

Decision Procedures

Jochen Hoenicke



Software Engineering
Albert-Ludwigs-University Freiburg

Summer 2012

Quantifier-free Theory of Equality

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

uninterpreted symbols:

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

- 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 ,

$$\forall x_1, \dots, x_n, y_1, \dots, y_n. \bigwedge_i x_i = y_i \rightarrow f(x_1, \dots, x_n) = f(y_1, \dots, y_n)$$
 (congruence)
- ⑤ for each positive integer n and n -ary predicate symbol p ,

$$\forall x_1, \dots, x_n, y_1, \dots, y_n. \bigwedge_i x_i = y_i \rightarrow (p(x_1, \dots, x_n) \leftrightarrow p(y_1, \dots, y_n))$$
 (equivalence)

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

$$F : \underbrace{s_1 = t_1 \wedge \cdots \wedge s_m = t_m}_{\text{generate congruence closure}} \wedge \underbrace{s_{m+1} \neq t_{m+1} \wedge \cdots \wedge s_n \neq t_n}_{\text{search for contradiction}}$$

$$F : \underbrace{s_1 = t_1 \wedge \cdots \wedge s_m = t_m}_{\text{generate congruence closure}} \wedge \underbrace{s_{m+1} \neq t_{m+1} \wedge \cdots \wedge s_n \neq t_n}_{\text{search for contradiction}}$$

The algorithm performs the following steps:

- 1 Construct the congruence closure \sim of

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

over the subterm set S_F . Then

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

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

$$F : \underbrace{s_1 = t_1 \wedge \cdots \wedge s_m = t_m}_{\text{generate congruence closure}} \wedge \underbrace{s_{m+1} \neq t_{m+1} \wedge \cdots \wedge s_n \neq t_n}_{\text{search for contradiction}}$$

The algorithm performs the following steps:

- 1 Construct the congruence closure \sim of

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

over the subterm set S_F . Then

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

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

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

Begin with the **finest congruence** relation \sim_0 :

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

Each term of S_F is only congruent to itself.

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 $i \in \{1, \dots, m\}$, impose $s_i = t_i$ by merging

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

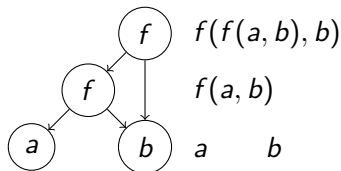
to form a new congruence relation \sim_i . To accomplish this merging,

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

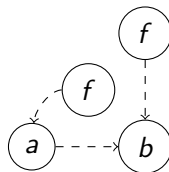
The new relation \sim_i is a congruence relation in which $s_i \sim t_i$.

Efficient data structure for computing the congruence closure.

- **Directed Acyclic Graph (DAG)** to represent terms.



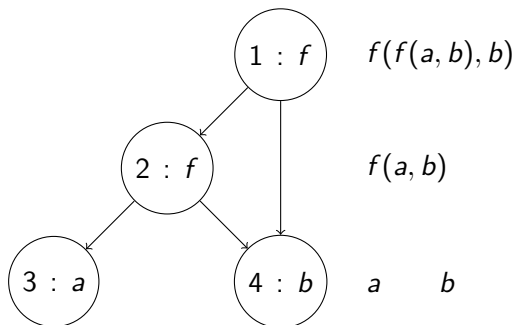
- **Union-Find data structure** to represent equivalence classes:



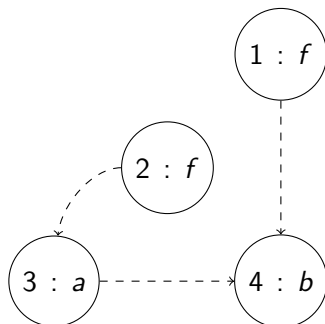
For every subterm of the Σ_E -formula F , create

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

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



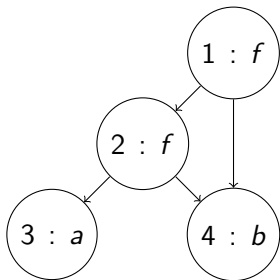
Equivalence classes are connected by a tree structure, with arrows pointing to the root node.



Two operations are defined:

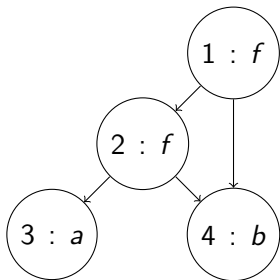
- **FIND**: Find the representative of an equivalence class by following the edges. $O(\log n)$
- **UNION**: Merge two classes by connecting the representatives. $O(1)$

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

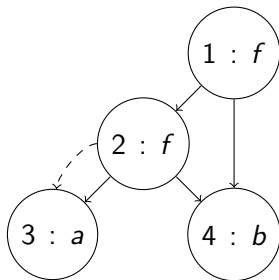


Initial DAG

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

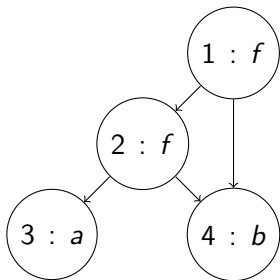


Initial DAG

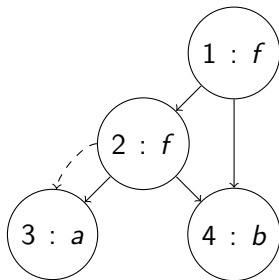


$f(a, b) = a \Rightarrow$
MERGE $f(a, b)$ a

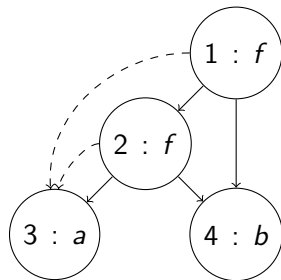
$$f(a, b) = a \wedge f(f(a, b), b) \neq a$$



