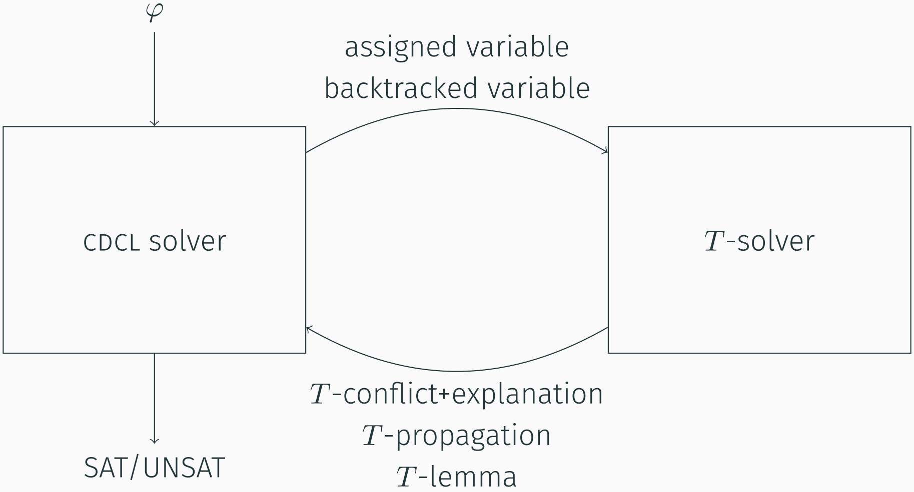
Selected Algorithms for Theory Solvers

IA085: Satisfiability and Automated Reasoning

Martin Jonáš

FI MUNI, Spring 2024

Last time



φ

assigned variable
backtracked variable

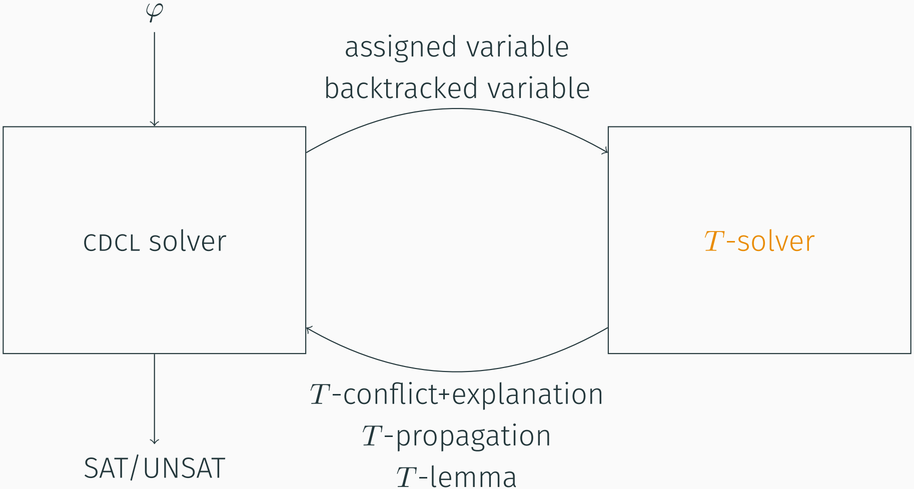
CDCL solver

T-solver

T-conflict+explanation
T-propagation
T-lemma

SAT/UNSAT

Last time


φ

assigned variable
backtracked variable

CDCL solver

T-solver

T-conflict+explanation
T-propagation
T-lemma

SAT/UNSAT

Last time

Possible interface of the T-solver

- `void notifyAtom(lit)`
- `void assignLiteral(lit)`
- `void push()`
- `void pop()`
- `result checkSat()`
- `option<result> checkSat_approx()`
- `list<lit> getConflictReason()`
- `option<lit> getPropagatedLiteral()`
- `list<lit> getExplanation(lit)`

Last time

Possible interface of the T-solver

- `void notifyAtom(lit)`
- `void assignLiteral(lit)`
- `void push()`
- `void pop()`
- `result checkSat()`
- `option<result> checkSat_approx()`
- `list<lit> getConflictReason()`
- `option<lit> getPropagatedLiteral()`
- `list<lit> getExplanation(lit)`

Equality and uninterpreted functions

Theory of equality and uninterpreted functions

Signature

- a countable set of function symbols $\Sigma^f = \{f, g, h, \dots\}$
- single predicate symbol $\Sigma^p = \{=\}$

