SAT/SMT Solvers
and
Applications

Vijay Ganesh
University of Waterloo
Winter 2013
Lessons learnt so far

- Implementation-related attacks (control-hazard, malware,...)
- Program analysis techniques can detect them, prevent them
- How do you build such program analysis-based detectors and bug-finders
- Often, using constraint solvers (aka SAT/SMT solvers)

Constraint solvers

- What are constraint solvers?
- Why you should care?
- Security and software engineering applications
What is a Constraint Solver?

**Engineer/Mathematician’s point of view**

- A “method” that takes as input a math formula, and produces a solution
- Examples: solving linear equations over the reals, polynomials, quadratic, Boolean logic,...
- Computing zeros of a polynomial

**Theoretical computer scientist/logician’s point of view**

- A computer program called as a satisfiability procedure that solves a specific kind of decision problem, namely, the SAT problem
- The input formula is in a specified logic (e.g., Boolean, first-order, reals, integers,...)
- Output of a satisfiability procedure
  - UNSAT, if input has no satisfying assignments
  - SAT, otherwise
Before modern conception of logic (Before Boole and Frege)

- From Babylon to late 1800’s: Huge amount of work on methods to solve (find roots of) polynomials over reals, integers,...
- System of linear equations over the reals (Chinese methods, Cramer’s method, Gauss elimination)
- These methods were typically not complete (e.g., worked for a special class of polynomials)

After modern conception of logic

- Systems of linear inequalities over the integers are solvable (Presburger, 1927)
- Peano arithmetic is undecidable (hence, not solvable) (Godel, 1931)
- First-order logic is undecidable (hence, not solvable) (Turing, 1936. Church, 1937)
- A exponential-time algorithm for Boolean SAT problem (Davis, Putnam, Loveland, Loggeman in 1962)
- Systems of Diophantine equations are not solvable (Matiyasevich. 1970)
- Boolean SAT problem is NP-complete (Cook 1971)
- Many efficient, scalable SAT procedures since 1971 for a variety of mathematical theories
Foundation of Software Engineering
Logic Abstractions of Computation

Formal Methods
Program Analysis

Program Reasoning

Automatic Testing
Program Synthesis
Foundation of Software Engineering

Logic Abstractions of Computation

Formal Methods
Program Analysis

Logics (Boolean,...)

Automatic Testing
Program Synthesis

Tuesday, 29 January, 13
Software Engineering & SAT/SMT Solvers
An Indispensable Tactic for Any Strategy

Formal Methods
Program Analysis

SE Goal: Reliable/Secure Software

Automatic Testing
Program Synthesis
Software Engineering & SAT/SMT Solvers
An Indispensable Tactic for Any Strategy

SE Goal: Reliable/Secure Software

SAT/SMT Solvers

Formal Methods
Program Analysis

Automatic Testing
Program Synthesis
Software Engineering using Solvers

Engineering, Usability, Novelty

Program Specification

Program Reasoning Tool

Logic Formulas

SAT/SMT Solver

SAT/UNSAT

Program is Correct?
or Generate Counterexamples (Test cases)
SAT/SMT Solver Research Story
A 1000x Improvement

- Solver-based programming languages
- Compiler optimizations using solvers
- Solver-based debuggers
- Solver-based type systems
- Solver-based concurrency bug finding
- Solver-based synthesis
- Bio & Optimization

- Concolic Testing
- Program Analysis
- Equivalence Checking
- Auto Configuration

- Bounded MC
- Program Analysis
- AI

1,000,000 Constraints
100,000 Constraints
10,000 Constraints
1,000 Constraints

The SAT/SMT Problem

- Rich logics (Modular arithmetic, Arrays, Strings,...)
- NP-complete, PSPACE-complete,...
- Practical, scalable, usable, automatic
- Enable novel software reliability approaches
The SAT/SMT Problem

- Closely related to the Validity Problem
- Soundness, completeness, termination
- Connecting model theory and proof theory

Logic

Formula

Solver

(q ∨ p ∨ ¬r)
(q ∨ ¬p ∨ r)
...

SAT

UNSAT
Lecture Outline

Points already covered

- Motivation for SAT/SMT solvers in software engineering
- High-level description of the SAT/SMT problem & logics
- Defined logic, models, truth, proofs, SAT procedure, soundness, completeness

Rest of the lecture

- Modern CDCL SAT solver architecture & techniques
- SAT/SMT-based applications
- Future of SAT/SMT solvers
- Some history (who, when,...) and references sprinkled throughout the talk
- Non-CDCL SAT techniques
int double (int v) {
    return 2*v;
}

void testme (int x, int y) {
    z = double (y);
    if (z == x) {
        if (x > y+10) {
            ERROR;
        }
    }
}
Concolic Testing: Example

```c
int double (int v) {
    return 2*v;
}

void testme (int x, int y) {
    int z = double (y);
    if (z == x) {
        if (x > y+10) {
            ERROR;
        }
    }
}
```
Concolic Testing Approach

int double (int v) {
    return 2*v;
}

void testme (int x, int y) {
    z = double (y);
    if (z == x) {
        if (x > y + 10) {
            ERROR;
        }
    }
}

<table>
<thead>
<tr>
<th>Concrete Execution</th>
<th>Symbolic Execution</th>
<th>Path Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>int double (int v)</td>
<td>return 2*v;</td>
<td></td>
</tr>
<tr>
<td>z = double (y);</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ERROR;</td>
<td>x = x, y = y_0</td>
<td></td>
</tr>
<tr>
<td>x = 22, y = 7</td>
<td>x = x_0, y = y_0</td>
<td></td>
</tr>
<tr>
<td>Tuesday, 29 January, 13</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Concolic Testing Approach

int double (int v) {
    return 2*v;
}

void testme (int x, int y) {
    z = double (y);
    if (z == x) {
        if (x > y + 10) {
            ERROR;
        }
    }
}

<table>
<thead>
<tr>
<th>Concrete Execution</th>
<th>Symbolic Execution</th>
<th>Path Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>concrete state</td>
<td>symbolic state</td>
<td>path condition</td>
</tr>
<tr>
<td>x = 22, y = 7, z = 14</td>
<td>x = x₀, y = y₀, z = 2*y₀</td>
<td></td>
</tr>
</tbody>
</table>

Concrete Execution:
- `int double (int v) {
    return 2*v;
}`

Symbolic Execution:
- `void testme (int x, int y) {
    z = double (y);
    if (z == x) {
        if (x > y + 10) {
            ERROR;
        }
    }
}`

