17. Bounded Model Checking

Computer-Aided Verification

Dave Parker

University of Birmingham

2017/18

Module syllabus

- Modelling sequential and parallel systems
  - labelled transitions systems, parallel composition
- Temporal logic
  - LTL, CTL and CTL*, etc.
- Model checking
  - CTL model checking algorithms
  - automata-theoretic model checking (LTL)
- Verification tools: SPIN
- Advanced verification techniques
  - bounded model checking via propositional satisfiability
  - (symbolic execution), symbolic model checking
- Quantitative verification
  - (real-time systems), probabilistic systems

Overview (next 2 lectures)

- Motivation & overview
  - model checking & scalability
  - bounded model checking via satisfiability
- Propositional logic and satisfiability (SAT)
- Encoding simple (Boolean variable) models with prop. logic
  - model checking invariants via SAT solving
- Software model checking via bounded model checking & SAT
  - loop unwinding
  - single static assignment
  - predicate logic

Scalability of model checking

- Model checking complexity
  - $O(|M| \cdot |\phi|)$ for CTL and $O(|M| \cdot 2^{|\psi|})$ for LTL
- Key issue: "state space explosion problem"
  - the size of the model M for real systems (e.g. software)
  - model size is exponential in the size of the model description

- Many model checking problems reduce to reachability
  - standard graph traversal,
  - e.g. depth-first/breadth-first search
- Efficiency & scalability
  - storage/look-up of visited states crucial
  - SPIN uses hash table of lists of states

Scalability of model checking

- Many solutions for scalability (and efficiency) proposed
  - abstraction (and automated generation of)
  - symmetry reduction
  - partial order reduction
  - symbolic model checking (binary decision diagrams)
  - on-the fly model checking
  - bounded model checking via propositional satisfiability

- Issues to consider:
  - symbolic vs. explicit-state model checking
  - verification vs falsification (bug hunting)
  - soundness, completeness

Bounded model checking

- Key idea
  - unroll model (e.g. control flow graph) for fixed number of steps k
  - construct a propositional formula which is satisfiable
    if and only if there is an error within k steps
  - reduce model checking problem to satisfiability (SAT) problem
  - check (efficiently) using SAT solver

- Bounded model checking (BMC) via SAT
  - originally proposed for hardware model checking
  - subsequently adapted to software verification
  - example software: CBMC (BMC for C and C++ programs)
  - many industrial applications, e.g. hardware, embedded software

Propositional logic and satisfiability

- Propositional logic formulae Φ:
  - Φ ::= true | false | b | $\Phi \wedge \Phi$ | $\Phi \vee \Phi$ | $\neg\Phi$ | $\Phi \rightarrow \Phi$ | $\Phi \leftrightarrow \Phi$ | …
  - where b is a Boolean variable
  - e.g. true, b, $\neg b$, $\neg(b_1 \wedge b_2)$, $b_1 \wedge (b_2 \vee \neg b_3)$
- Satisfiability
  - given propositional formula Φ over variables $b_1,\ldots,b_n$
  - Φ is satisfiable if there exists a valuation of $b_1,\ldots,b_n$ such that $\Phi(b_1,\ldots,b_n)$ evaluates to true
- Example
  - $\Phi(b_1,b_2,b_3) = (b_1 \leftrightarrow \neg b_2) \wedge (b_2 \rightarrow b_3) \wedge (b_3 \vee b_1)$
  - <u>is</u> satisfiable: b_1=true, b_2=false, b_3=true

Propositional satisfiability (SAT)

- Propositional satisfiability problem (SAT)
  - (or Boolean satisfiability problem)
  - "is propositional formula Φ satisfiable?"

- Theoretically important
  - one of the first problems to be proved to be NP-complete
  - (i.e. good example of a "hard" problem)

- Practically important
  - many practical (search) problems can be reduced to SAT
  - many efficient algorithms, tools (SAT solvers) exist
  - huge progress in recent years (big research field in own right)

Example: Z3 solver

- Example: $(b_1 \leftrightarrow \neg b_2) \wedge (b_2 \rightarrow b_3) \wedge (b_3 \vee b_1)$
- Z3 solver: http://rise4fun.com/Z3/

```
(declare-const b1 Bool)
(declare-const b2 Bool)
(declare-const b3 Bool)
(define-fun conjecture () Bool
  (and
    (= b1 (not b2))
    (and
      (=> b2 b3)
      (or b3 b1)
    )
  )
)
(assert conjecture)
(check-sat)
(get-model)
```

Conjunctive normal form

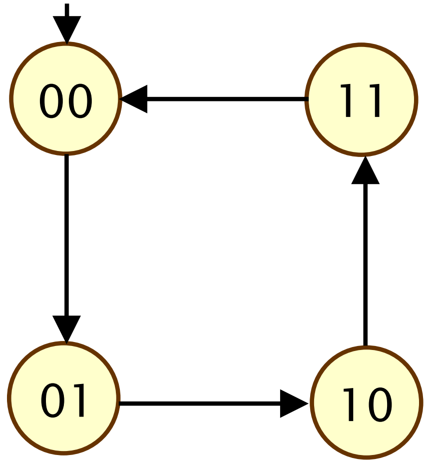
- Conjunctive normal form (CNF)
  - Φ is a conjunction of disjunctions, e.g. $(\neg b_1 \vee b_2) \wedge (b_2 \vee b_3)$
  - i.e. $\Phi = \wedge_{i=1\ldots n} \vee_{j=1..m_n} lit_{ij}$
  - where lit_{ij} is a literal b_k or $\neg b_k$