Initial DAG

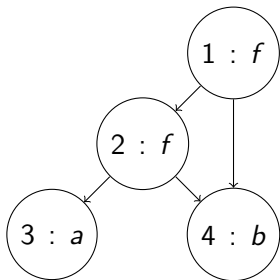


$f(a, b) = a \Rightarrow$
MERGE $f(a, b)$ a

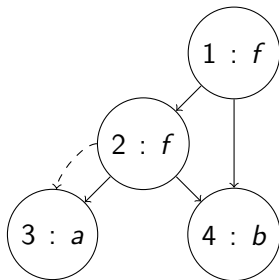


$f(a, b) \sim a, b \sim b \Rightarrow$
 $f(f(a, b), b) \sim f(a, b)$
MERGE $f(f(a, b), b)$
 $f(a, b)$

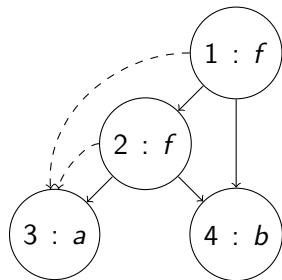
$$f(a, b) = a \wedge f(f(a, b), b) \neq a$$



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

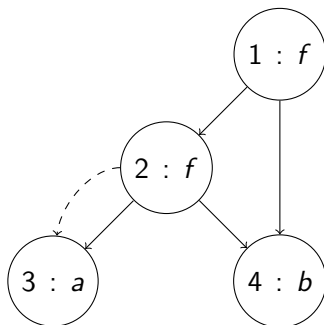
$$\left. \begin{array}{l} \text{FIND } f(f(a, b), b) = a = \text{FIND } a \\ f(f(a, b), b) \neq a \end{array} \right\} \Rightarrow \text{Unsatisfiable}$$

```
type node = {  
    id           : id  
                node's unique identification number  
  
    fn          : string  
                constant or function name  
  
    args       : id list  
                list of function arguments  
  
}
```

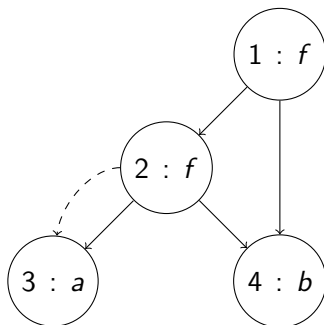
```
type node = {  
    id           : id  
                 node's unique identification number  
  
    fn          : string  
                 constant or function name  
  
    args        : id list  
                 list of function arguments  
  
    mutable find : id  
                 the edge to the representative  
  
}
```

```
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 cpar : id set  
                 if the node is the representative for its  
                 congruence class, then its cpar  
                 (congruence closure parents) are all  
                 parents of nodes in its congruence class  
  
}
```

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



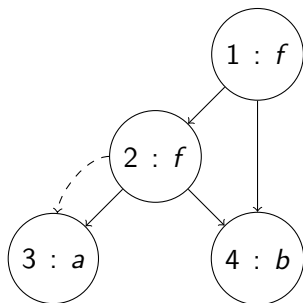
```
type node = {  
  id          : id      ... 3  
  fn          : string ... a  
  args       : idlist  ... []  
  mutable find : id      ... 3  
  mutable cpar : idset  ... {1,2}  
}
```



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  $n.find$ 
```

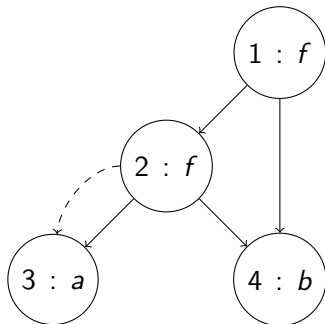


Example: $\text{FIND } 2 = \text{FIND } 3 = 3$
3 is the representative of 2.

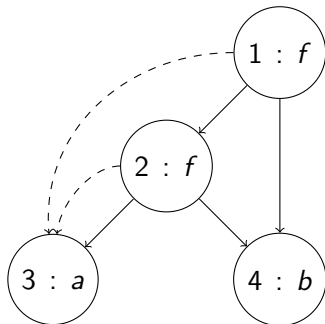
UNION function

