Algorithms for Computational Logic

Overview of Satisfiability Modulo Theories

based on slides of João Marques-Silva and Mikoláš Janota

IST, ULisboa

Context

- Propositional satisfiability (SAT)
  - Modern SAT algorithms – CDCL
- Propositional encodings
  - Cardinality constraints, PB constraints, etc.

Context

- Propositional satisfiability (SAT)
  - Modern SAT algorithms – CDCL
- Propositional encodings
  - Cardinality constraints, PB constraints, etc.
- Overconstrained problems
  - Optimization
  - Minimal sets

Context

- Propositional satisfiability (SAT)
  - Modern SAT algorithms – CDCL
- Propositional encodings
  - Cardinality constraints, PB constraints, etc.
- Overconstrained problems
  - Optimization
  - Minimal sets
- Duality

Context

- Propositional satisfiability (SAT)
  - Modern SAT algorithms – CDCL
- Propositional encodings
  - Cardinality constraints, PB constraints, etc.
- Overconstrained problems
  - Optimization
  - Minimal sets
- Duality
- Satisfiability Modulo Theories (SMT)
- Other topics:
  - Answer Set Programming (ASP)

Outline

Outline

What is SMT?

Application Examples

- Encoding a scheduling problem
- Encoding symbolic execution

A Glimpse of SMT

Eager Encodings

Lazy Solving

FOL and Terminology

- FOL: functions, predicates, quantifiers, e.g. $(\forall x \exists y)(f(x, y))$

FOL and Terminology

- FOL: functions, predicates, quantifiers, e.g. $(\forall x \exists y)(f(x, y))$
- FOL is generally undecidable but it is semi-decidable.

FOL and Terminology

- FOL: functions, predicates, quantifiers, e.g. $(\forall x \exists y)(f(x, y))$
- FOL is generally undecidable but it is semi-decidable.
- SMT: satisfiability of fragments of FOL

FOL and Terminology

- FOL: functions, predicates, quantifiers, e.g. $(\forall x \exists y)(f(x,y))$
- FOL is generally undecidable but it is semi-decidable.
- SMT: satisfiability of fragments of FOL
- Example fragment: *integer linear arithmetic*

FOL and Terminology

- FOL: functions, predicates, quantifiers, e.g. $(\forall x \exists y)(f(x, y))$
- FOL is generally undecidable but it is semi-decidable.
- SMT: satisfiability of fragments of FOL
- Example fragment: *integer linear arithmetic*
- these fragments are called theories

FOL and Terminology

- FOL: functions, predicates, quantifiers, e.g. $(\forall x \exists y)(f(x, y))$
- FOL is generally undecidable but it is semi-decidable.
- SMT: satisfiability of fragments of FOL
- Example fragment: *integer linear arithmetic*
- these fragments are called theories
- e.g. $(d - c = 1) \wedge ((d < 5) \vee (c > 10))$ is satisfiable with example solutions $\{c = 11, d = 12\}, \{c = 1, d = 2\}$.

FOL and Terminology

- FOL: functions, predicates, quantifiers, e.g. $(\forall x \exists y)(f(x,y))$
- FOL is generally undecidable but it is semi-decidable.
- SMT: satisfiability of fragments of FOL
- Example fragment: *integer linear arithmetic*
- these fragments are called theories
- e.g. $(d - c = 1) \wedge ((d < 5) \vee (c > 10))$ is satisfiable with example solutions $\{c = 11, d = 12\}, \{c = 1, d = 2\}$.
- interpreted functions, predicates, and constants, e.g. $+, -, =, <, 5$.

FOL and Terminology

- FOL: functions, predicates, quantifiers, e.g. $(\forall x \exists y)(f(x, y))$
- FOL is generally undecidable but it is semi-decidable.
- SMT: satisfiability of fragments of FOL
- Example fragment: *integer linear arithmetic*
- these fragments are called theories
- e.g. $(d - c = 1) \wedge ((d < 5) \vee (c > 10))$ is satisfiable with example solutions $\{c = 11, d = 12\}, \{c = 1, d = 2\}$.
- interpreted functions, predicates, and constants, e.g. $+, -, =, <, 5$.
- uninterpreted functions, predicates, and constants, e.g. f, c.

FOL and Terminology

- FOL: functions, predicates, quantifiers, e.g. $(\forall x \exists y)(f(x, y))$
- FOL is generally undecidable but it is semi-decidable.
- SMT: satisfiability of fragments of FOL
- Example fragment: *integer linear arithmetic*
- these fragments are called theories
- e.g. $(d - c = 1) \wedge ((d < 5) \vee (c > 10))$ is satisfiable with example solutions $\{c = 11, d = 12\}, \{c = 1, d = 2\}$.
- interpreted functions, predicates, and constants, e.g. $+, -, =, <, 5$.
- uninterpreted functions, predicates, and constants, e.g. f, c.
- e.g. $f(c) > c$ is satisfied by $\{f(x) = x + 1, c = 100\}$

FOL and Terminology

- FOL: functions, predicates, quantifiers, e.g. $(\forall x \exists y)(f(x, y))$
- FOL is generally undecidable but it is semi-decidable.
- SMT: satisfiability of fragments of FOL
- Example fragment: *integer linear arithmetic*
- these fragments are called theories
- e.g. $(d - c = 1) \wedge ((d < 5) \vee (c > 10))$ is satisfiable with example solutions $\{c = 11, d = 12\}, \{c = 1, d = 2\}$.
- interpreted functions, predicates, and constants, e.g. $+, -, =, <, 5$.
- uninterpreted functions, predicates, and constants, e.g. f, c.
- e.g. $f(c) > c$ is satisfied by $\{f(x) = x + 1, c = 100\}$
- term: $f(c)$, atom: $f(c) > d$, literal: $\neg(f(c) > d)$, proposition: $(c < d) \vee (d > z)$

FOL and Terminology

- FOL: functions, predicates, quantifiers, e.g. $(\forall x \exists y)(f(x, y))$
- FOL is generally undecidable but it is semi-decidable.
- SMT: satisfiability of fragments of FOL
- Example fragment: *integer linear arithmetic*
- these fragments are called theories
- e.g. $(d - c = 1) \wedge ((d < 5) \vee (c > 10))$ is satisfiable with example solutions $\{c = 11, d = 12\}, \{c = 1, d = 2\}$.
- interpreted functions, predicates, and constants, e.g. $+, -, =, <, 5$.
- uninterpreted functions, predicates, and constants, e.g. f, c.
- e.g. $f(c) > c$ is satisfied by $\{f(x) = x + 1, c = 100\}$
- term: $f(c)$, atom: $f(c) > d$, literal: $\neg(f(c) > d)$, proposition: $(c < d) \vee (d > z)$
- congruence: $(t_1 = s_1 \wedge t_2 = s_2) \rightarrow f(t_1, t_2) = f(s_1, s_2)$

