

Computer-Aided Reasoning for Software

CSSE507

A Survey of Theory Solvers

Emina Torlak

emina@cs.washington.edu

Today

Last lecture

- Introduction to Satisfiability Modulo Theories (SMT)

Today

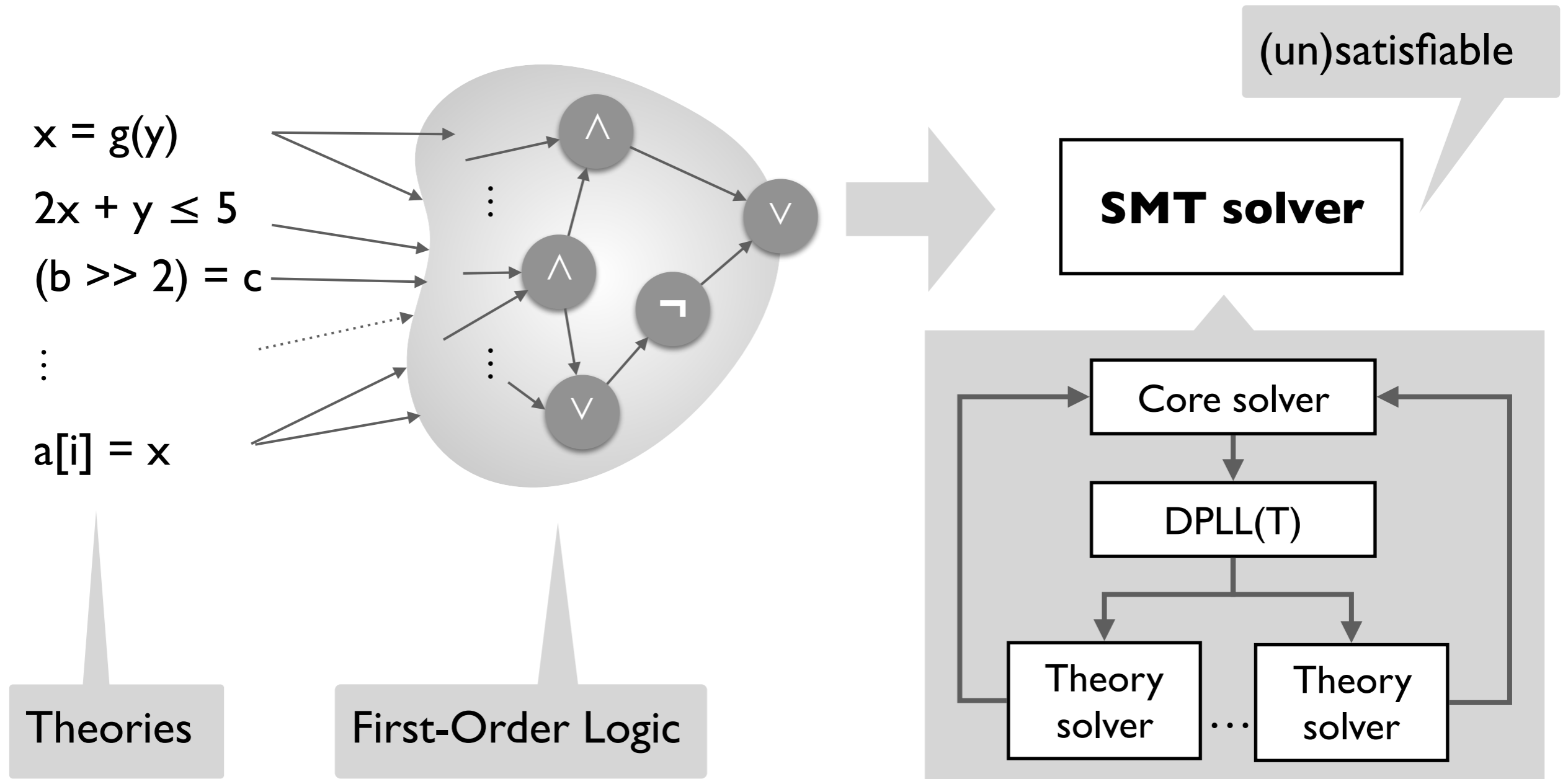
- A quick survey of theory solvers
- An in-depth look at the core theory solver (theory of equality and uninterpreted functions)

Reminder

- Project proposal due in a week.
- Find a partner and start brainstorming if you haven't already!



Recall: Satisfiability Modulo Theories (SMT)



A brief survey of common theory solvers

$$x = g(y)$$

Core solver

$$2x + y \leq 5$$

Theory
solver

$$2i + j \leq 5$$

Theory
solver

$$(b \gg 2) = c$$

Theory
solver

$$a[i] = x$$

Theory
solver

A brief survey of common theory solvers

$$x = g(y)$$

Equality and UF

$$2x + y \leq 5$$

Linear Real
Arithmetic

$$2i + j \leq 5$$

Linear Integer
Arithmetic

$$(b \gg 2) = c$$

Bitvectors

$$a[i] = x$$

Arrays

A brief survey of common theory solvers

$$x = g(y)$$

Equality and UF

$$2x + y \leq 5$$

Linear Real
Arithmetic

$$2i + j \leq 5$$

Linear Integer
Arithmetic

$$(b \gg 2) = c$$

Bitvectors

$$a[i] = x$$

Arrays

- **Conjunctions** of linear constraints over \mathbb{R}
 - Can be decided in polynomial time, but in practice solved with the **General Simplex** method (worst case exponential)
 - Can also be decided with **Fourier-Motzkin** elimination (exponential)

A brief survey of common theory solvers

$$x = g(y)$$

Equality and UF

$$2x + y \leq 5$$

Linear Real
Arithmetic

$$2i + j \leq 5$$

Linear Integer
Arithmetic

$$(b \gg 2) = c$$

Bitvectors

$$a[i] = x$$

Arrays

- **Conjunctions** of linear constraints over \mathbb{Z}
- **Branch-and-cut** (based on Simplex)
- **Omega Test** (extension of Fourier-Motzkin)
- **Small-Domain Encoding** used for arbitrary combinations of linear constraints over \mathbb{Z}
- NP-complete

A brief survey of common theory solvers

$$x = g(y)$$

Equality and UF

$$2x + y \leq 5$$

Linear Real
Arithmetic

$$2i + j \leq 5$$

Linear Integer
Arithmetic

$$(b \gg 2) = c$$

Bitvectors

$$a[i] = x$$

Arrays

- **Arbitrary combination** of constraints over bitvectors
- **Bit blasting** (reduction to SAT)
- NP-complete

A brief survey of common theory solvers

$$x = g(y)$$

Equality and UF

$$2x + y \leq 5$$

Linear Real
Arithmetic

$$2i + j \leq 5$$

Linear Integer
Arithmetic

$$(b \gg 2) = c$$

Bitvectors

$$a[i] = x$$

Arrays

- **Conjunctions** of constraints over read/write terms in the theory of arrays
- Reduce to $T =$ satisfiability
- NP-complete (because the reduction introduces disjunctions)

A brief survey of common theory solvers

$$x = g(y)$$

Equality and UF

- **Conjunctions** of equality constraints over uninterpreted functions
- **Congruence closure**
- Polynomial time

$$2x + y \leq 5$$

Linear Real
Arithmetic

$$2i + j \leq 5$$

Linear Integer
Arithmetic

$$(b \gg 2) = c$$

Bitvectors

$$a[i] = x$$

Arrays

Theory of equality and UF ($T_=$)

Signature (all symbols)

- $\{=, a, b, c, \dots, f, g, \dots, p, q, \dots\}$

Axioms

- reflexivity: $\forall x. x = x$
- symmetry: $\forall x, y. x = y \rightarrow y = x$
- transitivity: $\forall x, y, z. x = y \wedge y = z \rightarrow x = z$
- congruence: $\forall x_1, \dots, x_n, y_1, \dots, y_n. (\bigwedge_{1 \leq i \leq n} x_i = y_i) \rightarrow f(x_1, \dots, x_n) = f(y_1, \dots, y_n)$
- congruence: $\forall x_1, \dots, x_n, y_1, \dots, y_n. (\bigwedge_{1 \leq i \leq n} x_i = y_i) \rightarrow p(x_1, \dots, x_n) \leftrightarrow p(y_1, \dots, y_n)$

Theory of equality and UF ($T=$)

Signature (all symbols)

- $\{=, a, b, c, \dots, f, g, \dots, p, q, \dots\}$

Axioms

- reflexivity: $\forall x. x = x$
- symmetry: $\forall x, y. x = y \rightarrow y = x$
- transitivity: $\forall x, y, z. x = y \wedge y = z \rightarrow x = z$
- congruence: $\forall x_1, \dots, x_n, y_1, \dots, y_n. (\bigwedge_{1 \leq i \leq n} x_i = y_i) \rightarrow f(x_1, \dots, x_n) = f(y_1, \dots, y_n)$

- ~~X~~ congruence: $\forall x_1, \dots, x_n, y_1, \dots, y_n. (\bigwedge_{1 \leq i \leq n} x_i = y_i) \rightarrow p(x_1, \dots, x_n) \leftrightarrow p(y_1, \dots, y_n)$

Replace predicates with equality constraints over functions:

- introduce a fresh constant T
- for each predicate p , introduce a fresh function f_p
- $p(x_1, \dots, x_n) \rightsquigarrow f_p(x_1, \dots, x_n) = T$

Is a conjunction of $T=$ literals satisfiable?

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

Is a conjunction of $T=$ literals satisfiable?

