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 → path-sensitive
  - Static analysis → verification, model checking, test generation
  - BDDs → SMT solvers
    - Top-level optimizations in SMT solvers

From Testing to Verification

	Static Property Based	Dynamic Execution Based
Incomplete (Large programs)	**Static Analysis** Abstract the program conservatively Check a property Sound: no false-negatives--find all bugs False-positives: false warnings Too imprecise is useless	**Test case generation** Check a property opportunistically (e.g. unroll loops twice) Use analysis to generate test inputs No false-positives: generate a test False-negatives: cannot find all bugs No correctness/security guarantees
Complete (Small programs)	**Verification** Prove a property in a program Floyd-Hoare logic: {pre-condition} s {post-condition} Applicable to small programs	**(Symbolic) Model Checking** Given a system model, check if a property (e.g. linear temporal logic) is true for all possible inputs. Symbolic: many states all at once e.g. transitions captured as BDDs

Execution as Logic

Program

```
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

```
 3 i = 0;
 7 next = data[i];
 8 i < next && next < N
 9 i = i + 1;
10 i < next;
11 data[i] != cookie;
14 Process(data[i]);
10 i = i + 1;
10 !(i < next);
7  next = data[i];
8  !(i < next && next < N)
```

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

Execution as Logic

Program

```
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 $data_0\,[i_2] \mathrel{!=} cookie_0;$

14 $Process(data_0\,[i_2]);$

10 $i_3 = i_2 + 1;$

10 $!(i_3 < next_1);$

7 $next_2 = data_0\,[i_3];$

8 $!(i_3 < next_2 \;\&\&\; next_2 < N_0)$

Execution as Logic

Program

```
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)

$$\bigwedge \left\{ \begin{array}{ll}
3 & i_1 = 0; \\
 & \\
7 & next_1 = data_0\,[i_1]; \\
8 & i_1 < next_1 \;\&\&\; next_1 < N_0 \\
9 & i_2 = i_1 + 1; \\
10 & i_2 < next_1; \\
11 & data_0\,[i_2] \;!=\; cookie_0; \\
 & \\
14 & Process(data_0\,[i_2]); \\
10 & i_3 = i_2 + 1; \\
10 & !(i_3 < next_1); \\
7 & next_2 = data_0\,[i_3]; \\
8 & !(i_3 < next_2 \;\&\&\; next_2 < N_0)
\end{array} \right\}$$

Correctness

Program

```
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 }
```

$\neg(0 \leq i_1 \wedge i_1 < N_0)$

One execution path (SSA)

$$\bigwedge \left\{ \begin{array}{ll}
3 & i_1 = 0; \\
7 & next_1 = data_0\,[i_1]; \\
8 & i_1 < next_1 \ \&\& \ next_1 < N_0 \\
9 & i_2 = i_1 + 1; \\
10 & i_2 < next_1; \\
11 & data_0\,[i_2] \mathrel{!=} cookie_0; \\
14 & Process(data_0\,[i_2]); \\
10 & i_3 = i_2 + 1; \\
10 & !(i_3 < next_1); \\
7 & next_2 = data_0\,[i_3]; \\
8 & !(i_3 < next_2 \ \&\& \ next_2 < N_0)
\end{array} \right\}$$

$i_1 = 0 \wedge \neg(0 \leq i_1 \wedge i_1 < N_0)$

$\{i_1 \mapsto 0,\ N_0 \mapsto 0\}$ BUG!!

Execution as Logic

Program

```
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 }
```

$\neg(0 \le i_4 \wedge i_4 < N_0)$

Var	$\mapsto$
N_0	3
i_1	0
i_2	1
i_3	2
i_4	3
$next_1$	2
$data_0$	<2,6,5>
$cookie_0$	6

BUG!!

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 $i_3 = i_2 + 1;$

10 $i_4 = i_3 + 1;$
10 $!(i_4 < next_1);$
7 $next_2 = data_0[i_4];$

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 $F \wedge \neg A$ 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 ϕ functions

```
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;
  }
}
```

$$
\begin{array}{l}
\phi_1 = (i_0 < next_0);\\
\phi_2 = (data_0\,[i_0] == cookie_0);\\
i_1 = i_0 + 1;\\
\\
\\
i_2 = \phi_2\, ?\, i_1 : i_0;\\
i_3 = i_2 + 1;\\
\\
\phi_3 = (i_3 < next_0);\\
\phi_4 = (data_0\,[i_3] == cookie_0);\\
i_4 = i_3 + 1;\\
\\
\\
i_5 = \phi_4\, ?\, i_4 : i_3;\\
i_6 = i_5 + 1;\\
i_7 = \phi_3\, ?\, i_6 : i_3;\\
i_8 = \phi_1\, ?\, i_7 : i_0;
\end{array}
$$

Conservative Analysis for Unbounded Computation