An example

- All variables integer

An example

- All variables integer
- Solve:

$$
\begin{aligned}
&((x_4 - x_2 \leq 3) \vee (x_4 - x_3 \geq 5)) \wedge (x_4 - x_3 \leq 6) \wedge \\
&(x_1 - x_2 \leq -1) \wedge (x_1 - x_3 \leq -2) \wedge (x_1 - x_4 \leq -1) \wedge \\
&(x_2 - x_1 \leq 2) \wedge (x_3 - x_2 \leq -1) \wedge \\
&((x_3 - x_4 \leq -2) \vee (x_4 - x_3 \geq 2))
\end{aligned}
$$

An example

- All variables integer
- Solve:

$$
\begin{aligned}
&((x_4 - x_2 \leq 3) \vee (x_4 - x_3 \geq 5)) \wedge (x_4 - x_3 \leq 6) \wedge \\
&(x_1 - x_2 \leq -1) \wedge (x_1 - x_3 \leq -2) \wedge (x_1 - x_4 \leq -1) \wedge \\
&(x_2 - x_1 \leq 2) \wedge (x_3 - x_2 \leq -1) \wedge \\
&((x_3 - x_4 \leq -2) \vee (x_4 - x_3 \geq 2))
\end{aligned}
$$

- Integer difference logic (with Boolean structure)

An example

- All variables integer
- Solve:

$$
\begin{aligned}
&((x_4 - x_2 \leq 3) \vee (x_4 - x_3 \geq 5)) \wedge (x_4 - x_3 \leq 6) \wedge \\
&(x_1 - x_2 \leq -1) \wedge (x_1 - x_3 \leq -2) \wedge (x_1 - x_4 \leq -1) \wedge \\
&(x_2 - x_1 \leq 2) \wedge (x_3 - x_2 \leq -1) \wedge \\
&((x_3 - x_4 \leq -2) \vee (x_4 - x_3 \geq 2))
\end{aligned}
$$

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

Outline

A scheduling example

- Standard job-shop scheduling formulation: [Moura&Bjorner'11]
  - n jobs, each composed of m tasks to be performed chronologically on m machines
    - $d_{i,j}$: duration of task j for job i

A scheduling example

- Standard job-shop scheduling formulation: [Moura&Bjorner'11]
  - n jobs, each composed of m tasks to be performed chronologically 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*

A scheduling example

- Standard job-shop scheduling formulation: [Moura&Bjorner'11]
  - n jobs, each composed of m tasks to be performed chronologically 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 (task has number of time units and assigned machine):

$d_{i,j}$	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

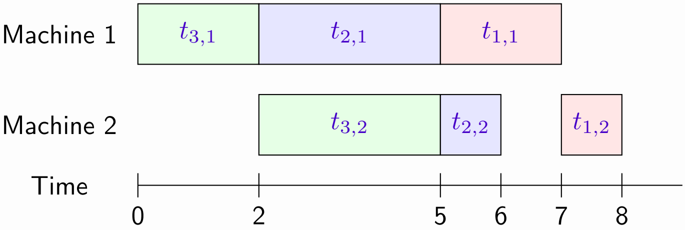
  with $\mathsf{max} = 8$

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 $\mathsf{max} = 8$



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 $\mathsf{max} = 8$
  - Formulation:

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

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 $\mathsf{max} = 8$

  - Formulation:

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

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

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;
  }
}
```

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 exactly twice?
  - If so, which inputs allow this to happen?

Software testing with symbolic execution (Cont.)

- Problem formulation as SMT formula:

Software testing with symbolic execution (Cont.)

- Problem formulation as SMT formula:

First build SSA Program
by loop unrolling
(now variables do not change)

```
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);
}
```

Software testing with symbolic execution (Cont.)

- Problem formulation as SMT formula:

First build SSA Program
by loop unrolling
(now variables do not change)

```
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

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

Software testing with symbolic execution (Cont.)

- Problem formulation as SMT formula:

First build SSA Program
by loop unrolling
(now variables do not change)

```
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

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

- **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;
  }
}
```

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)

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

Remark on SW Verification and Testing

- SW verification & testing one of the main driving forces behind SMT
- Testing: Increasing test coverage
  (which input reaches a given path?)
- Verification: proving correctness, absence of crashes, security, etc.
- Implications for security: prove that your program/protocol is secure

Outline

Example Theories – EUF

- Equality with Uninterpreted Functions (EUF)
- Is this formula satisfiable?

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

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

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?

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

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

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

- **Useful**: 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

- 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. $3\,x\,y - 4\,x^2\,z - 4\,y \leq 10$

Other Theories

- Bit vectors

Other Theories

- Bit vectors
- Arrays

Other Theories

- Bit vectors
- Arrays
- Pointer logic

Other Theories

- Bit vectors
- Arrays
- Pointer logic
- Quantified theories, e.g. quantified linear integer arithmetic is decidable

Algorithms for SMT

- Eager approaches

Algorithms for SMT

- Eager approaches
  - Encode problem to CNF and solve with SAT solver (1 SAT call)

Algorithms for SMT

- Eager approaches
  - Encode problem to CNF and solve with SAT solver (1 SAT call)

- Lazy approaches

Algorithms for SMT

- Eager approaches
  - Encode problem to CNF and solve with SAT solver (1 SAT call)

- Lazy approaches
  - Embed SAT solver with theory solver(s) (many SAT calls); theory solvers are responsible for deciding conjunctions of literals

Outline

Eager Approaches (among others)

- Bounded integers encoded as Boolean variables (log variables)

Eager Approaches (among others)

- Bounded integers encoded as Boolean variables (log variables)
- Encode AtMost-k, AtLeast-k, and pseudo-Boolean constraints to CNF.

Eager Approaches (among others)

- Bounded integers encoded as Boolean variables (log variables)
- Encode AtMost-k, AtLeast-k, and pseudo-Boolean constraints to CNF.
- **Recall:** Can encode arbitrary propositional constraints to CNF.

