# Résolution efficace des modèles logiques

## IA303

Alexandre Chapoutot

ENSTA ParisTech

2018-2019

# Course Outline

Main goals of the course:

- To be able to model decision problems using logical formulas combined with other theories.
- Know the solving algorithms for SAT/SMT problems

**Remarks:**

- consider unquantified logical formula
- consider semantic methods based on exhaustivity

# General information

- Teacher: Alexandre Chapoutot
- Mail: `alexandre.chapoutot@ensta-paristech.fr`
- Office: R216 at U2IS, ENSTA ParisTech, 828 bd des maréchaux, Palaiseau
- Class: Tuesday morning (9h-12h)
- Course grade: presentation of scientific papers
- Webpage:

  `http://perso.ensta-paristech.fr/~chapoutot/sat-smt/`

# Syllabus

Nov. 13, 2018 Propositional logic

Nov. 27, 2018 SAT solver algorithms

Dec. 04, 2018 First-order theories and SMT solver algorithms

Dec. 11, 2018 Theory of uninterpreted functions

Dec. 18, 2018 Theory of linear real expressions

Jan. 08, 2019 Theory of nonlinear real expressions

Jan. 15, 2019 Combination of theories

Jan. 22, 2019 Exam

# Theory of equality and uninterpreted functions

# First-order logic (FOL) – syntax 1

The syntax of FOL is made of

- variables: $x$, $y$, $z$, ...
- constants: $a$, $b$, $c$, ...
- functions: $f$, $g$, $h$, ...
- terms are defined inductively
  - variable or constants are terms
  - $n$-ary functions applied to $n$ terms as arguments ars terms
- predicates: $p$, $q$, $r$, ...
- atom: $\bot$, $\top$, or $n$-arry predicates applied to $n$ terms
- literal: atom or its negation

Remark:

- 0-ary functions are constants
- 0-ary predicates are propositional variables

# First-order logic (FOL) – syntax 2

A FOL formula is made of

- literals
- application of logical connectives

$$\neg, \vee, \wedge, \implies, \iff$$

to formula

**Remark**: one can also consider the use of quantifiers

- existential quantifier: $\exists x.f(x)$
- universal quantifier: $\forall x.f(x)$

Note

We focus on unquantified FOL

# Example

$$\forall x.p(f(x),x) \rightarrow (\exists y.p(f(g(x,y)),g(x,y)) \vee q(x,f(x)))$$

The length of one side of a triange is less than the sum of the lengths of the other two sides

$$\forall x,y,z.\text{triangle}(x,y,z) \rightarrow \text{length}(x) < \text{length}(y) + \text{length}(z)$$

# FOL Semantics

An interpretation $I = (D_I, \alpha_I)$ constits of:

- A domain $D_I$, *i.e.* a non empty set of values or objects
  The cardinality $|D_I|$ of $D_I$ can be finite, countably inifinite (*e.g.*, integers), or uncountably integers (*e.g.*, reals)
- An assignement $\alpha_I$ such that
  - each variable $x$ assigned a value $x_I \in D_I$
  - each $n$-ary function $f$ assigned an interpretation function $f_I$

    $$f_I : D_I^n \to D_I$$

    In particular each constant (*i.e.*, 0-ary function) assigned a value $a_I \in D_I$
  - each $n$-ary predicate $p$ assigned an interpretation predicate $p_I$

    $$f_I : D_I^n \to \{\texttt{true}, \texttt{false}\}$$

  In particular, each propositional variable (*e.g.*, 0-ary predicate) assigned a truth value

# FOL Semantics – example

$$F : p(f(x, y), z) \implies p(y, g(z, x))$$

Interpretation $I = (D_I, \alpha_I)$

- $D_I = \mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$
- $\alpha_I = \{f \mapsto +, g \mapsto -, p \mapsto >\}$
  Hence, one has

$$F_I = x + y > z \implies y > z - x$$

Also if $\alpha_I : \{x \mapsto 13, y \mapsto 42, z \mapsto 1\}$ then

$$F_I : 13 + 42 > 1 \implies 42 > 1 - 13$$

and $F$ is true under $I$

# Satisfiability and validity

- $F$ is **satisfiable** iff there exists $I$ s.t. $I \models F$
- $F$ is **valid** iff for all $I$, $I \models F$

$$F \quad \text{is valid} \quad \iff \quad \neg F \quad \text{is unsatisfiable}$$

# Decidability of FOL

FOL is undecidable There does not exist an algorithm for deciding if a FOL formula $F$ is valid, *i.e.*, always halt and says "yes" if $F$ is valid or say "no" if $F$ is invalid.

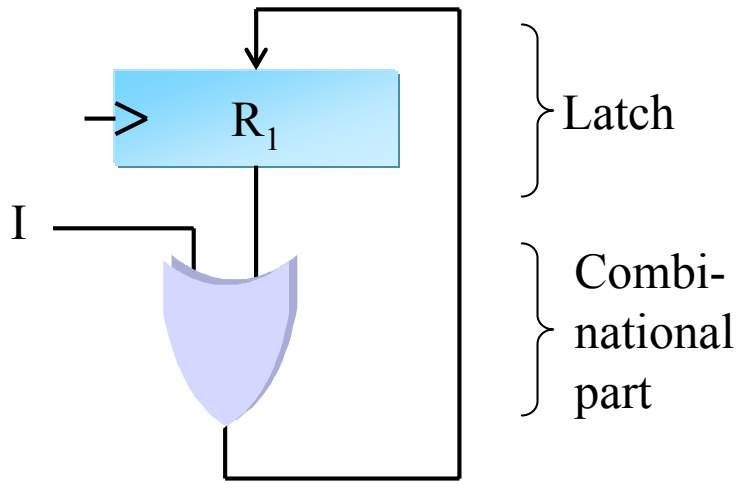
FOL is semi-decidable There is a procedure that always halts and says "yes" if $F$ is valid, but may not halt if $F$ is invalid.

