Decision Procedures

Jochen Hoenicke

Software Engineering
Albert-Ludwigs-University Freiburg

Winter Term 2019/2020

Quantifier-free Theory of Equality

$$\Sigma_E\ :\ \{=,\ a,\ b,\ c,\ \ldots,\ f,\ g,\ h,\ \ldots,\ p,\ q,\ r,\ \ldots\}$$

uninterpreted symbols:

- constants $a, b, c, \ldots$
- functions $f, g, h, \ldots$
- predicates $p, q, r, \ldots$

Axioms of T_E

1. $\forall x.\ x = x$ (reflexivity)
2. $\forall x, y.\ x = y \rightarrow y = x$ (symmetry)
3. $\forall x, y, z.\ x = y \wedge y = z \rightarrow x = z$ (transitivity)

define $=$ to be an equivalence relation.

Axiom schema

4. for each positive integer n and n-ary function symbol f,
$$\forall x_1, \ldots, x_n, y_1, \ldots, y_n.\ \bigwedge_i x_i = y_i \rightarrow f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n) \quad \text{(congruence)}$$
5. for each positive integer n and n-ary predicate symbol p,
$$\forall x_1, \ldots, x_n, y_1, \ldots, y_n.\ \bigwedge_i x_i = y_i \rightarrow (p(x_1, \ldots, x_n) \leftrightarrow p(y_1, \ldots, y_n)) \quad \text{(equivalence)}$$

Conjunctive Quantifier-free Fragment

In the next lectures, we consider conjunctive quantifier-free Σ-formulae, i.e., conjunctions of Σ-literals (Σ-atoms or negations of Σ-atoms).

Remark 1: From this an algorithm for arbitrary quantifier-free formulae can be built.
For given arbitrary quantifier-free Σ-formula F, convert it into DNF Σ-formula

$$F_1 \vee \ldots \vee F_k$$

where each F_i conjunctive.
F is T-satisfiable iff at least one F_i is T-satisfiable.

Remark 2: One can also combine a decision procedure for conjunctive fragment with DPLL.

Congruence Closure Algorithm

$$F : s_1 = t_1 \wedge \cdots \wedge s_m = t_m \wedge \; s_{m+1} \neq t_{m+1} \wedge \cdots \wedge s_n \neq t_n$$

The algorithm performs the following steps:

1. Construct the congruence closure $\sim$ of

$$\{s_1 = t_1, \ldots, s_m = t_m\}$$

   over the subterm set S_F. Then

$$\sim \models s_1 = t_1 \wedge \cdots \wedge s_m = t_m \,.$$

2. If for any $i \in \{m+1, \ldots, n\}$, $s_i \sim t_i$, return unsatisfiable.
3. Otherwise, $\sim \models F$, so return satisfiable.

How do we actually construct the congruence closure in Step 1?

Congruence Closure Algorithm (Details)

Begin with the finest congruence relation $\sim_0$:

$$\{\{s\} \;:\; s \in S_F\} \,.$$

Each term of S_F is only congruent to itself.

Then, for each $i \in \{1, \ldots, m\}$, impose $s_i = t_i$ by merging

$$[s_i]_{\sim_{i-1}} \quad \text{and} \quad [t_i]_{\sim_{i-1}}$$

to form a new congruence relation $\sim_i$. To accomplish this merging,

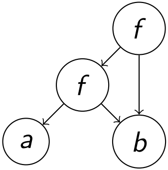
- form the union of $[s_i]_{\sim_{i-1}}$ and $[t_i]_{\sim_{i-1}}$
- propagate any new congruences that arise within this union.

The new relation $\sim_i$ is a congruence relation in which $s_i \sim t_i$.

Ingredients of Algorithm

Efficient data structure for computing the congruence closure.

- Directed Acyclic Graph (DAG) to represent terms.

