Software Validation and Verification
Section II: Model Checking

Topic 2. Model Checkers

Pedro Cabalar

Department of Computer Science and IT
University of Corunna, SPAIN
cabalar@udc.es

22 de febrero de 2023

A simple example of concurrent program

- Example from "A primer on Model Checking" [M. Ben-Ari 2010].
- Two processes `P` and `Q` may increment the value of a memory cell `n` using local registers `regP`, `regQ` respectively.
- They run concurrently: execution is interleaved.

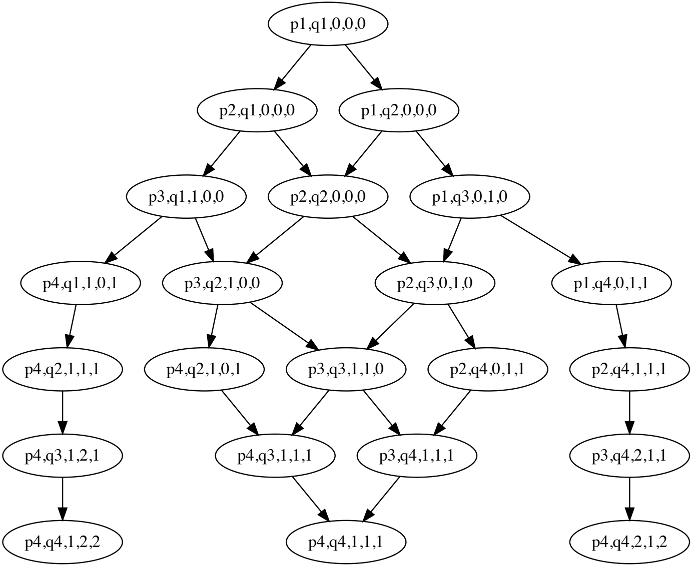
Process P	Process Q
integer regP=0;	integer regQ=0;
p1: load n into regP	q1: load n into regQ
p2: regP++	q2: regQ++
p3: store regP into n	q3: store regQ into n
p4: end	q4: end

A simple example of concurrent program

- A state has `(IPP,IPQ,regP,regQ,n)` where `IPP` and `IPQ` are the respective instruction pointers.
- Initial state is always `(p1,q1,0,0,0)`. Thus, each variable can take values `0,1,2` (at most, two increments are made).
- There exist $4 \times 4 \times 3 \times 3 \times 3 = 432$ states.
- We may build a non-deterministic finite automaton (NDFA) with all the transitions.

Model checking algorithms

- Real problems have a finite number of states (computers deal with a finite number of bits).
- But still, we deal with an unfeasible, astronomical number of cases: possible values in the memory $\times$ possible transitions in a path the NDFA $\times$ number of possible paths in the NDFA.
- Keypoint: not all the states are reachable. In our example, from 432, fixing initial state `(p1,q1,0,0,0)` only 22 are reachable.

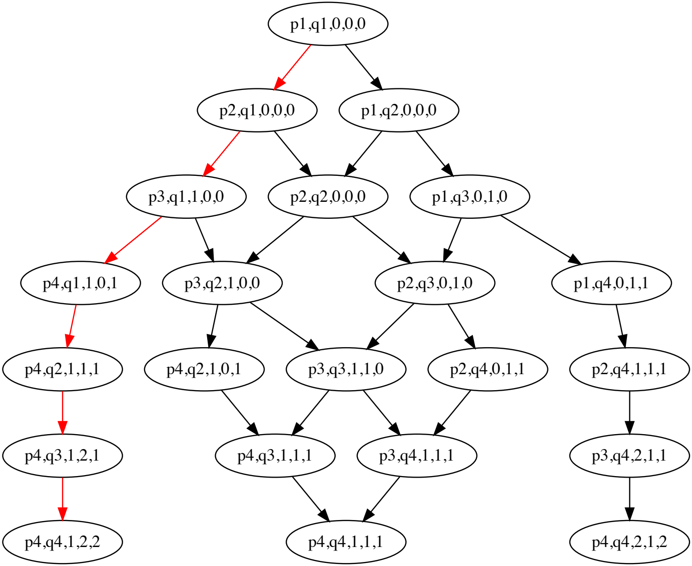


For instance, state `(p2,q2,0,0,1)` is unreachable

A simple example of concurrent program

- Example of computation:
  (p1,q1,0,0,0) → (p2,q1,0,0,0) →
  (p3,q1,1,0,0) → (p4,q1,1,0,1) →
  (p4,q2,1,1,1) → (p4,q3,1,2,1) →
  (p4,q4,1,2,2)

