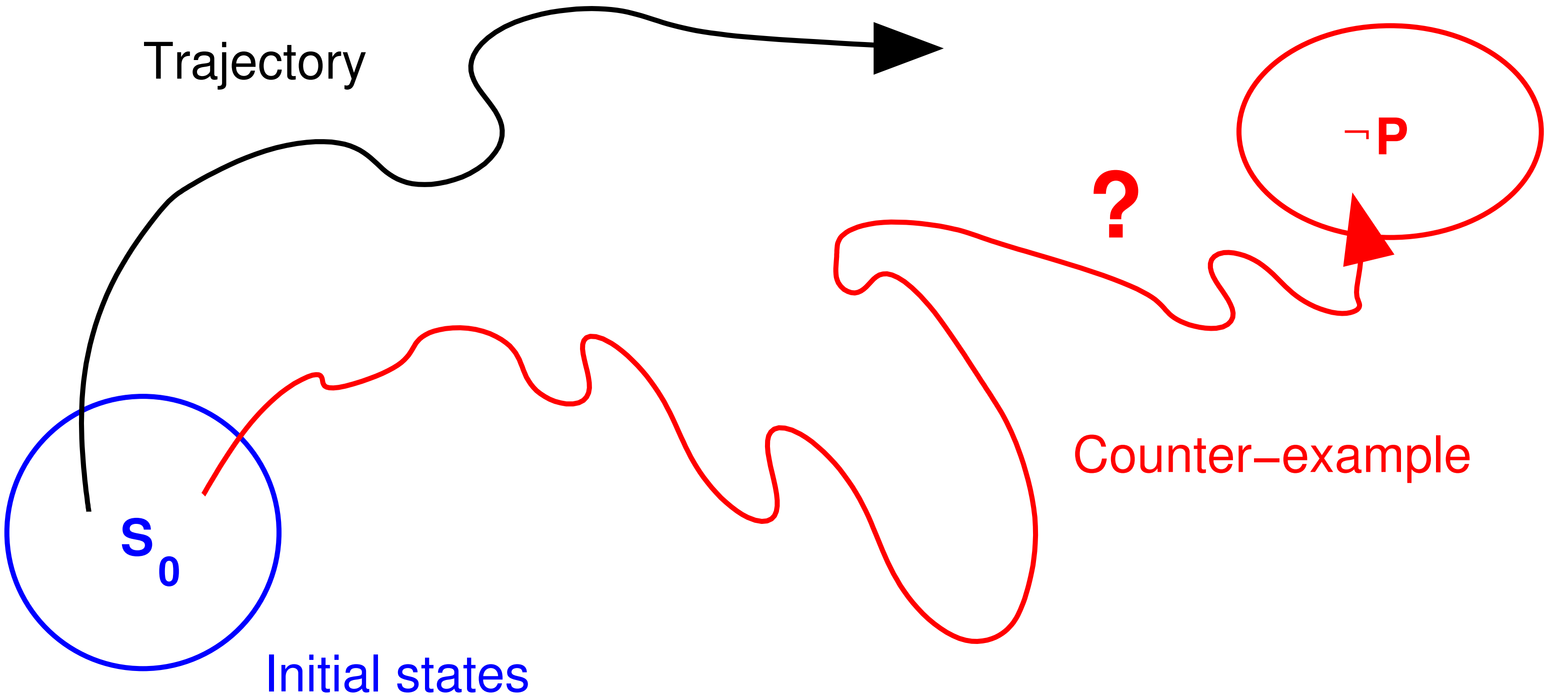
Introduction to Model Checking

Cyrille Artho and Elena Troubitsyna
KTH Royal Institute of Technology, Stockholm, Sweden
School of Electrical Engineering and Computer Science
Theoretical Computer Science
artho@kth.se

Goal: Compute all possible system behaviors

- **Question:** All possible outcomes of a program execution.
  No matter what the inputs/parameters are.
- **Answer:** Enumerate all possibilities! $\rightarrow$ Model Checking.
- Alternative: Reason about all outcomes (theorem proving).