Path Condition:
- `x = x₀, y = y₀, z = 2*y₀`
```c
int double (int v) {
    return 2*v;
}

void testme (int x, int y) {
    z = double (y);
    if (z == x) {
        if (x > y + 10) {
            ERROR;
        }
    }
}
```

**Concrete Execution**

- \( x = 22, y = 7, z = 14 \)

**Symbolic Execution**

- \( x = x_0, y = y_0, z = 2y_0 \)

- \( 2y_0 \neq x_0 \)
Concolic Testing Approach

```c
int double (int v) {
    return 2*v;
}

void testme (int x, int y) {
    z = double (y);
    if (z == x) {
        if (x > y+10) {
            ERROR;
        }
    }
}
```

Concrete Execution

Symbolic Execution

<table>
<thead>
<tr>
<th>concrete state</th>
<th>symbolic state</th>
<th>path condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solve: $2y_0 == x_0$</td>
<td>Solution: $x_0 = 2$, $y_0 = 1$</td>
<td>$2y_0 != x_0$</td>
</tr>
</tbody>
</table>

| $x = 22$, $y = 7$, $z = 14$ | $x = x_0$, $y = y_0$, $z = 2y_0$ |

Tuesday, 29 January, 13
Concolic Testing Approach: Use-case for SAT

```
int double (int v) {
    return 2*v;
}

void testme (int x, int y) {
    z = double (y);
    if (z == x) {
        if (x > y+10) {
            ERROR;
        }
    }
}
```

Concrete Execution

```
Solve: 2*y₀ == x₀
Solution: x₀ = 2, y₀ = 1
```

Symbolic Execution

```
2*y₀ != x₀
```

Concrete State: x = 22, y = 7, z = 14
Symbolic State: x = x₀, y = y₀, z = 2*y₀
Path Condition: 2*y₀ != x₀

Tuesday, 29 January, 13
Concolic Testing Approach: Use-case for SAT

```c
int double (int v) {
    return 2*v;
}

void testme (int x, int y) {
    z = double (y);
    if (z == x) {
        if (x > y + 10) {
            ERROR;
        }
    }
}
```

Concrete Execution:
- **Concrete state**: Solve: $2y_0 == x_0$
- **Symbolic state**: Solution: $x_0 = 2, y_0 = 1$
- **Path condition**: $2y_0 != x_0$

Observe that digital programs with finite loops are Boolean logic circuits.

*Example values*
- $x = 22, y = 7, z = 14$
- $x = x_0, y = y_0, z = 2y_0$
The Boolean SAT Problem

Basic Definitions and Format

A literal \( p \) is a Boolean variable \( x \) or its negation \( \neg x \).

A clause \( C \) is a disjunction of literals: \( x_2 \lor \neg x_{41} \lor x_{15} \)

A CNF is a conjunction of clauses: \( (x_2 \lor \neg x_1 \lor x_5) \land (x_6 \lor \neg x_2) \land (x_3 \lor \neg x_4 \lor \neg x_6) \)

All Boolean formulas assumed to be in CNF

Assignment is a mapping (binding) from variables to Boolean values (True, False).

A unit clause \( C \) is a clause with a single unbound literal

The SAT-problem is:

Find an assignment s.t. each input clause has a true literal (aka input formula has a solution or is SAT)

OR establish input formula has no solution (aka input formula is UNSAT)

The Input formula is represented in DIMACS Format:

```
c DIMACS
p cnf 6 3
2 -1 5 0
6 -2 0
3 -4 -6 0
```
DPLL SAT Solver Architecture

The Basic Solver

DPLL(\(\Theta_{\text{cnf}}, \text{assign}\)) {
    Propagate unit clauses;
    if ”conflict”: return FALSE;
    if ”complete assign”: return TRUE;
    ”pick decision variable x”;
    return DPLL(\(\Theta_{\text{cnf}} \land x=0\), assign[x=0]) || DPLL(\(\Theta_{\text{cnf}} \land x=1\), assign[x=1]);
}

- Propagate (Boolean Constant Propagation):
  - Propagate inferences due to unit clauses
  - Most time in solving goes into this
- Detect Conflict:
  - Conflict: partial assignment is not satisfying
- Decide (Branch):
  - Choose a variable & assign some value
- Backtracking:
  - Implicitly done by the recursion
Modern CDCL SAT Solver Architecture

Key Steps and Data-structures

Key steps

- Decide()
- Propagate() (BCP: Boolean constraint propagation)
- Conflict analysis and learning()
- Backjump()
- Forget()
- Restart()

CDCL: Conflict-Driven Clause-Learning

- Conflict analysis is a key step
- Results in learning a conflict clause
- Prunes the search space

Key data-structures (State):

- Stack or trail of partial assignments (AT)
- Input clause database
- Conflict clause database
- Conflict graph
- Decision level (DL) of a variable
Modern CDCL SAT Solver Architecture

Propagate(), Decide(), Analyze/Learn(), BackJump()

- **Propagate (Boolean Constant Propagation):**
  - Propagate inferences due to unit clauses
  - Most time in solving goes into this

- **No Conflict?**

- **All Vars Assigned?**
  - Conflict Analysis()
  - Analyze the reason (learn conflict clause)
  - Conflict clause blocks the non-satisfying & a large set of other 'no-good' assignments
  - Marques-Silva & Sakallah (1999)

- **TopLevel Conflict?**

- **Return SAT**

- **Return UNSAT**

- **Decide()**
  - Each decision is a decision level
  - Imposes dynamic variable order
  - Decision Level (DL): variable natural number

- **BackJump()**
  - Undo the decision(s) that caused no-good assignment
  - Assign 'decision variables' different values
  - Go back several decision levels
  - Marques-Silva & Sakallah (1999)
  - Backtrack (Davis, Putnam, Loveland, Logemann 1962)
Modern CDCL SAT Solver Architecture

Propagate(), Decide(), Analyze/Learn(), BackJump()

Input SAT Instance

Propagate() (BCP)

No Conflict?

All Vars Assigned?

Conflict Analysis()

Return SAT

Decide()

TopLevel Conflict?

