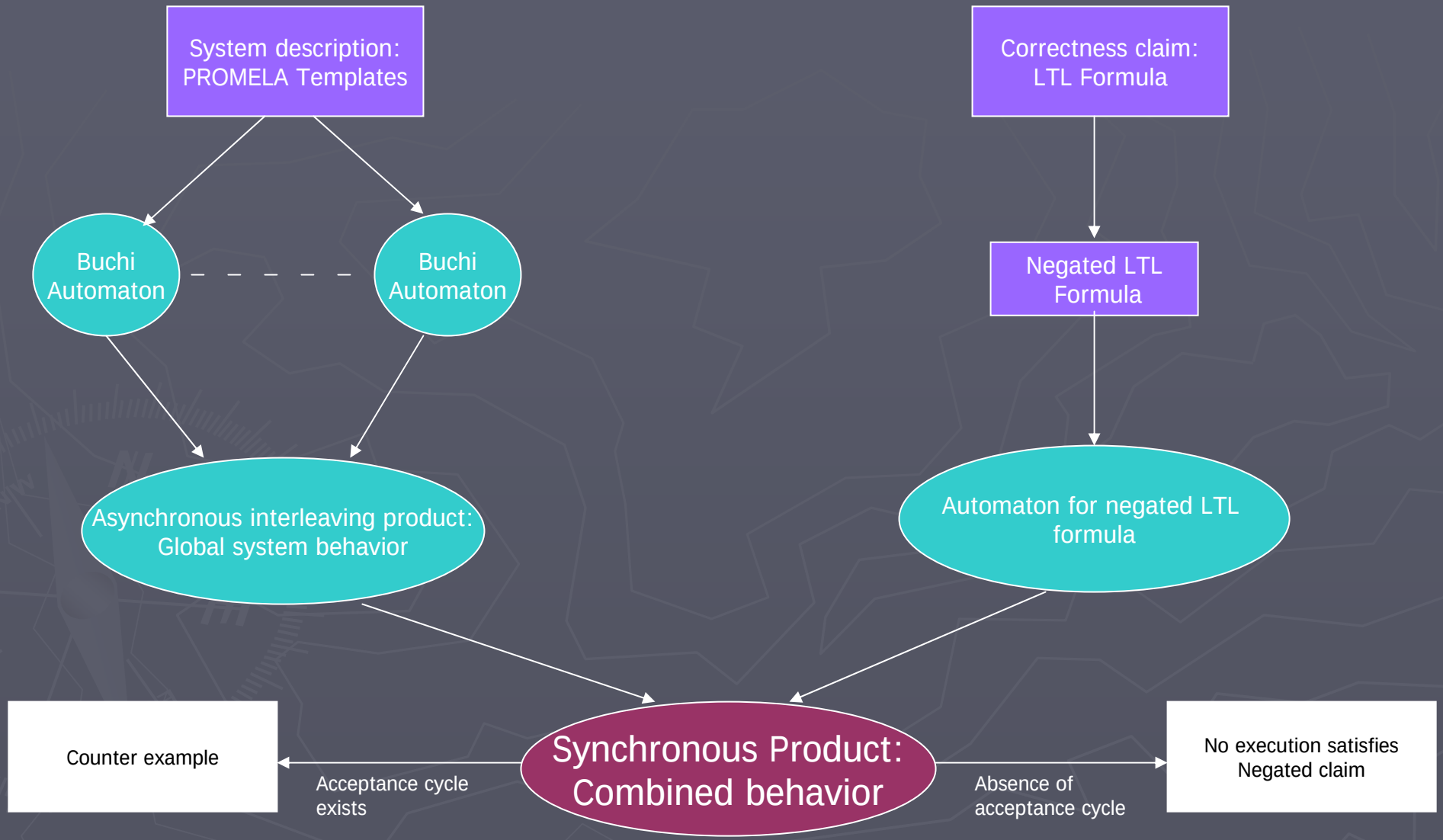
LTL Requirements

- SPIN uses negative correctness claims.
- Positive claim requires to prove that all executions of the system are included in the language of the claim.
  - Size of state space is
    - At most the Cartesian product of the system and claim.
    - At least size of their sum.
- Negative claim requires to prove that the intersection of language of the system and of the claim is empty.
  - Size of state space is
    - In the size of the Cartesian product of the system and claim in worst case.
    - Zero for the best case.

LTL Requirements(Cont.)

- A Buchi automaton accepts a system execution iff there exists at least one acceptance cycle.
- We need to prove the absence of acceptance cycles to prove correctness of the system.
- The nested depth-first search algorithm is used for this purpose.