```
int i = 0; j = 0

while (data[i] != '\n')
{
 i++;
 j = i;
}

assert (i==j)
```

```
int i = 0; j = 0

if (data[i] != '\n')
{
 i = *;
 j = *;
 i++;
 j = i;
}

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 $F \wedge \neg A$ 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

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

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

By arithmetic, this is equivalent to

$$b + 2 = c \wedge f(\text{read}(\text{write}(a, b, 3), b)) \neq f(3)$$

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

$$b + 2 = c \wedge f(3) \neq f(3)$$

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

Therefore, this formula is not satisfiable

Example 2

$$x \geq 0 \wedge f(x) \geq 0 \wedge f(y) \geq 0 \wedge x \neq y$$

This formula is satisfiable:

Example model

$$x \mapsto 1$$

$$y \mapsto 2$$

$$f(1) \mapsto 0$$

$$f(2) \mapsto 1$$

SMT Solvers

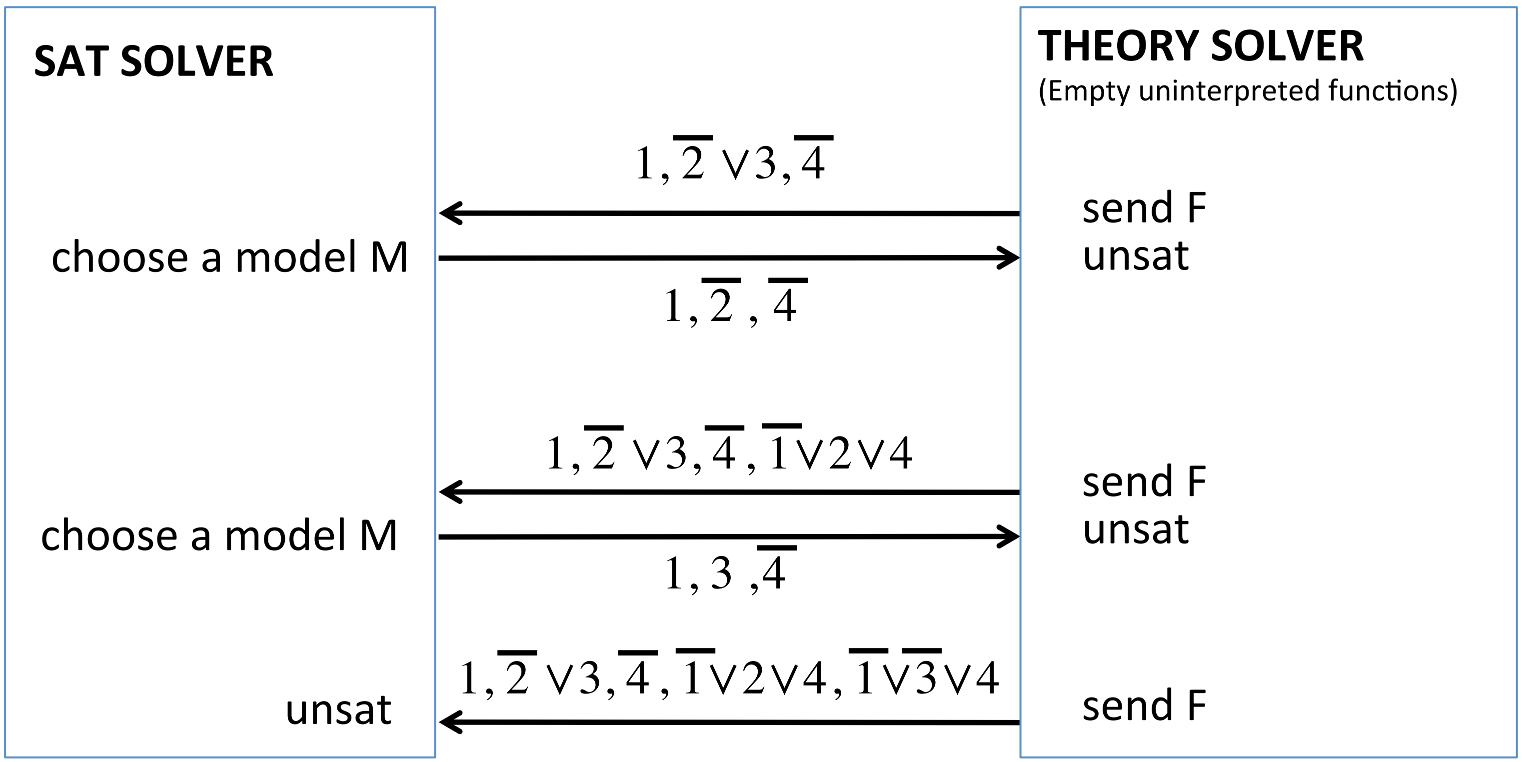
- Input: a first-order formula F
- Output
  - F is satisfiable, optionally: a model M
  - F is unsatisfiable, optionally: a proof of unsatisfiability
- Which is easier?
- Main issues
  - formula size (e.g. thousands of atoms or more)
  - formulas with complex Boolean structure
  - combination of theories

