Efficient resolution of logical models

ENSTA-IA303

Alexandre Chapoutot and Sergio Mover

ENSTA Paris

2020-2021

Lecture 4: A Decision Procedure for the Theory of Equality

Main goals for today

In class[1]:

- How to decide the $\mathcal{T}$-satisfiability for quantifier-free formula in the Equality

[1] Main references:

- The Calculus of Computation [Bradley and Manna, 2007], Chapter 9 (Section 9.1, 9.2, 9.3)

Main goals for today

In class[1]:

- How to decide the $\mathcal{T}$-satisfiability for quantifier-free formula in the Equality

In the tutorial:

- Implement the decision procedure

[1] Main references:

- The Calculus of Computation [Bradley and Manna, 2007], Chapter 9 (Section 9.1, 9.2, 9.3)

1. A Decision Procedure for the Theory of Equality
   - T_E-satisfiability
   - Deciding T_E via Congruence Closure
   - An algorithm to computing congruence closure

Why the Theory of Equality $\mathcal{T}_E$?

- *Base theory*: in most cases we assume the equality predicate $=$ to be part of any theory (i.e., interpreted as equality)
  Also called Theory of Equality and Uninterpreted Functions (EUF)

Why the Theory of Equality $\mathcal{T}_E$?

- *Base theory*: in most cases we assume the equality predicate $=$ to be part of any theory (i.e., interpreted as equality)
  Also called Theory of Equality and Uninterpreted Functions (EUF)
- We use $\mathcal{T}_E$ to *combine* different theories (i.e., exchanging equalities)

Why the Theory of Equality $\mathcal{T}_E$?

- *Base theory*: in most cases we assume the equality predicate $=$ to be part of any theory (i.e., interpreted as equality)
  Also called Theory of Equality and Uninterpreted Functions (EUF)
- We use $\mathcal{T}_E$ to *combine* different theories (i.e., exchanging equalities)
- We use T_E in the "layered" approach:
  - We can first check if a formula is satisfiable considering all the function symbols "uninterpreted"
  - If the formula is unsatisfiable with T_E, then the formula is unsatisfiable in the "original" theory.
    Example from [Barrett et al., 2009]:

    $$a * (f(b) + f(c)) = d \wedge \neg(b * (f(a) + f(c)) = d) \wedge a = b$$

    Already unsatisfiable if we consider $*$ and $+$ as an uninterpreted function.

Why the Theory of Equality $\mathcal{T}_E$?

- *Base theory*: in most cases we assume the equality predicate $=$ to be part of any theory (i.e., interpreted as equality)
  Also called Theory of Equality and Uninterpreted Functions (EUF)
- We use $\mathcal{T}_E$ to *combine* different theories (i.e., exchanging equalities)
- We use T_E in the "layered" approach:
  - We can first check if a formula is satisfiable considering all the function symbols "uninterpreted"
  - If the formula is unsatisfiable with T_E, then the formula is unsatisfiable in the "original" theory.
    Example from [Barrett et al., 2009]:

    $$a * (f(b) + f(c)) = d \wedge \neg(b * (f(a) + f(c)) = d) \wedge a = b$$

    Already unsatisfiable if we consider $*$ and $+$ as an uninterpreted function.
  - T_E solver is "cheap", so we can run it before calling more expensive theory solvers.

Theory of Equality

The Theory of Equality functions $\mathcal{T}_E$ is defined as:

- the signature $\Sigma_E := \{=, a, b, c, \ldots, f, g, h, \ldots, p, q, r, \ldots\}$
  - $=$ is a binary predicate and is *interpreted* as the equality
  - all the other function symbols in Σ_E are not interpreted

Theory of Equality

The Theory of Equality functions $\mathcal{T}_E$ is defined as:

- the signature $\Sigma_E := \{=, a, b, c, \ldots, f, g, h, \ldots, p, q, r, \ldots\}$
  - $=$ is a binary predicate and is *interpreted* as the equality
  - all the other function symbols in Σ_E are not interpreted
- the set of axioms $\mathcal{A}$:
  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]

Theory of Equality

The Theory of Equality functions $\mathcal{T}_E$ is defined as:

- the signature $\Sigma_E := \{=, a, b, c, \ldots, f, g, h, \ldots, p, q, r, \ldots\}$
  - $=$ is a binary predicate and is *interpreted* as the equality
  - all the other function symbols in Σ_E are not interpreted
- the set of axioms $\mathcal{A}$:
  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]
  4. Function and predicate congruence
     - ⋆ For each $n \in \mathbb{N}$ and n-ary function symbol f:

$$\forall x_1, \ldots, x_n, y_1, \ldots, y_n. \left(\bigwedge_{i=1}^{n} x_i = x_j \right) \rightarrow f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n)$$

Theory of Equality

The Theory of Equality functions $\mathcal{T}_E$ is defined as:

- the signature $\Sigma_E := \{=, a, b, c, \dots, f, g, h, \dots, p, q, r, \dots\}$
  - $=$ is a binary predicate and is *interpreted* as the equality
  - all the other function symbols in Σ_E are not interpreted
- the set of axioms $\mathcal{A}$:
  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]
  4. Function and predicate congruence
     - For each $n \in \mathbb{N}$ and n-ary function symbol f:

$$\forall x_1, \dots, x_n, y_1, \dots, y_n. \left(\bigwedge_{i=1}^{n} x_i = x_j \right) \rightarrow f(x_1, \dots, x_n) = f(y_1, \dots, y_n)$$

     - For each $n \in \mathbb{N}$ and n-ary predicate symbol p:

$$\forall x_1, \dots, x_n, y_1, \dots, y_n. \left(\bigwedge_{i=1}^{n} x_i = x_j \right) \rightarrow p(x_1, \dots, x_n) \leftrightarrow p(y_1, \dots, y_n)$$

Some concrete examples

$$a \neq b$$

Is sat? Is valid?