So we usually focus on decidable fragment of FOL

On the other hand, **PL is decidable**. There does exist an algorithm for deciding if a PL formula F is valid, *e.g.*, the truth-table procedure.

# Motivation – 1

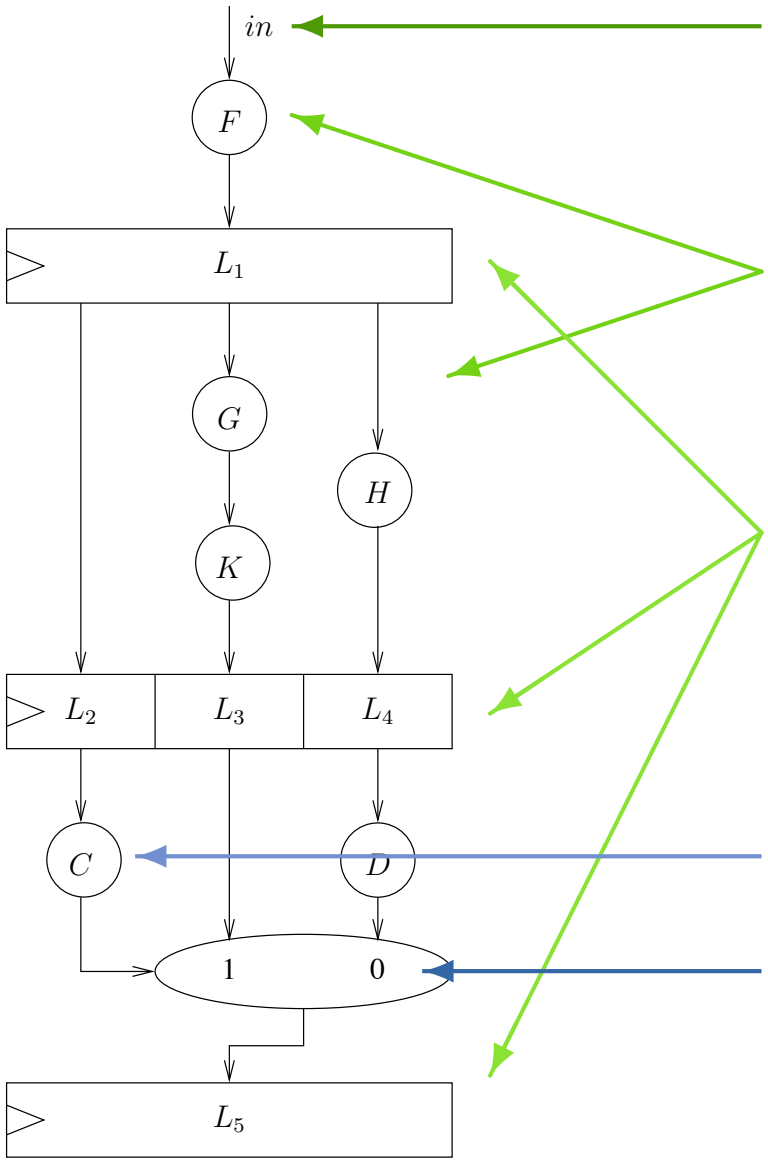
We consider a problem of circuit transformation.




- Circuits consist of combinational gates and latches (registers)
- The combinational gates can be modeled using functions
- The latches can be modeled with variables

$$f(x, y) := x \vee y \quad \text{and} \quad R' = f(R_1, I)$$

# Motivation – 2




$in$: a primary input of the circuit

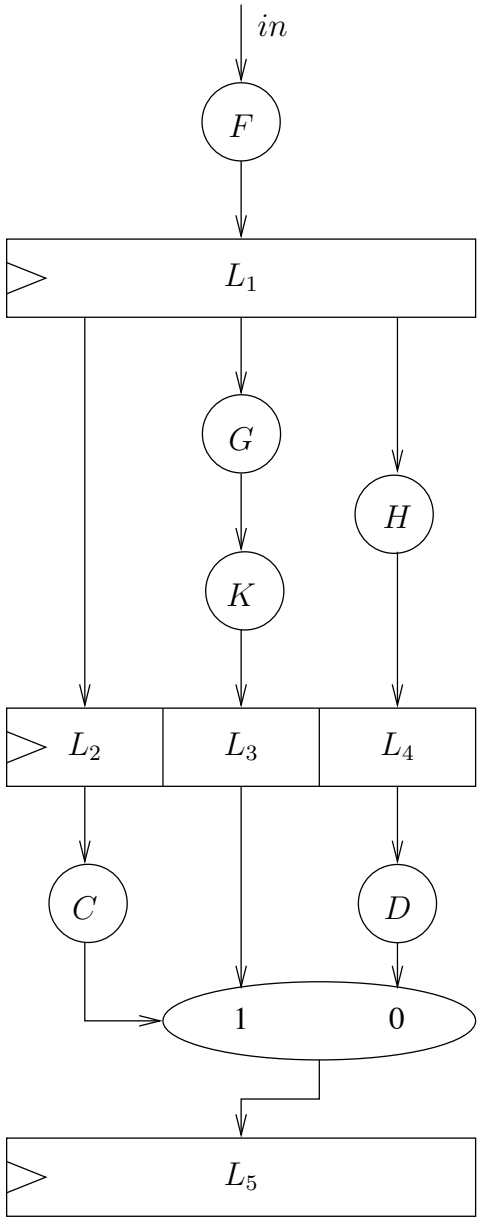
$F, G, H, K, D$: some functions over bit-vectors

