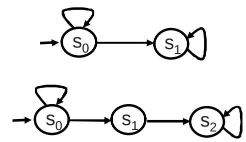
Algorithmic Verification

Comp4151

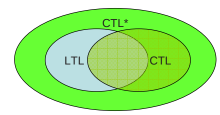
Lecture 9-A

Overview

Modelling
- Finite automata
- Büchi automata
- Kripke structures



Specification
- Linear Time Logic
- Computation Tree Logic
- CTL*



Overview

Model checking

Explicit state model checking
- Bottom-up recursive labelling algorithm for CTL
- LTL tableau or automaton-based algorithms
- Smart enumeration to combat state explosion problem (e.g. partial order reduction)

Symbolic model checking
- Reformulate model checking problem in terms of sets
- Represent as BDDs
- Efficient algorithm through efficient BDD operations

Overview

Today: SAT-based model checking

Basic Idea

Use algorithms that solve (in practice) difficult problems efficiently.

Transform model checking problem to an problem instance for that solver.

SAT-solver are such efficient solvers.

Satisfiability

Problem

- Given a propositional formula *f* over variables X.
- Does there exist an assignment X → {0,1} such that *f* becomes true?

- *Does there exist a satisfying assignment?*
- *Does there exist a model for f ?*

This presentation partly based presentation from Toby Walsh

Satisfiability

Example: A diplomatic problem

- As chief of staff, you are to sent out invitations to the embassy ball.
  - The ambassador instructs you to invite Peru or exclude Qatar.
  - The vice-ambassador wants you to invite Qatar or Romania or both.
  - A recent diplomatic incidents means that you cannot invite both Romania and Peru

Who do you invite?

Satisfiability

Example: A diplomatic problem

- Given the following constraint over P, Q and R

$$f=(P \vee \neg Q) \wedge (Q \vee R) \wedge \neg(R \wedge P)$$

- Does there exist an assignment to P, Q, R such that f= true ?

- Two satisfying assignments
  - $P \mapsto 1, Q \mapsto 1, R \mapsto 0$
  - $P \mapsto 0, Q \mapsto 0, R \mapsto 1$

Satisfiability

Example: A diplomatic problem

- Given the following constraint over P, Q and R

$$f=(P \vee \neg Q) \wedge (Q \vee R) \wedge \neg(R \wedge P)$$

- Does there exist an assignment to P, Q, R such that f= true ?

- Two satisfying assignments
  - $P \mapsto 1, Q \mapsto 1, R \mapsto 0$
  - $P \mapsto 0, Q \mapsto 0, R \mapsto 1$

P, Q, ¬ R
¬ P, ¬ Q, R

Satisfiability

- Satisfiability of a propositional formula was the first problem shown to be NP-complete
- Focus typically on formulas in clausal normal form (CNF) — *a.k.a conjunctive normal form*
  - Formula is a conjunction of clauses
    $C1 \wedge C2 \wedge \ldots$
  - Each clause is a disjunction of literals
    $L1 \vee L2 \vee L3 \ldots$
  - Each literal is variable or its negation
    $P, \neg P, Q, \neg Q, \ldots$

Satisfiability

Terminology

- A clause is a *unit clause* if it only contains one literal
- Each clause is *empty* (=false) is it contains no literal
- A literal is *pure* if appears if its negation does not occur in any clause.
- A *free literals* is an unassigned literal of a clause

Satisfiability

Conversion to CNF

- Eliminate iff and implies
  - replace $P \Leftrightarrow Q$ by $(P \Rightarrow Q) \wedge (Q \Rightarrow P)$
  - and $P \Rightarrow Q$ by $\neg P \vee Q$
- Push negation down
  - replace $\neg(P \wedge Q)$ by $\neg P \vee \neg Q$
  - and $\neg(P \vee Q)$ by $\neg P \wedge \neg Q$

Satisfiability

Conversion to CNF

- Clausify using De Morgan's laws
  - E.g. replace $P \vee (Q \wedge R)$ by $(P \wedge Q) \vee (P \wedge R)$
- In worst case, formula grows exponentially in size
- Introduction of auxiliary literals to prevent blow up

