Lecture 15
Satisfiability Modulo Theories

1. Motivation: Path Sensitivity
2. Introduction to SMT
3. Basic SMT Optimizations

Thanks to Clark Barrett, Nikolaj Bjørner Leonardo de Moura, Bruno Dutertre, Albert Oliveras, and Cesare Tinelli for contributing material used in this lecture.
1. Goals of this Lecture

- **High-level introduction** to alternatives that complement material so far
  - Path-insensitive \(\rightarrow\) path-sensitive
  - Static analysis \(\rightarrow\) verification, model checking, test generation
  - BDDs \(\rightarrow\) SMT solvers
    - Top-level optimizations in SMT solvers
## From Testing to Verification

<table>
<thead>
<tr>
<th></th>
<th>Static Property Based</th>
<th>Dynamic Execution Based</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Incomplete</strong></td>
<td><strong>Static Analysis</strong></td>
<td><strong>Test case generation</strong></td>
</tr>
<tr>
<td>(Large programs)</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>
<tr>
<td><strong>Complete</strong></td>
<td><strong>Verification</strong></td>
<td><strong>(Symbolic) Model Checking</strong></td>
</tr>
<tr>
<td>(Small programs)</td>
<td><strong>Verification</strong></td>
<td>Given a system model, check if a property (e.g. linear temporal logic) is true for all possible inputs.</td>
</tr>
<tr>
<td></td>
<td>Prove a property in a program</td>
<td>Symbolic: many states all at once</td>
</tr>
<tr>
<td></td>
<td>Floyd-Hoare logic:</td>
<td>e.g. transitions captured as BDDs</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>

M. Lam

CS243: SMT
### Execution as Logic

**Program**

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

**One execution path**

Assume data array bound is [0, N-1]
# Execution as Logic

**Program**

```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)**

```c
3 i₁ = 0;
7 next₁ = data₀ [i₁];
8 i₁ < next₁ && next₁ < N₀
9 i₂ = i₁ + 1;
10 i₂ < next₁;
11 data₀ [i₂] != cookie₀;
14 Process(data₀ [i₂]);
10 i₃ = i₂ + 1;
10 !(i₃ < next₁);
7 next₂ = data₀ [i₃];
8 !(i₃ < next₂ && next₂ < N₀)
```
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]);
        }
    }
}
Correctness

Program

```c
1 void ReadBlocks(int data[], int cookie)
2 {
3    int i = 0;
4    while (true)
5    {
6        int next;
7        next = data[i];  \( \neg(0 \leq i < N) \)
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)

```c
3 \( i_1 = 0; \)
7 \( \text{next}_1 = \text{data}_0[\text{i}_1]; \)
8 \( \text{i}_1 < \text{next}_1 && \text{next}_1 < N \)
9 \( \text{i}_2 = \text{i}_1 + 1; \)
10 \( \text{i}_2 < \text{next}_1; \)
11 \( \text{data}_0[\text{i}_2] != \text{cookie}_0; \)
14 \( \text{Process}(\text{data}_0[\text{i}_2]); \)
10 \( \text{i}_3 = \text{i}_2 + 1; \)
10 \( \neg(\text{i}_3 < \text{next}_1); \)
7 \( \text{next}_2 = \text{data}_0[\text{i}_3]; \)
8 \( \neg(\text{i}_3 < \text{next}_2 && \text{next}_2 < N) \)
```

\( i_1 = 0 \land \neg(0 \leq i_1 < N_0) \)
\( \{i_1 \mapsto 0, N_0 \mapsto 0\}^{BUG!!} \)
Execution as Logic

Program

```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)

