

CS156: The Calculus of Computation

Zohar Manna

Autumn 2008

Chapter 9: Quantifier-free Equality and Data Structures

The Theory of Equality T_E

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

uninterpreted symbols:

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

Example:

$$x = y \wedge f(x) \neq f(y) \quad T_E\text{-unsatisfiable}$$

$$f(x) = f(y) \wedge x \neq y \quad T_E\text{-satisfiable}$$

$$f(f(f(a))) = a \wedge f(f(f(f(f(a)))))) = a \wedge f(a) \neq a \quad T_E\text{-unsatisfiable}$$

$$x = g(y, z) \rightarrow f(x) = f(g(y, z)) \quad T_E\text{-valid}$$

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 ,

$$\forall \bar{x}, \bar{y}. \left(\bigwedge_{i=1}^n x_i = y_i \right) \rightarrow f(\bar{x}) = f(\bar{y})$$

(function)

For example, for unary f , the axiom is

$$\forall x', y'. x' = y' \rightarrow f(x') = f(y')$$

Therefore,

$$x = g(y, z) \rightarrow f(x) = f(g(y, z))$$

is T_E -valid. ($x' \rightarrow x, y' \rightarrow g(y, z)$).

Axiom schema

5. for each positive integer n and n -ary predicate symbol p ,

$$\forall \bar{x}, \bar{y}. \left(\bigwedge_{i=1}^n x_i = y_i \right) \rightarrow (p(\bar{x}) \leftrightarrow p(\bar{y}))$$

(predicate)

Thus, for unary p , the axiom is

$$\forall x', y'. x' = y' \rightarrow (p(x') \leftrightarrow p(y'))$$

Therefore,

$$a = b \rightarrow (p(a) \leftrightarrow p(b))$$

is T_E -valid. $(x' \rightarrow a, y' \rightarrow b)$.

We discuss T_E -formulae without predicates

For example, for Σ_E -formula

$$F : p(x) \wedge q(x, y) \wedge q(y, z) \rightarrow \neg q(x, z)$$

introduce fresh constant $\bullet$ and fresh functions f_p and f_q , and transform F to

$$G : f_p(x) = \bullet \wedge f_q(x, y) = \bullet \wedge f_q(y, z) = \bullet \rightarrow f_q(x, z) \neq \bullet .$$

Equivalence and Congruence Relations: Basics

Binary relation R over set S

- is an equivalence relation if
 - ▶ reflexive: $\forall s \in S. s R s$;
 - ▶ symmetric: $\forall s_1, s_2 \in S. s_1 R s_2 \rightarrow s_2 R s_1$;
 - ▶ transitive: $\forall s_1, s_2, s_3 \in S. s_1 R s_2 \wedge s_2 R s_3 \rightarrow s_1 R s_3$.

Example:

Define the binary relation $\equiv_2$ over the set $\mathbb{Z}$ of integers

$$m \equiv_2 n \quad \text{iff} \quad (m \bmod 2) = (n \bmod 2)$$

That is, $m, n \in \mathbb{Z}$ are related iff they are both even or both odd.

$\equiv_2$ is an equivalence relation

- is a congruence relation if in addition

$$\forall \bar{s}, \bar{t}. \bigwedge_{i=1}^n s_i R t_i \rightarrow f(\bar{s}) R f(\bar{t}) .$$

Classes

For $\left\{ \begin{array}{l} \text{equivalence} \\ \text{congruence} \end{array} \right\}$ relation R over set S ,
the $\left\{ \begin{array}{l} \text{equivalence} \\ \text{congruence} \end{array} \right\}$ class of $s \in S$ under R is

$$[s]_R \stackrel{\text{def}}{=} \{s' \in S : sRs'\}.$$

Example:

The equivalence class of 3 under $\equiv_2$ over $\mathbb{Z}$ is

$$[3]_{\equiv_2} = \{n \in \mathbb{Z} : n \text{ is odd}\}.$$

Partitions

A partition P of S is a set of subsets of S that is

► total $\left(\bigcup_{S' \in P} S' \right) = S$

► disjoint $\forall S_1, S_2 \in P. S_1 \neq S_2 \rightarrow S_1 \cap S_2 = \emptyset$

Quotient

The quotient S/R of S by $\left\{ \begin{array}{l} \text{equivalence} \\ \text{congruence} \end{array} \right\}$ relation R is the partition of S into $\left\{ \begin{array}{l} \text{equivalence} \\ \text{congruence} \end{array} \right\}$ classes

$$S/R = \{[s]_R : s \in S\}.$$

It satisfies total and disjoint conditions.