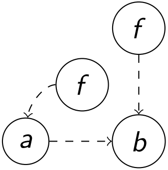


$f(f(a, b), b)$

$f(a, b)$

a b

- Union-Find data structure to represent equivalence classes:

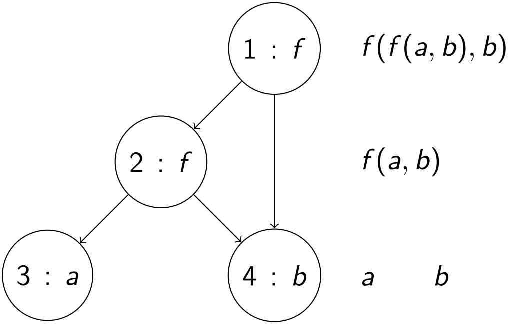


Directed Acyclic Graph (DAG)

For every subterm of the Σ_E-formula F, create

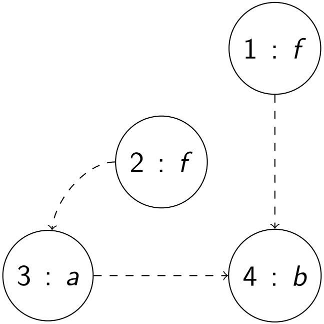
- a node labelled with the function symbols.
- and edges to the argument nodes.

If two subterms are equal, only one node is created.



Union-Find Data Structure

Equivalence classes are connected by a tree structure, with arrows pointing to the root node.

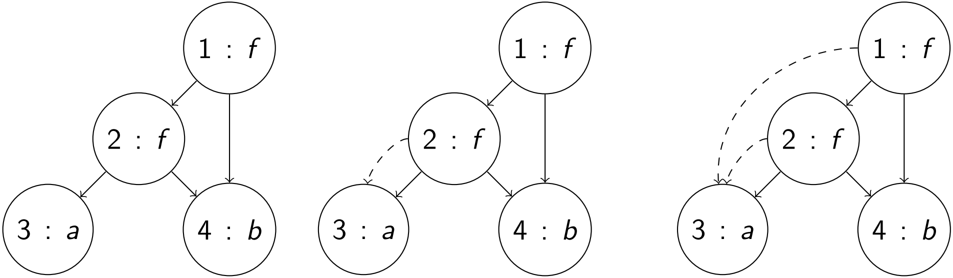


Two operations are defined:

- FIND: Find the representative of an equivalence class by following the edges. $O(\log n)$
- UNION: Merge two classes by connecting the representatives. $O(1)$

Summary of idea

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



Initial DAG

$f(a,b) = a \Rightarrow$
MERGE $f(a,b)$ a

$f(a,b) \sim a,\ b \sim b \Rightarrow$
$f(f(a,b),b) \sim f(a,b)$
MERGE $f(f(a,b),b)$
$f(a,b)$

$$\left.\begin{array}{r} \text{FIND } f(f(a,b),b) = a = \text{FIND } a \\ f(f(a,b),b) \neq a \end{array}\right\} \Rightarrow$$ **Unsatisfiable**

DAG representation

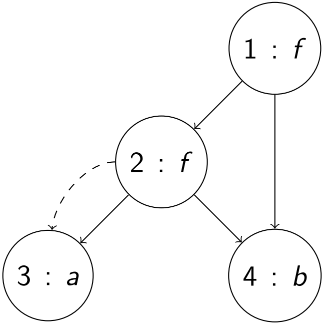
type **node** = {

`id`	:	**id** node's unique identification number
`fn`	:	**string** constant or function name
`args`	:	**id list** list of function arguments
`mutable find`	:	**id** the edge to the representative
`mutable ccpar`	:	**id set** if the node is the representative for its congruence class, then its `ccpar` (congruence closure parents) are all parents of nodes in its congruence class

}

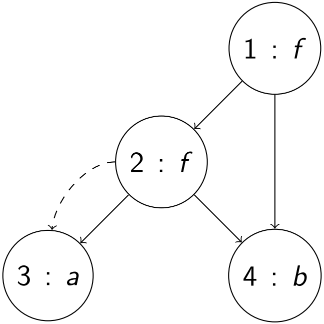
DAG Representation of node 2

type **node** = {			
id	:	**id**	... 2
fn	:	**string**	... f
args	:	**idlist**	... $[3, 4]$
mutable find	:	**id**	... 3
mutable ccpar	:	**idset**	... $\emptyset$
}			



DAG Representation of node 3

type **node** = {			
id	:	**id**	... 3
fn	:	**string**	... *a*
args	:	**idlist**	... []
mutable find	:	**id**	... 3
mutable ccpar	:	**idset**	... $\{1, 2\}$
}			

