Satisfiability Checking

(Full/Less) Lazy SMT-Solving for Equality Logic

Prof. Dr. Erika Ábrahám

RWTH Aachen University
Informatik 2
LuFG Theory of Hybrid Systems

WS 16/17

Reminder: Equality logic with uninterpreted functions

We extend the propositional logic with

- equalities and
- uninterpreted functions (UFs).

Syntax:

- variables x over an arbitrary domain D,
- constants c from the same domain D,
- function symbols F for functions of the type $D^n \to D$, and
- equality as predicate symbol.

Terms:	t	$::=$	c	$\mid$	x	$\mid$	$F(t, \ldots, t)$
Formulas:	φ	$::=$	$t = t$	$\mid$	$(\varphi \wedge \varphi)$	$\mid$	$(\neg\varphi)$

Semantics: straightforward

Reminder: Equality logic with uninterpreted functions

We extend the propositional logic with

- equalities and
- uninterpreted functions (UFs).

Syntax:

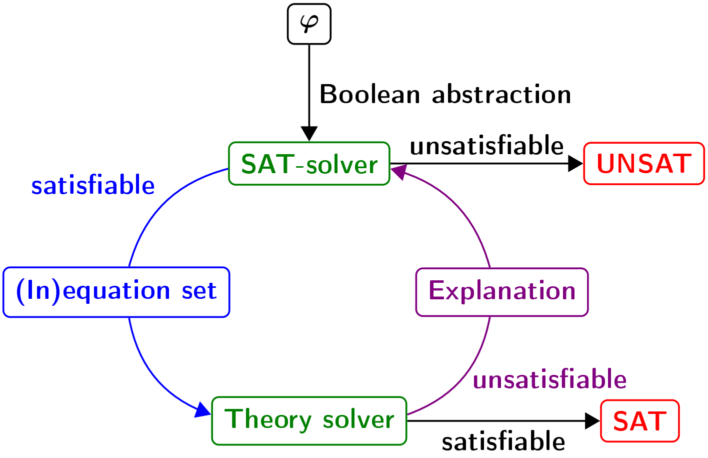
- variables x over an arbitrary domain D,
- constants c from the same domain D,
- function symbols F for functions of the type $D^n \rightarrow D$, and
- equality as predicate symbol.

$$\begin{array}{llllllll} \textit{Terms:} & t & ::= & c & \mid & x & \mid & F(t,\ldots,t) \\ \textit{Formulas:} & \varphi & ::= & t = t & \mid & (\varphi \wedge \varphi) & \mid & (\neg \varphi) \end{array}$$

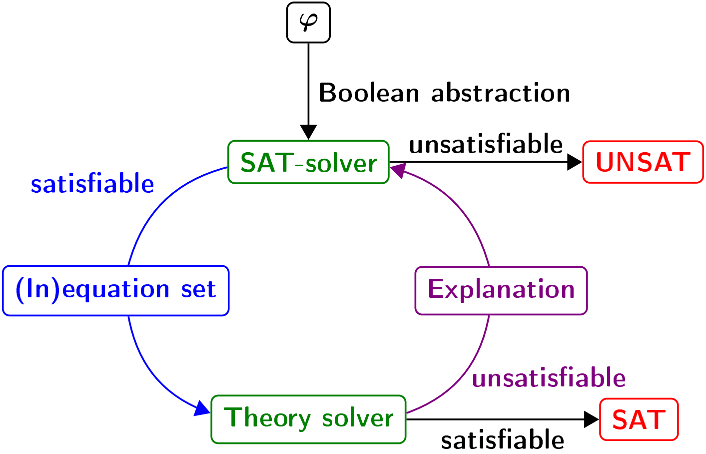
Semantics: straightforward

For simplicity in the following we assume that terms do not contain any constants.

Full lazy SMT-solving



Full lazy SMT-solving



We need a theory solver for conjunctions/sets of equations and disequations over terms (variables and uninterpreted functions, see page 2).

Basic idea for the theory solver (EQ+UF, full lazy)

Basic idea for the theory solver (EQ+UF, full lazy)

- Let T be the set of all subterms in a conjunctive EQ+UF formula φ, and assume an infinite (or "sufficiently large") theory domain.
- We compute a partition $\mathcal{C}$ of T (i.e., we define an equivalence relation over T) such that two terms $t, t' \in T$ are in the same equivalence class (written: $[t] = [t']$) if and only if all models that satisfy all equations in φ assign equal values to both terms.
- Then we check for each disequation whether the two operands (sides) are in different equivalence classes (yes: φ is SAT, no: φ is UNSAT).