$L_1, \ldots, L_5$: latches (registers)

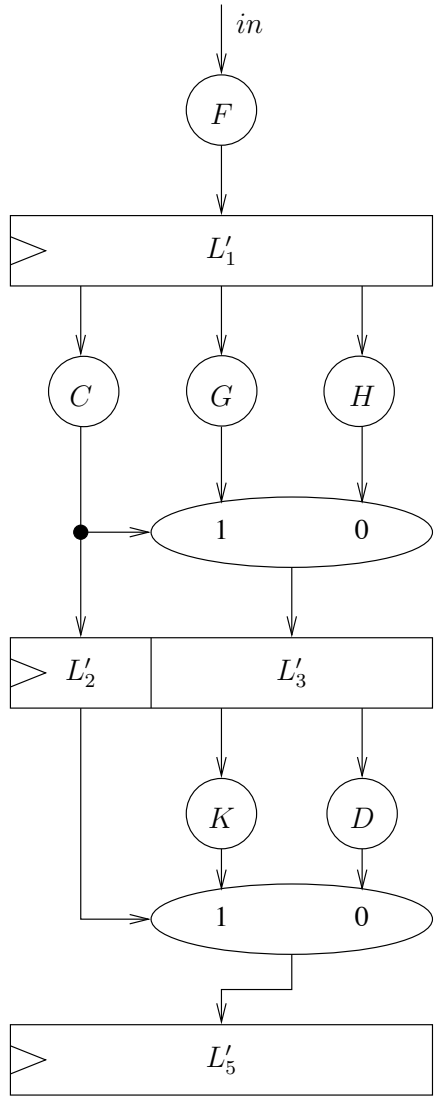
$C$: a predicate over bit-vectors

a multiplexer (case-split)

# Motivation – 3




$\Rightarrow$



# Motivation – 4

$$L_1 = f(I)$$
$$L_2 = L_1$$
$$L_3 = k(g(L_1))$$
$$L_4 = h(L_1)$$
$$L_5 = c(L_2)?L_3 : \ell(L_4)$$

$$L'_1 = f(I)$$
$$L'_2 = c(L'_1)$$
$$L'_3 = c(L'_1)?g(L'_1) : h(L'_1)$$
$$L'_5 = L'_2?k(L'_3) : \ell(L'_3)$$

## Question

Is $L_5 = L'_5$ ?

- Equivalence in this case holds regardless of the actual functions
- Consequence: can be decided using **equality logic and uninterpreted functions**

# Theory of equality and uninterpreted functions – 1

We condiser logic formula made s.t.

$$\Sigma : \{=, a, b, c, \ldots, f, g, h, \ldots\}$$

with uninterpreted symbols

- propositional constants: $a$, $b$, $c$, ...
- functions: $f$, $g$, $h$, ...

## Example – all are unsatisfiable in this logic

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

# Theory of equality and uninterpreted functions – 2

Theory **Axioms**

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

Hence $=$ is an **equivalence relation**

**Axiom schema** for functions

4. for each integer $n > 0$ and $n$-ary function symbol $f$

$$\forall x_1, x_2, \ldots, x_n, y_1, y_2 \ldots, y_n.$$
$$\bigwedge_i x_i = y_i \implies f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n)$$

## Example

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

is $T_E$-valid

# Equivalence and congruence relations – 1

Binary relation $\mathrm{R}$ over set $S$

- is an **equivalence relation** if
  - 
  - reflexive: $\forall s \in S. s\mathrm{R}s$
  - symmetric: $\forall s_1, s_2 \in S. s_1\mathrm{R}s_2 \implies s_2\mathrm{R}s_1$;
  - transitive: $\forall s_1, s_2, s_3 \in S. s_1\mathrm{R}s_2 \wedge s_2\mathrm{R}s_3 \implies s_1\mathrm{R}s_3$.
- is a **congruence relation** if in addition

$$\forall \mathbf{s}, \mathbf{t} \in S^n. \bigwedge_{i=1}^{n} s_i\mathrm{R}t_i \implies f(\mathbf{s})\mathrm{R}f(\mathbf{t})$$

## Example

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

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

Integers $m$, $n$ are related iff they are both odd or both even and $\equiv_2$ is an equivalence relation.

