Lecture 14
Satisfiability Modulo Theories

1. Motivation: Path Sensitivity Analysis
2. A Basic SMT Solver
3. Optimizing the SMT Solver

Thanks to Clark Barrett, Nikolaj Bjørner Leonardo de Moura, Bruno Dutertre, Albert Oliveras, and Cesare Tinelli for contributing material used in this lecture.
What is Satisfiability Modulo Theories (SMT)?

• Satisfiability
  – the problem of determining whether a formula has a model
    (an assignment that makes the formula true)

• SAT: Satisfiability of **propositional formulas**
  – A model is a truth assignment to Boolean variables
  – SAT solvers: check satisfiability of propositional formulas
    • Decidable, NP-complete

• SMT: Satisfiability modulo theories
  – Satisfiability of first-order formulas containing operations from
    background theories such as arithmetic, arrays, uninterpreted
    functions, etc.
    E.g. \( g(a) = c \land f(g(a)) \neq f(c) \lor g(a) = d \land c \neq d \)
  – SMT Solvers:
    • check satisfiability of SMT formulas in a decidable first-order theory
User of SMT for Program Correctness & Test Generation

- Precision: Path sensitivity

- Given an assertion $A$, can we generate an input that triggers an error on a given path $p$?
  - Let $F$ be the formula representing the execution of $p$
  - Is the formula $F \land \neg A$ satisfiable?
    - Not satisfiable? No error on that path
    - Satisfiable? Find 1 assignment that satisfies the formula (1 set of test input)
Each Statement is a Logical Clause

Program
Assume data array bound is [0, N-1]

```c
1 void ReadBlocks(int data[], int cookie)
2 {
3     int i = 0;
4     while (true) {
5         int next;
6         next = data[i];
7         if (!(i < next && next < N)) return;
8         i = i + 1;
9         for (; i < next; i = i + 1) {
10            if (data[i] == cookie) {
11                i = i + 1;
12                Process(data[i]);
13            } else
14                Process(data[i]);
15         }
16     }
17 }
```
An Execution Path as a Logic Formula

Program Assume data array bound is [0, N-1]

1 void ReadBlocks(int data[], int cookie)
2 {
3     int i = 0;
4     while (true)
5         {
6             int next;
7             next = data[i];
8             if (!((i < next && next < N)) return;
9             i = i + 1;
10            for (; i < next; i = i + 1){
11                if (data[i] == cookie)
12                    i = i + 1;
13                else
14                    Process(data[i]);
15            }
16        }
17    }

One execution path (SSA)

\[ F = \]

\[ 3 \quad i_1 = 0; \]

\[ 7 \quad next_1 = data_0[i_1]; \]

\[ 8 \quad i_1 < next_1 \&\& next_1 < N_0 \]

\[ 9 \quad i_2 = i_1 + 1; \]

\[ 10 \quad i_2 < next_1; \]

\[ 11 \quad data_0[i_2] = cookie_0; \]

\[ 12 \quad i_3 = i_2 + 1; \]

\[ 13 \quad !i_4 < next_1; \]

\[ 14 \quad !i_4 < next_1; \]

\[ 7 \quad next_2 = data_0[i_4]; \]
Checking for Out-of-Bound Array Access 1

Program  Assume data array bound is [0, N-1]
1 void ReadBlocks(int data[], int cookie)
2 {
3   int i = 0;
4   while (true)
5   {
6     int next;
7     next = data[i];
8     if (!(i < next && next < N)) return;
9     i= i + 1;
10    for (; i<next; i = i + 1){
11       if (data[i] == cookie)
12          i = i + 1;
13       else
14          Process(data[i]);
15    }
16  }
17 }

One execution path (SSA)

3 i_1 = 0;
7 next_1 = data_0 [i_1];
8 i_2 = i_1 + 1;
10 i_2 < next_1;
11 data_0 [i_2] = cookie_0;
12 i_3 = i_2 + 1;
10 ! (i_4 < next_1);
7 next_2 = data_0 [i_4];

Line 7: Array bound assertion A:
\( 0 \leq i_1 \land i_1 < N_0 \)

Check: Is \( F \land \neg A \) satisfiable?
\( i_1 = 0 \land \neg (0 \leq i_1 \land i_1 < N_0) \)
Answer for Out-of-Bound Array Access 1