```
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;
13 !(0 \leq i_4 \land i_4 < N_0)
14 (0 \leq i_4)
15 i_4 = i_3 + 1;
16 !(i_4 < next_1);
17 next_2 = data_0[i_4];
```

<table>
<thead>
<tr>
<th>Var</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>N_0</td>
<td>3</td>
</tr>
<tr>
<td>i_1</td>
<td>0</td>
</tr>
<tr>
<td>i_2</td>
<td>1</td>
</tr>
<tr>
<td>i_3</td>
<td>2</td>
</tr>
<tr>
<td>i_4</td>
<td>3</td>
</tr>
<tr>
<td>next_1</td>
<td>2</td>
</tr>
<tr>
<td>data_0</td>
<td>&lt;2,6,5&gt;</td>
</tr>
<tr>
<td>cookie_0</td>
<td>6</td>
</tr>
</tbody>
</table>

BUG!!
Test Generation

• Given an assertion $A$, can we generate an input that triggers an error on a given path $p$?
  – Let $F$ be the formula summarizing 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)
Encoding multiple paths using $\phi$ functions

```plaintext
1  if (i < next) {
2      if (data[i] == cookie)
3          i = i + 1;
4      else
5          Process(data[i]);
6  }
7  i = i + 1;
8
9  if (i < next) {
10     if (data[i] == cookie)
11        i = i + 1;
12     else
13        Process(data[i]);
14  }
15  i = i + 1;
16 }
17 }
1 $\phi_1 = (i_0 < next_0)$;
2 $\phi_2 = (data_0 [i_0] == cookie_0)$;
3 $i_1 = i_0 + 1$;
4
5 $i_2 = \phi_2 ? i_1 : i_0$;
6 $i_3 = i_2 + 1$;
7
8 $\phi_3 = (i_3 < next_0)$;
9 $\phi_4 = (data_0 [i_3] == cookie_0)$;
10 $i_4 = i_3 + 1$;
11
12 $i_5 = \phi_4 ? i_4 : i_3$;
13 $i_6 = i_5 + 1$;
14 $i_7 = \phi_3 ? i_6 : i_3$;
15 $i_8 = \phi_1 ? i_7 : i_0$;
```
Conservative Analysis for Unbounded Computation

1. int i = 0; j = 0
2
3. while (data[i] != ‘\n’)
4. {
5.   i++;
6.   j = i;
7. }
8
9. assert (i==j)

1. int i = 0; j = 0
2
3. if (data[i] != ‘\n’)
4. {
5.   i = *;
6.   j = *;
7.   i++;
8.   j = i;
9. }
10
11. assert (i==j)

- Replace unbounded loops with a conservative approximation
  - * = an unknown value
- Conservative: Find all bugs but may have false positives
Test Generation

• Given an assertion $A$, can we generate an input that triggers an error on some path?
  – Let $F$ be the formula representing all possible paths
  – Is the formula $F \land \neg A$ satisfiable?
    • Not satisfiable? The program is proven correct

• Issues: how to represent unbounded computation
  – Opportunistic: e.g. unroll 2 times (miss some errors)
  – Conservative: generates false positives
2. Introduction to SAT and SMT

- Satisfiability
  - the problem of determining whether a formula has a model

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

- SMT: Satisfiability modulo theories
  - An SMT instance is a formula in first-order-logic
    - where some function & predicate symbols have extra interpretations
    - E.g. linear arithmetic, uninterpreted functions, arrays, bitvectors
  - SMT Solvers:
    - check satisfiability of SMT instances in a decidable first-order theory
Example 1

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

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

By arithmetic, this is equivalent to

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

By array theory axiom, \( \text{read(write}(v, i, x), i) = x \)

\[ b + 2 = c \land f(3) \neq f(3) \]

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

Therefore, this formula is not satisfiable
Example 2

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

This formula is satisfiable:

Example model

\[
\begin{align*}
x & \mapsto 1 \\
y & \mapsto 2 \\