Basic idea for the theory solver (EQ+UF, full lazy)

- Let T be the set of all subterms in a conjunctive EQ+UF formula φ, and assume an infinite (or "sufficiently large") theory domain.
- We compute a partition $\mathcal{C}$ of T (i.e., we define an equivalence relation over T) such that two terms $t, t' \in T$ are in the same equivalence class (written: $[t] = [t']$) if and only if all models that satisfy all equations in φ assign equal values to both terms.
- Then we check for each disequation whether the two operands (sides) are in different equivalence classes (yes: φ is SAT, no: φ is UNSAT).

How to compute such a partition?

Basic idea for the theory solver (EQ+UF, full lazy)

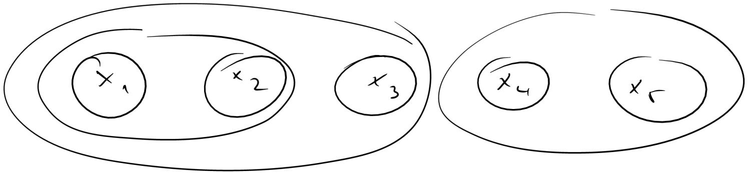
- Let T be the set of all subterms in a conjunctive EQ+UF formula φ, and assume an infinite (or "sufficiently large") theory domain.
- We compute a partition $\mathcal{C}$ of T (i.e., we define an equivalence relation over T) such that two terms $t, t' \in T$ are in the same equivalence class (written: $[t] = [t']$) if and only if all models that satisfy all equations in φ assign equal values to both terms.
- Then we check for each disequation whether the two operands (sides) are in different equivalence classes (yes: φ is SAT, no: φ is UNSAT).

How to compute such a partition?

- Initial partition: Each subterm from T has its own equivalence class.
- Equality: We assure reflexivity, symmetry and transitivity for equality by merging the equivance classes for the two sides of each equation in φ.
- Uninterpreted functions: We assure the congruence of uninterpreted functions by iteratively merging the equivance classes of function applications with equivalent (i.e., necessarily equal) operands.

Example: EQ (no UF, full lazy)

$\varphi^E: \quad x_1 = x_2 \wedge x_2 = x_3 \wedge x_4 = x_5 \wedge x_5 \neq x_1$



Example: EQ (no UF, full lazy)

$$\varphi^E: \quad x_1 = x_2 \wedge x_2 = x_3 \wedge x_4 = x_5 \wedge x_5 \neq x_1$$

Equivalence class 1 Equivalence class 2

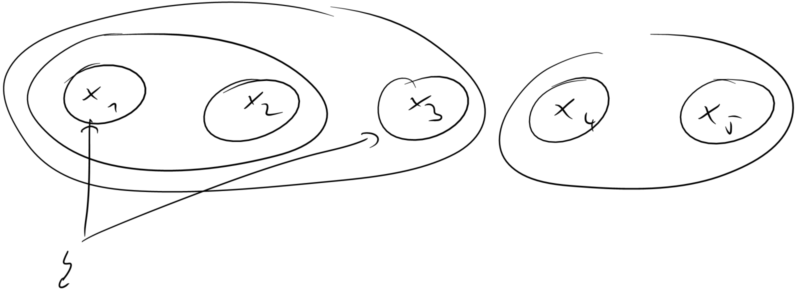
Example: EQ (no UF, full lazy)

$\varphi^E: \quad x_1 = x_2 \wedge x_2 = x_3 \wedge x_4 = x_5 \wedge x_5 \neq x_1$

Equivalence class 1 Equivalence class 2

Example: EQ (no UF, full lazy)

φ^E : $\quad x_1 = x_2 \wedge x_2 = x_3 \wedge x_4 = x_5 \wedge x_1 \neq x_3$



Example: EQ (no UF, full lazy)

$$\varphi^E : \quad x_1 = x_2 \wedge x_2 = x_3 \wedge x_4 = x_5 \wedge x_1 \neq x_3$$

Equivalence class 1   Equivalence class 2

Example: EQ (no UF, full lazy)