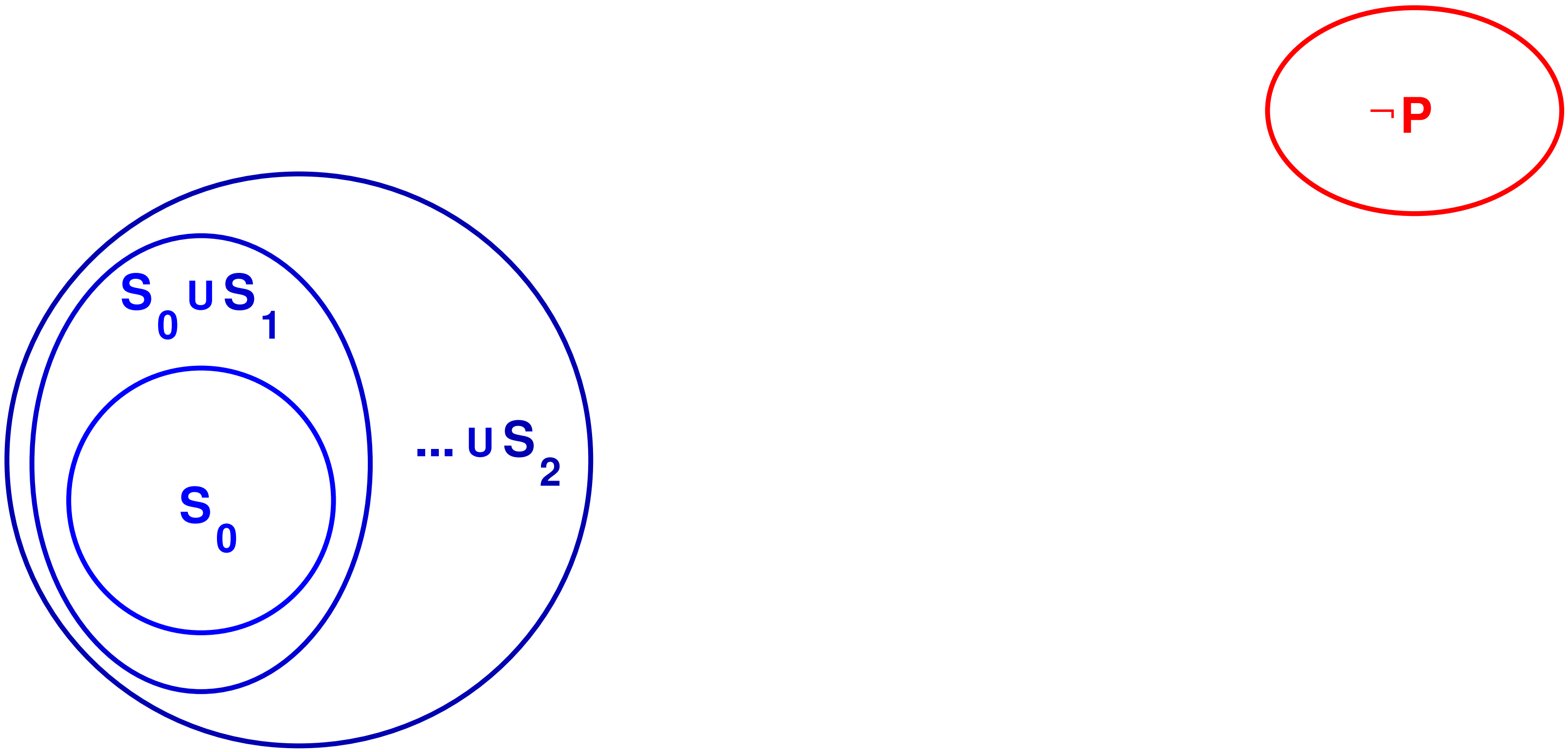
Model Checking = state space search



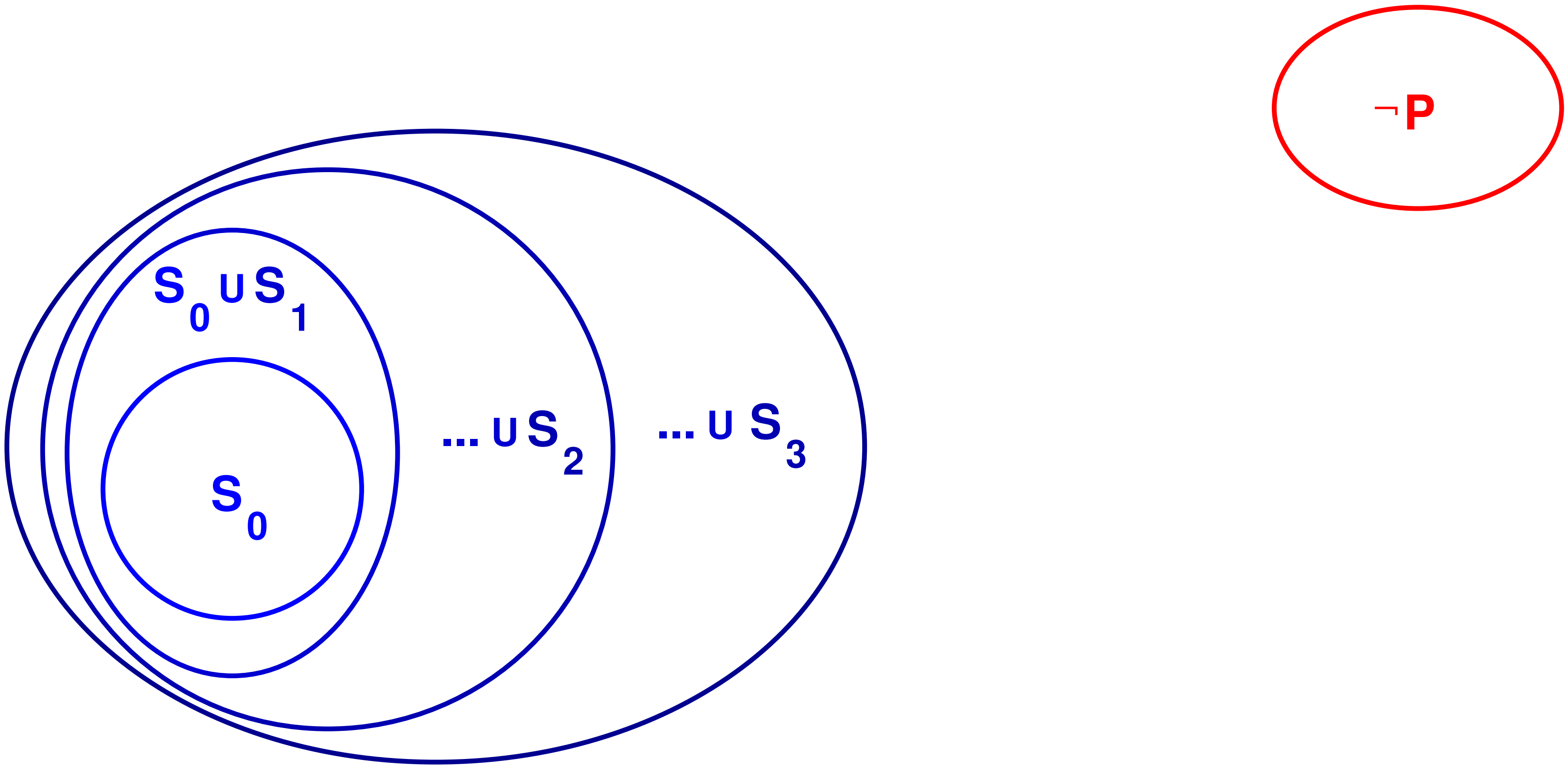
- Traditionally applied to specifications, protocols, algorithms.
- Certain types of software (embedded) can be mapped to such model checkers.