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

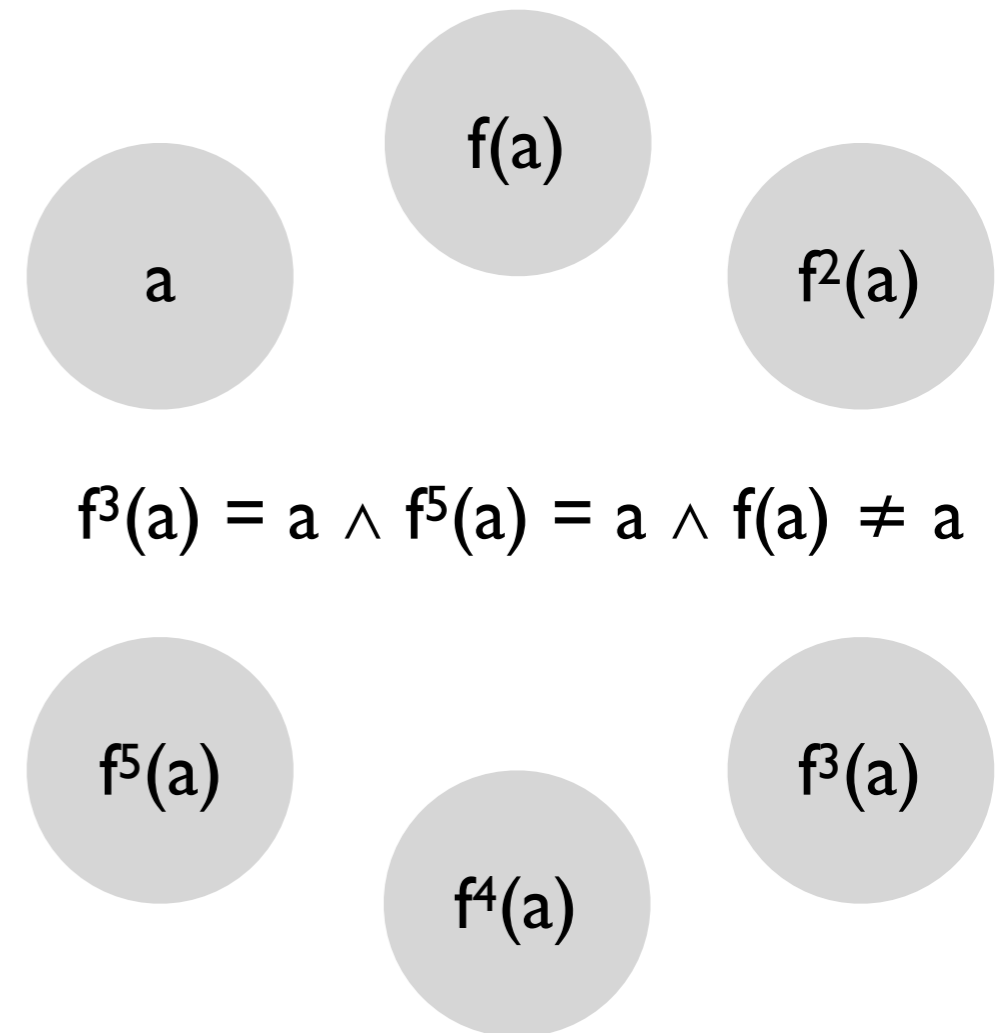
Congruence closure algorithm: example



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

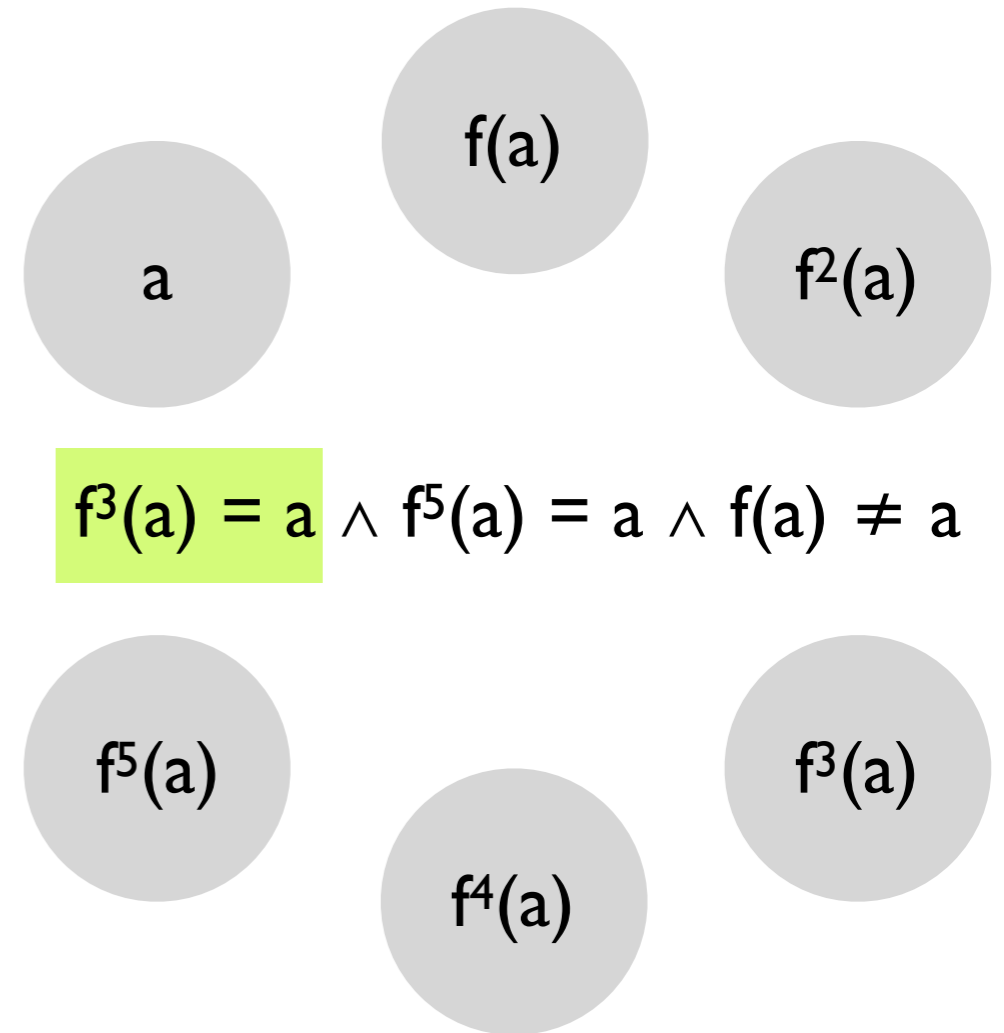
Congruence closure algorithm: example

- Place each subterm of F into its own **congruence class**



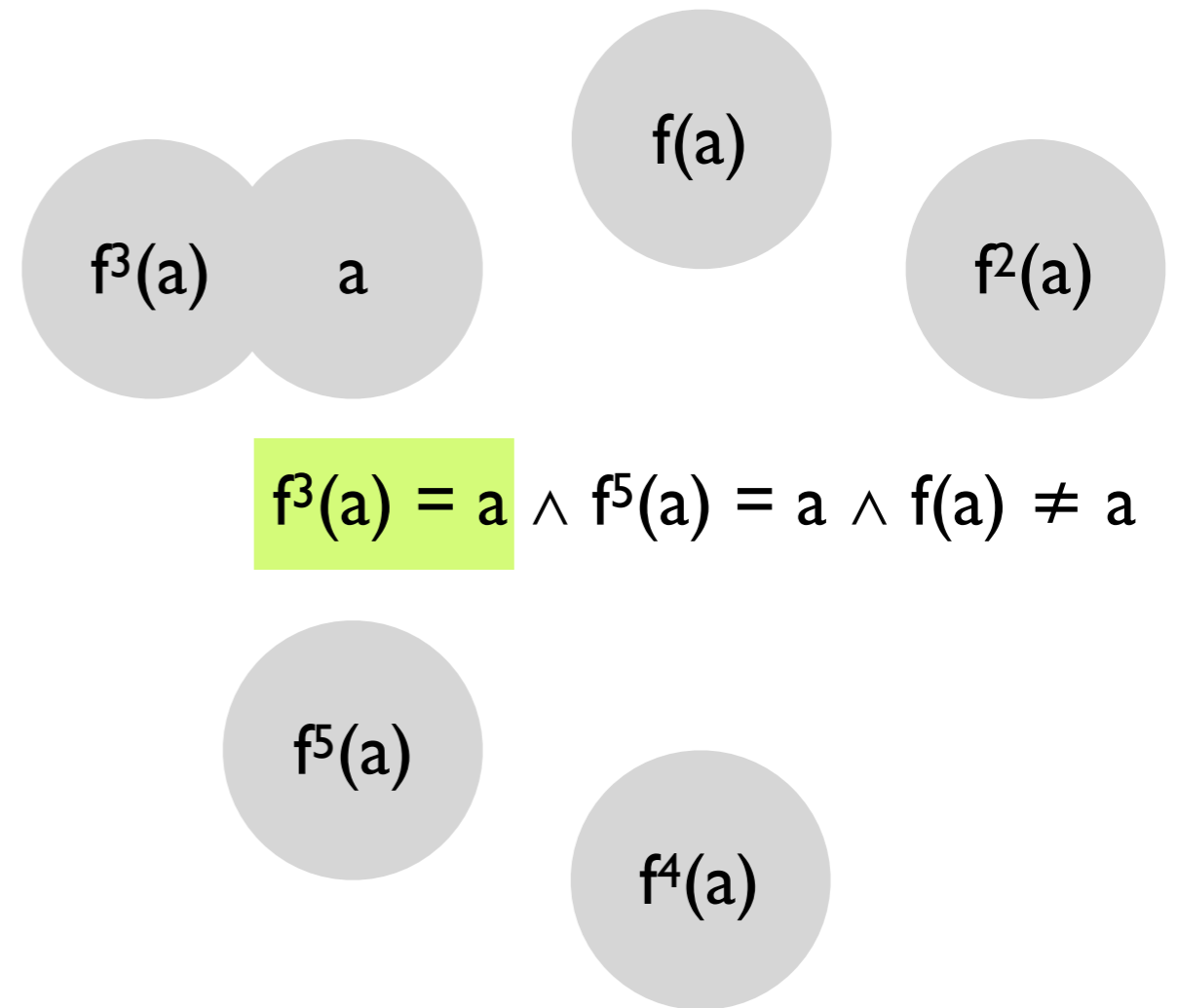
Congruence closure algorithm: example

- Place each subterm of F into its own **congruence class**
- For each positive literal $t_1 = t_2$ in F
 - Merge the classes for t_1 and t_2



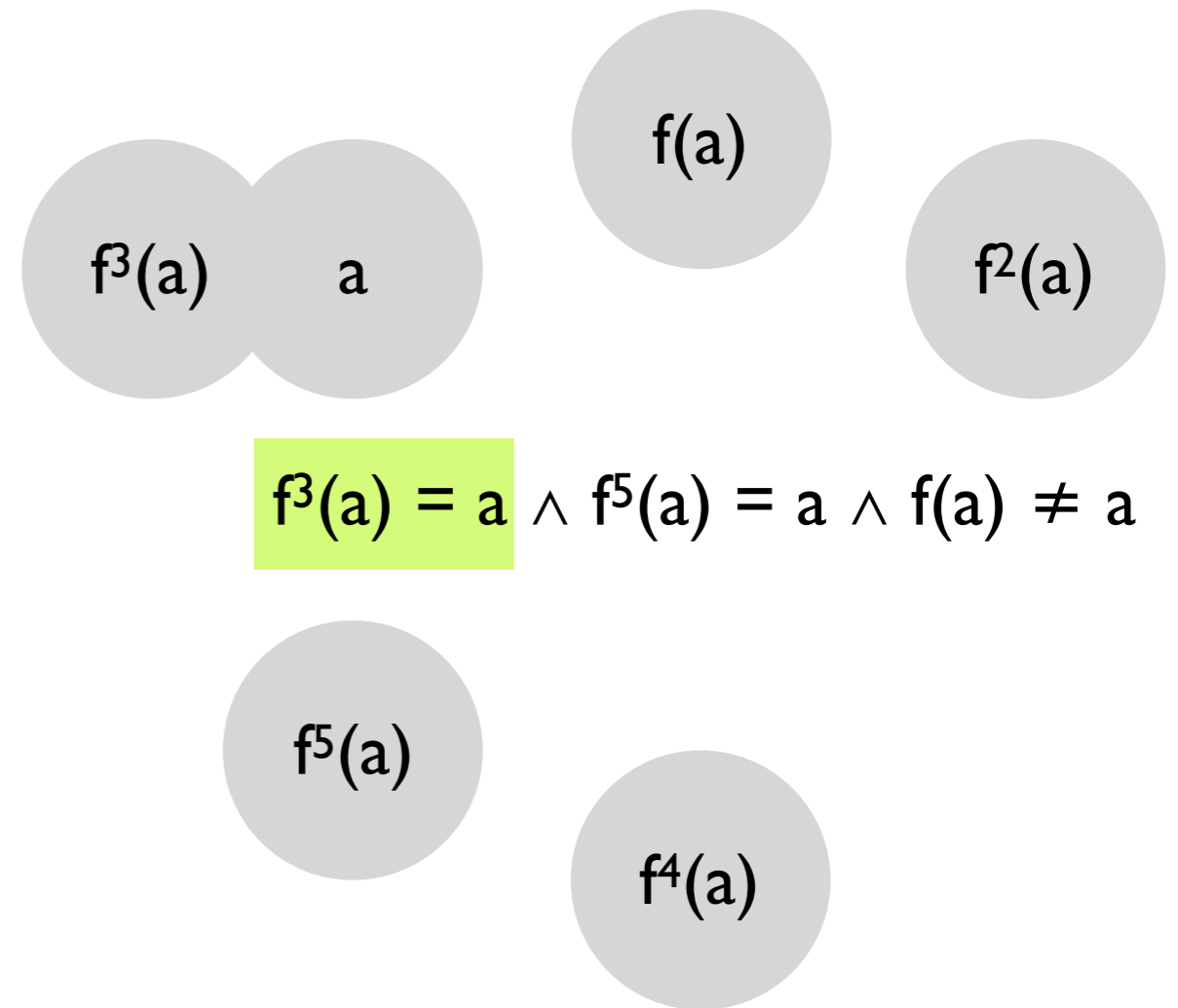
Congruence closure algorithm: example

- Place each subterm of F into its own **congruence class**
- For each positive literal $t_1 = t_2$ in F
 - Merge the classes for t_1 and t_2



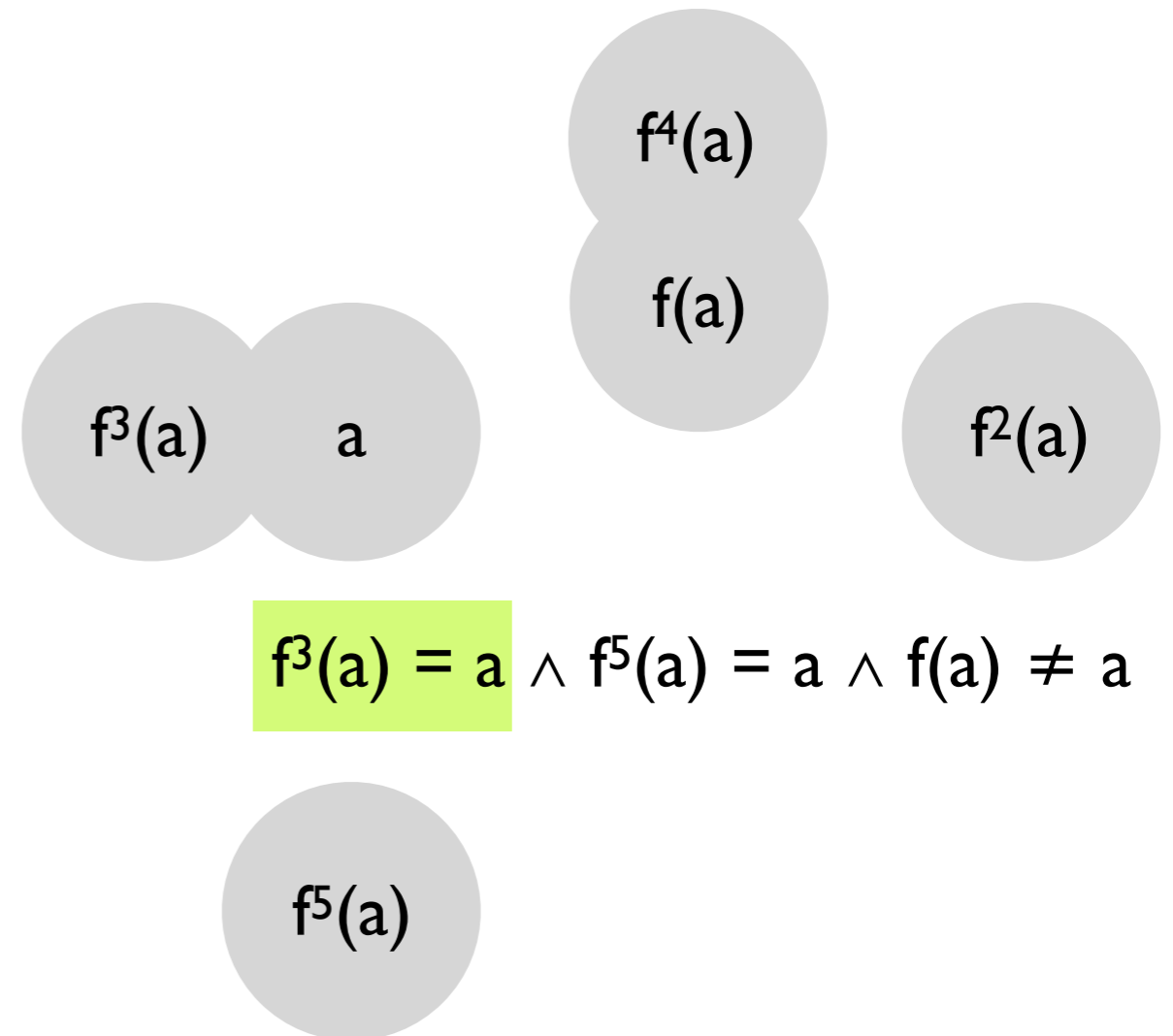
Congruence closure algorithm: example

- Place each subterm of F into its own **congruence class**
- For each positive literal $t_1 = t_2$ in F
 - Merge the classes for t_1 and t_2
 - Propagate the resulting congruences



