# Decision Procedures and Verification

Martin Blicha

Charles University

26.3.2018

# Satisfiability Modulo Theories (SMT)

# SMT intro

- Decision problem for formulas in first-order logic with respect to some background *theory*
  - SAT: $(a \vee b) \wedge (\neg a \vee \neg b)$
  - SMT: $(x \geq 0) \wedge (y \geq 0) \wedge (x + y < 0)$
- Today we consider only quantifier-free fragments of first-order logic.
- We assume the formulas are quantifier-free and in NNF.

# SMT - Logics

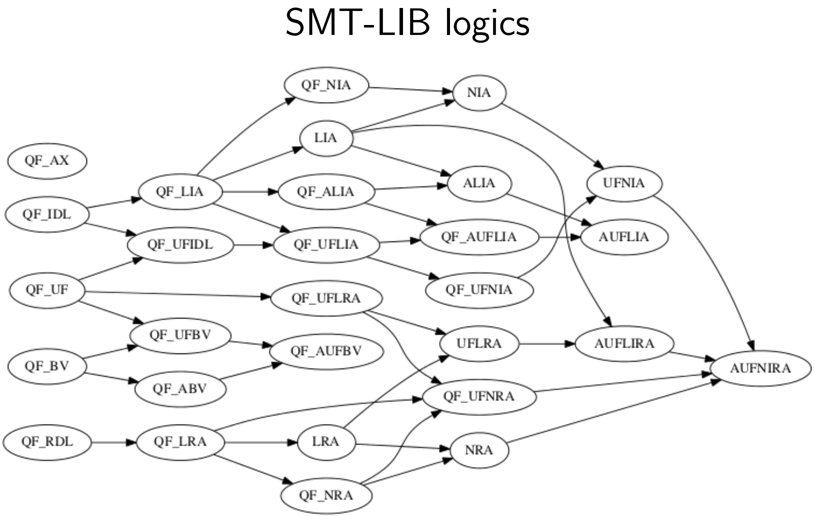



Image source: `http://smtlib.cs.uiowa.edu/logics.shtml`

# Decision procedure for conjunctive fragment

## Conjunctive fragment

Conjunctive fragment of $T$ consists of formulas that are conjunctions of $T$-literals.

- Today we assume we have a decision procedure $DP_T$ for a *conjunctive fragment* of $T$.

# Example: Decision procedure for the theory of equality

### Definition

*Equality graph* for a formula $\varphi$ from a conjunctive fragment of the theory of equality is $G(V, E_=, E_{\neq})$ where nodes from $V$ correspond to variables and edges correspond to equality and inequality literals.

### Decision procedure for the theory of equality

Formula $\varphi$ is unsatisfiable if and only if there exists an inequality edge (from $E_{\neq}$) such that its vertices are connected by a sequence of equality edges (from $E_=$).

# From conjunctive fragment to NNF formulas

## Direct approach

### *Case splitting*

#### Example

$(x_1 = x_2 \lor x_1 = x_3) \land (x_1 = x_2 \lor x_1 = x_4) \land x_1 \neq x_2 \land x_1 \neq x_3 \land x_1 \neq x_4$

- Four cases
  - $x_1 = x_2 \land x_1 = x_2 \land x_1 \neq x_2 \land x_1 \neq x_3 \land x_1 \neq x_4$
  - $x_1 = x_2 \land x_1 = x_4 \land x_1 \neq x_2 \land x_1 \neq x_3 \land x_1 \neq x_4$
  - $x_1 = x_3 \land x_1 = x_2 \land x_1 \neq x_2 \land x_1 \neq x_3 \land x_1 \neq x_4$
  - $x_1 = x_3 \land x_1 = x_4 \land x_1 \neq x_2 \land x_1 \neq x_3 \land x_1 \neq x_4$
- all unsatisfiable $\rightarrow$ the formula is unsatisfiable

- Case splitting is inefficient
  - In general number of cases exponential in the size of the original formula
  - Missed opportunities for learning

# From conjunctive fragment to NNF formulas

## SMT approach

- Idea: utilize the learning capabilities of SAT
  - Combination of $DP_T$ and a SAT solver
  - SAT solver chooses literals to satisfy in order to satisfy the Boolean structure of the formula
  - $DP_T$ checks if the choice is T-satisfiable.
- Modular (and efficient) solution
  - Avoids explicit case splitting

- *Boolean encoder* of an atom $at$ is a unique Boolean variable $e(at)$.
- *Propositional skeleton* of a formula $\varphi$ is denoted as $e(\varphi)$ and is a result of replacing each literal with its Boolean encoder.