$\varphi^E: \quad x_1 = x_2 \wedge x_2 = x_3 \wedge x_4 = x_5 \wedge x_1 \neq x_3$

Equivalence class 1

Equivalence class 2

Algorithm 1: satCheck (EQ, no UF, full lazy)

Input: Set V of variables, set E of equations and disequations over V

Output: Satisfiability of $\bigwedge_{e \in E} e$

Algorithm 1: satCheck (EQ, no UF, full lazy)

Input: Set V of variables, set E of equations and disequations over V
Output: Satisfiability of $\bigwedge_{e\in E} e$

1. Initial partition has an own equivalence class for each variable in V:
   $\mathcal{C} := \{\{x\} \mid x \in V\}$;
2. Assure transitivity for equality: For each input equation $x = x'$, if the equivalence classes $[x]$ and $[x']$ of the two sides differ then merge them:
   $\texttt{for each } (x = x') \in E$
   $\quad \texttt{if } ([x] \neq [x']) \texttt{ then } \mathcal{C} := (\mathcal{C} \setminus \{[x],[x']\}) \cup \{[x] \cup [x']\}$;
3. For each disequation $(x \neq x') \in E$, if the equivalence classes of the two sides coincide then return unsatisfiability:
   $\texttt{for each } (x \neq x') \in E \texttt{ if } ([x] = [x']) \texttt{ then return UNSAT}$;
4. Else return satisfiability:
   $\texttt{return SAT}$.

Next: Add uninterpreted functions (EQ+UF, full lazy)

Are these conjunctions satisfiable?

- $x = y \wedge F(x) = F(y)$:

Next: Add uninterpreted functions (EQ+UF, full lazy)

Are these conjunctions satisfiable?

- $x = y \wedge F(x) = F(y)$: satisfiable

Next: Add uninterpreted functions (EQ+UF, full lazy)

Are these conjunctions satisfiable?

- $x = y \land F(x) = F(y)$: satisfiable
- $x = y \land F(x) \neq F(y)$:

Next: Add uninterpreted functions (EQ+UF, full lazy)

Are these conjunctions satisfiable?

- $x = y \wedge F(x) = F(y)$: satisfiable
- $x = y \wedge F(x) \neq F(y)$: unsatisfiable

Next: Add uninterpreted functions (EQ+UF, full lazy)

Are these conjunctions satisfiable?

- $x = y \wedge F(x) = F(y)$: satisfiable
- $x = y \wedge F(x) \neq F(y)$: unsatisfiable
- $x \neq y \wedge F(x) = F(y)$:

Next: Add uninterpreted functions (EQ+UF, full lazy)

Are these conjunctions satisfiable?

- $x = y \wedge F(x) = F(y)$: satisfiable
- $x = y \wedge F(x) \neq F(y)$: unsatisfiable
- $x \neq y \wedge F(x) = F(y)$: satisfiable

Next: Add uninterpreted functions (EQ+UF, full lazy)

Are these conjunctions satisfiable?

- $x = y \wedge F(x) = F(y)$: satisfiable
- $x = y \wedge F(x) \neq F(y)$: unsatisfiable
- $x \neq y \wedge F(x) = F(y)$: satisfiable
- $x \neq y \wedge F(x) \neq F(y)$:

Next: Add uninterpreted functions (EQ+UF, full lazy)

Are these conjunctions satisfiable?

- $x = y \wedge F(x) = F(y)$: satisfiable
- $x = y \wedge F(x) \neq F(y)$: unsatisfiable
- $x \neq y \wedge F(x) = F(y)$: satisfiable
- $x \neq y \wedge F(x) \neq F(y)$: satisfiable

Next: Add uninterpreted functions (EQ+UF, full lazy)

Are these conjunctions satisfiable?

- $x = y \wedge F(x) = F(y)$: satisfiable
- $x = y \wedge F(x) \neq F(y)$: unsatisfiable
- $x \neq y \wedge F(x) = F(y)$: satisfiable
- $x \neq y \wedge F(x) \neq F(y)$: satisfiable

- $x = y \wedge F(G(x)) \neq F(G(y))$:

Next: Add uninterpreted functions (EQ+UF, full lazy)

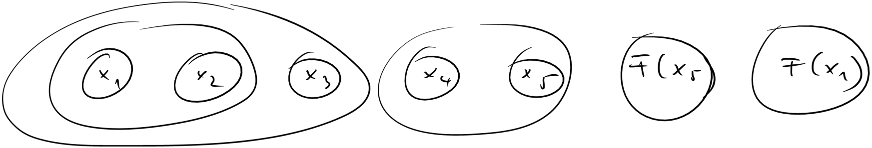
Are these conjunctions satisfiable?