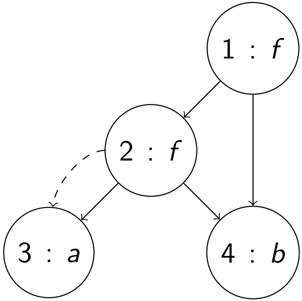


The Implementation: FIND

FIND function

returns the representative of node's congruence class

```
let rec FIND i =
  let n  =  NODE i in
  if n.find = i then i else FIND n.find
```



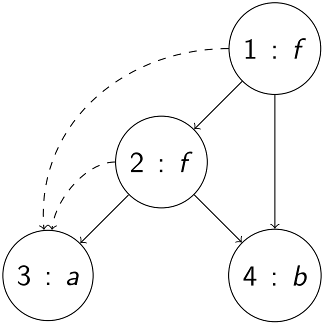
Example: FIND 2 = FIND 3 = 3
3 is the representative of 2.

UNION function

$$
\begin{array}{l}
\texttt{let}\ \textsc{union}\ i_1\ i_2 = \\
\quad \texttt{let}\ n_1 = \textsc{node}\ (\textsc{find}\ i_1)\ \texttt{in} \\
\quad \texttt{let}\ n_2 = \textsc{node}\ (\textsc{find}\ i_2)\ \texttt{in} \\
\quad n_1.\texttt{find} \leftarrow n_2.\texttt{find}; \\
\quad n_2.\texttt{ccpar} \leftarrow n_1.\texttt{ccpar} \cup n_2.\texttt{ccpar}; \\
\quad n_1.\texttt{ccpar} \leftarrow \emptyset
\end{array}
$$

n_2 is the representative of the union class

Example



UNION 1 2 $\quad n_1 = 1 \quad n_2 = 3$

$\quad 1.\texttt{find} \leftarrow 3$

$\quad 3.\texttt{ccpar} \leftarrow \{1, 2\}$

$\quad 1.\texttt{ccpar} \leftarrow \emptyset$

The Implementation: CONGRUENT

CCPAR function

Returns parents of all nodes in i's congruence class

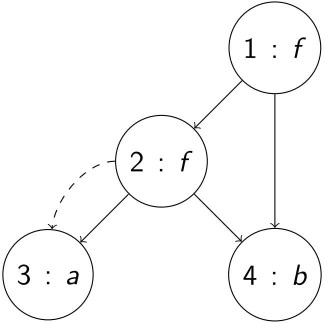
$$\texttt{let}\ \textsc{ccpar}\ i\ =\ (\textsc{node}\ (\textsc{find}\ i)).\texttt{ccpar}$$

CONGRUENT predicate

Test whether i_1 and i_2 are congruent

$$
\begin{array}{l}
\texttt{let}\ \textsc{congruent}\ i_1\ i_2\ = \\
\quad \texttt{let}\ n_1\ =\ \textsc{node}\ i_1\ \texttt{in} \\
\quad \texttt{let}\ n_2\ =\ \textsc{node}\ i_2\ \texttt{in} \\
\quad n_1.\texttt{fn} = n_2.\texttt{fn} \\
\qquad \wedge |n_1.\texttt{args}| = |n_2.\texttt{args}| \\
\qquad \wedge \forall i \in \{1, \ldots, |n_1.\texttt{args}|\}.\ \textsc{find}\ n_1.\texttt{args}[i] = \textsc{find}\ n_2.\texttt{args}[i]
\end{array}
$$

Example



Are 1 and 2 congruent?

`fn` fields	— both f
# of arguments	— same
left arguments $f(a,b)$ and a	— both congruent to 3
right arguments b and b	— both 4 (congruent)

Therefore 1 and 2 are congruent.

MERGE function

```
let rec MERGE i1 i2 =
  if FIND i1 ≠ FIND i2 then begin
    let P_i1 = CCPAR i1 in
    let P_i2 = CCPAR i2 in
    UNION i1 i2;
    foreach t1, t2 ∈ P_i1 × P_i2 do
      if FIND t1 ≠ FIND t2 ∧ CONGRUENT t1 t2
      then MERGE t1 t2
    done
  end
```

P_{i_1} and P_{i_2} store the current values of CCPAR i_1 and CCPAR i_2.