Satisfiability

Solving SAT

- Classic methods
  - Truth tables
  - Systematic assignment through binary-search
- First practical SAT-solving procedures
  - Davis-Putnam procedure
  - Davis-Putnam-Logemann-Loveland procedure (DPLL)

Satisfiability

Davis-Putnam procedure

- Introduced by Davis & Putnam in 1960
  - Resolution rule required exponential space
- Modified by Davis, Logemann and Loveland in 1962
  - Resolution rule replaced by splitting rule
  - Trades space for time
  - Modified algorithm often inaccurately called Davis Putnam procedure

SAT-solving

Davis Putnam procedure

- Consider
  $(X \vee Y) \wedge (\neg X \vee Z) \wedge (\neg Y \vee Z) \wedge \ldots$

SAT-solving

Davis Putnam procedure

- Consider
  $(X \vee Y) \wedge (\neg X \vee Z) \wedge (\neg Y \vee Z) \wedge \ldots$
- Basic idea
  - Try X=true

SAT-solving

Davis Putnam procedure

- Consider
  $(X \vee Y) \wedge (\neg X \vee Z) \wedge (\neg Y \vee Z) \wedge \ldots$
- Basic idea
  - Try X=true
  - Remove clauses which must be satisfied

SAT-solving

Davis Putnam procedure

- Consider
  $(\neg X \vee Z) \wedge (\neg Y \vee Z) \wedge \ldots$
- Basic idea
  - Try X=true
  - Remove clauses which must be satisfied

SAT-solving

Davis Putnam procedure

- Consider
  $(\neg X \vee Z) \wedge (\neg Y \vee Z) \wedge \ldots$
- Basic idea
  - Try X=true
  - Remove clauses which must be satisfied
  - Simplify clauses containing $\neg X$

SAT-solving

Davis Putnam procedure

- Consider
  $(Z) \wedge (\neg Y \vee Z) \wedge \ldots$
- Basic idea
  - Try X=true
  - Remove clauses which must be satisfied
  - Simplify clauses containing $\neg X$

SAT-solving

Davis Putnam procedure

- Consider

  $(Z) \wedge (\neg Y \vee Z) \wedge \ldots$
- Basic idea
  - Try X=true
  - Remove clauses which must be satisfied
  - Simplify clauses containing ¬ X
  - Deduce from unit clause (Z) that Z must be true

SAT-solving

Davis Putnam procedure

- Consider

  …
- Basic idea
  - Try X=true
  - Remove clauses which must be satisfied
  - Simplify clauses containing ¬ X
  - Deduce from unit clause (Z) that Z must be true

SAT-solving

Davis Putnam procedure

- Consider

  …
- Basic idea
  - Try X=true
  - Remove clauses which must be satisfied
  - Simplify clauses containing ¬ X
  - Deduce from unit clause (Z) that Z must be true
  - Backtrack if necessary

SAT-solving

Procedure DPLL

Given a formula *f*, let C be the set of clauses

DPLL(C) is computed as follows

(SAT) if C contains no clauses return *SAT*

(Empty) if C contains an empty clause return *UNSAT*

(Split) for any variable x

if DPLL(C[*x/1*])=*SAT* or DPLL(C[*x/0*])=SAT return SAT, else return UNSAT

SAT-solving

Procedure DPLL

(continued)

(Unit) if C contains unit clause (l) then DPLL(C[l/1])

(Pure) if l is pure in C then DPLL(C[l/1])

(Taut) if $x \vee \neg x$ in C then DPLL(C \ ($x \vee \neg x$))

- The last 3 rules are characteristic for the DPLL procedure
- Neither is necessary for completeness
- Pure and Taut contribute in practice little to efficiency
- Unit rule improves efficiency greatly

SAT-solving

Procedure DPLL(C)

Space complexity
O(n)

Time complexity
O(1.618^n)

Average and best case often much better than this

Comp 4151 Ansgar Fehnker

SAT-solving

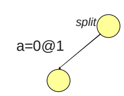
Organize the search in the form of a *decision tree*