Eager Approaches (among others)

- Bounded integers encoded as Boolean variables (log variables)
- Encode AtMost-k, AtLeast-k, and pseudo-Boolean constraints to CNF.
- **Recall:** Can encode arbitrary propositional constraints to CNF.
- Terms replaced by constants and encode congruence explicitly

Eager Approaches (among others)

- Bounded integers encoded as Boolean variables (log variables)
- Encode AtMost-k, AtLeast-k, and pseudo-Boolean constraints to CNF.
- **Recall:** Can encode arbitrary propositional constraints to CNF.
- Terms replaced by constants and encode congruence explicitly
  - E.g. replace $f(a), f(b), f(c)$ with A, B, C

Eager Approaches (among others)

- Bounded integers encoded as Boolean variables (log variables)
- Encode AtMost-k, AtLeast-k, and pseudo-Boolean constraints to CNF.
- **Recall:** Can encode arbitrary propositional constraints to CNF.
- Terms replaced by constants and encode congruence explicitly
  - E.g. replace $f(a), f(b), f(c)$ with A, B, C
  - Generate implications capturing congruence.

$$a = b \rightarrow A = B$$
$$a = c \rightarrow A = C$$
$$b = c \rightarrow B = C$$

Eager Approaches (among others)

- Bounded integers encoded as Boolean variables (log variables)
- Encode AtMost-k, AtLeast-k, and pseudo-Boolean constraints to CNF.
- **Recall:** Can encode arbitrary propositional constraints to CNF.
- Terms replaced by constants and encode congruence explicitly
  - E.g. replace $f(a), f(b), f(c)$ with A, B, C
  - Generate implications capturing congruence.

$$
\begin{gathered}
a = b \rightarrow A = B \\
a = c \rightarrow A = C \\
b = c \rightarrow B = C
\end{gathered}
$$

- Formula with only equality and n constants is satisfiable iff it has a model of size at most n. (small model property). [E.g. Pnueli et al.'02]. Constants encoded as bounded integers $1..n$.

Finite Models with Booleans

- Finding a model of finite size n can be encoded as SAT.

Finite Models with Booleans

- Finding a model of finite size n can be encoded as SAT.
- log-encoding: elements of the universe are represented by $k = \lceil \log_2 n \rceil$ Boolean variables.

Finite Models with Booleans

- Finding a model of finite size n can be encoded as SAT.
- log-encoding: elements of the universe are represented by $k = \lceil \log_2 n \rceil$ Boolean variables.
- equality: $(a_k, \ldots, a_0) = (b_k, \ldots, b_0) \rightsquigarrow \bigwedge_{i \in 0..k} (a_i \Leftrightarrow b_i)$

Finite Models with Booleans

- Finding a model of finite size n can be encoded as SAT.
- log-encoding: elements of the universe are represented by $k = \lceil \log_2 n \rceil$ Boolean variables.
- equality: $(a_k, \ldots, a_0) = (b_k, \ldots, b_0) \rightsquigarrow \bigwedge_{i \in 0..k} (a_i \Leftrightarrow b_i)$
- inequality: $(a_k, \ldots, a_0) < (b_k, \ldots, b_0) \rightsquigarrow$
  $(\neg a_k \wedge b_k) \vee ((a_k \Leftrightarrow b_k) \wedge (a_{k-1}, \ldots, a_0) < (b_{k-1}, \ldots, b_0))$

Finite Models with Booleans

- Finding a model of finite size n can be encoded as SAT.
- log-encoding: elements of the universe are represented by $k = \lceil \log_2 n \rceil$ Boolean variables.
- equality: $(a_k, \ldots, a_0) = (b_k, \ldots, b_0) \rightsquigarrow \bigwedge_{i \in 0..k} (a_i \Leftrightarrow b_i)$
- inequality: $(a_k, \ldots, a_0) < (b_k, \ldots, b_0) \rightsquigarrow$ $(\neg a_k \wedge b_k) \vee ((a_k \Leftrightarrow b_k) \wedge (a_{k-1}, \ldots, a_0) < (b_{k-1}, \ldots, b_0))$
- Other operations can be encoded, e.g. summation and multiplication by school method.

Finite Models with Booleans

- Finding a model of finite size n can be encoded as SAT.
- log-encoding: elements of the universe are represented by $k = \lceil \log_2 n \rceil$ Boolean variables.
- equality: $(a_k, \ldots, a_0) = (b_k, \ldots, b_0) \rightsquigarrow \bigwedge_{i \in 0..k} (a_i \Leftrightarrow b_i)$
- inequality: $(a_k, \ldots, a_0) < (b_k, \ldots, b_0) \rightsquigarrow$ $(\neg a_k \wedge b_k) \vee ((a_k \Leftrightarrow b_k) \wedge (a_{k-1}, \ldots, a_0) < (b_{k-1}, \ldots, b_0))$
- Other operations can be encoded, e.g. summation and multiplication by school method.
- unary-encoding: elements are represented by n Boolean variables with $a_i \Rightarrow a_{i-1}$ for $i \in 1..n$.

Finite Models with Booleans

- Finding a model of finite size n can be encoded as SAT.
- log-encoding: elements of the universe are represented by $k = \lceil \log_2 n \rceil$ Boolean variables.
- equality: $(a_k, \ldots, a_0) = (b_k, \ldots, b_0) \rightsquigarrow \bigwedge_{i \in 0..k}(a_i \Leftrightarrow b_i)$
- inequality: $(a_k, \ldots, a_0) < (b_k, \ldots, b_0) \rightsquigarrow$ $(\neg a_k \wedge b_k) \vee ((a_k \Leftrightarrow b_k) \wedge (a_{k-1}, \ldots, a_0) < (b_{k-1}, \ldots, b_0))$
- Other operations can be encoded, e.g. summation and multiplication by school method.
- unary-encoding: elements are represented by n Boolean variables with $a_i \Rightarrow a_{i-1}$ for $i \in 1..n$.
- equality: $(a_n, \ldots, a_0) = (b_n, \ldots, b_0) \rightsquigarrow \bigwedge_{i \in 0..n}(a_i \Leftrightarrow b_i)$