Some concrete examples

$$a \neq b$$

Is sat? *Yes* Is valid?

Some concrete examples

$$a \neq b$$

Is sat? *Yes*   Is valid? *No*

Some concrete examples

$$a \neq b$$

Is sat? *Yes*   Is valid? *No*

$$a = b \wedge b = c \leftrightarrow f(c) = f(a)$$

Is sat?   Is valid?

Some concrete examples

$$a \neq b$$

Is sat? *Yes* Is valid? *No*

$$a = b \wedge b = c \leftrightarrow f(c) = f(a)$$

Is sat? *Yes* Is valid?

Some concrete examples

$$a \neq b$$

Is sat? *Yes* Is valid? *No*

$$a = b \wedge b = c \leftrightarrow f(c) = f(a)$$

Is sat? *Yes* Is valid? *Yes*

Some concrete examples

$$a \neq b$$

Is sat? *Yes* Is valid? *No*

$$a = b \land b = c \leftrightarrow f(c) = f(a)$$

Is sat? *Yes* Is valid? *Yes*

$$a = b \land b = c \implies g(f(a), b) = g(f(c), a)$$

Is sat? Is valid?

Some examples are from [Bradley and Manna, 2007] and [Barrett et al., 2009]

Some concrete examples

$$a \neq b$$

Is sat? *Yes* Is valid? *No*

$$a = b \wedge b = c \leftrightarrow f(c) = f(a)$$

Is sat? *Yes* Is valid? *Yes*

$$a = b \wedge b = c \implies g(f(a), b) = g(f(c), a)$$

Is sat? *Yes* Is valid?

Some examples are from [Bradley and Manna, 2007] and [Barrett et al., 2009]

Some concrete examples

$$a \neq b$$

Is sat? *Yes* Is valid? *No*

$$a = b \wedge b = c \leftrightarrow f(c) = f(a)$$

Is sat? *Yes* Is valid? *Yes*

$$a = b \wedge b = c \implies g(f(a), b) = g(f(c), a)$$

Is sat? *Yes* Is valid? *Yes*

Some examples are from [Bradley and Manna, 2007] and [Barrett et al., 2009]

Our settings

- The problem we solve today: is a Σ_E-formula $\mathcal{T}_E$-satisfiable?
- We consider a conjunction of theory literals where atoms are equalities

$$x = y \wedge f(x) = y \wedge (\neg f(g(x, y)) = f(x))$$

We can enumerate such conjunctions for an arbitrary Σ_E-formula using the lazy approach.

Our settings

- The problem we solve today: is a Σ_E-formula $\mathcal{T}_E$-satisfiable?
- We consider a conjunction of theory literals where atoms are equalities

$$x = y \wedge f(x) = y \wedge (\neg f(g(x, y)) = f(x))$$

  We can enumerate such conjunctions for an arbitrary Σ_E-formula using the lazy approach.
- Here we do not consider predicates.
  In general: replace predicates with functions to get an *equisatisfiable* formula

Example

$$p(x, y) \wedge q(f(y)) \wedge f(x) = y \quad \Longrightarrow \quad f_p(x, y) = v_{\mathcal{T}} \wedge f_q(f(y)) = v_{\mathcal{T}} \wedge f(x) = y$$

$v_{\mathcal{T}}$ is a fresh value, f_p, q_p are fresh function symbols. Intuitively, the transformation assumes that:
$\forall x, y. p(x, y) \leftrightarrow f_p(x, y) = v_{\mathcal{T}}$ and $\forall x. q(x) \leftrightarrow f_q(x) = v_{\mathcal{T}}$

1. A Decision Procedure for the Theory of Equality
   - T_E-satisfiability
   - Deciding T_E via Congruence Closure
   - An algorithm to computing congruence closure

A first intuition about deciding T_E formulas[2]

$$\phi := f(f(f(a))) = a \land f(f(f(f(f(a))))) = a \land \neg f(a) = a$$

- From $f(f(f(a))) = a$:
  - Substitute $f(f(f(a)))$ with a in $f(f(f(f(f(a))))) = a$
  - Infer new equality $f(f(a)) = a$

[2] Example 9.1 from [Bradley and Manna, 2007]

A first intuition about deciding T_E formulas[2]

$$\phi := f(f(f(a))) = a \land f(f(f(f(f(a))))) = a \land \neg f(a) = a$$

- From $f(f(f(a))) = a$:
  - Substitute $f(f(f(a)))$ with a in $f(f(f(f(f(a))))) = a$
  - Infer new equality $f(f(a)) = a$
- From $f(f(a)) = a$:
  - Substitute $f(f(a))$ with a in $f(f(f(a))) = a$
  - Infer new equality: $f(a) = a$

[2] Example 9.1 from [Bradley and Manna, 2007]

A first intuition about deciding T_E formulas[2]

$$\phi := f(f(f(a))) = a \wedge f(f(f(f(f(a))))) = a \wedge \neg f(a) = a$$

- From $f(f(f(a))) = a$:
  - Substitute $f(f(f(a)))$ with a in $f(f(f(f(f(a))))) = a$
  - Infer new equality $f(f(a)) = a$
- From $f(f(a)) = a$:
  - Substitute $f(f(a))$ with a in $f(f(f(a))) = a$
  - Infer new equality: $f(a) = a$
- We have both $f(a) = a$ and $\neg f(a) = a$: contradiction. So, ϕ is unsatisfiable.