Return UNSAT

BackJump()

- Propagate (Boolean Constant Propagation):
  - Propagate inferences due to unit clauses
  - Most time in solving goes into this

Marques-Silva & Sakallah (1999)

BackTrack (Davis, Putnam, Loveland, Logemann 1962)
Modern CDCL SAT Solver Architecture

Propagate(), Decide(), Analyze/Learn(), BackJump()

- Propagate (Boolean Constant Propagation):
  - Propagate inferences due to unit clauses
  - Most time in solving goes into this

- Detect Conflict?
  - Conflict: partial assignment is not satisfying

- Decide (Branch):
  - Choose a variable & assign some value (decision)
  - Basic mechanism to do search
  - Imposes dynamic variable order

- Conflict analysis and clause learning:
  - Analyze the reason (learn conflict clause)
  - Conflict clause blocks the non-satisfying & a large set of other 'no-good' assignments

- BackJump:
  - Undo the decision(s) that caused no-good assignment
  - Assign 'decision variables' different values
  - Go back several decision levels

- Backtrack (Davis, Putnam, Loveland, Logemann 1962)
Modern CDCL SAT Solver Architecture

Propagate(), Decide(), Analyze/Learn(), BackJump()

Input SAT Instance

Propagate() (BCP)

No Conflict?

Conflict Analysis()

Conflict Clause?

All Vars Assigned?

Return SAT

Decide()

TopLevel Conflict?

Return UNSAT

BackJump()

• Propagate (Boolean Constant Propagation):
  • Propagate inferences due to unit clauses
  • Most time in solving goes into this

• Detect Conflict?
  • Conflict: partial assignment is not satisfying
Modern CDCL SAT Solver Architecture

Propagate(), Decide(), Analyze/Learn(), BackJump()

Input SAT Instance

- **Propagate (Boolean Constant Propagation):**
  - Propagate inferences due to unit clauses
  - Most time in solving goes into this

- **Detect Conflict?**
  - Conflict: partial assignment is not satisfying

- **Decide (Branch):**
  - Choose a variable & assign some value (*decision*)
  - Basic mechanism to do search
  - Imposes dynamic variable order
  - Decision Level (DL): variable ⇒ natural number

- **Conflict Analysis:**
  - Analyze the reason (learn conflict clause)
  - Conflict clause blocks the non-satisfying & a large set of other 'no-good' assignments

- **BackJump:**
  - Undo the decision(s) that caused no-good assignment
  - Assign 'decision variables' different values
  - Go back several decision levels

Marques-Silva & Sakallah (1999)

Backtrack (Davis, Putnam, Loveland, Logemann 1962)
Modern CDCL SAT Solver Architecture

Propagate(), Decide(), Analyze/Learn(), BackJump()

Input SAT Instance

Propagate() (BCP)

No Conflict?

All Vars Assigned?

Conflict Analysis()

Return SAT

Decide()

TopLevel Conflict?

Return UNSAT

BackJump()

• Propagate (Boolean Constant Propagation):
  • Propagate inferences due to unit clauses
  • Most time in solving goes into this

• Detect Conflict?
  • Conflict: partial assignment is not satisfying

• Decide (Branch):
  • Choose a variable & assign some value (decision)
  • Basic mechanism to do search
  • Imposes dynamic variable order
  • Decision Level (DL): variable ⇒ natural number

Marques-Silva & Sakallah (1999)

Marques-Silva & Sakallah (1999)

Backtrack (Davis, Putnam, Loveland, Logemann 1962)
Modern CDCL SAT Solver Architecture

Input SAT Instance

Propagate():
- Propagate inferences due to unit clauses
- Most time in solving goes into this

Detect Conflict?:
- Conflict: partial assignment is not satisfying

Decide (Branch):
- Choose a variable & assign some value (decision)
- Each decision is a decision level
- Imposes dynamic variable order
- Decision Level (DL): variable \( \Rightarrow \) natural number

Conflict analysis and clause learning:
- Compute assignments that lead to conflict (analysis)
- Construct conflict clause blocks the non-satisfying & a large set of other ‘no-good’ assignments (learning)
- Marques-Silva & Sakallah (1996)
Modern CDCL SAT Solver Architecture

Propagate(), Decide(), Analyze/Learn(), BackJump()

Input SAT Instance

- Propagate:
  - Propagate inferences due to unit clauses
  - Most time in solving goes into this

- Detect Conflict?
  - Conflict: partial assignment is not satisfying

- Decide (Branch):
  - Choose a variable & assign some value (decision)
  - Each decision is a decision level
  - Imposes dynamic variable order
  - Decision Level (DL): variable ⇒ natural number

- Conflict analysis and clause learning:
  - Compute assignments that lead to conflict (analysis)
  - Construct conflict clause blocks the non-satisfying & a large set of other ‘no-good’ assignments (learning)
  - Marques-Silva & Sakallah (1996)
Modern CDCL SAT Solver Architecture

Propagate(), Decide(), Analyze/Learn(), BackJump()

Input SAT Instance

Propagate()
(BCP)

No Conflict?

All Vars Assigned?

Conflict Analysis()

Return SAT

Decide()

TopLevel Conflict?

Return UNSAT

BackJump()

- **Propagate:**
  - Propagate inferences due to unit clauses
  - Most time in solving goes into this

- **Detect Conflict?**
  - Conflict: partial assignment is not satisfying

- **Decide:**
  - Choose a variable & assign some value (decision)
  - Each decision is a decision level
  - Imposes dynamic variable order
  - Decision Level (DL): variable → natural number

- **Conflict analysis and clause learning:**
  - Compute assignments that lead to conflict (analysis)
  - Construct conflict clause blocks the non-satisfying & a large set of other ‘no-good’ assignments (learning)
  - Marques-Silva & Sakallah (1996)

- **Conflict-driven BackJump:**
  - Undo the decision(s) that caused no-good assignment
  - Assign ‘decision variables’ different values
  - Go back several decision levels
  - Backjump: Marques-Silva, Sakallah (1999)
  - Backtrack: Davis, Putnam, Loveland, Logemann (1962)
Modern CDCL SAT Solver Architecture

Propagate(), Decide(), Analyze/Learn(), BackJump()

- **Propagate:**
  - Propagate inferences due to unit clauses
  - Most time in solving goes into this

- **Detect Conflict?**
  - Conflict: partial assignment is not satisfying

- **Decide:**
  - Choose a variable & assign some value (decision)
  - Each decision is a decision level
  - Imposes dynamic variable order
  - Decision Level (DL): variable \( \Rightarrow \) natural number

- **Conflict analysis and clause learning:**
  - Compute assignments that lead to conflict (analysis)
  - Construct conflict clause blocks the non-satisfying & a large set of other ‘no-good’ assignments (learning)
  - Marques-Silva & Sakallah (1996)

- **Conflict-driven BackJump:**
  - Undo the decision(s) that caused no-good assignment
  - Assign ‘decision variables’ different values
  - Go back several decision levels
  - Backjump: Marques-Silva, Sakallah (1999)
  - Backtrack: Davis, Putnam, Loveland, Logemann (1962)
Modern CDCL SAT Solver Architecture

Propagate(), Decide(), Analyze/Learn(), BackJump()

Input SAT Instance

Propagate()
(BCP)

No Conflict?