```
let UNION  $i_1$   $i_2$  =  
  let  $n_1$  = NODE (FIND  $i_1$ ) in  
  let  $n_2$  = NODE (FIND  $i_2$ ) in  
   $n_1$ .find  $\leftarrow$   $n_2$ .find;  
   $n_2$ .ccpar  $\leftarrow$   $n_1$ .ccpar  $\cup$   $n_2$ .ccpar;  
   $n_1$ .ccpar  $\leftarrow$   $\emptyset$ 
```

n_2 is the representative of the union class



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



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

1.find $\leftarrow 3$

3.ccpair $\leftarrow \{1, 2\}$

1.ccpair $\leftarrow \emptyset$

CCPAR function

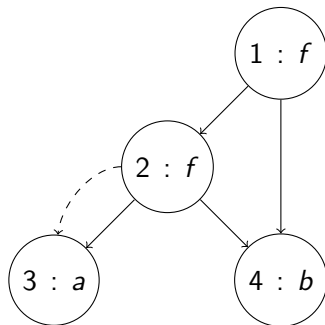
Returns parents of all nodes in i 's congruence class

```
let CCPAR  $i$  =  
  (NODE (FIND  $i$ )).ccpar
```

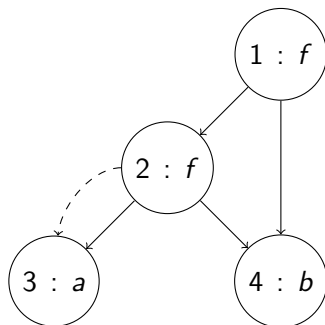
CONGRUENT predicate

Test whether i_1 and i_2 are congruent

```
let CONGRUENT  $i_1$   $i_2$  =  
  let  $n_1$  = NODE  $i_1$  in  
  let  $n_2$  = NODE  $i_2$  in  
   $n_1$ .fn =  $n_2$ .fn  
   $\wedge |n_1$ .args| =  $|n_2$ .args|  
   $\wedge \forall i \in \{1, \dots, |n_1$ .args|\}. FIND  $n_1$ .args[ $i$ ] = FIND  $n_2$ .args[ $i$ ]
```



Are 1 and 2 congruent?



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  $i_1$   $i_2$  =  
  if FIND  $i_1$   $\neq$  FIND  $i_2$  then begin  
    let  $P_{i_1}$  = CCPAR  $i_1$  in  
    let  $P_{i_2}$  = CCPAR  $i_2$  in  
    UNION  $i_1$   $i_2$ ;  
    foreach  $t_1, t_2 \in P_{i_1} \times P_{i_2}$  do  
      if FIND  $t_1$   $\neq$  FIND  $t_2$   $\wedge$  CONGRUENT  $t_1$   $t_2$   
      then MERGE  $t_1$   $t_2$   
    done  
  end
```

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

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 $i \in \{1, \dots, m\}$, MERGE s_i t_i .
- 3 If FIND $s_i =$ FIND t_i for some $i \in \{m + 1, \dots, n\}$, return unsatisfiable.
- 4 Otherwise (if FIND $s_i \neq$ FIND t_i for all $i \in \{m + 1, \dots, n\}$) return satisfiable.

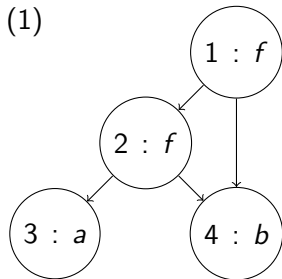
Example $f(a, b) = a \wedge f(f(a, b), b) \neq a$

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

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

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

(1)

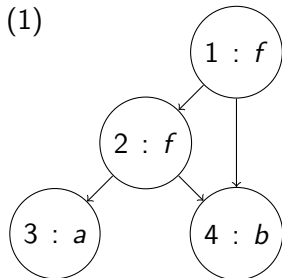


Initial DAG

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

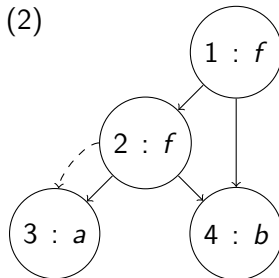
$$f(a, b) = a \wedge f(f(a, b), b) \neq a$$

(1)



Initial DAG

(2)



MERGE 2 3

UNION 2 3

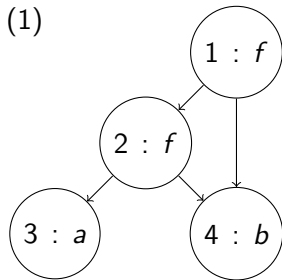
$$P_2 = \{1\}$$

$$P_3 = \{2\}$$

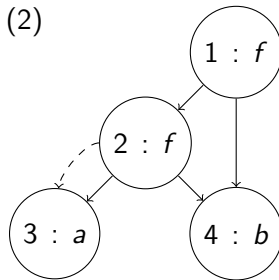
CONGRUENT 1 2

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

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



Initial DAG



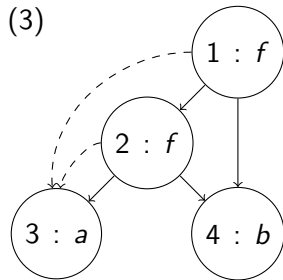
MERGE 2 3

UNION 2 3

$$P_2 = \{1\}$$

$$P_3 = \{2\}$$

CONGRUENT 1 2



MERGE 1 2

UNION 1 2

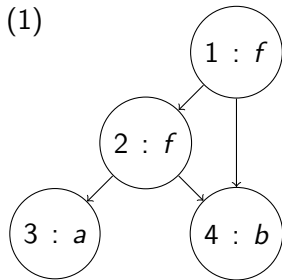
$$P_1 = \{\}$$

$$P_2 = \{1, 2\}$$

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

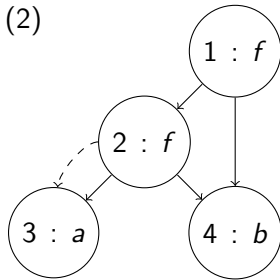
$$f(a, b) = a \wedge f(f(a, b), b) \neq a$$

