Satisfiability-Based Model Checking Algorithms

Joao Marques-Silva

School of Electronics and Computer Science
University of Southampton

RCRA Workshop, July 2007
SAT & Model Checking

- **Context:**
  - Model checking is a ubiquitous system verification technique
  - Explicit state model checking is unrealistic in many settings
  - BDD-based symbolic model checking unable to cope with growing system complexity
  - Performance improvements in SAT solvers since the mid 90s

- **Recent work:**
  - Use of SAT instead of BDDs as core technology for symbolic model checking
    - SAT solvers able to cope better with system complexity
  - Development of SAT-based (symbolic) model checking techniques

- **This talk:**
  - Main ideas and trends in SAT solver technology
  - Approaches for SAT-based model checking
Outline

Boolean Satisfiability

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

Resolution Proofs

Temporal Logic & Model Checking
  Temporal Logic
  Model Checking

Bounded Model Checking

Unbounded Model Checking
  Induction
  Interpolation

Some Research Topics
Boolean Satisfiability

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

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

- SAT is an NP-complete decision problem
  - SAT was the first problem to be shown NP-complete
  - There are no known polynomial time algorithms for SAT
  - Widely accepted conjecture:
    Any algorithm that solves SAT is exponential in the number of variables, in the worst-case

Representing Boolean Circuits / Formulas

• Satisfiability problems can be defined on Boolean circuits/formulas
• Can represent circuits/formulas as CNF formulas
  - 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

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

\begin{tabular}{|c|c|c|c|}
\hline
a & b & c & \varphi_c(a,b,c) \\
\hline
0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 \\
0 & 1 & 0 & 0 \\
0 & 1 & 1 & 1 \\
1 & 0 & 0 & 0 \\
1 & 0 & 1 & 1 \\
1 & 1 & 0 & 1 \\
1 & 1 & 1 & 0 \\
\hline
\end{tabular}
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: \( z = d \lor (c \land (\neg (a \land b))) \)
  - No distinction between Boolean circuits and formulas
Outline

Boolean Satisfiability

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

Resolution Proofs

Temporal Logic & Model Checking
  Temporal Logic
  Model Checking

Bounded Model Checking

Unbounded Model Checking
  Induction
  Interpolation

Some Research Topics
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)
    - Resolution
    - Stalmarck’s method
    - Recursive learning
    - Cutting planes
    - ...
  - Binary Decision Diagrams (BDDs)
  - Backtrack search / DPLL
    - Conflict-Driven Clause Learning (CDCL)
  - ...

- For model checking: only CDCL solvers are of interest!
Outline

Boolean Satisfiability

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

Resolution Proofs

Temporal Logic & Model Checking
  Temporal Logic
  Model Checking

Bounded Model Checking

Unbounded Model Checking
  Induction
  Interpolation

Some Research Topics
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
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

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

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

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

• Resolution forms the basis of a complete algorithm for SAT
  - Iteratively apply the following steps:
    1. Select variable \(x\)
    2. Apply resolution rule between every pair of clauses of the form \((x \lor \alpha)\) and \((\neg x \lor \beta)\)
    3. Remove all clauses containing either \(x\) or \(\neg x\)
    4. Apply the pure literal rule and unit propagation
  - Terminate when either the empty clause or the empty formula is derived
Outline

Boolean Satisfiability

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

Resolution Proofs

Temporal Logic & Model Checking
  Temporal Logic
  Model Checking

Bounded Model Checking

Unbounded Model Checking
  Induction
  Interpolation

Some Research Topics
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

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

Boolean Satisfiability

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

Resolution Proofs

Temporal Logic & Model Checking
  Temporal Logic
  Model Checking

Bounded Model Checking

Unbounded Model Checking
  Induction
  Interpolation

Some Research Topics
CDCL SAT Solvers

- Inspired on DPLL
  - Must be able to prove unsatisfiability
- New clauses learnt from conflicts [Marques-Silva et al.’96]
  - Based on information provided by unit propagation
- Backtracking can be non-chronological [Marques-Silva et al.’96]
- Structure of conflicts exploited (UIPs) [Marques-Silva et al.’96]
- Efficient data structures [Moskewicz et al.’01]
  - Compact and reduced maintenance overhead
- Backtrack search is periodically restart [Gomes et al.’98]