Example: The quotient $\mathbb{Z}/\equiv_2$ is a partition of $\mathbb{Z}$. The set of equivalence classes

$$\{\{n \in \mathbb{Z} : n \text{ is odd}\}, \{n \in \mathbb{Z} : n \text{ is even}\}\}$$

Note duality between relations and classes

Refinements

Two binary relations R_1 and R_2 over set S .

R_1 is a refinement of R_2 , $R_1 \prec R_2$, if

$$\forall s_1, s_2 \in S. s_1 R_1 s_2 \rightarrow s_1 R_2 s_2 .$$

R_1 refines R_2 .

Examples:

- ▶ For $S = \{a, b\}$,

$$R_1 : \{a R_1 b\} \quad R_2 : \{a R_2 b, b R_2 b\}$$

Then $R_1 \prec R_2$

- ▶ For set $\mathbb{Z}$

$$R_1 : \{x R_1 y : x \bmod 2 = y \bmod 2\}$$

$$R_2 : \{x R_2 y : x \bmod 4 = y \bmod 4\}$$

Then $R_2 \prec R_1$.

Closures

Given binary relation R over S .

The equivalence closure R^E of R is the equivalence relation s.t.

- ▶ R refines R^E , i.e. $R \prec R^E$;
- ▶ for all other equivalence relations R' s.t. $R \prec R'$,
either $R' = R^E$ or $R^E \prec R'$

That is, R^E is the “smallest” equivalence relation that “covers” R .

Example: If $S = \{a, b, c, d\}$ and $R = \{aRb, bRc, dRd\}$, then

- $aR^E b, bR^E c, dR^E d$ since $R \subseteq R^E$;
- $aR^E a, bR^E b, cR^E c$ by reflexivity;
- $bR^E a, cR^E b$ by symmetry;
- $aR^E c$ by transitivity;
- $cR^E a$ by symmetry.

Similarly, the congruence closure R^C of R is the “smallest” congruence relation that “covers” R .

T_E -satisfiability and Congruence Classes I

Definition: For Σ_E -formula

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

the subterm set S_F of F is the set that contains precisely the subterms of F .

Example: The subterm set of

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

is

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

Note: we consider only quantifier-free conjunctive Σ_E -formulae.

Convert non-conjunctive formula F to DNF $\bigvee_i F_i$, where each disjunct F_i is a conjunction of $=, \neq$. Check each disjunct F_i . F is T_E -satisfiable iff at least one disjunct F_i is T_E -satisfiable.

T_E -satisfiability and Congruence Classes II

Given Σ_E -formula F

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

with subterm set S_F , F is T_E -satisfiable iff there exists a congruence relation $\sim$ over S_F such that

- ▶ for each $i \in \{1, \dots, m\}$, $s_i \sim t_i$;
- ▶ for each $i \in \{m+1, \dots, n\}$, $s_i \not\sim t_i$.

Such congruence relation $\sim$ defines T_E -interpretation $I : (D_I, \alpha_I)$ of F . D_I consists of $|S_F / \sim|$ elements, one for each congruence class of S_F under $\sim$.

Instead of writing $I \models F$ for this T_E -interpretation, we abbreviate $\sim \models F$

The goal of the algorithm is to construct the congruence relation over S_F , or to prove that no congruence relation exists.

Congruence Closure Algorithm

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

Decide if F is T_E -satisfiable.

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?

Congruence Closure Algorithm (Details)

Initially, begin with the finest congruence relation $\sim_0$ given by the partition

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

That is, let each term over S_F be its own congruence class.

Then, for each $i \in \{1, \dots, m\}$, impose $s_i = t_i$ by merging the congruence classes

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

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 $s_i \sim t_i$.

Congruence Closure Algorithm: Example 1 I

Given Σ_E -formula

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

Construct initial partition by letting each member of the subterm set S_F be its own class:

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

According to the first literal $f(a, b) = a$, merge

$$\{f(a, b)\} \quad \text{and} \quad \{a\}$$

to form partition

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

According to the (function) congruence axiom,

$$f(a, b) \sim a, \quad b \sim b \quad \text{implies} \quad f(f(a, b), b) \sim f(a, b),$$

resulting in the new partition

3. $\{\{a, f(a, b), f(f(a, b), b)\}, \{b\}\}$

Congruence Closure Algorithm: Example 1 II

This partition represents the congruence closure of S_F .