(1)



Initial DAG

(2)



MERGE 2 3

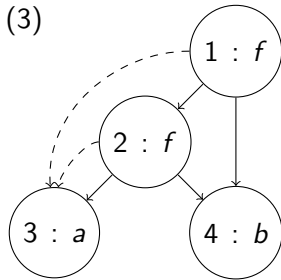
UNION 2 3

$$P_2 = \{1\}$$

$$P_3 = \{2\}$$

CONGRUENT 1 2

(3)



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

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

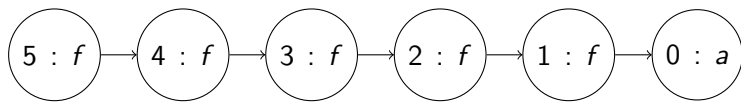
Does

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

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

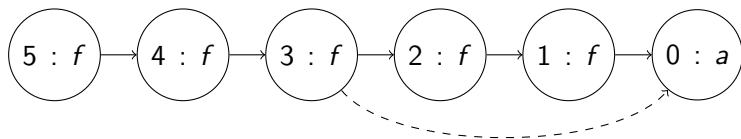
$$f(f(f(a))) = a \wedge f(f(f(f(f(a)))))) = a \wedge f(a) \neq a$$



Initial DAG

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

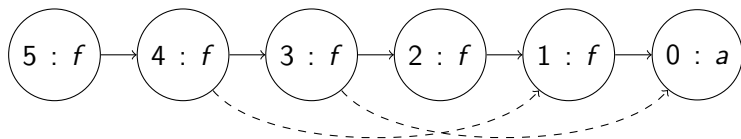
$$f(f(f(a))) = a \wedge f(f(f(f(f(a)))))) = a \wedge f(a) \neq a$$



$$f(f(f(a))) = a \Rightarrow \text{MERGE } 3 \ 0 \quad P_3 = \{4\} \quad P_0 = \{1\}$$

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

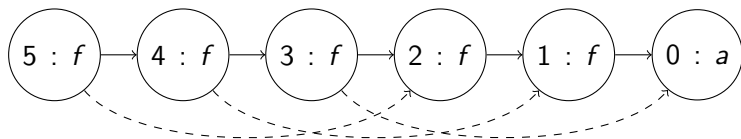
$$f(f(f(a))) = a \wedge f(f(f(f(f(a)))))) = a \wedge f(a) \neq a$$



$$\begin{aligned}
 f(f(f(a))) = a &\Rightarrow \text{MERGE } 3 \ 0 & P_3 &= \{4\} & P_0 &= \{1\} \\
 &\Rightarrow \text{MERGE } 4 \ 1 & P_4 &= \{5\} & P_1 &= \{2\}
 \end{aligned}$$

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

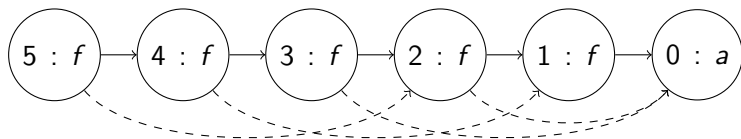
$$f(f(f(a))) = a \wedge f(f(f(f(f(a)))))) = a \wedge f(a) \neq a$$



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

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

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

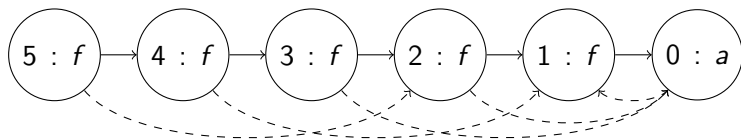


$$\begin{aligned} f(f(f(a))) = a &\Rightarrow \text{MERGE } 3 \ 0 & P_3 &= \{4\} & P_0 &= \{1\} \\ &\Rightarrow \text{MERGE } 4 \ 1 & P_4 &= \{5\} & P_1 &= \{2\} \\ &\Rightarrow \text{MERGE } 5 \ 2 & P_5 &= \{\} & P_2 &= \{3\} \end{aligned}$$

$$f(f(f(f(f(a)))))) = a \Rightarrow \text{MERGE } 5 \ 0 \quad P_5 = \{3\} \quad P_0 = \{1, 4\}$$

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

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

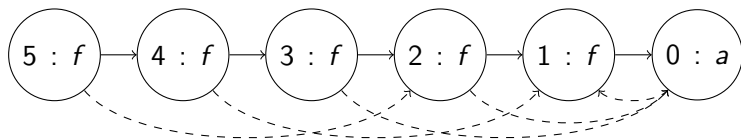


$$\begin{aligned} f(f(f(a))) = a &\Rightarrow \text{MERGE } 3 \ 0 & P_3 &= \{4\} & P_0 &= \{1\} \\ &\Rightarrow \text{MERGE } 4 \ 1 & P_4 &= \{5\} & P_1 &= \{2\} \\ &\Rightarrow \text{MERGE } 5 \ 2 & P_5 &= \{\} & P_2 &= \{3\} \end{aligned}$$

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

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

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



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

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

FIND $f(a) = f(a) = \text{FIND } a \Rightarrow$ **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.