Program Assume data array bound is \([0, N-1]\)

```c
void ReadBlocks(int data[], int cookie) {
    int i = 0;
    while (true) {
        int next;
        next = data[i];
        if (!((i < next && next < N))) return;
        i = i + 1;
        for (; i < next; i = i + 1) {
            if (data[i] == cookie) {
                i = i + 1;
            } else {
                Process(data[i]);
            }
        }
    }
}
```

One execution path (SSA)

```c
3 \(i_1 = 0\);
7 \(next_1 = data_0[i_1];\)
8 \(i_1 < next_1 && next_1 < N_0\)
9 \(i_2 = i_1 + 1;\)
10 \(i_2 < next_1;\)
11 \(data_0[i_2] = cookie_0;\)
12 \(i_3 = i_2 + 1;\)
10 \(! (i_4 < next_1);\)
7 \(next_2 = data_0[i_4];\)
```

Line 7: Array bound assertion \(A\): \((0 \leq i_1 \land i_1 < N_0)\)

⇒ maps to \(i_1 = 0 \land \neg (0 \leq i_1 \land i_1 < N_0)\)

Check: Is \(F \land \neg A\) satisfiable?

Yes! \(\{i_1 \mapsto 0, N_0 \mapsto 0\}\)
Checking for Out-of-Bound Array Access 2

Program: Assume data array bound is $[0, N-1]$

```c
1 void ReadBlocks(int data[], int cookie)
2 {
3   int i = 0;
4   while (true)
5     {
6       int next;
7       next = data[i];
8       if (!(i < next && next < N)) return;
9       i = i + 1;
10      for (; i < next; i = i + 1){
11          if (data[i] == cookie)
12             i = i + 1;
13          else
14             Process(data[i]);
15      }
16   }
17 }
```

One execution path (SSA)

Line 7: Array bound assertion $A$:

\[(0 \leq i_4 \land i_4 < N_0)\]

Check: Is $F \land \neg A$ satisfiable?

\[F \land \neg (0 \leq i_4 \land i_4 < N_0)\]
Checking for Out-of-Bound Array Access 2

Program Assume data array bound is [0, N-1]

```c
void ReadBlocks(int data[], int cookie)
{
    int i = 0;
    while (true)
    {
        int next;
        next = data[i];
        if (!(i < next && next < N)) return;
        i = i + 1;
        for (; i < next; i = i + 1){
            if (data[i] == cookie)
                i = i + 1;
            else
                Process(data[i]);
        }
    }
}
```

One execution path (SSA)

<table>
<thead>
<tr>
<th>Var</th>
<th>I \rightarrow</th>
</tr>
</thead>
<tbody>
<tr>
<td>N₀</td>
<td>3</td>
</tr>
<tr>
<td>i₁</td>
<td>0</td>
</tr>
<tr>
<td>i₂</td>
<td>1</td>
</tr>
<tr>
<td>i₃</td>
<td>2</td>
</tr>
<tr>
<td>i₄</td>
<td>3</td>
</tr>
<tr>
<td>next₁</td>
<td>2</td>
</tr>
<tr>
<td>data₀</td>
<td>&lt;2,6,5&gt;</td>
</tr>
<tr>
<td>cookie₀</td>
<td>6</td>
</tr>
</tbody>
</table>

Line 7: Array bound assertion A:

\[(0 \leq i₄ \land i₄ < N₀)\]

\[F = \]
Handling Multiple Paths

- A program has many execution paths
- Conditional statements
  - Represent alternative paths symbolically with one formula using SSA
- Loops
  - Optimistically: Unroll a few times
  - Catches many errors, but not all errors
Conditional Statements

- **Conditional statements:** $\phi$ functions in SSA

```plaintext
1  if (i > 0) {
2      a = 2;
3      b = 3;
4  } else
5  {
6      a = 3;
7      b = 2;
8  }
9      c = a+b;
```

- **Assert $A$:** $c_3 = 5$
- **Is $F \land \neg A$ satisfiable?**

$$
\phi_1 = (i_0 > 0) \land (\phi_1 \rightarrow c_3 = 5) \land (\neg \phi_1 \rightarrow c_3 = 5) \land (c_3 \neq 5)
$$
A Resolution Example

