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

Today: SAT-based model checking

Basic Idea
- Use algorithms that solve (in practice) difficult problems efficiently.
- Transform model checking problem to an 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 \( X \to \{0,1\} \) such that \( f \) becomes true?

- Does there exist a satisfying assignment?
- Does there exist a model for \( f \) ?

Example: A diplomatic problem

- As chief of staff, you are to send 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 incident means that you cannot invite both Romania and Peru

Who do you invite?

Example: A diplomatic problem

Given the following constraint over \( P, Q \) and \( R \)

\[
\begin{align*}
f &= (P \lor \neg Q) \land (Q \lor R) \land \neg (R \land P)
\end{align*}
\]

- Does there exist an assignment to \( P, Q, R \) such that \( f \) is true?

- Two satisfying assignments
  - \( P \mapsto 1, Q \mapsto 1, R \mapsto 0 \)
  - \( P \mapsto 0, Q \mapsto 0, R \mapsto 1 \)
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
    \[ C_1 \land C_2 \land \ldots \]
  - Each clause is a disjunction of literals
    \[ L_1 \lor L_2 \lor L_3 \ldots \]
  - Each literal is variable or its negation
    \[ P, \neg P, Q, \neg Q, \ldots \]

Terminology

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

Conversion to CNF

- Eliminate iff and implies
  - replace \( P \iff Q \) by \( (P \implies Q) \land (Q \implies P) \)
  - and \( P \implies Q \) by \( \neg P \lor Q \)
- Push negation down
  - replace \( \neg(P \land Q) \) by \( \neg P \lor \neg Q \)
  - and \( \neg(P \lor Q) \) by \( \neg P \land \neg Q \)

- Clausify using De Morgan’s laws
  - E.g. replace \( P \land (Q \land R) \) by \( (P \land Q) \lor (P \land R) \)
- 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)

SAT-solving

Davis Putnam procedure

- Consider
  
  $(X \lor Y) \land (\neg X \lor Z) \land (\neg Y \lor Z) \land \ldots$

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
  
  $(X \lor Y) \land (\neg X \lor Z) \land (\neg Y \lor Z) \land \ldots$

- Basic idea
  - Try $X=true$
SAT-solving

Davis Putnam procedure

- Consider
  \((X \lor Y) \land (\neg X \lor Z) \land (\neg Y \lor Z) \land \ldots\)
- Basic idea
  - Try \(X = \text{true}\)
  - Remove clauses which must be satisfied

Basic idea

Try \(X = \text{true}\)

Remove clauses which must be satisfied
SAT-solving

Davis Putnam procedure

- Consider
  \((Z) \land (\neg Y \lor Z) \land \ldots\)

- Basic idea
  - Try \(X=\text{true}\)
  - Remove clauses which must be satisfied
  - Simplify clauses containing \(\neg X\)
  - Deduce from unit clause \((Z)\) that \(Z\) must be true

Basic idea
- Try \(X=\text{true}\)
- Remove clauses which must be satisfied
- Simplify clauses containing \(\neg X\)
- 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

\[
\text{DPLL}(C) \text{ is computed as follows}
\]

(SAT) if \(C\) contains no clauses return \(\text{SAT}\)
(Empty) if \(C\) contains an empty clause return \(\text{UNSAT}\)
(Split) for any variable \(x\)
  - if \(\text{DPLL}(C[x'=1]) = \text{SAT}\) or \(\text{DPLL}(C[x'=0]) = \text{SAT}\)
  - return \(\text{SAT}\), else return \(\text{UNSAT}\)
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 \( x \lor \neg x \) in C then DPLL(C \ (x \lor \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

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: \( x=v@d \)
  \( x \in \{0,1\} \) is assigned to \( v \) at decision level \( d \)

SAT-solving

Example

\[-a \lor b \lor c \]
\[(a \lor c \lor d)\]
\[(a \lor c \lor d)\]
\[(a \lor c \lor d)\]
\[(\neg b \lor c \lor d)\]
\[(\neg a \lor b \lor c \lor d)\]
\[(\neg a \lor b \lor c \lor d)\]
SAT-solving

Example

\(a=0@1\)

\((-a \lor b \lor c)\)

\((a \lor c \lor d)\)

\((a \lor \neg c \lor \neg d)\)

\((a \lor \neg c \lor \neg d)\)

\((b \lor \neg c \lor d)\)

\((b \lor \neg c \lor d)\)

\((\neg a \lor b \lor \neg c)\)

\((\neg a \lor b \lor \neg c)\)

\(b=0@2\)

\((-a \lor b \lor c)\)

\((a \lor c \lor d)\)

\((a \lor \neg c \lor \neg d)\)

\((a \lor \neg c \lor \neg d)\)

\((b \lor \neg c \lor d)\)

\((b \lor \neg c \lor d)\)

\((\neg a \lor b \lor \neg c)\)

\((\neg a \lor b \lor \neg c)\)

\((\neg a \lor b \lor \neg c)\)

\(c=0@2\)

\((\neg a \lor b \lor \neg c)\)

\((\neg a \lor b \lor \neg c)\)

\((\neg a \lor b \lor \neg c)\)

\(d=0@3\)

\((-a \lor b \lor c)\)

\((a \lor c \lor d)\)

\((a \lor \neg c \lor \neg d)\)

\((a \lor \neg c \lor \neg d)\)