Theory

- T_{UF} is a set of all Σ-structures

Idea

- we only know that $=$ is equivalence and
- the symbols in Σ^f are interpreted as functions $\rightarrow$ for equal arguments return equal results $\rightarrow$ equality is congruence

Running example

$$f(x, y) = x \quad \wedge \quad f(f(x, y), y) = z \quad \wedge \quad g(x) \neq g(z)$$

Applications of uninterpreted functions

- verification of software and hardware (represent unknown external components/functions)
- abstraction of complex parts of the system
- SMT-solving – overapproximation; cheaper unsatisfiability check
- . . .

Flashbacks

Flashbacks from Mathematical Foundations of Computer Science

- each equivalence $\sim \subseteq X^2$ partitions X to a set of equivalence classes $X/_\sim$
- the equivalence class of $x \in X$ is denoted $[x]_\sim$

Flashbacks from Algebra

- given a set of functions Σ^f, an equivalence $\sim$ is congruence if for any $f \in \Sigma^f$ of arity k and $x_i, y_i \in X$

$$(x_1 \sim y_1) \wedge (x_2 \sim y_2) \wedge \ldots \wedge (x_k \sim y_k) \implies f(x_1, x_2, \ldots, x_k) \sim f(y_1, y_2, \ldots, y_k)$$

Herbrand universe

Herbrand universe

- a set of all terms over the signature Σ^f
- denoted $\mathcal{T}$
- can be turned to Σ^f-structure: all functions work syntactically f applied to t_1 and t_2 returns $f(t_1, t_2)$

Main theorem

Theorem

A set of equalities $\mathcal{E}$ and disequalities $\mathcal{D}$ is T_{UF} satisfiable if and only if there exists a congruence $\sim$ on $\mathcal{T}$ such that

- $t_l \sim t_r$ *for all* $(t_l = t_r) \in \mathcal{E}$*, and*
- $t_l \not\sim t_r$ *for all* $(t_l \neq t_r) \in \mathcal{D}$*.*

Proof.

- "$\Rightarrow$": Set $t_1 \sim t_2$ iff $[\![t_1]\!] = [\![t_2]\!]$ in the model.
- "$\Leftarrow$": Construct model from the equality classes.

□

Subterm set

Idea

- terms not occurring in the formula φ are not relevant to its satisfiability
- define a set $\mathcal{T}_\varphi$ of all subterms of φ
- compute equivalence classes only of $\mathcal{T}_\varphi$

Example

Given $\varphi \;=\; f(x,y) = x \;\wedge\; f(f(x,y),y) = z \;\wedge\; g(x) \neq g(z)$

$$\mathcal{T}_\varphi = \{x, y, z, f(x,y), g(x), g(z), f(f(x,y),y)\}$$

Congruence closure

Congruence closure of $R \subseteq X^2$

- the smallest congruence R^* that contains R
- alternatively: $R^* = \bigcap_{S \text{ is congruence }, R \subseteq S} \{S\}$

Congruence closure

Congruence closure of $R \subseteq X^2$

- the smallest congruence R^* that contains R
- alternatively: $R^* = \bigcap_{S \text{ is congruence }, R \subseteq S} \{S\}$

Computing congruence closure

Compute the least fixed point of R_i defined by

$$
\begin{aligned}
R_0 &= R \\
R_{i+1} &= R_i \cup \mathrm{id}_X \cup \\
&\quad \{(x, y) \in X^2 \mid (y, x) \in R_i\} \cup \\
&\quad \{(x, z) \in X^2 \mid (x, y) \in R_i, (y, z) \in R_i\} \cup \\
&\quad \{(f(x_1, \ldots, x_k), f(y_1, \ldots, y_k)) \in X^2 \mid (x_j, y_j) \in R_i \text{ for all } 1 \leq j \leq Ar(f)\}
\end{aligned}
$$

Congruence closure

Congruence closure of $R \subseteq X^2$

- the smallest congruence R^* that contains R
- alternatively: $R^* = \bigcap_{S \text{ is congruence }, R \subseteq S} \{S\}$

Computing congruence closure

Compute the least fixed point of R_i defined by

$$
\begin{aligned}
R_0 &= R \\
R_{i+1} &= R_i \cup \mathrm{id}_X \cup \\
&\quad \{(x, y) \in X^2 \mid (y, x) \in R_i\} \cup \\
&\quad \{(x, z) \in X^2 \mid (x, y) \in R_i, (y, z) \in R_i\} \cup \\
&\quad \{(f(x_1, \ldots, x_k), f(y_1, \ldots, y_k)) \in X^2 \mid (x_j, y_j) \in R_i \text{ for all } 1 \leq j \leq Ar(f)\}
\end{aligned}
$$

Does it have to terminate?

