# Algorithms for Computational Logic

## Overview of Satisfiability Modulo Theories

Emmanuel Hebrard (adapted from) João Marques Silva



## Outline

## Outline

- Motivation

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



## What is SMT?

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


- 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:

$$((x_4 - x_2 \leq 3) \vee (x_4 - x_3 \geq 5)) \wedge (x_4 - x_3 \leq 6) \wedge$$
$$(x_1 - x_2 \leq -1) \wedge (x_1 - x_3 \leq -2) \wedge (x_1 - x_4 \leq -1) \wedge (x_2 - x_1 \leq 2) \wedge$$
$$(x_3 - x_2 \leq -1) \wedge ((x_3 - x_4 \leq -2) \vee (x_4 - x_3 \geq 2))$$

- 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:

$$(t_{1,1} \geq 0) \wedge (t_{1,2} \geq t_{1,1} + 2) \wedge (t_{1,2} + 1 \leq 8) \wedge$$
$$(t_{2,1} \geq 0) \wedge (t_{2,2} \geq t_{1,1} + 3) \wedge (t_{2,2} + 1 \leq 8) \wedge$$
$$(t_{3,1} \geq 0) \wedge (t_{3,2} \geq t_{1,1} + 2) \wedge (t_{3,2} + 3 \leq 8) \wedge$$
$$((t_{1,1} \geq t_{2,1} + 3) \vee (t_{2,1} \geq t_{1,1} + 2)) \wedge$$
$$((t_{1,1} \geq t_{3,1} + 2) \vee (t_{3,1} \geq t_{1,1} + 2)) \wedge$$
$$((t_{2,1} \geq t_{3,1} + 2) \vee (t_{3,1} \geq t_{2,1} + 3)) \wedge$$
$$((t_{1,2} \geq t_{2,2} + 1) \vee (t_{2,2} \geq t_{1,2} + 1)) \wedge$$
$$((t_{1,2} \geq t_{3,2} + 3) \vee (t_{3,2} \geq t_{1,2} + 1)) \wedge$$
$$((t_{2,2} \geq t_{3,2} + 3) \vee (t_{3,2} \geq t_{2,2} + 1))$$

- 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: [Moura&Bjorner'11]
  - $n$ 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:

| $d_{i,j}$ | Machine 1 | Machine 2 |
|---|---|---|
| Job 1 | 2 | 1 |
| Job 2 | 3 | 1 |
| Job 3 | 2 | 3 |

with $max = 8$



## A scheduling example (Cont.)

- An SMT model for job-shop scheduling:
  - $t_{i,j}$: start time for task $j$ of job $i$
  - Example:

| $d_{i,j}$ | Machine 1 | Machine 2 |
|---|---|---|
| Job 1 | 2 | 1 |
| Job 2 | 3 | 1 |
| Job 3 | 2 | 3 |

with max = 8

  - Formulation:

$$
\begin{array}{l}
(t_{1,1} \geq 0) \wedge (t_{1,2} \geq t_{1,1} + 2) \wedge (t_{1,2} + 1 \leq 8) \wedge \\
(t_{2,1} \geq 0) \wedge (t_{2,2} \geq t_{1,1} + 3) \wedge (t_{2,2} + 1 \leq 8) \wedge \\
(t_{3,1} \geq 0) \wedge (t_{3,2} \geq t_{1,1} + 2) \wedge (t_{3,2} + 3 \leq 8) \wedge \\
((t_{1,1} \geq t_{2,1} + 3) \vee (t_{2,1} \geq t_{1,1} + 2)) \wedge \\
((t_{1,1} \geq t_{3,1} + 2) \vee (t_{3,1} \geq t_{1,1} + 2)) \wedge \\
((t_{2,1} \geq t_{3,1} + 2) \vee (t_{3,1} \geq t_{2,1} + 3)) \wedge \\
((t_{1,2} \geq t_{2,2} + 1) \vee (t_{2,2} \geq t_{1,2} + 1)) \wedge \\
((t_{1,2} \geq t_{3,2} + 3) \vee (t_{3,2} \geq t_{1,2} + 1)) \wedge \\
((t_{2,2} \geq t_{3,2} + 3) \vee (t_{3,2} \geq t_{2,2} + 1))
\end{array}
$$

  - Integer difference logic with Boolean structure
    - ★ Model: $t_{1,1} = 5; t_{1,2} = 7; t_{2,1} = 2; t_{2,2} = 6; t_{3,1} = 0; t_{3,2} = 7$

## Software testing with symbolic execution

- Example C program: [E.g. Moura&Bjorner'11]

```
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

```
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

$$
\begin{array}{ll}
(m_0 = x_0 \,\%\, y_0) & \wedge \\
\neg(m_0 = 0) & \wedge \\
(x_1 = y_0) & \wedge \\
(y_1 = m_0) & \wedge \\
(m_1 = x_1 \,\%\, y_0) & \wedge \\
(m_1 = 0) &
\end{array}
$$

- **Note**: SSA denotes static single assignment form

## Software testing with symbolic execution (Cont.)

- 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:

$$x_0 = 2; y_0 = 4; m_0 = 2;$$
$$x_1 = 4; y_1 = 2; m_1 = 0;$$

- 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?