- A resolution rule in propositional logic:

\[
\begin{array}{c}
\text{Given } p \lor A \text{ and } \neg p \lor B, \text{ add the resolvent } A \lor B \\
\text{Resolve} \\
p \lor A \quad \neg p \lor B \\
\hline \\
A \lor B
\end{array}
\]

- **Is** \(F \land \neg A\) **satisfiable**?

\[
\phi_1 = (i_0 > 0) \land (\phi_1 \rightarrow c_3 = 5) \land (\neg \phi_1 \rightarrow c_3 = 5) \land (c_3 \neq 5)
\]

- **Recall:** \(p \rightarrow q \equiv \neg p \lor q\)

\[
\phi_1 = (i_0 > 0) \land (\neg \phi_1 \lor c_3 = 5) \land (\phi_1 \lor c_3 = 5) \land (c_3 \neq 5)
\]

\[
\phi_1 = (i_0 > 0) \land (c_3 = 5) \land (c_3 \neq 5)
\]

- \(F \land \neg A\) **is not** satisfiable
- The assertion \(A\) is true.
Loops

- Optimistically: Unroll two times

```c
for (; i<next; i = i + 1) {
    if (data[i] == cookie)
        i = i + 1;
    else
        Process(data[i]);
}
```

```c
if (i < next) {
    if (data[i] == cookie)
        i = i + 1;
    else
        Process(data[i]);
    i = i + 1;
    if (i < next) {
        if (data[i] == cookie)
            i = i + 1;
        else
            Process(data[i]);
        i = i + 1;
    }
}
```
Loops: Apply SSA

1 if (i < next) {
2   if (data[i] == cookie) 
3      i = i + 1;
4   else 
5      Process(data[i]);
6   i = i + 1;
7 }
8
9 if (i < next) {
10  if (data[i] == cookie) 
11     i = i + 1;
12  else 
13     Process(data[i]);
14  i = i + 1;
15 }
16
1 φ₁ = (i₀ < next₀);
2 φ₂ = (data₀[i₀] == cookie₀);
3 i₁ = i₀ + 1;
4
5
6 i₂ = φ₂ ? i₁ : i₀;
7 i₃ = i₂ + 1;
8
9 φ₃ = (i₃ < next₀);
10 φ₄ = (data₀[i₃] == cookie₀);
11 i₄ = i₃ + 1;
12
13
14 i₅ = φ₄ ? i₄ : i₃;
15 i₆ = i₅ + 1;
16 i₇ = φ₃ ? i₆ : i₃;
17 i₈ = φ₁ ? i₇ : i₀;
# Major Categories of Program Analysis Tools

<table>
<thead>
<tr>
<th>Complete (Small programs)</th>
<th>Static Property Based</th>
<th>Dynamic Execution Based</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Verification</td>
<td>(Symbolic) Model Checking (SMT/BDD)</td>
</tr>
<tr>
<td></td>
<td>Prove a property in a program</td>
<td>Given a system model (sw/hw), simulate the execution to check if a property is true for all possible inputs. Symbolic: many states all at once</td>
</tr>
<tr>
<td></td>
<td>Floyd-Hoare logic:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>{pre-condition} s {post-condition}</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Applicable to small programs</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Incomplete (Large programs)</th>
<th>Static Analysis (Data flow)</th>
<th>Test case generation (SMT/BDD)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Abstract the program conservatively</td>
<td>Check a property opportunistically (e.g. unroll loops twice)</td>
</tr>
<tr>
<td></td>
<td>Check a property</td>
<td>Use analysis to generate test inputs</td>
</tr>
<tr>
<td></td>
<td>Sound: no false-negatives--find all bugs</td>
<td>No false-positives: generate a test</td>
</tr>
<tr>
<td></td>
<td>False-positives: false warnings</td>
<td>False-negatives: cannot find all bugs</td>
</tr>
<tr>
<td></td>
<td>Too imprecise is useless</td>
<td>No correctness/security guarantees</td>
</tr>
</tbody>
</table>
2. SMT Example with Linear Inequalities, Function Theories

\[ x \geq 0 \land f(x) \geq 0 \land f(y) \geq 0 \land x \neq y \]

Functions are “uninterpreted”:
Assumed to be pure, returning the same value for each input