SPIN Verification



Algorithms

- Nested depth-first search
- From LTL formula to Buchi automaton
- Partial order reduction
- Memory management

Nested Depth-First Search

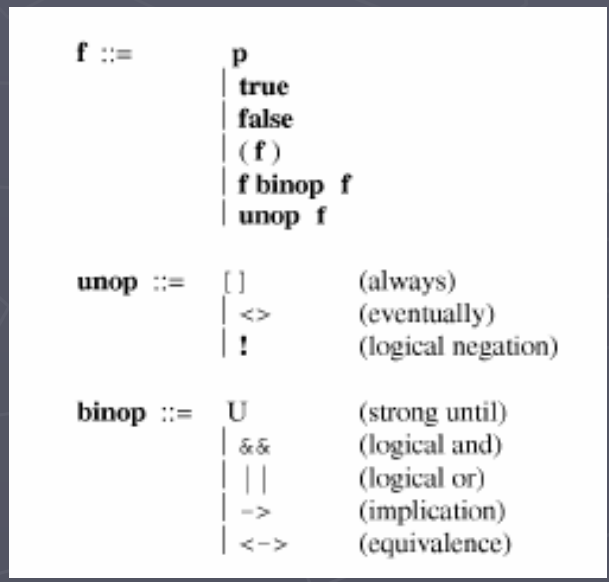
- Tarjan's depth-first search adds two integer numbers to every state reached: the *dfs*-number and the *lowlink*-number.
- This algorithm is not compatible with the bit-state hashing techniques.
- Nested depth-first search: *For an accepting cycle to exist in the reachability graph, at least one accepting state must be both reachable from the initial system state and it much be reachable from itself.*

Nested Depth-First Search(Cont.)

- Two searches are performed.
- If acceptance cycle exists, complete execution sequence is constructed which acts as the counter example of a correctness claim.
- It cannot detect all possible acceptance cycles.
- In SPIN, this algorithm is extended with an optional weak fairness constraint.

Buchi Automata

- LTL formula can contain:
  - Lower case propositional symbol p.
  - Combined with unary or binary, Boolean and/or temporal operators.



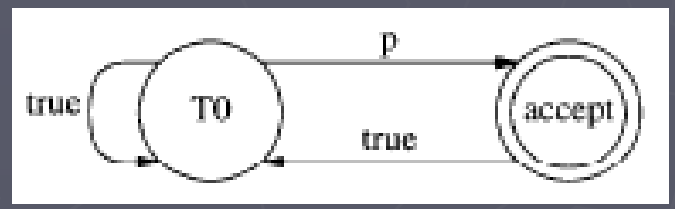
f ::=	**p**	
	true	
	false	
	(**f**)	
	f binop f	
	unop f	
unop ::=	[ ]	(always)
	<>	(eventually)
	!	(logical negation)
binop ::=	U	(strong until)
	&&	(logical and)
	||	(logical or)
	->	(implication)
	<->	(equivalence)

Buchi Automata(Cont.)

- [] (p U q) = G (p U q)


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

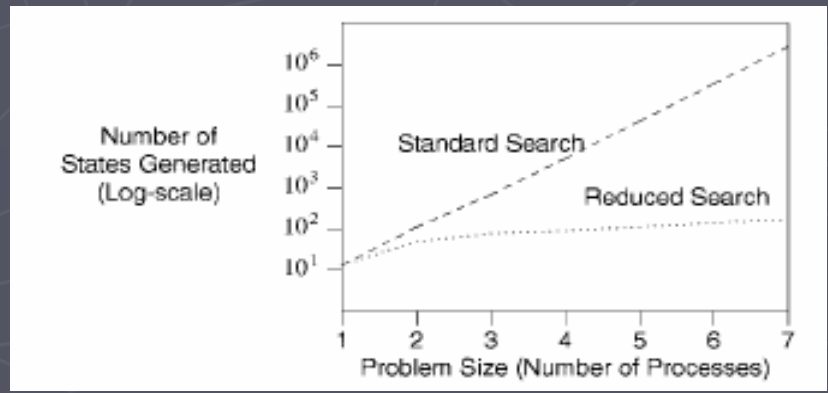
- [] (<> p) = GFP




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

Partial Order Reduction

- To reduce the number of reachable states that must be explored.
- Validity of an LTL formula is insensitive to the order in which concurrent and independent events are interleaved in the depth-first search.
- Implementation is based on a static reduction technique.
- Best case performance:



Partial Order Reduction(Cont.)

- It cannot cause significant increase of the memory requirement, compared to exhaustive searches.
- It is not sensitive to decisions about process or variable orderings.
- Correctness properties were verified using the theorem prover HOL.