Finite Models with Booleans

- Finding a model of finite size n can be encoded as SAT.
- log-encoding: elements of the universe are represented by $k = \lceil \log_2 n \rceil$ Boolean variables.
- equality: $(a_k, \ldots, a_0) = (b_k, \ldots, b_0) \rightsquigarrow \bigwedge_{i \in 0..k} (a_i \Leftrightarrow b_i)$
- inequality: $(a_k, \ldots, a_0) < (b_k, \ldots, b_0) \rightsquigarrow$ $(\neg a_k \wedge b_k) \vee ((a_k \Leftrightarrow b_k) \wedge (a_{k-1}, \ldots, a_0) < (b_{k-1}, \ldots, b_0))$
- Other operations can be encoded, e.g. summation and multiplication by school method.
- unary-encoding: elements are represented by n Boolean variables with $a_i \Rightarrow a_{i-1}$ for $i \in 1..n$.
- equality: $(a_n, \ldots, a_0) = (b_n, \ldots, b_0) \rightsquigarrow \bigwedge_{i \in 0..n} (a_i \Leftrightarrow b_i)$
- inequality: $(a_n, \ldots, a_0) < (b_n, \ldots, b_0) \rightsquigarrow \bigvee_{i \in 0..n} (\neg a_i \wedge b_i)$

Finite Models with Booleans

- Finding a model of finite size n can be encoded as SAT.
- log-encoding: elements of the universe are represented by $k = \lceil \log_2 n \rceil$ Boolean variables.
- equality: $(a_k, \ldots, a_0) = (b_k, \ldots, b_0) \rightsquigarrow \bigwedge_{i \in 0..k} (a_i \Leftrightarrow b_i)$
- inequality: $(a_k, \ldots, a_0) < (b_k, \ldots, b_0) \rightsquigarrow$ $(\neg a_k \wedge b_k) \vee ((a_k \Leftrightarrow b_k) \wedge (a_{k-1}, \ldots, a_0) < (b_{k-1}, \ldots, b_0))$
- Other operations can be encoded, e.g. summation and multiplication by school method.
- unary-encoding: elements are represented by n Boolean variables with $a_i \Rightarrow a_{i-1}$ for $i \in 1..n$.
- equality: $(a_n, \ldots, a_0) = (b_n, \ldots, b_0) \rightsquigarrow \bigwedge_{i \in 0..n} (a_i \Leftrightarrow b_i)$
- inequality: $(a_n, \ldots, a_0) < (b_n, \ldots, b_0) \rightsquigarrow \bigvee_{i \in 0..n} (\neg a_i \wedge b_i)$
- **Note:** log-encoding smaller but unary tends to give better behavior in SAT solvers.

Small Model Property

- **Idea**: in some theories, satisfiability can be encoded into finding a finite model.

Small Model Property

- **Idea**: in some theories, satisfiability can be encoded into finding a finite model.
- A formula with only equality and n variables is satisfiable iff it has a model of at most size n.

Small Model Property

- **Idea**: in some theories, satisfiability can be encoded into finding a finite model.
- A formula with only equality and n variables is satisfiable iff it has a model of at most size n.
- **Why?** For any larger model $\mathcal{A}'$ construct $\mathcal{A}$ by considering only elements used to interpret the variables in the formula.

Small Model Property

- **Idea**: in some theories, satisfiability can be encoded into finding a finite model.
- A formula with only equality and n variables is satisfiable iff it has a model of at most size n.
- **Why?** For any larger model $\mathcal{A}'$ construct $\mathcal{A}$ by considering only elements used to interpret the variables in the formula.
- $(x_1 = x_2) \vee (x_1 = x_3) \wedge (x_3 \neq x_2)$ is decided by looking for a model up to size 3.

Small Model Property: IDL

- What about Integer Difference Logic?

Small Model Property: IDL

- What about Integer Difference Logic?
- Let s be the sum of absolute values of weights and n number of variables.

Small Model Property: IDL

- What about Integer Difference Logic?
- Let s be the sum of absolute values of weights and n number of variables.
- It is sufficient to look for a model with integers in $0..(s + n)$.

Small Model Property: IDL

- What about Integer Difference Logic?
- Let s be the sum of absolute values of weights and n number of variables.
- It is sufficient to look for a model with integers in $0..(s+n)$.
- **Why?**

Small Model Property: IDL

- What about Integer Difference Logic?
- Let s be the sum of absolute values of weights and n number of variables.
- It is sufficient to look for a model with integers in $0..(s+n)$.
- **Why?**
  - Any solution can be shifted so it starts at 0.

Small Model Property: IDL

- What about Integer Difference Logic?
- Let s be the sum of absolute values of weights and n number of variables.
- It is sufficient to look for a model with integers in $0..(s+n)$.
- **Why?**
  - Any solution can be shifted so it starts at 0.
  - "Compress" any solution while preserving the satisfiability of the same literals.

IDL: Compressing Model Example

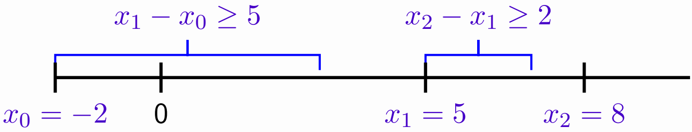
- $\neg(x_1 - x_0 \leq 4) \wedge (x_1 - x_2 \leq -2)$,
  model: $x_0 = -2$, $x_1 = 5$, $x_2 = 8$

IDL: Compressing Model Example

- $\neg(x_1 - x_0 \leq 4) \wedge (x_1 - x_2 \leq -2)$,
  model: $x_0 = -2$, $x_1 = 5$, $x_2 = 8$
- rewrite as: $x_1 - x_0 \geq 5$, $x_2 - x_1 \geq 2$

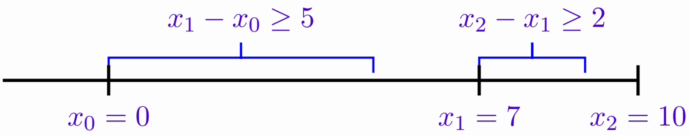
IDL: Compressing Model Example

- $\neg(x_1 - x_0 \leq 4) \wedge (x_1 - x_2 \leq -2)$,
  model: $x_0 = -2$, $x_1 = 5$, $x_2 = 8$
- rewrite as: $x_1 - x_0 \geq 5$, $x_2 - x_1 \geq 2$