This formula is satisfiable:
Example model
\[ x \mapsto 1 \]
\[ y \mapsto 2 \]
\[ f(1) \mapsto 0 \]
\[ f(2) \mapsto 1 \]
**SMT Example: with Arrays**

\[ b + 2 = c \land f(\text{read}(\text{write}(a, b, 3), c - 2)) \neq f(c - b + 1) \]

**Note:** write\((v, i, x)\) means \(v[i] := x;\)  
read\((v, i)\) means returns \(v[i]\)

By arithmetic, this is equivalent to  
\[ b + 2 = c \land f(\text{read}(\text{write}(a, b, 3), b)) \neq f(3) \]

By array theory axiom, \(\text{read}(\text{write}(v, i, x), i) = x\)  
\[ b + 2 = c \land f(3) \neq f(3) \]

By the theory of uninterpreted functions, \(f(3) \neq f(3)\) is not true

Therefore, this formula is not satisfiable
SMT Solvers

- Input: a first-order formula $F$
- Output
  - $F$ is satisfiable, optionally: a model $M$
  - $F$ is unsatisfiable, optionally: a proof of unsatisfiability
- Which is easier?
- Main issues
  - formula size (e.g. thousands of atoms or more)
  - formulas with complex Boolean structure
  - combination of theories
Overview of a SMT Solver

- SMT Solver = SAT Solver + Theory Solver
  - Given a formula $F$, the SAT solver enumerates possible truth assignments ($M$)
  - The theory solver is a decision procedure that checks whether the truth assignments are satisfiable in the theories
Example of a Basic Algorithm

\[ g(a) = c \land f(g(a)) \neq f(c) \lor g(a) = d \land c \neq d \]

SAT SOLVER

1. choose a model M
2. unsat
3. choose a model M
4. unsat

THEORY SOLVER

(Empty uninterpreted functions)

send F
unsat
send F
unsat
send F

M. Lam

CS243: SMT
Basic Algorithm

• T-conflict: check for conflicts with respect to theory T

Repeat

SAT Solver: propose a full propositional model M for formula F
if no M is found, F is unsatisfiable.

Theory Solver:
  Check for T-conflict on model M
  If M is satisfiable: F is satisfiable
  If M has a T-conflict, add constraint to F
3. Improvements (Example, Algorithm, Rules)

A. Incremental model decision:

Don't just guess the entire model (all the assignments)
Propagate the deduced assignments
Make one decision at a time
Check each assignment incrementally, not all at once.
(Propagate, Decide, T-Conflict, Learn, Restart)

B. Use the theory to propagate and learn (T-Propagate)

C. Backtrack to conflicting decision (Conflict, Explain, Backjump)
### A. Incremental: Example

\[ g(a) = c \land f(g(a)) \neq f(c) \lor g(a) = d \land c \neq d \]

<table>
<thead>
<tr>
<th>M</th>
<th>F</th>
<th>C</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1, 2 ∨ 3, 4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 2</td>
<td>1, 2 ∨ 3, 4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 2</td>
<td>1, 2 ∨ 3, 4</td>
<td>1 ∨ 2 ∨ 4</td>
<td>Propagate+, OK</td>
</tr>
<tr>
<td>2 4</td>
<td>1, 2 ∨ 3, 4</td>
<td>1 ∨ 2 ∨ 4</td>
<td>Decide</td>
</tr>
<tr>
<td>2 4</td>
<td>1, 2 ∨ 3, 4</td>
<td>1 ∨ 2 ∨ 4</td>
<td>T-Conflict</td>
</tr>
<tr>
<td>2 4</td>
<td>1, 2 ∨ 3, 4, 1 ∨ 2 ∨ 4</td>
<td>1 ∨ 2 ∨ 4</td>
<td>Learn</td>
</tr>
<tr>
<td>2 4</td>
<td>1, 2 ∨ 3, 4, 1 ∨ 2 ∨ 4</td>
<td>1 ∨ 2 ∨ 4</td>
<td>Restart</td>
</tr>
<tr>
<td>2 4</td>
<td>1, 2 ∨ 3, 4, 1 ∨ 2 ∨ 4</td>
<td>1 ∨ 2 ∨ 4</td>
<td>Propagate+</td>
</tr>
<tr>
<td>2 4</td>
<td>1, 2 ∨ 3, 4, 1 ∨ 2 ∨ 4</td>
<td>1 ∨ 3 ∨ 4 ∨ 2</td>
<td>T-Conflict, Learn</td>
</tr>
<tr>
<td>fail</td>
<td>1, 2 ∨ 3, 4, 1 ∨ 2 ∨ 4, 1 ∨ 3 ∨ 4 ∨ 2</td>
<td>1 ∨ 3 ∨ 4 ∨ 2</td>
<td>Fail</td>
</tr>
</tbody>
</table>
A. Incremental: Algorithm

