Boolean Satisfiability & Model Checking

Joao Marques-Silva

School of Electronics and Computer Science
University of Southampton

February 2008
Part I

Introduction to Boolean Satisfiability
Motivation – A Logic Puzzle

- If the unicorn is mythical, then it is immortal
- If the unicorn is not mythical, then it is a mortal mammal
- If the unicorn is either immortal or a mammal, then it is horned
- The unicorn is magical if it is horned

- Is the unicorn mythical? Is it magical? Is it horned?
Outline (First Lecture)

What is Boolean Satisfiability?

Applications

Modeling

Algorithms
- Fundamentals
- Local Search
- The DPLL Algorithm
- Conflict-Driven Clause Learning (CDCL)

Extensions
Outline

What is Boolean Satisfiability?

Applications

Modeling

Algorithms
  Fundamentals
  Local Search
  The DPLL Algorithm
  Conflict-Driven Clause Learning (CDCL)

Extensions
Boolean Formulas

- Boolean formula \( \varphi \) is defined over a set of propositional variables \( x_1, \ldots, x_n \), using the standard propositional connectives \( \neg, \land, \lor, \rightarrow, \leftrightarrow \), and parenthesis
  - The domain of propositional variables is \( \{0, 1\} \)
  - Example: \( \varphi(x_1, \ldots, x_3) = ((\neg x_1 \land x_2) \lor x_3) \land (\neg x_2 \lor x_3) \)

- A formula \( \varphi \) in conjunctive normal form (CNF) is a conjunction of disjunctions (clauses) of literals, where a literal is a variable or its complement
  - Example: \( \varphi(x_1, \ldots, x_3) = (\neg x_1 \lor x_2) \land (\neg x_2 \lor x_3) \)

- Can encode any Boolean formula into CNF (more later)
### Some Boolean Logic Rules

- $\neg \neg x_1 \vdash x_1$
- $x_1 \rightarrow x_2 \vdash (\neg x_1 \lor x_2)$
- $\neg (x_1 \land x_2) \vdash (\neg x_1 \lor \neg x_2)$
- $\neg (x_1 \lor x_2) \vdash (\neg x_1 \land \neg x_2)$
- $x_1 \rightarrow x_2, \neg x_2 \vdash \neg x_1$ (Modus Tollens)
- $(x_1 \lor \neg x_2) \land (\neg x_1 \lor x_3) \vdash (\neg x_2 \lor x_3)$ (Resolution)

... 

- **Note:** $\vdash$ denotes **valid consequence**
Boolean Satisfiability (SAT)

- The Boolean satisfiability (SAT) problem:
  - Find an assignment to the variables $x_1, \ldots, x_n$ such that $\varphi(x_1, \ldots, x_n) = 1$, or prove that no such assignment exists

- SAT is an **NP-complete** decision problem [Cook’71]
  - SAT was the first problem to be shown NP-complete
  - There are no known polynomial time algorithms for SAT
  - 36-year old conjecture:
    Any algorithm that solves SAT is exponential in the number of variables, in the worst-case
An Example Puzzle I

• If the unicorn is mythical, then it is immortal
• If the unicorn is not mythical, then it is a mortal mammal
• If the unicorn is either immortal or a mammal, then it is horned
• The unicorn is magical if it is the horned

• Is the unicorn mythical? Is it magical? Is it horned?
An Example Puzzle II

- M: The unicorn is mythical
- I: The unicorn is immortal
- L: The unicorn is mammal
- H: The unicorn is horned
- G: The unicorn is magical

- If the unicorn is mythical, then it is immortal
  \[ M \rightarrow I \]
- If the unicorn is not mythical, then it is a mortal mammal
  \[ \neg M \rightarrow (\neg I \land L) \]
- If the unicorn is either immortal or a mammal, then it is horned
  \[ (I \lor L) \rightarrow H \]
- The unicorn is magical if it is the horned
  \[ H \rightarrow G \]
• Putting it all together:
  – If the unicorn is mythical, then it is immortal
  – If the unicorn is not mythical, then it is a mortal mammal
  – If the unicorn is either immortal or a mammal, then it is horned
  – The unicorn is magical if it is horned
  – $M$: The unicorn is mythical
  – $I$: The unicorn is immortal
  – $L$: The unicorn is mammal
  – $H$: The unicorn is horned
  – $G$: The unicorn is magical

$\ (M \rightarrow I) \land (\neg M \rightarrow (\neg I \land L)) \land ((I \lor L) \rightarrow H) \land (H \rightarrow G) \)
An Example Puzzle III

• Is the unicorn **mythical**? Is it **magical**? Is it **horned**?

\[
(M \rightarrow I) \land (\neg M \rightarrow (\neg I \land L)) \land ((I \lor L) \rightarrow H) \land (H \rightarrow G) \models
\]

\[
(\neg M \lor I) \land (M \lor (\neg I \land L)) \land ((I \lor L) \rightarrow H) \land (H \rightarrow G) \models
\]

\[
(\neg M \lor I) \land (M \lor \neg I) \land (M \lor L) \land ((I \lor L) \rightarrow H) \land (H \rightarrow G) \models
\]

\[
(I \lor L) \land ((I \lor L) \rightarrow H) \land (H \rightarrow G) \models
\]

\[
H \land G
\]

• Hence, the unicorn is not necessarily mythical, but it is **horned** and **magical**!

• **NOTE:** Formula manipulation is often impractical
  
  – Formulas obtained from real-world applications can easily have **hundreds of thousands** of variables and **millions** of clauses
Outline

What is Boolean Satisfiability?

Applications

Modeling

Algorithms
  Fundamentals
  Local Search
  The DPLL Algorithm
  Conflict-Driven Clause Learning (CDCL)

Extensions
Applications of SAT I

- **Formal methods:**
  - Hardware model checking; Software model checking; Termination analysis of term-rewrite systems; Test pattern generation (testing of software & hardware); etc.

- **Artificial intelligence:**
  - Planning; Knowledge representation; Games (n-queens, sudoku, social golpher’s, etc.)

- **Bioinformatics:**
  - Haplotype inference; Pedigree checking; etc.

- **Design automation:**
  - Equivalence checking; Delay computation; Fault diagnosis; Noise analysis; etc.

- **Security:**
  - Cryptanalysis; Inversion attacks on hash functions; etc.
Applications of SAT II

• Computationally hard problems:
  – Graph coloring; Traveling salesperson; etc.

• Mathematical problems:
  – van der Waerden numbers; etc.

• Core engine for other solvers: 0-1 ILP; QBF; #SAT; SMT; ...

• Integrated into theorem provers: HOL; Isabelle; ...
Example: Graph Coloring I

- Decide whether one can assign one of $K$ colors to each of the vertices of graph $G = (V, E)$ such that adjacent vertices are assigned different colors.

- Valid coloring

- Invalid coloring
Example: Graph Coloring II

- Given $N = |V|$ vertices and $K$ colors, create $N \times K$ variables: $x_{ij} = 1$ iff vertex $i$ is assigned color $j$; 0 otherwise.

- For each edge $(u, v)$, require different assigned colors to $u$ and $v$:
  \[ 1 \leq j \leq K, \quad (\neg x_{uj} \lor \neg x_{vj}) \]

- Each vertex is assigned exactly one color:
  \[ 1 \leq i \leq N, \quad \sum_{j=1}^{K} x_{ij} = 1 \]
Example: The N-Queens Problem

The N-Queens Problem: Place N queens on a $N \times N$ board, such that no two queens attack each other.

Example for a $5 \times 5$ board:
Example: The N-Queens Problem II

- $x_{ij}$: 1 if queen placed in position $(i, j)$; 0 otherwise
- Each row must have exactly one queen:
  \[ 1 \leq i \leq N, \quad \sum_{j=1}^{N} x_{ij} = 1 \]
- Each column must have exactly one queen:
  \[ 1 \leq j \leq N, \quad \sum_{i=1}^{N} x_{ij} = 1 \]
- Also, need to define constraints on diagonals...
Example: The N-Queens Problem III

- Each diagonal can have at most one queen:

\[
\begin{align*}
&i = 1, \quad 2 \leq j < N, \quad \sum_{k=0}^{j-1} x_{i+k} j-k \leq 1 \\
i = N, \quad 1 \leq j < N, \quad \sum_{k=0}^{N-j} x_{i-k} j+k \leq 1 \\
j = 1, \quad 1 \leq i < N, \quad \sum_{k=0}^{N-i} x_{i+k} j+k \leq 1 \\
j = N, \quad 2 \leq i < N, \quad \sum_{k=0}^{i-1} x_{i-k} j-k \leq 1
\end{align*}
\]
Outline

What is Boolean Satisfiability?