[2] Example 9.1 from [Bradley and Manna, 2007]

A first intuition about deciding T_E formulas[2]

$$\phi := f(f(f(a))) = a \wedge f(f(f(f(f(a))))) = a \wedge \neg f(a) = a$$

- From $f(f(f(a))) = a$:
  - Substitute $f(f(f(a)))$ with a in $f(f(f(f(f(a))))) = a$
  - Infer new equality $f(f(a)) = a$
- From $f(f(a)) = a$:
  - Substitute $f(f(a))$ with a in $f(f(f(a))) = a$
  - Infer new equality: $f(a) = a$
- We have both $f(a) = a$ and $\neg f(a) = a$: contradiction. So, ϕ is unsatisfiable.

Use the equalities to infer new equalities, applying the T_E axioms, and then check for contradictions with the inequalities

[2]Example 9.1 from [Bradley and Manna, 2007]

Decision procedure for T_E

$$\phi := [s_1 = t_1, \ldots s_m = t_m, \neg(s_{m+1} = t_{m+1}), \ldots \neg(s_n = t_n)]$$

Decision procedure for T_E

$$\phi := [s_1 = t_1, \ldots s_m = t_m, \neg(s_{m+1} = t_{m+1}), \ldots \neg(s_n = t_n)]$$

1. Apply the T_E axioms (relfexivity, symmetry, transitivity, and congruence) to the existing equalities, inferring a set of new equalities.
   - Since the possible terms in ϕ are finite, then also the number of inferred equalities are finite.
   - So, this enumeration terminates.

Decision procedure for T_E

$$\phi := [s_1 = t_1, \dots s_m = t_m, \neg(s_{m+1} = t_{m+1}), \dots \neg(s_n = t_n)]$$

1. Apply the T_E axioms (relfexivity, symmetry, transitivity, and congruence) to the existing equalities, inferring a set of new equalities.
   - Since the possible terms in ϕ are finite, then also the number of inferred equalities are finite.
   - So, this enumeration terminates.

2. For every inequality $\neg s_i = t_i$ ($i \in [m+1, n]$), check if $s_i = t_i$ is in the set of inferred equalities.
   - If we find such $\neg s_i = t_i$ and $s_i = t_i$, then ϕ is unsatisfiable
   - Otherwise, ϕ is satisfiable

Decision procedure for T_E

$$\phi := [s_1 = t_1, \dots s_m = t_m, \neg(s_{m+1} = t_{m+1}), \dots \neg(s_n = t_n)]$$

1. Apply the T_E axioms (relfexivity, symmetry, transitivity, and congruence) to the existing equalities, inferring a set of new equalities.
   - Since the possible terms in ϕ are finite, then also the number of inferred equalities are finite.
   - So, this enumeration terminates.

This is called congruence closure

2. For every inequality $\neg s_i = t_i$ ($i \in [m+1, n]$), check if $s_i = t_i$ is in the set of inferred equalities.
   - If we find such $\neg s_i = t_i$ and $s_i = t_i$, then ϕ is unsatisfiable
   - Otherwise, ϕ is satisfiable

Equivalence and Congruence Relations

- S is a set, and R is a *binary relation* over S

Equivalence and Congruence Relations

- S is a set, and R is a *binary relation* over S
- R is an *equivalence relation* if:
  - Reflexivity: $\forall s_1 \in S.(s_1, s_1) \in R$
  - Symmetry: $\forall s_1, s_2 \in S.(s_1, s_2) \in R$
  - Transitivity: $\forall s_1, s_2, s_3 \in S.((s_1, s_2) \in R \wedge (s_2, s_3) \in R) \rightarrow (s_1, s_3) \in R$

Equivalence and Congruence Relations

- S is a set, and R is a *binary relation* over S
- R is an *equivalence relation* if:
  - Reflexivity: $\forall s_1 \in S.(s_1, s_1) \in R$
  - Symmetry: $\forall s_1, s_2 \in S.(s_1, s_2) \in R$
  - Transitivity: $\forall s_1, s_2, s_3 \in S.((s_1, s_2) \in R \wedge (s_2, s_3) \in R) \rightarrow (s_1, s_3) \in R$
- R is a *congruence relation* if:
  - R is an equivalence relation, and
  - for every the n-ary functions f:

$$\forall s_1, \dots, s_n, t_1, \dots, t_n \in R. \bigwedge_{i \in [1,n]} (s_i, t_i) \in R \rightarrow (f(s_1, \dots, s_n), f(t_1, \dots, t_n)) \in R$$

The T_E axioms express a congruence relation between terms

Equivalence and Congruence Classes

- $[s]_R$ is an equivalence (resp. congruence) class under the equivalence (resp. congruence) relation R:

$$[s]_R := \{s' \in S \mid (s, s') \in R\}$$

Example:

$$\phi := f(a, b) = a \wedge \neg(f(f(a, b), b) = a)$$

$$S = \{\text{set of sub-terms of } \phi\} = \{a, b, f(a, b), f(f(a, b), b)\}$$

$$[a]_= := \{f(a, b), a, f(f(a, b), b)\}$$

Equivalence and Congruence Classes

- $[s]_R$ is an equivalence (resp. congruence) class under the equivalence (resp. congruence) relation R:

$$[s]_R := \{s' \in S \mid (s, s') \in R\}$$

$$\begin{aligned} \text{Example:} \quad & \phi := f(a,b) = a \land \neg(f(f(a,b),b) = a) \\ & S = \{\text{set of sub-terms of } \phi\} = \{a, b, f(a,b), f(f(a,b),b)\} \\ & [a]_= := \{f(a,b), a, f(f(a,b),b)\} \end{aligned}$$