- Each node corresponds to a *decision,* i.e application of the rule (Split)
- Apply the (Unit) rule eagerly
- Depth of the node in the decision tree is called *decision level*
- Notation: *x=v@d*
  x 2 {0,1} is assigned to *v* at decision level *d*

Comp 4151 Ansgar Fehnker

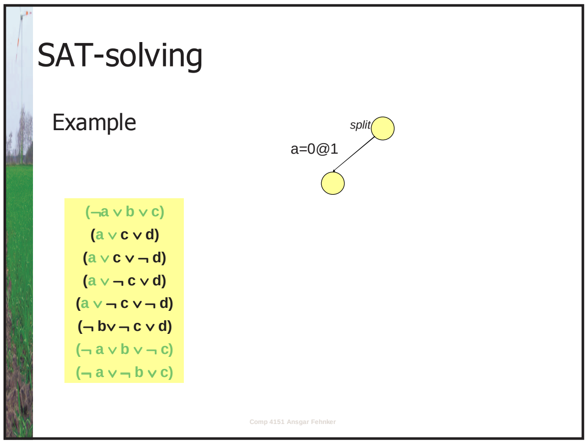
SAT-solving

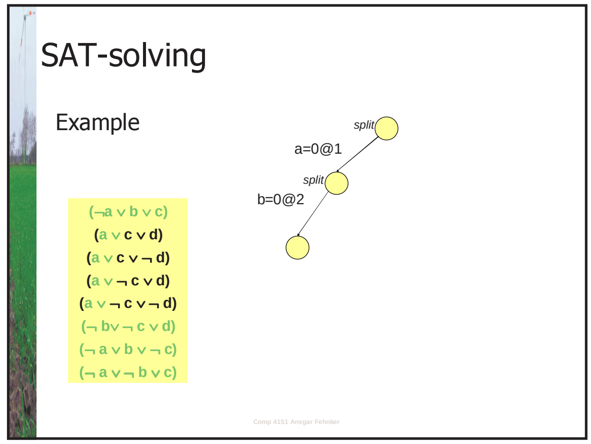
Example

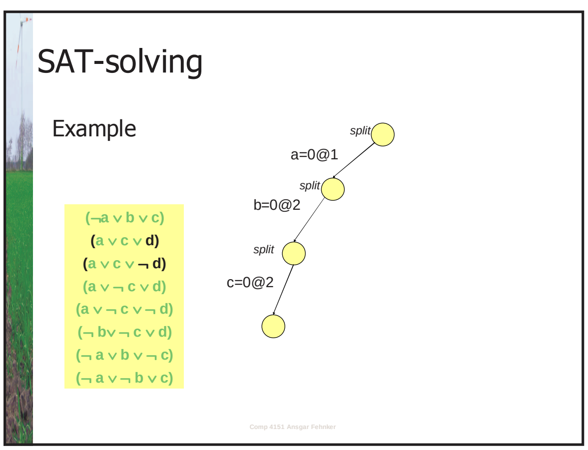


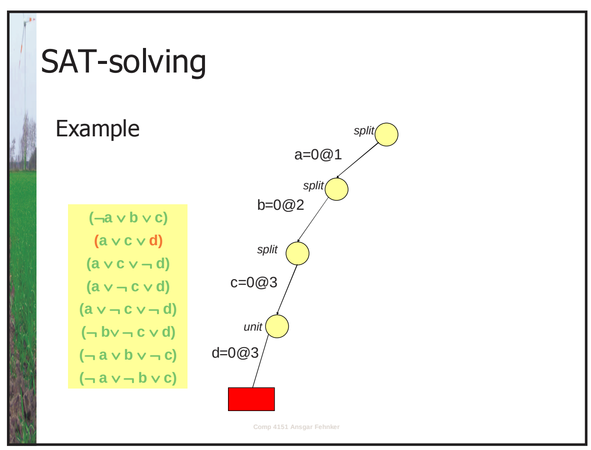
(¬a ∨ b ∨ c)
(a ∨ c ∨ d)
(a ∨ c ∨ ¬ d)
(a ∨ ¬ c ∨ d)
(a ∨ ¬ c ∨ ¬ d)
(¬ b ∨ ¬ c ∨ d)
(¬ a ∨ b ∨ ¬ c)
(¬ a ∨ ¬ b ∨ c)