More practical main theorem

Theorem
A formula

$$\varphi = \bigwedge \mathcal{E} \wedge \bigwedge \mathcal{D},$$

where $\mathcal{E}$ is a set of equalities and $\mathcal{D}$ is a set of disequalities, is T_{UF} satisfiable if and only if the congruence closure of $\{(t_l, t_r) \mid (t_l = t_r) \in \mathcal{E}\}$ over $\mathcal{T}_\varphi$ does not contain any (t_l, t_r) such that $(t_l \neq t_r) \in \mathcal{D}$.

Congruence closure algorithm: theoretical description

Algorithm

1. start with singleton equivalence classes $\{t\}$ for each $t \in \mathcal{T}_\varphi$
2. for each $(t_l = t_r) \in \mathcal{E}$, merge the equivalence classes $[t_l]$ and $[t_r]$ and all classes that need to be merged due to congruence
3. if at any point $[t_l] = [t_r]$ for some $(t_l \neq t_r) \in \mathcal{D}$, return unsat
4. otherwise return sat

Congruence closure: example

$$f(x, y) = x \quad \wedge \quad f(f(x, y), y) = z \quad \wedge \quad g(x) \neq g(z)$$

Congruence closure algorithm: reality

Questions

1. how to represent the equivalence classes?
2. how to merge the equivalence classes?
3. how to decide if two terms are in the same equivalence class?
4. we have $\mathcal{O}(|\varphi|)$ subterms, each of size $\mathcal{O}(|\varphi|)$; do we really need $\mathcal{O}(|\varphi|^2)$ memory to store the set $\mathcal{T}_\varphi$?

Reminder: Union-Find

Union-Find

- a data structure to store disjoint sets
- allows creating a singleton sets, merging two sets into one, and computing a representative of a given set
- internally represented by a forest:
  - set = tree in the forest
  - representative = root of a tree
- each element stores its `parent` and `rank`

Reminder: Union-Find

```
make_singleton_set(value) {
    return { value: value; parent = value; rank: 1}
}
```

```
find(item) {
    repr ← item
    while (repr ≠ repr.parent) {
        repr = repr.parent
    }
    return repr
}
```

Reminder: Union-Find

```
union(item1, item2) {
    repr1 ← find(item1)
    repr2 ← find(item2)
    if (repr1 = repr2) return

    if (repr1.rank > repr2.rank) {
        repr2.parent = repr1
    } else if (repr1.rank < repr2.rank) {
        repr1.parent = repr2
    } else {
        repr1.parent = repr2
        repr2.rank++
    }
}
```

E-graph

Efficient storage of terms with shared subterms.

Nodes

- constant/variable with 0 children
- function symbol f of arity k with k children

Example
Consider $f(x, y) = x \ \wedge \ f(f(x, y), y) = z \ \wedge \ g(x) \neq g(z)$.

E-graph

Efficient storage of terms with shared subterms.

Nodes

- constant/variable with 0 children
- function symbol f of arity k with k children

Example
Consider $f(x, y) = x \;\wedge\; f(f(x, y), y) = z \;\wedge\; g(x) \neq g(z)$.

We extend each node with parent pointer and rank to store equivalence classes of terms à la union-find.

Asserting new literals

```
def assert($t = s$):
    todo ← $[(t, s)]$
    while todo not empty:
        $(u, v)$ ← todo.pop()
        if find(u) = find(v): continue
        union(u,v)
        foreach $f(u_1, \ldots, u_k)$ and $f(v_1, \ldots, v_k)$ such that
            $u_i = u$, $v_i = v$ for some $i$ and
            find($u_j$) = find($v_j$) for all $j$
            find($f(u_1, \ldots, u_k)$) $\neq$ find($f(v_1, \ldots, v_k)$):
                todo.append(($f(u_1, \ldots, u_k), f(v_1, \ldots, v_k)$))
    foreach $(v \neq w) \in$ inequalities:
        if find(v) = find(w): return false
    return true

def assert($t \neq s$):
    if find(t) = find(s): return false
    inequalities.append($t \neq s$)
    return true
```

Computing explanations

Idea

- add explanations to each parent pointer (= edge of the E-graph)
- if $\texttt{find}(u) = \texttt{find}(v)$, the explanation is union of
  - sequence of explanations between u and the root $\texttt{find}(u)$ and
  - sequence of explanations between v and the root $\texttt{find}(v)$.