Theorem (Sound and Complete)

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

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 $i \in \{m + 1, \dots, n\}$ they cannot be merged.

Hence the algorithm returns satisfiable.

Proof:

- ⇐ 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 :

Proof:

⇐ 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 :

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

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

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

Proof:

⇐ 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 :

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

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

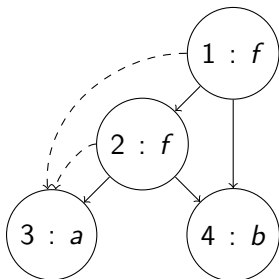
$$\alpha_I[=](v_1, v_2) = \top \text{ iff } v_1 = v_2$$

I is well-defined!

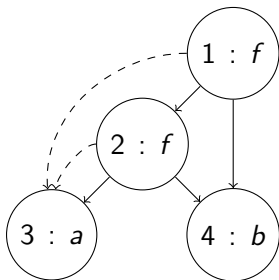
$\alpha_I[=]$ is a congruence relation,

$I \models F$.

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



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



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

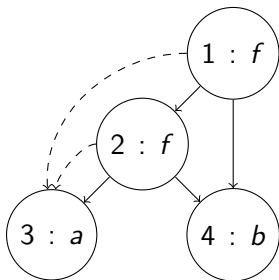
$$S/\sim = \{\{f(f(a, b), b), f(a, b), a\}, \{b\}\} = \{[a], [b]\}$$

$$D_I = \{[a], [b], \Omega\}$$

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

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

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



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

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

$$D_I = \{[a], [b], \Omega\}$$

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

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

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 .

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

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

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

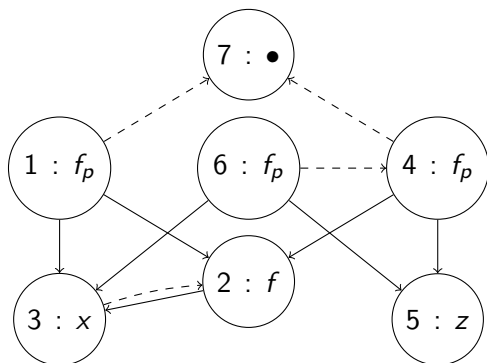
is rewritten to

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

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

is rewritten to

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



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

FIND $\bullet = \bullet$

\Rightarrow **Unsatisfiable**

Theory of Lists

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

- **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

- reflexivity, symmetry, transitivity
- congruence axioms:

$$\forall x_1, x_2, y_1, y_2. x_1 = x_2 \wedge y_1 = y_2 \rightarrow \text{cons}(x_1, y_1) = \text{cons}(x_2, y_2)$$

$$\forall x, y. x = y \rightarrow \text{car}(x) = \text{car}(y)$$

$$\forall x, y. x = y \rightarrow \text{cdr}(x) = \text{cdr}(y)$$

- equivalence axiom:

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

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

First simplify the formula:

- Consider only conjunctive $\Sigma_{\text{CONS}} \cup \Sigma_E$ -formulae.
Convert non-conjunctive formula to DNF and check each disjunct.
- $\neg \text{atom}(u_i)$ literals are removed:
replace $\neg \text{atom}(u_i)$ with $u_i = \text{cons}(u_i^1, u_i^2)$
by the (construction) axiom.

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:
replace $\neg \text{atom}(u_i)$ with $u_i = \text{cons}(u_i^1, u_i^2)$
by the (construction) axiom.

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

- $s = t$
- $s \neq t$
- $\text{atom}(u)$

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

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

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

- 1 Construct the initial DAG for S_F

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

- ① Construct the initial DAG for S_F
- ② for each node n with $n.\text{fn} = \text{cons}$
 - add $\text{car}(n)$ and MERGE $\text{car}(n) \ n.\text{args}[1]$
 - add $\text{cdr}(n)$ and MERGE $\text{cdr}(n) \ n.\text{args}[2]$by axioms (left projection), (right projection)
- ③ for $1 \leq i \leq m$, MERGE $s_i \ t_i$

- 1 Construct the initial DAG for S_F
- 2 for each node n with $n.\text{fn} = \text{cons}$
 - add $\text{car}(n)$ and $\text{MERGE } \text{car}(n) \ n.\text{args}[1]$
 - add $\text{cdr}(n)$ and $\text{MERGE } \text{cdr}(n) \ n.\text{args}[2]$by axioms (left projection), (right projection)
- 3 for $1 \leq i \leq m$, $\text{MERGE } s_i \ t_i$
- 4 for $m + 1 \leq i \leq n$, if $\text{FIND } s_i = \text{FIND } t_i$, return **unsatisfiable**