- $x = y \wedge F(x) = F(y)$: satisfiable
- $x = y \wedge F(x) \neq F(y)$: unsatisfiable
- $x \neq y \wedge F(x) = F(y)$: satisfiable
- $x \neq y \wedge F(x) \neq F(y)$: satisfiable

- $x = y \wedge F(G(x)) \neq F(G(y))$: unsatisfiable

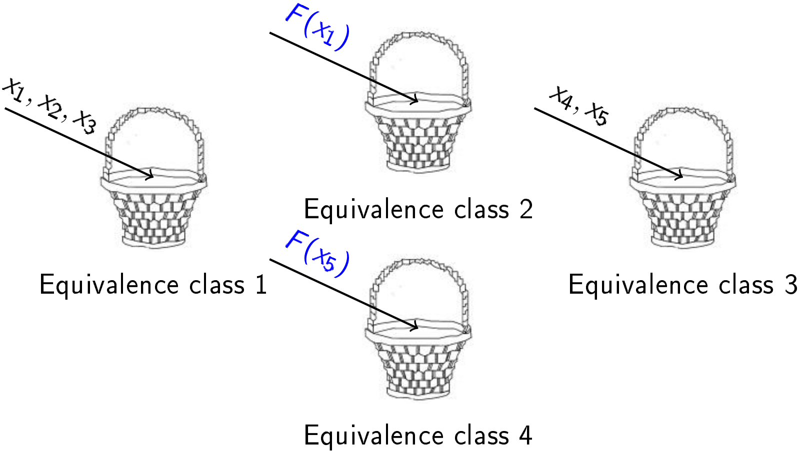
Next: Add uninterpreted functions (EQ+UF, full lazy)

$x_1 = x_2 \wedge x_2 = x_3 \wedge x_4 = x_5 \wedge \wedge {\color{blue}F(x_5) \neq F(x_1)}$



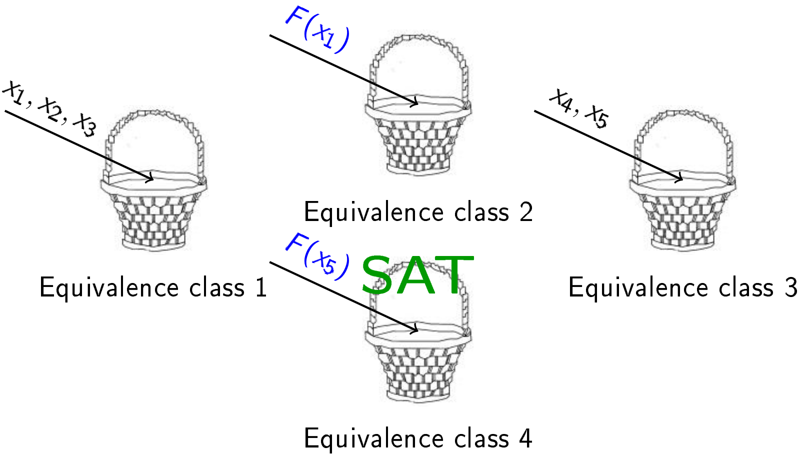
Next: Add uninterpreted functions (EQ+UF, full lazy)

$$x_1 = x_2 \wedge x_2 = x_3 \wedge x_4 = x_5 \wedge \wedge F(x_5) \neq F(x_1)$$

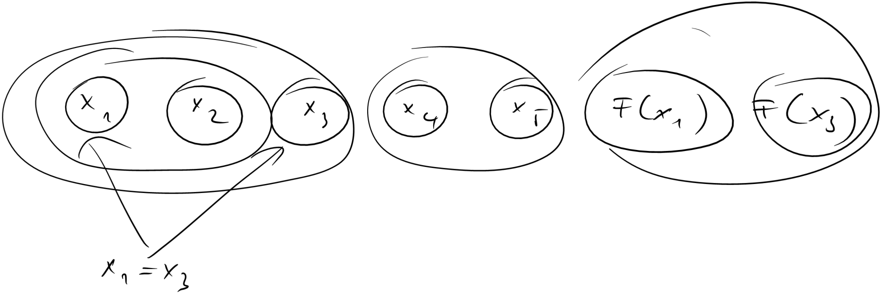


Next: Add uninterpreted functions (EQ+UF, full lazy)