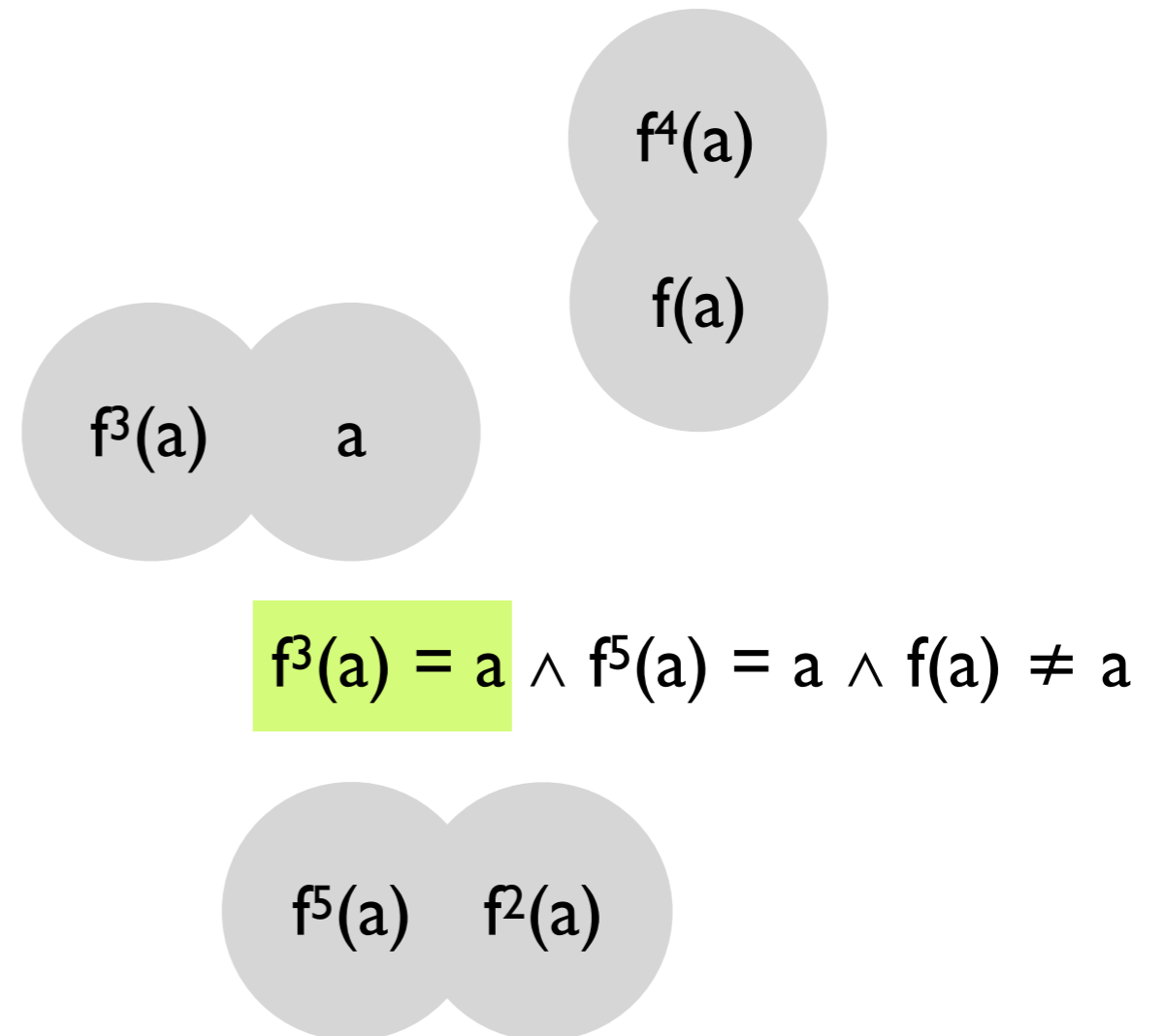
Congruence closure algorithm: example

- Place each subterm of F into its own **congruence class**
- For each positive literal $t_1 = t_2$ in F
 - Merge the classes for t_1 and t_2
 - Propagate the resulting congruences



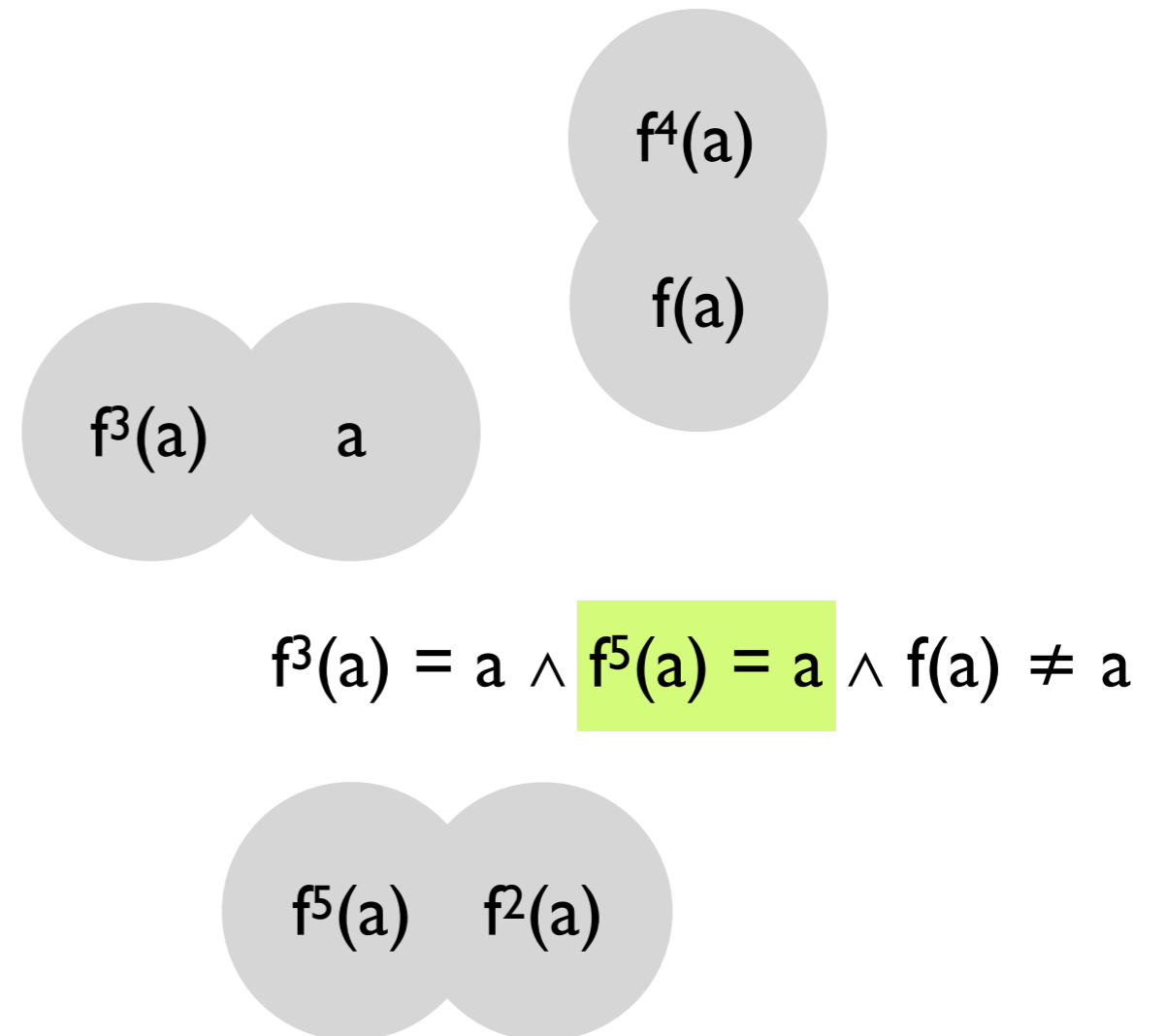
Congruence closure algorithm: example

- Place each subterm of F into its own **congruence class**
- For each positive literal $t_1 = t_2$ in F
 - Merge the classes for t_1 and t_2
 - Propagate the resulting congruences



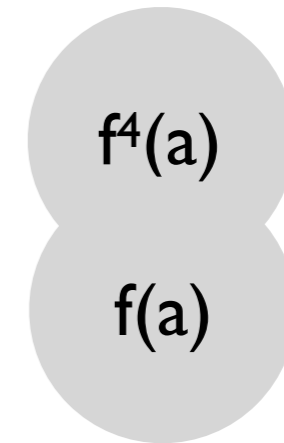
Congruence closure algorithm: example

- Place each subterm of F into its own **congruence class**
- For each positive literal $t_1 = t_2$ in F
 - Merge the classes for t_1 and t_2
 - Propagate the resulting congruences

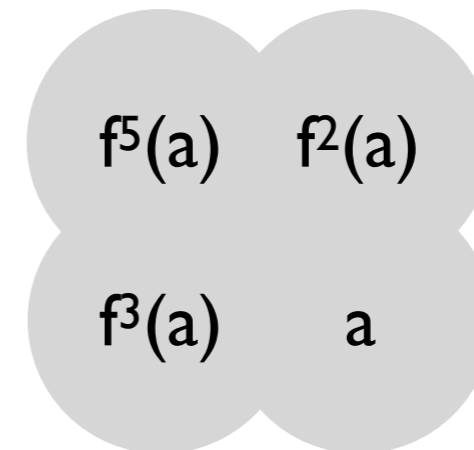


Congruence closure algorithm: example

- Place each subterm of F into its own **congruence class**
- For each positive literal $t_1 = t_2$ in F
 - Merge the classes for t_1 and t_2
 - Propagate the resulting congruences



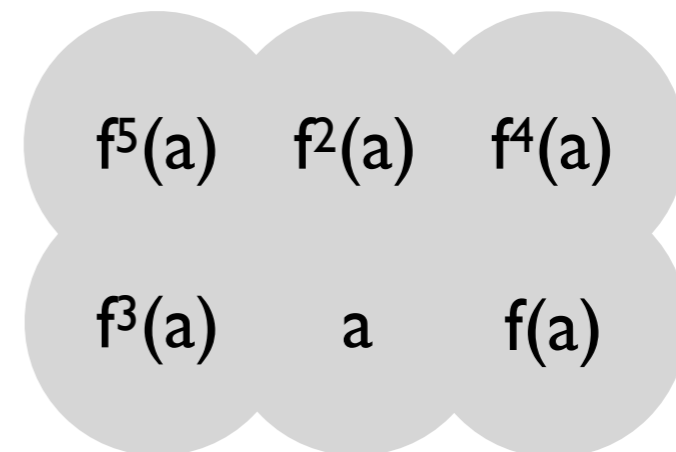
$$f^3(a) = a \wedge f^5(a) = a \wedge f(a) \neq a$$



Congruence closure algorithm: example

- Place each subterm of F into its own **congruence class**
- For each positive literal $t_1 = t_2$ in F
 - Merge the classes for t_1 and t_2
 - Propagate the resulting congruences

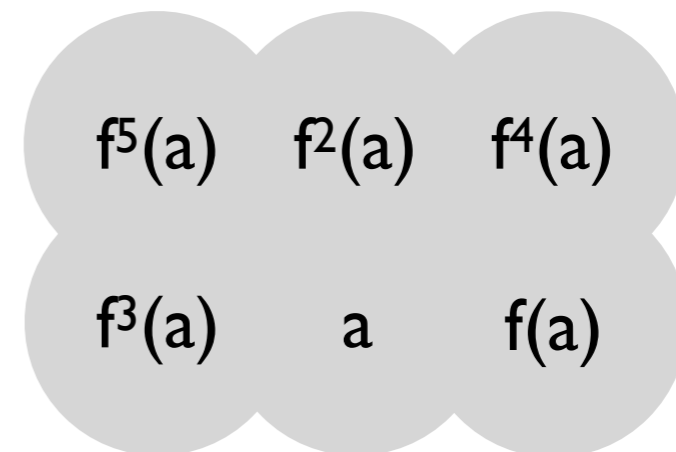
$$f^3(a) = a \wedge f^5(a) = a \wedge f(a) \neq a$$



Congruence closure algorithm: example

- Place each subterm of F into its own **congruence class**
- For each positive literal $t_1 = t_2$ in F
 - Merge the classes for t_1 and t_2
 - Propagate the resulting congruences
- If F has a negative literal $t_1 \neq t_2$ with both terms in the same congruence class, output UNSAT
- Otherwise, output SAT

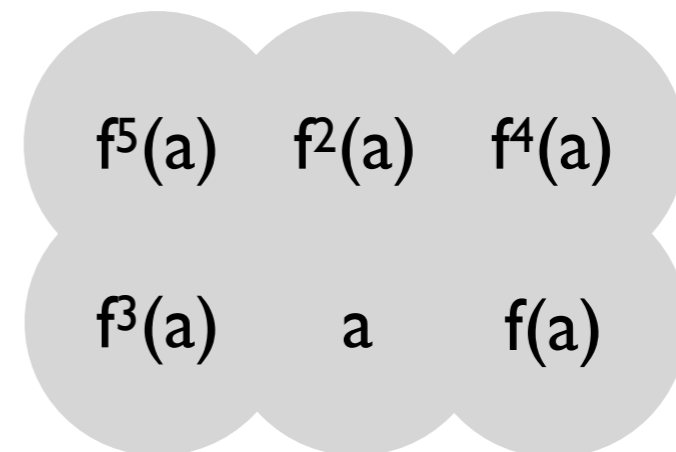
$$f^3(a) = a \wedge f^5(a) = a \wedge f(a) \neq a$$



Congruence closure algorithm: example

- Place each subterm of F into its own **congruence class**
- For each positive literal $t_1 = t_2$ in F
 - Merge the classes for t_1 and t_2
 - Propagate the resulting congruences
- If F has a negative literal $t_1 \neq t_2$ with both terms in the same congruence class, output UNSAT
- Otherwise, output SAT