f(1) & \mapsto 0 \\
f(2) & \mapsto 1
\end{align*}
\]
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 SMT Solving

• 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
Relationship between SAT and Theory Solver

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

SAT SOLVER

choose a model M

choose a model M

unsat

THEORY SOLVER
(Empty uninterpreted functions)

send F
unsat
send F
unsat
send F
Notation Introduction (Rules explained later)

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

<table>
<thead>
<tr>
<th>M</th>
<th>F</th>
<th>C</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1, 2 \lor 3, 4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1, 2, 4</td>
<td>1, 2 \lor 3, 4</td>
<td></td>
<td>Decide</td>
</tr>
<tr>
<td>1, 2, 4</td>
<td>1, 2 \lor 3, 4</td>
<td>\overline{1} \lor 2 \lor 4</td>
<td>T-Conflict</td>
</tr>
<tr>
<td>1, 2, 4</td>
<td>1, 2 \lor 3, 4, \overline{1} \lor 2 \lor 4</td>
<td>\overline{1} \lor 2 \lor 4</td>
<td>Learn</td>
</tr>
<tr>
<td></td>
<td>1, 2 \lor 3, 4, \overline{1} \lor 2 \lor 4</td>
<td></td>
<td>Restart</td>
</tr>
<tr>
<td>1, 3, 4</td>
<td>1, 2 \lor 3, 4, \overline{1} \lor 2 \lor 4</td>
<td></td>
<td>Decide</td>
</tr>
<tr>
<td>1, 3, 4</td>
<td>1, 2 \lor 3, 4, \overline{1} \lor 2 \lor 4</td>
<td>\overline{1} \lor 3 \lor 4</td>
<td>T-Conflict</td>
</tr>
<tr>
<td>1, 3, 4</td>
<td>1, 2 \lor 3, 4, \overline{1} \lor 2 \lor 4, \overline{1} \lor 3 \lor 4</td>
<td>\overline{1} \lor 3 \lor 4</td>
<td>Learn</td>
</tr>
<tr>
<td>fail</td>
<td>1, 2 \lor 3, 4, \overline{1} \lor 2 \lor 4, \overline{1} \lor 3 \lor 4</td>
<td></td>
<td>Fail</td>
</tr>
</tbody>
</table>
Outline

• SMT with full backtracking
  – Big picture: relationship between SAT and SMT
  – Introduce notation
• Basic algorithm
  – T-satisfiability means satisfiability with respect to theory T
  – Check T-satisfiability of a full propositional model $M$ for formula $F$
  – If $M$ is T-unsatisfiable, backtrack on the choice of a model
• 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)
### 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 \lor 3, 4)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(1 \overline{4})</td>
<td>(1, 2 \lor 3, 4)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(1 \overline{4} \cdot \overline{2})</td>
<td>(1, 2 \lor 3, 4)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(1 \overline{4} \cdot 2)</td>
<td>(1, 2 \lor 3, 4)</td>
<td>(\overline{1} \lor 2 \lor 4)</td>
<td>T-Conflict</td>
</tr>
<tr>
<td>(1 \overline{4} \cdot 2)</td>
<td>(1, 2 \lor 3, 4, \overline{1} \lor 2 \lor 4)</td>
<td>(\overline{1} \lor 2 \lor 4)</td>
<td>Learn</td>
</tr>
<tr>
<td>(1 \overline{4})</td>
<td>(1, 2 \lor 3, 4, \overline{1} \lor 2 \lor 4)</td>
<td></td>
<td>Restart</td>
</tr>
<tr>
<td>(1 \overline{4} 2 3)</td>
<td>(1, 2 \lor 3, 4, \overline{1} \lor 2 \lor 4)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(1 \overline{4} 2 3)</td>
<td>(1, 2 \lor 3, 4, \overline{1} \lor 2 \lor 4, \overline{1} \lor 3 \lor 4)</td>
<td>(\overline{1} \lor 3 \lor 4)</td>
<td>T-Conflict, Learn</td>
</tr>
<tr>
<td>fail</td>
<td></td>
<td></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
- 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 $C := \overline{l_1} \lor \cdots \lor \overline{l_n}$
  - **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></td>
<td>$l_1 \lor \ldots \lor l_n \lor l \in F$</td>