- 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 learnt from conflicts
  - Based on information provided by unit propagation
- Backtracking can be non-chronological
- Structure of conflicts exploited (UIPs)
- Efficient data structures
  - Compact and reduced maintenance overhead
- Backtrack search is periodically restarted

- 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 \((a \lor c \lor f)\)
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 \((a \lor c \lor f)\)

- **Traced clauses:** Clauses involved in unit propagation resulting in conflict
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 $\Rightarrow$ 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 $a = 1$
- A conflict is again reached: $(\neg d \lor \neg e \lor f)$ is unsatisfied
Non-Chronological Backtracking

- During backtrack search, for each conflict backtrace 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

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

- 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
- \((c = 0) \land (f = 0) \Rightarrow (\varphi = 0)\)
- \((\varphi = 1) \Rightarrow (c = 1) \lor (f = 1)\)

- Learn new clause \((c \lor f)\)

- **Traced clauses:** Clauses involved in unit propagation resulting in conflict
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
Additional/Recent Topics

- Always backtrack given learnt clause
  [Moskewicz et al.'01]
- Conflict-guided branching heuristics (e.g. VSIDS, etc.)
  [ditto]
- Clause deletion policies
  [Goldberg et al.'02]
- Eliminate top-level assignments in learnt clauses
  [e.g. Een et al.'03]
- Formula simplification
  [e.g. Een et al.'05]
## 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.
- Steady evolution, but no major improvements in recent years.
Outline

Boolean Satisfiability

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

Resolution Proofs

Temporal Logic & Model Checking
  Temporal Logic
  Model Checking

Bounded Model Checking

Unbounded Model Checking
  Induction
  Interpolation

Some Research Topics
Example Resolution Proof

• Recall resolution operation:
  Given \((x \lor \alpha)\) and \((\neg x \lor \beta)\) derive \((\alpha \lor \beta)\)

• Consider the formula:
  \(\varphi = (a \lor b) \land (\neg a \lor c) \land (\neg b) \land (\neg c)\)

• Resolution proof:
Generating Resolution Proofs

• Modern SAT solvers learn clauses
• For unsat instances, learnt clauses encode resolution refutation

• Steps for generating resolution refutation with CDCL SAT solver:
  – Proof tracing: Associate with each learnt clause, all traced clauses (see clause learning)
  – Resolution proof: Traverse proof trace from empty clause, and identify resolution steps
An Example

- Formula:
  \( \varphi = \omega_1 \land \omega_2 \land \omega_3 \land \omega_4 = (a \lor b) \land (\neg a \lor c) \land (\neg b) \land (\neg c) \)

- Implications:
  \( c = 0, b = 0, a = 0, \kappa \equiv \text{conflict} \)

- Traced clauses for learning \( \bot \):
  \( \bot : \omega_1, \omega_2, \omega_3, \omega_4 \)

- Resolution proof:

  \[
  \begin{array}{c}
  (a \lor b) \\
  (\neg a \lor c) \\
  (\neg b) \\
  (b \lor c) \\
  (\neg c) \\
  (c) \\
  \bot
  \end{array}
  \]

- Resolution proof is linear in number of learnt clauses
Outline

Boolean Satisfiability

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

Resolution Proofs

Temporal Logic & Model Checking
  Temporal Logic
  Model Checking

Bounded Model Checking

Unbounded Model Checking
  Induction
  Interpolation

Some Research Topics
Outline

Boolean Satisfiability

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

Resolution Proofs

Temporal Logic & Model Checking
  Temporal Logic
  Model Checking

Bounded Model Checking

Unbounded Model Checking
  Induction
  Interpolation

Some Research Topics
Computation-Tree Logic (CTL)

- Syntax of CTL formulas:

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

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

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

- Focus on safety properties: \( AG p \)
  - In all computation paths it is globally true that nothing bad happens (i.e. \( p = \neg b \))
Transition Systems

- A transition system (or model) $\mathcal{M} = (S, \rightarrow, L)$ 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:
    $\{s_0 \rightarrow s_1, s_1 \rightarrow s_2, s_0 \rightarrow s_2, s_1 \rightarrow s_2, s_2 \rightarrow s_2\}$
  - 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 $s_i$

```
\[ s_0: p, q \quad s_1: q, r \quad s_2: r \]
```

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

• 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 $\mathcal{M}, s \models \phi_2$
  
  – ... 