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

  - Formula is unsatisfiable

- And this formula?

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

  - 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
    - ★ $x \bowtie k, \bowtie \in \{<, >, \leq, \geq, =\}$
  - Difference Logic
    - ★ $x - y \bowtie k, \bowtie \in \{<, >, \leq, \geq, =\}$
  - UTVPI (Unit Two-Variable Per Inequality)
    - ★ $\pm x \pm y \bowtie k, \bowtie \in \{<, >, \leq, \geq, =\}$
  - Linear Arithmetic
    - ★ $\sum a_i x_i \bowtie k, \bowtie \in \{<, >, \leq, \geq, =\}$
  - Non-Linear Arithmetic
    - ★ E.g. $3xy - 4x^2z - 4y \leq 10$



## Other Theories

- Equality with Uninterpreted Functions (EUF)
- (Restricted) (linear/non-linear) arithmetic over the integers / reals
- Bit vectors

- Arrays
- Pointer logic

- Quantified fragments

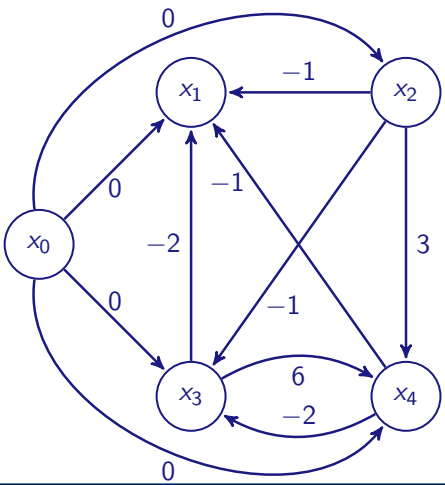
## Integer difference logic

- 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



## Integer difference logic (Cont.)

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

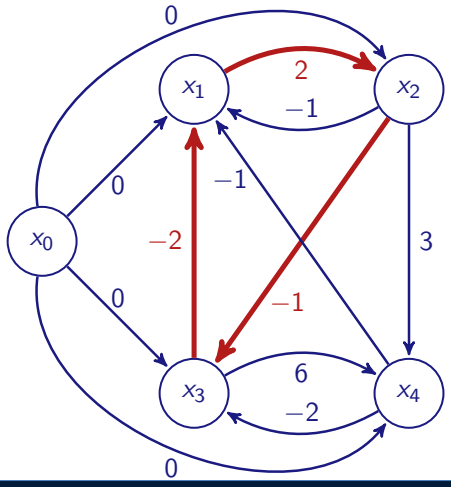



Satisfiable:

$$x_1 = -4$$
$$x_2 = 0$$
$$x_3 = -2$$
$$x_4 = 0$$

## Integer difference logic (Cont.)

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




Unsatisfiable





## 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:

$$g(a) = c \wedge (f(g(a)) \neq f(c) \vee g(a) = d) \wedge c \neq d$$

- Represent Boolean structure as CNF formula:

$$(x) \wedge (\neg y \vee z) \wedge (\neg w)$$

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

| SAT Outcome | Model/Core | EUF Outcome | Explanation clause (sent to SAT solver) |
|---|---|---|---|
| SAT | $\{x, \neg y, \neg w\}$ | UNSAT | $(\neg x \vee y)$ |
| SAT | $\{x, y, z, \neg w\}$ | UNSAT | $(\neg x \vee \neg z \vee w)$ |
| UNSAT | $(x) \wedge (\neg y \vee z) \wedge (\neg x \vee y) \wedge (\neg w) \wedge (\neg x \vee \neg z \vee w)$ | | |

## Lazy approaches – another example

- Example SMT formula:

$$((x_4 - x_2 \leq 3) \vee (x_4 - x_3 \geq 5)) \wedge (x_4 - x_3 \leq 6) \wedge$$
$$(x_1 - x_2 \leq -1) \wedge (x_1 - x_3 \leq -2) \wedge (x_1 - x_4 \leq -1) \wedge (x_2 - x_1 \leq 2) \wedge$$
$$(x_3 - x_2 \leq -1) \wedge ((x_3 - x_4 \leq -2) \vee (x_4 - x_3 \geq 2))$$

- Represent Boolean structure as CNF formula:

$$(a \vee b) \wedge (c) \wedge (d) \wedge (e) \wedge (f) \wedge (g) \wedge (h) \wedge (i \vee j)$$

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

| SAT Outcome | Model/Core | IDL Outcome | Explanation clause (sent to SAT solver) |
|---|---|---|---|
| SAT | $\{a, c, \ldots, h, i\}$ | UNSAT | $(\neg e \vee \neg g \vee \neg h)$ |
| UNSAT | $(e) \wedge (g) \wedge (h) \wedge (\neg e \vee \neg g \vee \neg h)$ | | |



## 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.

## Lazy Approaches – Key Techniques

- 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



## DPLL(T)

- What is DPLL($T$)?

$$\text{DPLL}(T) = \text{DPLL}(X) + T\text{-Solver}$$

- 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

# SMT vs. Hybrid SAT/CP

- 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