Lecture 15

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. $g(a) = c \wedge f(g(a)) \neq f(c) \vee g(a) = d \wedge c \neq d$
  - SMT Solvers:
    - check satisfiability of SMT formulas with respect to a theory

Use 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 $F \wedge \neg A$ satisfiable?
    - Not satisfiable? No error on that path
    - Satisfiable? Find 1 assignment that satisfies the formula (1 set of test inputs)

Each Statement is a Logical Clause

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

Static Single Assignment (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 $next_2 = data_0\ [i_4];$

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 $i_1 = 0;$ $\}$

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 $next_2 = data_0[i_4];$

Line 7: Array bound assertion A:

$(0 \le i_1 \wedge i_1 < N_0)$

Checking for Out-of-Bound Array Access (Line 7, iteration 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)

$$F = \left\{ \begin{array}{ll} 3 & i_1 = 0; \\ 7 & next_1 = data_0\,[i_1]; \\ 8 & i_1 < next_1 \;\&\&\; next_1 < N_0 \\ 9 & i_2 = i_1 + 1; \\ 10 & i_2 < next_1; \\ 11 & data_0\,[i_2] = cookie_0; \\ 12 & i_3 = i_2 + 1; \\ 10 & i_4 = i_3 + 1; \\ 10 & !(i_4 < next_1); \\ 7 & next_2 = data_0\,[i_4]; \end{array} \right\}$$

Line 7: Array bound assertion A:

$$(0 \le i_1 \wedge i_1 < N_0)$$

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

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

Answer for Out-of-Bound Array Access (Line 7, iteration 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)

$$F = \left\{ \begin{array}{rl} 3 & i_1 = 0; \\ & \\ 7 & next_1 = data_0\,[i_1]; \\ 8 & i_1 < next_1 \;\&\&\; next_1 < N_0 \\ 9 & i_2 = i_1 + 1; \\ 10 & i_2 < next_1; \\ 11 & data_0\,[i_2] = cookie_0; \\ 12 & i_3 = i_2 + 1; \\ & \\ 10 & i_4 = i_3 + 1; \\ 10 & !(i_4 < next_1); \\ 7 & next_2 = data_0\,[i_4]; \end{array} \right\}$$

Line 7: Array bound assertion A:

$$(0 \le i_1 \wedge i_1 < N_0)$$

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

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

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

$\mapsto$ maps to

Checking for Out-of-Bound Array Access (Line 7, iteration 2)

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)

$$F = \bigwedge \left\{ \begin{array}{rl} 3 & \texttt{i}_1 = 0; \\ & \\ 7 & \texttt{next}_1 = \texttt{data}_0\,[\texttt{i}_1]; \\ 8 & \texttt{i}_1 < \texttt{next}_1 \;\&\&\; \texttt{next}_1 < \texttt{N}_0 \\ 9 & \texttt{i}_2 = \texttt{i}_1 + 1; \\ 10 & \texttt{i}_2 < \texttt{next}_1; \\ 11 & \texttt{data}_0\,[\texttt{i}_2] = \texttt{cookie}_0; \\ 12 & \texttt{i}_3 = \texttt{i}_2 + 1; \\ & \\ 10 & \texttt{i}_4 = \texttt{i}_3 + 1; \\ 10 & !(\texttt{i}_4 < \texttt{next}_1); \\ 7 & \texttt{next}_2 = \texttt{data}_0\,[\texttt{i}_4]; \end{array} \right\}$$

Line 7: Array bound assertion A:

$$(0 \le \texttt{i}_4 \wedge \texttt{i}_4 < \texttt{N}_0)$$

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

$$F \wedge \neg(0 \le \texttt{i}_4 \wedge \texttt{i}_4 < \texttt{N}_0)$$

Answer for Out-of-Bound Array Access (Line 7, iteration 2)

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 }
```

Line 7: Array bound assertion A:

$$(0 \le \mathtt{i}_4 \wedge \mathtt{i}_4 < \mathtt{N}_0)$$

One execution path (SSA)

$$F = \bigwedge \left\{ \begin{array}{ll} 3 & \mathtt{i}_1 = 0; \\ 7 & \mathtt{next}_1 = \mathtt{data}_0\,[\mathtt{i}_1]; \\ 8 & \mathtt{i}_1 < \mathtt{next}_1 \;\&\&\; \mathtt{next}_1 < \mathtt{N}_0 \\ 9 & \mathtt{i}_2 = \mathtt{i}_1 + 1; \\ 10 & \mathtt{i}_2 < \mathtt{next}_1; \\ 11 & \mathtt{data}_0\,[\mathtt{i}_2] = \mathtt{cookie}_0; \\ 12 & \mathtt{i}_3 = \mathtt{i}_2 + 1; \\ 10 & \mathtt{i}_4 = \mathtt{i}_3 + 1; \\ 10 & !(\mathtt{i}_4 < \mathtt{next}_1); \\ 7 & \mathtt{next}_2 = \mathtt{data}_0\,[\mathtt{i}_4]; \end{array} \right\}$$

Var	$\mapsto$
$\mathtt{N}_0$	3
$\mathtt{i}_1$	0
$\mathtt{i}_2$	1
$\mathtt{i}_3$	2
$\mathtt{i}_4$	3
$\mathtt{next}_1$	2
$\mathtt{data}_0$	<2,0,0>
$\mathtt{cookie}_0$	0

BUG!!

Checking the Whole Program All at Once

- 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: φ functions in SSA

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

9         c = a+b;
```