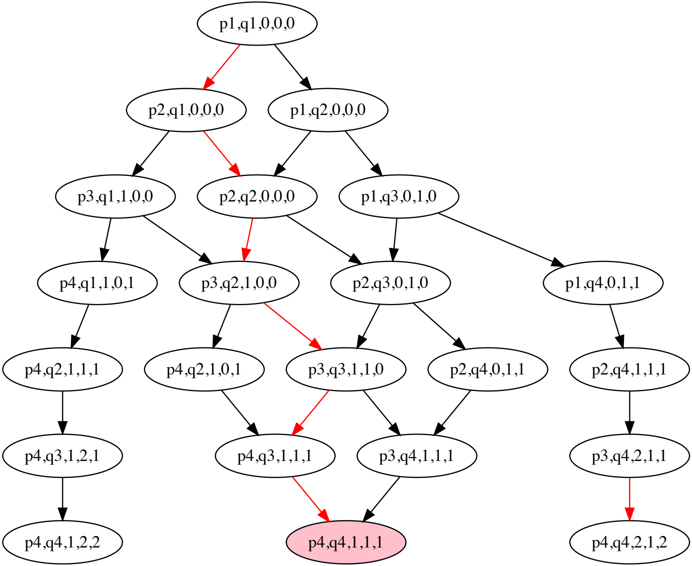
p1: load n into regP	q1: load n into regQ
p2: increment regP	q2: increment regQ
p3: store regP into n	q3: store regQ into n
p4: end	q4: end



It corresponds to execute first P and then Q (no interleaving)

A simple example of concurrent program

- Checking properties = finding paths in the automaton
- Example: We want to check that, after termination, `n=2`. That is, $p4 \wedge q4 \Rightarrow n = 2$.
- We check whether there exists a path (counterexample) from the initial state to a final state satisfying the negation of the property: $p4 \wedge q4 \wedge n \neq 2$.


p1,q1,0,0,0
p2,q1,0,0,0
p1,q2,0,0,0
p3,q1,1,0,0
p2,q2,0,0,0
p1,q3,0,1,0
p4,q1,1,0,1
p3,q2,1,0,0
p2,q3,0,1,0
p1,q4,0,1,1
p4,q2,1,1,1
p4,q2,1,0,1
p3,q3,1,1,0
p2,q4,0,1,1
p2,q4,1,1,1
p4,q3,1,2,1
p4,q3,1,1,1
p3,q4,1,1,1
p3,q4,2,1,1
p4,q4,1,2,2
p4,q4,1,1,1
p4,q4,2,1,2

A simple example of concurrent program

- This counterexample path corresponds to:
  (p1,q1,0,0,0) → (p2,q1,0,0,0) →
  (p2,q2,0,0,0) → (p3,q2,1,0,0) →
  (p3,q3,1,1,0) → (p4,q3,1,1,1) →
  (p4,q4,1,1,1)

```
p1: load n into regP        q1: load n into regQ
p2: increment regP          q2: increment regQ
p3: store regP into n       q3: store regQ into n
p4: end                     q4: end
```

SPIN

- **Promela** (PROcess MEta LAnguage) derived from Dijkstra's Guarded Command Language
- Our example in Promela:

```
byte n=0;
active [2] proctype P() {
  byte reg=0;
  reg=n;
  reg++;
  n=reg;
}
```

- Properties are specified either using local `assert` statments or globally using linear temporal logic (LTL).
- No pointers, functions, parameters, classes, etc. Focused on concurrency and verification.

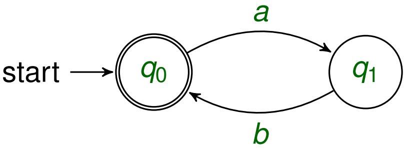
Automata construction

- For each program P we can generate its automaton $\mathcal{A}_P$ capturing the execution paths of P. Memory demand: $\mathcal{A}_P$ can be very large!
- Hash table used for checking existence of newly generated states
- Property α to check can be an LTL formula describing paths
  Generate a second automaton $\mathcal{A}_{\neg\alpha}$ for its negation $\neg\alpha$.
- Take the intersection automaton $\mathcal{A}_P \cap \mathcal{A}_{\neg\alpha}$
  1. If no path, the property is satisfied
  2. If we find a path, it is a counterexample
- Efficient model checking became available thanks to "On-the-fly" techniques [Gerth,Peled,Vardi & Wolper 95]

  = Detecting path for $\neg\alpha$ before complete construction of $\mathcal{A}_P \cap \mathcal{A}_{\neg\alpha}$

Büchi automata

- Processes P and Q had finite traces (they stop in 4 steps)
- Model checkers usually work with reactive systems that run forever (infinite traces)
- LTL allows expressing properties on infinite traces
- Def. ω-language = set of words of infinite length
- A (non-deterministic) Büchi automaton (BA) is like a regular automaton that accepts an ω-language
- A word is accepted if it visits some “final” state infinitely often
- For instance, this Büchi automaton:



accepts infinite alternating sequences $a\ b\ a\ b \dots$

Explicit vs Symbolic

Two possibilities:

- Explicit model checking: each automaton node is an individual state. A hash table indexes all the expanded states. SPIN uses this method.
- Symbolic model checking each node actually represents a set of states. Typically, each set of states is represented with a Binary Decision Diagram (BDD). SMV uses this method.

Model checking techniques

- Partial order reduction: the keypoint is detecting when the ordering of interleavings is irrelevant.

  Example: n processes can execute instructions $I_1, I_2, \ldots I_n$ in any ordering. We have $n!$ combinations, but we can fix an arbitrary one when ordering is irrelevant for the property to check.
- This is common, for instance, when no shared variables are involved

Model checking techniques

- Bounded model checking: when we want to check if property α is violated in k or fewer steps ($k \geq 0$ finite).
- Fixing the path length $i \leq k$ we can translate the problem to SAT (propositional satisfiability). Iterative deepening goes increasing $i = 1, 2, \ldots, k$ until a counterexample is found or k reached.