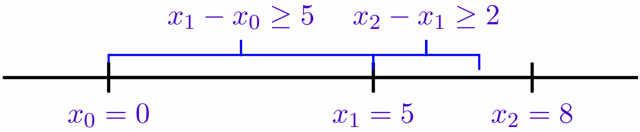
IDL: Compressing Model Example

- $\neg(x_1 - x_0 \leq 4) \wedge (x_1 - x_2 \leq -2)$,
  model: $x_0 = -2$, $x_1 = 5$, $x_2 = 8$
- rewrite as: $x_1 - x_0 \geq 5$, $x_2 - x_1 \geq 2$



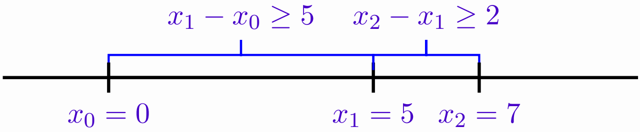
IDL: Compressing Model Example

- $\neg(x_1 - x_0 \leq 4) \wedge (x_1 - x_2 \leq -2)$,
  model: $x_0 = -2$, $x_1 = 5$, $x_2 = 8$
- rewrite as: $x_1 - x_0 \geq 5$, $x_2 - x_1 \geq 2$



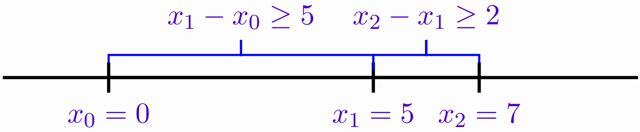
IDL: Compressing Model Example

- $\neg(x_1 - x_0 \leq 4) \wedge (x_1 - x_2 \leq -2)$,
  model: $x_0 = -2$, $x_1 = 5$, $x_2 = 8$
- rewrite as: $x_1 - x_0 \geq 5$, $x_2 - x_1 \geq 2$



IDL: Compressing Model Example

- $\neg(x_1 - x_0 \leq 4) \wedge (x_1 - x_2 \leq -2)$,
  model: $x_0 = -2$, $x_1 = 5$, $x_2 = 8$
- rewrite as: $x_1 - x_0 \geq 5$, $x_2 - x_1 \geq 2$



- Max number: $0 + 5 + 2$
- Note that: $5 = 4 + 1$, that is why we add 1 for each variable since original constraints may be negated

Uninterpreted Functions: Ackerman's Reduction

- For each application $f(\vec{a})$ introduce a fresh variable f_a

Uninterpreted Functions: Ackerman's Reduction

- For each application $f(\vec{a})$ introduce a fresh variable f_a
- For each pair of applications $f(\vec{a})$, $f(\vec{c})$ add the implication $\vec{a} = \vec{c} \Rightarrow f_{\vec{a}} = f_{\vec{c}}$.

Uninterpreted Functions: Ackerman's Reduction

- For each application $f(\vec{a})$ introduce a fresh variable f_a
- For each pair of applications $f(\vec{a})$, $f(\vec{c})$ add the implication $\vec{a} = \vec{c} \Rightarrow f_{\vec{a}} = f_{\vec{c}}$.

Example

Uninterpreted Functions: Ackerman's Reduction

- For each application $f(\vec{a})$ introduce a fresh variable f_a
- For each pair of applications $f(\vec{a})$, $f(\vec{c})$ add the implication $\vec{a} = \vec{c} \Rightarrow f_{\vec{a}} = f_{\vec{c}}$.

Example

- $f(a) \neq f(c) \wedge (a = c \vee f(a) = c)$

Uninterpreted Functions: Ackerman's Reduction

- For each application $f(\vec{a})$ introduce a fresh variable f_a
- For each pair of applications $f(\vec{a})$, $f(\vec{c})$ add the implication $\vec{a} = \vec{c} \Rightarrow f_{\vec{a}} = f_{\vec{c}}$.

Example

- $f(a) \neq f(c) \wedge (a = c \vee f(a) = c)$
- $f_a \neq f_c \wedge (a = c \vee f_a = c) \wedge (a = c \Rightarrow f_a = f_c)$

Uninterpreted Functions: Ackerman's Reduction Contd.

Careful: Need to consider also sub-terms.

Example

Uninterpreted Functions: Ackerman's Reduction Contd.

Careful: Need to consider also sub-terms.

Example

- $f(f(a)) = f(a) \wedge f(f(f(a))) \neq f(a)$

Uninterpreted Functions: Ackerman's Reduction Contd.

Careful: Need to consider also sub-terms.

Example

- $f(f(a)) = f(a) \wedge f(f(f(a))) \neq f(a)$
- Applications: $\{f(a), f(f(a)), f(f(f(a)))\}$

Uninterpreted Functions: Ackerman's Reduction Contd.

Careful: Need to consider also sub-terms.

Example

- $f(f(a)) = f(a) \wedge f(f(f(a))) \neq f(a)$
- Applications: $\{f(a), f(f(a)), f(f(f(a)))\}$
- Reduction:

$$\begin{array}{ll} & f_{f(a)} = f_a \wedge f_{f(f(a))} \neq f_a \\ \wedge & a = f_a \Rightarrow f_a = f_{f(a)} \\ \wedge & a = f_{f(a)} \Rightarrow f_a = f_{f(f(a))} \\ \wedge & f_a = f_{f(a)} \Rightarrow f_{f(a)} = f_{f(f(a))} \end{array}$$

Uninterpreted Functions: Ackerman's Reduction Contd.

Careful: Need to consider also sub-terms.

Example

- $f(f(a)) = f(a) \land f(f(f(a))) \neq f(a)$
- Applications: $\{f(a), f(f(a)), f(f(f(a)))\}$
- Reduction:

$$\begin{array}{rl} & f_{f(a)} = f_a \land f_{f(f(a))} \neq f_a \\ \land & a = f_a \Rightarrow f_a = f_{f(a)} \\ \land & a = f_{f(a)} \Rightarrow f_a = f_{f(f(a))} \\ \land & f_a = f_{f(a)} \Rightarrow f_{f(a)} = f_{f(f(a))} \end{array}$$

- Propagate $f_{f(a)} = f_a$ in last implication $\rightsquigarrow f_{f(a)} = f_{f(f(a))}$

Uninterpreted Functions: Ackerman's Reduction Contd.

Careful: Need to consider also sub-terms.