$x_1 = x_2 \wedge x_2 = x_3 \wedge x_4 = x_5 \wedge \wedge F(x_5) \neq F(x_1)$

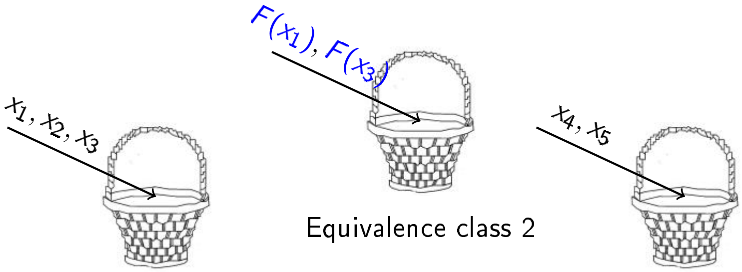


$$x_1 = x_2 \wedge x_2 = x_3 \wedge x_4 = x_5 \wedge \wedge F(x_1) \neq F(x_3)$$



Next: Add uninterpreted functions (EQ+UF, full lazy)

$x_1 = x_2 \wedge x_2 = x_3 \wedge x_4 = x_5 \wedge \wedge F(x_1) \neq F(x_3)$



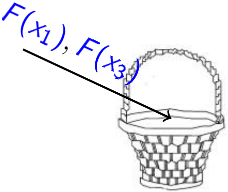
Equivalence class 2

Equivalence class 1

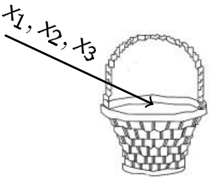
Equivalence class 3

Next: Add uninterpreted functions (EQ+UF, full lazy)

$$x_1 = x_2 \wedge x_2 = x_3 \wedge x_4 = x_5 \wedge \wedge F(x_1) \neq F(x_3)$$

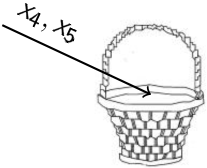


Equivalence class 2



Equivalence class 1

UNSAT



Equivalence class 3

Algorithm 2: satCheck (EQ+UF, full lazy)

Input: Set T of terms, set E of equalities and disequalities over T

Output: Satisfiability of $\bigwedge_{e \in E} e$

Algorithm 2: satCheck (EQ+UF, full lazy)

Input: Set T of terms, set E of equalities and disequalities over T

Output: Satisfiability of $\bigwedge_{e \in E} e$

1. Initial partition has an own equivalence class for each term in T:
   $\mathcal{C} := \{\{t\} \mid t \in T\}$;
2. Assure transitivity for equality: For each input equation $t = t'$, if the equivalence classes $[t]$ and $[t']$ of the two sides differ then merge them:
   $\texttt{for each } (t = t') \in E$
   $\quad \texttt{if } ([t] \neq [t']) \texttt{ then } \mathcal{C} := (\mathcal{C} \setminus \{[t], [t']\}) \cup \{[t] \cup [t']\}$;
3. Assure functional congruence for uninterpreted functions:
   $\texttt{while exist subterms } F(t), F(t') \texttt{ in } T \texttt{ with } [t]{=}[t'] \texttt{ and } [F(t)]{\neq}[F(t')]$
   $\quad \mathcal{C} := (\mathcal{C} \setminus \{[F(t)], [F(t')]\}) \ \cup\ \{[F(t)] \cup [F(t')]\}$; $t = t' \Rightarrow F(t) = F(t')$
4. For each disequation $(t \neq t') \in E$, if the equivalence classes of the two sides coincide then return unsatisfiability:
   $\texttt{for each } (t \neq t') \in E \texttt{ if } ([t] = [t']) \texttt{ then return UNSAT}$;
5. Else return satisfiability:
   $\texttt{return SAT}$.

$t = t'$ ✓ $\quad F(G(t)) \neq F(G(t'))$

$t \quad t' \qquad G(t) \qquad G(t') \qquad F(G(t)) \qquad F(G(t'))$

2\.