- Build incrementally a satisfying truth assignment $M$ for a CNF formula $F$
  - CNF: conjunction of disjunctions of literals

- Algorithm
  Apply rules until there is a satisfying model or Fail, in decreasing priority
  
  T-conflict: if all the literals $l_1, \ldots, l_n$ in $M$ cannot be satisfied by $T$, set the conflict clause $C := \overline{t_1} \lor \ldots \lor \overline{t_n}$
  
  Learn: add the new conflict constraint to $F$
  
  Restart: Restart the SAT server after learning a new constraint
  
  Propagate: deduce the truth value of a literal from $M$ and $F$
  
  Decide: guess a truth value
  
  Fail: if there is no decision to roll back
## A. Incremental: Rules

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Propagate</strong></td>
<td>Deduce the truth value of a literal from $M$ and $F$</td>
</tr>
<tr>
<td>$l_1 \lor \ldots \lor l_n \in F$</td>
<td>$\overline{T_1}, \ldots, \overline{T_n} \in M$</td>
</tr>
<tr>
<td>$l, \overline{T} \notin M$</td>
<td>$M := M \cdot l$</td>
</tr>
<tr>
<td><strong>Decide</strong></td>
<td>Guess a truth value</td>
</tr>
<tr>
<td>$l \in \text{Lit}(F)$</td>
<td>$l, \overline{l} \notin M$</td>
</tr>
<tr>
<td>$M := M \cdot l$</td>
<td></td>
</tr>
<tr>
<td><strong>T-Conflict</strong></td>
<td>If all the literals $l_1, \ldots, l_n$ in $M$ cannot be satisfied by $T$, set the conflict clause $C := \overline{T_1} \lor \ldots \lor \overline{T_n}$</td>
</tr>
<tr>
<td>$C = \text{no}$</td>
<td>$l_1, \ldots, l_n \in M$</td>
</tr>
<tr>
<td>$l_1, \ldots, l_n \models_T \bot$</td>
<td>$C := \overline{T_1} \lor \ldots \lor \overline{T_n}$</td>
</tr>
<tr>
<td><strong>Learn</strong></td>
<td>Add the new learned constraint to formula $F$</td>
</tr>
<tr>
<td>$F \models_{ \mathcal{P} } C$</td>
<td>$C \notin F$</td>
</tr>
<tr>
<td>$F := F \cup {C}$</td>
<td></td>
</tr>
<tr>
<td><strong>Restart</strong></td>
<td>Restart the SAT solver</td>
</tr>
<tr>
<td>$M := M^{[0]}$</td>
<td>$C := \text{no}$</td>
</tr>
<tr>
<td>Each Decide defines a new level</td>
<td></td>
</tr>
<tr>
<td>$M^{[i]}$ means Model $M$ up to level $i$</td>
<td></td>
</tr>
</tbody>
</table>
A. Incremental: Rules

Fail if there is no decision to roll back

$$l_1 \lor \ldots \lor l_n \in F \quad \overline{T_1}, \ldots, \overline{T_n} \in M \quad \bullet \notin M$$

fail
Improvements (Example, Algorithm, Rules)

A. Incremental model decision
   (Propagate, Decide, T-Conflict, Learn, Restart)

B. Use the theory to propagate and learn (T-Propagate)

C. Backtrack to conflicting decision (Conflict, Explain, Backjump)
### B: T-Propagate: Example

\[
\begin{align*}
g(a) &= c \land f(g(a)) \neq f(c) \lor g(a) = d \land c \neq d
\end{align*}
\]