Example

- $f(f(a)) = f(a) \wedge f(f(f(a))) \neq f(a)$
- Applications: $\{f(a), f(f(a)), f(f(f(a)))\}$
- Reduction:

$$\begin{array}{ll} & f_{f(a)} = f_a \wedge f_{f(f(a))} \neq f_a \\ \wedge & a = f_a \Rightarrow f_a = f_{f(a)} \\ \wedge & a = f_{f(a)} \Rightarrow f_a = f_{f(f(a))} \\ \wedge & f_a = f_{f(a)} \Rightarrow f_{f(a)} = f_{f(f(a))} \end{array}$$

- Propagate $f_{f(a)} = f_a$ in last implication $\rightsquigarrow f_{f(a)} = f_{f(f(a))}$
- Transitivity of $f_{f(a)} = f_a$ and $f_{f(a)} = f_{f(f(a))} \rightsquigarrow f_a = f_{f(f(a))}$

Uninterpreted Functions: Ackerman's Reduction Contd.

Careful: Need to consider also sub-terms.

Example

- $f(f(a)) = f(a) \wedge f(f(f(a))) \neq f(a)$
- Applications: $\{f(a), f(f(a)), f(f(f(a)))\}$
- Reduction:

$$
\begin{array}{ll}
& f_{f(a)} = f_a \wedge f_{f(f(a))} \neq f_a \\
\wedge & a = f_a \Rightarrow f_a = f_{f(a)} \\
\wedge & a = f_{f(a)} \Rightarrow f_a = f_{f(f(a))} \\
\wedge & f_a = f_{f(a)} \Rightarrow f_{f(a)} = f_{f(f(a))}
\end{array}
$$

- Propagate $f_{f(a)} = f_a$ in last implication $\rightsquigarrow f_{f(a)} = f_{f(f(a))}$
- Transitivity of $f_{f(a)} = f_a$ and $f_{f(a)} = f_{f(f(a))} \rightsquigarrow f_a = f_{f(f(a))}$
- Contradiction $f_a = f_{f(f(a))}$ and $f_a \neq f_{f(f(a))} \rightsquigarrow$ unsatisfiable

Outline

Lazy solving

- Use SAT for Boolean structure.

Lazy solving

- Use SAT for Boolean structure.
- Use theory solver for conjunctions of literals.

Lazy solving

- Use SAT for Boolean structure.
- Use theory solver for conjunctions of literals.
- Conceptually the following is needed:

Lazy solving

- Use SAT for Boolean structure.
- Use theory solver for conjunctions of literals.
- Conceptually the following is needed:
  - $\mathcal{T}2\mathcal{B}$: abstracts theory formula to Boolean formula, e.g.
    $\mathcal{T}2\mathcal{B}((x < y) \vee \neg(z > y)) \rightsquigarrow e_{x<y} \vee \neg e_{z>y}$

Lazy solving

- Use SAT for Boolean structure.
- Use theory solver for conjunctions of literals.
- Conceptually the following is needed:
  - $\mathcal{T}2\mathcal{B}$: abstracts theory formula to Boolean formula, e.g.
    $\mathcal{T}2\mathcal{B}((x < y) \lor \neg(z > y)) \rightsquigarrow e_{x<y} \lor \neg e_{z>y}$
  - $\mathcal{B}2\mathcal{T}$: converts Boolean literal to theory literal, e.g.
    $\mathcal{B}2\mathcal{T}(\neg e_{z>y}) \rightsquigarrow \neg(z > y)$

Lazy solving

- Use SAT for Boolean structure.
- Use theory solver for conjunctions of literals.
- Conceptually the following is needed:
  - $\mathcal{T}2\mathcal{B}$: abstracts theory formula to Boolean formula, e.g.
    $\mathcal{T}2\mathcal{B}((x < y) \vee \neg(z > y)) \rightsquigarrow e_{x<y} \vee \neg e_{z>y}$
  - $\mathcal{B}2\mathcal{T}$: converts Boolean literal to theory literal, e.g.
    $\mathcal{B}2\mathcal{T}(\neg e_{z>y}) \rightsquigarrow \neg(z > y)$
  - $\mathcal{T}$-SAT$(\mathcal{L})$: theory solver for set of literals $\mathcal{L}$ determines whether $\mathcal{T}$-satisfiable or $\mathcal{T}$-unsatisfiable.

Lazy solving

- Use SAT for Boolean structure.
- Use theory solver for conjunctions of literals.
- Conceptually the following is needed:
  - $\mathcal{T}2\mathcal{B}$: abstracts theory formula to Boolean formula, e.g.
    $\mathcal{T}2\mathcal{B}((x < y) \vee \neg(z > y)) \rightsquigarrow e_{x<y} \vee \neg e_{z>y}$
  - $\mathcal{B}2\mathcal{T}$: converts Boolean literal to theory literal, e.g.
    $\mathcal{B}2\mathcal{T}(\neg e_{z>y}) \rightsquigarrow \neg(z > y)$
  - $\mathcal{T}$-SAT$(\mathcal{L})$: theory solver for set of literals $\mathcal{L}$ determines whether $\mathcal{T}$-satisfiable or $\mathcal{T}$-unsatisfiable.
  - If unsatisfiable, provides explanation $\mathcal{L}' \subseteq \mathcal{L}$ so that $\mathcal{L}'$ is also $\mathcal{T}$-unsatisfiable.

Lazy solving

- Use SAT for Boolean structure.
- Use theory solver for conjunctions of literals.
- Conceptually the following is needed:
  - $\mathcal{T}2\mathcal{B}$: abstracts theory formula to Boolean formula, e.g.
    $\mathcal{T}2\mathcal{B}((x < y) \vee \neg(z > y)) \rightsquigarrow e_{x<y} \vee \neg e_{z>y}$
  - $\mathcal{B}2\mathcal{T}$: converts Boolean literal to theory literal, e.g.
    $\mathcal{B}2\mathcal{T}(\neg e_{z>y}) \rightsquigarrow \neg(z > y)$
  - $\mathcal{T}$-SAT$(\mathcal{L})$: theory solver for set of literals $\mathcal{L}$ determines whether $\mathcal{T}$-satisfiable or $\mathcal{T}$-unsatisfiable.
  - If unsatisfiable, provides explanation $\mathcal{L}' \subseteq \mathcal{L}$ so that $\mathcal{L}'$ is also $\mathcal{T}$-unsatisfiable.
  - SAT(α): SAT solver for formula α determines if SAT or UNSAT. If SAT, provides model as a set of true literals.