$$F = \bigwedge \left\{ \begin{array}{lll} 1 & \varphi_1 = (i_0 > 0) & \\ 2 & a_1 = 2 & \\ 3 & b_1 = 3 & \\ & & \\ 6 & a_2 = 3 & \\ 7 & b_2 = 2 & \\ 8 & a_3 = \varphi_1 ? a_1 : a_2; & // (\varphi_1 \rightarrow a_3 = a_1) \vee (\neg\varphi_1 \rightarrow a_3 = a_2) \\ 8 & b_3 = \varphi_1 ? b_1 : b_2; & // (\varphi_1 \rightarrow b_3 = b_1) \vee (\neg\varphi_1 \rightarrow b_3 = b_2) \\ 9 & c_3 = a_3 + b_3 & \end{array} \right\}$$

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

$$\varphi_1 = (i_0 > 0) \wedge (\varphi_1 \rightarrow c_3 = 5) \wedge (\neg\varphi_1 \rightarrow c_3 = 5) \wedge (c_3 \neq 5)$$

Applying the Resolution Rule to Example

- A resolution rule in propositional logic:

Given $p \vee A$ and $\neg p \vee B$, add the resolvent $A \vee B$

Resolve $$\frac{p \vee A \qquad \neg p \vee B}{A \vee B}$$

- Is $F \wedge \neg A$ satisfiable?

$\varphi_1 = (i_0 > 0) \wedge (\varphi_1 \rightarrow c_3 = 5) \wedge (\neg\varphi_1 \rightarrow c_3 = 5) \wedge (c_3 \neq 5)$

- Recall: $p \rightarrow q \equiv \neg p \vee q$

$\varphi_1 = (i_0 > 0) \wedge (\neg\varphi_1 \vee c_3 = 5) \wedge (\varphi_1 \vee c_3 = 5) \wedge (c_3 \neq 5)$

$\varphi_1 = (i_0 > 0) \wedge (c_3 = 5) \wedge (c_3 \neq 5)$

- $F \wedge \neg A$ is not satisfiable
- The assertion A is true.

Loops

- Optimistically: Unroll two times

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

```
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