Decision Procedure: T_E-satisfiability

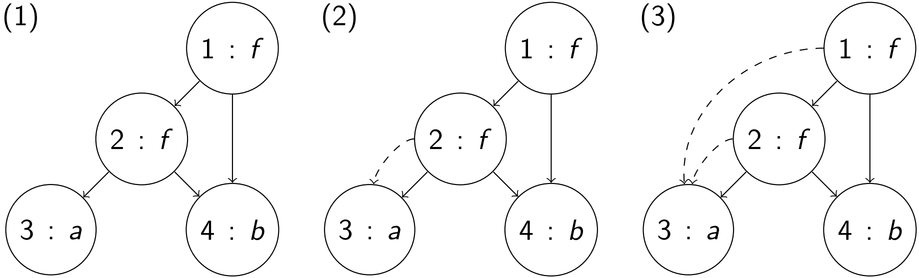
Given Σ_E-formula

$$F : s_1 = t_1 \wedge \cdots \wedge s_m = t_m \wedge s_{m+1} \neq t_{m+1} \wedge \cdots \wedge s_n \neq t_n ,$$

with subterm set S_F, perform the following steps:

1. Construct the initial DAG for the subterm set S_F.
2. For $i \in \{1, \ldots, m\}$, MERGE s_i t_i.
3. If FIND s_i = FIND t_i for some $i \in \{m+1, \ldots, n\}$, return unsatisfiable.
4. Otherwise (if FIND $s_i \neq$ FIND t_i for all $i \in \{m+1, \ldots, n\}$) return satisfiable.

$$f(a,b) = a \;\land\; f(f(a,b),b) \neq a$$



Initial DAG

MERGE 2 3

UNION 2 3

$P_2 = \{1\}$

$P_3 = \{2\}$

CONGRUENT 1 2

MERGE 1 2

UNION 1 2

$P_1 = \{\}$

$P_2 = \{1, 2\}$

FIND $f(f(a,b),b) = a =$ FIND $a \Rightarrow$ **Unsatisfiable**

Given Σ_E-formula

$$F : f(a,b) = a \wedge f(f(a,b),b) \neq a \ .$$

The subterm set is

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

resulting in the initial partition

$$(1)\ \{\{a\},\ \{b\},\ \{f(a,b)\},\ \{f(f(a,b),b)\}\}$$

in which each term is its own congruence class. Fig (1).

Final partition

$$(2)\ \{\{a, f(a,b), f(f(a,b),b)\},\ \{b\}\}$$

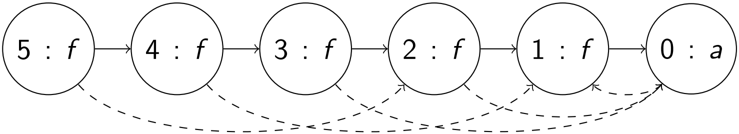
Does

$$(3)\ \{\{a, f(a,b), f(f(a,b),b)\},\ \{b\}\} \models F\ ?$$

No, as $f(f(a,b),b) \sim a$, but F asserts that $f(f(a,b),b) \neq a$. Hence, F is T_E-unsatisfiable.

Example $f^3(a) = a \wedge f^5(a) = a \wedge f(a) \neq a$

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



Initial DAG

$$\begin{aligned} f(f(f(a))) = a \ &\Rightarrow \ \textsc{merge}\ 3\ 0 \quad P_3 = \{4\} \ \ P_0 = \{1\} \\ &\Rightarrow \ \textsc{merge}\ 4\ 1 \quad P_4 = \{5\} \ \ P_1 = \{2\} \\ &\Rightarrow \ \textsc{merge}\ 5\ 2 \quad P_5 = \{\} \ \ P_2 = \{3\} \end{aligned}$$

$$\begin{aligned} f(f(f(f(f(a))))) = a \ &\Rightarrow \ \textsc{merge}\ 5\ 0 \quad P_5 = \{3\} \ \ P_0 = \{1, 4\} \\ &\Rightarrow \ \textsc{merge}\ 3\ 1 \quad P_3 = \{1, 3, 4\}, P_1 = \{2, 5\} \end{aligned}$$