State space search

State space search — 2



State space search — 3



Explicit-state model checking

- Enumerate all states one by one.
- Stop search when property violation found or no new states left.

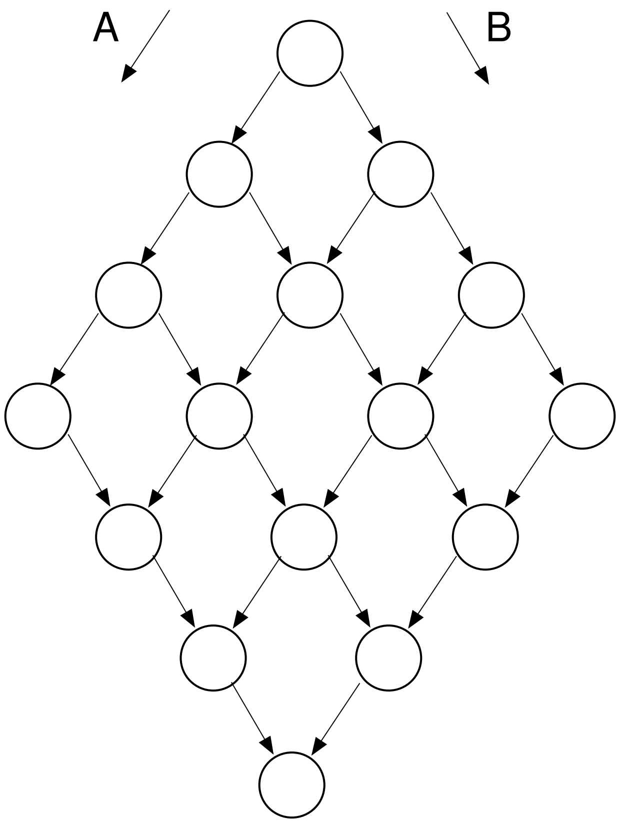
```
def search(model: Model) = {
  currentStates = new Set(model.initialStates)
  visitedStates = new Set()
  var result = None
  // new search queue with init. state
  while (!currentStates.isEmpty && result == None) {
    currentState = currentStates.choose
    // choose removes returned state from currentStates
    result = explore(currentState.successors)
  }
  return result
}
```

State space exploration

```
def explore(states: List[State]) {
  // explore each successor of the current state
  // stop search at target
  for (s <- states) {
    if (s.isAccepting) {
      return Found
    }
    if (!visitedStates.contains(s)) {
      currentStates += s
    }
    visitedStates += s
  }
}
```

- This code shows the key concepts.
- Liveness properties require extra transformations or bookkeeping about loops.

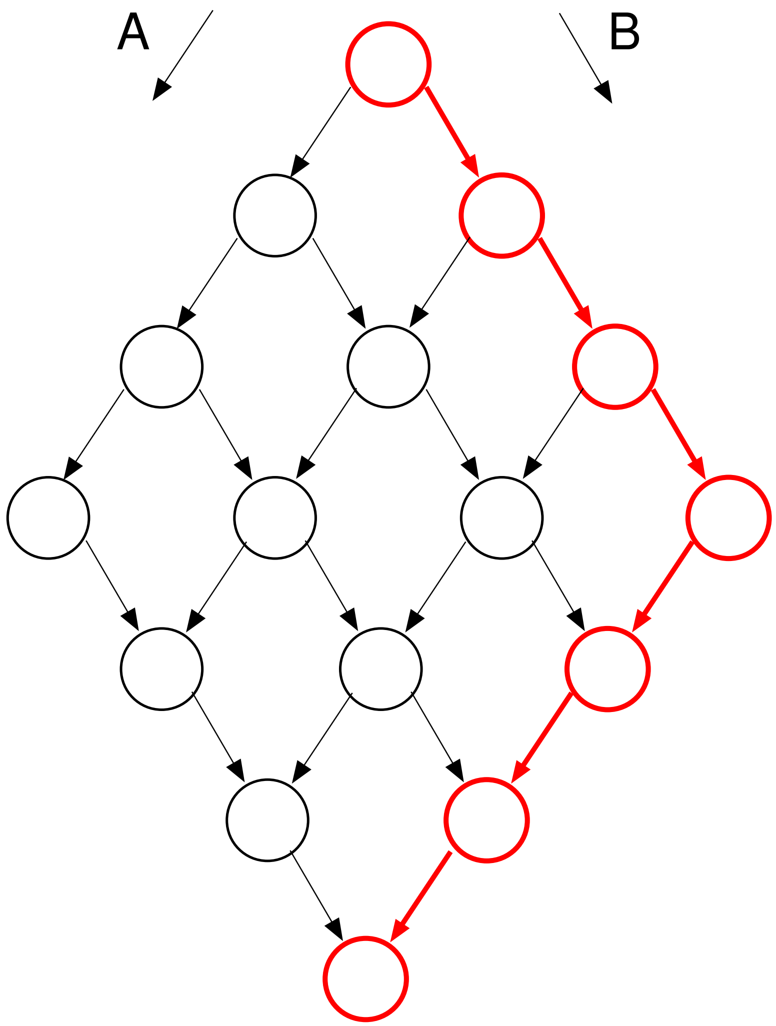
Problem: State space explosion!