Applications

Modeling

Algorithms
  Fundamentals
  Local Search
  The DPLL Algorithm
  Conflict-Driven Clause Learning (CDCL)

Extensions
Representing AtLeast, AtMost and Equals Constraints

• How to represent in CNF the constraint $\sum_{j=1}^{N} x_j \geq 1$?
  – Standard solution: $(x_1 \lor \ldots \lor x_N)$

• How to represent in CNF the constraint $\sum_{j=1}^{N} x_{ij} \leq 1$?
  – Naive solution: $\forall j_1 = 1..N \forall j_2 = j_1 + 1..N \neg x_{ij_1} \lor \neg x_{ij_2}$
    ▶ Number of clauses grows quadratically with $N$
  – More compact (e.g. linear) solutions possible

• How to represent in CNF the constraint $\sum_{j=1}^{N} x_{ij} = 1$?
  – Standard solution: one AtMost 1 and one AtLeast 1 constraints
Examples of CNF Encodings

- Encode $\sum_{j=1}^{n} x_j \leq 1$ with sequential counter:

$$(\neg x_1 \lor s_1) \land (\neg x_n \lor \neg s_{n-1}) \land \bigwedge_{1<i<n} ((\neg x_i \lor s_i) \land (\neg s_{i-1} \lor s_i) \land (\neg x_i \lor \neg s_{i-1}))$$

- $O(n)$ clauses
- $O(n)$ auxiliary variables
Examples of CNF Encodings

- Encode $\sum_{j=1}^{n} x_j \leq 1$ with bitwise encoding:
  - Define $r = \lceil \log n \rceil$ (with $n > 1$)
  - Auxiliary variables $v_0, \ldots, v_{r-1}$
  - Associate with each $x_j$ the binary representation of $j - 1$
  - Create clauses $\neg x_j \lor p_i$, $i = 0, \ldots, r - 1$, where
    - $p_i = v_i$ if the binary representation of $j - 1$ has value 1 in position $i$
    - $p_i = \neg v_i$ otherwise
  - If $x_j = 1$, assignment to $v_i$ variables must encode binary representation of $j - 1$
  - If all $x_j = 0$, any assignment to $v_i$ variables is consistent
  - $O(n \log n)$ clauses
  - $O(\log n)$ auxiliary variables
Satisfiability problems can be defined on Boolean circuits/formulas. Can represent circuits/formulas as CNF formulas [Tseitin’68].

For each (simple) gate, CNF formula encodes the consistent assignments to the gate’s inputs and output.

- Given $z = \text{OP}(x, y)$, represent in CNF $z \leftrightarrow \text{OP}(x, y)$.
- CNF formula for the circuit is the conjunction of CNF formula for each gate.

$$\varphi_c = (a \lor c) \land (b \lor c) \land (\neg a \lor \neg b \lor \neg c)$$

$$\varphi_t = (\neg r \lor t) \land (\neg s \lor t) \land (r \lor s \lor \neg t)$$
Representing Boolean Circuits / Formulas II

\[ \varphi_c = (a \lor c) \land (b \lor c) \land (\neg a \lor \neg b \lor \neg c) \]

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>\varphi_c(a,b,c)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Representing Boolean Circuits / Formulas III

- CNF formula for the circuit is the conjunction of the CNF formula for each gate
  - Can specify objectives with additional clauses

\[
\varphi = (a \lor x) \land (b \lor x) \land (\neg a \lor \neg b \lor \neg x) \land (x \lor \neg y) \land (c \lor \neg y) \land (\neg x \lor \neg c \lor y) \land (\neg y \lor z) \land (\neg d \lor z) \land (y \lor d \lor \neg z) \land (z)
\]

- Note: \( z = d \lor (c \land (\neg (a \land b))) \)
Outline

What is Boolean Satisfiability?

Applications

Modeling

Algorithms
- Fundamentals
- Local Search
- The DPLL Algorithm
- Conflict-Driven Clause Learning (CDCL)

Extensions
Algorithms for SAT

• Incomplete algorithms (i.e. cannot prove unsatisfiability):
  – Local search / hill-climbing
  – Genetic algorithms
  – Simulated annealing
  –...

• Complete algorithms (i.e. can prove unsatisfiability):
  – Proof system(s)
    ▶ Natural deduction
    ▶ Resolution
    ▶ Stalmarck’s method
    ▶ Recursive learning
    ▶...
  – Binary Decision Diagrams (BDDs)
  – Backtrack search / DPLL
    ▶ Conflict-Driven Clause Learning (CDCL)
  –...

[e.g. Huth & Ryan’04]
Outline

What is Boolean Satisfiability?

Applications

Modeling

Algorithms

Fundamentals
  Local Search
  The DPLL Algorithm
  Conflict-Driven Clause Learning (CDCL)

Extensions
Definitions

- Propositional variables can be assigned value 0 or 1
  - In some contexts variables may be unassigned

- A clause is **satisfied** if at least one of its literals is assigned value 1
  \[ x_1 \lor \neg x_2 \lor \neg x_3 \]

- A clause is **unsatisfied** if all of its literals are assigned value 0
  \[ x_1 \lor \neg x_2 \lor \neg x_3 \]

- A clause is **unit** if it contains one single unassigned literal and all other literals are assigned value 0
  \[ x_1 \lor \neg x_2 \lor \neg x_3 \]

- A formula is **satisfied** if all of its clauses are satisfied
- A formula is **unsatisfied** if at least one of its clauses is unsatisfied
Pure Literals

- A literal is pure if only occurs as a positive literal or as a negate literal in a CNF formula
  - Example:
    \[ \varphi = (\neg x_1 \lor x_2) \land (x_3 \lor \neg x_2) \land (x_4 \lor \neg x_5) \land (x_5 \lor \neg x_4) \]
  - \(x_1\) and \(x_3\) and pure literals

- Pure literal rule:
  Clauses containing pure literals can be removed from the formula (i.e. just assign pure literals to the values that satisfy the clauses)
  - For the example above, the resulting formula becomes:
    \[ \varphi = (x_4 \lor \neg x_5) \land (x_5 \lor \neg x_4) \]

- A reference technique until the mid 90s; nowadays seldom used
Unit Propagation

- **Unit clause rule:**
  Given a unit clause, its only unassigned literal must be assigned value 1 for the clause to be satisfied
  - Example: for unit clause \((x_1 \lor \neg x_2 \lor \neg x_3)\), \(x_3\) must be assigned value 0

- **Unit propagation**
  Iterated application of the unit clause rule
  \[
  (x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_3 \lor x_4) \land (\neg x_1 \lor \neg x_2 \lor x_4)
  \]
Unit Propagation

- **Unit clause rule:**
  Given a unit clause, its only unassigned literal must be assigned value 1 for the clause to be satisfied
  - Example: for unit clause \((x_1 \lor \neg x_2 \lor \neg x_3)\), \(x_3\) must be assigned value 0

- **Unit propagation**
  Iterated application of the unit clause rule
  \[ (x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_3 \lor x_4) \land (\neg x_1 \lor \neg x_2 \lor x_4) \]
Unit Propagation

- **Unit clause rule:**
  Given a unit clause, its only unassigned literal must be assigned value 1 for the clause to be satisfied
  - Example: for unit clause \((x_1 \lor \neg x_2 \lor \neg x_3)\), \(x_3\) must be assigned value 0

- **Unit propagation**
  Iterated application of the unit clause rule

\[
(x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_3 \lor x_4) \land (\neg x_1 \lor \neg x_2 \lor x_4)
\]
Unit Propagation

- **Unit clause rule:**
  Given a unit clause, its only unassigned literal must be assigned value 1 for the clause to be satisfied
  - Example: for unit clause \((x_1 \lor \neg x_2 \lor \neg x_3)\), \(x_3\) must be assigned value 0

- **Unit propagation**
  Iterated application of the unit clause rule
  
  \[
  (x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_3 \lor x_4) \land (\neg x_1 \lor \neg x_2 \lor x_4)
  \]

  \[
  (x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_3 \lor x_4) \land (\neg x_1 \lor \neg x_2 \lor \neg x_4)
  \]
Unit Propagation

- **Unit clause rule:**
  Given a unit clause, its only unassigned literal must be assigned value 1 for the clause to be satisfied
  
  - Example: for unit clause \((x_1 \lor \neg x_2 \lor \neg x_3)\), \(x_3\) must be assigned value 0

- **Unit propagation**
  Iterated application of the unit clause rule

\[
(x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_3 \lor x_4) \land (\neg x_1 \lor \neg x_2 \lor x_4)
\]

\[
(x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_3 \lor x_4) \land (\neg x_1 \lor \neg x_2 \lor \neg x_4)
\]
Unit Propagation

• **Unit clause rule:**
  Given a unit clause, its only unassigned literal must be assigned value 1 for the clause to be satisfied
  - Example: for unit clause \((x_1 \lor \neg x_2 \lor \neg x_3)\), \(x_3\) must be assigned value 0

• **Unit propagation**
  Iterated application of the unit clause rule

\[
(x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_3 \lor x_4) \land (\neg x_1 \lor \neg x_2 \lor x_4)
\]

\[
(x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_3 \lor x_4) \land (\neg x_1 \lor \neg x_2 \lor \neg x_4)
\]
Unit Propagation

- **Unit clause rule:**
  Given a unit clause, its only unassigned literal must be assigned value 1 for the clause to be satisfied
  - Example: for unit clause \((x_1 \lor \neg x_2 \lor \neg x_3)\), \(x_3\) must be assigned value 0

- **Unit propagation**
  Iterated application of the unit clause rule
  \[(x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_3 \lor x_4) \land (\neg x_1 \lor \neg x_2 \lor x_4)\]
  \[(x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_3 \lor x_4) \land (\neg x_1 \lor \neg x_2 \lor \neg x_4)\]

- **Unit propagation** can satisfy clauses but can also unsatisfy clauses (i.e. conflicts)
Resolution

- Resolution rule:
  - If a formula $\varphi$ contains clauses $(x \lor \alpha)$ and $(\neg x \lor \beta)$, then one can infer $(\alpha \lor \beta)$

\[(x \lor \neg \alpha) \land (\neg x \lor \beta) \models (\alpha \lor \beta)\]