$\textsc{find}\ f(a) = f(a) = \textsc{find}\ a \ \Rightarrow$ **Unsatisfiable**

Given Σ_E-formula

$$F : \; f(f(f(a))) = a \wedge f(f(f(f(f(a))))) = a \wedge f(a) \neq a \,,$$

which induces the initial partition

1. $\{\{a\}, \; \{f(a)\}, \; \{f^2(a)\}, \; \{f^3(a)\}, \; \{f^4(a)\}, \; \{f^5(a)\}\}$.
   The equality $f^3(a) = a$ induces the partition
2. $\{\{a, \; f^3(a)\}, \; \{f(a), \; f^4(a)\}, \; \{f^2(a), \; f^5(a)\}\}$.
   The equality $f^5(a) = a$ induces the partition
3. $\{\{a, \; f(a), \; f^2(a), \; f^3(a), \; f^4(a), \; f^5(a)\}\}$.
   Now, does

$$\{\{a, f(a), f^2(a), f^3(a), f^4(a), f^5(a)\}\} \models F \;?$$

No, as $f(a) \sim a$, but F asserts that $f(a) \neq a$. Hence, F is T_E-unsatisfiable.

Correctness of the Algorithm

Theorem (Sound and Complete)

Quantifier-free conjunctive Σ_E-formula F is T_E-satisfiable iff the congruence closure algorithm returns satisfiable.

Proof:

$\Rightarrow$ Let I be a satisfying interpretation.
By induction over the steps of the algorithm one can prove:
Whenever the algorithm merges nodes t_1 and t_2, $I \models t_1 = t_2$ holds.

Since $I \models s_i \neq t_i$ for $i \in \{m+1, \ldots, n\}$ they cannot be merged.

Hence the algorithm returns satisfiable.

Correctness of the Algorithm (2)

Proof:

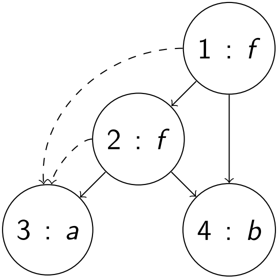
$\Leftarrow$ Let S denote the nodes of the graph and
Let $[t] := \{t' \mid t \sim t'\}$ denote the congruence class of t and
$S/\sim := \{[t] \mid t \in S\}$ denote the set of congruence classes.
Show that there is an interpretation I:

$$D_I = S/\sim \cup \{\Omega\}$$

$$\alpha_I[f](v_1, \ldots, v_n) = \begin{cases} [f(t_1, \ldots, t_n)] & v_1 = [t_1], \ldots, v_n = [t_n], \\ & f(t_1, \ldots, t_n) \in S \\ \Omega & \text{otherwise} \end{cases}$$

$$\alpha_I[=](v_1, v_2) = \top \text{ iff } v_1 = v_2$$

I is well-defined!
$\alpha_I[=]$ is a congruence relation,
$I \models F$.



$S = \{f(f(a, b), b), f(a, b), a, b\}$
$S/\sim = \{\{f(f(a, b), b), f(a, b), a\}, \{b\}\} = \{[a], [b]\}$
$D_I = \{[a], [b], \Omega\}$

$\alpha_I[f]$	$[a]$	$[b]$	Ω
$[a]$	Ω	$[a]$	Ω
$[b]$	Ω	Ω	Ω
Ω	Ω	Ω	Ω

$\alpha_I[=]$	$[a]$	$[b]$	Ω
$[a]$	$\top$	$\bot$	$\bot$
$[b]$	$\bot$	$\top$	$\bot$
Ω	$\bot$	$\bot$	$\top$

How to handle predicates?

We can get rid of predicates by

- Introduce fresh constant $\bullet$ corresponding to $\top$.
- Introduce a fresh function f_p for each predicate p.
- Replace $p(t_1, \dots, t_n)$ with $f_p(t_1, \dots, t_n) = \bullet$.

Compare the equivalence axiom for p
with the congruence axiom for f_p.

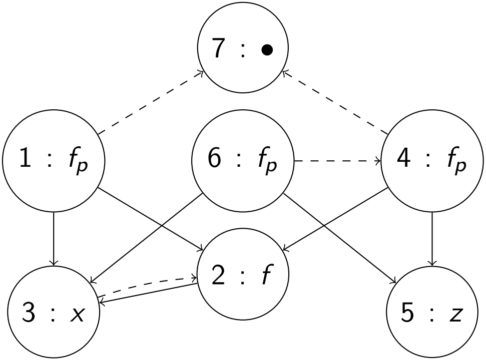
- $\forall x_1, x_2, y_1, y_2.\ x_1 = y_1 \wedge x_2 = y_2 \rightarrow p(x_1, x_2) \leftrightarrow p(y_1, y_2)$
- $\forall x_1, x_2, y_1, y_2.\ x_1 = y_1 \wedge x_2 = y_2 \rightarrow f_p(x_1, x_2) = f_p(y_1, y_2)$