# Equivalence and congruence relations – 2

**Classes** For an equivalence (or congruence) relation $\mathtt{R}$ over the set $S$, the **equivalence (or congruence) class** of $s \in S$ under $\mathtt{R}$ is

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

**Partitions** A **partition** $P$ of $S$ is a set of subsets of $S$ such that

- $P$ is total: $\left(\bigcup_{S' \in P} S'\right) = S$
- elements of $P$ are disjoints: $\forall S_1, S_2 \in P . S_1 \cap S_2 = \emptyset$

**Quotient** the **quotient** $S/\mathtt{R}$ of $S$ by equivalence (or congruence) relation $\mathtt{R}$ is the set of equivalence (or congruence) classes

$$S/\mathtt{R} = \{[s]_{\mathtt{R}} : s \in S\} \quad \text{which forms a partition of } S$$

## Examples

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

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

- The quotien $\mathbb{Z}/\equiv_2$ is a partition of $\mathbb{Z}$, *i.e.*, the set of equivalent classes $\{\{n \in \mathbb{Z} : n \text{ is odd}\}, \{n \in \mathbb{Z} : n \text{ is even}\}\}$

# Equivalence and congruence relations – 3

- **Refinements** Let two binary relations $\mathrm{R}_1$ and $\mathrm{R}_2$ over set $S$. $\mathrm{R}_1$ is a refinement of $\mathrm{R}_2$ (or $\mathrm{R}_1$ refines $\mathrm{R}_2$), $\mathrm{R}_1 \prec \mathrm{R}_2$, if

$$\forall s_1, s_2 \in S. s_1 \mathrm{R}_1 s_2 \implies s_1 \mathrm{R}_2 s_2$$

## Examples

- For $S = \{a, b\}$, if $\mathrm{R}_1 : \{a\mathrm{R}_1 b\}$ and $\mathrm{R}_2 : \{a\mathrm{R}_2 b, b\mathrm{R}_2 b\}$ then $\mathrm{R}_1 \prec \mathrm{R}_2$
- For set S, if

  $\mathrm{R}_1$ induced by the partitions $P_1 : \{\{s\} : s \in S\}$
  $\mathrm{R}_2$ induced by the partitions $P_2 : \{S\}$

  then $\mathrm{R}_1 \prec \mathrm{R}_2$
- For set $\mathbb{Z}$,

  $\mathrm{R}_1 : \{x\mathrm{R}_1 y : x \mod 2 = y \mod 2\}$
  $\mathrm{R}_2 : \{x\mathrm{R}_2 y : x \mod 4 = y \mod 4\}$

  then $\mathrm{R}_2 \prec \mathrm{R}_1$

# Equivalence and congruence relations – 4

- **Closures** Given a binary relation $\mathtt{R}$ over $S$. The **equivalence closure** $\mathtt{R}^E$ of $\mathtt{R}$ is the equivalence relation such that
  - $\mathtt{R}$ refines $\mathtt{R}^E$, *i.e.*, $\mathtt{R} \prec \mathtt{R}^E$
  - for all other equivalence relation $\mathtt{R}'$ such that $\mathtt{R} \prec \mathtt{R}'$ either $\mathtt{R}' = \mathtt{R}^E$ or $\mathtt{R}^E \prec \mathtt{R}'$

  That is $\mathtt{R}^E$ is the "smallest" equivalence relation that "covers" $\mathtt{R}$
  NB: it is similar for **congruence closure**

## Examples

If $S = \{a, b, c, d\}$ and $\mathtt{R} = \{a\mathtt{R}b, b\mathtt{R}c, d\mathtt{R}d\}$ then

- $a\mathtt{R}b$, $b\mathtt{R}c$, $d\mathtt{R}d \in \mathtt{R}^E$ since $\mathtt{R} \subset \mathtt{R}^E$
- $a\mathtt{R}a$, $b\mathtt{R}b$, $c\mathtt{R}c \in \mathtt{R}^E$ by reflexivity
- $b\mathtt{R}a$, $c\mathtt{R}b \in \mathtt{R}^E$ by symmetry
- $a\mathtt{R}c \in \mathtt{R}^E$ by transitivity
- $c\mathtt{R}a \in \mathtt{R}^E$ by symmetry

Hence,

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

# Congruence closure algorithm – 1

**Goal of this algorithm** Given a $\Sigma_E$-formula (in NNF)

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

decides if $F$ is $\Sigma_E$-satisfiable.

## Definition

For $\Sigma_E$ formula, the **subterm set** $S_F$ of $F$ is the set containing precisely the subterms of $F$.

## Example

The subterm set of

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

is

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

# Congruence closure algorithm – 2

**Algorithm**
Given a $\Sigma_E$ formula $F$

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

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

- for each $i \in \{1, \ldots, m\}$, $s_i \sim t_i$
- for each $i \in \{m+1, \ldots, n\}$, $s_i \not\sim t_i$

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 $\sim$.
The goal of the algorithm is to constuct the congruence relation of $S_F$, or to prove that no congruence relation exists.

# Congruence closure algorithm – 3

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

The algorithm perfoms these steps

1. Construct the congruence closure $\sim$ of

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

   over the subterm set $S_F$, then

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

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

## Question

How do we construct the congruence closure in Step 1?

# Congruence closure algorithm – 3

## Question

How do we construct the congruence closure in Step 1?

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

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

That is, let eeach term in $S_F$ be its own congruence class.
Then, for each $i \in \{1, \ldots, m\}$, 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 uniont of $[s_i]_{\sim_{i-1}}$ and $[t_i]_{\sim_{i-1}}$
- propagate any new congruences that arise with this union

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

# Example – 1

**Input $\Sigma_E$ formula** $F$: $F : f(a,b) = a \wedge f(f(a,b),b) \neq a$
Constuct initial partition by letting each element 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 $f(a,b) = a$ in $F$ merge $\{f(a,b)\}$ and $\{a\}$ to form the partition

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

According to the (congruence) axiom,

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

resulting in the new partition

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

This partition represents the congruence closure of $S_F$. Is it the case that

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

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

# Example – 2

**Input $\Sigma_E$ formula $F$:**
$F : f(f(f(a))) = a \wedge f(f(f(f(f(a))))) = a \wedge f(a) \neq a$
From the subterm set $F$, the initial partition is