Each union

- of t and s due to $\texttt{assert}(t = s)$
  - explanation = the equality
- of $f(u_1, \ldots, u_n)$ and $f(v_1, \ldots, v_n)$ due to $\texttt{find}(u_i) = \texttt{find}(v_i)$ for all $1 \leq i \leq n$
  - explanation = the union of explanations of all $\texttt{find}(u_i) = \texttt{find}(v_i)$

Theory propagation

Notation

- $t \in [s]$ = t is in the same subtree as s

After each union $\texttt{merge}(t, s)$

- propagate $t' = s'$, where $t' \in [t]$ and $s' \in [s]$
- propagate $t' \neq r$, where $t' \in [t]$, $r' \in [r]$ and there exists $s' \in [s]$ with $s' \neq r' \in \texttt{inequalities}$

After assertion of $t \neq s$

- propagate $t' \neq s'$, where $t' \in [t]$ and $s' \in [s]$

Efficient implementation

Implemented in most of the existing SMT solvers.

For efficient implementation and description of backtracking, see

- R. Nieuwenhuis, A. Oliveras: **Fast congruence closure and extensions**, 2007

Difference logic

Difference logic

Difference logic

- all atoms of form $(x - y) \bowtie k$ for $\bowtie \in \{\leq, <, \geq, >, =, \neq\}$ and a number k
- can be over any numeric theory:
    - $\mathrm{DL}(\mathbb{Q})$
    - $\mathrm{DL}(\mathbb{Z})$

Applications of difference logic

- planning
- scheduling
- verification of timed automata
- $\ldots$

$$\begin{aligned}
a_{end} - a_{start} &\geq 10 \land \\
b_{start} - a_{end} &\geq 0 \land \\
b_{end} - b_{start} &\geq 5 \land \\
b_{end} - a_{start} &\leq 13
\end{aligned}$$

Normalization

Atoms can be normalized to $x - y \leq k$

- $x - y \geq k \quad \rightsquigarrow \quad y - x \leq -k$
- $x - y < k \quad \rightsquigarrow \quad x - y \leq k'$ with k' a smaller number than k (theory-dependent)
- $x - y > k \quad \rightsquigarrow \quad x - y \geq k'$ with k' a bigger number than k (theory-dependent)
- $x - y = k \quad \rightsquigarrow \quad (x - y \leq k) \wedge (x - y \geq k)$
- $x - y \neq k \quad \rightsquigarrow \quad (x - y < k) \vee (x - y > k)$ (needs to be done in the original formula)

Need theory solver only for $\varphi = \bigwedge (x_i - y_j \leq k_j)$

Running examples

Example

$$(x - y \leq 3) \quad \wedge \quad (y - z \leq -11) \quad \wedge \quad (x - z \leq -1) \wedge$$
$$(v - y \leq 15) \quad \wedge \quad (z - v \leq 5) \quad \wedge \quad (v - x \leq 2)$$

Running examples

Example

$$(x - y \leq 3) \;\wedge\; (y - z \leq -11) \;\wedge\; (x - z \leq -1) \wedge$$
$$(v - y \leq 15) \;\wedge\; (z - v \leq 5) \;\wedge\; (v - x \leq 2)$$

Example

$$(x - y \leq 3) \;\wedge\; (y - z \leq -7) \;\wedge\; (x - z \leq -1) \wedge$$
$$(v - y \leq 15) \;\wedge\; (z - v \leq 5) \;\wedge\; (v - x \leq 2)$$

Constraint graph

Given a formula $\varphi = \bigwedge(x_i - y_j \leq k_j)$, we can construct a constraint graph G_φ.