All Vars Assigned?

Return SAT

Conflict Analysis()

Decide()

TopLevel Conflict?

Return UNSAT

BackJump()

{3, 6, -7, 8}
{1, 4, 7}
{-8, 4}
{-1, -3, 8}
{-3, -4, -8}
{-1, -2, 3, 4, -6}

{3, 6, -7, 8}
{1, 4, 7}
{-8, 4}
{-1, -3, 8}
{-3, -4, -8}
{-1, -2, 3, 4, -6}

{3, 6, -7, 8}
{1, 4, 7}
{-8, 4}
{-1, -3, 8}
{-3, -4, -8}
{-1, -2, 3, 4, -6}

{3, 6, -7, 8}
{1, 4, 7}
{-8, 4}
{-1, -3, 8}
{-3, -4, -8}
{-1, -2, 3, 4, -6}

CONFLICT!
(Trigger to analyze & backjump)

Unit clause
(BCP)

...
Modern CDCL SAT Solver Architecture

Decide() Details: VSIDS Heuristic

• Decide() or Branching():
  • Choose a variable & assign some value (decision)
  • Imposes dynamic variable order (Malik et al. 2001)

• How to choose a variable:
  • VSIDS heuristics
  • Each variable has an activity
  • Activity is bumped additively, if variable occurs in conflict clause
  • Activity of all variables is decayed by multiplying by const < 1
  • Next decision variable is the variable with highest activity
  • Over time, truly important variables get high activity
  • This is pure magic, and seems to work for many problems
Modern CDCL SAT Solver Architecture

Propagate() Details: Two-watched Literal Scheme

Input SAT Instance

Propagate() (BCP)

No Conflict?

All Vars Assigned?

Conflict Analysis()

Return SAT

Decide()

TopLevel Conflict?

Return UNSAT

BackJump()

Watched Literal | Watcher List
--- | ---
-1 | {-1,-3,8},...
-3 | {-1,-3,8},...
... | ...

Watched Literal | Watcher List
--- | ---
-1 | ...
-3 | ...
8 | {-1,-3,8},...
... | ...

The constraint propagates 8

{3, 6, -7, 8}
{1, 4, 7}
{-8, 4}
{-1, -3, 8}
{-3, -4, -8}
{-1, -2, 3, 4, -6}

{3, 6, -7, 8}
{1, 4, 7}
{-8, 4}
{-1, -3, 8}
{-3, -4, -8}
{-1, -2, 3, 4, -6}

{3, 6, -7, 8}
{1, 4, 7}
{-8, 4}
{-1, -3, 8}
{-3, -4, -8}
{-1, -2, 3, 4, -6}
Modern CDCL SAT Solver Architecture

Propagate(), Decide(), Analyze/Learn(), BackJump()

Input SAT Instance

- Propagate() (BCP)
  - No Conflict?
    - All Vars Assigned?
      - Conflict Analysis()
      - Decide()
      - TopLevel Conflict?
      - BackJump()
      - Return UNSAT
      - Return SAT

- D ecide()
  - \{3, 6, -7, 8\}
  - \{1, 4, 7\}
  - \{-8, 4\}
  - \{-1, -3, 8\}
  - \{-3, -4, -8\}
  - \{-1, -2, 3, 4, -6\}

- Conflict Analysis()
  - Unit clause (BCP)
    - More unit clauses (more BCP)
      - \{3, 6, -7, 8\}
      - \{1, 4, 7\}
      - \{-8, 4\}
      - \{-1, -3, 8\}
      - \{-3, -4, -8\}
      - \{-1, -2, 3, 4, -6\}

- Basic Backtracking Search
  - Flip the last decision \(1\)
  - Try setting \(1\) to False
  - Highly inefficient
  - No learning from mistakes

CONFLICT! (Trigger to analyze & backjump)
Some Definitions

- **Decision Level (DL)**
  - Map from Boolean variables in input to natural numbers
  - All unit clauses in input & resultant propagations get DL = 0
  - Every decision var gets a DL in increasing order >= 1
  - All propagations due to decision var at DL=x get the DL=x

- **Conflict Graph (CG) or Implication Graph**
  - Directed Graph that records decisions & propagations
  - Vertices: literals, Edge: unit clauses

- **Conflict Clause (CC)**
  - Clause returned by Conflict Analysis(), added to conflict DB
  - Implied by the input formula
  - A cut in the CG
  - Prunes the search

- **Assignment Trail (AT)**
  - A stack of partial assignment to literals, with DL info
Modern CDCL SAT Solver Architecture

Conflict Analysis/Learn() Details: Implication Graph

Current Assignment Trail: \{X_9 = 0@1, X_{10} = 0@3, X_{11} = 0@3, X_{12} = 1@2, X_{13} = 1@2, \ldots\}

Current decision: \{X_1 = 1@6\}

Clause DB

\begin{align*}
W_1 &= (\neg X_1 + X) \\
W_2 &= (\neg X_1 + X_3 + X_9) \\
W_3 &= (\neg X_2 + \neg X_3 + X_4) \\
W_4 &= (\neg X_4 + X_5 + X_{10}) \\
W_5 &= (\neg X_4 + X_6 + X_{11}) \\
W_6 &= (\neg X_5 + \neg X_6) \\
W_7 &= (X_1 + X_7 + \neg X_{12}) \\
W_8 &= (X_1 + X_8) \\
W_9 &= (\neg X_7 + \neg X_8 + \neg X_{13})
\end{align*}
Current Assignment Trail: {X_9 = 0@1, X_{10} = 0@3, X_{11} = 0@3, X_{12} = 1@2, X_{13} = 1@2, ...}