Is it the case that

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

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.

Congruence Closure Algorithm: Example 2 I

Example: Given Σ_E -formula

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

From the subterm set S_F , the initial partition is

$$1. \{\{a\}, \{f(a)\}, \{f^2(a)\}, \{f^3(a)\}, \{f^4(a)\}, \{f^5(a)\}\}$$

where, for example, $f^3(a)$ abbreviates $f(f(f(a)))$.

According to the literal $f^3(a) = a$, merge

$$\{f^3(a)\} \text{ and } \{a\}.$$

From the union,

$$2. \{\{a, f^3(a)\}, \{f(a)\}, \{f^2(a)\}, \{f^4(a)\}, \{f^5(a)\}\}$$

deduce the following congruence propagations:

$$f^3(a) \sim a \Rightarrow f(f^3(a)) \sim f(a) \text{ i.e. } f^4(a) \sim f(a)$$

and

$$f^4(a) \sim f(a) \Rightarrow f(f^4(a)) \sim f(f(a)) \text{ i.e. } f^5(a) \sim f^2(a)$$

Thus, the final partition for this iteration is the following:

$$3. \{\{a, f^3(a)\}, \{f(a), f^4(a)\}, \{f^2(a), f^5(a)\}\}.$$

Congruence Closure Algorithm: Example 2 II

$$3. \{ \{a, f^3(a)\}, \{f(a), f^4(a)\}, \{f^2(a), f^5(a)\} \} .$$

From the second literal, $f^5(a) = a$, merge

$$\{f^2(a), f^5(a)\} \quad \text{and} \quad \{a, f^3(a)\}$$

to form the partition

$$4. \{ \{a, f^2(a), f^3(a), f^5(a)\}, \{f(a), f^4(a)\} \} .$$

Propagating the congruence

$$f^3(a) \sim f^2(a) \Rightarrow f(f^3(a)) \sim f(f^2(a)) \text{ i.e. } f^4(a) \sim f^3(a)$$

yields the partition

$$5. \{ \{a, f(a), f^2(a), f^3(a), f^4(a), f^5(a)\} \} ,$$

which represents the congruence closure in which all of S_F are equal. Now,

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

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

Congruence Closure Algorithm: Example 3

Given Σ_E -formula

$$F : f(x) = f(y) \wedge x \neq y .$$

The subterm set S_F induces the following initial partition:

1. $\{\{x\}, \{y\}, \{f(x)\}, \{f(y)\}\}$.

Then $f(x) = f(y)$ indicates to merge

$$\{f(x)\} \quad \text{and} \quad \{f(y)\} .$$

The union $\{f(x), f(y)\}$ does not yield any new congruences, so the final partition is

2. $\{\{x\}, \{y\}, \{f(x), f(y)\}\}$.

Does

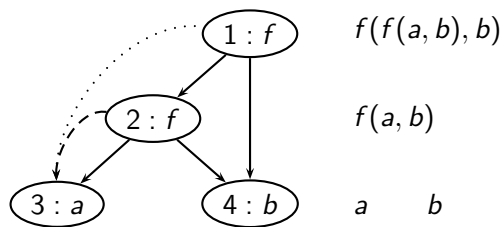
$$\{\{x\}, \{y\}, \{f(x), f(y)\}\} \models F ?$$

Yes, as $x \not\sim y$, agreeing with $x \neq y$. Hence, F is T_E -satisfiable.

Implementation of Algorithm

Directed Acyclic Graph (DAG)

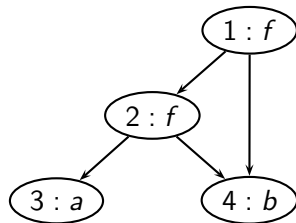
For Σ_E -formula F , graph-based data structure for representing the subterms of S_F (and congruence relation between them).



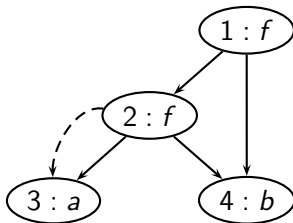
Efficient way for computing the congruence closure.