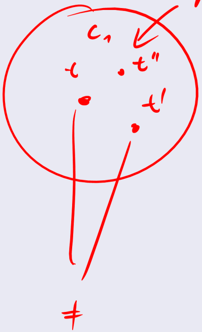
1) $F(G(t)) \qquad F(G(t'))$ => not merge => merge

2) $G(t) \qquad G(t')$ => merge

Full lazy EQ+UF theory solver

Algorithm 3: getExplanation (EQ+UF, full lazy)

All eq. m'e terms from C_1 + $\neq$



$t = t' \wedge t = t'' \wedge t \neq t'$

? ? ?

$\Rightarrow$ take $\Rightarrow$ no $\Rightarrow$ take

Full lazy EQ+UF theory solver

Algorithm 3: getExplanation (EQ+UF, full lazy)

Input: none

Output: if the current state is UNSAT then an unsatisfiable subset of the previously received equations and disequations else the empty set

1 $\varphi := \emptyset$;

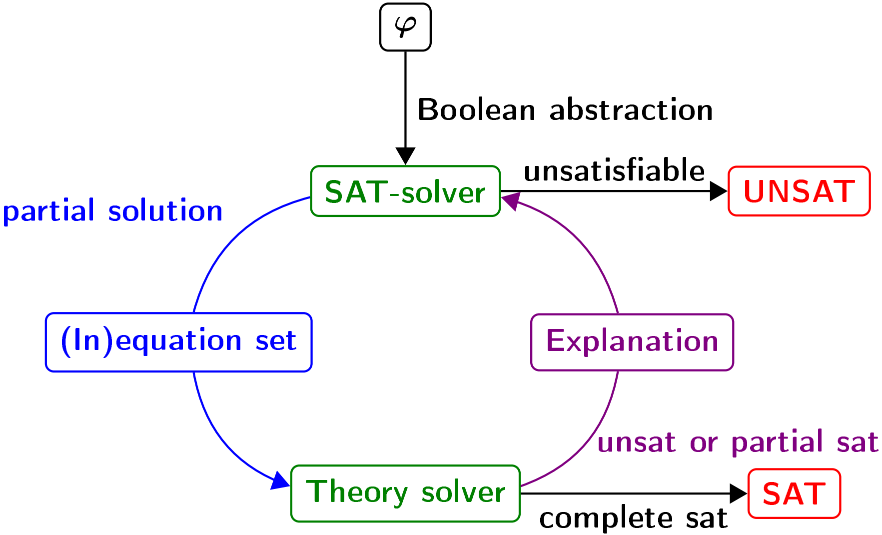
2 $E_= = \{(t = t') \in E\}$;

3 **for each** $(t \neq t') \in E$
  if $\left(t \neq t' \wedge \bigwedge_{e \in E_=} e\right)$ **is UNSAT (use Alg.2) then**
    $\varphi := \{t \neq t'\}$;
    break;
  fi

4 **if** $(\varphi = \emptyset)$ **return** $\emptyset$;

5 **while** $E_= \neq \emptyset$
  let $(t = t') \in E_=$; $E_= := E_= \setminus \{(t = t')\}$;
  if $(\bigwedge_{e \in \varphi \cup E_=} e)$ **is SAT then** $\varphi := \varphi \cup \{t = t'\}$;
od

6 **return** φ;

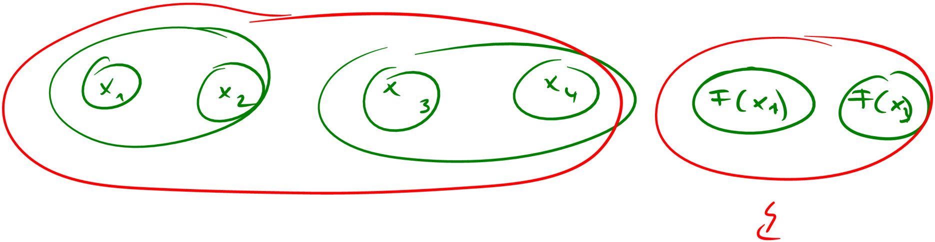

φ
Boolean abstraction
SAT-solver
unsatisfiable
UNSAT
partial solution
(In)equation set
Explanation
unsat or partial sat
Theory solver
SAT
complete sat

Requirements on the theory solver

Needed for less lazy SMT solving:

1. Incrementality: In less lazy solving we extend the set of constraints. The solver should make use of the previous satisfiability check for the check of the extended set.
2. (Preferably minimal) infeasible subsets: Compute a reason for unsatisfaction
3. Backtracking: The theory solver should be able to remove constraints in inverse chronological order.