Comp 4151 Ansgar Fehnker


SAT-solving
Example
(¬a ∨ b ∨ c)
(a ∨ c ∨ d)
(a ∨ c ∨ ¬ d)
(a ∨ ¬ c ∨ d)
(a ∨ ¬ c ∨ ¬ d)
(¬ b∨ ¬ c ∨ d)
(¬ a ∨ b ∨ ¬ c)
(¬ a ∨ ¬ b ∨ c)
split
a=0@1
Comp 4151 Ansgar Fehnker


SAT-solving
Example
(¬a ∨ b ∨ c)
(a ∨ c ∨ d)
(a ∨ c ∨ ¬ d)
(a ∨ ¬ c ∨ d)
(a ∨ ¬ c ∨ ¬ d)
(¬ b∨ ¬ c ∨ d)
(¬ a ∨ b ∨ ¬ c)
(¬ a ∨ ¬ b ∨ c)
split
a=0@1
split
b=0@2
Comp 4151 Ansgar Fehnker

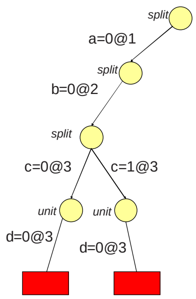

SAT-solving
Example
(¬a ∨ b ∨ c)
(a ∨ c ∨ d)
(a ∨ c ∨ ¬ d)
(a ∨ ¬ c ∨ d)
(a ∨ ¬ c ∨ ¬ d)
(¬ b∨ ¬ c ∨ d)
(¬ a ∨ b ∨ ¬ c)
(¬ a ∨ ¬ b ∨ c)
split
a=0@1
split
b=0@2
split
c=0@2
Comp 4151 Ansgar Fehnker


SAT-solving
Example
(¬a ∨ b ∨ c)
(a ∨ c ∨ d)
(a ∨ c ∨ ¬ d)
(a ∨ ¬ c ∨ d)
(a ∨ ¬ c ∨ ¬ d)
(¬ b∨ ¬ c ∨ d)
(¬ a ∨ b ∨ ¬ c)
(¬ a ∨ ¬ b ∨ c)
split
a=0@1
split
b=0@2
split
c=0@3
unit
d=0@3
Comp 4151 Ansgar Fehnker

SAT-solving

Example

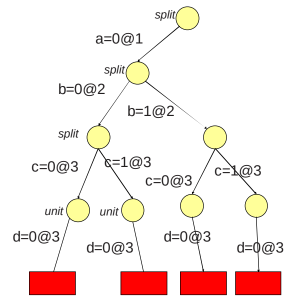
(¬a ∨ b ∨ c)
(a ∨ c ∨ d)
(a ∨ c ∨ ¬ d)
(a ∨ ¬ c ∨ d)
(a ∨ ¬ c ∨ ¬ d)
(¬ b ∨ ¬ c ∨ d)
(¬ a ∨ b ∨ ¬ c)
(¬ a ∨ ¬ b ∨ c)



SAT-solving

Example

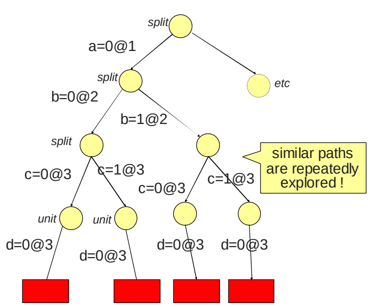
(¬a ∨ b ∨ c)
(a ∨ c ∨ d)
(a ∨ c ∨ ¬ d)
(a ∨ ¬ c ∨ d)
(a ∨ ¬ c ∨ ¬ d)
(¬ b ∨ ¬ c ∨ d)
(¬ a ∨ b ∨ ¬ c)
(¬ a ∨ ¬ b ∨ c)