Example

$$x = f(x) \land p(x, f(x)) \land p(f(x), z) \land \neg p(x, z)$$

is rewritten to

$$x = f(x) \land f_p(x, f(x)) = \bullet \land f_p(f(x), z) = \bullet \land f_p(x, z) \neq \bullet$$



FIND $f_p(x, z) = \bullet$

FIND $\bullet = \bullet$

$\Rightarrow$ **Unsatisfiable**

Theory of Lists

Theory of Lists T_{cons}

Σ_{cons} : $\{\text{cons}, \text{car}, \text{cdr}, \text{atom}, =\}$

- constructor cons: $\text{cons}(a, b)$ list constructed by prepending a to b
- left projector car: $\text{car}(\text{cons}(a, b)) = a$
- right projector cdr: $\text{cdr}(\text{cons}(a, b)) = b$
- atom: unary predicate

Axioms of T_{cons}

- reflexivity, symmetry, transitivity
- congruence axioms:

$$\forall x_1, x_2, y_1, y_2.\ x_1 = x_2 \ \wedge\ y_1 = y_2 \ \rightarrow\ \mathsf{cons}(x_1, y_1) = \mathsf{cons}(x_2, y_2)$$
$$\forall x, y.\ x = y \ \rightarrow\ \mathsf{car}(x) = \mathsf{car}(y)$$
$$\forall x, y.\ x = y \ \rightarrow\ \mathsf{cdr}(x) = \mathsf{cdr}(y)$$

- equivalence axiom:

$$\forall x, y.\ x = y \ \rightarrow\ (\mathsf{atom}(x) \ \leftrightarrow\ \mathsf{atom}(y))$$

- $\forall x, y.\ \mathsf{car}(\mathsf{cons}(x, y)) = x$ (left projection)
  $\forall x, y.\ \mathsf{cdr}(\mathsf{cons}(x, y)) = y$ (right projection)
  $\forall x.\ \neg\mathsf{atom}(x) \rightarrow \mathsf{cons}(\mathsf{car}(x), \mathsf{cdr}(x)) = x$ (construction)
  $\forall x, y.\ \neg\mathsf{atom}(\mathsf{cons}(x, y))$ (atom)

First simplify the formula:

- Consider only conjunctive $\Sigma_{\text{cons}} \cup \Sigma_{\text{E}}$-formulae.
  Convert non-conjunctive formula to DNF and check each disjunct.
- $\neg\text{atom}(u_i)$ literals are removed:

  $$\text{replace} \quad \neg\text{atom}(u_i) \quad \text{with} \quad u_i = \text{cons}(u_i^1, u_i^2)$$

  by the (construction) axiom.

Result is a conjunctive $\Sigma_{\text{cons}} \cup \Sigma_{\text{E}}$-formula with the literals:

- $s = t$
- $s \neq t$
- $atom(u)$

where s, t, u are $T_{\text{cons}} \cup T_{\text{E}}$-terms.

Algorithm: T_{cons}-Satisfiability (the idea)

$$
\begin{array}{llc}
F: & & \underbrace{s_1 = t_1 \ \wedge \ \cdots \ \wedge \ s_m = t_m}_{\text{generate congruence closure}} \\
 & \wedge & \underbrace{s_{m+1} \neq t_{m+1} \ \wedge \ \cdots \ \wedge \ s_n \neq t_n}_{\text{search for contradiction}} \\
 & \wedge & \underbrace{\mathsf{atom}(u_1) \ \wedge \ \cdots \ \wedge \ \mathsf{atom}(u_\ell)}_{\text{search for contradiction}}
\end{array}
$$

where s_i, t_i, and u_i are $T_{\mathsf{cons}} \cup T_{\mathsf{E}}$-terms.