- A *partition* P of the set S is $P \subseteq 2^S$ such that:
  - $\bigcup_{S' \in P} S' = S$ and $\forall S_1, S_2 \in P.(S_1 \neq s_2 \rightarrow S_1 \cap S_2 = \emptyset)$

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

Equivalence and Congruence Classes

- $[s]_R$ is an equivalence (resp. congruence) class under the equivalence (resp. congruence) relation R:

$$[s]_R := \{s' \in S \mid (s, s') \in R\}$$

Example:
$$\begin{aligned} \phi &:= f(a,b) = a \wedge \neg(f(f(a,b),b) = a) \\ S &= \{\text{set of sub-terms of } \phi\} = \{a, b, f(a,b), f(f(a,b),b)\} \\ [a]_= &:= \{f(a,b), a, f(f(a,b),b)\} \end{aligned}$$

- A *partition* P of the set S is $P \subseteq 2^S$ such that:
  - $\bigcup_{S' \in P} S' = S$ and $\forall S_1, S_2 \in P.(S_1 \neq s_2 \rightarrow S_1 \cap S_2 = \emptyset)$

Example: $\{\{a\}, \{b\}, \{f(a,b)\}, \{f(f(a,b),b)\}\}$

- The *quotient* S/R is the partition of S formed by the equivalence classes of S under R:

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

Example: $\{[a]_=, [b]_=\}$

Congruence Closure

- R_1 *refines* R_2 ($R_1 \preceq R_1$) if:

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

Congruence Closure

- R_1 *refines* R_2 ($R_1 \preceq R_1$) if:

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

- R^C is the *congruence closure* for the congruence relation R if
  - $R \preceq R^C$
  - for all R' such that $R \preceq R'$, we either have $R' = R^C$ or $R^C \preceq R$. R^C is the "smallest" congruence relation.

Congruence Closure

- R_1 *refines* R_2 ($R_1 \preceq R_1$) if:

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

- R^C is the *congruence closure* for the congruence relation R if
  - $R \preceq R^C$
  - for all R' such that $R \preceq R'$, we either have $R' = R^C$ or $R^C \preceq R$. R^C is the "smallest" congruence relation.

Computing the congruence closure:

- Start with the finest congruence relation (every element in its own congruence class)

Congruence Closure

- R_1 *refines* R_2 ($R_1 \preceq R_1$) if:

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

- R^C is the *congruence closure* for the congruence relation R if
  - $R \preceq R^C$
  - for all R' such that $R \preceq R'$, we either have $R' = R^C$ or $R^C \preceq R$.
    R^C is the "smallest" congruence relation.

Computing the congruence closure:

- Start with the finest congruence relation (every element in its own congruence class)
- For each equality $s_i = t_i$, merge the congruence classes for $[s_i]_R$ and $[t_i]_R$:
  - First union the elements of $[s_i]_R$ and $[t_i]_R$, to define the new class $[s_i]_R$
  - Then, propagate the congruences that arise between the new pairs of elements in the union

$\mathcal{T}_E$-Satisfiability

$$\phi := [s_1 = t_1, \dots s_m = t_m, \neg(s_{m+1} = t_{m+1}), \dots \neg(s_n = t_n)]$$

1. Construct the congruence closure of $\{s_1 = t_1, \dots s_m = t_m\}$, over the sub-terms of ϕ.
2. If any of the atoms in the inequalities $s_i = t_i$, for $i \in [m+1, n]$, is such that s_i and t_i are in the same congruence class, then returns unsatisfiable
3. Otherwise, return satisfiable

Example - congruence closure computation

$$\phi := f(a, b) = a \land \neg f(f(a, b), b) = a$$

Example - congruence closure computation

$$\phi := f(a, b) = a \wedge \neg f(f(a, b), b) = a$$

1. Finest partition of sub-terms:

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

Example - congruence closure computation

$$\phi := f(a,b) = a \land \neg f(f(a,b),b) = a$$

1. Finest partition of sub-terms:

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

2. From the equality $f(a,b) = a$, we *merge* $\{a\}$ and $\{f(a,b)\}$

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

Example - congruence closure computation

$$\phi := f(a, b) = a \land \neg f(f(a, b), b) = a$$

1. Finest partition of sub-terms:

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

2. From the equality $f(a, b) = a$, we *merge* $\{a\}$ and $\{f(a, b)\}$

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

3. Apply congruence - $f(a, b) = f(f(a, b), b)$:

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

This partition is the congruence closure.

Example - congruence closure computation

$$\phi := f(a,b) = a \wedge \neg f(f(a,b),b) = a$$

1. Finest partition of sub-terms:

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

2. From the equality $f(a,b) = a$, we *merge* $\{a\}$ and $\{f(a,b)\}$

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

3. Apply congruence - $f(a,b) = f(f(a,b),b)$:

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

This partition is the congruence closure.

Is ϕ satisfiable? No, since ϕ requires $\neg f(f(a,b),b) = a$, but $f(f(a,b),b)$ and a are in the same congruence class.