SAT-solving

Example

(¬a ∨ b ∨ c)
(a ∨ c ∨ d)
(a ∨ c ∨ ¬ d)
(a ∨ ¬ c ∨ d)
(a ∨ ¬ c ∨ ¬ d)
(¬ b ∨ ¬ c ∨ d)
(¬ a ∨ b ∨ ¬ c)
(¬ a ∨ ¬ b ∨ c)



Improvements

Conflict Analysis

Conflict clause

- For each conflict clause that *explains* the conflict
- Add negation to *prevent* recurrence of same conflict

Non-chronological backtracking

- During backtrack search backtrack to one of the *causes* of the conflict

Conflict Analysis

Implication Graphs

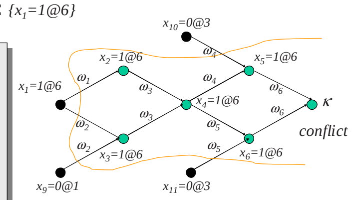
- Nodes are variable assignments to variables
- Predecessors are assignments that responsible for forcing the value of the assignment
- No predecessors for decision assignments (SPLIT)
- Conflict vertices have assignments to variables in the unsatisfied clauses

Implication graphs and learning

Current assignment: $\{x_9=0@1, x_{10}=0@3, x_{11}=0@3, x_{12}=1@2, x_{13}=1@2\}$

Current decision: $\{x_1=1@6\}$

$\omega_1 = (\neg x_1 \vee x_2)$
$\omega_2 = (\neg x_1 \vee x_3 \vee x_9)$
$\omega_3 = (\neg x_2 \vee \neg x_3 \vee x_4)$
$\omega_4 = (\neg x_4 \vee x_5 \vee x_{10})$
$\omega_5 = (\neg x_4 \vee x_6 \vee x_{11})$
$\omega_6 = (\neg x_5 \vee \neg x_6)$
$\omega_7 = (x_1 \vee x_7 \vee \neg x_{12})$
$\omega_8 = (x_1 \vee x_8)$
$\omega_9 = (\neg x_7 \vee \neg x_8 \vee \neg x_{13})$



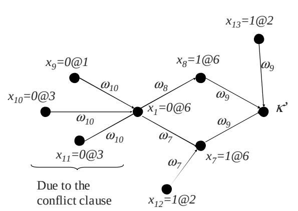
We learned $x_1 \wedge \neg x_9 \wedge \neg x_{11} \wedge \neg x_{10}$ implies f unsat

We add *conflict clause* $(\neg x_1 \vee x_9 \neg x_{11} \neg x_{10})$

Implication graphs and learning

After learning

$\omega_1 = (\neg x_1 \vee x_2)$
$\omega_2 = (\neg x_1 \vee x_3 \vee x_9)$
$\omega_3 = (\neg x_2 \vee \neg x_3 \vee x_4)$
$\omega_4 = (\neg x_4 \vee x_5 \vee x_{10})$
$\omega_5 = (\neg x_4 \vee x_6 \vee x_{11})$
$\omega_6 = (\neg x_5 \vee x_6)$
$\omega_7 = (x_1 \vee x_7 \vee \neg x_{12})$
$\omega_8 = (x_1 \vee x_8)$
$\omega_9 = (\neg x_7 \vee \neg x_8 \vee \neg x_{13})$
$\omega_{10} = (\neg x_1 \vee x_9 \neg x_{11} \neg x_{10})$

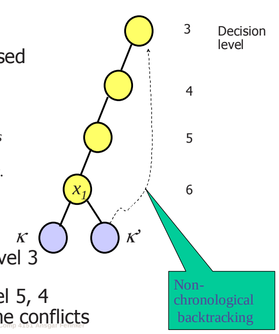


Non-chronological backtracking

Which assignments caused the conflicts ?

$x_9= 0@1$
$x_{10}= 0@3$
$x_{11}= 0@3$
$x_{12}= 1@2$
$x_{13}= 1@2$

These assignments Are sufficient for Causing a conflict.