- Resolution forms the basis of a complete algorithm for SAT
  - Iteratively apply the following steps: [Davis&Putnam’60]
    - Select variable $x$
    - Apply resolution rule between every pair of clauses of the form $(x \lor \alpha)$ and $(\neg x \lor \beta)$
    - Remove all clauses containing either $x$ or $\neg x$
    - Apply the pure literal rule and unit propagation
  - Terminate when either the empty clause or the empty formula is derived
Resolution – An Example

\[(x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_2 \lor \neg x_3) \land (x_2 \lor x_3) \land (x_3 \lor x_4) \land (x_3 \lor \neg x_4) \models\]
Resolution – An Example

\[(x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_2 \lor \neg x_3) \land (x_2 \lor x_3) \land (x_3 \lor x_4) \land (x_3 \lor \neg x_4) \models \]
\[
(\neg x_2 \lor \neg x_3) \land (x_2 \lor x_3) \land (x_3 \lor x_4) \land (x_3 \lor \neg x_4) \models \]
Resolution – An Example

\[(x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_2 \lor \neg x_3) \land (x_2 \lor x_3) \land (x_3 \lor x_4) \land (x_3 \lor \neg x_4) \vdash\]

\[(\neg x_2 \lor \neg x_3) \land (x_2 \lor x_3) \land (x_3 \lor x_4) \land (x_3 \lor \neg x_4) \vdash\]

\[(x_3 \lor \neg x_3) \land (x_3 \lor x_4) \land (x_3 \lor \neg x_4) \vdash\]
Resolution – An Example

\[(x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_2 \lor \neg x_3) \land (x_2 \lor x_3) \land (x_3 \lor x_4) \land (x_3 \lor \neg x_4) \models \]

\[(\neg x_2 \lor \neg x_3) \land (x_2 \lor x_3) \land (x_3 \lor x_4) \land (x_3 \lor \neg x_4) \models \]

\[(x_3 \lor \neg x_3) \land (x_3 \lor x_4) \land (x_3 \lor \neg x_4) \models \]

\[(x_3 \lor x_4) \land (x_3 \lor \neg x_4) \models \]
Resolution – An Example

\[(x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_2 \lor \neg x_3) \land (x_2 \lor x_3) \land (x_3 \lor x_4) \land (x_3 \lor \neg x_4) \vdash \]
\[(\neg x_2 \lor \neg x_3) \land (x_2 \lor x_3) \land (x_3 \lor x_4) \land (x_3 \lor \neg x_4) \]
\[(x_3 \lor \neg x_3) \land (x_3 \lor x_4) \land (x_3 \lor \neg x_4) \]
\[(x_3 \lor x_4) \land (x_3 \lor \neg x_4) \]
\[(x_3) \]

- Formula is SAT
Outline

What is Boolean Satisfiability?

Applications

Modeling

Algorithms
  - Fundamentals
  - Local Search
    - The DPLL Algorithm
    - Conflict-Driven Clause Learning (CDCL)

Extensions
Organization of Local Search

- Local search is incomplete; it cannot prove unsatisfiability
  - Very effective in specific contexts

- Example:

\[(x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_3 \lor x_4) \land (\neg x_1 \lor \neg x_2 \lor x_4)\]
Organization of Local Search

- Local search is incomplete; it cannot prove unsatisfiability
  - Very effective in specific contexts

- Example:

\[(x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_3 \lor x_4) \land (\neg x_1 \lor \neg x_2 \lor x_4)\]

- Start with (possibly random) assignment:  \(x_4 = 0, x_1 = x_2 = x_3 = 1\)
- And repeat a number of times:
Organization of Local Search

- Local search is incomplete; it cannot prove unsatisfiability
  - Very effective in specific contexts

- Example:

\[
(x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_3 \lor x_4) \land (\neg x_1 \lor \neg x_2 \lor x_4)
\]

- Start with (possibly random) assignment: \(x_4 = 0, x_1 = x_2 = x_3 = 1\)
- And repeat a number of times:
Organization of Local Search

- Local search is incomplete; it cannot prove unsatisfiability
  - Very effective in specific contexts

- Example:

\[(x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_3 \lor x_4) \land (\neg x_1 \lor \neg x_2 \lor x_4)\]

- Start with (possibly random) assignment: \(x_4 = 0, x_1 = x_2 = x_3 = 1\)
- And repeat a number of times:
  - If not all clauses satisfied, flip variable (e.g. \(x_4\))
Organization of Local Search

- Local search is incomplete; it cannot prove unsatisfiability
  - Very effective in specific contexts

- Example:

\[(x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_3 \lor x_4) \land (\neg x_1 \lor \neg x_2 \lor x_4)\]

- Start with (possibly random) assignment: \(x_4 = 0, x_1 = x_2 = x_3 = 1\)
- And repeat a number of times:
  - If not all clauses satisfied, flip variable (e.g. \(x_4\))
Organization of Local Search

- Local search is incomplete; it cannot prove unsatisfiability
  - Very effective in specific contexts

- Example:

\[(x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_3 \lor x_4) \land (\neg x_1 \lor \neg x_2 \lor x_4)\]

- Start with (possibly random) assignment: \(x_4 = 0, x_1 = x_2 = x_3 = 1\)
- And repeat a number of times:
  - If not all clauses satisfied, flip variable (e.g. \(x_4\))
  - Done if all clauses satisfied
Organization of Local Search

- Local search is incomplete; it cannot prove unsatisfiability
  - Very effective in specific contexts

- Example:

\[(x_1 \lor \neg x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_3 \lor x_4) \land (\neg x_1 \lor \neg x_2 \lor x_4)\]

- Start with (possibly random) assignment: \(x_4 = 0, x_1 = x_2 = x_3 = 1\)
- And repeat a number of times:
  - If not all clauses satisfied, flip variable (e.g. \(x_4\))
  - Done if all clauses satisfied

- Repeat (random) selection of assignment a number of times
Outline

What is Boolean Satisfiability?

Applications

Modeling

Algorithms
  Fundamentals
  Local Search
  The DPLL Algorithm
  Conflict-Driven Clause Learning (CDCL)

Extensions
Historical Perspective

- In 1960, M. Davis and H. Putnam proposed the DP algorithm:
  - Resolution used to eliminate 1 variable at each step
  - Applied the pure literal rule and unit propagation
- Original algorithm was inefficient
- In 1962, M. Davis, G. Logemann and D. Loveland proposed an alternative algorithm:
  - Instead of eliminating variables, the algorithm would split on a given variable at each step
  - Also applied the pure literal rule and unit propagation
- The 1962 algorithm is actually an implementation of backtrack search
- Over the years the 1962 algorithm became known as the DPLL (sometimes DLL) algorithm
Basic Algorithm for SAT – DPLL

- **Standard** backtrack search
- At each step:
  - Select decision assignment
  - Apply unit propagation and (optionally) the pure literal rule
  - If conflict identified, then backtrack
    - If cannot backtrack further, return **UNSAT**
    - Otherwise, proceed with unit propagation
  - If formula satisfied, return **SAT**
  - Otherwise, proceed with another decision
An Example of DPLL

\[ \varphi = (a \lor \neg b \lor d) \land (a \lor \neg b \lor e) \land \\
(\neg b \lor \neg d \lor \neg e) \land \\
(a \lor b \lor c \lor d) \land (a \lor b \lor c \lor \neg d) \land \\
(a \lor b \lor \neg c \lor e) \land (a \lor b \lor \neg c \lor \neg e) \]
An Example of DPLL

\[ \varphi = (a \lor \neg b \lor d) \land (a \lor \neg b \lor e) \land \]
\[ (\neg b \lor \neg d \lor \neg e) \land \]
\[ (a \lor b \lor c \lor d) \land (a \lor b \lor c \lor \neg d) \land \]
\[ (a \lor b \lor \neg c \lor e) \land (a \lor b \lor \neg c \lor \neg e) \]
An Example of DPLL

\[ \varphi = (a \lor \neg b \lor d) \land (a \lor \neg b \lor e) \land \\
(\neg b \lor \neg d \lor \neg e) \land \\
(a \lor b \lor c \lor d) \land (a \lor b \lor c \lor \neg d) \land \\
(a \lor b \lor \neg c \lor e) \land (a \lor b \lor \neg c \lor \neg e) \]
An Example of DPLL

\[ \varphi = (a \lor \neg b \lor d) \land (a \lor \neg b \lor e) \land \\
(\neg b \lor \neg d \lor \neg e) \land \\
(a \lor b \lor c \lor d) \land (a \lor b \lor c \lor \neg d) \land \\
(a \lor b \lor \neg c \lor e) \land (a \lor b \lor \neg c \lor \neg e) \]
An Example of DPLL

\[ \varphi = (a \lor \neg b \lor d) \land (a \lor \neg b \lor e) \land \\
(\neg b \lor \neg d \lor \neg e) \land \\
(a \lor b \lor c \lor d) \land (a \lor b \lor c \lor \neg d) \land \\
(a \lor b \lor \neg c \lor e) \land (a \lor b \lor \neg c \lor \neg e) \]

conflict
An Example of DPLL

\[ \varphi = (a \lor \neg b \lor d) \land (a \lor \neg b \lor e) \land
\neg b \lor \neg d \lor \neg e \land
(a \lor b \lor c \lor d) \land (a \lor b \lor c \lor \neg d) \land
(a \lor b \lor \neg c \lor e) \land (a \lor b \lor \neg c \lor \neg e) \land
\]
An Example of DPLL

\[ \varphi = (a \lor \neg b \lor d) \land (a \lor \neg b \lor e) \land \\
(\neg b \lor \neg d \lor \neg e) \land \\
(a \lor b \lor c \lor d) \land (a \lor b \lor c \lor \neg d) \land \\
(a \lor b \lor \neg c \lor e) \land (a \lor b \lor \neg c \lor \neg e) \]

conflict
An Example of DPLL

\[ \varphi = (a \lor \neg b \lor d) \land (a \lor \neg b \lor e) \land
(\neg b \lor \neg d \lor \neg e) \land
(a \lor b \lor c \lor d) \land (a \lor b \lor c \lor \neg d) \land
(a \lor b \lor \neg c \lor e) \land (a \lor b \lor \neg c \lor \neg e) \]

[Diagram showing a DPLL tree with vertices labeled a, b, c, and a solution path highlighted with green.]
Outline

What is Boolean Satisfiability?

