Outline

1 What is SMT?
Motivation

1. What is SMT?
   - Some use cases
   - Example: Encoding a scheduling problem
   - Example: Encoding symbolic execution
   - SMT basics
   - Difference and similarities with Hybrid SAT/CP

Automate reasoning in (fragments of) first-order logic (FOL)

SAT + Theory Solvers = SMT

Equality+UF Arithmetic etc.

Problem representation in propositional logic (PL):
  - Positive: Efficient (in practice) SAT algorithms
  - Negative: Expresiveness via CNF encodings

PL + domain-specific reasoning
  - Positive: Improved expressiveness
  - Negative: Less efficient than SAT

Note: Standard definitions of FOL apply (more later)
An example

- All variables integer
- Solve:
  \[
  (x_4 - x_2 \leq 3) \lor (x_4 - x_3 \geq 5) \land (x_4 - x_3 \leq 6) \land
  (x_1 - x_2 \leq -1) \land (x_1 - x_3 \leq -2) \land (x_1 - x_4 \leq -1) \land (x_2 - x_1 \leq 2) \land
  (x_3 - x_2 \leq -1) \land ((x_3 - x_4 \leq -2) \lor (x_4 - x_3 \geq 2))
  \]

- Integer difference logic (with Boolean structure)
- Unsatisfiable (Why?)
- How to solve formulas like the above?

Another example

- Let \( t_{i,j} \) be integer variables
- Solve:
  \[
  (t_{1,1} \geq 0) \land (t_{1,2} \geq t_{1,1} + 2) \land (t_{1,2} + 1 \leq 8) \land
  (t_{2,1} \geq 0) \land (t_{2,2} \geq t_{1,1} + 3) \land (t_{2,2} + 1 \leq 8) \land
  (t_{3,1} \geq 0) \land (t_{3,2} \geq t_{1,1} + 2) \land (t_{3,2} + 3 \leq 8) \land
  ((t_{1,1} \geq t_{2,1} + 3) \lor (t_{2,1} \geq t_{1,1} + 2)) \land
  ((t_{1,1} \geq t_{3,1} + 2) \lor (t_{3,1} \geq t_{1,1} + 2)) \land
  ((t_{2,1} \geq t_{3,1} + 2) \lor (t_{3,1} \geq t_{2,1} + 3)) \land
  ((t_{1,2} \geq t_{2,2} + 1) \lor (t_{2,2} \geq t_{1,2} + 1)) \land
  ((t_{1,2} \geq t_{3,2} + 3) \lor (t_{3,2} \geq t_{1,2} + 1)) \land
  ((t_{2,2} \geq t_{3,2} + 3) \lor (t_{3,2} \geq t_{2,2} + 1))
  \]

- Another example of integer difference logic (with Boolean structure)
- Satisfiable, with model: \( t_{1,1} = 5; t_{1,2} = 7; t_{2,1} = 2; t_{2,2} = 6; t_{3,1} = 0; t_{3,2} = 7; \)
- How to solve formulas like the above?
A scheduling example

○ Standard job-shop scheduling formulation:
  ▶ \( n \) jobs, each composed of \( m \) tasks to be performed consecutively on \( m \) machines
    ★ \( d_{i,j} \): duration of task \( j \) for job \( i \)

○ Types of constraints:
  ★ Precedence: between two tasks in the same job
  ★ Resource: No two different tasks requiring the same machine can execute simultaneously
  ★ All jobs must terminate by a time limit \( \max \)

○ An example:

<table>
<thead>
<tr>
<th>( d_{i,j} )</th>
<th>Machine 1</th>
<th>Machine 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job 1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Job 2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Job 3</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

with \( \max = 8 \)

An SMT model for job-shop scheduling:

○ \( t_{i,j} \): start time for task \( j \) of job \( i \)

○ Example:

<table>
<thead>
<tr>
<th>( d_{i,j} )</th>
<th>Machine 1</th>
<th>Machine 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job 1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Job 2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Job 3</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

with \( \max = 8 \)

○ Formulation:

\[
(t_{1,1} \geq 0) \land (t_{1,2} \geq t_{1,1} + 2) \land (t_{1,2} + 1 \leq 8) \land \\
(t_{2,1} \geq 0) \land (t_{2,2} \geq t_{1,1} + 3) \land (t_{2,2} + 1 \leq 8) \land \\
(t_{3,1} \geq 0) \land (t_{3,2} \geq t_{1,1} + 2) \land (t_{3,2} + 3 \leq 8) \land \\
((t_{1,1} \geq t_{2,1} + 3) \lor (t_{2,1} \geq t_{1,1} + 2)) \land \\
((t_{1,1} \geq t_{3,1} + 2) \lor (t_{3,1} \geq t_{1,1} + 2)) \land \\
((t_{2,1} \geq t_{3,1} + 2) \lor (t_{3,1} \geq t_{2,1} + 3)) \land \\
((t_{1,2} \geq t_{2,2} + 1) \lor (t_{2,2} \geq t_{1,2} + 1)) \land \\
((t_{1,2} \geq t_{3,2} + 3) \lor (t_{3,2} \geq t_{1,2} + 1)) \land \\
((t_{2,2} \geq t_{3,2} + 3) \lor (t_{3,2} \geq t_{2,2} + 1))
\]

○ Integer difference logic with Boolean structure
Software testing with symbolic execution

- Example C program:

  ```c
  int GCD (int x, int y) {
    while (true) {
      int m = x % y;
      if (m == 0) return y;
      x = y;
      y = m;
    }
  }
  ```

- Can the while loop test be executed twice?
  - If so, which inputs allow this to happen?