Memory Management

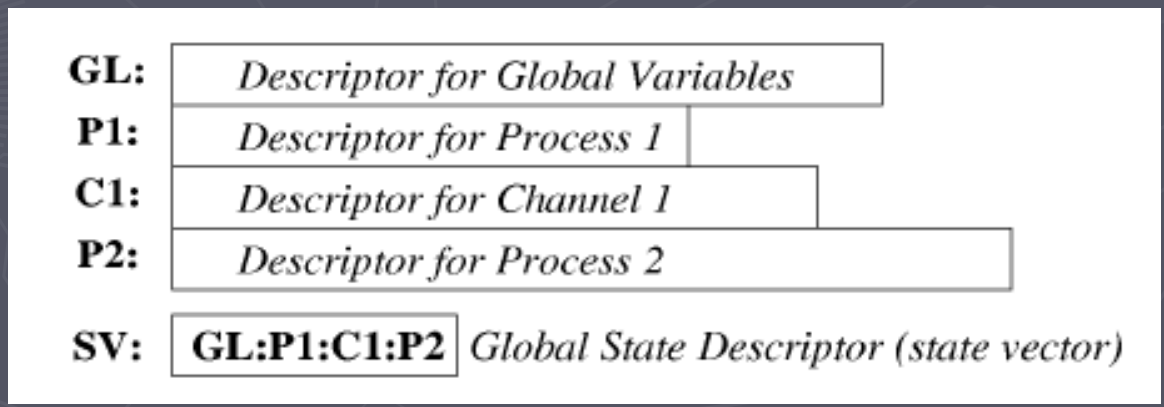
- A main goal of researchers is to devise techniques that can economize the memory requirements without incurring unrealistic increases in runtime requirements.

- Two such techniques are used in SPIN:
  - State compression
  - Bit-state hashing

State Compression

- Static Huffman encoding and run-length encoding techniques can be effective for this purpose.

- A different technique was added to SPIN in late 1995.
  - Every process and channel has relatively small number of local states.
  - Store local states separately from global states and use unique indices to the local state tables inside the global state tables.



State Compression

- In practice, most commonly observed reduction is 60 to 80 percent.

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

Bit-State Hashing

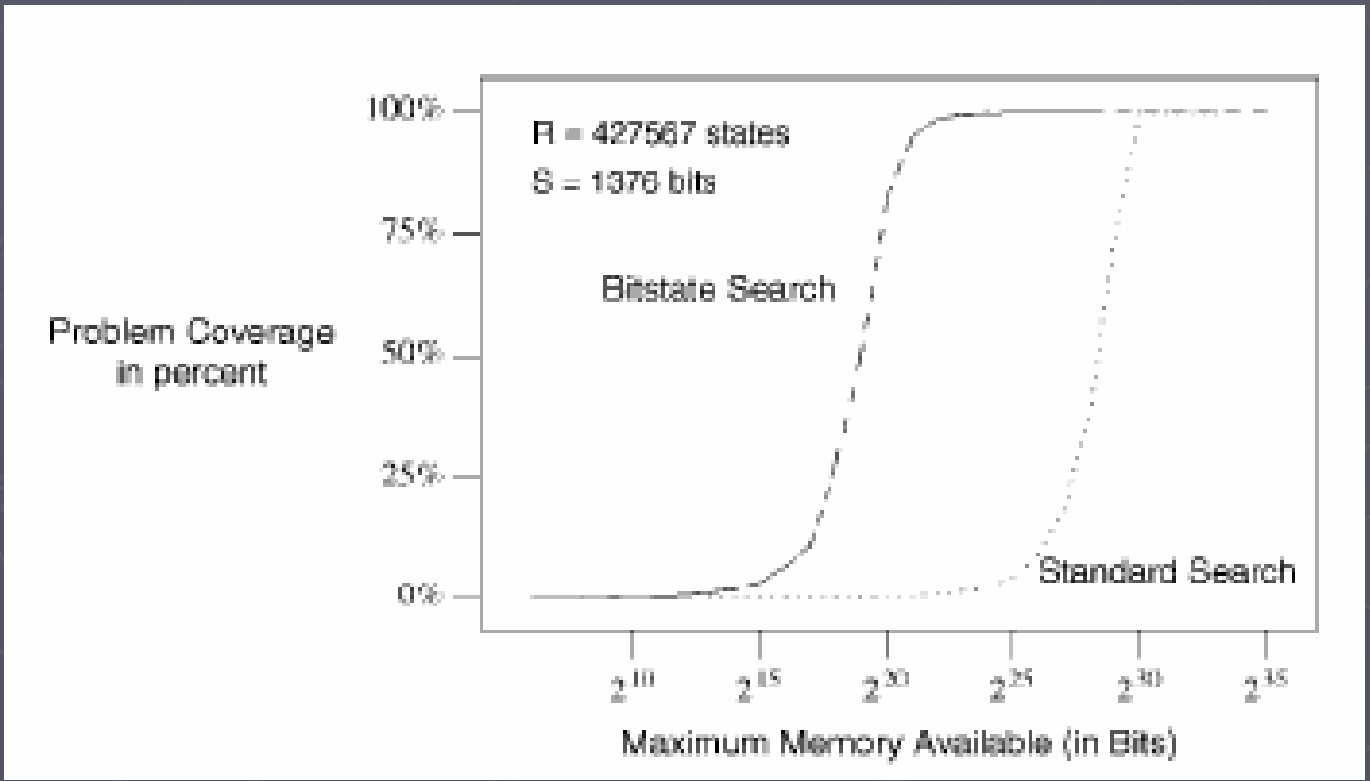
- 2 bits of memory are used to store a reachable state.