- ① Construct the initial DAG for S_F
- ② for each node n with $n.\text{fn} = \text{cons}$
 - add $\text{car}(n)$ and $\text{MERGE } \text{car}(n) \ n.\text{args}[1]$
 - add $\text{cdr}(n)$ and $\text{MERGE } \text{cdr}(n) \ n.\text{args}[2]$by axioms (left projection), (right projection)
- ③ for $1 \leq i \leq m$, $\text{MERGE } s_i \ t_i$
- ④ for $m + 1 \leq i \leq n$, if $\text{FIND } s_i = \text{FIND } t_i$, return **unsatisfiable**
- ⑤ for $1 \leq i \leq \ell$, if $\exists v. \text{FIND } v = \text{FIND } u_i \wedge v.\text{fn} = \text{cons}$, return **unsatisfiable**

- 1 Construct the initial DAG for S_F
- 2 for each node n with $n.\text{fn} = \text{cons}$
 - add $\text{car}(n)$ and MERGE $\text{car}(n) \ n.\text{args}[1]$
 - add $\text{cdr}(n)$ and MERGE $\text{cdr}(n) \ n.\text{args}[2]$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 $s_i = \text{FIND } t_i$, return **unsatisfiable**
- 5 for $1 \leq i \leq \ell$, if $\exists v. \text{FIND } v = \text{FIND } u_i \wedge v.\text{fn} = \text{cons}$, return **unsatisfiable**
- 6 Otherwise, return **satisfiable**

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

$$F : \quad \text{car}(x) = \text{car}(y) \wedge \text{cdr}(x) = \text{cdr}(y) \\ \wedge \neg \text{atom}(x) \wedge \neg \text{atom}(y) \wedge f(x) \neq f(y)$$

where the function symbol f is in Σ_E

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

$$F : \quad \begin{aligned} & \text{car}(x) = \text{car}(y) \wedge \text{cdr}(x) = \text{cdr}(y) \\ & \wedge \neg \text{atom}(x) \wedge \neg \text{atom}(y) \wedge f(x) \neq f(y) \end{aligned}$$

where the function symbol f is in Σ_E

$$\text{car}(x) = \text{car}(y) \quad \wedge \quad (1)$$

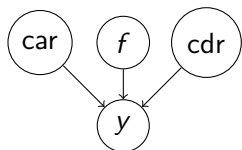
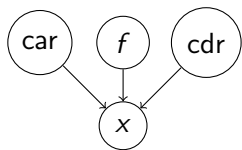
$$\text{cdr}(x) = \text{cdr}(y) \quad \wedge \quad (2)$$

$$F' : \quad x = \text{cons}(x_1, x_2) \quad \wedge \quad (3)$$

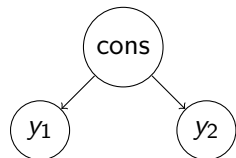
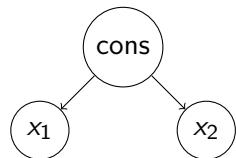
$$y = \text{cons}(y_1, y_2) \quad \wedge \quad (4)$$

$$f(x) \neq f(y) \quad (5)$$

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

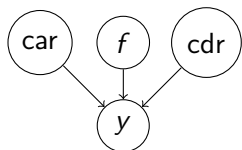
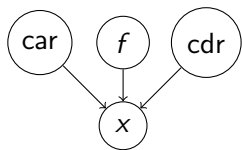


Step 1

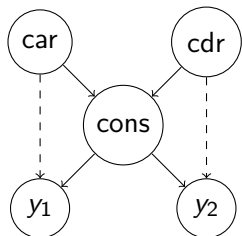
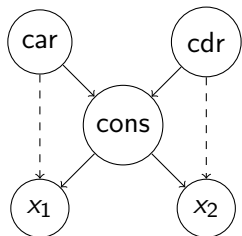


--> congruence

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

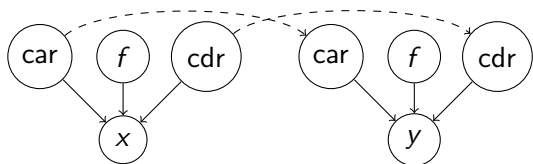


Step 1
Step 2



--> congruence

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



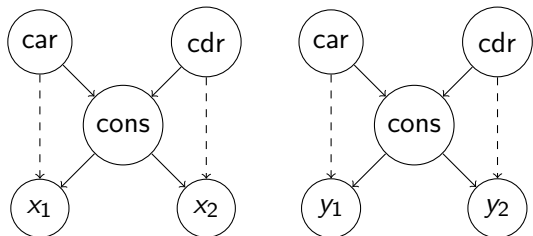
Step 1

Step 2

Step 3 :