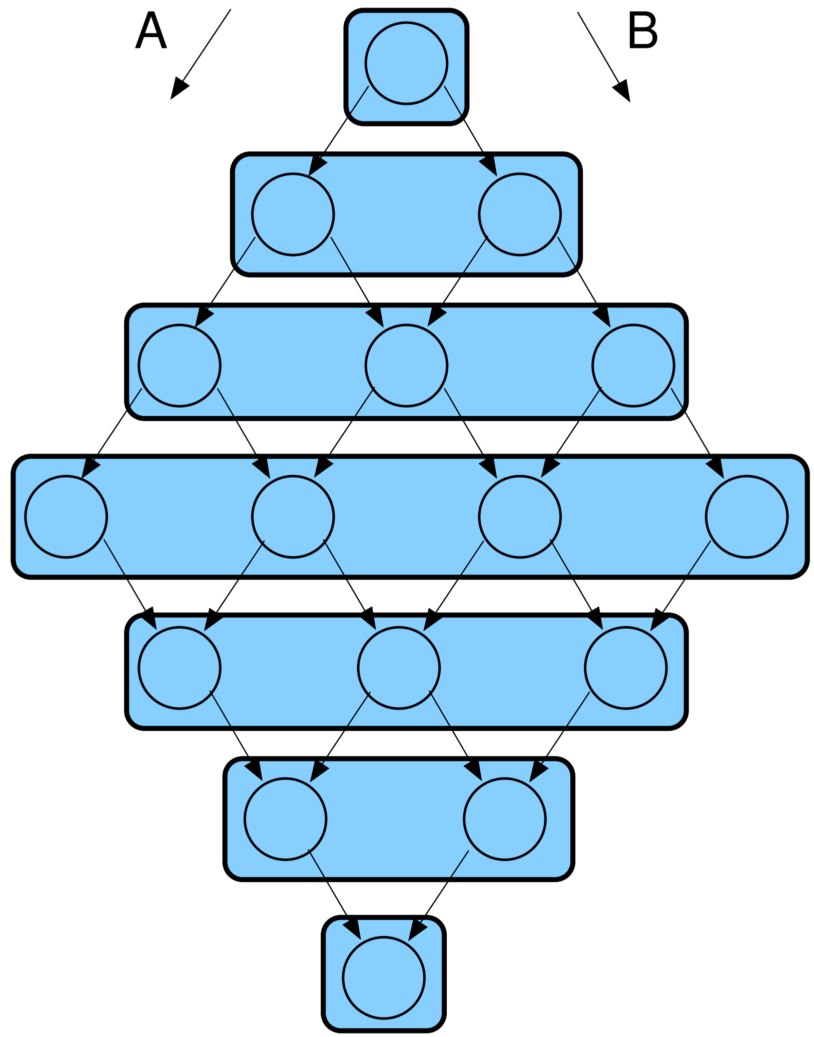
- Process state space exponential in the size of the state.
- Cross-product of processes.