<table>
<thead>
<tr>
<th>M</th>
<th>F</th>
<th>C</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1, 2 \lor 3, 4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 4</td>
<td>1, 2 \lor 3, 4</td>
<td></td>
<td>Propagate+</td>
</tr>
<tr>
<td>1 4 2</td>
<td>1, 2 \lor 3, 4</td>
<td></td>
<td>T-Propagate (1 \models_T 2)</td>
</tr>
<tr>
<td>1 4 2 3</td>
<td>1, 2 \lor 3, 4</td>
<td></td>
<td>T-Propagate (1, 4 \models_T 3)</td>
</tr>
<tr>
<td>1 4 2 3</td>
<td>1, 2 \lor 3, 4</td>
<td>2 \lor 3</td>
<td>Conflict</td>
</tr>
<tr>
<td>fail</td>
<td>1, 2 \lor 3, 4</td>
<td></td>
<td>Fail</td>
</tr>
</tbody>
</table>

**Notation:**

1 \models_T 2: predicate 1 entails predicate 2 under theory $T$

If predicate 1 is true, predicate 2 is true under theory $T$
B. T-Propagate: Algorithm

- Add T-Propagate to increase deduced values using theory T

- Algorithm
  Apply rules until there is a satisfying model or Fail, in decreasing priority
  - **T-conflict**: if all the literals $l_1, \ldots, l_n$ in $M$ cannot be satisfied by $T$, set the conflict clause $C := \overline{T_1} \lor \cdots \lor \overline{T_n}$
  - **Learn**: add the new conflict constraint to $F$
  - **Restart**: Restart the SAT server after learning a new constraint
  - **Propagate**: deduce the truth value of a literal from $M$ and $F$
  - **T-Propagate**: deduce the truth value of a literal using theory $T$
  - **Decide**: guess a truth value
  - **Fail**: if there is no decision to roll back
B. T-Propagate: Rules

Deduce the truth value of a literal using theory $T$

$$
\text{M} := \text{M} \perp l
$$

T-Propagate

\[ l \in \text{Lit}(F) \quad \text{M} \models_T l \quad l, \bar{l} \notin \text{M} \]

\[ \text{M} := \text{M} \perp l \]
Improvements (Example, Algorithm, Rules)

A. Incremental model decision
   (Propagate, Decide, T-Conflict, Learn, Restart)

B. Use the theory to propagate and learn (T-Propagate)

C. Backtrack to conflicting decision (Conflict, Explain, Backjump)
### C. Backjumping: Example

\[ F := \{1, 1\lor 2, 3\lor 4, 5\lor 6, 1\lor 5\lor 7, 2\lor 5\lor 6\lor 7\} \]

<table>
<thead>
<tr>
<th>M</th>
<th>F</th>
<th>C</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(F)</td>
<td>(F)</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>(F)</td>
<td>(F)</td>
<td>Propagate</td>
</tr>
<tr>
<td>12</td>
<td>(F)</td>
<td>(F)</td>
<td>Propagate</td>
</tr>
<tr>
<td>12\cdot3</td>
<td>(F)</td>
<td>(F)</td>
<td>Decide</td>
</tr>
<tr>
<td>12\cdot34</td>
<td>(F)</td>
<td>(F)</td>
<td>Propagate</td>
</tr>
<tr>
<td>12\cdot34\cdot5</td>
<td>(F)</td>
<td>(F)</td>
<td>Decide</td>
</tr>
<tr>
<td>12\cdot34\cdot5\cdot66</td>
<td>(F)</td>
<td>(F)</td>
<td>Propagate</td>
</tr>
<tr>
<td>12\cdot34\cdot5\cdot67</td>
<td>(F)</td>
<td>(F)</td>
<td>Propagate</td>
</tr>
<tr>
<td>12\cdot34\cdot5\cdot67</td>
<td>(F)</td>
<td>(F)</td>
<td>Conflict</td>
</tr>
</tbody>
</table>

\[ 2\lor 5\lor 6\lor 7 \]
C. Backjumping: Example Details

\[ F := \{1, \overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor 6, \overline{1} \lor \overline{5} \lor 7, \overline{2} \lor \overline{5} \lor 6 \lor 7\} \]

\[ M := 12 \cdot 34 \cdot 567 \]

\[ C := \overline{2} \lor \overline{5} \lor 6 \lor 7 \]