Example - congruence closure computation (2)

$$\phi := f^3(a) = a \wedge f(f(f^3(a))) = a \wedge \neg f(a) = a$$

Example - congruence closure computation (2)

$$\phi := f^3(a) = a \wedge f(f(f^3(a))) = a \wedge \neg f(a) = a$$

1. Finest partition of sub-terms (notation: $f^0(v) = v$ and $f^k = f(f^{k-1}(v)))$:

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

Example - congruence closure computation (2)

$$\phi := f^3(a) = a \wedge f(f(f^3(a))) = a \wedge \neg f(a) = a$$

1. Finest partition of sub-terms (notation: $f^0(v) = v$ and $f^k = f(f^{k-1}(v)))$:

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

2. From the equality $f^3(a) = a$, we merge $\{a\}$ and $f^3(a)$:

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

Example - congruence closure computation (2)

$$\phi := f^3(a) = a \wedge f(f(f^3(a))) = a \wedge \neg f(a) = a$$

1. Finest partition of sub-terms (notation: $f^0(v) = v$ and $f^k = f(f^{k-1}(v)))$:

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

2. From the equality $f^3(a) = a$, we merge $\{a\}$ and $f^3(a)$:

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

3. Apply congruence - $f(a) = f(f^3(a)) = f^4(a)$:

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

Example - congruence closure computation (2)

$$\phi := f^3(a) = a \wedge f(f(f^3(a))) = a \wedge \neg f(a) = a$$

1. Finest partition of sub-terms (notation: $f^0(v) = v$ and $f^k = f(f^{k-1}(v)))$:

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

2. From the equality $f^3(a) = a$, we merge $\{a\}$ and $f^3(a)$:

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

3. Apply congruence - $f(a) = f(f^3(a)) = f^4(a)$:

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

4. From congruence we have $f(f(a)) = f(f^4(a)) = f^5(a)$:

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

Example - congruence closure computation (2)

$$\phi := f^3(a) = a \wedge f(f(f^3(a))) = a \wedge \neg f(a) = a$$

1. Finest partition of sub-terms (notation: $f^0(v) = v$ and $f^k = f(f^{k-1}(v))$):

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

2. From the equality $f^3(a) = a$, we merge $\{a\}$ and $f^3(a)$:

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

3. Apply congruence - $f(a) = f(f^3(a)) = f^4(a)$:

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

4. From congruence we have $f(f(a)) = f(f^4(a)) = f^5(a)$:

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

5. From the equality $f^5(a) = a$:

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

Example - congruence closure computation (2)

$$\phi := f^3(a) = a \wedge f(f(f^3(a))) = a \wedge \neg f(a) = a$$

1. Finest partition of sub-terms (notation: $f^0(v) = v$ and $f^k = f(f^{k-1}(v)))$:

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

2. From the equality $f^3(a) = a$, we merge $\{a\}$ and $f^3(a)$:

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

3. Apply congruence - $f(a) = f(f^3(a)) = f^4(a)$:

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

4. From congruence we have $f(f(a)) = f(f^4(a)) = f^5(a)$:

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

5. From the equality $f^5(a) = a$:

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

6. Apply congruence - $f(f^2(a) = f(f^3(a)) = f^4 a$:

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

Example - congruence closure computation (2)

$$\phi := f^3(a) = a \wedge f(f(f^3(a))) = a \wedge \neg f(a) = a$$

We have the congruence closure:

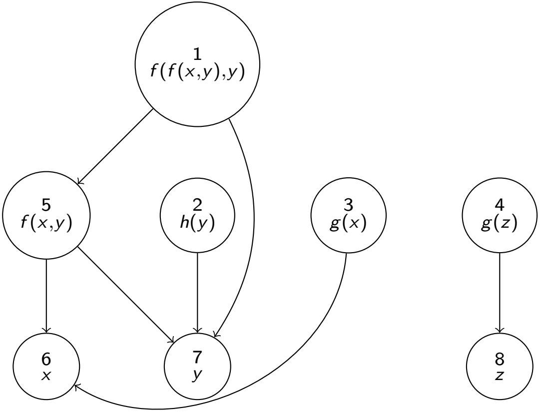
$$\{\{a, f(a), f^2(a), f^3(a), f^4(a), f^5(a)\}\}$$

We have $\neg f(a) = a$, but a and $f(a)$ are in the same congruence class, so ϕ is unsatisfiable!

1. A Decision Procedure for the Theory of Equality
   - T_E-satisfiability
   - Deciding T_E via Congruence Closure
   - An algorithm to computing congruence closure

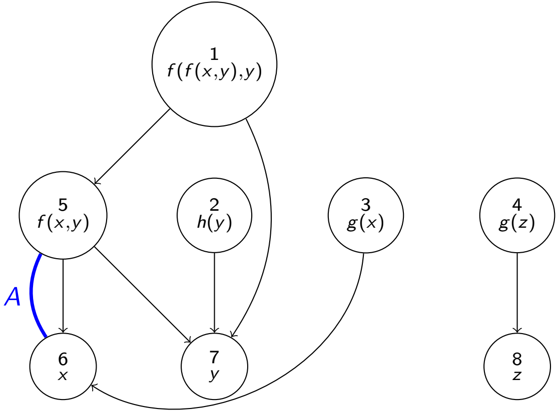
Congruence Closure via DAG

$$\phi := f(x, y) = x \wedge h(y) = g(x) \wedge f(f(x, y), y) = z \wedge \neg g(x) = g(z)$$