- S : width of system state
  M : total machine memory (bytes)
  L : M/S (maximum number of states that can be stored)
  R : total number of reachable states generated by SPIN
  When R>L, *problem coverage* of verification run is M/(R x S).

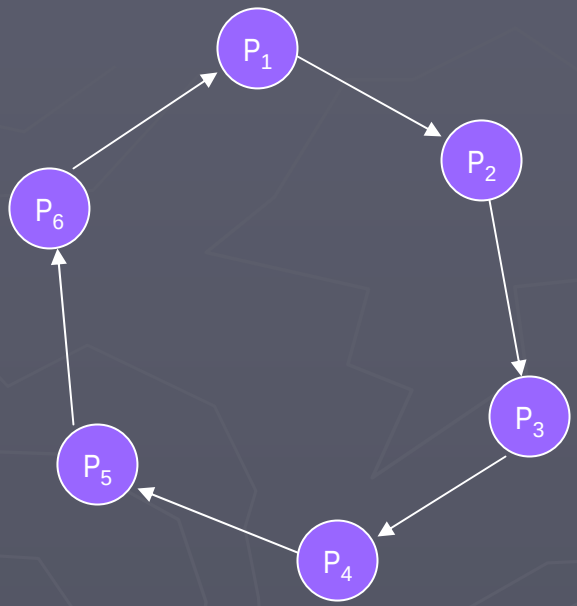
  Ex: M=10^8 bytes, S=10^3 bytes and R=10^6 states
  problem coverage=0.1 (10%)

Bit-State Hashing

- Bit-state hashing technique produces problem coverage close to 1 (100%).



Example – Leader Election



$P_{received} < P_i$: reject

$P_{received} > P_i$: pass

$P_{received} == P_i$:

- Winner has the largest identifier.
- Unidirectional ring.
- Process can decide to join at a later point in the execution.

Example – Leader Election

- Properties:
  - No more than on leader : [] (nr_leader <= 1)
  - There must be one leader: <>[] (nr_leader == 1)
  - Leader must have largest identifier
- Modify algorithm for processes to defer their decision to participate in the election.
- *A process would decide to participate (only when) first receiving a message concerning the election.*

PROMELA Example

Promela Example

```
byte x,t1,t2;

proctype Thread1()
{ do :: t1 = x;
        t2 = x;
        x = t1 + t2
  od }

proctype Thread2()
{ do :: t1 = x;
        t2 = x;
        x = t1 + t2
  od }

init
{ x = 1;
  run Thread1(); run Thread2();
  assert(x != N) }
```

variable declaration

system process

system process

initial process

start threads

assertion

Copied from: *Lecture SPIN-INTRO: Introduction To SPIN (Matt Dwyer, John Hatcliff)*

SPIN Data Structures

SPIN's Basic Data Structures

- State vector
  - holds the value of all variables as well as program counters (current position of execution) for each process.
- Depth-first stack
  - holds the states (or transitions) encountered down a certain path in the computation tree.
- Seen state set
  - holds the state vectors for all the states that have been checked already (seen) in the depth-first search.

Copied from: *Lecture SPIN-INTRO: Introduction To SPIN (Matt Dwyer, John Hatcliff)*

SPIN Demo

http://www.spinroot.com/spin/Man/GettingStarted.html

Question 1

- Partial order reduction algorithm generates reduced state space with only representatives of classes of execution sequences that are distinguishable for a given correctness property.

What criteria is used to distinguish the execution sequences?

Question 2

- Second property of partial order reduction: *The reduction method is not sensitive to decision about process or variable ordering.*

How does it affect the performance of SPIN?

Question 3

- How does the “weak fairness constraint” in SPIN affects nested depth-first search?