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



UNSAT

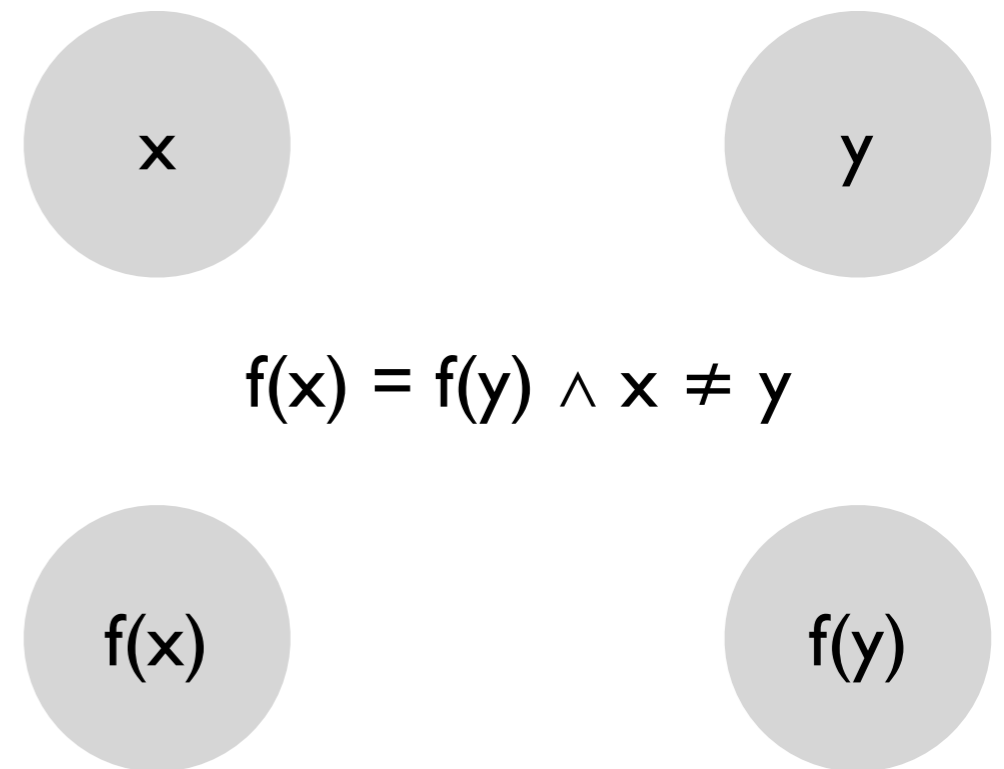
Congruence closure algorithm: another example

- Place each subterm of F into its own **congruence class**
- For each positive literal $t_1 = t_2$ in F
 - Merge the classes for t_1 and t_2
 - Propagate the resulting congruences
- If F has a negative literal $t_1 \neq t_2$ with both terms in the same congruence class, output UNSAT
- Otherwise, output SAT

$$f(x) = f(y) \wedge x \neq y$$

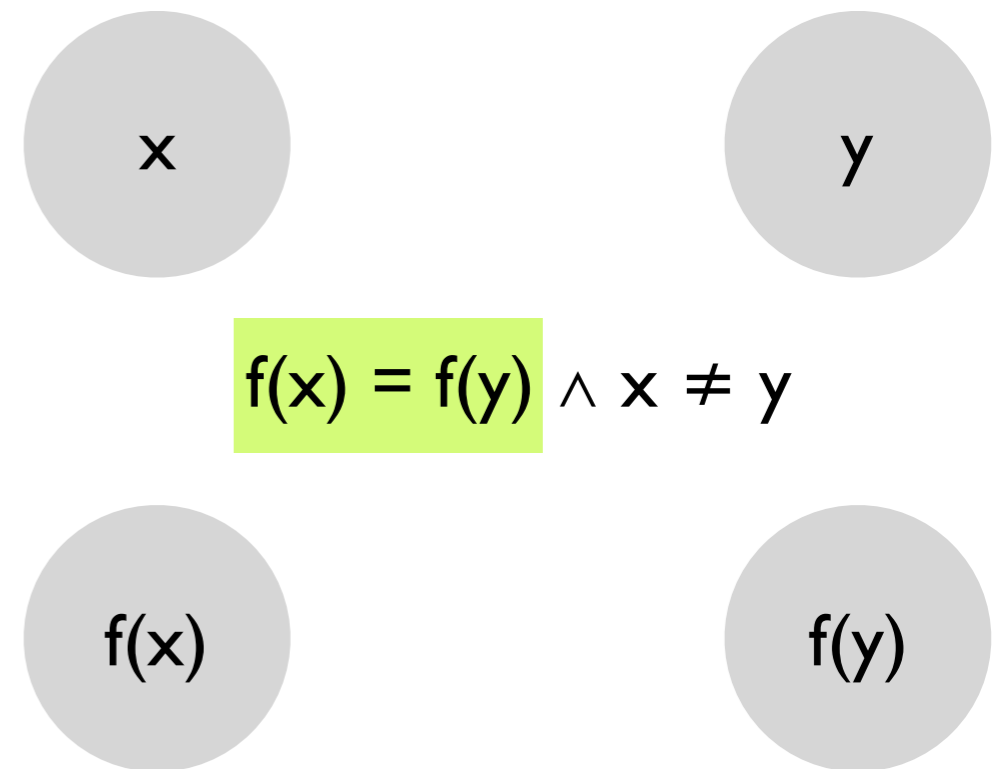
Congruence closure algorithm: another example

- Place each subterm of F into its own **congruence class**
- For each positive literal $t_1 = t_2$ in F
 - Merge the classes for t_1 and t_2
 - Propagate the resulting congruences
- If F has a negative literal $t_1 \neq t_2$ with both terms in the same congruence class, output UNSAT
- Otherwise, output SAT



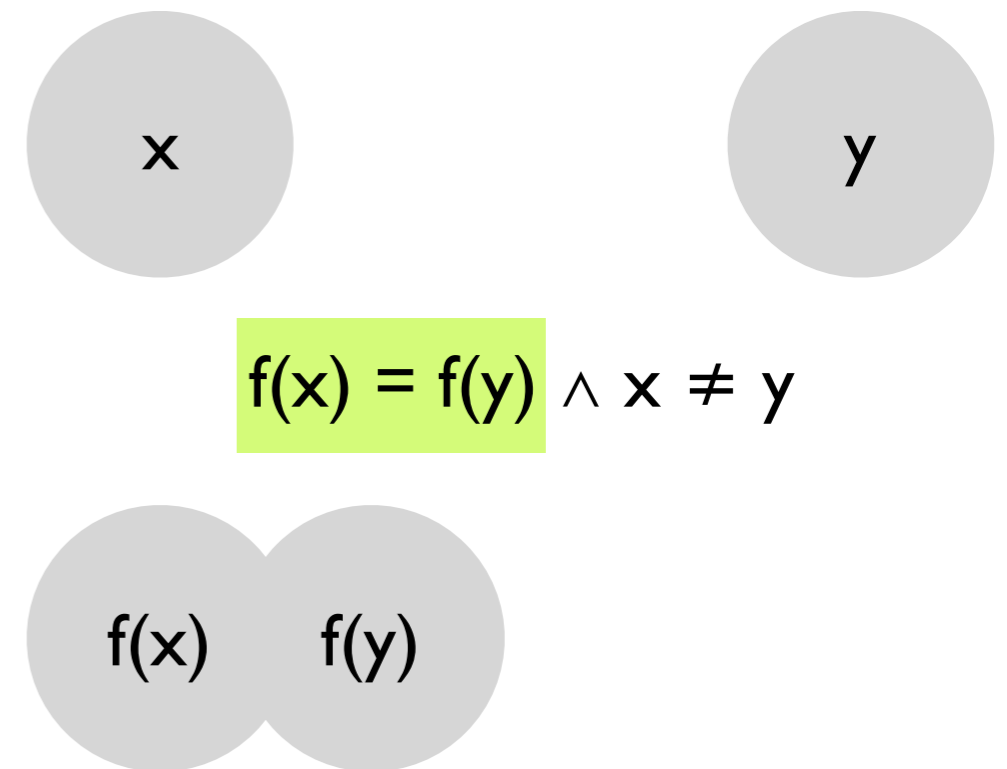
Congruence closure algorithm: another example

- Place each subterm of F into its own **congruence class**
- For each positive literal $t_1 = t_2$ in F
 - Merge the classes for t_1 and t_2
 - Propagate the resulting congruences
- If F has a negative literal $t_1 \neq t_2$ with both terms in the same congruence class, output UNSAT
- Otherwise, output SAT



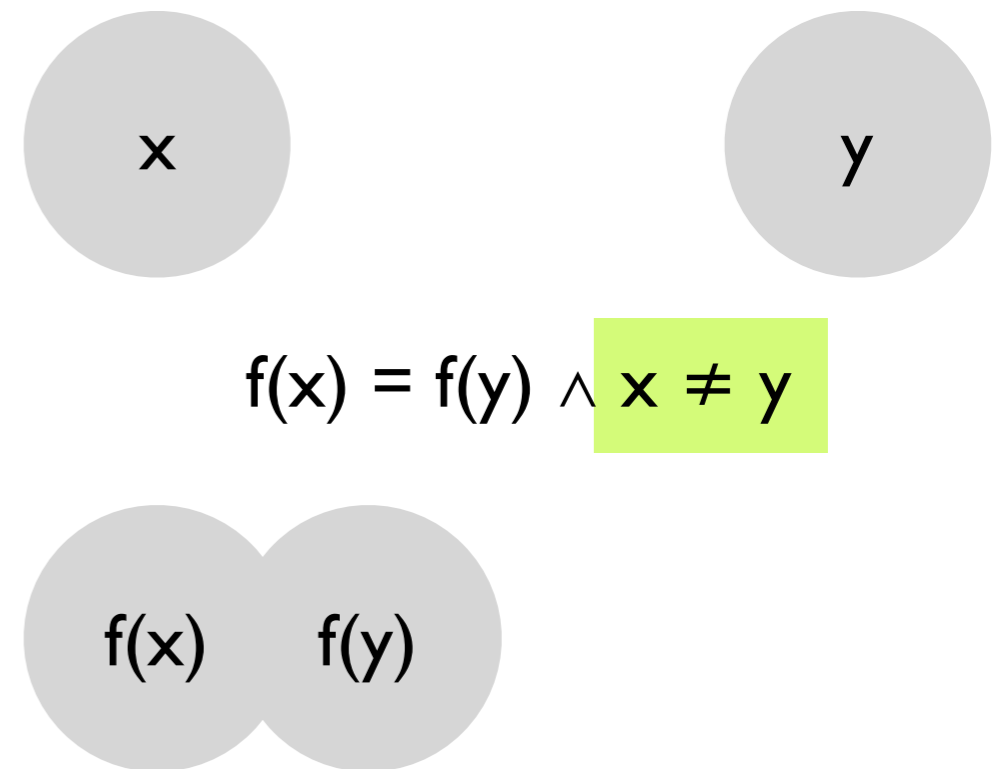
Congruence closure algorithm: another example

- Place each subterm of F into its own **congruence class**
- For each positive literal $t_1 = t_2$ in F
 - Merge the classes for t_1 and t_2
 - Propagate the resulting congruences
- If F has a negative literal $t_1 \neq t_2$ with both terms in the same congruence class, output UNSAT
- Otherwise, output SAT



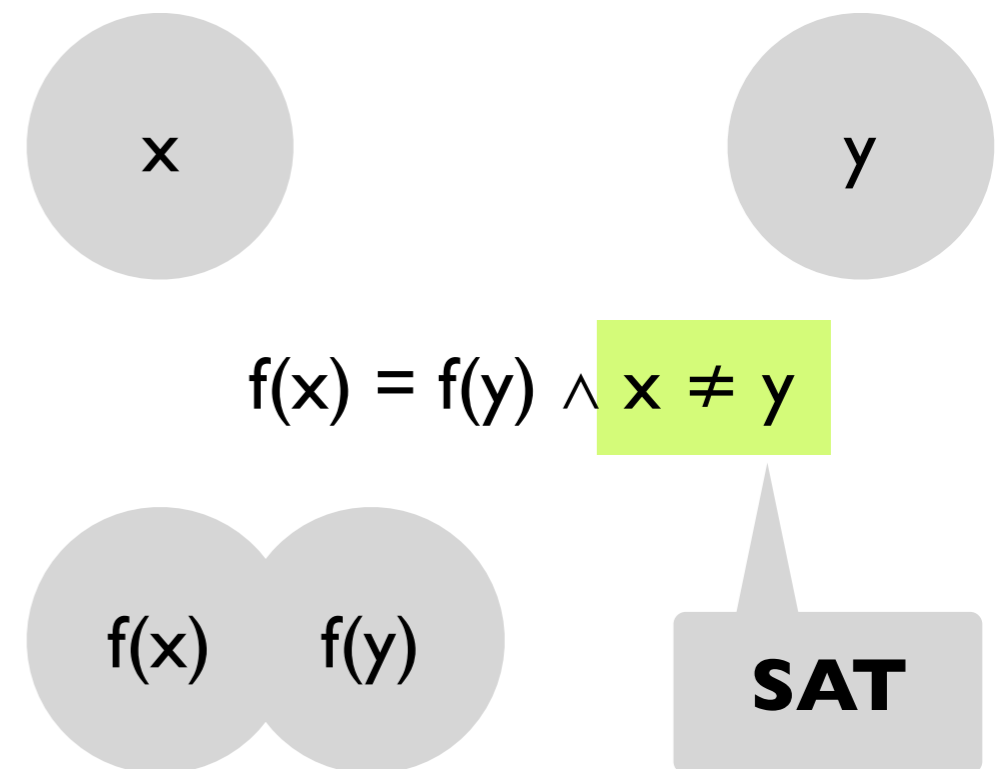
Congruence closure algorithm: another example

- Place each subterm of F into its own **congruence class**
- For each positive literal $t_1 = t_2$ in F
 - Merge the classes for t_1 and t_2
 - Propagate the resulting congruences
- If F has a negative literal $t_1 \neq t_2$ with both terms in the same congruence class, output UNSAT
- Otherwise, output SAT



Congruence closure algorithm: another example

- Place each subterm of F into its own **congruence class**
- For each positive literal $t_1 = t_2$ in F
 - Merge the classes for t_1 and t_2
 - Propagate the resulting congruences
- If F has a negative literal $t_1 \neq t_2$ with both terms in the same congruence class, output UNSAT
- Otherwise, output SAT



Congruence closure algorithm: definitions

A binary relation R is an **equivalence relation** if it is reflexive, symmetric, and transitive.

Congruence closure algorithm: definitions

A binary relation R is an **equivalence relation** if it is reflexive, symmetric, and transitive.

An equivalence relation R is a **congruence relation** if for every n -ary function f

$$\forall \bar{x}, \bar{y}. \bigwedge R(x_i, y_i) \rightarrow R(f(\bar{x}), f(\bar{y}))$$

Congruence closure algorithm: definitions

A binary relation R is an **equivalence relation** if it is reflexive, symmetric, and transitive.