Summary of idea

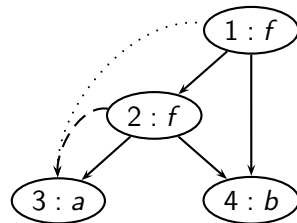
$$f(a, b) = a \wedge f(f(a, b), b) \neq a$$



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

--- explicit equation by congruence

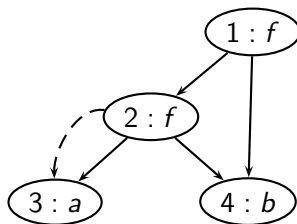
$$\left. \begin{array}{l} \text{FIND } f(f(a, b), b) = a \\ \text{FIND } f(f(a, b), b) \neq a \end{array} \right\} \Rightarrow \text{Unsatisfiable}$$

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

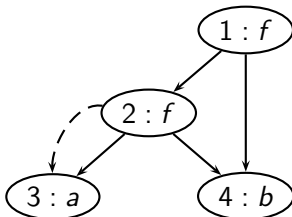
DAG Representation of node 2

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



DAG Representation of node 3

```
type node = {  
    id          : id      ... 3  
    fn          : string  ... a  
    args        : id list ... []  
    mutable find : id      ... 3  
    mutable ccpair : id set ... {1,2}  
}
```

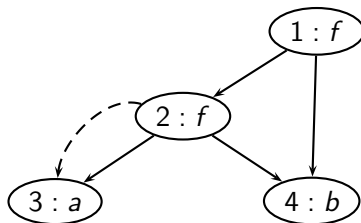


The Implementation I

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 = 3
FIND 3 = 3

3 is the representative of {2, 3}.

The Implementation II

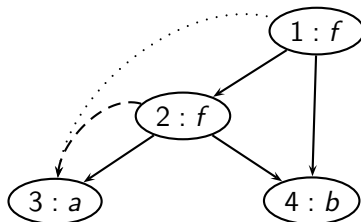
UNION function

```
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

The Implementation III

Example



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

1.find $\leftarrow 3$

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

1.ccpair $\leftarrow \emptyset$

The Implementation IV

CCPAR function

Returns parents of all nodes in i 's congruence class

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

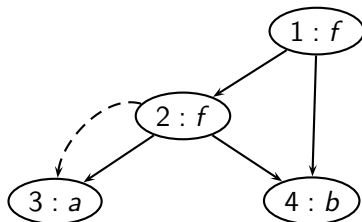
CONGRUENT predicate

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[ $i$ ]
```

The Implementation V

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.

The Implementation VI

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  $t_1 \in P_{i_1}, t_2 \in P_{i_2}$  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 values of CCPAR i_1 and CCPAR i_2 (before the union).

Decision Procedure: T_E -satisfiability

Given Σ_E -formula

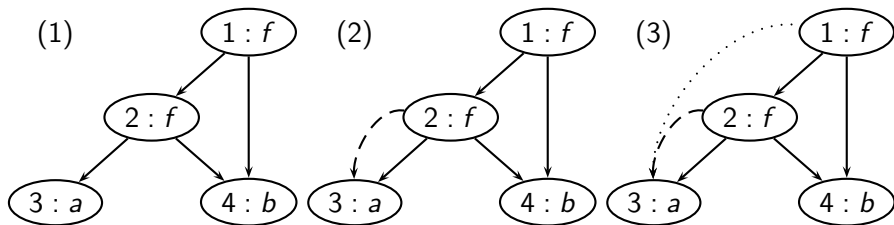
$$F : s_1 = t_1 \wedge \cdots \wedge s_m = t_m \wedge s_{m+1} \neq t_{m+1} \wedge \cdots \wedge s_n \neq t_n ,$$

with subterm set S_F , perform the following steps:

1. Construct the initial DAG for the subterm set S_F .
2. For $i \in \{1, \dots, m\}$, 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 1: T_E -Satisfiability

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



$$P_2 = \{1\}$$

$$P_3 = \{2\}$$

UNION 2 3

$$P_1 = \{\}$$

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

UNION 1 2

CONGRUENT 1 2

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

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

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

Final partition (Fig (3))

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

Note: dash edge _ _ _ _ merge dictated by equalities in F
dotted edge deduced merge

Does

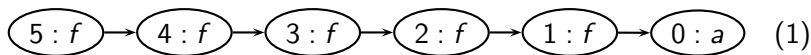
$$\{\{a, f(a, b), f(f(a, b), b)\}, \{b\}\} \models F ?$$

No, as $f(f(a, b), b) \sim a$, but F asserts that $f(f(a, b), b) \neq a$.