Remedies

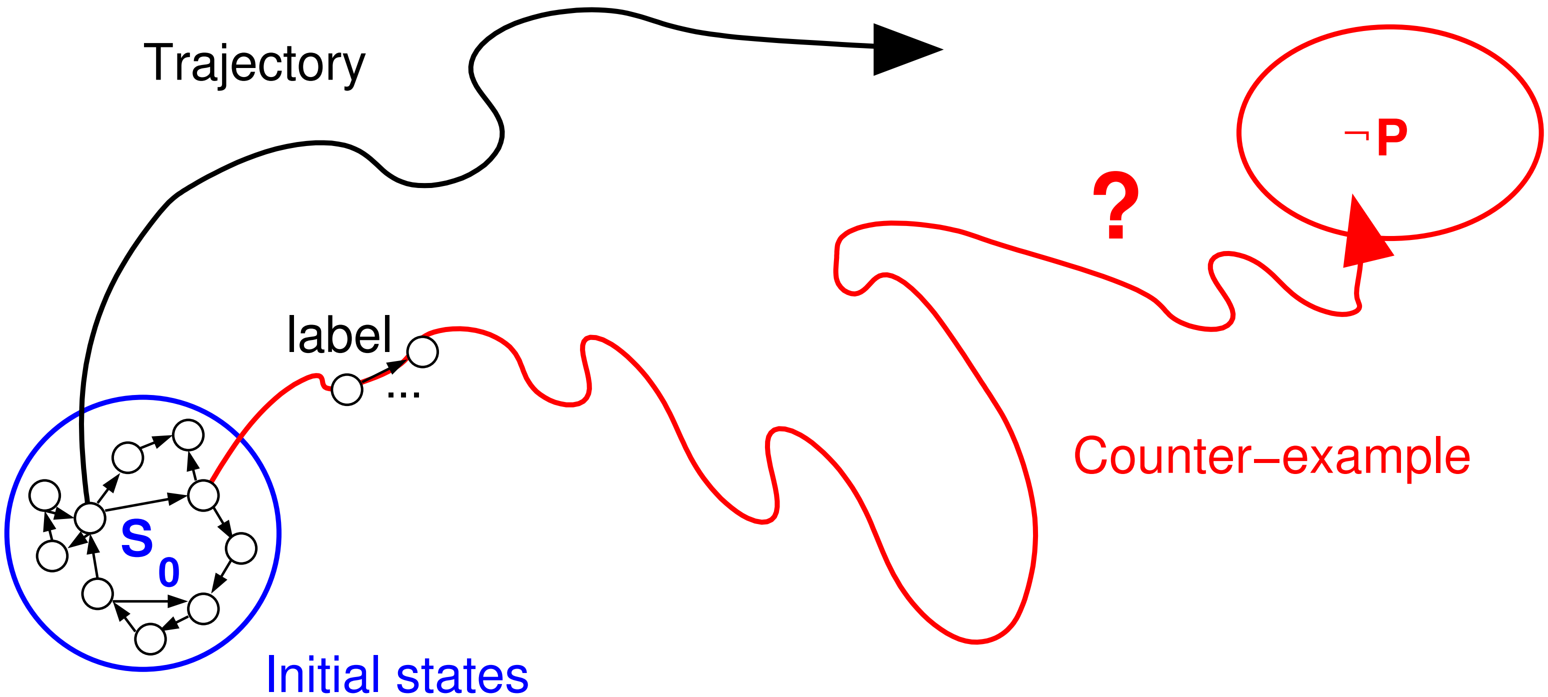
Partial-order reduction



System abstraction



How to define models: Kripke structures



State transition system $M = (S, S_0, R, L)$.

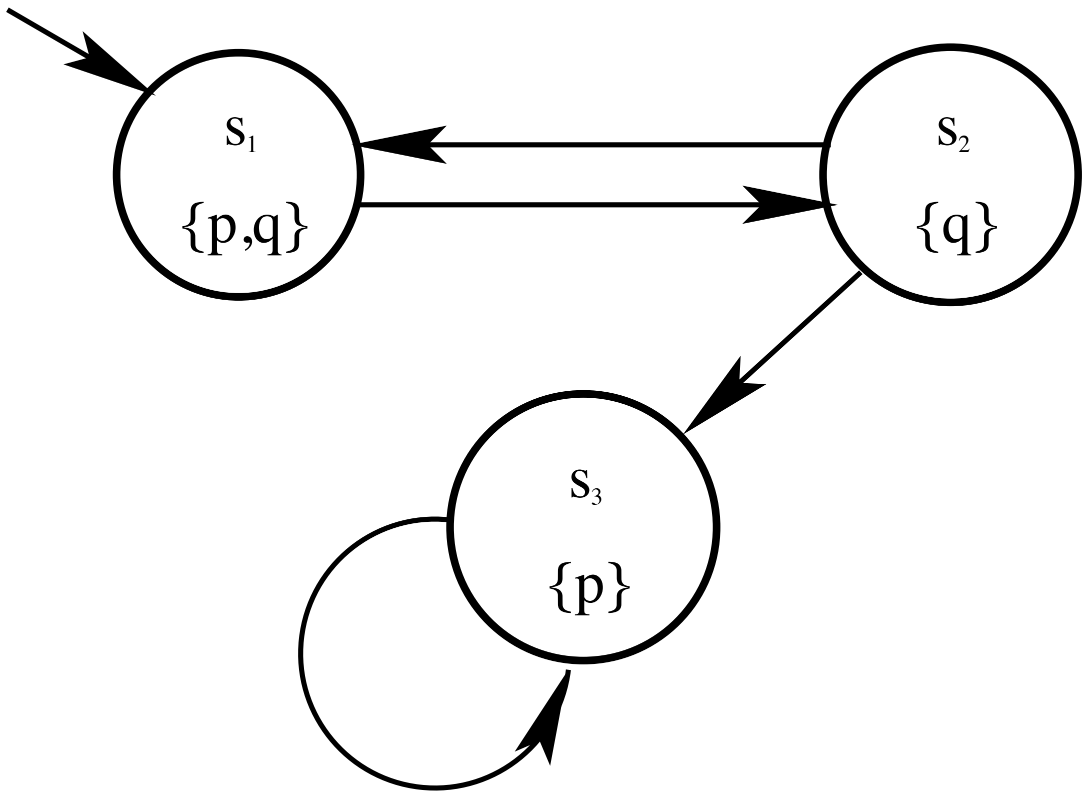
Each transition affects AP : atomic properties.

S: set of states $\quad$ $R : S \times S$: transition relation

S_0: set of initial states $\quad$ L: set of (action) labels: $S \to 2^{AP}$

2^{AP}: power set: each atomic property is true or false at given state.