An equivalence relation R is a **congruence relation** if for every n -ary function f

$$\forall \bar{x}, \bar{y}. \bigwedge R(x_i, y_i) \rightarrow R(f(\bar{x}), f(\bar{y}))$$

The **equivalence class** of an element $s \in S$ under an equivalence relation R :

$$\{ s' \in S \mid R(s, s') \}$$

Congruence closure algorithm: definitions

A binary relation R is an **equivalence relation** if it is reflexive, symmetric, and transitive.

An equivalence relation R is a **congruence relation** if for every n -ary function f

$$\forall \bar{x}, \bar{y}. \bigwedge R(x_i, y_i) \rightarrow R(f(\bar{x}), f(\bar{y}))$$

The **equivalence class** of an element $s \in S$ under an equivalence relation R :

$$\{ s' \in S \mid R(s, s') \}$$

What is the equivalence class of 9 under \equiv_3 ?

Congruence closure algorithm: definitions

A binary relation R is an **equivalence relation** if it is reflexive, symmetric, and transitive.

An equivalence relation R is a **congruence relation** if for every n -ary function f

$$\forall \bar{x}, \bar{y}. \bigwedge R(x_i, y_i) \rightarrow R(f(\bar{x}), f(\bar{y}))$$

The **equivalence class** of an element $s \in S$ under an equivalence relation R :

$$\{ s' \in S \mid R(s, s') \}$$

An equivalence class is called a **congruence class** if R is a congruence relation.

Congruence closure algorithm: definitions

The **equivalence closure** R^E of a binary relation R is the smallest equivalence relation that contains R .

What is the equivalence closure of $R = \{\langle a, b \rangle, \langle b, c \rangle, \langle d, d \rangle\}$?

Congruence closure algorithm: definitions

The **equivalence closure** R^E of a binary relation R is the smallest equivalence relation that contains R .

What is the equivalence closure of $R = \{\langle a, b \rangle, \langle b, c \rangle, \langle d, d \rangle\}$?

$R^E = \{\langle a, a \rangle, \langle b, b \rangle, \langle c, c \rangle, \langle d, d \rangle$
 $\langle a, b \rangle, \langle b, a \rangle, \langle b, c \rangle, \langle c, b \rangle,$
 $\langle a, c \rangle, \langle c, a \rangle\}$

Congruence closure algorithm: definitions

The **equivalence closure** R^E of a binary relation R is the smallest equivalence relation that contains R .

The **congruence closure** R^C of a binary relation R is the smallest congruence relation that contains R .

The congruence closure algorithm computes the congruence closure of the equality relation over terms asserted by a conjunctive quantifier-free formula in $T=$.

Congruence closure algorithm: data structure

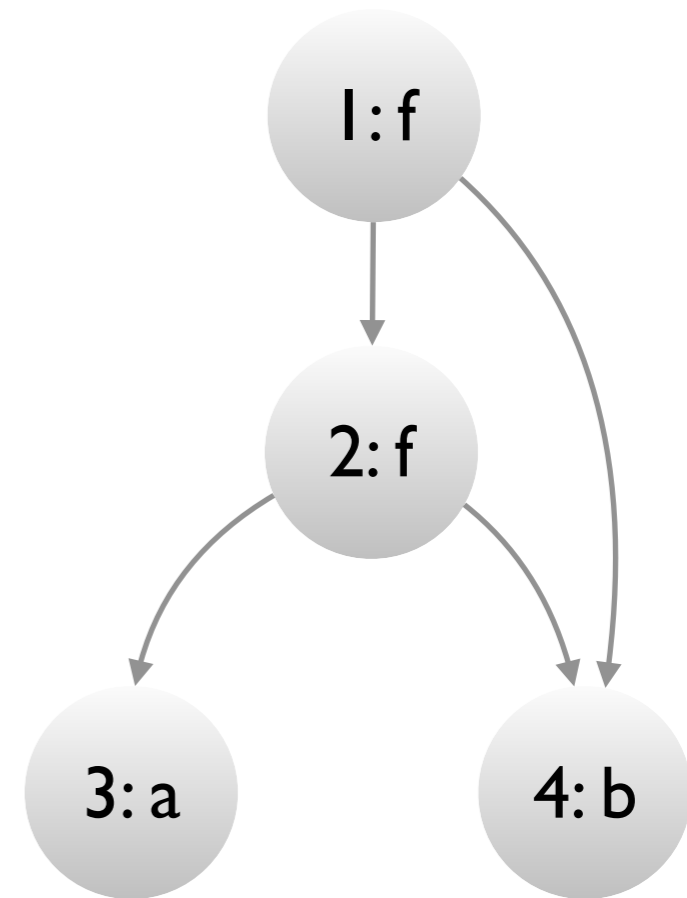


$$f(a, b) = a \wedge f(f(a, b), b) \neq a$$

Congruence closure algorithm: data structure

- Represent subterms with a DAG

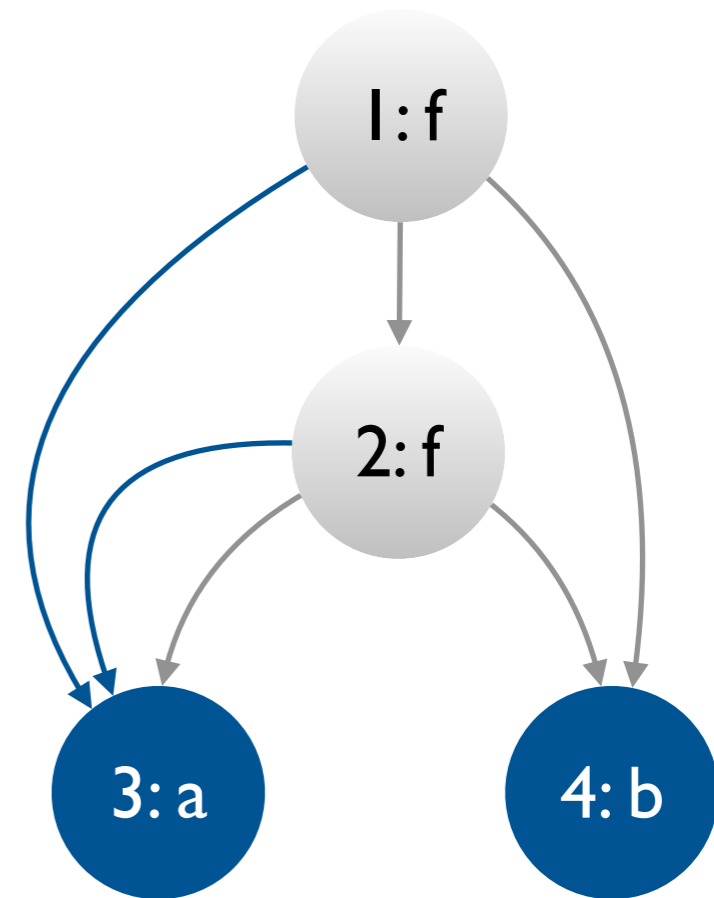
$$f(a, b) = a \wedge f(f(a, b), b) \neq a$$



Congruence closure algorithm: data structure

- Represent subterms with a DAG
- Each node has a **find** pointer to another node in its congruence class (or to itself if it is the **representative**)

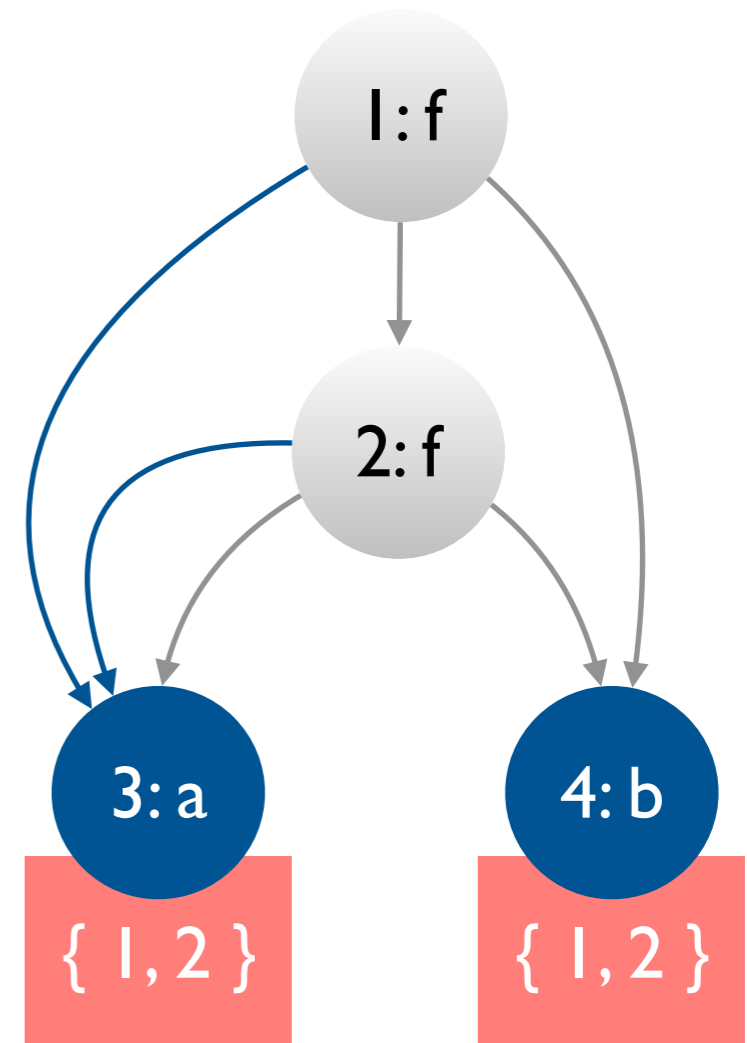
$$f(a, b) = a \wedge f(f(a, b), b) \neq a$$



Congruence closure algorithm: data structure

- Represent subterms with a DAG
- Each node has a **find** pointer to another node in its congruence class (or to itself if it is the **representative**)
- Each representative has a **ccp** field that stores all parents of all nodes in its congruence class.

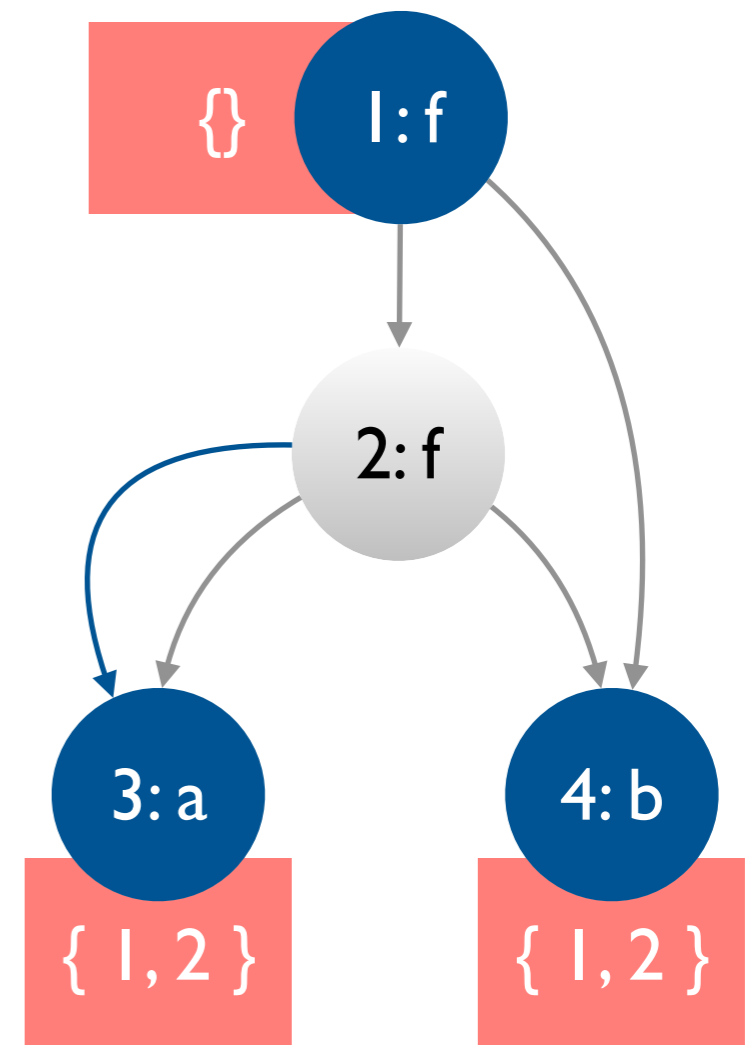
$$f(a, b) = a \wedge f(f(a, b), b) \neq a$$



Congruence closure algorithm: union-find



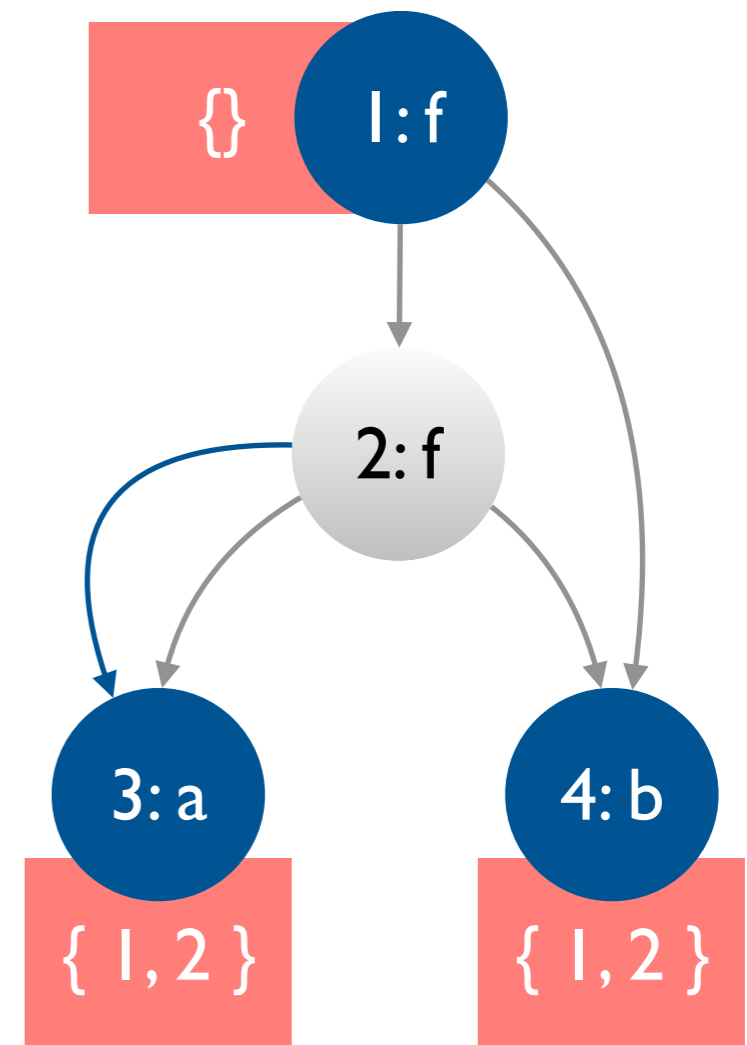
$$f(a, b) = a \wedge f(f(a, b), b) \neq a$$



Congruence closure algorithm: union-find

- FIND returns the representative of a node's equivalence class by following **find** pointers until it finds a self-loop.