Congruence Closure via DAG

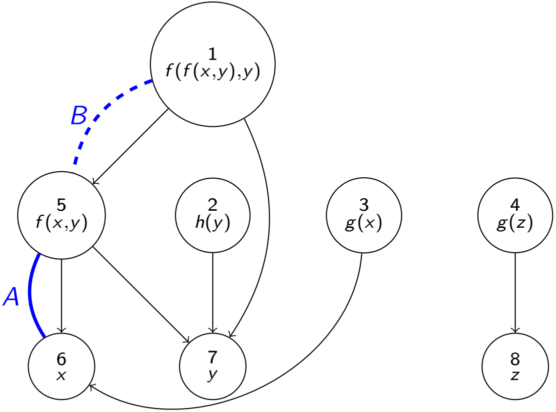
$$\phi := f(x,y) = x \wedge h(y) = g(x) \wedge f(f(x,y),y) = z \wedge \neg g(x) = g(z)$$



A. merge(5,5)

Congruence Closure via DAG

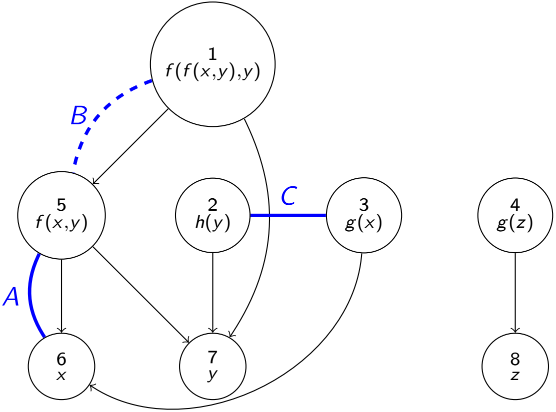
$$\phi := f(x,y) = x \wedge h(y) = g(x) \wedge f(f(x,y),y) = z \wedge \neg g(x) = g(z)$$



A. merge(5,5)
B. merge(1,5)

Congruence Closure via DAG

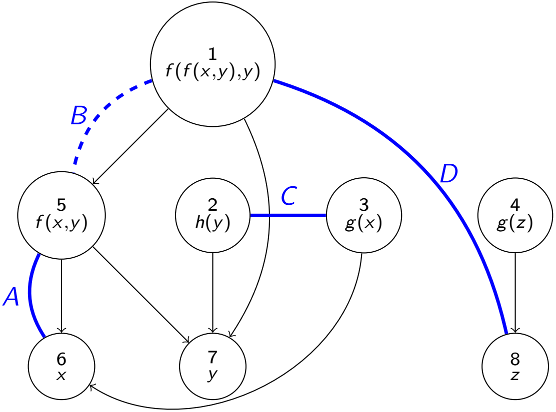
$$\phi := f(x,y) = x \land h(y) = g(x) \land f(f(x,y),y) = z \land \neg g(x) = g(z)$$



A. merge(5,5)
B. merge(1,5)
C. merge(2,3)

Congruence Closure via DAG

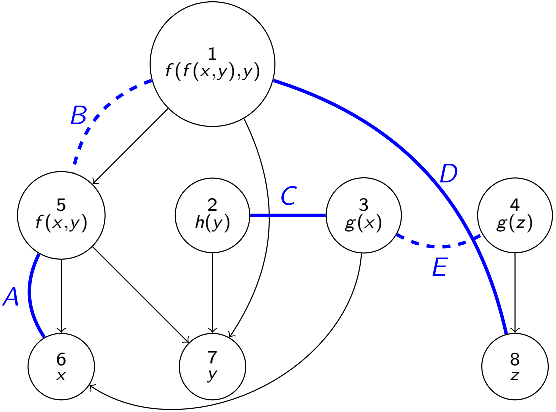
$$\phi := f(x, y) = x \wedge h(y) = g(x) \wedge f(f(x, y), y) = z \wedge \neg g(x) = g(z)$$



A. merge(5,5)
B. merge(1,5)
C. merge(2,3)
D. merge(1,8)

Congruence Closure via DAG

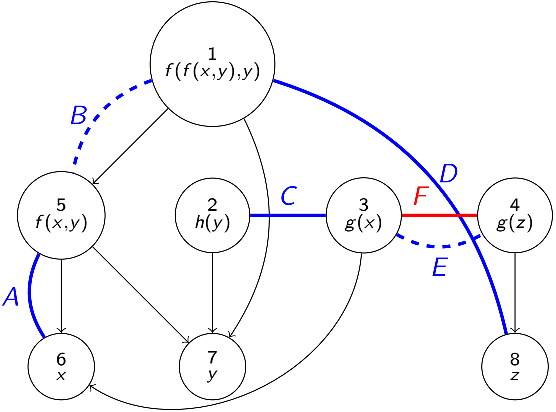
$$\phi := f(x,y) = x \wedge h(y) = g(x) \wedge f(f(x,y),y) = z \wedge \neg g(x) = g(z)$$



A. merge(5,5)
B. merge(1,5)
C. merge(2,3)
D. merge(1,8)
E. merge(2,4)

Congruence Closure via DAG

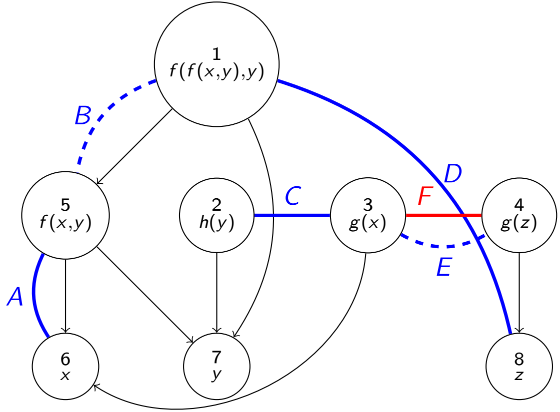
$$\phi := f(x,y) = x \land h(y) = g(x) \land f(f(x,y),y) = z \land \neg g(x) = g(z)$$



