Course 02158

Verification

Hans Henrik Løvengreen

DTU Compute

Verification

- Given a program P and a property ϕ, show

$$P \models \phi$$

Sequential Programs

- Execution model: $I \rightarrow O$
- Hoare logic for input/output correctness:

$$\{P\}\ S\ \{Q\}$$

Reactive Programs

- Execution model: $s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \ldots$
- Temporal logic [Pnueli 77], e.g. $\Box(P \Rightarrow \Diamond Q)$
- Program given by model:

$$M_P \models \phi$$

Verification Approaches

Paper-and-Pencil Proofs

- OK, for small problems (papers)
- Tedious and error-prone for real systems

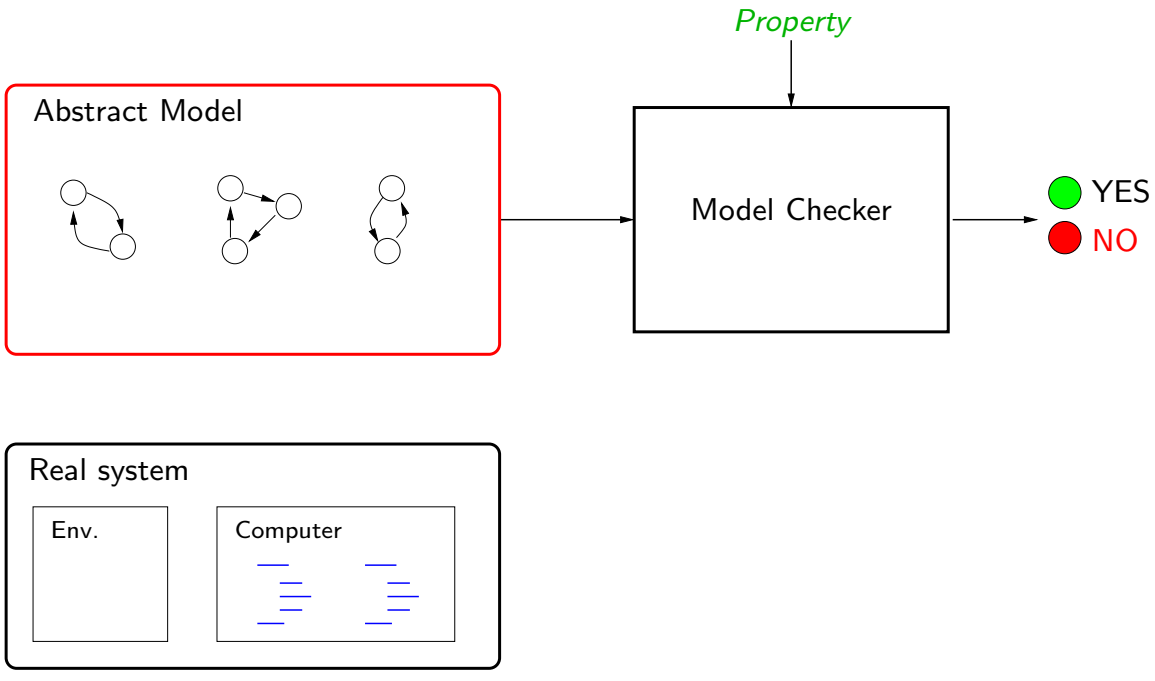
Proof Assistants

- Tools for conducting and checking manual proofs
- May have semi-automatic sub-tools (e.g. decision procedures)
- Hard to learn

Model Checkers

- Automatic proof tools for *finite state* systems
- Less hard to learn
- Suffers from *state explosion*

Model Checking



SPIN

Model checker developed at Bell-Labs
Characteristics

- Based on PAN — Protocol Analyzer (1980)
- First version of SPIN in 1989
- Components:
  - A textual *modelling langauge* (Promela)
  - A *property lanaguage* based on Linear Temporal Logic (ltl)
  - A *verification engine* (pan)
  - A simulator
  - A simple graphical user interface (ispin)
- Open source — implemented in C
- `www.spinroot.com`
- Supplemented by other front ends, e.g. JSpin

Promela

- A textual language based on Dijkstra's Guarded Commands

```
if :: (x <= y) -> skip
   :: (x >  y) -> t = x; x = y; y = t
fi
```

- Allows for both *shared variables* and communication over *channels*
- Only *integer types* (of different sizes) and *arrays*
- Arbitrary big *atomic statements*
- *Macros*, but no proper procedures
- Allows for *nondeterminism*
- Processes can be spawned dynamically

Promela: Peterson's Algorithm

```
bool in1, in2 = 0;
byte turn = 1;

proctype P1() {
  do ::
        in1 = 1;
        turn = 2;

        (in2 == 0 || turn == 1);

        /* critical section */

        in1 = 0;

        /* non-critical section */
        do :: true -> skip :: break od
  od
}
```

```
proctype P2() {
  do ::
        in2 = 1;
        turn = 1;

        (in1 == 0 || turn == 2);

        /* critical section */

        in2 = 0;

        /* non-critical section */
        do :: true -> skip :: break od
  od
}
```

```
init { run P1();  run P2(); }
```

Safety of Peterson's Algorithm

```
bool in1, in2 = 0;
byte turn = 1;
byte incrit = 0;

active proctype P1() {
  do ::
        in1 = 1;
        turn = 2;

        (in2 == 0 || turn == 1);

        /* critical section */
        incrit++;
        assert(incrit == 1);
        incrit--;

        in1 = 0;

        /* non-critical section */
        do :: true -> skip :: break od
  od
}
```

```
active proctype P2() {
  do ::
        in2 = 1;
        turn = 1;

        (in1 == 0 || turn == 2);

        /* critical section */
        incrit++;
        assert(incrit == 1);
        incrit--;

        in2 = 0;

        /* non-critical section */
        do :: true -> skip :: break od
  od
}
```

SPIN Property Language

Safety

- Assertions
- Test-processes
- Global invariants

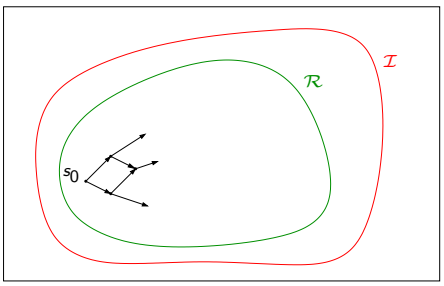
Liveness

- Expressed in Linear Temporal Logic
- Translated into Büchi Automata
- Progress of processes can be assumed (weak fairness)

SPIN Verification Engine

Generates a specialized C-program (pan.c) that:

- Constructs an *state-machine* for each process
- Generates the set of *reachable states*



- Checks properties *on-the-fly*
- Can handle varying length state vectors
- Checks liveness by non-recurrence of states
- Only relevant checks are compiled