</tr>
<tr>
<td></td>
<td>$\overline{T}_1, \ldots, \overline{T}_n \in M$</td>
</tr>
<tr>
<td></td>
<td>$l, \overline{T} \notin M$</td>
</tr>
<tr>
<td></td>
<td>$M := M \cdot l$</td>
</tr>
<tr>
<td><strong>Decide</strong></td>
<td>Guess a truth value</td>
</tr>
<tr>
<td></td>
<td>$l \in \text{Lit}(F)$</td>
</tr>
<tr>
<td></td>
<td>$l, \overline{l} \notin M$</td>
</tr>
<tr>
<td></td>
<td>$M := M \cdot l$</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 $T$,</td>
</tr>
<tr>
<td></td>
<td>set the conflict clause $C := \overline{T}_1 \lor \ldots \lor \overline{T}_n$</td>
</tr>
<tr>
<td></td>
<td>$C = \text{no}$</td>
</tr>
<tr>
<td></td>
<td>$l_1, \ldots, l_n \in M$</td>
</tr>
<tr>
<td></td>
<td>$l_1, \ldots, l_n \models_T \perp$</td>
</tr>
<tr>
<td></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></td>
<td>$F \models P$</td>
</tr>
<tr>
<td></td>
<td>$C \notin F$</td>
</tr>
<tr>
<td></td>
<td>$F := F \cup {C}$</td>
</tr>
<tr>
<td><strong>Restart</strong></td>
<td>Restart the SAT solver</td>
</tr>
<tr>
<td></td>
<td>$M := M^{[0]} \quad C := \text{no}$</td>
</tr>
<tr>
<td></td>
<td>Each Decide defines a new level</td>
</tr>
<tr>
<td></td>
<td>$M^{[i]}$</td>
</tr>
<tr>
<td></td>
<td>means Model $M$ up to level $i$</td>
</tr>
</tbody>
</table>
A. Incremental: Rules

Fail if there is no decision to roll back

\[
\text{Fail} \quad l_1 \lor \ldots \lor l_n \in F \quad \overline{T_1}, \ldots, \overline{T_n} \in M \quad \bullet \notin M
\]

\[
\text{fail}
\]
Outline

• SMT with full backtracking
  – Big picture: relationship between SAT and SMT
  – Introduce notation
• Basic algorithm
  – T-satisfiability means satisfiability with respect to theory T
  – Check T-satisfiability of a full propositional model M for formula F
  – If M is T-unsatisfiable, backtrack on the choice of a model
• 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

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

<table>
<thead>
<tr>
<th>M</th>
<th>F</th>
<th>C</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</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>\overline{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>
B. T-Propagate: Algorithm

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

$T$-Propagate \[ l \in \text{Lit}(F) \quad M \models_T l \quad l', l \notin M \]

$M := M \lor l$
Outline

• SMT with full backtracking
  – Big picture: relationship between SAT and SMT
  – Introduce notation
• Basic algorithm
  – T-satisfiability means satisfiability with respect to theory T
  – Check T-satisfiability of a full propositional model M for formula F
  – If M is T-unsatisfiable, backtrack on the choice of a model
• 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

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

<table>
<thead>
<tr>
<th>M</th>
<th>F</th>
<th>C</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( F )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>( F )</td>
<td></td>
<td>Propagate</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\cdot56</td>
<td>( F )</td>
<td></td>
<td>Propagate</td>
</tr>
<tr>
<td>12\cdot34\cdot567</td>
<td>( F )</td>
<td>( \overline{2} \lor \overline{5} \lor 6 \lor 7 )</td>
<td>Conflict</td>
</tr>
</tbody>
</table>

M. Lam  
CS243: SMT  
30
C. Backjumping: Example Details

\[ F := \{1, \overline{1}v2, \overline{3}v4, \overline{3}v6, \overline{1}v5v7, \overline{2}v5v6v\overline{7}\} \]