  – $\mathcal{M}, s \models AG \phi$ 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 All computation paths beginning in $s$, $\phi$ holds Globally

  – $\mathcal{M}, s \models EF \phi$ 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

  – ...
Outline

Boolean Satisfiability

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

Resolution Proofs

Temporal Logic & Model Checking
   Temporal Logic
   Model Checking

Bounded Model Checking

Unbounded Model Checking
   Induction
   Interpolation

Some Research Topics
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 state $s$

• Approaches to model checking:
  – **Explicit** manipulation of system’s states
  – **Implicit** manipulation of system’s states – **symbolic** model checking
    ▶ Explicit state manipulation may be unfeasible
    ▶ Original work: based on BDDs (late 80s to late 90s)
    ▶ Recent work: SAT replaced BDDs
      Bounded Model Checking (BMC)
      Unbounded Model Checking (UMC)
      [Biere et al.'99]]
      [Sheeran et al.'00;McMillan'03]
Define next state equations:

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

Define transition relation as a characteristic function:

\[ s_{PS} = y_1y_0 \]
\[ s_{NS} = y'_1 y'_0 \]

\[ T(s_{PS}, s_{NS}) = (y'_0 \leftrightarrow (x \land \bar{y}_0)) \land (y'_1 \leftrightarrow y_0) \]

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

Boolean Satisfiability

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

Resolution Proofs

Temporal Logic & Model Checking
  Temporal Logic
  Model Checking

Bounded Model Checking

Unbounded Model Checking
  Induction
  Interpolation

Some Research Topics
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 \( k \) computation steps 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 $AG \ p$
  - To use SAT negate property, i.e. $EF \ \neg p$ and check for $\neg p$
  - Define copy of negated property for each unfolded computation step, i.e. $\neg p_i$

- Initial state predicate: $I(s_0)$

- Transition relation represented by predicate: $T(s_i, s_{i+1})$

- SAT formulation:

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

- If for some $k$, formula is satisfiable, then:
  - There exists state reachable from initial state where $\neg p$ holds
  - $EF \ \neg p$ holds
  - $AG \ p$ does not hold
  - Solution for SAT instance represents a counterexample
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
  - Otherwise unfold further

- BMC is in general incomplete
  - Can find bugs and provide counterexamples
  - May be unable to prove that properties hold
  - Completeness achieved if 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

\[ y_{0,0} = 0 \]
\[ y_{1,0} = 0 = z_0 \]
\[ y_{1,1} = z_1 \]
\[ y_{1,2} = z_2 \]
\[ y_{1,3} = z_3 \]
\[ y_{1,4} = z_4 \]

- 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 \(y_{1,1} = 1\)
- Property \(AG(\neg y_1)\) does not hold
BMC in Practice – Representing the Transition Relation

- Transition relation most often represented as Boolean circuit
- Identify **cone of influence** given property
  - Discard other parts of circuit
- Standard circuit representation typically used:
  - Boolean Expression Diagrams (BEDs)
  - Reduced Boolean Circuits (RBCs)
  - And-Inverter Graphs (AIGs)

- Optimizations can be applied to representation of transition relation
- Can provide structural information to the SAT solver

[Andersen'97] [Abdulla et al.'00] [e.g. Ganai et al.'01]
BMC in Practice – Interacting with SAT Solvers

- At each step a new copy of the CNF formula of the transition relation is used
  - Use incremental SAT solver [e.g. Een et al.'03]
    - Need to add clauses representing new transition relation copy
    - Need to rearrange property clause

- Dedicated branching heuristics [Strichman'00]

- Some learnt clauses can be reused in between calls to the SAT solver [Marques-Silva et al.'97; Strichman'01]

- Dedicated solvers, e.g. non-clausal solvers [Iyer et al.'03]
Reusing Learnt Clauses

- Original clauses split into classes of clauses: \( \{C_1, C_2, \ldots\} \)

- Use information from conflict analysis to decide clauses to reuse
  - When analyzing conflict, record classes of traced clauses
    - E.g. \( NC = \{C_{i1}, C_{i2}, \ldots\} \)
  - Associate learnt clause \( \omega \) with classes in \( NC \)
  - If next iteration of BMC loop reuses all classes of clauses in \( NC \), then \( \omega \) can be reused