Example (by Ashutosh Gupta)



$S = \{s_1, s_2, s_3\}$

$S_0 = \{ \quad \}$

$R = \{(s_1, s_2), (s_2, s_1),$

$\}$

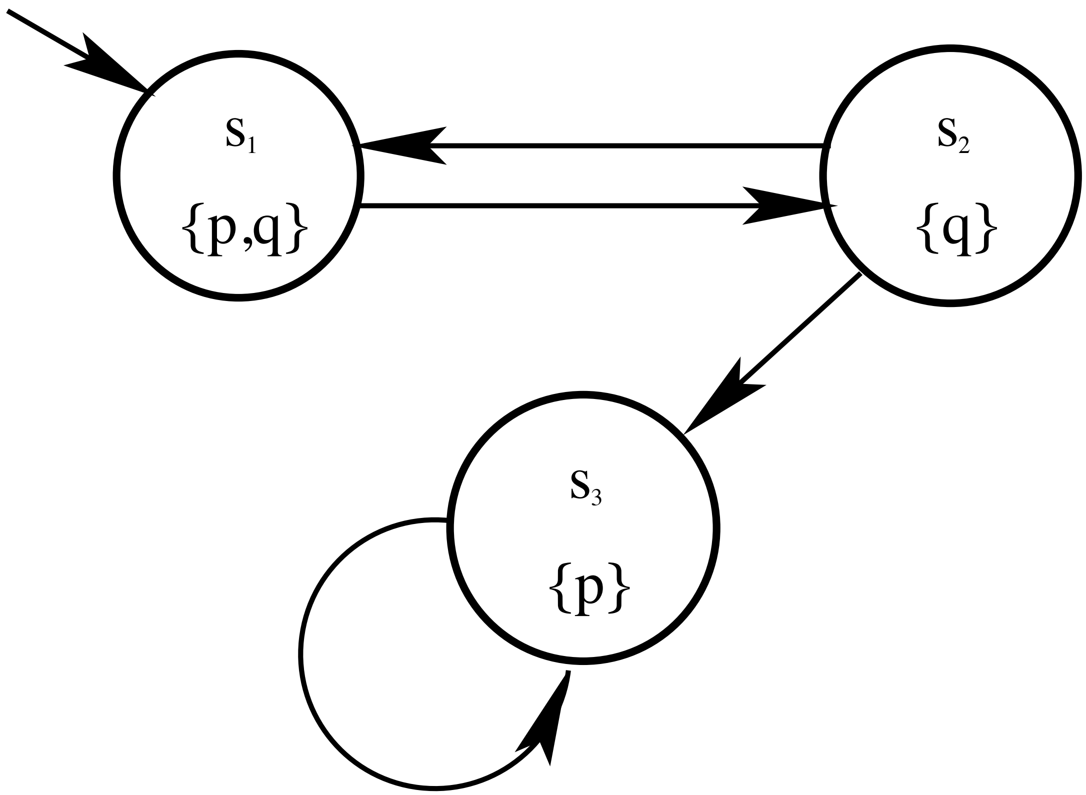
$L = \{(s_1, \{p, q\}),$

$\}$

Fill in the gaps

Give some example paths that this system can generate!

Example (by Ashutosh Gupta)



$S = \{s_1, s_2, s_3\}$

$S_0 = \{s_1\}$

$R = \{(s_1, s_2), (s_2, s_1),$
$\quad (s_2, s_3), (s_3, s_3)\}$

$L = \{(s_1, \{p, q\}),$
$\quad (s_2, \{q\}), (s_3, \{p\})\}$

Example paths:

$\langle s_1, s_2, s_3, s_3 \rangle$

$\langle s_1, s_2, s_1, s_2, s_3 \rangle$

$\langle \overline{s_1, s_2} \rangle$

(overline = inf. path)

Words generated by transition systems

- Example: $\{p,q\},\{q\},\{p,q\},\{q\},\{p\},\ldots$
- Words can be infinitely long.
- We need to reason about words (sequence of atomic properties).