Nodes

- variables of φ

Edges

- edge between x and y of weight k for each conjunct $(x - y \leq k)$ of φ

Main theorem

Theorem

The formula $\varphi = \bigwedge(x_i - y_j \le k_j)$ *is* DL*-satisfiable if and only if* G_φ *does not contain negative cycle.*

Main theorem

Theorem
The formula $\varphi = \bigwedge(x_i - y_j \leq k_j)$ *is* DL*-satisfiable if and only if* G_φ *does not contain negative cycle.*

Proof.

- "⇒": Show by induction that if there is a path between x and y in G_φ of weight k, then $\varphi \models_{\mathrm{DL}} (x - y) \leq k$
- "⇐": Construct a model from shortest paths.

□

Algorithm for difference logic theory solver

Algorithm

1. Construct the graph G_φ.
2. Add a new node s with edges of weight 0 to all nodes of G_φ
3. Run Bellman-Ford algorithm from s.
4. If the algorithm finds negative cycle, return unsat; otherwise return sat.

Computing explanations

Idea

- edges of G_φ = conjuncts φ
- unsatisfiability reason = cycle of negative weight
- unsatisfiability explanation = conjuncts on the cycle

Theory propagation

Idea

1. Compute (or maintain) shortest paths between all pairs of vertices x and y
2. If $\mathrm{dist}(x, y) = d$, propagate all $x - y \leq k$ with $k \geq d$

Efficient implementation

Implemented in most of the existing SMT solvers that deal with arithmetic.

For efficient implementation and description of backtracking and theory propagation, see

- A. Armando, C. Castellini, E. Giunchiglia, M. Maratea: **A SAT-Based Decision Procedure for the Boolean Combination of Difference Constraints**, SAT 2004
- S. Cotton, O. Maler: **Fast and Flexible Difference Constraint Propagation for DPLL(T)**, SAT 2006

Other theories (quick overview)

Linear Real Arithmetic

Normalization

- atoms of form $a_1x_1 + a_2x_2 + \ldots + a_kx_k \leq b$

Theory solver

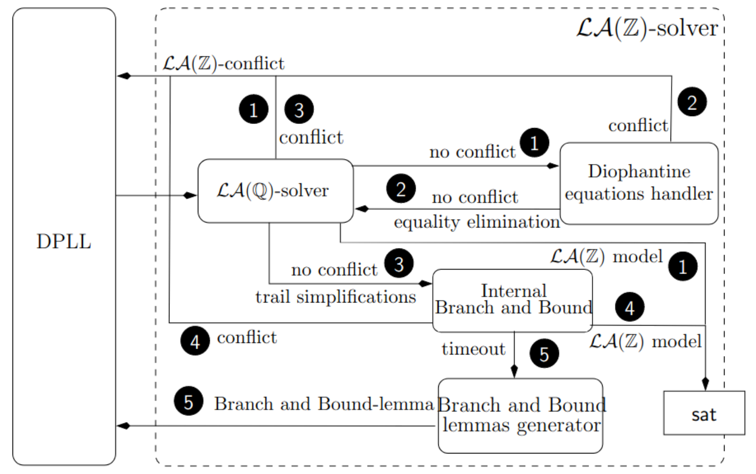
- decide satisfiability conjunctions of atoms of form $a_1x_1 + a_2x_2 + \ldots + a_kx_k \leq b$ and their negations
- simplex algorithm
- needs changes to be incremental and backtrackable, see
  - B. Dutetre, L. de Moura: **A Fast Linear-Arithmetic Solver for DPLL(T)**, CAV 2006

Linear Integer Arithmetic

Much more complicated. Combination of:

- simplex on the LRA relaxation of the formula
- branch and bound
- cutting planes
- diophantine equation solving
- $\ldots$

Linear Integer Arithmetic



[from A.Griggio: A Practical Approach to Satisfiability Modulo Linear Integer Arithmetic]

Bit-vectors

Combinations of

1. heavy preprocessing
2. converting all operations to Boolean circuits that compute them (usually and-inverter graphs)
3. more preprocessing
4. computing propositional formula that encodes the circuit
5. often done eagerly

The conversion of bit-vector formula to the equisatisfiable propositional formula is called bit-blasting.

Arrays

Lazy approach

1. treat `read` and `write` as uninterpreted functions
2. check UF-satisfiability
3. if unsat, return unsat
4. if sat, check whether the model satisfies array axioms
   - if it does, return sat
   - if not, add the violated axioms and check UF-satisfiability again

Next time

- combination of theories
- Nelson-Oppen algorithm