Applications

Modeling

Algorithms
  Fundamentals
  Local Search
  The DPLL Algorithm
  Conflict-Driven Clause Learning (CDCL)

Extensions
CDCL SAT Solvers

• Inspired on DPLL
  – Must be able to prove unsatisfiability

• New clauses are learnt from conflicts

• Structure of conflicts exploited (UIPs)

• Backtracking can be non-chronological

• Efficient data structures
  – Compact and reduced maintenance overhead

• Backtrack search is periodically restart

• Can solve instances with hundreds of thousand variables and tens of million clauses
CDCL SAT Solvers

• Inspired on DPLL
  – Must be able to prove unsatisfiability
• New clauses are learnt from conflicts
• Structure of conflicts exploited (UIPs)
• Backtracking can be non-chronological
• Efficient data structures
  – Compact and reduced maintenance overhead
• Backtrack search is periodically restart

• Can solve instances with hundreds of thousand variables and tens of million clauses
Clause Learning

- During backtrack search, for each conflict learn new clause, which explains and prevents repetition of the same conflict

\[ \varphi = (a \lor b) \land (\neg b \lor c \lor d) \land (\neg b \lor e) \land (\neg d \lor \neg e \lor f) \ldots \]
Clause Learning

- During backtrack search, for each conflict learn new clause, which explains and prevents repetition of the same conflict

$$\varphi = (a \lor b) \land (\neg b \lor c \lor d) \land (\neg b \lor e) \land (\neg d \lor \neg e \lor f) \ldots$$

- Assume decisions $c = 0$ and $f = 0$
Clause Learning

- During backtrack search, for each conflict learn new clause, which explains and prevents repetition of the same conflict

$$\varphi = (a \lor b) \land (\neg b \lor c \lor d) \land (\neg b \lor e) \land (\neg d \lor \neg e \lor f) \ldots$$

- Assume decisions $c = 0$ and $f = 0$
- Assign $a = 0$ and imply assignments
Clause Learning

• During backtrack search, for each conflict learn new clause, which explains and prevents repetition of the same conflict

\[ \varphi = (a \lor b) \land (\neg b \lor c \lor d) \land (\neg b \lor e) \land (\neg d \lor \neg e \lor f) \ldots \]

- Assume decisions \( c = 0 \) and \( f = 0 \)
- Assign \( a = 0 \) and imply assignments
Clause Learning

- During backtrack search, for each conflict learn new clause, which explains and prevents repetition of the same conflict

\[ \varphi = (a \lor b) \land (\neg b \lor c \lor d) \land (\neg b \lor e) \land (\neg d \lor \neg e \lor f) \ldots \]

- Assume decisions \( c = 0 \) and \( f = 0 \)
- Assign \( a = 0 \) and imply assignments
- A conflict is reached: \((\neg d \lor \neg e \lor f)\) is unsatisfied
Clause Learning

- During backtrack search, for each conflict learn new clause, which explains and prevents repetition of the same conflict

$$\varphi = (a \lor b) \land (\neg b \lor c \lor d) \land (\neg b \lor e) \land (\neg d \lor \neg e \lor f) \ldots$$

- Assume decisions $c = 0$ and $f = 0$
- Assign $a = 0$ and imply assignments
- A conflict is reached: $(\neg d \lor \neg e \lor f)$ is unsatisfied
- $(a = 0) \land (c = 0) \land (f = 0) \Rightarrow (\varphi = 0)$
Clause Learning

• During backtrack search, for each conflict learn new clause, which explains and prevents repetition of the same conflict

$$\varphi = (a \lor b) \land (\neg b \lor c \lor d) \land (\neg b \lor e) \land (\neg d \lor \neg e \lor f) \ldots$$

- Assume decisions $c = 0$ and $f = 0$
- Assign $a = 0$ and imply assignments
- A conflict is reached: $(\neg d \lor \neg e \lor f)$ is unsatisfied
- $(a = 0) \land (c = 0) \land (f = 0) \Rightarrow (\varphi = 0)$
- $(\varphi = 1) \Rightarrow (a = 1) \lor (c = 1) \lor (f = 1)$
Clause Learning

- During backtrack search, for each conflict learn new clause, which explains and prevents repetition of the same conflict

\[ \varphi = (a \lor b) \land (\neg b \lor c \lor d) \land (\neg b \lor e) \land (\neg d \lor \neg e \lor f) \ldots \]

- Assume decisions \( c = 0 \) and \( f = 0 \)
- Assign \( a = 0 \) and imply assignments
- A conflict is reached: \( (\neg d \lor \neg e \lor f) \) is unsatisfied
- \( (a = 0) \land (c = 0) \land (f = 0) \Rightarrow (\varphi = 0) \)
- \( (\varphi = 1) \Rightarrow (a = 1) \lor (c = 1) \lor (f = 1) \)
- Learn new clause \( (a \lor c \lor f) \)
Non-Chronological Backtracking

- During backtrack search, for each conflict backtrack to one of the causes of the conflict

\[ \varphi = (a \lor b) \land (\neg b \lor c \lor d) \land (\neg b \lor e) \land (\neg d \lor \neg e \lor f) \land (a \lor c \lor f) \land (\neg a \lor g) \land (\neg g \lor b) \land (\neg h \lor j) \land (\neg i \lor k) \]
Non-Chronological Backtracking

- During backtrack search, for each conflict backtrack to one of the causes of the conflict

\[ \varphi = (a \lor b) \land (\neg b \lor c \lor d) \land (\neg b \lor e) \land (\neg d \lor \neg e \lor f) \land (a \lor c \lor f) \land (\neg a \lor g) \land (\neg g \lor b) \land (\neg h \lor j) \land (\neg i \lor k) \]

- Assume decisions \( c = 0, f = 0, h = 0 \) and \( i = 0 \)
Non-Chronological Backtracking

• During backtrack search, for each conflict backtrack to one of the causes of the conflict

\[ \varphi = (a \lor b) \land (\neg b \lor c \lor d) \land (\neg b \lor e) \land (\neg d \lor \neg e \lor f) \land (a \lor c \lor f) \land (\neg a \lor g) \land (\neg g \lor b) \land (\neg h \lor j) \land (\neg i \lor k) \]

- Assume decisions \( c = 0, f = 0, h = 0 \) and \( i = 0 \)
- Assignment \( a = 0 \) caused conflict \( \Rightarrow \) learnt clause \( (a \lor c \lor f) \) implies \( a = 1 \)
Non-Chronological Backtracking

- During backtrack search, for each conflict backtrack to one of the causes of the conflict

\[ \varphi = (a \lor b) \land (\neg b \lor c \lor d) \land (\neg b \lor e) \land (\neg d \lor \neg e \lor f) \land (a \lor c \lor f) \land (\neg a \lor g) \land (\neg g \lor b) \land (\neg h \lor j) \land (\neg i \lor k) \]

- Assume decisions \( c = 0, \ f = 0, \ h = 0 \) and \( i = 0 \)
- Assignment \( a = 0 \) caused conflict \( \Rightarrow \) learnt clause \( (a \lor c \lor f) \) implies \( a = 1 \)
Non-Chronological Backtracking

• During backtrack search, for each conflict backtrack to one of the causes of the conflict

$$\varphi = (a \lor b) \land (\neg b \lor c \lor d) \land (\neg b \lor e) \land (\neg d \lor \neg e \lor f) \land$$

$$(a \lor c \lor f) \land (\neg a \lor g) \land (\neg g \lor b) \land (\neg h \lor j) \land (\neg i \lor k)$$

- Assume decisions \( c = 0 \), \( f = 0 \), \( h = 0 \) and \( i = 0 \)
- Assignment \( a = 0 \) caused conflict ⇒ learnt clause \((a \lor c \lor f)\) implies \( a = 1 \)
Non-Chronological Backtracking

- During backtrack search, for each conflict backtrack to one of the causes of the conflict

\[ \varphi = (a \lor b) \land (\neg b \lor c \lor d) \land (\neg b \lor e) \land (\neg d \lor \neg e \lor f) \land (a \lor c \lor f) \land (\neg a \lor g) \land (\neg g \lor b) \land (\neg h \lor j) \land (\neg i \lor k) \]

- Assume decisions \( c = 0, f = 0, h = 0 \) and \( i = 0 \)
- Assignment \( a = 0 \) caused conflict \( \Rightarrow \) learnt clause \((a \lor c \lor f)\) implies \( a = 1 \)
- A conflict is again reached: \((\neg d \lor \neg e \lor f)\) is unsatisfied
Non-Chronological Backtracking

- During backtrack search, for each conflict backtrack to one of the causes of the conflict

\[
\varphi = (a \lor b) \land (\neg b \lor c \lor d) \land (\neg b \lor e) \land (\neg d \lor \neg e \lor f) \land \\
(a \lor c \lor f) \land (\neg a \lor g) \land (\neg g \lor b) \land (\neg h \lor j) \land (\neg i \lor k)
\]

- Assume decisions \( c = 0, f = 0, h = 0 \) and \( i = 0 \)
- Assignment \( a = 0 \) caused conflict \( \Rightarrow \) learnt clause \((a \lor c \lor f)\) implies \( a = 1 \)
- A conflict is again reached: \((\neg d \lor \neg e \lor f)\) is unsatisfied
- \((c = 0) \land (f = 0) \Rightarrow (\varphi = 0)\)
Non-Chronological Backtracking