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

According to literal $f^3(a) = a$, merge $\{f^3(a)\}$ and $\{a\}$

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

In which the following congruence propagation can be deduced:

$$f^3(a) \sim a \implies f(f^3(a)) \sim f(a)$$

and

$$f^4(a) \sim f(a) \implies f(f^4(a)) \sim f^2(a)$$

Thus

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

# Example – 2

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

From the literal $f^5(a) = a$, merge $\{f^2(a), f^5(a)\}$ and $\{a, f^3(a)\}$ to form the partition

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

Propagating the congruence

$$f^3(a) \sim f^2(a) \implies f(f^3(a)) \sim f(f^2(a))$$

yields the partion

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

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

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

No as $f(a) \sim a$ and $F$ asserts that $f(a) \neq a$ so UNSAT

# Example – 3

**Input $\Sigma_E$ formula** $F$: $F : f(x) = f(y) \wedge x \neq y$
The subterm set $S_F$ induces the 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. $\{\{x\}, \{y\}, \{f(x), f(y)\}\}$

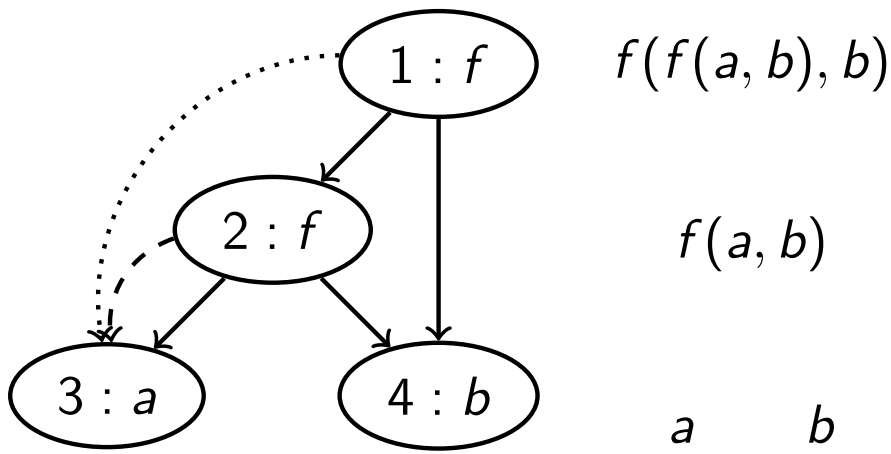
So

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

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

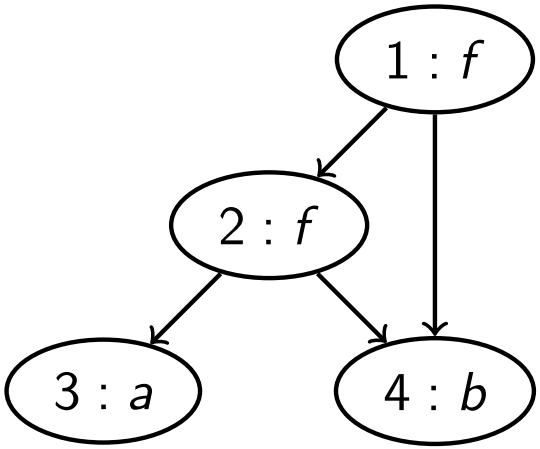
# Efficient implementation of congruence closure

An efficient representation of subterm set $S_F$ is given by graph-based data structure, *i.e.*, a directed acyclic graph (DAG).

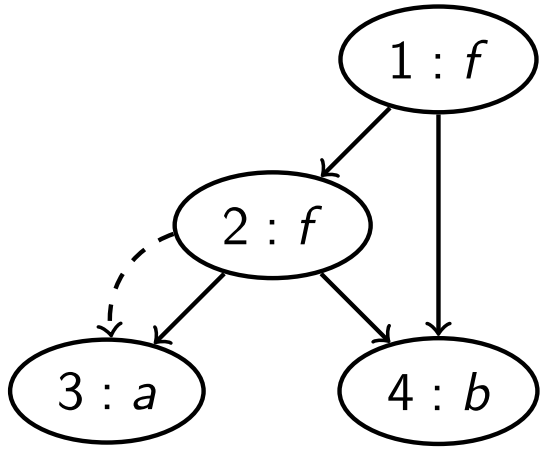


# Intuition of the implementation

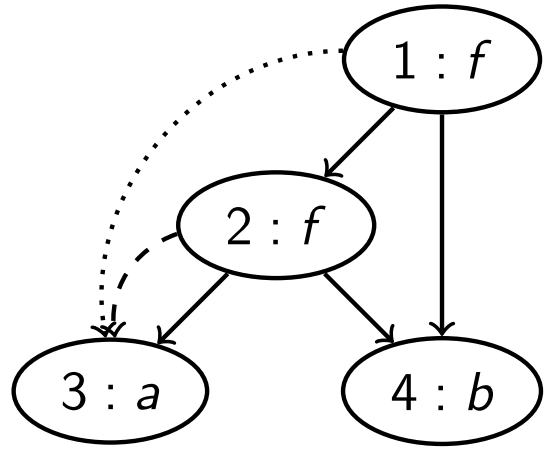
1. Build the DAG to compute congruence closure classes




Initial DAG




$f(a,b) = a \implies$
MERGE $f(a,b)$ and $a$




$f(a,b) \sim a, b \sim b \implies$
$f(f(a,b),b) \sim f(a,b)$
MERGE $f(f(a,b),b)$
and $f(a,b)$