Software testing with symbolic execution (Cont.)

- Problem formulation as SMT formula:

  SSA Program
  ```c
  int GCD (int x0, int y0) {
    int m0 = x0 % y0;
    assert(m0 != 0);
    int x1 = y0;
    int y1 = m0;
    int m1 = x1 % y1;
    assert(m1 == 0);
  }
  ```

  Path Formula in SMT
  ```
  (m0 = x0 % y0) ∧
  ¬(m0 = 0) ∧
  (x1 = y0) ∧
  (y1 = m0) ∧
  (m1 = x1 % y0) ∧
  (m1 = 0)
  ```

- Note: SSA denotes static single assignment form
• Problem formulation as SMT formula:

```
C Program

int GCD (int x, int y) {
  while (true) {
    int m = x % y;
    if (m == 0) return y;
    x = y;
    y = m;
  }
}

Solution

• Model:
  x₀ = 2; y₀ = 4; m₀ = 2;
  x₁ = 4; y₁ = 2; m₁ = 0;

• Function call: GCD(2,4)
```

• Recall: This testing approach is known as dynamic symbolic execution
  ▶ Example tools: CUTE, Klee, DART, SAGE, Pex, Yogi

---

Example Theories – EUF

• Equality with Uninterpreted Functions (EUF)

• Is this formula satisfiable?

  \[ a \times (f(b) + f(c)) = d \land [b \times (f(a) + f(c)) \neq d] \land [a = b] \]

  ▶ Formula is unsatisfiable

• And this formula?

  \[ h(a, g(f(b), f(c))) = d \land [h(b, g(f(a), f(c))) \neq d] \land [a = b] \]

  ▶ Formula is also unsatisfiable

• Goal: Abstract non-supported operations (functions)
  ▶ E.g. multiplication; ALUs in circuits; etc.
Example Theories – Arithmetic

- Wide range of applications
- Variables are either integers or reals
- Decidable, but fairly high complexity

- Fragments can be solved with more efficient methods
  - **Bounds**
    - \( x \gg k, \gg \in \{<, >, \leq, \geq, = \} \)
  - **Difference Logic**
    - \( x - y \gg k, \gg \in \{<, >, \leq, \geq, = \} \)
  - **UTVPI (Unit Two-Variable Per Inequality)**
    - \( \pm x \pm y \gg k, \gg \in \{<, >, \leq, \geq, = \} \)
  - **Linear Arithmetic**
    - \( \sum a_i x_i \gg k, \gg \in \{<, >, \leq, \geq, = \} \)
  - **Non-Linear Arithmetic**
    - E.g. \( 3x y - 4x^2 z - 4y \leq 10 \)

What is SMT?

Other Theories

- Equality with Uninterpreted Functions (EUF)
- (Restricted) (linear/non-linear) arithmetic over the integers / reals
- Bit vectors
- Arrays
- Pointer logic
- Quantified fragments
Integer difference logic

- Integer variables
- Conjunction of linear inequalities of the form $x_i - x_j \leq k$

Algorithm:
- Add edge between $x_j$ and $x_i$ with weight $k$, for inequality $x_i - x_j \leq k$
- Add additional source vertex $x_0$
- Add edge from $x_0$ to $x_i$, for each other vertex $x_i$
- Use Bellman-Ford algorithm to check for negative cycles
  - Negative cycle: Elimination of variables in (some) inequalities yields $0 \leq -k$, $k > 0$
  - Note: More efficient algorithms exist

What is SMT?