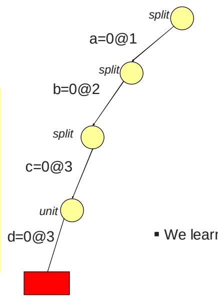
Backtrack to decision level 3

Backtracking to any level 5, 4 would generate the same conflicts

Conflict Analysis

Example

(¬a ∨ b ∨ c)
(a ∨ c ∨ d)
(a ∨ c ∨ ¬ d)
(a ∨ ¬ c ∨ d)
(a ∨ ¬ c ∨ ¬ d)
(¬ b∨ ¬ c ∨ d)
(¬ a ∨ b ∨ ¬ c)
(¬ a ∨ ¬ b ∨ c)

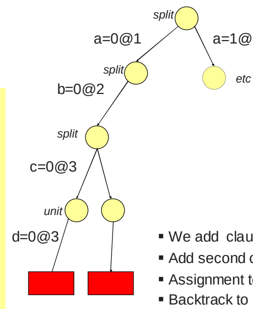


- We learn ¬ a ∧ ¬ c implies UNSAT

Conflict Analysis

Example

(¬a ∨ b ∨ c)
(a ∨ c ∨ d)
(a ∨ c ∨ ¬ d)
(a ∨ ¬ c ∨ d)
(a ∨ ¬ c ∨ ¬ d)
(¬ b∨ ¬ c ∨ d)
(¬ a ∨ b ∨ ¬ c)
(¬ a ∨ ¬ b ∨ c)
(a ∨ c)
(a)



- We add clause a ∨ c
- Add second clause a ∨ ¬c
- Assignment to b does not matter
- Backtrack to level1

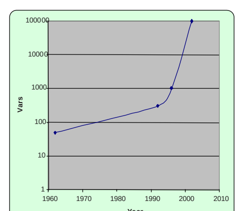
Improvements

- Non-chronological backtracking
- Clause learning
- Branching heuristics
- 2 literal watching
- Search restarts
- Randomization
- Fast data structures

SAT-solving

History

- 1st generation (1960s)
  - DP, DLL
- 2nd generation (1980s/90s)
  - POSIT, Tableau, …
- 3rd generation (mid 1990s)
  - SATO, satz, grasp, …
- 4th generation (2000s)
  - Chaff, BerkMin, …
- 5th generation?



Number of variables tackled by SAT-solvers

graphs thanks to Daniel Kroening

SAT Solving and Model Checking

Reminder

Set of initial states and transition relation can be represented as boolean functions

SAT-solvers for model checking?

- Bounded Model Checking
- SAT-based Abstraction Refinement

Bounded Model Checking

Basic Idea

- Show absence of counterexamples of length k
- Only complete for sufficiently large k
- Bounded model checking problem can be formulated as SAT problem
- For LTL, ACTL or ECTL

Either A or E path quantifiers only

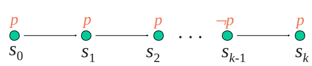
Based on presentations Daniel Kroening and Ofer Shtrichman

Bounded Model Checking

Formulation as SAT problem

Safety

Is a state reachable within k steps, which satisfies ¬p ?



Counterexample for safety properties are finite paths

Bounded Model Checking

Given a Kripke Stucture M=(S, S_0, R, L)

The reachable states in k steps are captured by:

$$I(S_0) \wedge R(S_0, S_1) \wedge \ldots \wedge R(S_{k-1}, S_k)$$

The property p fails in one of the states 1..k if

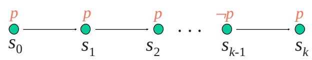
$$\neg P(S_0) \vee \neg P(S_1) \vee \ldots \vee \neg P(S_k)$$

Bounded Model Checking

Formulation as SAT problem

- The safety property p is valid up to step k iff Ω(k) is unsatisfiable:

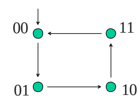
$$\Omega(k) = I(S_0) \wedge \bigwedge_{i=0}^{k-1} R(S_i, S_{i+1}) \wedge \bigvee_{i=0}^{k} \neg P(s_i)$$



Comp 4151 Ansgar Fehnker

Bounded Model Checking

Example: a two bit counter



Initial state: $I_0 = \neg l \wedge \neg r$

Transition: R: $l' = (l \neq r) \wedge r' = \neg r$

Property: **G** $(\neg l \vee \neg r)$.