Legend of links: _ _ _ by explicit equations; ..... by congruence

2. Look for a contradiction

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

# DAG implementation

```
type node = {
  id: id_t;
  fn: string;
  args: id_t list;
  mutable find: id_t;
  mutable ccpar: id_t set;
}
```

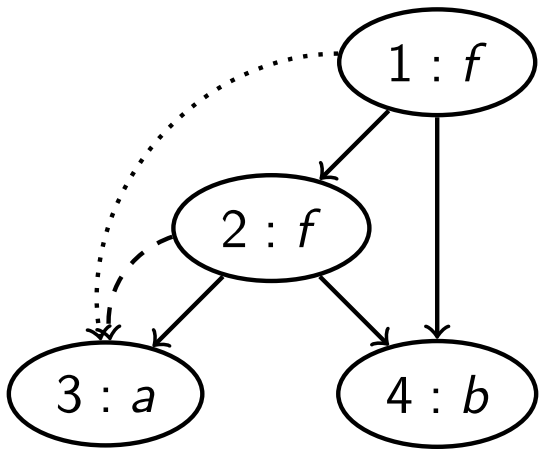
with

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

# DAG implementation – example 1

DAG node implementation for node 2

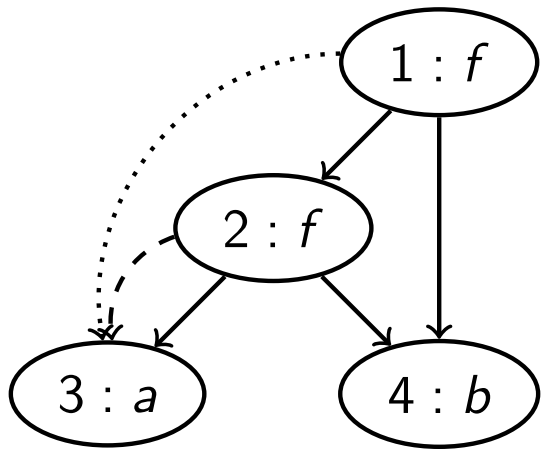
```
type node = {
   id = 2;
   fn = "f";
   args= [3, 4];
   find = 3;
   ccpar: ∅;
}
```



# DAG implementation – example 2

DAG node implementation for node 3

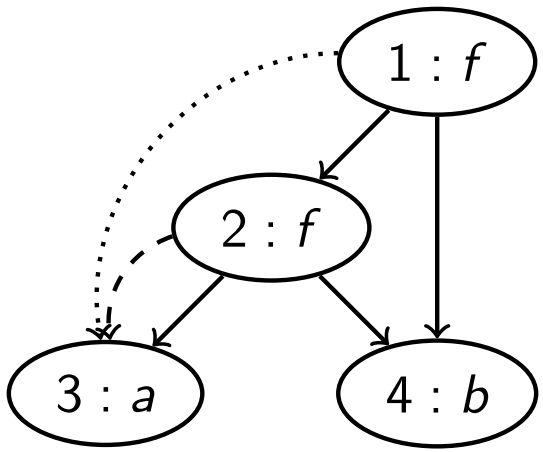
```
type node = {
   id = 3;
   fn = ``a'';
   args= [];
   find = 3;
   ccpar: {1, 2};
}
```



# DAG implementation – find

- Function **node** *i* returns the node with id i in a given DAG.
- Function **find** returns the representative of the 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 retruns 3
- find 3 returns 3

3 is the representative of 2

# DAG implementation – union

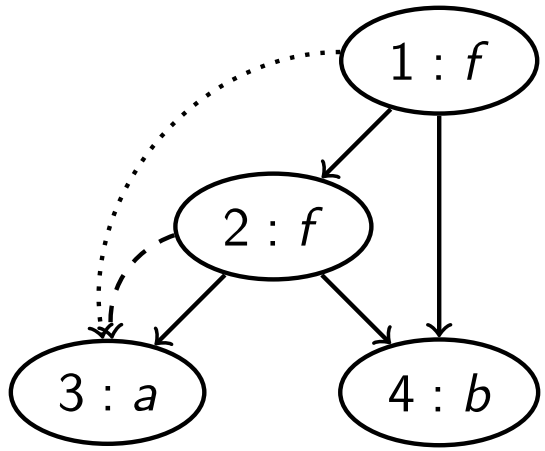
- Function **union**

```
let union i1 i2 =
    let n1 = node (find i1) in
    let n2 = node (find i2) in
    n1.find ← n2.find;
    n2.ccpar ← n1.ccpar ∪ n2.ccpar;
    n1.ccpar ← ∅
```

n2 is the representative of the union class

# DAG implementation – union

## Example




The call of "union 1 2" produces

- n1 = 1 and n2 = 3
- 1.find $\leftarrow$ 3
- 3.ccpar $\leftarrow \{1, 2\}$
- 1.ccpar $\leftarrow \emptyset$

# DAG implementation – ccpar and congruent

- Function **ccpar** returns parents of all nodes in i's congruence class

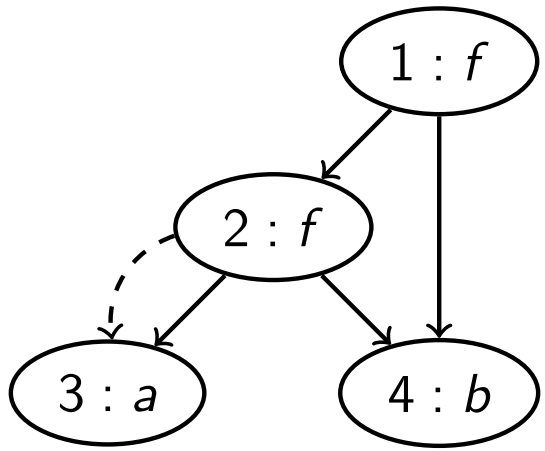
```
let ccpar i =
    (node (find i)).ccpar
```