Algorithm: T_{cons}-Satisfiability

1. Construct the initial DAG for S_F
2. for each node n with $n.\texttt{fn} = \text{cons}$
   - add $\text{car}(n)$ and MERGE $\text{car}(n)$ $n.\texttt{args}[1]$
   - add $\text{cdr}(n)$ and MERGE $\text{cdr}(n)$ $n.\texttt{args}[2]$

   by axioms (left projection), (right projection)
3. for $1 \leq i \leq m$, MERGE s_i t_i
4. for $m + 1 \leq i \leq n$, if FIND s_i = FIND t_i, return **unsatisfiable**
5. for $1 \leq i \leq \ell$, if $\exists v.$ FIND v = FIND $u_i \land v.\texttt{fn} = \text{cons}$, return **unsatisfiable**
6. Otherwise, return **satisfiable**

Example

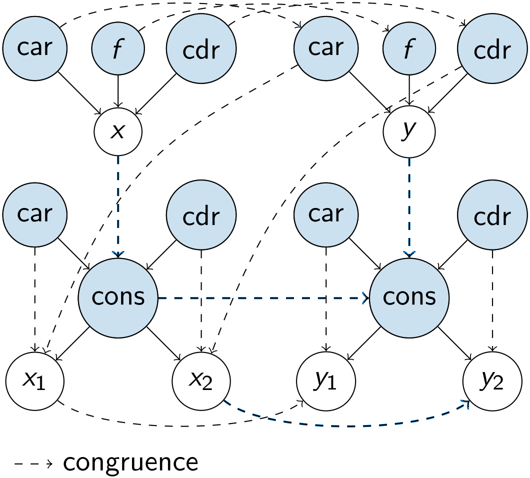
Given $(\Sigma_{\text{cons}} \cup \Sigma_{\text{E}})$-formula

$$F : \quad \begin{array}{c} \text{car}(x) = \text{car}(y) \;\wedge\; \text{cdr}(x) = \text{cdr}(y) \\ \wedge\, \neg\text{atom}(x) \;\wedge\; \neg\text{atom}(y) \;\wedge\; f(x) \neq f(y) \end{array}$$

where the function symbol f is in Σ_{E}

$$F' : \quad \begin{array}{lr} \text{car}(x) = \text{car}(y) \quad \wedge & (1) \\ \text{cdr}(x) = \text{cdr}(y) \quad \wedge & (2) \\ x = \text{cons}(x_1, x_2) \quad \wedge & (3) \\ y = \text{cons}(y_1, y_2) \quad \wedge & (4) \\ f(x) \neq f(y) & (5) \end{array}$$

Example: $\mathrm{car}(x) = \mathrm{car}(y) \wedge \mathrm{cdr}(x) = \mathrm{cdr}(y) \wedge$ $x = cons(x_1, x_2) \wedge y = cons(y_1, y_2) \wedge f(x) \neq f(y)$



Step 1

Step 2

Step 3 :

MERGE $\mathrm{car}(x)$ $\mathrm{car}(y)$

MERGE $\mathrm{cdr}(x)$ $\mathrm{cdr}(y)$

MERGE x $\mathrm{cons}(x_1, x_2)$

MERGE $\mathrm{car}(x)$ $\mathrm{car}(\mathrm{cons}(x_1, x_2))$

MERGE $\mathrm{cdr}(x)$ $\mathrm{cdr}(\mathrm{cons}(x_1, x_2))$

MERGE y $\mathrm{cons}(y_1, y_2)$

MERGE $\mathrm{car}(y)$ $\mathrm{car}(\mathrm{cons}(y_1, y_2))$

MERGE $\mathrm{cdr}(y)$ $\mathrm{cdr}(\mathrm{cons}(y_1, y_2))$

MERGE $\mathrm{cons}(x_1, x_2)$ $\mathrm{cons}(y_1, y_2)$

MERGE $f(x)$ $f(y)$

Step 4 :

FIND $f(x)$ = FIND $f(y)$

$\Rightarrow$ *unsatisfiable*

Theorem (Sound and Complete)

Quantifier-free conjunctive Σ_{cons}-formula F is T_{cons}-satisfiable iff the congruence closure algorithm for T_{cons} returns satisfiable.

Proof:

$\Rightarrow$ Let I be a satisfying interpretation.
By induction over the steps of the algorithm one can prove:
Whenever the algorithm merges nodes t_1 and t_2, $I \models t_1 = t_2$ holds.

Since $I \models s_i \neq t_i$ for $i \in \{m+1, \ldots, n\}$ they cannot be merged.
From $I \models \neg atom(\text{cons}(t_1, t_2))$ and $I \models atom(u_i)$
follows $I \models u_i \neq \text{cons}(t_1, t_2)$ by equivalence axiom.
Thus u_i for $i \in \{1, \ldots, \ell\}$ cannot be merged with a cons node.