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	Bool Model	EUF Outcome	Explanation clause (sent to 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	Bool Model	EUF Outcome	Explanation clause (sent to SAT solver)
SAT	$\{x, \neg y, \neg w\}$	UNSAT	$(\neg x \vee y \vee w)$

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	Bool Model	EUF Outcome	Explanation clause (sent to SAT solver)
SAT	$\{x, \neg y, \neg w\}$	UNSAT	$(\neg x \vee y \vee w)$
SAT	$\{x, y, z, \neg w\}$	UNSAT	$(\neg x \vee \neg y \vee \neg z \vee w)$

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

Lazy solving

input : formula ϕ in theory $\mathcal{T}$
output: truth value

$\alpha \leftarrow \mathcal{T}2\mathcal{B}(\phi)$ // abstract input formula
while true **do**
  $(\mathsf{res}, \tau) \leftarrow \mathsf{SAT}(\alpha)$ // Boolean model
  if res = false **then return** false
  $\mathcal{L} \leftarrow \bigcup_{l \in \tau} \mathcal{B}2\mathcal{T}(l)$ // convert to theory model
  $(\mathsf{res}, \mathcal{L}') \leftarrow \mathcal{T}\text{-}\mathsf{SAT}(\mathcal{L})$ // theory check
  if res = true **then return** true
  $\alpha \leftarrow \alpha \wedge \bigvee_{l \in \mathcal{L}'} \neg \mathcal{T}2\mathcal{B}(l)$ // block explanation
end

Theory Solver: Equality — Congruence closure

- Divide the set of literals $\mathcal{L}$ into positive E and negative D.

Theory Solver: Equality — Congruence closure

- Divide the set of literals $\mathcal{L}$ into positive E and negative D.
- Build a set of all sub-terms S in $\mathcal{L}$.

Theory Solver: Equality — Congruence closure

- Divide the set of literals $\mathcal{L}$ into positive E and negative D.
- Build a set of all sub-terms S in $\mathcal{L}$.
- Build congruence closure as a partitioning of S.

Theory Solver: Equality — Congruence closure

- Divide the set of literals $\mathcal{L}$ into positive E and negative D.
- Build a set of all sub-terms S in $\mathcal{L}$.
- Build congruence closure as a partitioning of S.
  - Put each term $t \in S$ in its own partition.

Theory Solver: Equality — Congruence closure

- Divide the set of literals $\mathcal{L}$ into positive E and negative D.
- Build a set of all sub-terms S in $\mathcal{L}$.
- Build congruence closure as a partitioning of S.
  - Put each term $t \in S$ in its own partition.
  - For each $(s = t) \in E$, merge partitions of s and t.

Theory Solver: Equality — Congruence closure

- Divide the set of literals $\mathcal{L}$ into positive E and negative D.
- Build a set of all sub-terms S in $\mathcal{L}$.
- Build congruence closure as a partitioning of S.
  - Put each term $t \in S$ in its own partition.
  - For each $(s = t) \in E$, merge partitions of s and t.
  - For $s_1, \ldots, s_k$ and $t_1, \ldots, t_k$ s.t. s_i is in the same partition as t_i, merge partitions of $f(s_1, \ldots, s_k)$ and $f(t_1, \ldots, t_k)$.

Theory Solver: Equality — Congruence closure

- Divide the set of literals $\mathcal{L}$ into positive E and negative D.
- Build a set of all sub-terms S in $\mathcal{L}$.
- Build congruence closure as a partitioning of S.
  - Put each term $t \in S$ in its own partition.
  - For each $(s = t) \in E$, merge partitions of s and t.
  - For $s_1, \ldots, s_k$ and $t_1, \ldots, t_k$ s.t. s_i is in the same partition as t_i, merge partitions of $f(s_1, \ldots, s_k)$ and $f(t_1, \ldots, t_k)$.
  - Repeat until no congruence applies.

Theory Solver: Equality — Congruence closure

- Divide the set of literals $\mathcal{L}$ into positive E and negative D.
- Build a set of all sub-terms S in $\mathcal{L}$.
- Build congruence closure as a partitioning of S.
  - Put each term $t \in S$ in its own partition.
  - For each $(s = t) \in E$, merge partitions of s and t.
  - For $s_1, \ldots, s_k$ and $t_1, \ldots, t_k$ s.t. s_i is in the same partition as t_i, merge partitions of $f(s_1, \ldots, s_k)$ and $f(t_1, \ldots, t_k)$.
  - Repeat until no congruence applies.
- If there is a $s \neq t \in D$, s.t. s and t are in the same partition, return unsatisfiable otherwise satisfiable.

Example

$$f(a,b) = a \wedge f(f(a,b),b) \neq a$$

Example

$$f(a,b) = a \wedge f(f(a,b),b) \neq a$$

- Congruence closure algorithm – iteratively merge equivalence classes:

Example

$$f(a,b) = a \wedge f(f(a,b),b) \neq a$$

- Congruence closure algorithm – iteratively merge equivalence classes:

$$\{\{a\},\{b\},\{f(a,b)\},\{f(f(a,b),b)\}\}$$

Example

$$f(a,b) = a \wedge f(f(a,b),b) \neq a$$

- Congruence closure algorithm – iteratively merge equivalence classes:

$$\{\{a\},\{b\},\{f(a,b)\},\{f(f(a,b),b)\}\}$$

$$\{\{a, f(a,b)\},\{b\},\{f(f(a,b),b)\}\}$$

Example

$$f(a,b) = a \wedge f(f(a,b),b) \neq a$$

- Congruence closure algorithm – iteratively merge equivalence classes:

$$\{\{a\},\{b\},\{f(a,b)\},\{f(f(a,b),b)\}\}$$

$$\{\{a,f(a,b)\},\{b\},\{f(f(a,b),b)\}\}$$

$$\{\{a,f(a,b),f(f(a,b),b)\},\{b\}\}$$

Example

$$f(a,b) = a \wedge f(f(a,b),b) \neq a$$

- Congruence closure algorithm – iteratively merge equivalence classes:

$$\{\{a\},\{b\},\{f(a,b)\},\{f(f(a,b),b)\}\}$$
$$\{\{a,f(a,b)\},\{b\},\{f(f(a,b),b)\}\}$$
$$\{\{a,f(a,b),f(f(a,b),b)\},\{b\}\}$$

  - But $f(f(a,b),b) \neq a$.