$x_1 = x_2 \qquad x_3 = x_4 \qquad F(x_1) \neq F(x_3) \qquad x_2 = x_3$



Requirements on the theory solver

Solution:

1. Incrementality:
   - When a new equation is added, update the partition and check the previously added disequations for satisfiability.
   - When a new disequation is added, check the satisfiability of the new disequation.
2. (Preferably minimal) infeasible subsets: A conflict appears when a disequation $t \neq t'$ cannot be true together with the current equations; build the set of this disequation $t \neq t'$ and (a minimal number of) equations that imply $t = t'$ by transitivity and congruence.
3. Backtracking: Remember computation history.

Less lazy EQ+UF theory solver

Algorithm 4: Init (EQ+UF, less lazy)

Less lazy EQ+UF theory solver

Algorithm 4: Init (EQ+UF, less lazy)

Input: Set T of all subterms which can be used in (dis)equations
Output: none

1. No equations or disequations received yet:
   $E := \emptyset$;
2. Initial partition over T is empty:
   $\mathcal{C} := \{\{t\} \mid t \in T\}$;
3. Remember current state of satisfiability:
   $state := SAT$;

Less lazy EQ+UF theory solver

Algorithm 5: addEquation (EQ+UF, less lazy)

Less lazy EQ+UF theory solver

Algorithm 5: addEquation (EQ+UF, less lazy)

Input: Equation $t_1 = t_2$ with subterms from T

Output: Satisfiability of the conjunction of all received (and not yet removed) equations and disequations

1. Remember the new equation: $E := E \cup \{t_1 = t_2\}$;
2. If the problem was already unsatisfiable then it is still unsatisfiable:
   `if` $(state = UNSAT)$ `then return UNSAT`;
3. Update the partition using the new equation and re-check satisfiability of the disequations:

```
if ([t1] ≠ [t2]) then
  C := (C \ {[t1], [t2]}) ∪ {[t1] ∪ [t2]};
  while exist subterms F(t), F(t') ∈ T with [t]=[t'] and [F(t)]≠[F(t')]
    C := (C \ {[F(t)], [F(t')]}) ∪ {[F(t)] ∪ [F(t')]};
  for each (t ≠ t') ∈ E
    if ([t] = [t']) then
      state := UNSAT; return UNSAT;
    fi
fi
return SAT.
```

Less lazy EQ+UF theory solver

Algorithm 6: addDisequation (EQ+UF, less lazy)

Less lazy EQ+UF theory solver

Algorithm 6: addDisequation (EQ+UF, less lazy)

Input: Disequation $t_1 \neq t_2$ with subterms from T
Output: Satisfiability of the conjunction of all received (and not yet removed) equations and disequations

1. Remember the new disequation:
   $E := E \cup \{t_1 \neq t_2\}$;
2. If the problem was already unsatisfiable then it is still unsatisfiable:
   `if` $(state = UNSAT)$ `then return UNSAT;`
3. Check satisfiability of the new disequation:
   `if` $([t_1] = [t_2])$ `then`
   $state := UNSAT$;
   `return UNSAT;`
   `fi`
   `return SAT.`

Less lazy EQ+UF theory solver

Algorithm 7: getExplanation (EQ+UF, less lazy)

Same as for full lazy.

Less lazy EQ+UF theory solver

This is just a rather naive (and informal) solution for backtracking... more optimal solutions need smart datastructures for efficient book-keeping.

Algorithm 8: backtrack (EQ+UF, less lazy)

Less lazy EQ+UF theory solver

This is just a rather naive (and informal) solution for backtracking... more optimal solutions need smart datastructures for efficient book-keeping.

Algorithm 8: backtrack (EQ+UF, less lazy)

Input: Equation and disequation set S

Output: none

1. Remove the equations and disequations:
   $E := E \setminus S$;
2. Apply `Algorithm 2` (page 11) to compute satisfiability for T and the conjunction of all equations and disequations from E.
3. Remember the satisfiability result in `status`.
4. Return the satisfiability result.