### Example

$e(\varphi) := e(x = y) \vee e(x = z)$ for $\varphi := (x = y) \vee (x = z)$

# Integration of a SAT solver and $DP_T$ - intuitively (1)

Given a NNF formula $\varphi = (x = y) \wedge ((y = z \wedge x \neq z) \vee (x = z))$ proceed as follows:

- Compute the propositional skeleton $e(\varphi)$.
- SAT solver will be iteratively queried for satisfiability of a propositional formula **B**
  - At the begining $\mathbf{B} := e(\varphi)$
- Suppose SAT solver returns a satisfying assignment of **B**.
  - $\alpha = \{e(x = y) \mapsto TRUE, e(y = z) \mapsto TRUE, e(x = z) \mapsto FALSE\}$
- Decision procedure $DP_T$ is queried for satisfiability of a conjunction of literals corresponding to the assignments of the Boolean encoders.

# Integration of a SAT solver and $DP_T$ - intuitively (2)

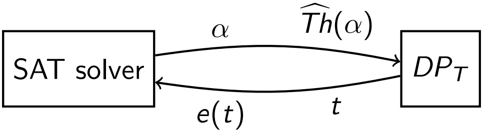
- $DP_T$ is queried for the satisfiability of the conjunction of literals corresponding to the found assignment $\alpha$.
- Let $Th(\alpha)$ denote the set of literals corresponding to the assignment $\alpha$
  - $at \in Th(\alpha)$ if $\alpha(e(at)) = TRUE$
  - $\neg at \in Th(\alpha)$ if $\alpha(e(at)) = FALSE$
- Let $\widehat{Th}(\alpha)$ denote the conjunction of literals in $Th(\alpha)$
- Then $DP_T$ is queried for the satisfiability of $\widehat{Th}(\alpha)$
  - In our case: $\widehat{Th}(\alpha) = (x = y) \wedge (y = z) \wedge \neg(x = z)$

# Integration of a SAT solver and $DP_T$ - intuitively (3)

- If $DP_T$ declares the query satisfiable, the original input formula $\varphi$ is satisfiable.
- If $DP_T$ declares the query unsatisfiable, then $\neg\widehat{Th}(\alpha)$ is a $T$-valid clause and can be added to **B**.
  - **B** and $\mathbf{B} \wedge \neg\widehat{Th}(\alpha)$ are equisatisfiable w.r.t. $T$.
  - $\neg\widehat{Th}(\alpha)$ blocks the current assignment $\alpha$ found by the SAT solver (blocking clause, blocking lemma, $T$-lemma).
  - $\neg\widehat{Th}(\alpha)$ is added to **B** and the process starts again by querying SAT solver.
- Continuing with our example:
  - $DP_T$ declares that $(x = y) \wedge (y = z) \wedge \neg(x = z)$ is unsatisfiable.
  - A new clause is learned at the propositional level: $\neg\widehat{Th}(\alpha) = \neg(e(x = y)) \vee \neg(e(y = z)) \vee e(x = z)$
  - SAT solver is now queried for $\mathbf{B} := \mathbf{B} \wedge \neg\widehat{Th}(\alpha)$.

# Integration of a SAT solver and $DP_T$ - intuitively (3)

- Finishing the example:
  - SAT solver founds an assignment $\alpha = \{e(x = y) \mapsto TRUE, e(y = z) \mapsto TRUE, e(x = z) \mapsto TRUE\}$
  - $DP_T$ checks that $x = y \wedge y = z \wedge x = z$ is indeed satisfiable.
  - The result is that the original input formula $\varphi$ is satisfiable.