Example

$$f(a, b) = a \wedge f(f(a, b), b) \neq a$$

- Congruence closure algorithm – iteratively merge equivalence classes:

$$\{\{a\}, \{b\}, \{f(a, b)\}, \{f(f(a, b), b)\}\}$$

$$\{\{a, f(a, b)\}, \{b\}, \{f(f(a, b), b)\}\}$$

$$\{\{a, f(a, b), f(f(a, b), b)\}, \{b\}\}$$

  - But $f(f(a, b), b) \neq a$.
  - Formula is unsatisfiable.

Integer Difference Logic: Theory Solver

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

Integer Difference Logic: Theory Solver

- 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 $x_i - x_j \leq k$

Integer Difference Logic: Theory Solver

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

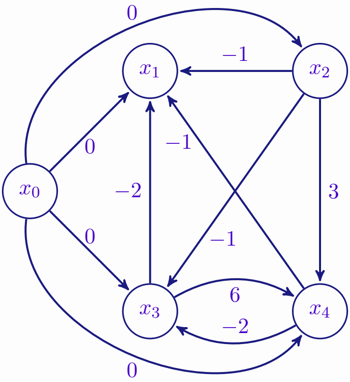
- Algorithm:
  - Add edge between x_j and x_i with weight k, for inequality $x_i - x_j \leq k$
  - Add additional source vertex x_0
  - Add edge from x_0 to x_i, for each other vertex x_i
  - Use Bellman-Ford algorithm to check for negative cycles
    - **Negative cycle:** Elimination of variables in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

Integer Difference Logic (Cont.)

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

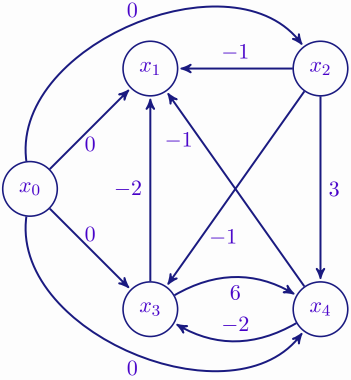
Integer Difference Logic (Cont.)

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



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:

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

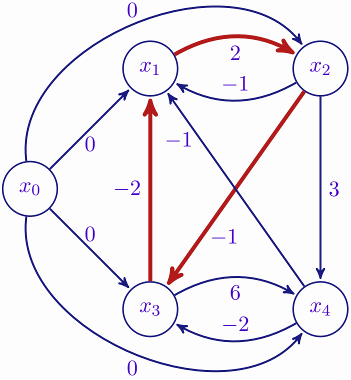
$$x_i = \mathsf{dist}(x_0, x_i)$$

Integer Difference Logic (Cont.)

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

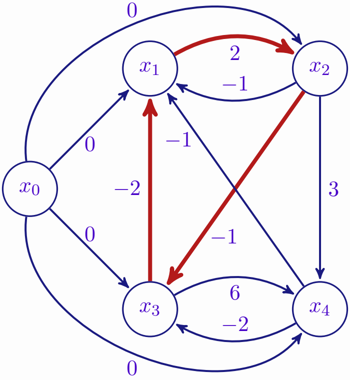
Integer Difference Logic (Cont.)

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



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

$x_2 - x_1 \leq 2$

$x_3 - x_2 \leq -1$

$x_1 - x_3 \leq -2$

Sum: $0 \leq -1$

Note: *Cycle serves as explanation, UNSAT as long as present.*

IDL: Theory Solver

- Convert all literals to positive (i.e. remove negations):

IDL: Theory Solver

- Convert all literals to positive (i.e. remove negations):
  - $\neg(x - y \leq c)$

IDL: Theory Solver

- Convert all literals to positive (i.e. remove negations):
  - $\neg(x - y \leq c)$
  - $\rightsquigarrow x - y > c$

IDL: Theory Solver

- Convert all literals to positive (i.e. remove negations):
  - $\neg(x - y \leq c)$
  - $\rightsquigarrow x - y > c$
  - $\rightsquigarrow y - x < -c$

IDL: Theory Solver

- Convert all literals to positive (i.e. remove negations):
  - $\neg(x - y \leq c)$
  - $\rightsquigarrow x - y > c$
  - $\rightsquigarrow y - x < -c$
  - $\rightsquigarrow y - x \leq -c - 1$

IDL: Theory Solver

- Convert all literals to positive (i.e. remove negations):
  - $\neg(x - y \leq c)$
  - $\rightsquigarrow x - y > c$
  - $\rightsquigarrow y - x < -c$
  - $\rightsquigarrow y - x \leq -c - 1$
- **Note:** Also possible on reals/rationals but care is needed because we cannot directly convert strict inequalities to non-strict ones.

Recap example for IDL

- 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	Boolean model	IDL Outcome	Explanation clause (sent to SAT solver)

Recap example for IDL

- 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	Boolean model	IDL Outcome	Explanation clause (sent to SAT solver)
SAT	$\{a, c, \ldots, h, i\}$	UNSAT	$(\neg e \vee \neg g \vee \neg h)$

Recap example for IDL

- 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	Boolean model	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 are they called lazy?
  - Theory solver called only as needed, to check T-consistency
- Key properties:
  - Avoiding large encodings
  - Modular implementation
    - Easy to add theory solvers
  - Currently, the most efficient algorithms
  - Clear separation between Boolean and theory domains
- 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)?

DPLL(T)

- What is DPLL(T)?

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

DPLL(T)

- What is DPLL(T)?

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

- DPLL(X)
  - SAT solver capable of enumerating models

DPLL(T)

- What is DPLL(T)?

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

- DPLL(X)
  - SAT solver capable of enumerating models
  - Preferable: partial model detection

DPLL(T)

- What is DPLL(T)?

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

- DPLL(X)
  - SAT solver capable of enumerating models
  - Preferable: partial model detection
  - **Cannot use**: pure literals, blocked literals, etc.

DPLL(T)

- What is DPLL(T)?

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

- DPLL(X)
  - SAT solver capable of enumerating models
  - Preferable: partial model detection
  - **Cannot use**: pure literals, blocked literals, etc.
- T-Solver:

DPLL(T)

- What is DPLL(T)?

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

- DPLL(X)
  - SAT solver capable of enumerating models
  - Preferable: partial model detection
  - **Cannot use**: pure literals, blocked literals, etc.
- T-Solver:
  - Checks T-consistency of conjunctions of literals
  - Performs theory propagation
  - Computes explanations of inconsistency
  - **Note:** T-propagation should be incremental and backtrackable