A. merge(5,5)
B. merge(1,5)
C. merge(2,3)
D. merge(1,8)
E. merge(2,4)
F. Conflict

Congruence Closure via DAG

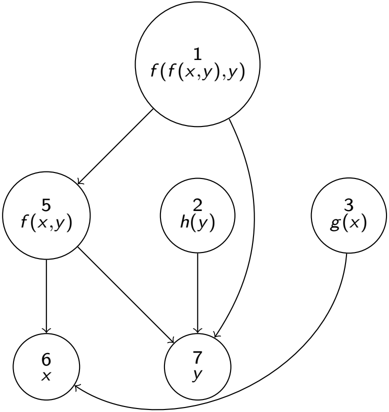
$$\phi := f(x, y) = x \wedge h(y) = g(x) \wedge f(f(x, y), y) = z \wedge \neg g(x) = g(z)$$



A. merge(5,5)
B. merge(1,5)
C. merge(2,3)
D. merge(1,8)
E. merge(2,4)
F. Conflict

This is what you will implement in the tutorial

A DAG data structure congruence closure




```
Node {
  id : integer;
  // id of the class representative
  find : integer;
  // name of the node
  name : string;
  // ids of the children
  args : list of integers;
  // ids of the class parents
  parents : list of integers;
}
```

Node for $f(x, y)$

```
Node {
  id = 5
  find = 5
  name : f;
  args : [6,7];
  parents : [1];
}
```

UNION/FIND functions

```
procedure NODE(i)
```

Returns the node that has the id *i*

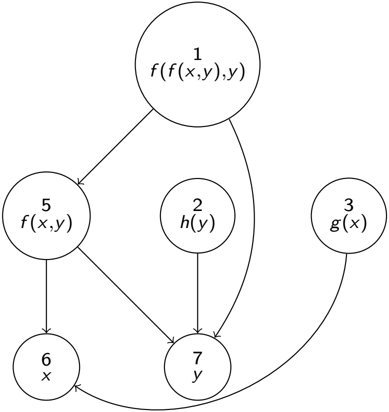
```
procedure FIND(i)
  n = NODE(i)
  if n.find = i then
    return i
  else
    return FIND(n.find)
```

Returns the id of the equivalence class for the node i.

```
procedure UNION(i1,i2)
  n1 = NODE(i1)
  n2 = NODE(i2)
  n1.find = n2.find
  n2.parents =
    n1.parents ∪ n2.parents
  n1.parents = []
```

Computes the union of i1 and i2

UNION/FIND example




```
Node {
  id = 5
  find = 5
  name : f;
  args : [6,7];
  parents : [1];
}
```

```
Node {
  id = 6
  find = 6
  name : x;
  args : [];
  parents : [5];
}
```

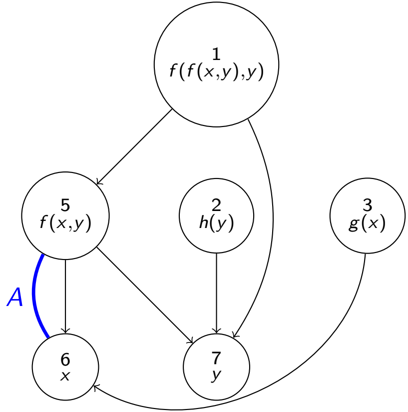
```
UNION(6,5)
```

```
Node {
  id = 5
  find = 6
  name : f;
  args : [6,7];
  parents : [];
}
```

```
Node {
  id = 6
  find = 6
  name : x;
  args : [];
  parents : [5,1];
}
```

```
FIND(5) now returns node 6
```

UNION/FIND example




```
Node {
  id = 5
  find = 5
  name : f;
  args : [6,7];
  parents : [1];
}
```

```
Node {
  id = 6
  find = 6
  name : x;
  args : [];
  parents : [5];
}
```

```
UNION(6,5)
```

```
Node {
  id = 5
  find = 6
  name : f;
  args : [6,7];
  parents : [];
}
```

```
Node {
  id = 6
  find = 6
  name : x;
  args : [];
  parents : [5,1];
}
```

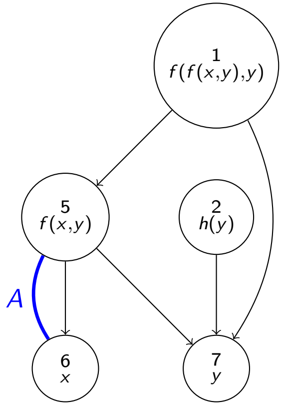
```
FIND(5) now returns node 6
```

CONGRUENT function

Returns true if the node in i1 and in i2 are congruent

procedure CONGRUENT(i1,i2)
  n1 = NODE(i1)
  n2 = NODE(i2)
  if n1.name $\neq$ n2.name **then**
    return False
  else if len(n1.args) $\neq$ len(n2.args) **then**
    return False
  else if len(n1.args) $\neq$ len(n2.args) **then**
    return $\forall i \in \{1, \ldots, len(n1.args)\}$.
      FIND(n1.args[i]) = FIND(n2.args[i])

CONGRUENT example




```
n5 := {              n6 := {              n1 := {
  id = 5               id = 6               id = 1
  find = 6             find = 6             find = 1
  name : f;            name : x;            name : f;
  args : [6,7];        args : [];           args : [5,7];
  parents : [1];       parents : [5,1];     parents : [];
}                    }                    }
```