Integer difference logic (Cont.)

\[(x_4 - x_2 \leq 3) \wedge (x_4 - x_3 \leq 6) \wedge (x_1 - x_2 \leq -1) \wedge (x_1 - x_3 \leq -2) \wedge (x_1 - x_4 \leq -1) \wedge (x_3 - x_2 \leq -1) \wedge (x_3 - x_4 \leq -2)\]

Satisfiable:
\[
\begin{align*}
x_1 &= -4 \\
x_2 &= 0 \\
x_3 &= -2 \\
x_4 &= 0
\end{align*}
\]
Integer difference logic (Cont.)

\[(x_4 - x_2 \leq 3) \land (x_4 - x_3 \leq 6) \land (x_1 - x_2 \leq -1) \land (x_1 - x_3 \leq -2) \land (x_2 - x_1 \leq -1) \land (x_3 - x_4 \leq -2)\]

Algorithms for SMT

- **Eager** approaches
  - Encode problem to CNF and solve with SAT solver

- **Lazy** approaches
  - Embed SAT solver with theory solver(s)
Eager Approaches

- Encode each theory to CNF
  - Integer variables encoded with Boolean variables
  - Encode AtMost\(k\), AtLeast\(k\), and pseudo-Boolean constraints to CNF
  - Recall: Can encode arbitrary constraints to CNF
  - Function/predicate symbols replaced by constants
    - E.g. replace \(f(a), f(b), f(c)\) with \(A, B, C\)
    - Add clauses:
      \[
      a = b \rightarrow A = B \\
      a = c \rightarrow A = C \\
      b = c \rightarrow B = C
      \]
- Solve CNF formula with SAT solver

Lazy approaches – example

- Example SMT formula:
  \[
  g(a) = c \land (f(g(a)) \neq f(c) \lor g(a) = d) \land c \neq d
  \]
- Represent Boolean structure as CNF formula:
  \[
  (x) \land (\neg y \lor z) \land (\neg w)
  \]
- Interaction between SAT solver & theory solver (EUF):

<table>
<thead>
<tr>
<th>SAT Outcome</th>
<th>Model/Core</th>
<th>EUF Outcome</th>
<th>Explanation clause (sent to SAT solver)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAT</td>
<td>({x, \neg y, \neg w})</td>
<td>UNSAT</td>
<td>((\neg x \lor y))</td>
</tr>
<tr>
<td>SAT</td>
<td>({x, y, z, \neg w})</td>
<td>UNSAT</td>
<td>((\neg x \lor \neg z \lor w))</td>
</tr>
<tr>
<td>UNSAT</td>
<td>((x) \land (\neg y \lor z) \land (\neg x \lor y) \land (\neg w) \land (\neg x \lor \neg z \lor w))</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Lazy approaches – another example

- Example SMT formula:

\[ ((x_4 - x_2 \leq 3) \lor (x_4 - x_3 \geq 5)) \land (x_4 - x_3 \leq 6) \land \\
(x_1 - x_2 \leq -1) \land (x_1 - x_3 \leq -2) \land (x_1 - x_4 \leq -1) \land (x_2 - x_3 \leq 2) \land \\
(x_3 - x_2 \leq -1) \land ((x_3 - x_4 \leq -2) \lor (x_4 - x_3 \geq 2)) \]

- Represent Boolean structure as CNF formula:

\[ (a \lor b) \land (c) \land (d) \land (e) \land (f) \land (g) \land (h) \land (i \lor j) \]

- Interaction between SAT solver & theory solver (IDL):

<table>
<thead>
<tr>
<th>SAT Outcome</th>
<th>Model/Core</th>
<th>IDL Outcome</th>
<th>Explanation clause (sent to SAT solver)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAT</td>
<td>{a, c, \ldots, h, i}</td>
<td>UNSAT</td>
<td>(\neg e \lor \neg g \lor \neg h)</td>
</tr>
<tr>
<td>UNSAT</td>
<td>(e \land (g) \land (h) \land (\neg e \lor \neg g \lor \neg h))</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Lazy approaches – remarks

- Why lazy?
  - Theory solver called as needed, to check T-consistency

- Key properties:
  - Very flexible organization
  - Modular implementation
    - Easy to add theory solvers
  - Currently, the most efficient algorithms
  - Clear separation between Boolean and theory domains
  - Theory information unable to guide search

- Widely used by modern SMT solvers
  - Z3, Yices, OpenSMT, MathSAT, CVC, Barcelogic, etc.
Key techniques in all efficient SMT solvers:

- Check T-consistency of partial assignments
- Given T-inconsistent assignment $M$, compute $M' \subseteq M$ and add $\neg M'$ as a clause
- Given T-inconsistent assignment, backtrack to where assignment is T-consistent

What is SMT?

$DPLL(T) = DPLL(X) + T$-Solver

- $DPLL(X)$
  - SAT solver
  - Cannot use: pure literals
- $T$-Solver:
  - Checks T-consistency of conjunctions of literals
  - Performs theory propagation
  - Computes explanations of inconsistency
  - Note: T-propagation should be incremental and backtrackable
Both methods combine dedicated algorithm (Theories / Propagators)

- The difference is subtle: a propagator could embed a Theory solver
  - They perform the same tasks (propagation, inconsistency detection, explanation)
  - But propagators are tied to CSP domains, and theories are difficult to combine

- **Advantages of SMT**: Theory domains can be open or even infinite (new literals can be added lazily)
  - May allow a stronger reasoning

- **Advantages of CP**: Propagators can be combined, theories cannot (as easily)
  - Interesting applications require solving formulas involving multiple theories