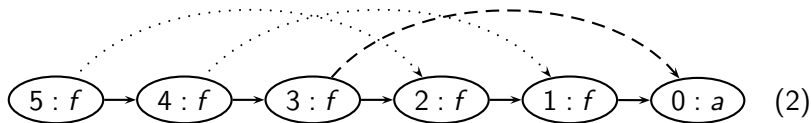
Hence, F is T_E -unsatisfiable.

Example 2: T_E -Satisfiability

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



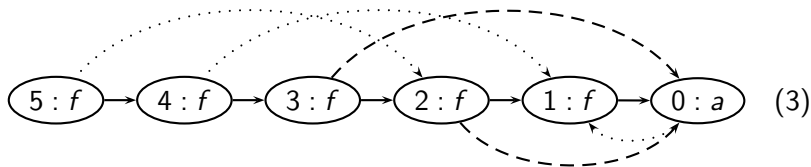
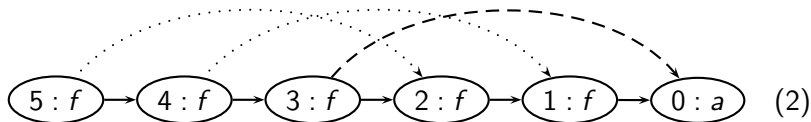
Initial DAG



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

Example 2: T_E -Satisfiability

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



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

$$\text{UNION } 5 \ 0$$

$$\Rightarrow \text{MERGE } 3 \ 1 : \text{ STOP. Why?}$$

$$\text{UNION } 3 \ 1$$

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

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.

Recursive Data Structures

Quantifier-free Theory of Lists T_{cons}

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

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

Axioms of T_{cons}

- ▶ reflexivity, symmetry, transitivity
- ▶ function (congruence) axioms:

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

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

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

- ▶ predicate (congruence) axiom:

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



$$(A1) \forall x, y. \text{car}(\text{cons}(x, y)) = x \quad (\text{left projection})$$

$$(A2) \forall x, y. \text{cdr}(\text{cons}(x, y)) = y \quad (\text{right projection})$$

$$(A3) \forall x. \neg \text{atom}(x) \rightarrow \text{cons}(\text{car}(x), \text{cdr}(x)) = x \quad (\text{construction})$$

$$(A4) \forall x, y. \neg \text{atom}(\text{cons}(x, y)) \quad (\text{atom})$$

Simplifications

- ▶ Consider only quantifier-free conjunctive Σ_{cons} -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 of a conjunctive Σ_{cons} -formula with literals

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

- ▶ Because of similarity to Σ_E , we sometimes combine $\Sigma_{\text{cons}} \cup \Sigma_E$.

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

$$F : \quad \underbrace{s_1 = t_1 \wedge \cdots \wedge s_m = t_m}_{\text{generate congruence closure}}$$

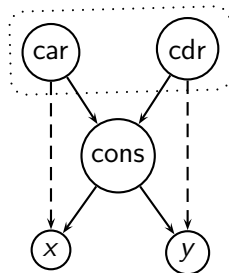
$$\wedge \quad \underbrace{s_{m+1} \neq t_{m+1} \wedge \cdots \wedge s_n \neq t_n}_{\text{search for contradiction}}$$

$$\wedge \quad \underbrace{\text{atom}(u_1) \wedge \cdots \wedge \text{atom}(u_\ell)}_{\text{search for contradiction}}$$

where s_i , t_i , and u_i are T_{cons} -terms

Algorithm: T_{cons} -Satisfiability

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 (A1), (A2)
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**



Example

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

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

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

$$F' : \quad x = \text{cons}(u_1, v_1) \quad \wedge \quad (3)$$

$$y = \text{cons}(u_2, v_2) \quad \wedge \quad (4)$$

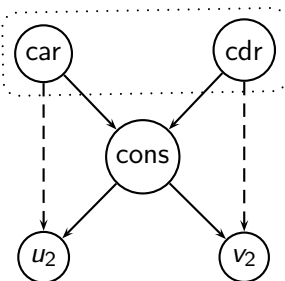
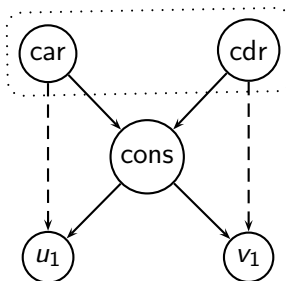
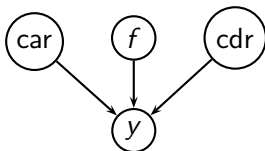
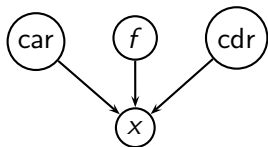
$$f(x) \neq f(y) \quad (5)$$

