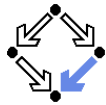


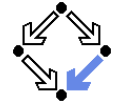
# Verifying Properties of Concurrent Systems

Wolfgang Schreiner  
Wolfgang.Schreiner@risc.uni-linz.ac.at

Research Institute for Symbolic Computation (RISC)  
Johannes Kepler University, Linz, Austria  
<http://www.risc.uni-linz.ac.at>



# The Model Checker Spin

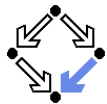


- Spin system:
  - Gerard J. Holzmann et al, Bell Labs, 1980–.
  - Freely available since 1991.
  - Workshop series since 1995 (12th workshop “Spin 2005”).
  - ACM System Software Award in 2001.
- Spin resources:
  - Web site: <http://spinroot.com>.
  - Survey paper: Holzmann “The Model Checker Spin”, 1997.
  - Book: Holzmann “The Spin Model Checker — Primer and Reference Manual”, 2004.

Goal: verification of (concurrent/distributed) software models.



# The Model Checker Spin



On-the-fly LTL model checking of finite state systems.

- System  $S$  modeled by automaton  $S_A$ .
  - Explicit representation of automaton states.
  - There exist various other approaches (discussed later).
- On-the-fly model checking.
  - Reachable states of  $S_A$  are only expanded on demand.
  - Partial order reduction to keep state space manageable.
- LTL model checking.
  - Property  $P$  to be checked described in PLTL.
    - Propositional linear temporal logic.
  - Description converted into property automaton  $P_A$ .
    - Automaton accepts only system runs that do not satisfy the property.

Model checking based on automata theory.

# The Spin System Architecture

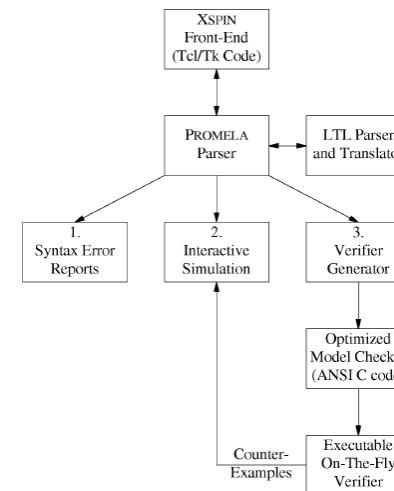
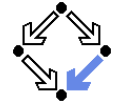
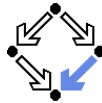


Fig. 1. The structure of SPIN simulation and verification.

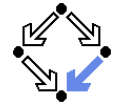
## Features of Spin



- System description in Promela.
  - Promela = Process Meta-Language.
    - Spin = Simple Promela Interpreter.
  - Express coordination and synchronization aspects of a real system.
  - Actual computation can be e.g. handled by embedded C code.
- **Simulation mode.**
  - Investigate individual system behaviors.
  - Inspect system state.
  - Graphical interface XSpin for visualization.
- **Verification mode.**
  - Verify properties shared by all possible system behaviors.
  - Properties specified in PTL and translated to “never claims”.
    - Promela description of automaton for negation of the property.
  - Generated counter examples may be investigated in simulation mode.

Verification and simulation are tightly integrated in Spin.

## The Client/Server System in Promela



```
/* definition of a constant MESSAGE */
mtype = { MESSAGE };

/* two arrays of channels of size 2,
   each channel has a buffer size 1 */
chan request[2] = [1] of { mtype };
chan answer [2] = [1] of { mtype };

/* two global arrays for monitoring
   the states of the clients */
bool inC[2] = false;
bool wait[2] = false;

/* the system of three processes */
init
{
  run client(1);
  run client(2);
  run server();
}

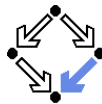
/* the client process type */
proctype client(byte id)
{
  do :: true ->
    request[id-1] ! MESSAGE;

    wait[id-1] = true;
    answer[id-1] ? MESSAGE;
    wait[id-1] = false;

    inC[id-1] = true;
    skip; // the critical region
    inC[id-1] = false;

    request[id-1] ! MESSAGE
  od;
}
```