- **Conflict:** \( \overline{2} \lor \overline{5} \lor 6 \lor 7 \) last literal choice is 7
- **Explain:** Choice of 7 is due to \( \overline{1} \lor \overline{5} \lor 7 \)
- **Learn:** \( \overline{1} \lor \overline{2} \lor \overline{5} \lor 6 \) = resolvent of \( \overline{2} \lor \overline{5} \lor 6 \lor 7 \) and \( \overline{1} \lor \overline{5} \lor 7 \)
- **Conflict:** \( \overline{1} \lor \overline{2} \lor \overline{5} \lor 6 \) last literal choice is 6
- **Explain:** Choice of 6 is due to \( \overline{5} \lor \overline{6} \)
- **Learn:** \( \overline{1} \lor \overline{2} \lor \overline{5} \) = resolvent of \( \overline{1} \lor \overline{2} \lor \overline{5} \lor 6 \) and \( \overline{5} \lor \overline{6} \)
- **Conflict:** \( \overline{1} \lor \overline{2} \lor \overline{5} \)
- **Backjump:** Choice of 5 was a decision
  - Conflict involves literals 1, 2, 5, the decision of 5 is at level 2
  - 1, 2 are both level 0
  - Back jump to level 0, propagate 1,2 and choose \( \overline{5} \)
### C. Backjumping: Example

\( F := \{1, \overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor 6, \overline{1} \lor \overline{5} \lor 7, \overline{2} \lor \overline{5} \lor 6 \lor 7\} \)

<table>
<thead>
<tr>
<th>M</th>
<th>F</th>
<th>C</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( F )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>( F )</td>
<td></td>
<td>Propagate</td>
</tr>
<tr>
<td>12\cdot3</td>
<td>( F )</td>
<td></td>
<td>Decide</td>
</tr>
<tr>
<td>12\cdot34</td>
<td>( F )</td>
<td></td>
<td>Propagate</td>
</tr>
<tr>
<td>12\cdot34\cdot5</td>
<td>( F )</td>
<td></td>
<td>Decide</td>
</tr>
<tr>
<td>12\cdot34\cdot5\cdot6</td>
<td>( F )</td>
<td></td>
<td>Propagate</td>
</tr>
<tr>
<td>12\cdot34\cdot5\cdot6\cdot7</td>
<td>( F )</td>
<td>( \overline{2} \lor \overline{5} \lor 6 \lor 7 )</td>
<td>Conflict</td>
</tr>
<tr>
<td>12\cdot34\cdot5\cdot6\cdot7</td>
<td>( F )</td>
<td>( \overline{1} \lor \overline{2} \lor \overline{5} \lor 6 )</td>
<td>Explain with ( \overline{1} \lor \overline{5} \lor 7 )</td>
</tr>
<tr>
<td>12\cdot34\cdot5\cdot6\cdot7</td>
<td>( F )</td>
<td>( \overline{1} \lor \overline{2} \lor \overline{5} )</td>
<td>Explain with ( \overline{5} \lor \overline{6} )</td>
</tr>
<tr>
<td>( 1 )\cdot\overline{5}</td>
<td>( F )</td>
<td></td>
<td>Backjump</td>
</tr>
<tr>
<td>( 1 )\cdot\overline{5}\cdot3</td>
<td>( F )</td>
<td></td>
<td>Decide</td>
</tr>
<tr>
<td>( 1 )\cdot\overline{5}\cdot34</td>
<td>( F )</td>
<td></td>
<td>Propagate (SAT)</td>
</tr>
</tbody>
</table>
C. Backjumping: Algorithm

- If $M$ is T-unsatisfiable, backtrack to some point where the assignment was still T-satisfiable

- Find the root cause that causes the conflict $C$
  - **Explain:** Given conflict $C$ involving latest choice $l$, $\overline{t}$ chosen due to clause $C_1$ in $F$ (explanation),
    new conflict = resolvent of $C$ and $C_1$
  - Since $l$ is forced -> not the root cause, backtracking on $l$ is meaningless
  - The resolvent distills down the constraint, eliminating the choice of $l$
  - Repeat application of “Explain” until a decision was made

- Backtrack by skipping decisions immaterial to conflict $C$
  - **Backjump:** Keep model up to level $i$,
    (highest level of satisfiable decisions involved in $C$); 
    add the latest literal $l$ in $C$
C. Backjumping Rules