```
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;
  }
}
```

$\varphi_1 = (i_0 < next_0);$

$\varphi_2 = (data_0[i_0] == cookie_0);$

$i_1 = i_0 + 1;$

$i_2 = \varphi_2 \, ? \, i_1 : i_0;$

$i_3 = i_2 + 1;$

$\varphi_3 = (i_3 < next_0);$

$\varphi_4 = (data_0[i_3] == cookie_0);$

$i_4 = i_3 + 1;$

$i_5 = \varphi_4 \, ? \, i_4 : i_3;$

$i_6 = i_5 + 1;$

$i_7 = \varphi_3 \, ? \, i_6 : i_3;$

$i_8 = \varphi_1 \, ? \, i_7 : i_0;$

Major Categories of Program Analysis Tools

	Static Property Based	Dynamic Execution Based
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 (SMT/BDD)** 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
Incomplete (Large programs)	**Static Analysis (Data flow)** 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 (SMT/BDD)** 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

2. A Basic SMT Solver

- SMT: Satisfiability modulo theories
  - Satisfiability of first-order formulas containing operations from background theories such as arithmetic, arrays, uninterpreted functions, etc.
- SMT Solvers:
  - check satisfiability of SMT formulas with respect to a theory

SMT with Linear Inequalities & Function Theories

Uninterpreted function theory:

- Functions assumed to be pure:
  - A function always returns the same value for a given input

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

Is this formula satisfiable?

SMT with Linear Inequalities & Function Theories

Uninterpreted function theory:

Functions assumed to be pure:

A function always returns the same value for a given input

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

This formula is satisfiable

An example model satisfying the formula

$$x \mapsto 0$$
$$y \mapsto 1$$
$$f(0) \mapsto 0$$
$$f(1) \mapsto 0$$

Adding a Theory of Arrays

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

Array theory axioms:

$$\text{read}(\text{write}(v, i, x), i) = x$$

$$i \neq j \;\rightarrow\; \text{read}(\text{write}(v, i, x), j) = \text{read}(v, j)$$

$$v \neq w \;\rightarrow\; \text{read}(v, k) \neq \text{read}(w, k) \text{ for some } k$$

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

Is this formula satisfiable?

Adding a Theory of Arrays

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

Array theory axioms:

$$\text{read}(\text{write}(v, i, x), i) = x$$
$$i \neq j \;\rightarrow\; \text{read}(\text{write}(v, i, x), j) = \text{read}(v, j)$$
$$v \neq w \;\rightarrow\; \text{read}(v, k) \neq \text{read}(w, k) \text{ for some } k$$

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

By arithmetic reasoning, this is equivalent to

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

By first array axiom, $b + 2 = c \wedge f(3) \neq f(3)$

But in 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 checks whether the truth assignments are satisfiable in the theories

Example of a Basic Algorithm

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

SAT SOLVER		THEORY SOLVER (Uninterpreted Functions)
	$\xleftarrow{1,\overline{2}\vee 3,\overline{4}}$	send F
choose a model M	$\xrightarrow[1,\overline{2},\overline{4}]{}$	unsat
	$\xleftarrow{1,\overline{2}\vee 3,\overline{4},\overline{1}\vee 2\vee 4}$	send F
choose a model M	$\xrightarrow[1,3,\overline{4}]{}$	unsat
unsat	$\xleftarrow{1,\overline{2}\vee 3,\overline{4},\overline{1}\vee 2\vee 4,\overline{1}\vee\overline{3}\vee 4}$	send F

Basic Algorithm

- DEFINITION
  T-conflict: check for conflicts with respect to theory T
- DESIGN
  Two independent solvers:
  - SAT solver that is independent of theory
  - Theory solver that checks for T-conflicts clause by clause
- ALGORITHM

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)
   Check each assignment incrementally, not all at once.
   Two kinds of assignments
   - Propagated: deduced from the Boolean expression
   - Arbitrary <u>decisions</u> (marked with •)

   If unsatisfiable, backtrack on decisions
   (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

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

M	F	C	Rule
	$1, \overline{2} \vee 3, \overline{4}$		
$1\ \overline{4}$	$1, \overline{2} \vee 3, \overline{4}$		Propagate+, OK
$1\ \overline{4} \bullet \overline{2}$	$1, \overline{2} \vee 3, \overline{4}$		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+
$1\ \overline{4}\ 2\ 3$	$1, \overline{2} \vee 3, \overline{4}, \overline{1} \vee 2 \vee 4, \overline{1} \vee \overline{3} \vee 4 \vee \overline{2}$	$\overline{1} \vee \overline{3} \vee 4 \vee \overline{2}$	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

- 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 $:= \overline{l}_1 \vee \ldots \vee \overline{l}_n$

  Learn: add the new conflict constraint to F

  Restart: Restart the SAT solver 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

$$\frac{l_1 \vee \ldots \vee l_n \in F \qquad \bar{l}_1, \ldots, \bar{l}_n \in M \qquad \bullet \notin M}{\text{fail}}$$

Improvements (Example, Algorithm, Rules)

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

B. Use the theory to propagate and learn (T-Propagate)
In A, propagation is based only on the Boolean expression;
Here, we add propagation due to the Theories

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 ($1, \overline{4} \models_T \overline{3}$)
Fail	$1, \overline{2} \vee 3, \overline{4}$		

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

  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

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

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)
   Find the root cause that causes the conflict
   Backtrack by skipping decisions immaterial to the conflict

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

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

$M := 12\bullet 34\bullet 5\overline{6}7$

$C := \overline{2}\vee\overline{5}\vee 6\vee\overline{7}$

Given $p \vee A$ and $\neg p \vee B$, add the resolvent $A \vee B$

Resolve $\dfrac{p \vee A \quad \neg p \vee B}{A \vee B}$

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

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

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

C. Backjumping: Algorithm

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

- Trace back to the decision that causes the conflict C
  - Let l be the last literal choice that causes conflict C,
    if l is a decision, proceed to the next step
  - Explain:
    if $\overline{l}$ is chosen due to clause C_1 in F (explanation),
    new conflict C = resolvent of C and C_1 (eliminating l)
  - Repeat the above

- 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

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

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

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

$l <_{\text{M}} l'$ 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: If the last literal l in conflict C is not a decision,
  If $\overline{l}$ 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)
- More work is needed to handle combinations of theories etc
- Practical tools:
  - Z3 SMT solver
    - A widely used, open-source project from Microsoft
  - CVC4 SMT solver (developed at Stanford)

Applications of SMT

- Program Analysis and Verification!
- Security (e.g. checking access policies at AWS)
- Hardware configuration and verification
- Verification of smart contracts for blockchains
- Checking database integrity constraints
- Network configuration checking
- Program synthesis
- And many more…

Research in SMT

- New theories and theory solvers, often adapted to an application domain
- Independently checkable proofs for unsatisfiable formulas
- Extensions to reasoning about quantifiers
- Extensions to synthesis
- Better decision heuristics
- Better algorithms and performance
- Theory combination mechanisms
- Parallel SMT solving

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