## The Client/Server System in Promela



```
/* the server process type */
proctype server()
{
  /* three variables of two bit each */
  unsigned given : 2 = 0;
  unsigned waiting : 2 = 0;
  unsigned sender : 2;

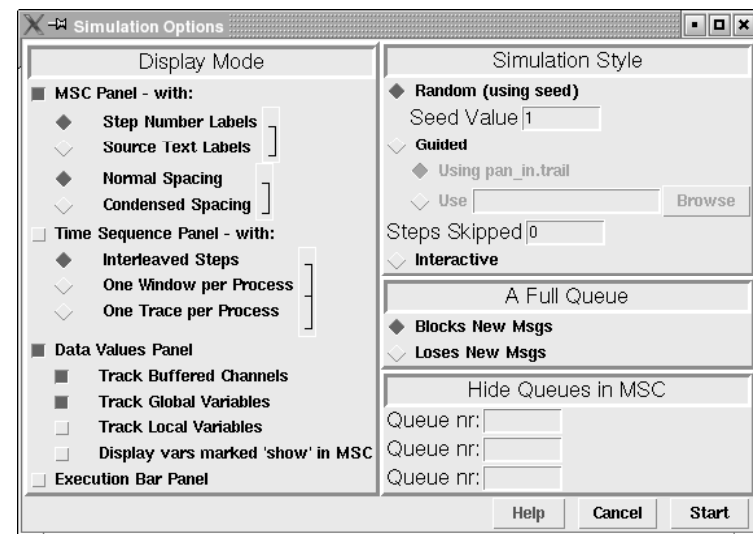
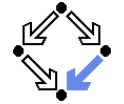
  do :: true ->

    /* receiving the message */
    if
    :: request[0] ? MESSAGE ->
      sender = 1
    :: request[1] ? MESSAGE ->
      sender = 2
    fi;

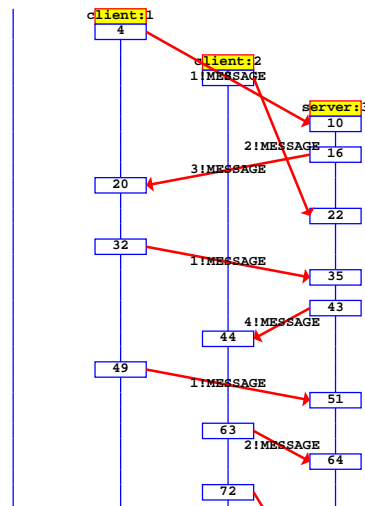
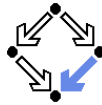
    /* answering the message */
    if
    :: sender == given ->
      if
      :: waiting == 0 ->
        given = 0
      :: else ->
        given = waiting;
        waiting = 0;
        answer[given-1] ! MESSAGE
      fi;
    :: given == 0 ->
      given = sender;
      answer[given-1] ! MESSAGE
    :: else
      waiting = sender
    fi;

    od;
}
```

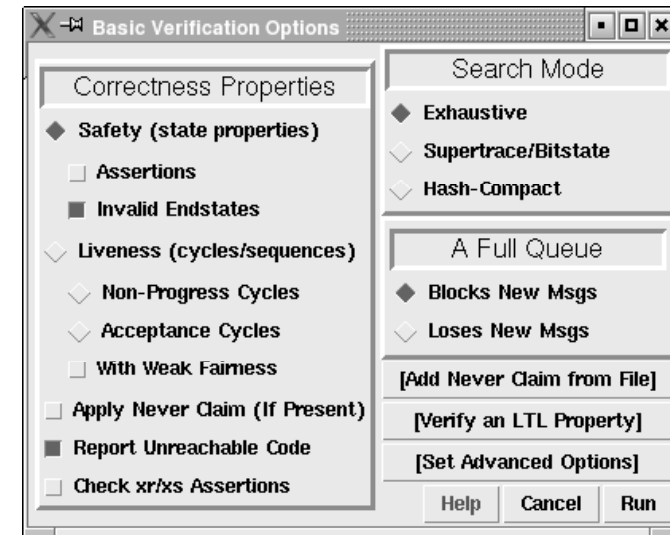
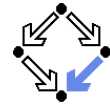
## Spin Simulation Options



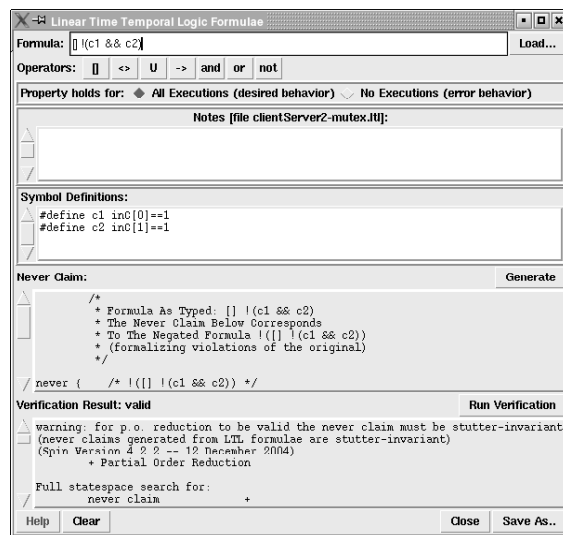
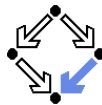
## Simulating the System Execution in Spin