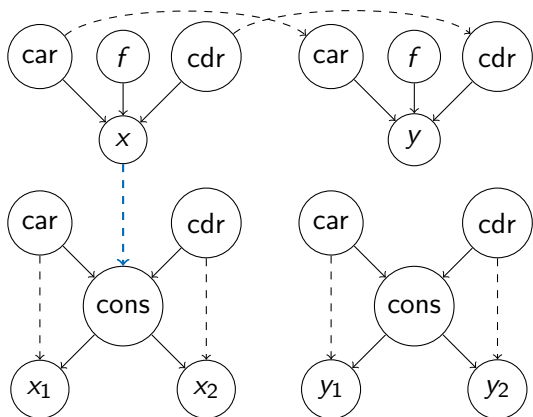
MERGE $\text{car}(x)$ $\text{car}(y)$

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



--> congruence

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



--> congruence

Step 1

Step 2

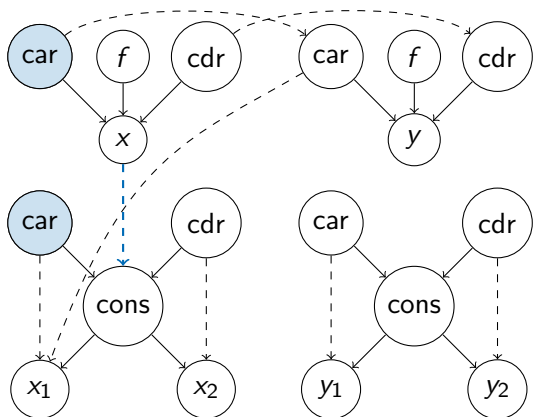
Step 3 :

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

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

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

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



--> congruence

Step 1

Step 2

Step 3 :

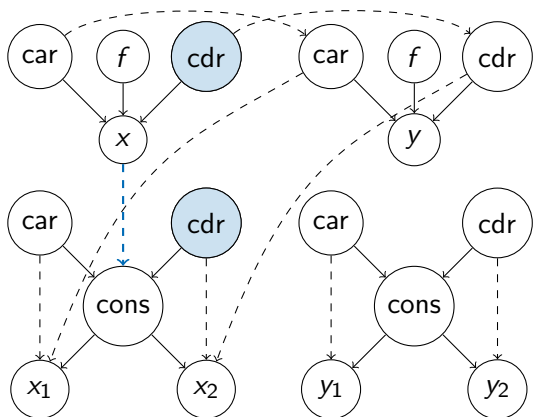
MERGE $\text{car}(x)$ $\text{car}(y)$

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

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

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

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



--> congruence

Step 1

Step 2

Step 3 :

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

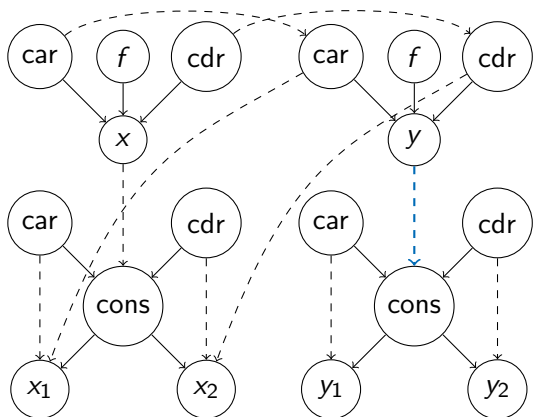
MERGE $\text{cdr}(x)$ $\text{cdr}(y)$

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

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

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

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



--> congruence

Step 1

Step 2

Step 3 :

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

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

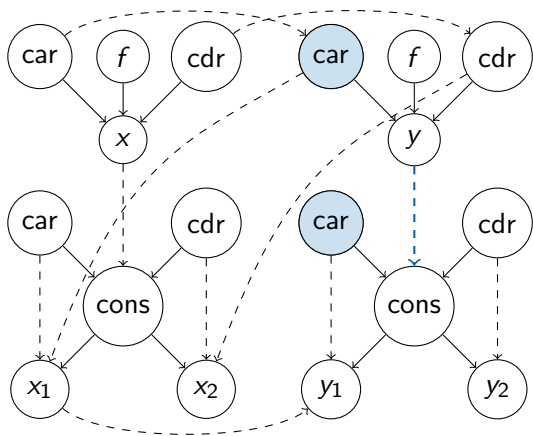
MERGE x $\text{cons}(x_1, x_2)$

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

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

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

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



--> congruence

Step 1

Step 2

Step 3 :

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

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

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

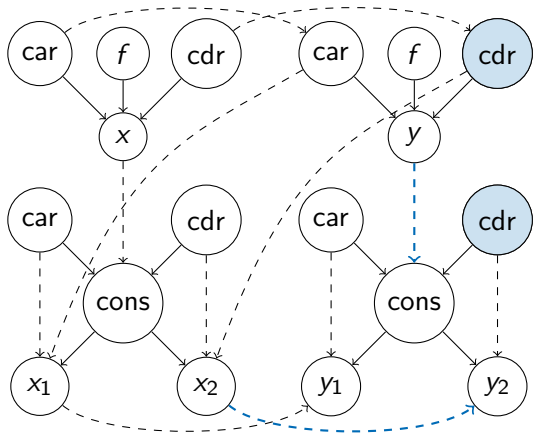
MERGE $\text{car}(x)$ $\text{car}(\text{cons}(x_1, x_2))$

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

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

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

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



--> congruence

Step 1

Step 2

Step 3 :