- During backtrack search, for each conflict backtrack to one of the causes of the conflict

\[ \varphi = (a \lor b) \land (\neg b \lor c \lor d) \land (\neg b \lor e) \land (\neg d \lor \neg e \lor f) \land (a \lor c \lor f) \land (\neg a \lor g) \land (\neg g \lor b) \land (\neg h \lor j) \land (\neg i \lor k) \]

  - Assume decisions \( c = 0, f = 0, h = 0 \) and \( i = 0 \)
  - Assignment \( a = 0 \) caused conflict \( \Rightarrow \) learnt clause \( (a \lor c \lor f) \) implies \( a = 1 \)
  - A conflict is again reached: \( (\neg d \lor \neg e \lor f) \) is unsatisfied
  - \( (c = 0) \land (f = 0) \Rightarrow (\varphi = 0) \)
  - \( (\varphi = 1) \Rightarrow (c = 1) \lor (f = 1) \)
Non-Chronological Backtracking

- During backtrack search, for each conflict backtrack to one of the causes of the conflict

\[
\phi = (a \lor b) \land (\neg b \lor c \lor d) \land (\neg b \lor e) \land (\neg d \lor \neg e \lor f) \land \\
(\neg c \lor c \lor f) \land (\neg a \lor g) \land (\neg g \lor b) \land (\neg h \lor j) \land (\neg i \lor k)
\]

- Assume decisions \( c = 0 \), \( f = 0 \), \( h = 0 \) and \( i = 0 \)
- Assignment \( a = 0 \) caused conflict \( \Rightarrow \) learnt clause \((a \lor c \lor f)\) implies \( a = 1 \)
- A conflict is again reached: \((\neg d \lor \neg e \lor f)\) is unsatisfied
- \((c = 0) \land (f = 0) \Rightarrow (\phi = 0)\)
- \((\phi = 1) \Rightarrow (c = 1) \lor (f = 1)\)
- Learn new clause \((c \lor f)\)
Non-Chronological Backtracking

\[(a + c + f) (c + f)\]
Non-Chronological Backtracking

- Learnt clause: \((c \lor f)\)
- Need to backtrack, given new clause
- Backtrack to most recent decision: \(f = 0\)

- Clause learning and non-chronological backtracking are hallmarks of modern SAT solvers
Data Structures

- **Standard data structures (adjacency lists):**
  - Each variable $x$ keeps a reference to all clauses containing a literal in $x$
    - If variable $x$ is assigned, then all clauses containing a literal in $x$ are evaluated
    - If search backtracks, then all clauses of all newly unassigned variables are updated
  - Total number of references is $L$, where $L$ is the number of literals

- **Lazy data structures (watched literals):**
  - For each clause, only two variables keep a reference to the clause, i.e. only 2 literals are watched
    - If variable $x$ is assigned, only the clauses where literals in $x$ are watched need to be evaluated
    - If search backtracks, then nothing needs to be done
  - Total number of references is $2 \times C$, where $C$ is the number of clauses
    - In general $L \gg 2 \times C$, in particular if clauses are learnt
### Evolution of SAT Solvers

<table>
<thead>
<tr>
<th>Instance</th>
<th>Posit’94</th>
<th>Grasp’96</th>
<th>Chaff’01</th>
<th>Siege’04</th>
</tr>
</thead>
<tbody>
<tr>
<td>ssa2670-136</td>
<td>28.53</td>
<td>0.36</td>
<td>0.02</td>
<td>0.01</td>
</tr>
<tr>
<td>bf1355-638</td>
<td>772.45</td>
<td>0.04</td>
<td>0.02</td>
<td>0.01</td>
</tr>
<tr>
<td>design_1</td>
<td>&gt; 7200</td>
<td>65.35</td>
<td>1.27</td>
<td>0.29</td>
</tr>
<tr>
<td>design_3</td>
<td>&gt; 7200</td>
<td>9.13</td>
<td>0.52</td>
<td>0.41</td>
</tr>
<tr>
<td>f_ind</td>
<td>&gt; 7200</td>
<td>4663.89</td>
<td>17.91</td>
<td>6.52</td>
</tr>
<tr>
<td>splitter_42</td>
<td>&gt; 7200</td>
<td>&gt; 7200</td>
<td>28.81</td>
<td>4.46</td>
</tr>
<tr>
<td>c6288</td>
<td>&gt; 7200</td>
<td>&gt; 7200</td>
<td>&gt; 7200</td>
<td>2847.46</td>
</tr>
<tr>
<td>pipe_64_32</td>
<td>&gt; 7200</td>
<td>&gt; 7200</td>
<td>&gt; 7200</td>
<td>&gt; 7200</td>
</tr>
</tbody>
</table>

- Modern SAT algorithms can solve instances with hundreds of thousands of variables and tens of millions of clauses
Outline

What is Boolean Satisfiability?

Applications

Modeling

Algorithms
  Fundamentals
  Local Search
  The DPLL Algorithm
  Conflict-Driven Clause Learning (CDCL)

Extensions
Well-Known Extensions of SAT

- Quantify the variables
  - Quantified Boolean Formulas (QBF):
    Boolean formulas where variables are existentially or universally quantified
- Maximize number of satisfied clauses
  - (Weighted) (Partial) Maximum Satisfiability
- Consider extended constraints
  - Pseudo-Boolean formulas (PBS/PBO):
    Linear inequalities over Boolean variables w/o or w/ cost function
- Consider decidable fragments of FOL
  - Satisfiability Modulo Theories
    Decision procedures for a number of theories exist
    ▶ Linear Integer Arithmetic
    ▶ Difference Arithmetic
    ▶ Uninterpreted Functions
    ▶ ...

- Some extensions promising; still far from the impact of SAT solvers
References on Boolean Satisfiability


Part II

Model Checking with Boolean Satisfiability
Outline of Part I (First Lecture)

What is Boolean Satisfiability?

Applications

Modeling

Algorithms
  Fundamentals
  Local Search
  The DPLL Algorithm
  Conflict-Driven Clause Learning (CDCL)

Extensions
Outline of Part II (Second Lecture)

Temporal Logics

Model Checking
  What is Model Checking?
  Model Checking by Explicit State Manipulation
  Symbolic Model Checking

Bounded Model Checking

Unbounded Model Checking
  Induction
  Interpolation

Concluding Remarks
Temporal Logics

Model Checking
  What is Model Checking?
  Model Checking by Explicit State Manipulation
  Symbolic Model Checking

Bounded Model Checking

Unbounded Model Checking
  Induction
  Interpolation

Concluding Remarks
Temporal Logics

- Allow representing temporal properties of systems
  - Bad state is never reached
  - Some target state can always be reached
  - ...

- Example temporal logics:
  - Literal-time logic (LTL)
    - No quantification over computation paths
    - No restriction on nesting temporal operators
  - Computation-Tree Logic (CTL)
    - Quantification over computation paths
    - Restriction on nesting temporal operators
  - Other temporal logics: CTL*, ATL, etc.

- Focus on CTL
  - Some of the algorithms are easier to describe
  - SAT can be applied to either CTL or LTL model checking
Linear-Time Logic (LTL)

• Syntax of LTL formulas:

\[ \phi ::= \top \mid \bot \mid p \mid (\neg \phi) \mid (\phi \lor \phi) \mid (\phi \land \phi) \mid (\phi \to \phi) \mid (X \phi) \mid (F \phi) \mid (G \phi) \mid (\phi U \phi) \mid (\phi W \phi) \mid (\phi U \phi) \]

where \( p \) is a propositional atom from set \( \text{Atoms} \)

• LTL temporal connectives:
  – \( X \): neXt operator
  – \( F \): some Future state
  – \( G \): all future states (Globally)
  – \( U \): Until; \( W \): Weak-until; \( R \): Release

• Convention on operator binding:
  – Most tight: \( \neg \), \( X \), \( F \), \( G \)
  – \( U \), \( R \), \( W \)
  – \( \land \), \( \lor \)
  – Least tight: \( \to \)
Computation-Tree Logic (CTL)

• Syntax of CTL formulas:

\[ \phi ::= \top | \bot | p | (\neg \phi) | (\phi \lor \phi) | (\phi \land \phi) | (\phi \rightarrow \phi) | (AX \phi) | (AF \phi) | (AG \phi) | A[\phi U \phi] \\
| (EX \phi) | (EF \phi) | (EG \phi) | E[\phi U \phi] \]

where \( p \) is a propositional atom from set \( \text{Atoms} \)

• CTL temporal connectives:
  - \( A \): All paths; \( E \): Exists a path
  - \( X \): neXt operator
  - \( F \): some Future state
  - \( G \): all future states (Globally)
  - \( U \): Until

• Convention on operator binding:
  - Most tight: \( \neg, AG, EG, AF, EF, AX, EX \)
  - \( \land, \lor \)
  - Least tight: \( \rightarrow, AU, EU \)
Transition Systems

- A transition system (or model) $M = (S, \rightarrow, L)$ is defined by:
  - Set of states $S$
  - Transition relation $\rightarrow$, binary relation defined on $S$
  - Labelling function $L : S \rightarrow \mathcal{P}(\text{Atoms})$

- An example:
  - States: $\{s_0, s_1, s_2\}$
  - Transition Relation: $\{s_0 \rightarrow s_1, s_1 \rightarrow s_0, s_0 \rightarrow s_2, s_1 \rightarrow s_2, s_2 \rightarrow s_2\}$
  - Labels: $\{p, q, r\}$
    - E.g. $L(s_1) = \{q, r\}$
Paths in Transition Systems