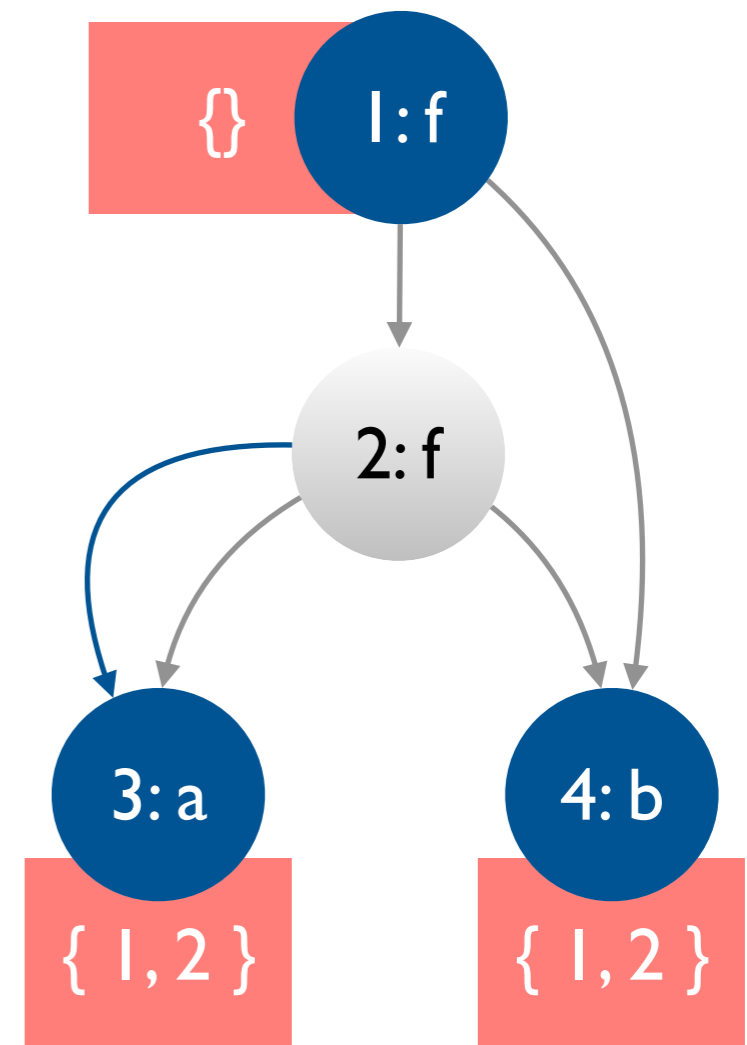
$$f(a, b) = a \wedge f(f(a, b), b) \neq a$$



Congruence closure algorithm: union-find

- FIND returns the representative of a node's equivalence class by following **find** pointers until it finds a self-loop.
- UNION combines equivalence classes for nodes i_1 and i_2 :
 - $n_1, n_2 \leftarrow \text{FIND}(i_1), \text{FIND}(i_2)$
 - $n_1.\text{find} \leftarrow n_2$
 - $n_2.\text{ccp} \leftarrow n_1.\text{ccp} \cup n_2.\text{ccp}$
 - $n_1.\text{ccp} \leftarrow \emptyset$

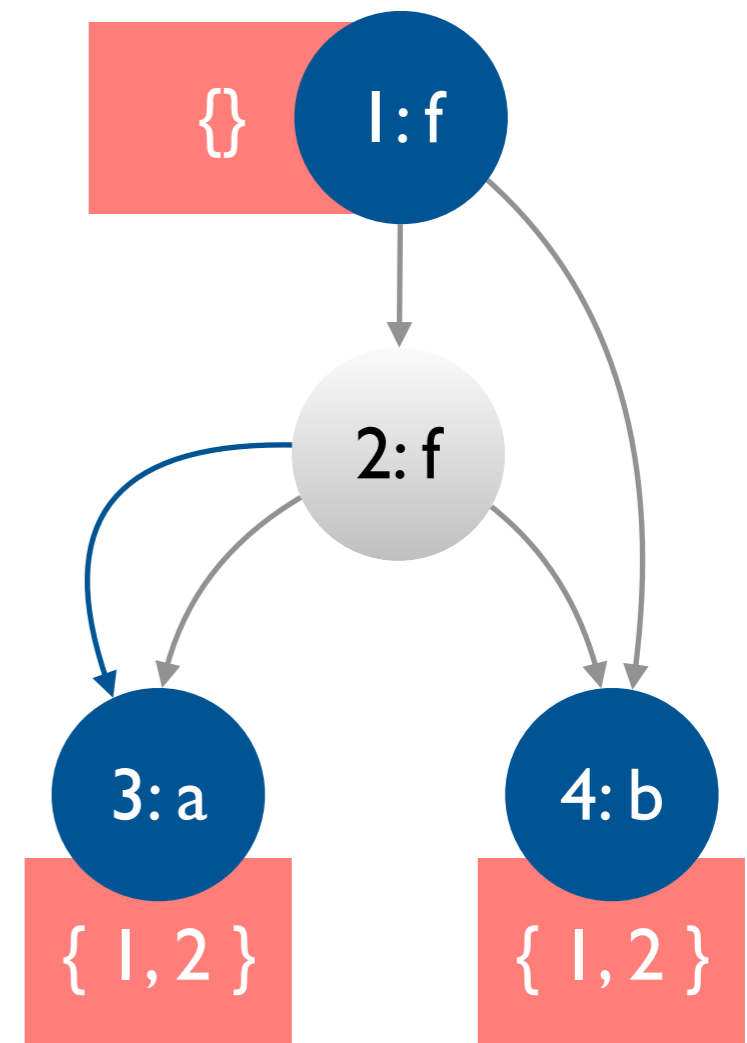
$$f(a, b) = a \wedge f(f(a, b), b) \neq a$$



Congruence closure algorithm: union-find

- FIND returns the representative of a node's equivalence class by following **find** pointers until it finds a self-loop.
- UNION combines equivalence classes for nodes i_1 and i_2 :
 - $n_1, n_2 \leftarrow \text{FIND}(i_1), \text{FIND}(i_2)$
 - $n_1.\text{find} \leftarrow n_2$
 - $n_2.\text{ccp} \leftarrow n_1.\text{ccp} \cup n_2.\text{ccp}$
 - $n_1.\text{ccp} \leftarrow \emptyset$

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

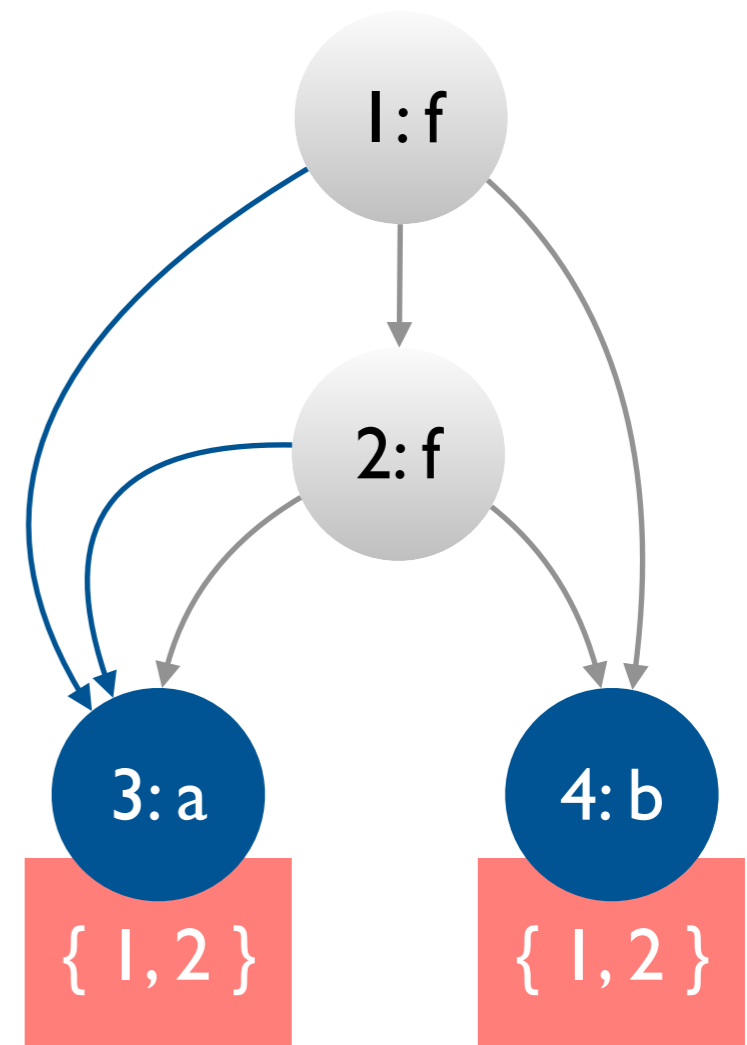


What is UNION(1, 2)?

Congruence closure algorithm: union-find

- FIND returns the representative of a node's equivalence class by following **find** pointers until it finds a self-loop.
- UNION combines equivalence classes for nodes i_1 and i_2 :
 - $n_1, n_2 \leftarrow \text{FIND}(i_1), \text{FIND}(i_2)$
 - $n_1.\text{find} \leftarrow n_2$
 - $n_2.\text{ccp} \leftarrow n_1.\text{ccp} \cup n_2.\text{ccp}$
 - $n_1.\text{ccp} \leftarrow \emptyset$

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

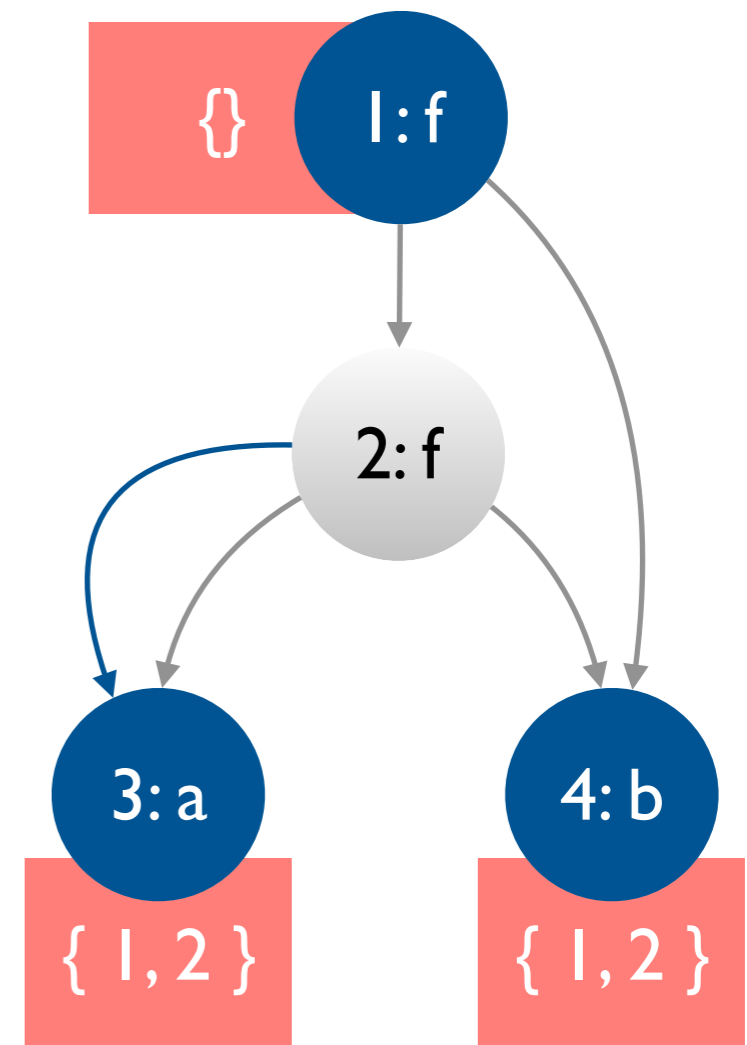


What is UNION(1, 2)?

Congruence closure algorithm: congruent

- CONGRUENT takes as input two nodes and returns true iff their
 - functions are the same
 - corresponding arguments are in the same congruence class

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

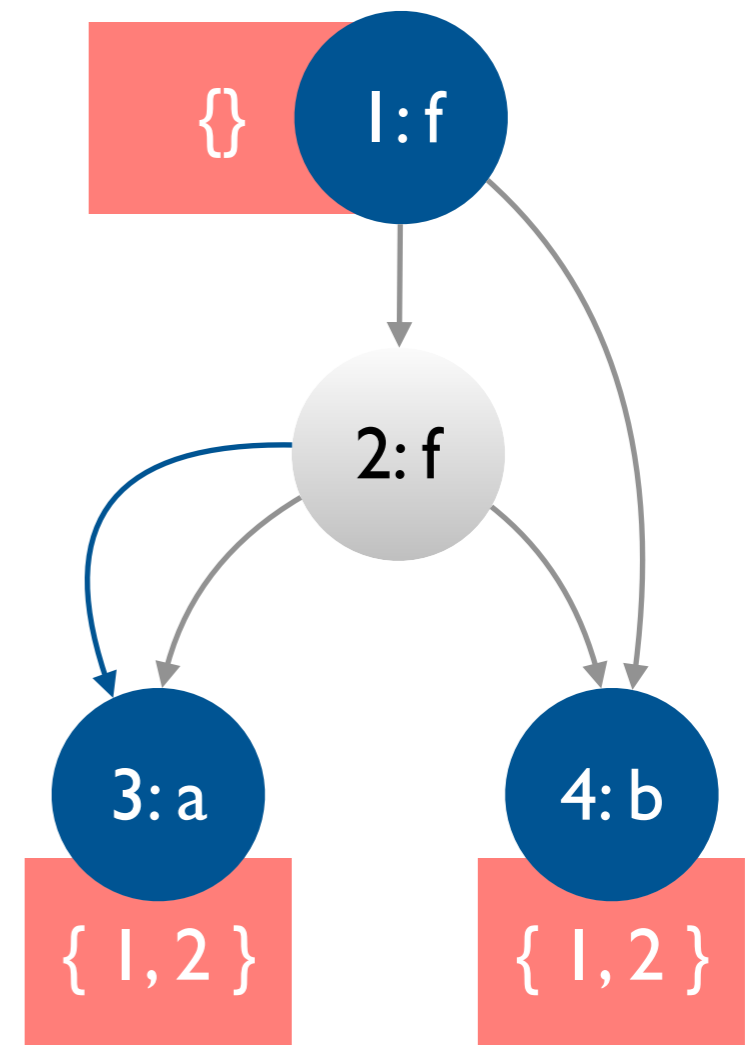


Congruence closure algorithm: congruent

- CONGRUENT takes as input two nodes and returns true iff their
 - functions are the same
 - corresponding arguments are in the same congruence class

CONGRUENT(1, 2)?