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

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

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

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

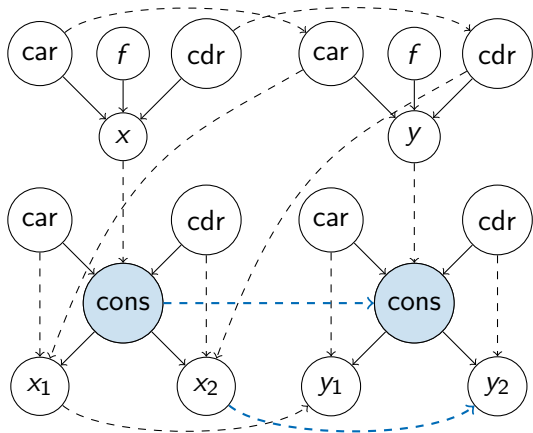
MERGE $\text{cdr}(x)$ $\text{cdr}(\text{cons}(x_1, x_2))$

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

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

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

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



--> congruence

Step 1

Step 2

Step 3 :

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

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

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

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

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

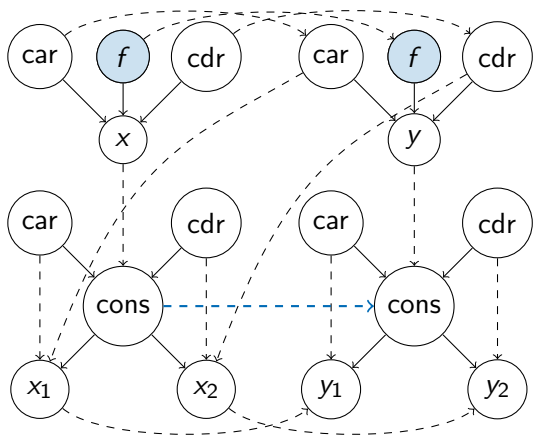
MERGE y $\text{cons}(y_1, y_2)$

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

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

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

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



--> congruence

Step 1

Step 2

Step 3 :

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

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

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

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

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

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

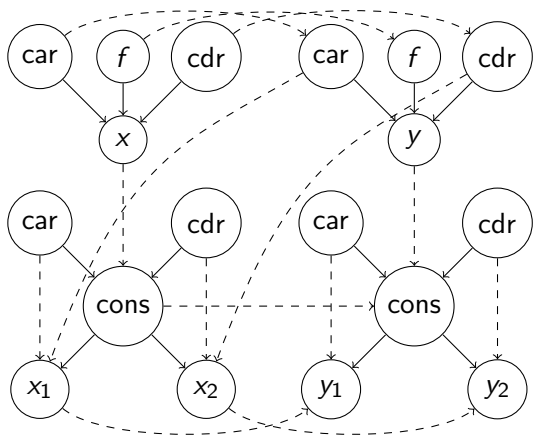
MERGE $\text{car}(y)$ $\text{car}(\text{cons}(y_1, y_2))$

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

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

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

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



--> congruence

Step 1

Step 2

Step 3 :

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

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

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

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

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

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

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

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

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

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

Step 4 :

FIND $f(x) = \text{FIND } f(y)$

\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 $i \in \{m+1, \dots, n\}$ they cannot be merged.

From $I \models \neg \text{atom}(\text{cons}(t_1, t_2))$ and $I \models \text{atom}(u_i)$

follows $I \models u_i \neq \text{cons}(t_1, t_2)$ by equivalence axiom.

Thus u_i for $i \in \{1, \dots, \ell\}$ cannot be merged with a cons node.

Hence the algorithm returns satisfiable.

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 :

$$D_I = \{\text{binary trees with leaves labelled with } S/\sim\}$$

$$\setminus \{\text{trees with subtree } \begin{array}{c} \swarrow \quad \searrow \\ [t_1] \quad [t_2] \end{array} \text{ with } \text{cons}(t_1, t_2) \in S\}$$

$$\text{cons}_I(v_1, v_2) = \begin{cases} [\text{cons}(t_1, t_2)] & v_1 = [t_1], v_2 = [t_2], \text{cons}(t_1, t_2) \in S \\ \begin{array}{c} \swarrow \quad \searrow \\ v_1 \quad v_2 \end{array} & \text{otherwise} \end{cases}$$

$$\text{car}_I(v) = \begin{cases} [\text{car}(t)] & \text{if } v = [t], \text{car}(t) \in S \\ v_1 & \text{if } v = \begin{array}{c} \swarrow \quad \searrow \\ v_1 \quad v_2 \end{array} \\ \text{arbitrary} & \text{otherwise} \end{cases}$$

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

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

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

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

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

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

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

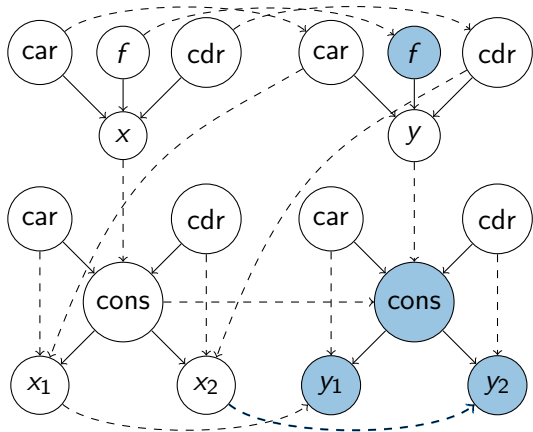
$$\forall x, y. \text{car}(\text{cons}(x, y)) = x \quad (\text{left projection})$$

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

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

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

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



--> congruence