\[ M := 12\cdot34\cdot5\overline{67} \]
\[ C := \overline{2}v5v6v\overline{7} \]

- Conflict: \( \overline{2}v5v6v\overline{7} \) last literal choice is 7
- Explain: Choice of 7 is due to \( \overline{1}v5v7 \)
- Learn: \( \overline{1}v2v5v6 \) = resolvent of \( \overline{2}v5v6v\overline{7} \) and \( \overline{1}v5v7 \)
- Conflict: \( \overline{1}v2v5v6 \) last literal choice is 6
- Explain: Choice of \( \overline{6} \) is due to \( \overline{5}v6 \)
- Learn: \( \overline{1}v2v\overline{5} \) = resolvent of \( \overline{1}v2v5v6 \) and \( \overline{5}v6 \)
- Conflict: \( \overline{1}v2v\overline{5} \)
- 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 \( \overline{5} \)

Resolve Rule: Given \( p \lor A \) and \( \neg p \lor B \) add the resolvent \( (A \lor B) \)
C. Backjumping: Example

\( F := \{1, \overline{1v2}, \overline{3v4}, \overline{5v6}, \overline{1v5v7}, \overline{2v5v6v7}\} \)

<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>Propagate</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>Decide</td>
</tr>
<tr>
<td>12\cdot3</td>
<td>( F )</td>
<td>( F )</td>
<td>Propagate</td>
</tr>
<tr>
<td>12\cdot34</td>
<td>( F )</td>
<td>( F )</td>
<td>Decide</td>
</tr>
<tr>
<td>12\cdot34\cdot5</td>
<td>( F )</td>
<td>( F )</td>
<td>Propagate</td>
</tr>
<tr>
<td>12\cdot34\cdot\overline{567}</td>
<td></td>
<td>( 2v5v6\overline{7} )Conflict</td>
<td></td>
</tr>
<tr>
<td>12\cdot34\cdot\overline{567}</td>
<td></td>
<td>( 1v2v5v6 )Explain with ( 1v5v7 )</td>
<td></td>
</tr>
<tr>
<td>12\cdot34\cdot\overline{567}</td>
<td></td>
<td>( 1v2v5 )Explain with ( 5v6 )</td>
<td></td>
</tr>
<tr>
<td>12\overline{5}</td>
<td></td>
<td>( 1v2v5 )Backjump</td>
<td></td>
</tr>
<tr>
<td>12\overline{5}\cdot3</td>
<td>( F )</td>
<td>( F )</td>
<td>Decide</td>
</tr>
<tr>
<td>12\overline{5}\cdot34</td>
<td>( F )</td>
<td>( F )</td>
<td>Propagate (SAT)</td>
</tr>
</tbody>
</table>
C. Backjumping: Algorithm

- **Resolve Rule:** Given $(p \lor A)$ and $(\neg p \lor B)$ add the resolvent $(A \lor B)$
- 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 $l$, $T$ 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

**Conflict**

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

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

**Explain**

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

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

**Backjump**

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

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

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

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

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

Replace

<table>
<thead>
<tr>
<th>Fail</th>
<th>Fail if there is no decision to roll back</th>
</tr>
</thead>
<tbody>
<tr>
<td>$l_1 \lor \ldots \lor l_n \in F$</td>
<td>$T_1, \ldots, T_n \in M$</td>
</tr>
</tbody>
</table>

fail

with

<table>
<thead>
<tr>
<th>Fail</th>
<th>Fail if there is a conflict and there is no decision to roll back</th>
</tr>
</thead>
<tbody>
<tr>
<td>C \neq no</td>
<td>$\cdot \not\in M$</td>
</tr>
</tbody>
</table>

fail
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 $C := \neg l_1 \lor \ldots \lor \neg l_n$
  – 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 $C_1$
  – 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