- Predicate **congruent** tests whether i1 and i2 are congruent

```
let congruent i1 i2 =
  let n1 = node i1 in
  let n2 = node i2 in
  let len = List.length n1.args in
  n1.fn = n2.fn &&
  len = List.length n2.args &&
  ∀i ∈ {1,...,len} find n1.args[i] = find n2.args[i]
```

# DAG implementation – ccpar and congruent

## Example




Are 1 and 2 congruent?

- fn fields are the same
- number of arguments are the same
- left arguments $f(a, b)$ and $a$ are both congruent to 3
- right arguments $b$ and $b$ are both congruent to 4

Therefore 1 and 2 are congruent

# DAG implementation – merge

- Function **merge**

```
let rec merge i1 i2 =
    if find i1 <> find i2
    then begin
        let pi1 = ccpar i1 in
        let pi2 = ccpar i2 in
        union i1 i2;
        foreach t1, t2 in (pi1, pi2) do
            if find t1 <> find t2 && congruent t1 t2
                then merge t1 t2
        done
    end
```

pi1 and pi2 store the current values of ccpar i1 and ccpar i2

# Decision procedure

Given a $Sigma_E$ formula

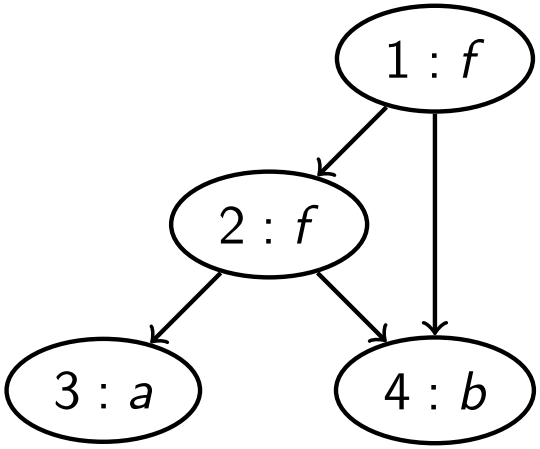
$$F : s_1 = t_1 \wedge \cdots \wedge s_m = t_m \wedge s_{m+1} \neq t_{m+1} \wedge \cdots s_n \neq t_n$$

with subterm sets $S_F$, perform the following steps:

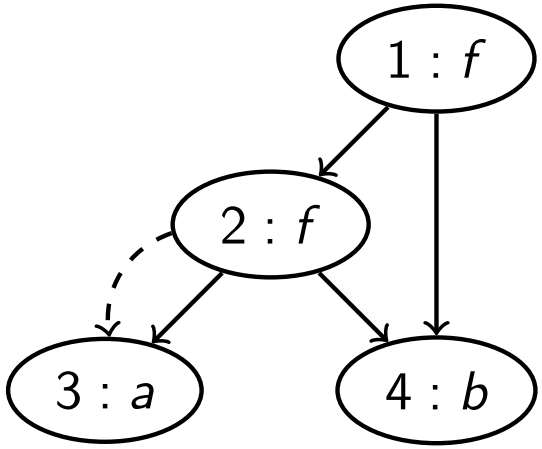
1. construct the initial DAG for the subterm set $S_F$;
2. For $i \in \{1, \ldots, m\}$, merge $s_i$ $t_i$;
3. If find$s_i$ = find$t_i$ for some $i \in \{m+1, \ldots, n\}$, return UNSAT
4. Otherwise (if find$s_i \neq$ find$t_i$ for all $i \in \{m+1, \ldots, n\}$), return SAT

# Decision procedure – Example 1

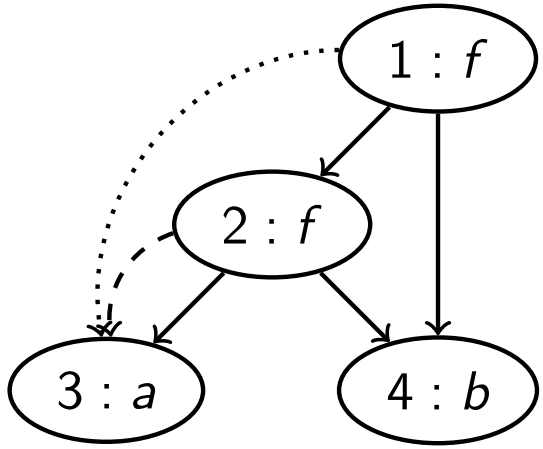
$$f(a,b) = a \wedge f(f(a,b),b) \neq a$$




Initial DAG




merge 2 3
union 2 3
$P_2 = \{1\}$
$P_3 = \{2\}$
congruent 1 2




merge 1 2
union 1 2
$P_1 = \{\}$
$P_2 = \{1,2\}$

find $f(f(a,b),b) = a =$ find $a \implies$ UNSAT

# Decision procedure – Example 1

Given $\Sigma_E$ 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 the inital partition

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

in which each term is its own congruence class.