If one of the literals $\overline{T_1}, \ldots, \overline{T_n}$ in M must be inverted in F, set the conflict clause $C := l_1 \lor \ldots \lor l_n$

**Conflict**
\[
\begin{align*}
C &= \text{no} \\
\overline{l_1} \lor \ldots \lor \overline{l_n} &\in F \\
T_1, \ldots, T_n &\in M \\
C := l_1 \lor \ldots \lor l_n
\end{align*}
\]

Given conflict C involving latest $l$, chosen due to a clause in F, their resolvent is the new conflict

**Explain**
\[
\begin{align*}
C &= l \lor D \\
\overline{l_1} \lor \ldots \lor \overline{l_n} &\in F \\
\overline{T_1}, \ldots, T_n &<_M \overline{T} \\
C := l_1 \lor \ldots \lor l_n \lor D
\end{align*}
\]

Keep model up to level $i$ (highest level of sat. decisions involved in C); add latest $l$ in C

**Backjump**
\[
\begin{align*}
C &= l_1 \lor \ldots \lor l_n \lor l \\
\text{lev} \overline{T_1}, \ldots, \text{lev} \overline{T_n} &\leq i < \text{lev} \overline{T} \\
C := \text{no} \\
M := M^{[i]} l
\end{align*}
\]

$l <_M l'$ if $l$ occurs before $l'$ in M

$M^{[i]}$ means Model M up to level $i$

lev $l = i$ iff $l$ occurs in decision level $i$ of $l$
C. Backjumping Rules (cont.)

Replace

Fail if there is no decision to roll back

<table>
<thead>
<tr>
<th>Fail</th>
<th>( \bigvee \limits_{i=1}^{n} l_i \in F ) \quad \bigvee \limits_{i=1}^{n} \overline{T_i} \in M \quad \bullet \notin M</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>fail</td>
</tr>
</tbody>
</table>

with

Fail if there is a conflict and there is no decision to roll back

<table>
<thead>
<tr>
<th>Fail</th>
<th>( C \neq \text{no} ) \quad \bullet \notin M</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>fail</td>
</tr>
</tbody>
</table>
Putting it All Together

Apply rules until there is a satisfying model or Fail, in decreasing priority

**T-conflict:** if all the literals $l_1, \ldots, l_n$ in $M$ cannot be satisfied by $T$, set the conflict clause $C := \overline{l_1} \lor \cdots \lor \overline{l_n}$

Explain: Given conflict $C$ involving latest choice $l$,
- $\overline{C}$ chosen due to clause $C_1$ in $F$ (explanation),
- new conflict = resolvent of $C$ and $C_1$

Backjump: Keep model up to level $i$,
- (highest level of satisfiable decisions involved in $C$);
- add the latest literal $l$ in $C$

Learn: add the new conflict constraint to $F$

Propagate: deduce the truth value of a literal from $M$ and $F$

T-Propagate: deduce the truth value of a literal using theory $T$

Decide: guess a truth value

Fail: if there is no decision to roll back

Restart: Restart on the learned $F$ if too many conflicts have been found
Summary

• Use of SMT to handle path sensitivity in test generation & static analysis

• Basic optimizations in SMT Solver
  – Incremental model decision (Propagate, Decide, T-Conflict, Learn, Restart)
  – Use the theory to propagate and learn (T-Propagate)
  – Smart backtracking (Conflict, Explain, Backjump)

• Many more optimizations to handle combinations of theory etc
• Practical tool: Z3 SMT solver
  – A widely used, open-source project from Microsoft
Further Readings

• “Satisfiability Modulo Theories”
  Clark Barrett and Cesare Tinelli.
  In Handbook of Model Checking,
  (Ed Clarke, Thomas Henzinger, and Helmut Veith, eds.), 2016.
  In preparation.

• “Satisfiability Modulo Theories”
  Clark Barrett, Roberto Sebastiani, Sanjit Seshia, and Cesare Tinelli.
  In Handbook of Satisfiability,
  vol. 185 of Frontiers in Artificial Intelligence and Applications,
  (Armin Biere, Marijn J. H. Heule, Hans van Maaren, and Toby Walsh, eds.),

• Satisfiability Modulo Theories: Introduction and Applications
  Leonardo De Moura, Nikolaj Bjørner
  Communications of the ACM, Vol. 54 No. 9, Pages 69-77
  Sept 2011