- We will assume the use of propositional formula in CNF
  - in practice, solvers require inputs to be in CNF

- Can always convert to CNF
  - (de Morgan, double negation, distributive laws)
  - e.g. $\neg((\neg b_1 \rightarrow \neg b_2) \wedge b_3) \implies (\neg b_1 \vee b_2) \wedge (b_2 \vee b_3)$

Encoding a model

- Simple example: 2-bit counter

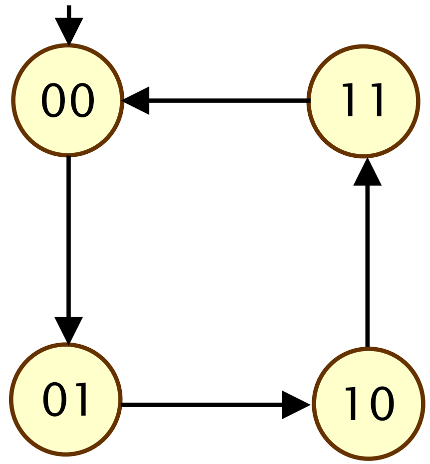


- Encode states in propositional logic:
  - using two Boolean variables l, r
  - e.g. state 10 (representing binary encoding of 2) has $l=1, r=0$
  - (use true=1, false=0)

“symbolic” encoding

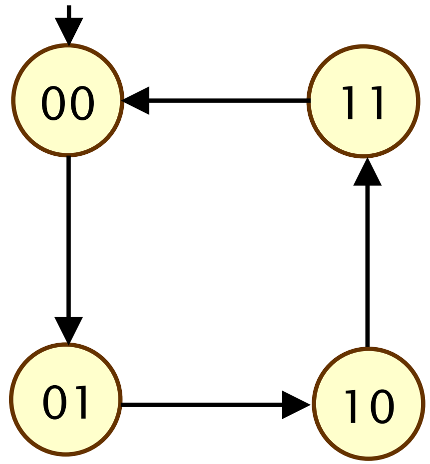
- Encode model in propositional logic:
  - initial state(s): $Init(l_0, r_0) = \neg l_0 \wedge \neg r_0$
  - transition relation: $T(l_i, r_i, l_{i+1}, r_{i+1}) = (l_{i+1} = (l_i \neq r_i)) \wedge (r_{i+1} = \neg r_i)$

Encoding a path



- Using same example…
- Path of length k:
  - $(l_0,r_0), (l_1,r_1), \ldots, (l_k,r_k)$
- Encoding paths of length k (in propositional logic)
  - $Init \wedge T_1 \wedge \ldots \wedge T_k$
  - $\Phi_k = Init(l_0,r_0) \wedge T(l_0,r_0,l_1,r_1) \wedge T(l_1,r_1,l_2,r_2) \wedge \ldots \wedge T(l_{k-1},r_{k-1},l_k,r_k)$
  - e.g. $\Phi_2 = (\neg l_0 \wedge \neg r_0) \wedge (l_1=(l_0 \neq r_0) \wedge r_1=\neg r_0) \wedge (l_2=(l_1 \neq r_1) \wedge r_2=\neg r_1)$
- There is a path of length k…
  - if and only if Φ_k is satisfiable
  - e.g. (for k=2): $l_0=0, r_0=0,\ l_1=0, r_1=1, l_2=1, r_2=0$

Encoding model checking



- Example:
  - invariant: "the counter is always less than 3"
  - i.e. "always P_i", where $P_i = \neg(l_i \wedge r_i)$ ("not 11")
- Encoding in propositional logic
  - property is false if there exists a counterexample
  - $\mathrm{Init} \wedge (T_1 \wedge \ldots \wedge T_k) \wedge (\neg P_0 \vee \neg P_1 \vee \ldots \vee \neg P_k)$
- Example (k=2):
  - $(\neg l_0 \wedge \neg r_0) \wedge (l_1 = (l_0 \neq r_0) \wedge r_1 = \neg r_0) \wedge (l_2 = (l_1 \neq r_1) \wedge r_2 = \neg r_1) \wedge ((l_0 \wedge r_0) \vee (l_1 \wedge r_1) \vee (l_2 \wedge r_2))$
  - not satisfiable ⟵ incomplete!
  - (but is satisfiable for k=3)

In Z3

```
(declare-const l0 Bool)
(declare-const r0 Bool)
(declare-const l1 Bool)
(declare-const r1 Bool)
(declare-const l2 Bool)
(declare-const r2 Bool)
(define-fun init ((l Bool) (r Bool)) Bool
	(and (not l) (not r))
)
(define-fun trans ((li Bool) (ri Bool) (lj Bool) (rj Bool)) Bool
	(and
		(= lj (not (= li ri)))
		(= rj (not ri))
	)
)
(assert (and (init l0 r0) and (trans l0 r0 l1 r1) (trans l1 r1 l2 r2)))
(check-sat)
(get-model)
```

Software model checking

- Simple example program
  - x and y are integer variables

```
x = x + y;
if (x ≠ 1) {
      x := 2;
} else {
      x++;
}
assert (x≤3);
```

→

$(x_1 = x_0 + y_0) \wedge$
$(x_2 = 2) \wedge$
$(x_3 = x_1 + 1) \wedge$
$(x_4 = (x_1 \neq 1) ? x_2 : x_3) \wedge$
$\neg(x_4 \leq 3)$

- Notice
  - simple imperative language (close to Java, C++)
  - variables can be uninitialised
  - properties specified as assertions (invariants?)