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

$$\underbrace{g(a)=c}_{1} \wedge \underbrace{f(g(a))\neq f(c)}_{\overline{2}} \vee \underbrace{g(a)=d}_{3} \wedge \underbrace{c\neq d}_{\overline{4}}$$



Notation Introduction (Rules explained later)

$$\underbrace{g(a)=c}_{1} \wedge \underbrace{f(g(a)) \neq f(c)}_{\overline{2}} \vee \underbrace{g(a)=d}_{3} \wedge \underbrace{c \neq d}_{\overline{4}}$$

M	F	C	Rule
	$1, \overline{2} \vee 3, \overline{4}$		
$1, \overline{2}, \overline{4}$	$1, \overline{2} \vee 3, \overline{4}$		Decide
$1, \overline{2}, \overline{4}$	$1, \overline{2} \vee 3, \overline{4}$	$\overline{1} \vee 2 \vee 4$	T-Conflict
$1, \overline{2}, \overline{4}$	$1, \overline{2} \vee 3, \overline{4}, \overline{1} \vee 2 \vee 4$	$\overline{1} \vee 2 \vee 4$	Learn
	$1, \overline{2} \vee 3, \overline{4}, \overline{1} \vee 2 \vee 4$		Restart
$1, 3, \overline{4}$	$1, \overline{2} \vee 3, \overline{4}, \overline{1} \vee 2 \vee 4$		Decide
$1, 3, \overline{4}$	$1, \overline{2} \vee 3, \overline{4}, \overline{1} \vee 2 \vee 4$	$\overline{1} \vee \overline{3} \vee 4$	T-Conflict
$1, 3, \overline{4}$	$1, \overline{2} \vee 3, \overline{4}, \overline{1} \vee 2 \vee 4, \overline{1} \vee \overline{3} \vee 4$	$\overline{1} \vee \overline{3} \vee 4$	Learn
fail			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)

A. Incremental: Example

$$\underbrace{g(a)=c}_{1} \wedge \underbrace{f(g(a)) \neq f(c)}_{\overline{2}} \vee \underbrace{g(a)=d}_{3} \wedge \underbrace{c \neq d}_{\overline{4}}$$

M	F	C	Rule
	$1, \overline{2} \vee 3, \overline{4}$		
$1\ \overline{4}$	$1, \overline{2} \vee 3, \overline{4}$		Propagate+
$1\ \overline{4} \bullet \overline{2}$	$1, \overline{2} \vee 3, \overline{4}$		Decide
$1\ \overline{4} \bullet \overline{2}$	$1, \overline{2} \vee 3, \overline{4}$	$\overline{1} \vee 2 \vee 4$	T-Conflict
$1\ \overline{4} \bullet \overline{2}$	$1, \overline{2} \vee 3, \overline{4}, \overline{1} \vee 2 \vee 4$	$\overline{1} \vee 2 \vee 4$	Learn
$1\ \overline{4}$	$1, \overline{2} \vee 3, \overline{4}, \overline{1} \vee 2 \vee 4$		Restart
$1\ \overline{4}\ 2\ 3$	$1, \overline{2} \vee 3, \overline{4}, \overline{1} \vee 2 \vee 4$		Propagate+
$1\ \overline{4}\ 2\ 3$	$1, \overline{2} \vee 3, \overline{4}, \overline{1} \vee 2 \vee 4, \overline{1} \vee \overline{3} \vee 4$	$\overline{1} \vee \overline{3} \vee 4$	T-Conflict, Learn
fail			Fail

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 := $\overline{l_1} \vee \ldots \vee \overline{l_n}$
  - 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