\((b \lor \neg c \lor d)\)

\((b \lor \neg c \lor d)\)

\((\neg a \lor b \lor \neg c)\)

\((\neg a \lor b \lor \neg c)\)

\(c=0@3\)

\((\neg a \lor b \lor \neg c)\)

\((\neg a \lor b \lor \neg c)\)

\((\neg a \lor b \lor \neg c)\)

\(d=0@3\)
SAT-solving

Example

\[
\begin{align*}
\neg a \lor b \lor c \\
(\neg a \lor c \lor d) \\
(\neg a \lor c \lor \neg d) \\
(\neg a \lor \neg c \lor d) \\
(\neg a \lor \neg c \lor \neg d) \\
(\neg b \lor c \lor d) \\
(\neg a \lor \neg b \lor c) \\
(\neg a \lor \neg b \lor \neg c)
\end{align*}
\]

\[
\begin{align*}
\neg a, c, d, & a, c, b & a, c, \neg b, d & a, c, \neg b, \neg d & a, \neg c, b, d & a, \neg c, b, \neg d & a, \neg c, \neg b, d & a, \neg c, \neg b, \neg d \\
\neg a, c, \neg d, & a, c, b & a, c, \neg b, d & a, c, \neg b, \neg d & a, \neg c, b, d & a, \neg c, b, \neg d & a, \neg c, \neg b, d & a, \neg c, \neg b, \neg d \\
\neg a, \neg c, d, & a, c, b & a, c, \neg b, d & a, c, \neg b, \neg d & a, \neg c, b, d & a, \neg c, b, \neg d & a, \neg c, \neg b, d & a, \neg c, \neg b, \neg d \\
\neg a, \neg c, \neg d, & a, c, b & a, c, \neg b, d & a, c, \neg b, \neg d & a, \neg c, b, d & a, \neg c, b, \neg d & a, \neg c, \neg b, d & a, \neg c, \neg b, \neg d \\
\neg b, c, d, & a, c, b & a, c, \neg b, d & a, c, \neg b, \neg d & a, \neg c, b, d & a, \neg c, b, \neg d & a, \neg c, \neg b, d & a, \neg c, \neg b, \neg d \\
\neg b, c, \neg d, & a, c, b & a, c, \neg b, d & a, c, \neg b, \neg d & a, \neg c, b, d & a, \neg c, b, \neg d & a, \neg c, \neg b, d & a, \neg c, \neg b, \neg d \\
\end{align*}
\]

Similar paths are repeatedly explored!

Improvements

Conflict Analysis

- 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_1=0@1, x_2=0@3, x_3=0@1, x_4=1@2, x_5=1@2\}

Current decision: \{x_6=1@1\}

After learning

Non-chronological backtracking

Which assignments caused the conflicts?

- \(x_1 = 0@1\)
- \(x_2 = 0@3\)
- \(x_3 = 0@1\)
- \(x_4 = 1@2\)
- \(x_5 = 1@2\)

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

- We learn $\neg a \land \neg c$ implies UNSAT

- We add clause $a \lor c$
- Add second clause $a \lor \neg c$
- Assignment to $b$ does not matter
- Backtrack to level 1

**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?
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

Formulation as SAT problem

Safety

Is a state reachable within \( k \) steps, which satisfies \( \neg p \) ?

\[
\begin{array}{cccccc}
S_0 & S_1 & S_2 & \ldots & S_{k-1} & S_k \\
\hline
p & p & p & \ldots & \neg p & p \\
\end{array}
\]

Counterexample for safety properties are finite paths

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

The reachable states in \( k \) steps are captured by:

\[
I(S_0) \land R(S_0, S_1) \land \ldots \land R(S_{k-1}, S_k)
\]

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

\[
\neg P(S_0) \lor \neg P(S_1) \lor \ldots \lor \neg P(S_k)
\]
Bounded Model Checking

Formulation as SAT problem

The safety property $p$ is valid up to step $k$ iff $\Omega(k)$ is unsatisfiable:

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

Example: a two bit counter

Initial state: $I_0 = \neg l \land \neg r$
Transition: $R: l' = (l \neq r) \land r' = \neg r$
Property: $G (\neg l \lor \neg r)$.

The property holds within 2 steps if $\Omega(k)$ is unsatisfiable:

$$\Omega(2) = (\neg l_0 \land \neg r_0) \land (l_1 = (l_0 \neq r_0) \land r_1 = \neg r_0) \land (l_2 = (l_1 \neq r_1) \land r_2 = \neg r_1) \land (l_0 \land r_0) \lor (l_1 \land r_1) \lor (l_2 \land r_2)$$

Bounded Model Checking

Liveness

For Liveness, add a disjunction of possible loops

Counterexamples for liveness properties end in a loop
Bounded Model Checking

Liveness

- The liveness property \( F_p \) is valid up to cycle \( k \) iff \( \Omega(k) \) is unsatisfiable:

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

Bounded Model Checking

Completeness Threshold

- For every finite model \( M \) and LTL property \( \phi \) there exists \( k \) s.t.

\( M \models_k \phi \implies M \models \phi \)

- The Completeness Threshold (CT) is the minimal such \( k \)

- Clearly if \( M \models \phi \) 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 \( RD(M) \) start from an initial state
Bounded Model Checking

Completeness Threshold

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

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