Current decision: {X_1 = 1@6}

Clause DB

W_1 = (¬X_1 + X_2)
W_2 = (¬X_1 + X_3 + X_9)
W_3 = (¬X_2 + ¬X_3 + X_4)
W_4 = (¬X_4 + X_5 + X_{10})
W_5 = (¬X_4 + X_6 + X_{11})
W_6 = (¬X_5 + ¬X_6)
W_7 = (X_1 + X_7 + ¬X_{12})
W_8 = (X_1 + X_8)
W_9 = (¬X_7 + ¬X_8 + ¬X_{13})
Current Assignment Trail: \{X_9 = 0@1, X_{10} = 0@3, X_{11} = 0@3, X_{12} = 1@2, X_{13} = 1@2, \ldots\}

Current decision: \{X_1 = 1@6\}

Clause DB

\begin{align*}
W_1 &= (\neg X_1 + X_2) \\
W_2 &= (\neg X_1 + X_3 + X_9) \\
W_3 &= (\neg X_2 + \neg X_3 + X_4) \\
W_4 &= (\neg X_4 + X_5 + X_{10}) \\
W_5 &= (\neg X_4 + X_6 + X_{11}) \\
W_6 &= (\neg X_5 + \neg X_6) \\
W_7 &= (X_1 + X_7 + \neg X_{12}) \\
W_8 &= (X_1 + X_8) \\
W_9 &= (\neg X_7 + \neg X_8 + \neg X_{13})
\end{align*}
Current Assignment Trail: \{X_9 = 0@1, X_{10} = 0@3, X_{11} = 0@3, X_{12} = 1@2, X_{13} = 1@2, \ldots\}

Current decision: \{X_1 = 1@6\}

Clause DB

\[W_1 = (\neg X_1 + X_2)\]
\[W_2 = (\neg X_1 + X_3 + X_9)\]
\[W_3 = (\neg X_2 + \neg X_3 + X_4)\]
\[W_4 = (\neg X_4 + X_5 + X_{10})\]
\[W_5 = (\neg X_4 + X_6 + X_{11})\]
\[W_6 = (\neg X_5 + \neg X_6)\]
\[W_7 = (X_1 + X_7 + \neg X_{12})\]
\[W_8 = (X_1 + X_8)\]
\[W_9 = (\neg X_7 + \neg X_8 + \neg X_{13})\]
Conflict Analysis/Learn() Details: Implication Graph

Current Assignment Trail: \{X_9 = 0@1, X_{10} = 0@3, X_{11} = 0@3, X_{12} = 1@2, X_{13} = 1@2, ...\}

Current decision: \{X_1 = 1@6\}

Clause DB

\begin{align*}
W_1 &= (\neg X_1 + X_2) \\
W_2 &= (\neg X_1 + X_3 + X_9) \\
W_3 &= (\neg X_2 + \neg X_3 + X_4) \\
W_4 &= (\neg X_4 + X_5 + X_{10}) \\
W_5 &= (\neg X_4 + X_6 + X_{11}) \\
W_6 &= (\neg X_5 + \neg X_6) \\
W_7 &= (X_1 + X_7 + \neg X_{12}) \\
W_8 &= (X_1 + X_8) \\
W_9 &= (\neg X_7 + \neg X_8 + \neg X_{13})
\end{align*}
Modern CDCL SAT Solver Architecture

Conflict Analysis/Learn() Details: Implication Graph

Current Assignment Trail: \{X_9 = 0@1, X_{10} = 0@3, X_{11} = 0@3, X_{12} = 1@2, X_{13} = 1@2, ...\}

Current decision: \{X_1 = 1@6\}

Clause DB

- \(W_1 = (\neg X_1 + X_2)\)
- \(W_2 = (\neg X_1 + X_3 + X_9)\)
- \(W_3 = (\neg X_2 + \neg X_3 + X_4)\)
- \(W_4 = (\neg X_4 + X_5 + X_{10})\)
- \(W_5 = (\neg X_4 + X_6 + X_{11})\)
- \(W_6 = (\neg X_5 + \neg X_6)\)
- \(W_7 = (X_1 + X_7 + \neg X_{12})\)
- \(W_8 = (X_1 + X_8)\)
- \(W_9 = (\neg X_7 + \neg X_8 + \neg X_{13})\)
Conflict Analysis/Learn() Details: Implication Graph

Current Assignment Trail: \{X_9 = 0@1, X_{10} = 0@3, X_{11} = 0@3, X_{12} = 1@2, X_{13} = 1@2, ...\}

Current decision: \{X_1 = 1@6\}

Clause DB

\begin{align*}
W_1 &= (\neg X_1 + X_2) \\
W_2 &= (\neg X_1 + X_3 + X_9) \\
W_3 &= (\neg X_2 + \neg X_3 + X_4) \\
W_4 &= (\neg X_4 + X_5 + X_{10}) \\
W_5 &= (\neg X_4 + X_6 + X_{11}) \\
W_6 &= (\neg X_5 + \neg X_6) \\
W_7 &= (X_1 + X_7 + \neg X_{12}) \\
W_8 &= (X_1 + X_8) \\
W_9 &= (\neg X_7 + X_8 + \neg X_{13})
\end{align*}
Modern CDCL SAT Solver Architecture

Conflict Analysis/Learn() Details: Implication Graph

Current Assignment Trail: \( \{X_9 = 0\oplus 1, X_{10} = 0\oplus 3, X_{11} = 0\oplus 3, X_{12} = 1\oplus 2, X_{13} = 1\oplus 2, \ldots\} \)

Current decision: \( \{X_1 = 1\oplus 6\} \)

Clause DB

\[ W_1 = (\neg X_1 + X_2) \]
\[ W_2 = (\neg X_1 + X_3 + X_9) \]
\[ W_3 = (\neg X_2 + \neg X_3 + X_4) \]
\[ W_4 = (\neg X_4 + X_5 + X_{10}) \]
\[ W_5 = (\neg X_4 + X_6 + X_{11}) \]
\[ W_6 = (\neg X_5 + \neg X_6) \]
\[ W_7 = (X_1 + X_7 + \neg X_{12}) \]
\[ W_8 = (X_1 + X_8) \]
\[ W_9 = (\neg X_7 + \neg X_8 + \neg X_{13}) \]
Modern CDCL SAT Solver Architecture
Conflict Analysis/Learn() Details: Implication Graph

Current Assignment Trail: {X_9 = 0@1, X_{10} = 0@3, X_{11} = 0@3, X_{12} = 1@2, X_{13} = 1@2, ...}
Current decision: {X_1 = 1@6}