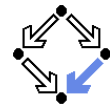
## Spin Verification Options



## Specifying a System Property in Spin



## Spin Verification Output



(Spin Version 4.2.2 -- 12 December 2004)  
+ Partial Order Reduction

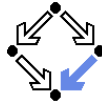
Full statespace search for:  
never claim +  
assertion violations + (if within scope of claim)  
acceptance cycles + (fairness disabled)  
invalid end states - (disabled by never claim)

State-vector 48 byte, depth reached 477, **errors: 0**  
499 states, stored  
395 states, matched  
894 transitions (= stored+matched)  
0 atomic steps  
hash conflicts: 0 (resolved)

Stats on memory usage (in Megabytes):

```
...
0.00user 0.01system 0:00.01elapsed 83%CPU (0avgtext+0avgdata 0maxresident)k
0inputs+0outputs (0major+737minor)pagefaults 0swaps
```

## More Promela Features



Active processes, inline definitions, atomic statements, output.

```
mtype = { P, C, N }
mtype turn = P;

inline request(x, y) { atomic { x == y -> x = N } }
inline release(x, y) { atomic { x = y } }
#define FORMAT "Output: %s\n"

active proctype producer()
{
  do
  :: request(turn, P) -> printf(FORMAT, "P"); release(turn, C);
  od
}

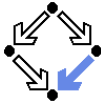
active proctype consumer()
{
  do
  :: request(turn, C) -> printf(FORMAT, "C"); release(turn, P);
  od
}
```

Wolfgang Schreiner

<http://www.risc.uni-linz.ac.at>

13/16

## More Promela Features



Embedded C code.

```
/* declaration is added locally to proctype main */
c_state "float f" "Local main"

active proctype main()
{
  c_code { Pmain->f = 0; }
  do
  :: c_expr { Pmain->f <= 300 };
  c_code { Pmain->f = 1.5 * Pmain->f ; };
  c_code { printf("%4.0f\n", Pmain->f); };
  od;
}
```

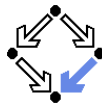
Can embed computational aspects into a Promela model (only works in verification mode where a C program is generated from the model).

Wolfgang Schreiner

<http://www.risc.uni-linz.ac.at>

14/16

## The Implementation of Spin



Translation of the original problem to a problem in automata theory.

- **Original problem:**  $S \models P$ .
  - $S = \langle I, R \rangle$ , PLTL formula  $P$ .
  - Does property  $P$  hold for every run of system  $S$ ?
- Construct **system automaton**  $S_A$  with language  $\mathcal{L}(S_A)$ .
  - A **language** is a set of infinite words.
  - Each such word describes a system run.
  - $\mathcal{L}(S_A)$  describes the set of runs of  $S$ .
- Construct **property automaton**  $P_A$  with language  $\mathcal{L}(P_A)$ .
  - $\mathcal{L}(P_A)$  describes the set of runs satisfying  $P$ .
- **Equivalent Problem:**  $\mathcal{L}(S_A) \subseteq \mathcal{L}(P_A)$ .
  - The language of  $S_A$  must be contained in the language of  $P_A$ .

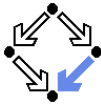
There exists an efficient algorithm to solve this problem.

Wolfgang Schreiner

<http://www.risc.uni-linz.ac.at>

15/16

## The Basic Approach



- **Problem:**  $\mathcal{L}(S_A) \subseteq \mathcal{L}(P_A)$ 
  - Equivalent to:  $\mathcal{L}(S_A) \cap \overline{\mathcal{L}(P_A)} = \emptyset$ .
    - Complement  $\overline{L} := \{w : w \notin L\}$ .
  - Equivalent to:  $\mathcal{L}(S_A) \cap \mathcal{L}(\neg P_A) = \emptyset$ .
    - $\overline{\mathcal{L}(A)} = \mathcal{L}(\neg A)$ .
- **Equivalent Problem:**  $\mathcal{L}(S_A) \cap \mathcal{L}((\neg P)_A) = \emptyset$ .
  - We introduce the **synchronized product automaton**  $A \otimes B$ .
    - A transition of  $A \otimes B$  represents a simultaneous transition of  $A$  and  $B$ .
  - Property:  $\mathcal{L}(A) \cap \mathcal{L}(B) = \mathcal{L}(A \otimes B)$ .
- **Final Problem:**  $\mathcal{L}(S_A \otimes (\neg P)_A) = \emptyset$ .
  - We have to check whether the language of this automaton is empty.
  - We have to look for a word  $w$  accepted by this automaton.
    - If no such  $w$  exists, then  $S \models P$ .
    - If such a  $w = w(r)$  exists, then  $r$  is a **counterexample**, i.e. a run of  $S$  such that  $r \not\models P$ .

Solved by a search in reachability graph of the product automaton.

Wolfgang Schreiner

<http://www.risc.uni-linz.ac.at>

16/16