Propagate

Deduce the truth value of a literal from M and F

$$\frac{l_1 \vee \ldots \vee l_n \vee l \in \mathrm{F} \qquad \overline{l}_1, \ldots, \overline{l}_n \in \mathrm{M} \qquad l, \overline{l} \notin \mathrm{M}}{\mathrm{M} := \mathrm{M}\ l}$$

Decide

Guess a truth value

$$\frac{l \in \mathrm{Lit(F)} \qquad l, \overline{l} \notin \mathrm{M}}{\mathrm{M} := \mathrm{M} \bullet l}$$

T-Conflict

If all the literals $l_1, \ldots, l_n$ in M cannot be satisfied by T, set the conflict clause $\mathrm{C} := \overline{l}_1 \vee \ldots \vee \overline{l}_n$

$$\frac{\mathrm{C} = \mathrm{no} \qquad l_1, \ldots, l_n \in \mathrm{M} \qquad l_1, \ldots, l_n \models_T \in \bot}{\mathrm{C} := \overline{l}_1 \vee \ldots \vee \overline{l}_n}$$

Learn

Add the new learned constraint to formula F

$$\frac{\mathrm{F} \models_{\mathrm{P}} C \qquad C \notin \mathrm{F}}{\mathrm{F} := \mathrm{F} \cup \{C\}}$$

Restart

Restart the SAT solver

$$\frac{}{\mathrm{M} := \mathrm{M}^{[0]} \qquad \mathrm{C} := \mathrm{no}}$$

Each Decide defines a new level

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

A. Incremental: Rules

Fail if there is no decision to roll back

Fail $$\frac{l_1 \vee \ldots \vee l_n \in F \qquad \overline{l_1}, \ldots, \overline{l_n} \in M \qquad \bullet \notin M}{\text{fail}}$$

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

$$\underbrace{g(a)=c}_{1} \wedge \underbrace{f(g(a)) \neq f(c)}_{\overline{2}} \vee \underbrace{g(a)=d}_{3} \wedge \underbrace{c \neq d}_{\overline{4}}$$

M	F	C	Rule
	$1, \overline{2} \vee 3, \overline{4}$		
$1\ \overline{4}$	$1, \overline{2} \vee 3, \overline{4}$		Propagate+
$1\ \overline{4}\ 2$	$1, \overline{2} \vee 3, \overline{4}$		T-Propagate ($1 \models_T 2$)
$1\ \overline{4}\ 2\ \overline{3}$	$1, \overline{2} \vee 3, \overline{4}$		T-Propagate ($1, \overline{4} \models_T \overline{3}$)
$1\ \overline{4}\ 2\ \overline{3}$	$1, \overline{2} \vee 3, \overline{4}$	$\overline{2} \vee 3$	Conflict
fail			Fail

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 := $\overline{l_1} \vee \ldots \vee \overline{l_n}$
  - 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
  - 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

$$\text{T-Propagate}\ \frac{l \in \mathrm{Lit}(\mathsf{F}) \quad \mathsf{M} \models_T l \quad l, \bar{l} \notin \mathsf{M}}{\mathsf{M} := \mathsf{M}\ l}$$

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

$F := \{1, \overline{1} \vee 2, \overline{3} \vee 4, \overline{5} \vee \overline{6}, \overline{1} \vee \overline{5} \vee 7, \overline{2} \vee \overline{5} \vee 6 \vee \overline{7}\}$

M	F	C	Rule
	F		
1	F		Propagate
12	F		Propagate
$12\bullet 3$	F		Decide
$12\bullet 34$	F		Propagate
$12\bullet 34\bullet 5$	F		Decide
$12\bullet 34\bullet 5\overline{6}$	F		Propagate
$12\bullet 34\bullet 5\overline{6}7$	F		Propagate
$12\bullet 34\bullet 5\overline{6}7$	F	$\overline{2} \vee \overline{5} \vee 6 \vee \overline{7}$	Conflict

C. Backjumping: Example Details

$F := \{1, \bar{1} \vee 2, \bar{3} \vee 4, \bar{5} \vee \bar{6}, \bar{1} \vee \bar{5} \vee 7, \bar{2} \vee \bar{5} \vee 6 \vee \bar{7}\}$
$M := 12 \bullet 34 \bullet 5\bar{6}7$
$C := \bar{2} \vee \bar{5} \vee 6 \vee \bar{7}$

Resolve Rule:
Given $(p \vee A)$ and $(\neg p \vee B)$
add the resolvent $(A \vee B)$