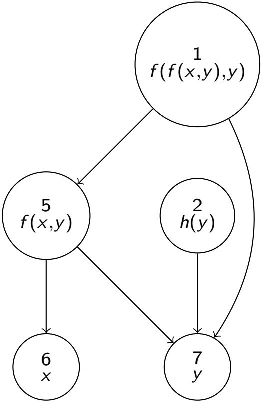
```
Execution of CONGRUENT(1,5)
- n1 = NODE(1)
- n5 = NODE(5)
- n1.name == f == n5.name
- len(n1.args) == len(n2.args)
  - FIND(6) == 6 == FIND(5)
  - FIND(7) == 7 == FIND(7)

So node 1 and 5 are congruent.
```

MERGE function

Merge the congruent classes of the node i1 and node i2

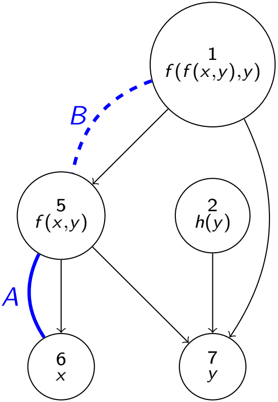
```
procedure MERGE(i1,i2)
  if FIND(i1) ≠ FIND(i2) then
    P1 = NODE(FIND(i1)).parents
    P2 = NODE(FIND(i2)).parents
    UNION(i1, i2)
    for t1,t2 ∈ P1 × P2 do
      if FIND(t1) ≠ FIND(t2) and CONGRUENT(t1,t2) then
        MERGE(t1,t2)
```

MERGE example




```
n5 := {                n6 := {                n1 := {
  id = 5                 id = 6                 id = 1
  find = 5               find = 6               find = 1
  name : f;              name : x;              name : f;
  args : [6,7];          args : [];             args : [5,7];
  parents : [1];         parents : [5];         parents : [];
}                      }                      }

Execution of MERGE(5,6)
  - FIND(5) != FIND(6)
  - P1 = [1]
  - P2 = [5]
  - UNION(5,6) - example we saw earlier
  - P1 x P2 = [(1,5)]
    - FIND(1) != FIND(5)
    - CONGRUENT(1,5)
    => So we recursively merge 1 and 5: MERGE(1,5)
    => 1,5,6 are in the same congruence class
```

MERGE example




```
n5 := {
  id = 5
  find = 5
  name : f;
  args : [6,7];
  parents : [1];
}
```

```
n6 := {
  id = 6
  find = 6
  name : x;
  args : [];
  parents : [5];
}
```

```
n1 := {
  id = 1
  find = 1
  name : f;
  args : [5,7];
  parents : [];
}
```

```
Execution of MERGE(5,6)
  - FIND(5) != FIND(6)
  - P1 = [1]
  - P2 = [5]
  - UNION(5,6) - example we saw earlier
  - P1 x P2 = [(1,5)]
    - FIND(1) != FIND(5)
    - CONGRUENT(1,5)
    => So we recursively merge 1 and 5: MERGE(1,5)
    => 1,5,6 are in the same congruence class
```

Revisiting the decision procedure using the union-find algorithm

$$\phi := [s_1 = t_1, \dots s_m = t_m, \neg(s_{m+1} = t_{m+1}), \dots \neg(s_n = t_n)]$$

1. Construct the DAG G
2. For all $(s_i, t_i) \in [1, m]$ call MERGE(s_i, t_i) – (in practice the id of s_i and t_i)
3. If for any inequalities $(s_i, t_i) \in [m + 1, n]$:
   - $FIND(s_i) = FIND(t_i)$, then return unsatisfiable
4. Otherwise return satisfiable.

Revisiting the decision procedure using the union-find algorithm

$$\phi := [s_1 = t_1, \dots s_m = t_m, \neg(s_{m+1} = t_{m+1}), \dots \neg(s_n = t_n)]$$

1. Construct the DAG G
2. For all $(s_i, t_i) \in [1, m]$ call MERGE(s_i, t_i) – (in practice the id of s_i and t_i)
3. If for any inequalities $(s_i, t_i) \in [m + 1, n]$:
   - $FIND(s_i) = FIND(t_i)$, then return unsatisfiable
4. Otherwise return satisfiable.

Properties:

- The algorithm is sound and complete for quantifier-free conjunctive Σ_E-formulas.
- This algorithm runs in time $O(e^2)$ for $O(n)$ merges
  More efficient algorithms exists that run in $O(e \log e)$ for $O(n)$ merges (e.g., see [Detlefs et al., 2005])

To sum up

What did we see today?

- We can decide the $\mathcal{T}_E$-satisfiability of a conjunctive formula ϕ computing the congruence closure:
  - We use a graph (UNION/FIND data structures) to represent and merge congruence classes
  - We obtain the congruence classes from the equalities in ϕ
  - Once we ave the congruence classes, we check for inconsistencies with the inequalities of ϕ
- The computation is efficient (there are some optimization that can run in polynomial time ($O(n \log n)$))

Next week: how to decide consistency for the theory of linear arithmetic

References I

Barrett, C. W., Sebastiani, R., Seshia, S. A., and Tinelli, C. (2009).
Satisfiability modulo theories.
In Biere, A., Heule, M., van Maaren, H., and Walsh, T., editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 825–885. IOS Press.

Bradley, A. R. and Manna, Z. (2007).
The calculus of computation - decision procedures with applications to verification.
Springer.

Detlefs, D., Nelson, G., and Saxe, J. B. (2005).
Simplify: a theorem prover for program checking.
J. ACM, 52(3):365–473.