Recall the projection axioms:

$$(A1) \quad \forall x, y. \text{car}(\text{cons}(x, y)) = x$$

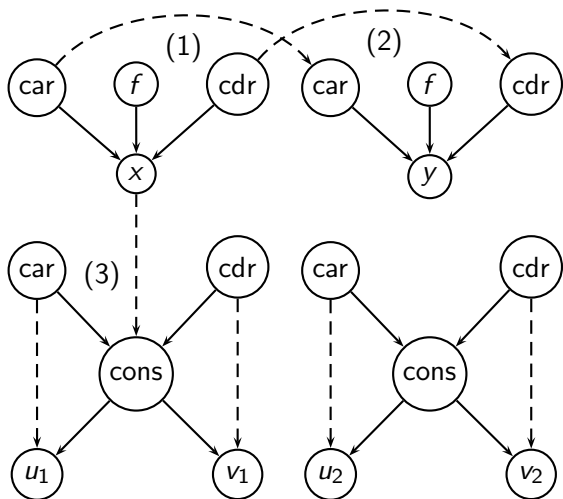
$$(A2) \quad \forall x, y. \text{cdr}(\text{cons}(x, y)) = y$$

Example (cont): Initial DAG



axioms (A1), (A2)

Example (cont): MERGE



-- explicit equation

... by congruence

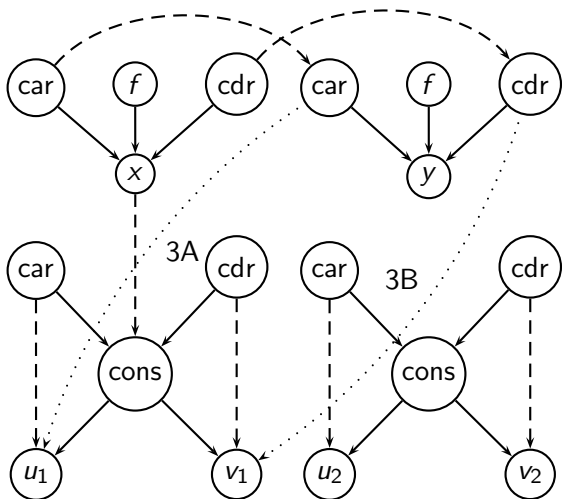
1 : MERGE `car(x)` `car(y)`

2 : MERGE `cdr(x)` `cdr(y)`

3 : MERGE `x` `cons(u1, v1)`

⇓

Example (cont): Propagation



Congruent:

$\text{car}(x) \text{ car}(\text{cons}(u_1, v_1))$

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

FIND $\text{car}(\text{cons}(\dots)) = u_1$

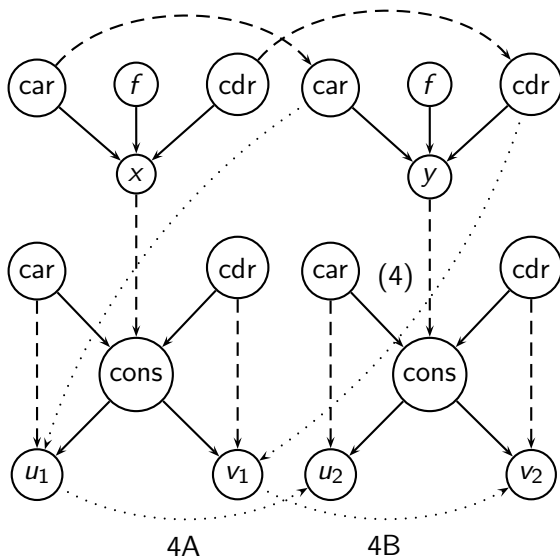
Congruent:

$\text{cdr}(x) \text{ cdr}(\text{cons}(u_1, v_1))$

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

FIND $\text{cdr}(\text{cons}(\dots)) = v_1$

Example (cont): MERGE



4 : MERGE y cons(u_2, v_2)



Congruent:

car(y) car(cons(u_2, v_2))

FIND car(y) = u_1

FIND car(cons(...)) = u_2

Congruent:

cdr(y) cdr(cons(u_2, v_2))

FIND cdr(y) = v_1

FIND cdr(cons(...)) = v_2



Example (cont): CONGRUENCE