Final partition

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

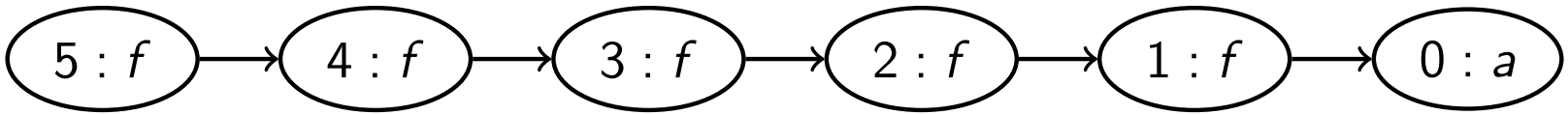
Does

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

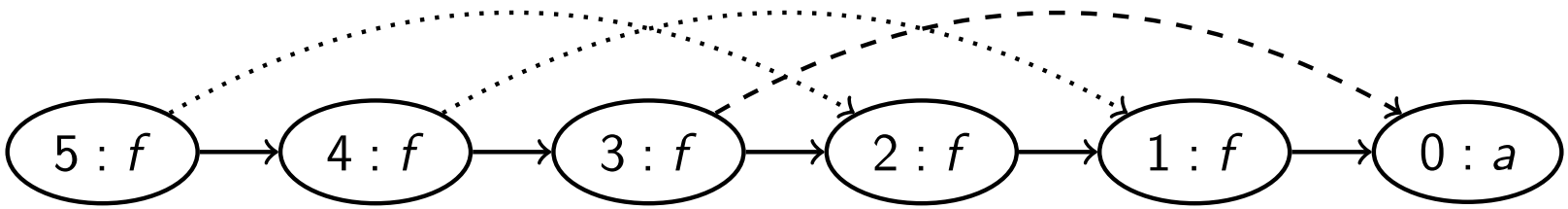
No as $f(f(a,b),b) \sim a$ but $F$ asserts that $f(f(a,b),b) \neq a$
Hence $F$ is UNSAT

# Decision procedure – Example 2

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



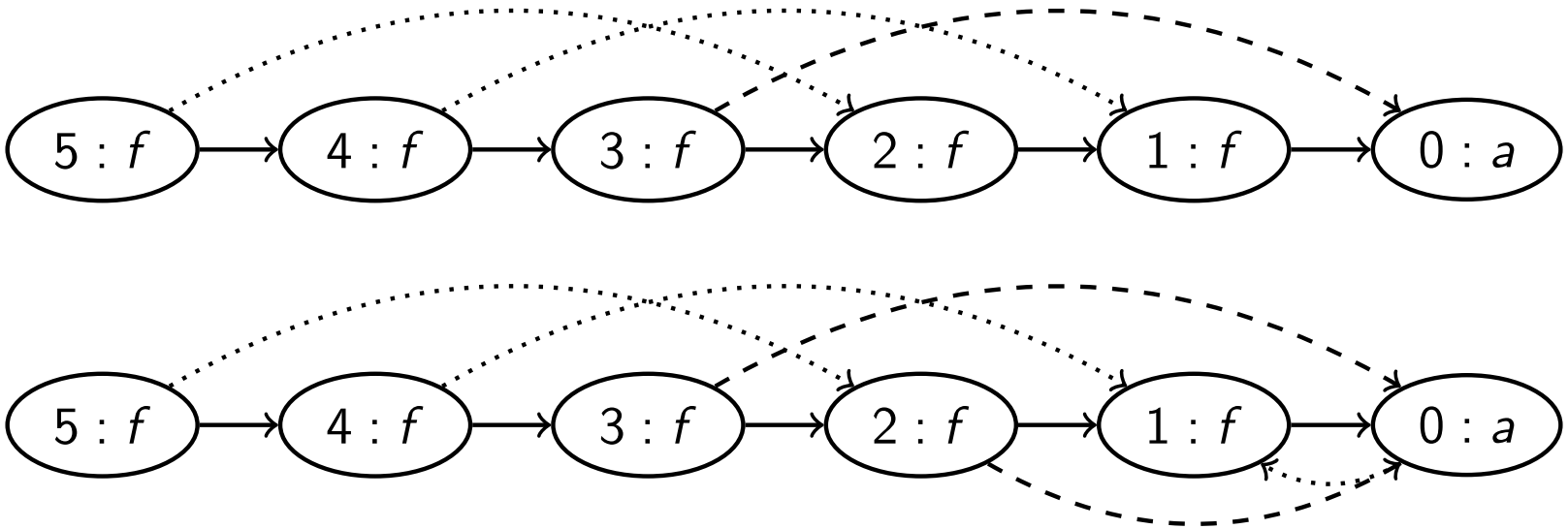

Initial DAG




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

# Decision procedure – Example 2

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




$$f(f(f(f(f(a))))) = a \implies \text{merge}\,5\,0 \quad P_5 = \{3\} \quad P_0 = \{1,4\}$$
$$\implies \text{merge}\,3\,1 \quad \text{STOP}$$

$$\text{find}\ f(a) = f(a) = \text{find}\ a \implies \text{UNSAT}$$

# Decision procedure – Example 2

Given $\Sigma_E$ formula

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

1. which induces the initial partition

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

2. The equality $f^3(a) = a$ induces the partition

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

3. The equality $f^5(a) = a$ induces the partition

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

Now does

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

No as $f(a) \sim a$ but $F$ asserts that $f(a) \neq a$. Hence $F$ UNSAT

# Soudness and completness

Quantifier-free conjunction of $\Sigma_E$ formula $F$ is $T_E$-satisfiable iff congruence closure algorithm returns satisfiable.