- Given a model $\mathcal{M} = (S, \rightarrow, L)$, a path is an infinite sequence of states $s_1, s_2, s_3, \ldots$ in $S$ such that, for each $i \geq 1$, $s_i \rightarrow s_{i+1}$
- A path is written as $\pi = s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \ldots$
- $\pi^i$ represents the suffix starting at $s_i$

![Diagram](image)

- $s_0 \rightarrow s_1 \rightarrow s_0 \rightarrow s_1 \rightarrow \ldots$
- $s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow s_2 \rightarrow \ldots$
- $s_0 \rightarrow s_1 \rightarrow s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \ldots$
Semantics of CTL I

- Let $\mathcal{M} = (S, \rightarrow, L)$ be a model for CTL, $s$ in $S$, $\phi$ a CTL formula. The relation $\mathcal{M}, s \models \phi$ is defined by structural induction on $\phi$:
  - $\mathcal{M}, s \models \top$ and $\mathcal{M}, s \not\models \bot$
  - $\mathcal{M}, s \models p$ iff $p \in L(s)$
  - $\mathcal{M}, s \models \neg \phi$ iff $\mathcal{M}, s \not\models \phi$
  - $\mathcal{M}, s \models \phi_1 \land \phi_2$ iff $\mathcal{M}, s \models \phi_1$ and $\mathcal{M}, s \models \phi_2$
  - $\mathcal{M}, s \models \phi_1 \lor \phi_2$ iff $\mathcal{M}, s \models \phi_1$ or $\mathcal{M}, s \models \phi_2$
  - $\mathcal{M}, s \models \phi_1 \rightarrow \phi_2$ iff ...
  - $\mathcal{M}, s \models AX \phi$ holds iff ...
  - $\mathcal{M}, s \models EX \phi$ holds iff for some state $s_1$, such that $s \rightarrow s_1$ it is true that $\mathcal{M}, s_1 \models \phi$
    - $\phi$ is true in some neXt state
  - $\mathcal{M}, s \models AG \phi$ holds iff for all paths $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \ldots$, where $s_1$ equals $s$, and all $s_i$ along the path, it is true that $\mathcal{M}, s_i \models \phi$
    - Along A computation paths beginning in $s$, $\phi$ holds Globally
  - $\mathcal{M}, s \models EG \phi$ holds iff ...
Semantics of CTL II

- Let $\mathcal{M} = (S, \rightarrow, L)$ be a model for CTL, $s$ in $S$, $\phi$ a CTL formula. The relation $\mathcal{M}, s \models \phi$ is defined by structural induction on $\phi$:
  - $\mathcal{M}, s \models AF \phi$ holds iff for all paths $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \ldots$, where $s_1$ equals $s$, there is some $s_i$ along each path such that $\mathcal{M}, s_i \models \phi$
    ▶ For All computation paths beginning in $s$ there will be some Future state where $\phi$ holds
  - $\mathcal{M}, s \models EF \phi$ holds iff there is a path $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \ldots$, where $s_1$ equals $s$, there is $s_i$ along that path such that $\mathcal{M}, s_i \models \phi$
    ▶ There Exists a computation path beginning in $s$ such that $\phi$ holds in some Future state
  - $\mathcal{M}, s \models A[\phi_1 U \phi_2]$ holds iff $\ldots$
  - $\mathcal{M}, s \models E[\phi_1 U \phi_2]$ holds iff there is a path $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \ldots$, where $s_1$ equals $s$, and $\phi_1 U \phi_2$ is satisfied, i.e. there is some $s_i$ along the path such that $\mathcal{M}, s_i \models \phi_2$, and, for each $j < i$, it is true that $\mathcal{M}, s_j \models \phi_1$
Examples

\[ \mathcal{M}, s_0 \models p \land q \]
Examples

$\mathcal{M}, s_0 \vDash p \land q$

$\mathcal{M}, s_0 \vDash EG \, r$
Examples

\[ \mathcal{M}, s_0 \models p \land q \]
\[ \mathcal{M}, s_0 \models \text{EG } r \]
\[ \mathcal{M}, s_0 \models \text{AF } q \]
Examples

\[ M, s_0 \models p \land q \]
\[ M, s_0 \models EG r \]
\[ M, s_0 \models AF q \]
\[ M, s_0 \models EF q \]
Examples

\[ M, s_0 \models p \land q \]
\[ M, s_0 \models EG r \]
\[ M, s_0 \models AF q \]
\[ M, s_0 \not\models EF q \]
\[ M, s_0 \not\models AG r \]
Examples

Typical CTL property:
- Safety property: $\text{AG } \neg b$
  
  In all computation paths it is globally true that nothing bad happens (i.e. $\neg b$)
More Examples

\[ M, s_0 \models EG r ? \]

\[ s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \ldots \]
More Examples

$\mathcal{M}, s_0 \models EF q$?

$s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \ldots$

$\mathcal{M}, s_0 \models EG r$?
More Examples

\[ \mathcal{M}, s_0 \models EG\ r ? \]
\[ s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \ldots \]

\[ \mathcal{M}, s_0 \models EF\ q ? \]

\[ \mathcal{M}, s_0 \not\models E[t \cup q] ? \]
\[ s_0 \text{ not labelled with } t \]
More Examples

\[ M, s_0 \models EG \ r \ ? \]
\[ s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \ldots \]

\[ M, s_0 \models EF \ q \ ? \]

\[ M, s_0 \not\models E[t \cup q] \ ? \]
\[ s_0 \text{ not labelled with } t \]

\[ M, s_0 \models E[r \cup q] \ ? \]
More Examples

\[ \mathcal{M}, s_0 \models EG r \]?

\[ s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \ldots \]

\[ \mathcal{M}, s_0 \models EF q \]?

\[ \mathcal{M}, s_0 \not\models E[t U q] \]?

\[ s_0 \text{ not labelled with } t \]

\[ \mathcal{M}, s_0 \models E[r U q] \]?

\[ \mathcal{M}, s_0 \not\models EG (AX r) \]?
More Examples

\[ \mathcal{M}, s_0 \models \text{EG } r \, ? \]
\[ s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \ldots \]
\[ \mathcal{M}, s_0 \models \text{EF } q \, ? \]
\[ \mathcal{M}, s_0 \not\models \text{E}[t \cup q] \, ? \]
\[ s_0 \text{ not labelled with } t \]
\[ \mathcal{M}, s_0 \models \text{E}[r \cup q] \, ? \]
\[ \mathcal{M}, s_0 \not\models \text{EG (AX } r) \, ? \]
\[ \mathcal{M}, s_0 \models \text{EG (EX } r) \, ? \]
More Examples

\[
\begin{align*}
\mathcal{M}, s_0 &\models \text{EG } r \\
&\quad s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \ldots \\
\mathcal{M}, s_0 &\models \text{EF } q \\
\mathcal{M}, s_0 &\not\models \text{E}[t \cup q] \\
&\quad s_0 \text{ not labelled with } t \\
\mathcal{M}, s_0 &\models \text{E}[r \cup q] \\
\mathcal{M}, s_0 &\not\models \text{EG (AX } r) \\
\mathcal{M}, s_0 &\models \text{EG (EX } r) \\
\mathcal{M}, s_3 &\models \text{AG (EX } r)
\end{align*}
\]
More Examples

\[ \mathcal{M}, s_0 \models EG \, r \ ? \]  
\[ s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \ldots \]

\[ \mathcal{M}, s_0 \models EF \, q \ ? \]

\[ \mathcal{M}, s_0 \not\models E[t \cup q] \ ? \]
\[ s_0 \text{ not labelled with } t \]

\[ \mathcal{M}, s_0 \models E[r \cup q] \ ? \]

\[ \mathcal{M}, s_0 \not\models EG (AX \, r) \ ? \]

\[ \mathcal{M}, s_0 \models EG (EX \, r) \ ? \]

\[ \mathcal{M}, s_3 \models AG (EX \, r) \ ? \]

\[ \mathcal{M}, s_3 \models AG (AX \, r) \ ? \]
Adequate Set of Connectives

- AF, EU, EX, $\land$, $\neg$, and $\bot$ are an adequate set of connectives
  - All other connectives can be expressed as combinations of these connectives
  - There are other sets of adequate connectives

- Expressing the other temporal connectives in terms of the chosen connectives (AF, EU, EX):

  \[
  \begin{align*}
  EF \phi & \equiv E[\top \mathbf{U} \phi] \\
  AG \phi & \equiv \neg EF \neg \phi \\
  EG \phi & \equiv \neg AF \neg \phi \\
  AX \phi & \equiv \neg EX \neg \phi \\
  A[\phi_1 \mathbf{U} \phi_2] & \equiv \neg (E[\neg \phi_2 \mathbf{U} (\neg \phi_1 \land \phi_2)] \lor EG \neg \phi_2)
  \end{align*}
  \]
Adequate Set of Connectives

- AF, EU, EX, $\land$, $\neg$, and $\bot$ are an adequate set of connectives
  - All other connectives can be expressed as combinations of these connectives
  - There are other sets of adequate connectives

- Expressing the other temporal connectives in terms of the chosen connectives (AF, EU, EX):

  \[
  \begin{align*}
  EF \phi & \equiv E[\top U \phi] \\
  AG \phi & \equiv \neg E[\top U \neg \phi] \\
  EG \phi & \equiv \neg AF \neg \phi \\
  AX \phi & \equiv \neg EX \neg \phi
  \end{align*}
  \]

  \[
  A[\phi_1 U \phi_2] \equiv \neg (E[\neg \phi_2 U (\neg \phi_1 \land \phi_2)] \lor \neg AF \phi_2)
  \]
Outline