- Conflict: $\bar{2} \vee \bar{5} \vee 6 \vee \bar{7}$ last literal choice is 7
- Explain: Choice of 7 is due to $\bar{1} \vee \bar{5} \vee 7$
- Learn: $\bar{1} \vee \bar{2} \vee \bar{5} \vee 6$ = resolvent of $\bar{2} \vee \bar{5} \vee 6 \vee \bar{7}$ and $\bar{1} \vee \bar{5} \vee 7$
- Conflict: $\bar{1} \vee \bar{2} \vee \bar{5} \vee 6$ last literal choice is 6
- Explain: Choice of $\bar{6}$ is due to $\bar{5} \vee \bar{6}$
- Learn: $\bar{1} \vee \bar{2} \vee \bar{5}$ = resolvent of $\bar{1} \vee \bar{2} \vee \bar{5} \vee 6$ and $\bar{5} \vee \bar{6}$
- Conflict: $\bar{1} \vee \bar{2} \vee \bar{5}$
- 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 $\bar{5}$

C. Backjumping: Example

$F := \{1, \overline{1}\vee 2, \overline{3}\vee 4, \overline{5}\vee\overline{6}, \overline{1}\vee\overline{5}\vee 7, \overline{2}\vee\overline{5}\vee 6\vee\overline{7}\}$

M	F	C	Rule
	F		
1	F		Propagate
12	F		Propagate
$12\bullet 3$	F		Decide
$12\bullet 34$	F		Propagate
$12\bullet 34\bullet 5$	F		Decide
$12\bullet 34\bullet 5\overline{6}$	F		Propagate
$12\bullet 34\bullet 5\overline{6}7$	F		Propagate
$12\bullet 34\bullet 5\overline{6}7$	F	$\overline{2}\vee\overline{5}\vee 6\vee\overline{7}$	Conflict
$12\bullet 34\bullet 5\overline{6}7$	F	$\overline{1}\vee\overline{2}\vee\overline{5}\vee 6$	Explain with $\overline{1}\vee\overline{5}\vee 7$
$12\bullet 34\bullet 5\overline{6}7$	F	$\overline{1}\vee\overline{2}\vee\overline{5}$	Explain with $\overline{5}\vee\overline{6}$
$12\overline{5}$	F		Backjump
$12\overline{5}\bullet 3$	F		Decide
$12\overline{5}\bullet 3\overline{4}$	F		Propagate (SAT)

C. Backjumping: Algorithm

- Resolve Rule: Given $(p \vee A)$ and $(\neg p \vee B)$ add the resolvent $(A \vee B)$
- 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,
    $\overline{l}$ 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 $\overline{l}_1,\ldots,\overline{l}_n$ in M must be inverted in F, set the conflict clause C := $l_1 \vee \ldots \vee l_n$

Conflict

$$\frac{\text{C = no} \qquad l_1 \vee \ldots \vee l_n \in \text{F} \qquad \overline{l}_1,\ldots,\overline{l}_n \in \text{M}}{\text{C} := l_1 \vee \ldots \vee l_n}$$

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

Explain

$$\frac{\text{C} = l \vee D \qquad l_1 \vee \ldots \vee l_n \vee \overline{l} \in \text{F} \qquad \overline{l}_1,\ldots,\overline{l}_n <_{\text{M}} \overline{l}}{\text{C} := l_1 \vee \ldots \vee l_n \vee D}$$

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

Backjump

$$\frac{\text{C} = l_1 \vee \ldots \vee l_n \vee l \qquad \text{lev}\,\overline{l}_1,\ldots,\text{lev}\,\overline{l}_n \leq i < \text{lev}\,\overline{l}}{\text{C := no} \qquad \text{M} := \text{M}^{[i]}\, l}$$

$l <_{\text{M}} l'$ if l occurs before l' in M

$\text{M}^{[i]}$ 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

Fail if there is no decision to roll back

$$\frac{l_1 \vee \ldots \vee l_n \in F \qquad \overline{l_1}, \ldots, \overline{l_n} \in M \qquad \bullet \notin M}{\text{fail}}$$

with

Fail

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

$$\frac{C \neq \text{no} \qquad \bullet \notin M}{\text{fail}}$$

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 := $\overline{l_1} \vee \ldots \vee \overline{l_n}$
  - Explain: Given conflict C involving latest choice l, $\overline{l}$ 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.
  http://theory.stanford.edu/~barrett/pubs/BT16-abstract.html

- "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.),
  Feb. 2009, pp. 825-885. http://theory.stanford.edu/~barrett/pubs/BSST09-abstract.html

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