Clause DB

\[ W_1 = ( \neg X_1 + X_2 ) \]
\[ W_2 = ( \neg X_1 + X_3 + X_9 ) \]
\[ W_3 = ( \neg X_2 + \neg X_3 + X_4 ) \]
\[ W_4 = ( \neg X_4 + X_5 + X_{10} ) \]
\[ W_5 = ( \neg X_4 + X_6 + X_{11} ) \]
\[ W_6 = ( \neg X_5 + \neg X_6 ) \]
\[ W_7 = ( X_1 + X_7 + \neg X_{12} ) \]
\[ W_8 = ( X_1 + X_8 ) \]
\[ W_9 = ( \neg X_7 + \neg X_8 + \neg X_{13} ) \]
Current Assignment Trail: \{X_9 = 0@1, X_{10} = 0@3, X_{11} = 0@3, X_{12} = 1@2, X_{13} = 1@2, ...\}

Current decision: \{X_1 = 1@6\}

**Clause DB**

- \(W_1 = (\neg X_1 + X_2)\)
- \(W_2 = (\neg X_1 + X_3 + X_9)\)
- \(W_3 = (\neg X_2 + \neg X_3 + X_4)\)
- \(W_4 = (\neg X_4 + X_5 + X_{10})\)
- \(W_5 = (\neg X_4 + X_6 + X_{11})\)
- \(W_6 = (\neg X_5 + \neg X_6)\)
- \(W_7 = (X_1 + X_7 + \neg X_{12})\)
- \(W_8 = (X_1 + X_8)\)
- \(W_9 = (\neg X_7 + \neg X_8 + \neg X_{13})\)
Modern CDCL SAT Solver Architecture

Conflict Analysis/Learn() Details: Implication Graph

Current Assignment Trail: \{X_9 = 0@1, X_{10} = 0@3, X_{11} = 0@3, X_{12} = 1@2, X_{13} = 1@2, \ldots\}

Current decision: \{X_1 = 1@6\}

Clause DB

\begin{align*}
W_1 &= (\neg X_1 + X_2) \\
W_2 &= (\neg X_1 + X_3 + X_9) \\
W_3 &= (\neg X_2 + \neg X_3 + X_4) \\
W_4 &= (\neg X_4 + X_5 + X_{10}) \\
W_5 &= (\neg X_4 + X_6 + X_{11}) \\
W_6 &= (\neg X_5 + \neg X_6) \\
W_7 &= (X_1 + X_7 + \neg X_{12}) \\
W_8 &= (X_1 + X_8) \\
W_9 &= (\neg X_7 + \neg X_8 + \neg X_{13})
\end{align*}
Current Assignment Trail: \{X_9 = 0@1, X_{10} = 0@3, X_{11} = 0@3, X_{12} = 1@2, X_{13} = 1@2, \ldots\}

Current decision: \{X_1 = 1@6\}

**Clause DB**

\begin{align*}
W_1 &= \neg X_1 + X_2 \\
W_2 &= \neg X_1 + X_3 + X_9 \\
W_3 &= \neg X_2 + \neg X_3 + X_4 \\
W_4 &= \neg X_4 + X_5 + X_{10} \\
W_5 &= \neg X_4 + X_6 + X_{11} \\
W_6 &= \neg X_5 + \neg X_6 \\
W_7 &= X_1 + X_7 + \neg X_{12} \\
W_8 &= X_1 + X_8 \\
W_9 &= \neg X_7 + \neg X_8 + \neg X_{13}
\end{align*}
Current Assignment Trail: \{X_9 = 0@1, X_{10} = 0@3, X_{11} = 0@3, X_{12} = 1@2, X_{13} = 1@2, \ldots\}

Current decision: \{X_1 = 1@6\}

Clause DB

\begin{align*}
W_1 &= (\neg X_1 + X_2) \\
W_2 &= (\neg X_1 + X_3 + X_9) \\
W_3 &= (\neg X_2 + \neg X_3 + X_4) \\
W_4 &= (\neg X_4 + X_5 + X_{10}) \\
W_5 &= (\neg X_4 + X_6 + X_{11}) \\
W_6 &= (\neg X_5 + \neg X_6) \\
W_7 &= (X_1 + X_7 + \neg X_{12}) \\
W_8 &= (X_1 + X_8) \\
W_9 &= (\neg X_7 + \neg X_8 + \neg X_{13})
\end{align*}
Modern CDCL SAT Solver Architecture

Conflict Analysis/Learn() Details: Implication Graph

Current Assignment Trail: \{X_9 = 0@1, X_{10} = 0@3, X_{11} = 0@3, X_{12} = 1@2, X_{13} = 1@2, \ldots\}

Current decision: \{X_1 = 1@6\}

Clause DB

- \(W_1 = (\neg X_1 + X_2)\)
- \(W_2 = (\neg X_1 + X_3 + X_9)\)
- \(W_3 = (\neg X_2 + \neg X_3 + X_4)\)
- \(W_4 = (\neg X_4 + X_5 + X_{10})\)
- \(W_5 = (\neg X_4 + X_6 + X_{11})\)
- \(W_6 = (\neg X_5 + \neg X_6)\)
- \(W_7 = (X_1 + X_7 + \neg X_{12})\)
- \(W_8 = (X_1 + X_8)\)
- \(W_9 = (\neg X_7 + \neg X_8 + \neg X_{13})\)
Modern CDCL SAT Solver Architecture

Conflict Analysis/Learn() Details: Implication Graph

Current Assignment Trail: \{X_9 = 0@1, X_{10} = 0@3, X_{11} = 0@3, X_{12} = 1@2, X_{13} = 1@2, ...\}

Current decision: \{X_1 = 1@6\}

Clause DB

\[
\begin{align*}
W_1 &= (\neg X_1 + X_2) \\
W_2 &= (\neg X_1 + X_3 + X_9) \\
W_3 &= (\neg X_2 + \neg X_3 + X_4) \\
W_4 &= (\neg X_4 + X_5 + X_{10}) \\
W_5 &= (\neg X_4 + X_6 + X_{11}) \\
W_6 &= (\neg X_5 + \neg X_6) \\
W_7 &= (X_1 + X_7 + \neg X_{12}) \\
W_8 &= (X_1 + X_8) \\
W_9 &= (\neg X_7 + \neg X_8 + \neg X_{13})
\end{align*}
\]
Current Assignment Trail: \{X_9 = 0 @ 1, X_{10} = 0 @ 3, X_{11} = 0 @ 3, X_{12} = 1 @ 2, X_{13} = 1 @ 2, ...\}