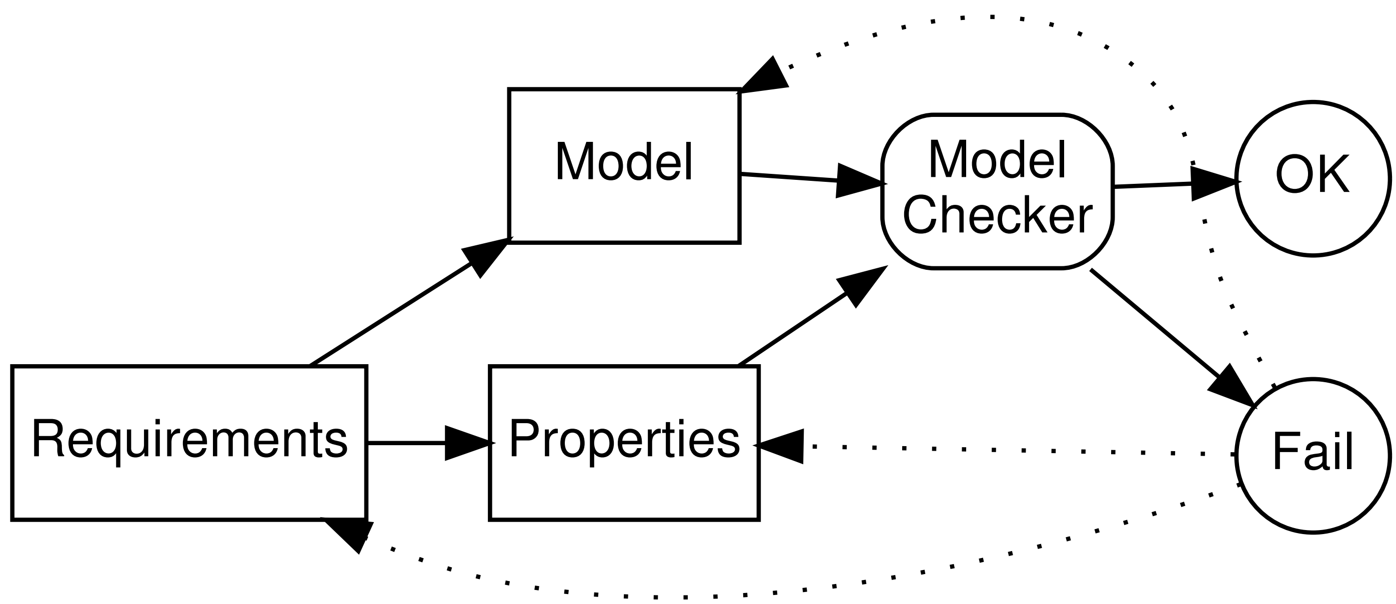
How model is designed

- We typically think of a „state” as certain (state) variables having certain values.
- State transitions have preconditions and actions (more in next lecture).
- Model checker translates this into (simpler but larger) Kripke structure.
- Efficient algorithms to check reachability (more on that soon).
- We could just label „bad” states but that’s not convenient.

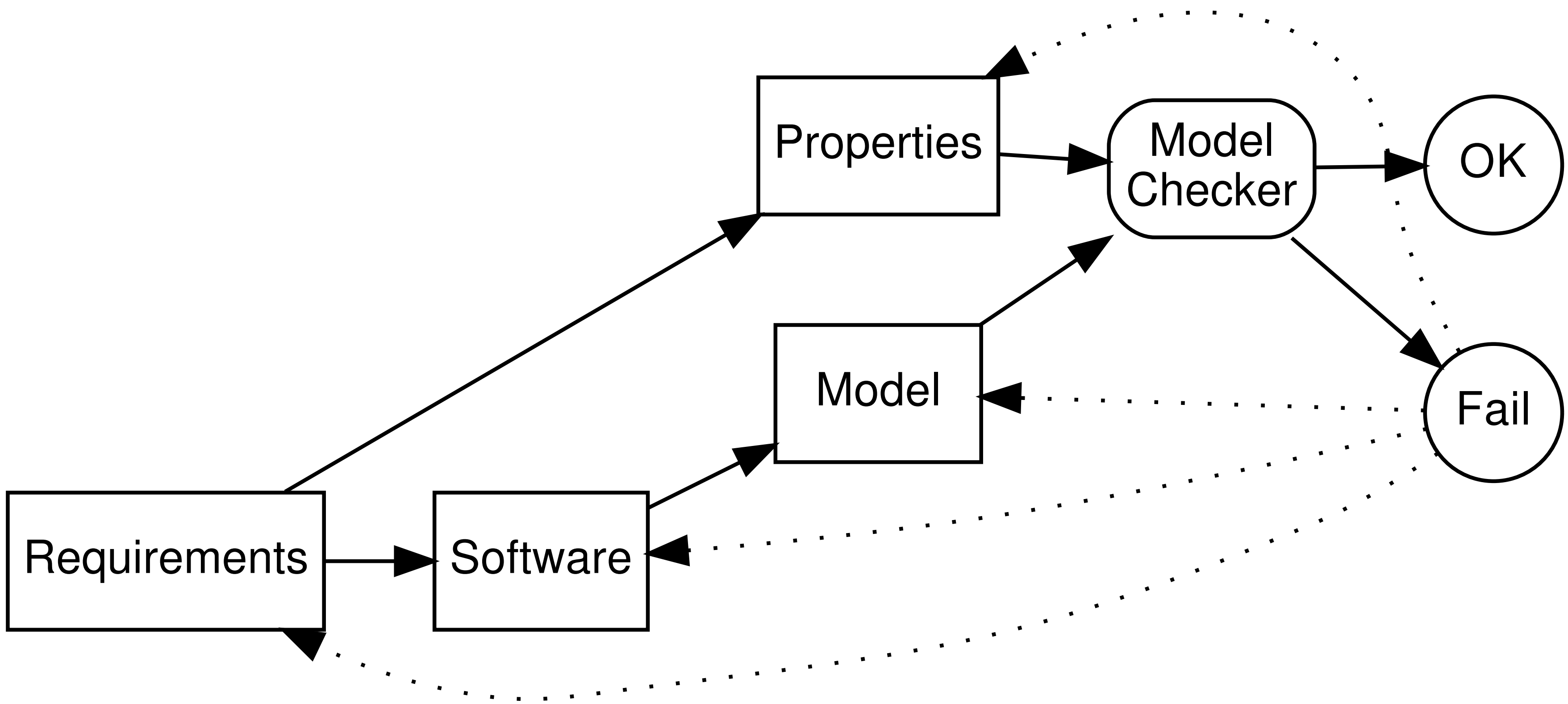
How to describe properties?

Protocol/algorithm verification

- Knowledge on security/safety/reliability concerns.
- Logics to express temporal properties.
- Tools to verify transition systems.