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



Congruence closure algorithm: merge

MERGE (i_1, i_2)

$n_1, n_2 \leftarrow \text{FIND}(i_1), \text{FIND}(i_2)$

if $n_1 = n_2$ **then return**

$p_1, p_2 \leftarrow n_1.\text{ccp}, n_2.\text{ccp}$

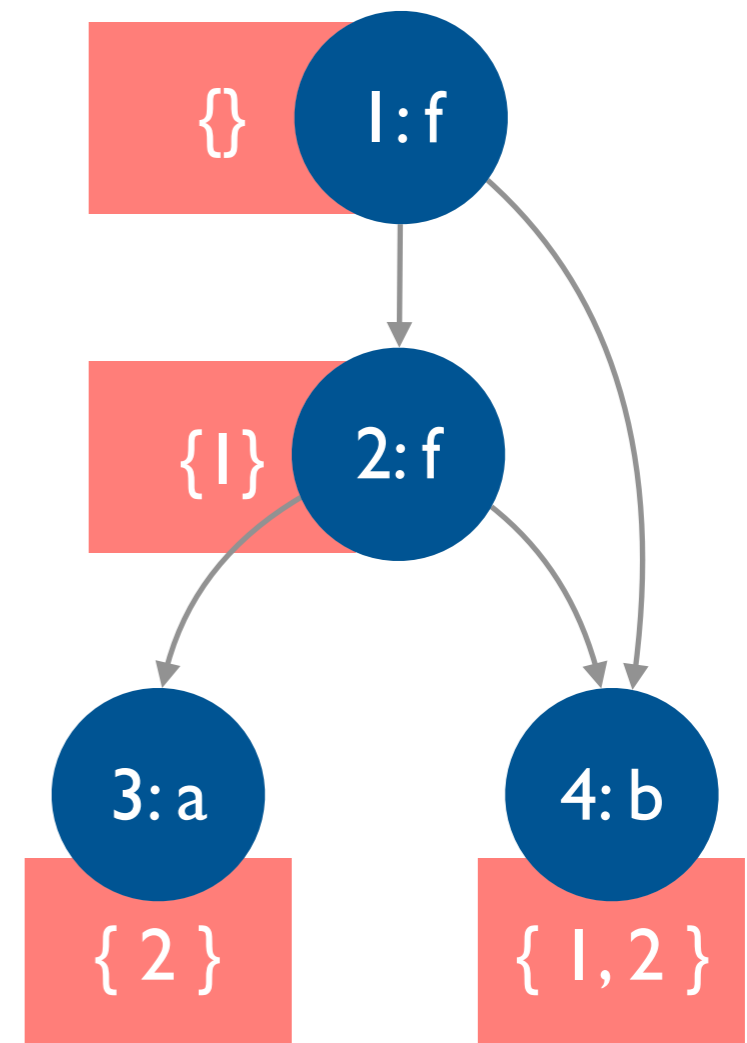
UNION(n_1, n_2)

for each $t_1, t_2 \in p_1 \times p_2$

if $\text{FIND}(t_1) \neq \text{FIND}(t_2) \wedge \text{CONGRUENT}(t_1, t_2)$

then MERGE(t_1, t_2)

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



Congruence closure algorithm: merge

MERGE (i_1, i_2)

$n_1, n_2 \leftarrow \text{FIND}(i_1), \text{FIND}(i_2)$

if $n_1 = n_2$ **then return**

$p_1, p_2 \leftarrow n_1.\text{ccp}, n_2.\text{ccp}$

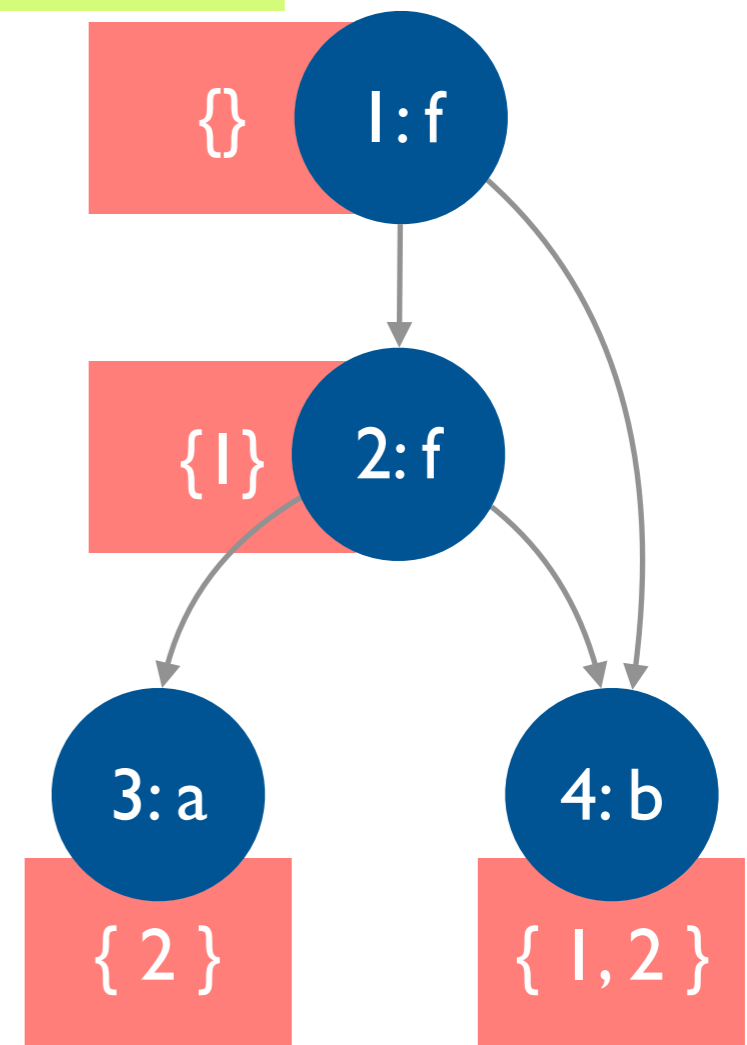
UNION(n_1, n_2)

for each $t_1, t_2 \in p_1 \times p_2$

if $\text{FIND}(t_1) \neq \text{FIND}(t_2) \wedge \text{CONGRUENT}(t_1, t_2)$

then MERGE(t_1, t_2)

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



Congruence closure algorithm: merge

MERGE (i_1, i_2)

$n_1, n_2 \leftarrow \text{FIND}(i_1), \text{FIND}(i_2)$

if $n_1 = n_2$ **then return**

$p_1, p_2 \leftarrow n_1.\text{ccp}, n_2.\text{ccp}$

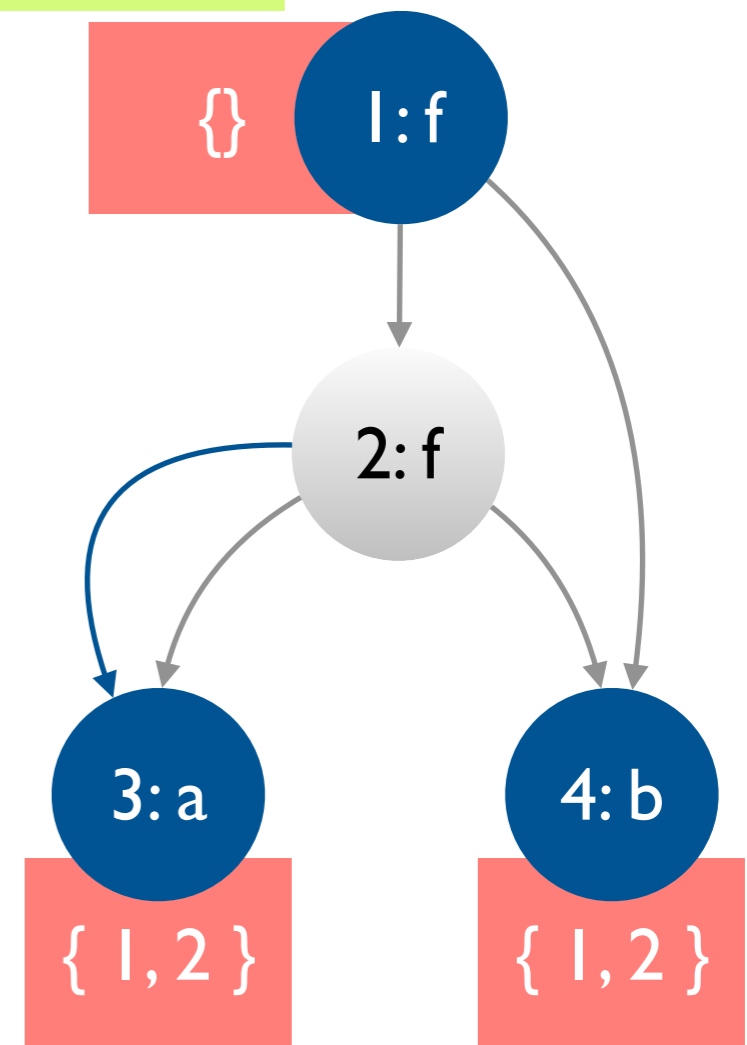
UNION(n_1, n_2)

for each $t_1, t_2 \in p_1 \times p_2$

if $\text{FIND}(t_1) \neq \text{FIND}(t_2) \wedge \text{CONGRUENT}(t_1, t_2)$

then MERGE(t_1, t_2)

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



Congruence closure algorithm: merge

MERGE (i_1, i_2)

$n_1, n_2 \leftarrow \text{FIND}(i_1), \text{FIND}(i_2)$

if $n_1 = n_2$ **then return**

$p_1, p_2 \leftarrow n_1.\text{ccp}, n_2.\text{ccp}$

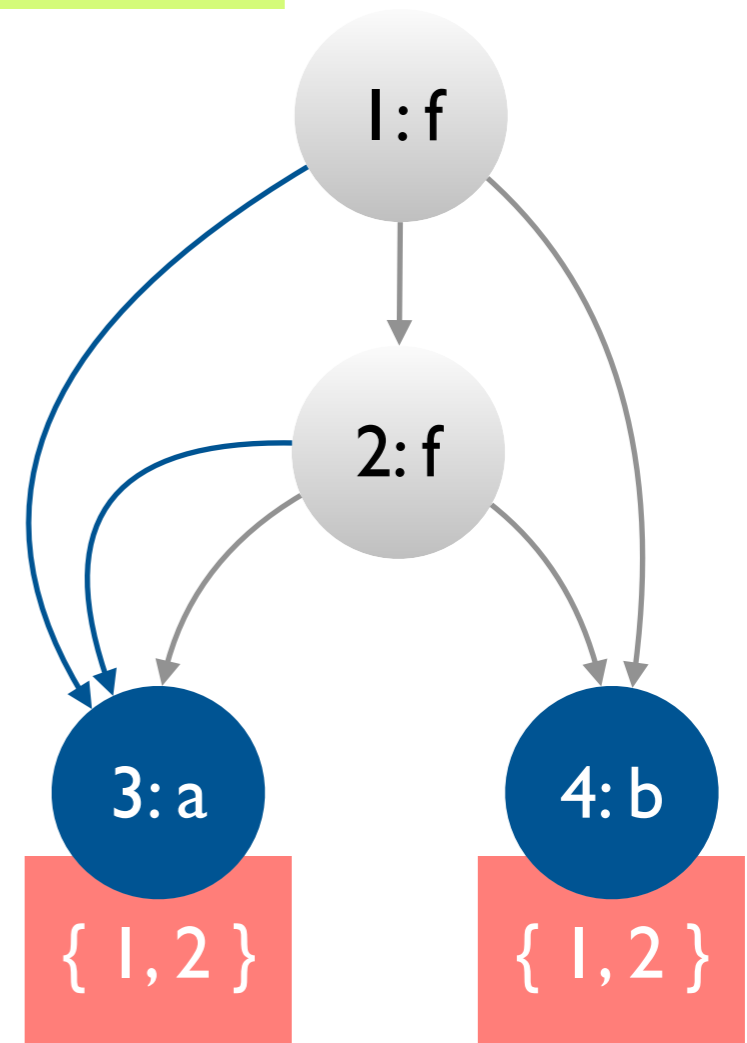
UNION(n_1, n_2)

for each $t_1, t_2 \in p_1 \times p_2$

if $\text{FIND}(t_1) \neq \text{FIND}(t_2) \wedge \text{CONGRUENT}(t_1, t_2)$

then MERGE(t_1, t_2)