Temporal Logics

Model Checking
  What is Model Checking?
  Model Checking by Explicit State Manipulation
  Symbolic Model Checking

Bounded Model Checking

Unbounded Model Checking
  Induction
  Interpolation

Concluding Remarks
Outline

Temporal Logics

Model Checking

What is Model Checking?
Model Checking by Explicit State Manipulation
Symbolic Model Checking

Bounded Model Checking

Unbounded Model Checking
Induction
Interpolation

Concluding Remarks
Model Checking

• Given:
  – A model $\mathcal{M} = (S, \rightarrow, L)$
  – A CTL formula $\phi$
  – A state $s$

• $\mathcal{M}, s \models \phi$ is said to hold iff given $\mathcal{M}$, $\phi$ holds in $s$

• Approaches to model checking:
  – **Explicit** manipulation of system’s states
    ▶ Some successful model checkers are based on explicit state manipulation, e.g. SPIN
  – **Implicit** manipulation of system’s states – **symbolic** model checking
    ▶ Explicit state manipulation may be unfeasible
      Suppose microprocessor with a 64-bit program counter; number of possible states is $2^{64}$; infeasible to explicit manipulate $2^{64}$ possible states
    ▶ Existing approaches: BDDs, SAT
Outline

Temporal Logics

Model Checking
  What is Model Checking?
  Model Checking by Explicit State Manipulation
  Symbolic Model Checking

Bounded Model Checking

Unbounded Model Checking
  Induction
  Interpolation

Concluding Remarks
Explicit State Manipulation by Labelling I

- Given temporal formula $\phi$, start from smallest subformulas and work towards $\phi$
- Consider adequate set of connectives
- Assume each target subformula $\psi$ of $\phi$, and the sets of states satisfying the subformulas of $\psi$ have been labelled
- Decide next labellings given that $\psi$ is (logic operators):
  - $\bot$: no states are labelled with $\bot$
  - $p$: label $s$ with $p$ if $p \in L(s)$
  - $\psi_1 \land \psi_2$: label $s$ with $\psi_1 \land \psi_2$ if $s$ is already labelled with $\psi_1$ and $\psi_2$
  - $\neg \psi_1$: label $s$ with $\neg \psi_1$ if $s$ is not already labelled with $\psi_1$
Decide next labellings given that $\psi$ is (temporal operators):

- **EX $\psi_1$:** label any state with EX $\psi_1$ if any of its successors is labelled with $\psi_1$
- **AF $\psi_1$:**
  - If any state is labelled with $\psi_1$, label it with AF $\psi_1$
  - Repeat: label any state with AF $\psi_1$ if all its successor states are labelled with AF $\psi_1$, until there is not change
- **E[$\psi_1$ U $\psi_2$]:**
  - If any state is labelled with $\psi_2$, label it with E[$\psi_1$ U $\psi_2$]
  - Repeat: label any state with E[$\psi_1$ U $\psi_2$] if it is labelled with $\psi_1$ and at least one of its successors is labelled with E[$\psi_1$ U $\psi_2$], until there is not change
Example 1

- $\mathcal{M}, s_0 \models \phi$, with $\phi = \text{AF } q$
  - Label $\{s_0, s_3\}$ with $\phi$, due to $q$
  - Label $s_2$ with $\phi$
  - Cannot label $s_1$
  - States where $\phi$ holds: $S_f = \{s_0, s_2, s_3\}$
  - Since, $s_0 \in S_f$, then $\mathcal{M}, s_0 \models \phi$ holds
  - Note that $\mathcal{M}, s_1 \not\models \phi$, since $s_1 \not\in S_f$
Example II

$M, s_0 \models \phi$, with $\phi = \text{EX (EX } r \text{)}$

- States where $\psi_1 = \text{EX } r$ is true: $\{s_0, s_1, s_2\}$
- Label states $\{s_0, s_1, s_2\}$ with $\psi_1$
- States where $\text{EX } \psi_1$ is true:
  $S_f = \{s_0, s_1, s_2, s_3\}$
- Since, $s_0 \in S_f$, then $M, s_0 \models \phi$ holds
Example III

- $\mathcal{M}, s_0 \models \phi$, with $\phi = \text{AF } t$?
  - States where $\psi_1 = t$ is true: $\{s_2\}$
  - Label state $\{s_2\}$ with $\psi_1$
  - States where AF $\psi_1$ is true: $S_f = \{s_2\}$
  - Since, $s_0 \not\in S_f$, then $\mathcal{M}, s_0 \not\models \phi$ does not hold
Outline

Temporal Logics

Model Checking
  - What is Model Checking?
  - Model Checking by Explicit State Manipulation
    Symbolic Model Checking

Bounded Model Checking

Unbounded Model Checking
  - Induction
  - Interpolation

Concluding Remarks
Symbolic Model Checking

- Algorithms based on explicit set of states manipulation are still used, e.g. SPIN
- In some contexts, explicit set of states manipulation is unrealistic
  - For a 64-bit program counter, number of possible states is $2^{64}$
- An alternative considers the implicit representation of sets of states
- Original work (from late 80s to late 90s) was based on BDDs
- Most recently SAT essentially replaced BDDs in symbolic model checking
  - Focus on SAT-based symbolic model checking
Modeling Transition Relations

- Define next state equations:
  \[ y_0' = x \land \bar{y}_0 \]
  \[ y_1' = y_0 \]

- Define transition relation as a characteristic function:
  \[ T(s_{PS}, s_{NS}) = (y_0' \leftrightarrow (x \land \bar{y}_0)) \land (y_1' \leftrightarrow y_0) \]

- Can represent \( T(s_{PS}, s_{NS}) \) either with BDDs or with CNF
Outline

Temporal Logics

Model Checking
  What is Model Checking?
  Model Checking by Explicit State Manipulation
  Symbolic Model Checking

Bounded Model Checking

Unbounded Model Checking
  Induction
  Interpolation

Concluding Remarks
Unfolding the Transition Relation

• Initial state predicate: $I(s_0)$
• Transition relation represented by predicate: $T(s_i, s_{i+1})$
• Unfold transition relation for $k$ time frames starting in an initial state:
  $$I(s_0) \land \bigwedge_{i=0}^{k-1} T(s_i, s_{i+1})$$
• Interpretation:
  – Create $k$ copies of transition relation predicate, requiring one of the initial states
Representing Safety Properties

- Safety property $\text{AG} p$
  - To use SAT negate property, i.e. $\text{EF} \neg p$ and check for $\neg p$
  - Define copy of property for each unfolded computation step, i.e. $p_i$
- Initial state predicate: $I(s_0)$
- Transition relation represented by predicate: $T(s_i, s_{i+1})$
- SAT formulation:

$$I(s_0) \land \bigwedge_{i=0}^{k-1} T(s_i, s_{i+1}) \land \bigvee_{i=0}^{i=k} \neg p_i$$
Standard BMC loop

- Start from given unfolding size \( B \)
- While unfolding less than user-specified bound
  - Generate SAT formula for target unfolding size
  - Invoke SAT solver
  - If formula is SAT return counterexample
  - Unfold further

BMC is in general incomplete
- Can find bugs and provide counterexamples
- May be unable to prove that properties hold
- Completeness achieved if length of reachability diameter is known
  - Reachability diameter: longest shortest path in transition system
Define next state equations:

\[ y'_0 = x \land \bar{y}_0 \]
\[ y'_1 = y_0 \]

Define transition relation:

\[ s_i = y_{1,i}y_{0,i} \]
\[ s_{i+1} = y_{1,i+1}y_{0,i+1} \]

\[ T(s_i, s_{i+1}) = (y_{0,i+1} \leftrightarrow (x_i \land \bar{y}_{0,i})) \land (y_{1,i+1} \leftrightarrow y_{0,i}) \]

Can represent \( T(s_i, s_{i+1}) \) in CNF

Initial state predicate:

\[ I(s_0) = (\neg y_{0,0}) \land (\neg y_{1,0}) \]

Example temporal property: \( AG(\neg y_1) \)
An Example II

- Unfolding considered: 3 computation steps
- Easy to represent in CNF

\[
\begin{align*}
y_{0,0} &= 0 \\
y_{1,0} &= 0 = z_0 \\
y_{1,1} &= z_1 \\
y_{1,2} &= z_2 \\
y_{1,3} &= z_3 \\
y_{1,4} &= z_4 \\
\end{align*}
\]

\[
\begin{align*}
x_0 &\quad y_{0,0} \quad y_{0,1} \\
x_1 &\quad y_{0,1} \quad y_{0,2} \\
x_2 &\quad y_{0,2} \quad y_{0,3} \\
\end{align*}
\]

- Property considered \((y_{1,0} \lor y_{1,1} \lor y_{1,2} \lor y_{1,3})\)
- Counterexample: enough to set \(x_0 = 1\)
  - Which sets \(y_{1,2} = 1\)
- Property \(AG(\neg y_1)\) does not hold
• At each step a new copy of the circuit’s CNF formula is used
  – Use incremental SAT solver
    ▶ Need to add clauses representing new circuit copy
    ▶ Need to rearrange property clause

• SAT solvers learn clauses
  – Some learnt clause can be reused in between calls to the SAT solver
Outline

Temporal Logics

Model Checking
  What is Model Checking?
  Model Checking by Explicit State Manipulation
  Symbolic Model Checking