Current decision: \{X_1 = 1 @ 6\}

Clause DB

\begin{align*}
W_1 &= (\neg X_1 + X_2) \\
W_2 &= (\neg X_1 + X_3 + X_9) \\
W_3 &= (\neg X_2 + \neg X_3 + X_4) \\
W_4 &= (\neg X_4 + X_5 + X_{10}) \\
W_5 &= (\neg X_4 + X_6 + X_{11}) \\
W_6 &= (\neg X_5 + \neg X_6) \\
W_7 &= (X_1 + X_7 + \neg X_{12}) \\
W_8 &= (X_1 + X_8) \\
W_9 &= (\neg X_7 + \neg X_8 + \neg X_{13})
\end{align*}
Current Assignment Trail: \( \{X_9 = 0@1, X_{10} = 0@3, X_{11} = 0@3, X_{12} = 1@2, X_{13} = 1@2, \ldots\} \)

Current decision: \( \{X_1 = 1@6\} \)

Clause DB

\[
\begin{align*}
W_1 &= (\neg X_1 + X_2) \\
W_2 &= (\neg X_1 + X_3 + X_9) \\
W_3 &= (\neg X_2 + \neg X_3 + X_4) \\
W_4 &= (\neg X_4 + X_5 + X_{10}) \\
W_5 &= (\neg X_4 + X_6 + X_{11}) \\
W_6 &= (\neg X_5 + \neg X_6) \\
W_7 &= (X_1 + X_7 + \neg X_{12}) \\
W_8 &= (X_1 + X_8) \\
W_9 &= (\neg X_7 + \neg X_8 + \neg X_{13}) \\
\end{align*}
\]
Modern CDCL SAT Solver Architecture

Conflicting Analysis/Learn() Details: Implication Graph

Current Assignment Trail: \{X_9 = 0@1, X_{10} = 0@3, X_{11} = 0@3, X_{12} = 1@2, X_{13} = 1@2, \ldots\}

Current decision: \{X_1 = 1@6\}

Clause DB

\begin{align*}
W_1 &= (\neg X_1 + X_2) \\
W_2 &= (\neg X_1 + X_3 + X_9) \\
W_3 &= (\neg X_2 + \neg X_3 + X_{10}) \\
W_4 &= (\neg X_4 + X_5 + X_{10}) \\
W_5 &= (\neg X_4 + X_6 + X_{11}) \\
W_6 &= (\neg X_5 + \neg X_6) \\
W_7 &= (X_1 + X_7 + \neg X_{12}) \\
W_8 &= (X_1 + X_8) \\
W_9 &= (\neg X_7 + \neg X_8 + \neg X_{13})
\end{align*}
Conflict Analysis/Learn() Details: Implication Graph

Current Assignment Trail: \{X_9 = 0@1, X_{10} = 0@3, X_{11} = 0@3, X_{12} = 1@2, X_{13} = 1@2, \ldots\}

Current decision: \{X_1 = 1@6\}

Clause DB

\begin{align*}
W_1 &= (\neg X_1 + X_2) \\
W_2 &= (\neg X_1 + X_3 + X_9) \\
W_3 &= (\neg X_2 + \neg X_3 + X_4) \\
W_4 &= (\neg X_4 + X_5 + X_{10}) \\
W_5 &= (\neg X_4 + X_6 + X_{11}) \\
W_6 &= (\neg X_5 + \neg X_6) \\
W_7 &= (X_1 + X_7 + \neg X_{12}) \\
W_8 &= (X_1 + X_8) \\
W_9 &= (\neg X_7 + \neg X_8 + \neg X_{13})
\end{align*}
Modern CDCL SAT Solver Architecture

Conflict Analysis/Learn() Details: Implication Graph

Current Assignment Trail: \( \{X_9 = 0@1, X_{10} = 0@3, X_{11} = 0@3, X_{12} = 1@2, X_{13} = 1@2, \ldots\} \)

Current decision: \( \{X_1 = 1@6\} \)

Clause DB

\[
egin{align*}
W_1 &= (\neg X_1 + X_2) \\
W_2 &= (\neg X_1 + X_3 + X_9) \\
W_3 &= (\neg X_2 + \neg X_3 + X_4) \\
W_4 &= (\neg X_4 + X_5 + X_{10}) \\
W_5 &= (\neg X_4 + X_6 + X_{11}) \\
W_6 &= (\neg X_5 + \neg X_6) \\
W_7 &= (X_1 + X_7 + \neg X_{12}) \\
W_8 &= (X_1 + X_8) \\
W_9 &= (\neg X_7 + \neg X_8 + \neg X_{13})
\end{align*}
\]

CONFLICT GRAPH

\[
\begin{align*}
X_1 &= 1@6 \\
X_9 &= 0@1 \\
W_2_1 &= (\neg X_1 + X_2) \\
W_2_3 &= (\neg X_1 + X_3 + X_9) \\
W_3_3 &= (\neg X_2 + \neg X_3 + X_4) \\
W_4_4 &= (\neg X_4 + X_5 + X_{10}) \\
W_5_5 &= (\neg X_4 + X_6 + X_{11}) \\
W_6_6 &= (\neg X_5 + \neg X_6) \\
W_7_7 &= (X_1 + X_7 + \neg X_{12}) \\
W_8_8 &= (X_1 + X_8) \\
W_9_9 &= (\neg X_7 + \neg X_8 + \neg X_{13})
\end{align*}
\]
Conflict Analysis/Learn() Details: Conflict Clause

Current Assignment Trail: \{X_9 = 0@1, X_{10} = 0@3, X_{11} = 0@3, X_{12} = 1@2, X_{13} = 1@2, \ldots\}

Current Decision: \{X_1 = 1@6\}

Simplest strategy is to traverse the conflict graph backwards until decision variables:
conflict clause includes only decision variables (¬X_1 + X_9 + X_{10} + X_{11})
**Modern CDCL SAT Solver Architecture**

**Conflict Analysis/Learn() Details: Conflict Clause**

Current Assignment Trail: \{X_9 = 0 @ 1, X_{10} = 0 @ 3, X_{11} = 0 @ 3, X_{12} = 1 @ 2, X_{13} = 1 @ 2, \ldots\}\n
Current Decision: \{X_1 = 1 @ 6\}

Another strategy is to use First Unique Implicant Point (UIP):
Traverse graph backwards in breadth-first, expand literals of conflict, stop at first UIP