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



Congruence closure algorithm: deciding $T=$

DECIDE (F)

construct the DAG for F's subterms

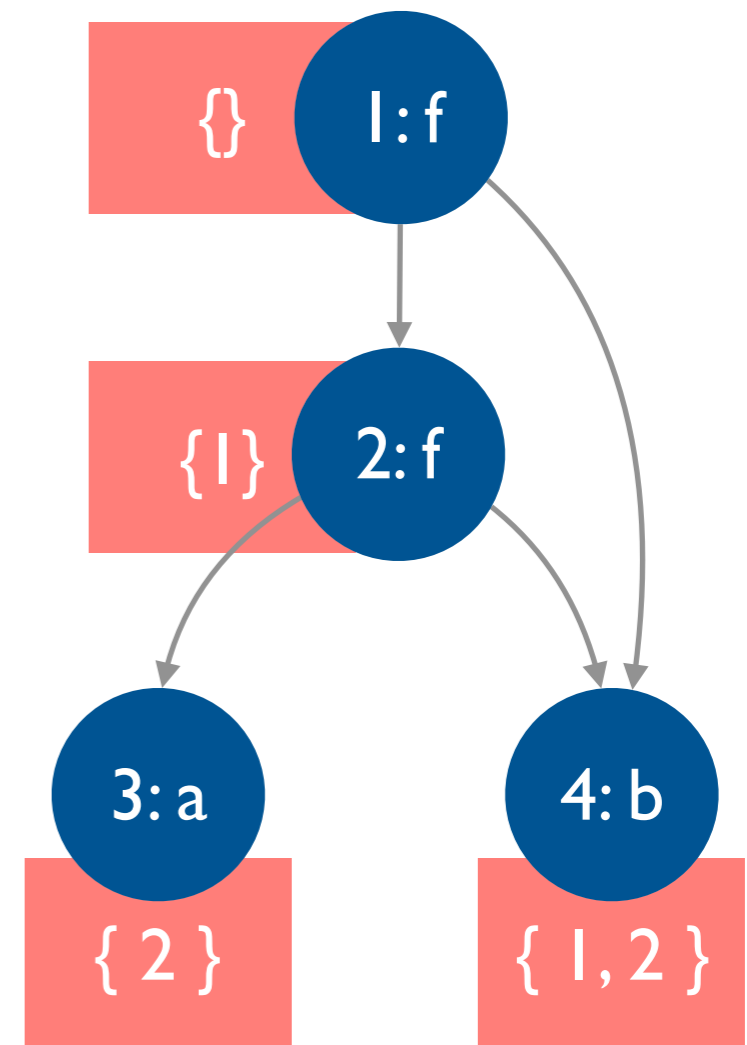
for $s_i = t_i \in F$

MERGE(s_i, t_i)

for $s_i \neq t_i \in F$

if FIND(s_i) = FIND(t_i) **then return** UNSAT
return SAT

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



Congruence closure algorithm: deciding $T=$

DECIDE (F)

construct the DAG for F's subterms

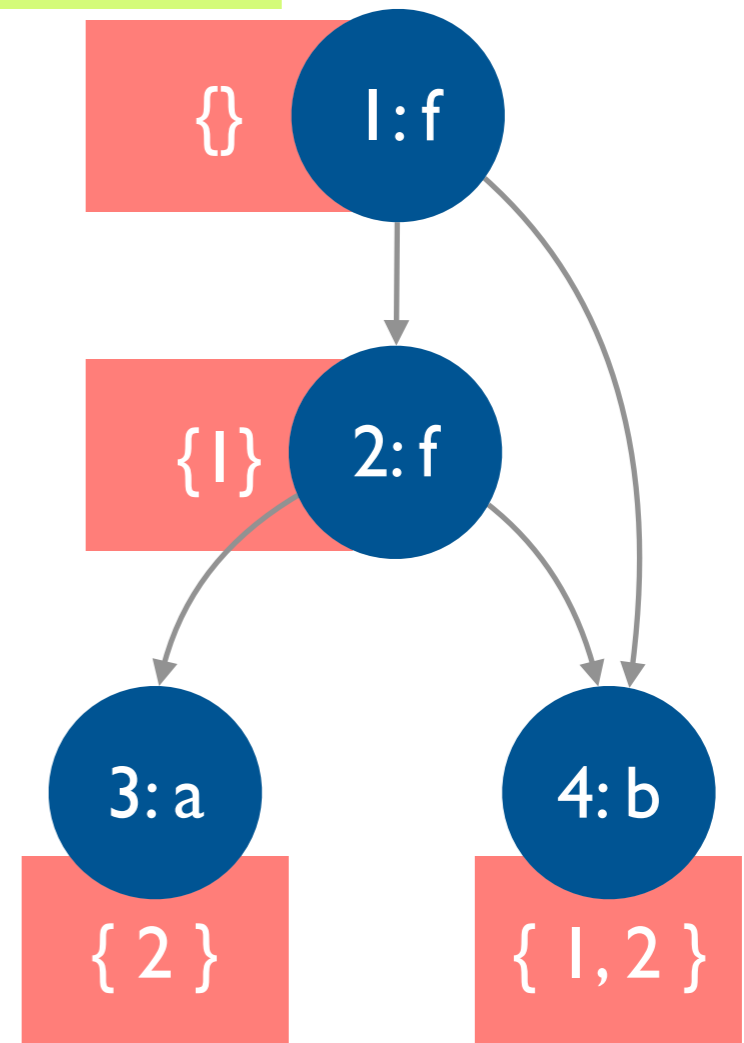
for $s_i = t_i \in F$

MERGE(s_i, t_i)

for $s_i \neq t_i \in F$

if FIND(s_i) = FIND(t_i) **then return** UNSAT
return SAT

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



Congruence closure algorithm: deciding $T=$

DECIDE (F)

construct the DAG for F's subterms

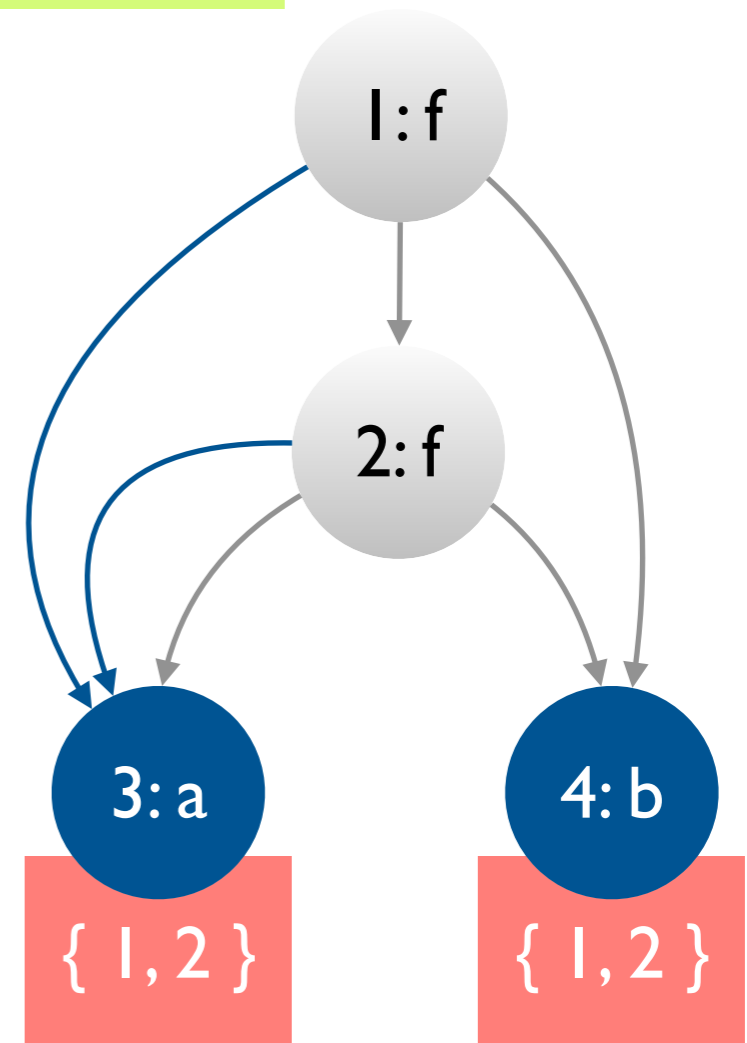
for $s_i = t_i \in F$

MERGE(s_i, t_i)

for $s_i \neq t_i \in F$

if FIND(s_i) = FIND(t_i) **then return** UNSAT
return SAT

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



Congruence closure algorithm: deciding $T=$

DECIDE (F)

construct the DAG for F's subterms

for $s_i = t_i \in F$

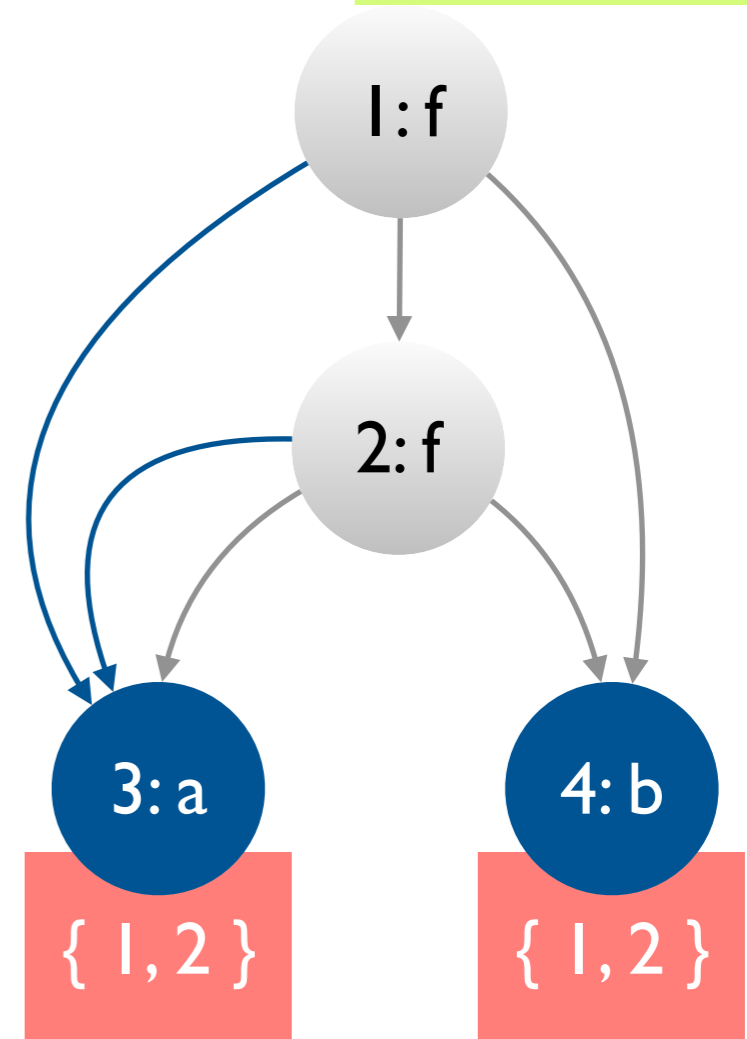
MERGE(s_i, t_i)

for $s_i \neq t_i \in F$

if FIND(s_i) = FIND(t_i) **then return UNSAT**

return SAT

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



Congruence closure algorithm: deciding $T=$

DECIDE (F)

construct the DAG for F's subterms

for $s_i = t_i \in F$

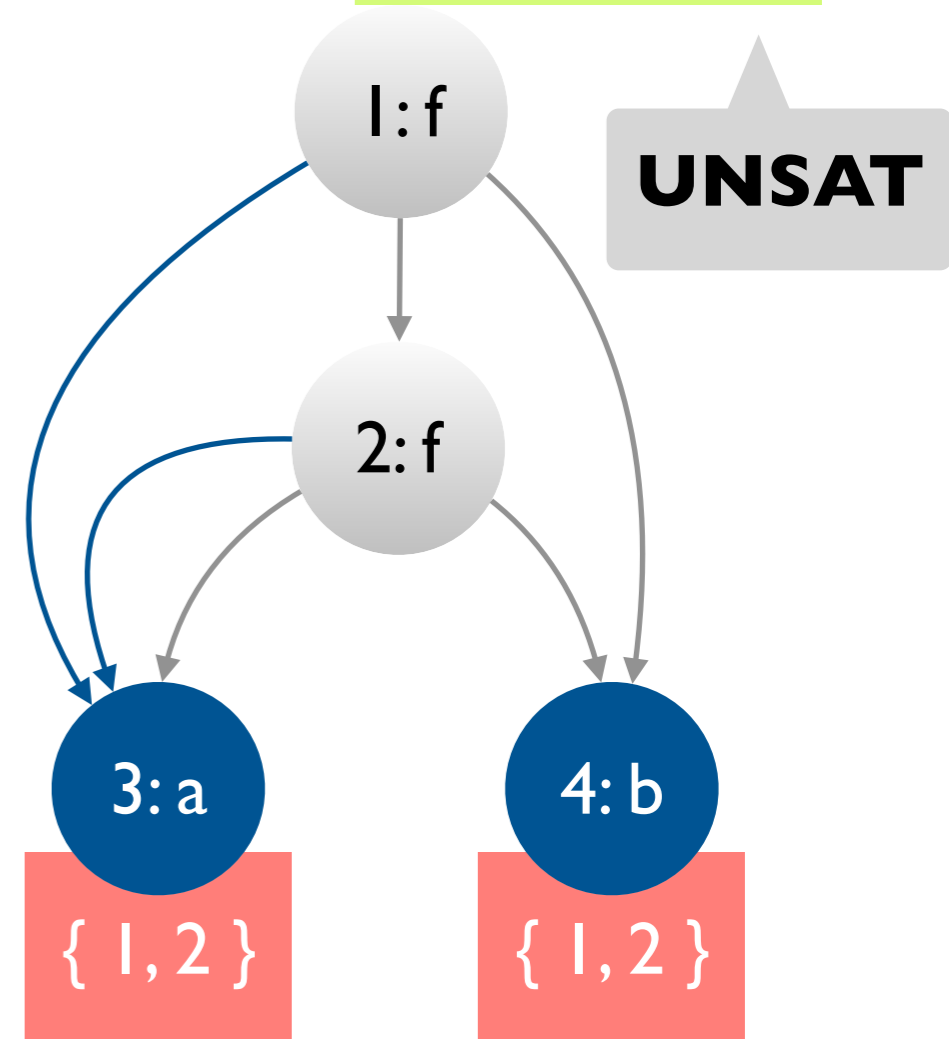
MERGE(s_i, t_i)

for $s_i \neq t_i \in F$

if FIND(s_i) = FIND(t_i) **then return** UNSAT

return SAT

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



Summary

Today

- A brief survey of theory solvers
- Congruence closure algorithm for deciding conjunctive $T=$ formulas

Next lecture

- Combining (decision procedures for different) theories