Hence the algorithm returns satisfiable.

Correctness of the Algorithm (2)

Proof:

$\Leftarrow$ Let S denote the nodes of the graph and
let $S/\!\sim$ denote the congruence classes computed by the algorithm.
Show that there is an interpretation I:

$$
\begin{aligned}
D_I = {} & \{\text{binary trees with leaves labelled with } [t] \in S/\!\sim \text{ or } \Omega\} \\
& \setminus \{\text{trees with subtree } {}_{[t_1]}\!\swarrow\!\searrow_{[t_2]} \text{ with } \mathrm{cons}(t_1, t_2) \in S\}
\end{aligned}
$$

$$
\mathrm{cons}_I(v_1, v_2) = \begin{cases} [cons(t_1, t_2)] & v_1 = [t_1], v_2 = [t_2], \mathrm{cons}(t_1, t_2) \in S \\ {}_{v_1}\!\swarrow\!\searrow_{v_2} & \text{otherwise} \end{cases}
$$

$$
\mathrm{car}_I(v) = \begin{cases} [car(t)] & \text{if } v = [t], \mathrm{car}(t) \in S \\ v_1 & \text{if } v = {}_{v_1}\!\swarrow\!\searrow_{v_2} \\ \Omega & \text{otherwise} \end{cases}
$$

$$\mathrm{cdr}_I(v) = \begin{cases} [cdr(t)] & \text{if } v = [t], \mathrm{cdr}(t) \in S \\ v_2 & \text{if } v = \overset{\swarrow\ \searrow}{v_1 \quad v_2} \\ \Omega & \text{otherwise} \end{cases}$$

$$atom_I(v) = \begin{cases} \text{false} & \text{if } v = [cons(t_1, t_2)] \\ \text{false} & \text{if } v = \overset{\swarrow\ \searrow}{v_1 \quad v_2} \\ \text{true} & \text{otherwise} \end{cases}$$

$$\alpha_I[=](v_1, v_2) = \text{true iff } v_1 = v_2$$

I is well-defined! $\alpha_I[=]$ is obviously a congruence relation.

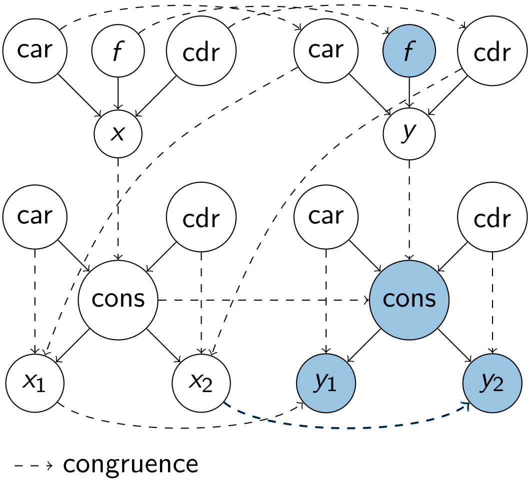
$\forall x, y.\ \mathrm{car}(\mathrm{cons}(x, y)) = x$ (left projection)

$\forall x, y.\ \mathrm{cdr}(\mathrm{cons}(x, y)) = y$ (right projection)

$\forall x.\ \neg\mathrm{atom}(x) \rightarrow \mathrm{cons}(\mathrm{car}(x), \mathrm{cdr}(x)) = x$ (construction)

$\forall x, y.\ \neg\mathrm{atom}(\mathrm{cons}(x, y))$ (atom)

Example: car(x) = car(y) $\wedge$ cdr(x) = cdr(y) $\wedge$ $x = cons(x_1, x_2) \wedge y = cons(y_1, y_2)$



$$S/\sim \; = \{[y], [f(y)], [y_1], [y_2]\}$$

$$D_I = \left\{ \begin{array}{l} \text{trees over } [y], [f(y)], [y_1], [y_2], \text{ and } \Omega \text{ not containing} \\ \quad [y_1] \swarrow \searrow [y_2] \end{array} \right\}$$

$$cons([y], \Omega) = [y] \swarrow \searrow \Omega$$

$$cons([y_1], [y_2]) = [y]$$

$$car([y]) = [y_1]$$

$$cdr([y]) = [y_2]$$

$$car([f(y)]) = \Omega$$

$$car(\; [y] \swarrow \searrow \Omega \;) = [y]$$