# Integration of a SAT solver and $DP_T$ (1)

**Input:** Formula $\varphi$
**Output:** SAT if $\varphi$ is satisfiable, UNSAT if it is unsatisfiable

**procedure** $\text{LAZY-BASIC}(\varphi)$
  $\mathbf{B} \leftarrow e(\varphi)$
  **while** TRUE **do**
    $(\alpha, res) \leftarrow \text{SAT-SOLVER}(\mathbf{B})$
    **if** $res == UNSAT$ **then return** $UNSAT$
    $(t, res) \leftarrow \text{DEDUCTION}(\widehat{Th}(\alpha))$
    **if** $res == SAT$ **then return** $SAT$
    $\mathbf{B} \leftarrow \mathbf{B} \wedge e(t)$

# Integration of a SAT solver and $DP_T$ (2)

- Consider the following three requirements on DEDUCTION:
  1. The formula $t$ is $T$-valid.
  2. The atoms in $t$ are restricted to those appearing in $\varphi$.
  3. The encoding of $t$ contradicts $\alpha$, i.e. $e(t)$ is a blocking clause.
- Requirement 1 guarantees soundness.
- Requirements 2 and 3 guarantee termination.
- The cooperation can be much more efficient if $DP_T$ is integrated directly into the CDCL procedure of the SAT solver.

# Lazy-CDCL

```
procedure LAZY-CDCL(φ)
    ADDCLAUSES(cnf(e(φ)))
    while TRUE do
        while BCP() == conflict do
            backtrack-level ← ANALYZE-CONFLICT()
            if backtrack-level < 0 then return UNSAT
            BACKTRACK(backtrack-level)
        if DECIDE() == NULL then
            //Full satisfying assignment α found
            (t, res) ← DEDUCTION(Th^(α))
            if res == SAT then return SAT
            ADDCLAUSES(e(t))
```

# Improving Lazy-CDLC

- Sending *partial* assignment to DEDUCTION
  - This has two advantages:
    1. theory-level conflicts are detected earlier and stronger lemmas are returned to the SAT solver,
    2. theory can deduce a value for some literals $\Rightarrow$ *theory propagation*.
- Example: Suppose atoms $x \geq 10$ and $x < 0$ are present in $\varphi$
  - Assignment $e(x \geq 10) \mapsto \mathit{TRUE}$ and $e(x < 0) \mapsto \mathit{TRUE}$ cannot be extended to a satisfying assignment.
  - From $e(x \geq 10) \mapsto \mathit{TRUE}$, linear arithmetic can deduce that $x < 0$ is FALSE, so the assignment can be extended by $e(x < 0) \mapsto \mathit{FALSE}$.

# Algorithm DPLL($T$)

```
procedure DPLL(T)(φ)
    AddClauses(cnf(e(φ)))
    while TRUE do
        repeat
            while BCP() == conflict do
                backtrack-level ← Analyze-Conflict()
                if backtrack-level < 0 then return UNSAT
                Backtrack(backtrack-level)
            (t, res) ← Deduction(T̂h(α))
            AddClauses(e(t))
        until t == TRUE
        if α is a full assignment then return SAT
        Decide()
```

# Possible modifications

- Exhaustive theory propagation
  - Propagate all literals implied by $\widehat{Th}(\alpha)$ in $T$.
  - Example: In equality logic, for each unassigned atom $x_i = x_j$ check if the current assignment forms a path in $E_=$. If yes this atom is implied. If current assignment forms a disequality path, then negation is implied.
  - In practice, usually too expensive and only simple, cheap propagations are performed.
- Generating strong lemmas
  - DEDUCTION returns a lemma to block current assignment $\alpha$ (in case of conflict).
  - Stronger lemma block more assignments.
  - Identify those literals that are sufficient to prove the conflict (*unsatisfiable core*).

## Summary

- Decision procedure for quantifier-free theory can be obtained from a combination of SAT solver and a decision procedure for a conjunctive fragment of the theory.
- More effective if $DP_T$
  - can generate strong explanations for conflict;
  - can derive values of yet unassigned literals (theory propagation);
  - is incremental.