<table>
<thead>
<tr>
<th>Clause DB</th>
<th>CONFLICT GRAPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>(W_1 = (¬X_1 + X_2))</td>
<td>(X_1 = 1 @ 6)</td>
</tr>
<tr>
<td>(W_2 = (¬X_1 + X_3 + X_9))</td>
<td>(X_2 = 1 @ 6)</td>
</tr>
<tr>
<td>(W_3 = (¬X_2 + ¬X_3 + X_4))</td>
<td>(X_3 = 1 @ 6)</td>
</tr>
<tr>
<td>(W_4 = (¬X_4 + X_5 + X_{10}))</td>
<td>(X_4 = 1 @ 6)</td>
</tr>
<tr>
<td>(W_5 = (¬X_4 + X_6 + X_{11}))</td>
<td>(X_5 = 1 @ 6)</td>
</tr>
<tr>
<td>(W_6 = (¬X_5 + ¬X_6))</td>
<td>(X_6 = 1 @ 6)</td>
</tr>
<tr>
<td>(W_7 = (X_1 + X_7 + ¬X_{12}))</td>
<td>Conflict</td>
</tr>
<tr>
<td>(W_8 = (X_1 + X_8))</td>
<td></td>
</tr>
<tr>
<td>(W_9 = (¬X_7 + ¬X_8 + ¬X_{13}))</td>
<td></td>
</tr>
</tbody>
</table>

Vijay Ganesh

Tuesday, 29 January, 13
Current Assignment Trail: \(\{X_9 = 0@1, X_{10} = 0@3, X_{11} = 0@3, X_{12} = 1@2, X_{13} = 1@2, \ldots\}\)

Current decision: \(\{X_1 = 1@6\}\)

Strategy: Closest decision level (DL) \(\leq\) current DL for which conflict clause is unit. Undo \(\{X_1 = 1@6\}\)

Clause DB

\[
\begin{align*}
W_1 &= \neg X_1 + X_2 \\
W_2 &= \neg X_1 + X_3 + X_9 \\
W_3 &= \neg X_2 + \neg X_3 + X_4 \\
W_4 &= \neg X_4 + X_5 + X_{10} \\
W_5 &= \neg X_4 + X_6 + X_{11} \\
W_6 &= \neg X_5 + \neg X_6 \\
W_7 &= X_1 + X_7 + \neg X_{12} \\
W_8 &= X_1 + X_8 \\
W_9 &= \neg X_7 + \neg X_8 + \neg X_{13}
\end{align*}
\]
\[ \neg X_1 \text{ was implied literal, leading to another conflict described below} \]

Conflict clause: \((X_9 + X_{10} + X_{11} + \neg X_{12} + \neg X_{13})\)

BackJump strategy: Closest decision level (DL) \(\leq\) current DL for which conflict clause is unit. Undo \(\{X_{10} = 0@3\}\)
Modern CDCL SAT Solver Architecture

Restarts and Forget

Input SAT Instance

Propagate() (BCP)

No Conflict?

All Vars Assigned?

Conflict Analysis()

Return SAT

Decide()

TopLevel Conflict?

Return UNSAT

BackJump()

- **Restarts**
  - Clear the Trail and start again
  - Start searching with a different variable order
  - Only Conflict Clause (CC) database is retained

- **Forget**: throw away less active learnt conflict clauses routinely
  - Routinely throw away very large CC
  - Logically CC are implied
  - Hence no loss in soundness/completeness
  - Time Savings: smaller DB means less work in propagation
  - Space savings
Why is SAT efficient?

Input SAT Instance

Propagate()
(BCP)

No Conflict?

All Vars Assigned?

Conflict Analysis()

Return SAT

Return UNSAT

Decide()

TopLevel Conflict?

BackJump()

• VSIDS branching heuristic and propagate (BCP)
• Conflict-Driven Clause-Learning (CDCL)
• Forget conflict clauses if DB goes too big
• BackJump
• Restarts
• All the above elements are needed for efficiency
• Deeper understanding lacking
• No predictive theory
Modern CDCL SAT Solver Architecture

Propagate(), Decide(), Analyze/Learn(), BackJump()

- Conflict-Driven Clause-Learning (CDCL) (Marques-Silva & Sakallah 1996)
- Decide/branch and propagate (BCP) (Malik et al. 2001, Zabih & McAllester 1988)
- BackJump (McAllester 1980, Marques-Silva & Sakallah 1999)
- Restarts (Selman & Gomes 2001)
- Follows MiniSAT (Een & Sorensson 2003)
Soundness: A solver is said to be sound, if, for any input formula \( F \), the solver terminates and produces a solution, then \( F \) is indeed SAT.

Proof: (Easy) SAT is returned only when all vars have been assigned a value (True, False) by Decide or BCP, and solver checks the solution.
Completeness: A solver is said to be complete, if, for any input formula \( F \) that is SAT, the solver terminates and produces a solution (i.e., solver does not miss solutions)

**Proof:** (Harder)
- Backtracking + BCP + decide is complete (easy)
- Conflict clause is implied by input formula (easy)
- Only need to see backjumping does not skip assignments
  - Observe backjumping occurs only when conflict clause (CC) vars < decision level (DL) of conflicting var
  - Backjumping to max(DL of vars in CC)
  - Decision tree rooted at max(DL of vars in CC)+1 is guaranteed to not satisfy CC
  - Hence, backjumping will not skip assignments
Modern CDCL SAT Solver Architecture
Soundness, Completeness & Termination

Termination: Some measure decreases every iteration

Proof Sketch:
- Loop guarantees either conflict clause (CC) added OR assign extended
- CC added. What stops CC addition looping forever?
  - Recall that CC is remembered
  - No CC duplication possible
- CC blocks UNSAT assign exploration in decision tree. No duplicate UNSAT assign exploration possible
  - Size of decision tree explored decreases for each CC add
Modern CDCL SAT Solver Architecture

References & Important SAT Solvers


7. zChaff SAT Solver by Lintao Zhang 2002.


9. MiniSAT Solver by Niklas Een and Niklas Sorenson 2005 - present

10. SAT Live: http://www.satlive.org/

11. SAT Competition: http://www.satcompetition.org/

1. SAT solvers are crucial for software engineering

2. Huge impact in formal methods, program analysis and testing

3. Key ideas that make SAT efficient
   1. Conflict-driven clause learning
   2. VSIDS (or similar) variable selection heuristics
   3. Backjumping
   4. Restarts

4. Techniques I didn’t discuss
   1. Survey propagation (belief propagation) by Selman & Gomes
   2. Works well for randomized SAT, not yet for industrial instances
   3. Physics-inspired
   4. Combining CDCL with survey propagation (?)