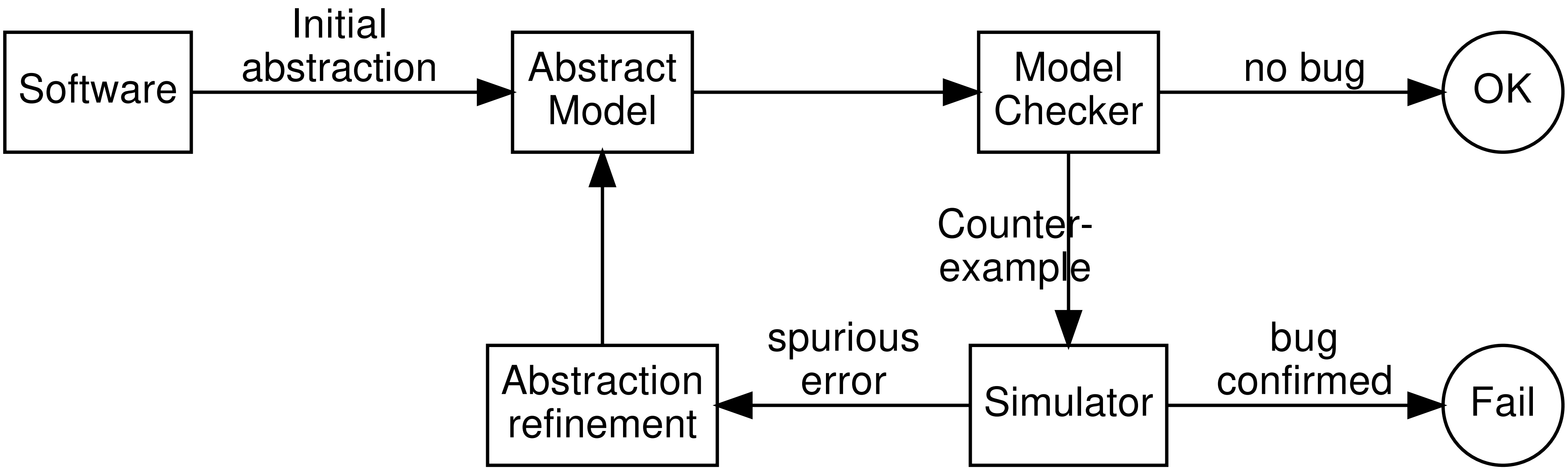


Software verification



Challenging to maintain model by hand.

Counter-Example Guided Abstraction Refinement (CEGAR)



Practical program verification for small systems.

The SLAM Toolkit

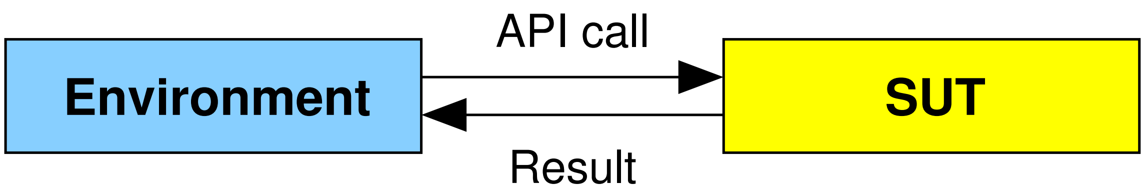
- Goal: Verify Windows NT device drivers by model checking.
- System calls approximated by model.

$\rightarrow$ Model includes state changes in kernel.

- Model used to check thousands of device drivers.
- Continuous effort (tens of person-years).
- For MC a single application, manual abstraction more economical.

Assignment: Read paper on SLAM, answer quiz.

Model-based Testing vs. Model Checking

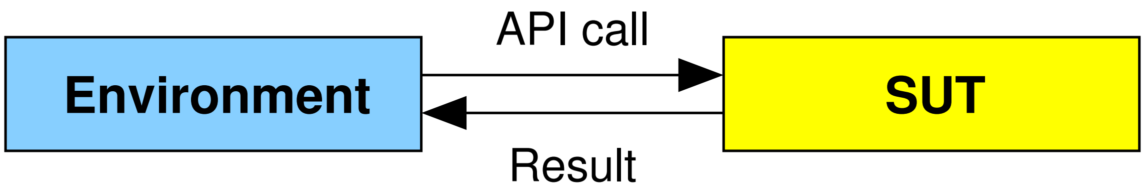


SUT = System under test; API = Application programming interface

Test model **System model**

What **How**

Test Model vs. System Model



SUT = System under test; API = Application programming interface

Test model

- Represents **environment**.
- Models system **behavior**.
- Used to generate **test** cases.
- Model, test one module at a time; SUT itself provides counterpart.
- Model-based testing.

System model

- Represents **system** itself.
- Models system **implementation**.
- Used to **verify** system.
- Need model of most components to analyze system behavior.
- Model checking, theorem proving.

How to describe events and properties

Assignment: Install NuSMV

1. Download NuSMV from
   `http://nusmv.fbk.eu/NuSMV/download/getting-v2.html`

2. Install a binary or download the source
   (***/usr/local*** can be changed to another location):

   (a) **`cd /tmp`**
   (b) **`tar -xzf ~/Downloads/NuSMV-2.6.0.tar.gz`**
   (c) **`cd NuSMV-2.6.0/NuSMV`**
   (d) **`mkdir build`**
   (e) **`cd build`**
   (f) **`cmake .. -DCMAKE_INSTALL_PREFIX=`*****`/usr/local`***
   (g) **`make`**
   (h) **`make install`**
   (i) include ***/usr/local/bin*** in **`PATH`** if needed