- In practice number of classes is small
  - E.g. one class for all clauses but property clause, and another class for property clause
Outline

Boolean Satisfiability

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

Resolution Proofs

Temporal Logic & Model Checking
  Temporal Logic
  Model Checking

Bounded Model Checking

Unbounded Model Checking
  Induction
  Interpolation

Some Research Topics
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
  - **Induction** [Sheeran et al.'00]
  - Use SAT to implement universal quantification [e.g. McMillan'02]
    - Need to enumerate all solutions of SAT formula
  - Mixed use of SAT and BDDs
    - Counterexample guided abstraction [Chauhan et al.'02]
    - Proof-based abstraction [McMillan et al.'03]
  - **Interpolation** [McMillan'03]
  - Circuit-based techniques
    - ATPG-based model checking [Iyer et al.'04]
    - SAT solvers operating on circuits [Somenzi et al.'04]
  - ...
Outline

Boolean Satisfiability

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

Resolution Proofs

Temporal Logic & Model Checking
  Temporal Logic
  Model Checking

Bounded Model Checking

Unbounded Model Checking
  Induction
  Interpolation

Some Research Topics
UMC by \( k \)-Induction

- Define:

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

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

- Start from \( i = 0 \) and increase \( i \) until one of the termination conditions (defined next) holds
  - If number of states is finite, algorithm is \textit{guaranteed} to terminate
Termination Condition for Counterexample

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

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

- There is a counterexample if:
  1. \( I(s_0) \land path(s_{[0..i]}) \land \neg p_i \) is SAT
    - If there exists state \((s_i)\), reachable from an initial state \((s_0)\), where \(\neg p\) holds, then there is a counterexample.
**Termination Condition for Proving Property**

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

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

- **EF \( \neg p \) cannot** be satisfied (i.e. AG \( p \) holds) if for some \( i \), either:
  1. \( I(s_0) \land \text{loopFree}(s_{0..i}) \) is UNSAT
     - There are no loop-free paths of length \( i \) reachable from initial state
  
  or,

  2. \( \text{loopFree}(s_{0..i}) \land \neg p_i \) is UNSAT
     - There are no loop-free paths of length \( i \) that reach a state where the target property fails

∴ There are no states reachable from initial state where \( \neg p \) holds
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
Outline

Boolean Satisfiability

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

Resolution Proofs

Temporal Logic & Model Checking
  Temporal Logic
  Model Checking

Bounded Model Checking

Unbounded Model Checking
  Induction
  Interpolation

Some Research Topics
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 $B$
Computing Interpolants from Resolution Proofs

- Interpolants can be generated from resolution proofs in linear time
  [Pudlák’97]

- 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

\[ A = (r \lor y) \land (\neg r \lor x) \text{ and } B = (\neg y \lor a) \land (\neg y \lor \neg a) \land (\neg x) \]
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 \( s_1 \)
Abstraction of Reachable States II

• 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 $s_1$

Note: $P$ described solely with the state variables of $s_1$, and if $A$ holds then $P$ holds

\[ \forall_{s_0,s_1} I(s_0) \land T(s_0,s_1) \rightarrow P(s_1) \]

• $P$ represents an abstraction of the states reachable from $I(s_0)$ in one computation step
Abstraction of Reachable States III

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

• Let,

$$
A = P_i \land \bigwedge_{i=1}^{k-1} T(s_i, s_{i+1})
$$

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

• 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” in the abstraction of the states reachable in \( i + 1 \) computation steps already represented in one of the abstractions of states reachable in less than \( i + 1 \) computation steps
- Hence,
  - Goal \( \neg p \) cannot be satisfied
  - EF \( \neg p \) cannot be satisfied, and so
  - AG \( p \) is true

- Procedure is guaranteed to terminate

For some unfolding:
- Either a counterexample is found starting from \( I(s_0) \)
- Or property is proved

[McMillan’03]
Outline

Boolean Satisfiability

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

Resolution Proofs

Temporal Logic & Model Checking
  Temporal Logic
  Model Checking

Bounded Model Checking

Unbounded Model Checking
  Induction
  Interpolation

Some Research Topics
Research Directions

• Alternative fixed point conditions for UMC

• Extensive evaluation of SAT-based UMC
  – Interpolation among the most promising, but thorough independent evaluation needs to be conducted

• 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