The property holds within 2 steps if Ω(k) is unsatisfiable

$$\Omega(2) = (\neg l_0 \wedge \neg r_0) \wedge \begin{pmatrix} l_1 = (l_0 \neq r_0) \wedge r_1 = \neg r_0 \wedge \\ l_2 = (l_1 \neq r_1) \wedge r_2 = \neg r_1 \end{pmatrix} \wedge \begin{pmatrix} (l_0 \wedge r_0) \vee \\ (l_1 \wedge r_1) \vee \\ (l_2 \wedge r_2) \end{pmatrix}$$

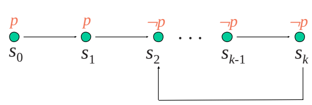
Comp 4151 Ansgar Fehnker

Bounded Model Checking

Formulation as SAT problem

Liveness

For Liveness, add a disjunction of possible loops



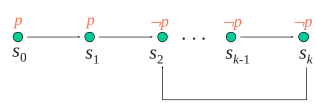
Counterexamples for liveness properties end in a loop

Comp 4151 Ansgar Fehnker

Bounded Model Checking

Liveness

For Liveness, add a disjunction of possible loops



Counterexamples for liveness properties end in a loop

Comp 4151 Ansgar Fehnker

Bounded Model Checking

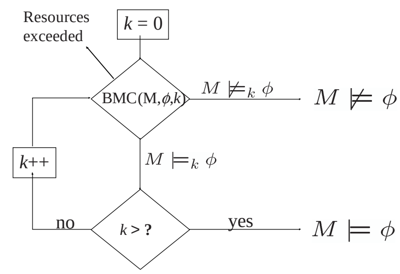
Liveness

- The liveness property **F**p is valid up to cycle k iff $\Omega(k)$ is unsatisfiable:

$$\Omega(k) = I(S_0) \wedge \bigwedge_{i=0}^{k-1} R(S_i, S_{i+1}) \wedge \bigwedge_{i=0}^{k} \neg P(S_i) \wedge \bigvee_{i=0}^{k} (S_i = S_k)$$

Bounded Model Checking

BMC-loop



Bounded Model Checking

Completeness Threshold

- For every finite model M and LTL property ϕ there exists *k* s.t.

$$M \models_k \phi \rightarrow M \models \phi$$

- The *Completeness Threshold* (CT) is the minimal such *k*
- Clearly if $M \not\models \phi$ then CT = 0
- Computing CT is a model checking by itself

Bounded Model Checking

Completeness Threshold

- *Diameter* D(M) = longest shortest path between any two reachable states.
- *Recurrence Diameter* RD(M) = longest loop-free path between any two reachable states.
- The initialized versions: DI(M) and RDI(M) start from an initial state

Bounded Model Checking

Completeness Threshold

- DI(M) is an upper bound for safety properties
- RDI(M) +1 is an upper bound for liveness properties
- However, in practice the CT is of little intrest. Too hard to compute, and too large.
- BMC is good for finding counterexamples fast

Bounded Model Checking

Tuning SAT for BMC

- *Variable ordering*
- *Incremental SAT*: reusability of conflict clauses between different (yet related) SAT instances.
- *Replicating Conflict Clauses*: generation of conflict clauses 'for free', based on the unique structure of BMC invariant properties.

Bounded Model Checking

Outlook

- BMC is available e.g. in NuSMV
- SAT-based BMC can solve instance that BDD symbolic model checkers cannot.
- Today: BMC with SAT for finding shallow errors. BDD-based procedures for proving their absence.
- BMC and BDD model checkers used as complementary methods

Next lecture: Counterexample guided abstraction refinement