Bounded Model Checking

Unbounded Model Checking
  Induction
  Interpolation

Concluding Remarks
Towards Completeness

- Plain BMC is incomplete
  - Can find bugs and provide counterexamples
  - May be unable to prove that properties hold

- Unbounded model checking (UMC) develops conditions which allow proving that properties hold
  - Can use induction, interpolation, universal quantification with SAT solvers, etc.
Outline

Temporal Logics

Model Checking
  What is Model Checking?
  Model Checking by Explicit State Manipulation
  Symbolic Model Checking

Bounded Model Checking

Unbounded Model Checking
  Induction
  Interpolation

Concluding Remarks
Induction-Based UMC

• Define: \[ \text{path}(s_{[0..n]}) = \bigwedge_{0 \leq i \leq n} T(s_i, s_{i+1}) \]

\[ \text{loopFree}(s_{[0..n]}) = \text{path}(s_{[0..n]}) \land \bigwedge_{0 \leq i < j \leq n} s_i \neq s_j \]

• There is a counterexample if:
  \[ I(s_0) \land \text{path}(s_{[0..i]}) \land \neg p_i \text{ is SAT} \]

• Property is true if for some \( i \), either:
  \[ I(s_0) \land \text{loopFree}(s_{[0..i]}) \text{ is UNSAT, or} \]
  \[ \text{loopFree}(s_{[0..i]}) \land \neg p_i \text{ is UNSAT} \]

• Start from \( i = 0 \) and increase \( i \) until one of the above conditions holds
  – If number of states is finite, algorithm is \textbf{guaranteed} to terminate
Evaluation of Induction-Based UMC

- Number of iterations grows with **longest** simple path between any two states
  - Longest simple path between any two states can be exponentially larger than longest shortest path between those two states (i.e. reachability diameter)
- Alternatives exist where number of iterations grows with the longest **shortest** path between any two states
  - E.g. interpolation
Exercise

- Example circuit:

- Example temporal property: $AG \left( \neg y_1 \right)$
- Apply induction to prove property true
Outline

Temporal Logics

Model Checking
  - What is Model Checking?
  - Model Checking by Explicit State Manipulation
  - Symbolic Model Checking

Bounded Model Checking

Unbounded Model Checking
  - Induction
  - Interpolation

Concluding Remarks
Resolution Proofs

- Resolution operation:
  Given \((x \lor \alpha)\) and \((\neg x \lor \beta)\) derive \((\alpha \lor \beta)\)

- Consider the formula:
  \[\varphi = (\neg c) \land (\neg b) \land (\neg a \lor c) \land (a \lor b) \land (a \lor \neg d) \land (\neg a \lor \neg d)\]

- Resolution proof:

  ![Resolution Proof Diagram]

- A modern SAT solver can generate resolution proofs using clauses learnt by the solver.
An Example

- CNF formula:

\[ \varphi = \omega_1 \land \omega_2 \land \omega_3 \land \omega_4 \land \omega_5 \land \omega_6 \]
\[ = (\neg c) \land (\neg b) \land (\neg a \lor c) \land (a \lor b) \land (a \lor \neg d) \land (\neg a \lor \neg d) \]

Implication graph with conflict
An Example

- CNF formula:

\[ \varphi = \omega_1 \land \omega_2 \land \omega_3 \land \omega_4 \land \omega_5 \land \omega_6 \]
\[ = (\neg c) \land (\neg b) \land (\neg a \lor c) \land (a \lor b) \land (a \lor \neg d) \land (\neg a \lor \neg d) \]

\[ \xrightarrow{\omega_2} b = 0 \quad \xrightarrow{\omega_4} a = 1 \]
\[ \xrightarrow{\omega_1} c = 0 \quad \xrightarrow{\omega_3} \kappa \]

Proof trace \( \bot: \omega_3 \quad \omega_4 \quad \omega_1 \quad \omega_2 \)
An Example

- CNF formula:

\[ \varphi = \omega_1 \land \omega_2 \land \omega_3 \land \omega_4 \land \omega_5 \land \omega_6 = (\neg c) \land (\neg b) \land (\neg a \lor c) \land (a \lor b) \land (a \lor \neg d) \land (\neg a \lor \neg d) \]

Resolution proof follows structure of conflicts
An Example

- CNF formula:

$$\varphi = \omega_1 \land \omega_2 \land \omega_3 \land \omega_4 \land \omega_5 \land \omega_6$$

$$= (\neg c) \land (\neg b) \land (\neg a \lor c) \land (a \lor b) \land (a \lor \neg d) \land (\neg a \lor \neg d)$$

Unsatisfiable core: $\omega_1, \omega_2, \omega_3, \omega_4$
Interpolants

- **Definition of interpolant**
  - Consider two sets of clauses $A$ and $B$
  - Let $A \land B$ be UNSAT
  - There exists interpolant $P$ such that
    1. $A \rightarrow P$
    2. $P \land B$ is UNSAT
    3. $P$ contains only variables common to $A$ and $B$

- **Example:**
  - $A = p \land q$ and $B = \neg q \land r$
  - Interpolant: $P = q$
    - $A \rightarrow P$
    - $P \land B$ is UNSAT
    - $P$ contains only variable $q$, common to $A$ and $B$
Interpolants can be generated from resolution proofs in linear time.

Use SAT solver to generate resolution proof.

Generate interpolant from resolution proof:
- If SAT solver terminates and proves unsatisfiability, then it was able to generate resolution proof.

Creating interpolant from resolution proof: [McMillan’03]
- For root nodes in $A$ keep variables common to $A$ and $B$, or $\bot$ if no common variables exist.
- For root nodes in $B$ use $\top$.
- For resolution nodes:
  - If resolved variable only occurs in $A$, create OR gate.
  - Otherwise, create AND gate.
- See reference for proof.
Example: Computing Interpolant from Resolution Proof

\[ A = (r \lor y) \land (\neg r \lor x) \quad \text{and} \quad B = (\neg y \lor a) \land (\neg y \lor \neg a) \land (\neg x) \]
Abstraction of Reachable States I

- Consider BMC formulation with:
  \[ I(s_0) \land \bigwedge_{i=0}^{k-1} T(s_i, s_{i+1}) \land \left( \bigvee_{i=1}^{i=k} \neg p_i \right) \]

- Let,
  \[ A = I(s_0) \land \bigwedge T(s_0, s_1) \]
  \[ B = \bigwedge_{i=1}^{k-1} T(s_i, s_{i+1}) \land \left( \bigvee_{i=1}^{i=k} \neg p_i \right) \]

- If \( A \land B \) UNSAT, then interpolant \( P \) for \( A \land B \) has the following properties:
  1. \( A \rightarrow P \)
  2. \( P \land B \) is UNSAT
  3. \( P \) described only with variables common to \( A \) and \( B \), i.e. the state variables of \( s_1 \)
Recall that interpolant $P$ for $A \land B$ has the following properties:

1. $A \rightarrow P$
2. $P \land B$ is UNSAT
3. $P$ described only with variables common to $A$ and $B$, i.e. the state variables of $s_1$

$P$ represents an abstraction of the states reachable from $I(s_0)$ in one computation step

- Note that if $A$ holds then $P$ holds, and $P$ described solely with the state variables of $s_1$
Abstraction of Reachable States III

• Let $P_i$ denote the an abstraction of the reachable states in $i$ computation steps

• Let,

$$A = P_i \land \bigwedge_{s_0, s_1} T(s_0, s_1)$$

$$B = \bigwedge_{i=1}^{k-1} T(s_i, s_{i+1}) \land \left( \bigvee_{i=1}^{i=k} \neg p_i \right)$$

• If $A \land B$ UNSAT, then interpolant $P_{i+1}$ for $A \land B$ has the following properties:
  1. $A \rightarrow P_{i+1}$
  2. $P_{i+1} \land B$ is UNSAT
  3. $P_{i+1}$ described only with variables common to $A$ and $B$, i.e. the state variables of $s_1$

• $P_{i+1}$ represents an abstraction of the states reachable from $I(s_0)$ in $i + 1$ computation steps
Interpolant-Based Fixed Point Condition

• If at some point the following condition holds:

$$P_{i+1} \rightarrow I(s_0) \land \bigwedge_{j=1}^{i} P_j$$

- Any state that can be reached in \(i + 1\) computation steps can also be reached in less than \(i + 1\) computation steps
- Hence,
  - Property \(p\) cannot be satisfied
  - \(EF \neg p\) cannot be satisfied, and so
  - \(AG \neg p\) is true

• Procedure is guaranteed to terminate (McMillan’03)
For some unfolding:
  - Either a counterexample is found starting from \(I(s_0)\)
  - Or property is proved
Additional Work

- Use SAT to implement universal quantification
  - Need to enumerate all solutions of SAT formula

- Circuit-based techniques
  - SAT solvers operating on circuits
  - ATPG-based model checking

- Mixed use of SAT and BDDs
  - Proof-based abstraction
  - Counterexample guided abstraction

- Use SMT
  - Richer modeling structures
  - Recent, promising algorithms
Temporal Logics

Model Checking
- What is Model Checking?
- Model Checking by Explicit State Manipulation
- Symbolic Model Checking

Bounded Model Checking

Unbounded Model Checking
- Induction
- Interpolation

Concluding Remarks
Research Directions

- More effective uses of SAT solvers in model checking
  - Additional information from resolution proofs
  - Reuse of learnt clauses

- Alternative fixed point conditions for UMC

- Replace SAT solvers with higher-level decision procedures
  - Satisfiability Modulo Theories (SMT) solvers
    - Can accommodate a vast range of decidable fragments of first-order logic
    - Note: SAT solvers are a key engine in modern SMT solvers
References on Model Checking