Axioms of $T_A$

To define “intended semantics of array read and write”, we need to provide axioms of $T_A$.

Axioms of $T_A$ include reflexivity, symmetry, and transitivity.

In addition, they include axioms unique to arrays:

1. $\forall a, i, j. \ i = j \rightarrow a[i] = a[j]$ (array congruence)
2. $\forall a, v, i, j. \ i = j \rightarrow a(i \triangledown v)[j] = v$ (read-over-write 1)
3. $\forall a, v, i, j. \ i \neq j \rightarrow a(i \triangledown v)[j] = a[j]$ (read-over-write 2)

Example

Is the following $T_A$ formula valid?

$F : a[i] = e \rightarrow (\forall j. \ a(i \triangledown e)[j] = a[j])$

Yes! For any $j \neq i$, $a(i \triangledown e)[j] = a[j]$ according to read-over-write 2 axiom. For any $j = i$, old value of $j$ was already $e$, so its value didn’t change.

Let’s prove its validity using the semantic argument method.

Assume there exists a model $M$ and variable assignment $\sigma$ that does not satisfy $F$ and derive contradiction.
Combination of Theories

- So far, we only talked about individual first-order theories.
- Examples: $T_w$, $T_{PA}$, $T_Z$, $T_A$, ...
- But in many applications, we need combined reasoning about several of these theories
- Example: The formula $f(x) + 3 = y$ isn’t a well-formed formula in any individual theory, but belongs to combined theory $T_Z \cup T_w$

Decision Procedures for Combined Theories

- Given two theories $T_1$ and $T_2$ that have the $=$ predicate, we define a combined theory $T_1 \cup T_2$
- Signature of $T_1 \cup T_2$: $\Sigma_1 \cup \Sigma_2$
- Axioms of $T_1 \cup T_2$: $A_1 \cup A_2$
- Is this a well-formed $T_w \cup T_Z$ formula? Yes
  \[
  1 \leq x \land x \leq 2 \land f(x) \neq f(1) \land f(x) \neq f(2)
  \]
- Is this formula satisfiable according to axioms $A_Z \cup A_w$? No

Decidability Results for $T_A$

- The full theory of arrays if not decidable.
- The quantifier-free fragment of $T_A$ is decidable.
- Unfortunately, the quantifier-free fragment not sufficiently expressive in many contexts
- Thus, people have studied other richer fragments that are still decidable.
- Example: array property fragment (disallows nested arrays, restrictions on where quantified variables can occur)

Combined Theories

- Given two theories $T_1$ and $T_2$ that have the $=$ predicate, we define a combined theory $T_1 \cup T_2$
- Signature of $T_1 \cup T_2$: $\Sigma_1 \cup \Sigma_2$
- Axioms of $T_1 \cup T_2$: $A_1 \cup A_2$
- Is this a well-formed $T_w \cup T_Z$ formula? Yes
  \[
  1 \leq x \land x \leq 2 \land f(x) \neq f(1) \land f(x) \neq f(2)
  \]
- Is this formula satisfiable according to axioms $A_Z \cup A_w$? No

Road Map

- Talk about decision procedures for widely-used first order-theories: equality, LRA, Presburger arithmetic
- Initially, we’ll only focus on decision procedures for formulas without disjunctions
- Ok because we can always convert to DNF to deal with disjunctions – just not very efficient!
- Later in the course, we’ll see about how to handle disjunctions much more efficiently
- Rest of today’s lecture: Decision procedure for qff theory of equality
Review

- **Previous lecture:** talked about signature and axioms of $T_=$
  \[ \Sigma_\_\_ : \{=, a, b, c, ..., f, g, h, ..., p, q, r, ... \} \]

- **Axioms:**
  1. $\forall x, x = x$ (reflexivity)
  2. $\forall x, y, x = y \rightarrow y = x$ (symmetry)
  3. $\forall x, y, z, x = y \land y = z \rightarrow x = z$ (transitivity)
  4. $\forall x_1, ..., x_n, y_1, ..., y_n, \land \_ = = = y_i = y_i$ \[ f(x_1, ..., x_n) = f(y_1, ..., y_n) \] (congruence)
  5. for each positive integer $n$ and $n$-ary predicate $p$,
    $\forall x_1, ..., x_n, y_1, ..., y_n, \land \_ = = = y_i = y_i$ \[ (p(x_1, ..., x_n) \leftrightarrow p(y_1, ..., y_n)) \] (equivalence)

Overview

- **Today:** look at decision procedures for deciding satisfiability in the quantifier-free fragment of $T_=$

  - However, our decision procedure has two “restrictions”:
    - formulas consist of conjunctions of literals
    - we’ll allow functions, but no predicates

  - However, these “restrictions” are not real restrictions
  - For formulas with disjunctions, can convert to DNF and check each clause separately (will consider efficient methods later)

  - Furthermore, any formula containing predicates can be converted to equisatisfiable formula containing only functions!

Eliminating Predicates

- Simple transformation yields equisatisfiable formula with only functions
- **The trick:** For each relation constant $p$:
  1. introduce a fresh function constant $f_p$
  2. rewrite $p(x_1, ..., x_n)$ as $f_p(x_1, ..., x_n) = t$
    where $t$ is a fresh object constant
- **Example:** How do we transform $x = y \rightarrow (p(x) \leftrightarrow p(y))$ to equisatisfiable formula? $x = y \rightarrow (f_p(x) = t \leftrightarrow f_p(y) = t)$

Examples

- Let’s consider some examples
  - Is the formula $x = y \land f(x) \neq f(y)$ sat, unsat, valid? **unsat**
  - What about $x \neq y \land f(x) = f(y)$? **sat**
  - What about $x = g(y, z) \rightarrow f(x) = f(g(y, z))$? **valid**
  - What about $f(a) = a \land f(f(a)) \neq a$? **unsat**

Example, cont.

- What about $f(f(f(a))) = a \land f(f(f(f(f(a)))))) = a \land f(a) \neq a$? **unsat**
- **Reasoning:** Substitute $a$ for $f(f(f(a)))$ in second equality, this yields: $f(f(a)) = a$
- Since $f(f(a)) = a$, by congruence $f(f(f(a))) = f(a)$
- By first equality, we have $f(a) = a \Rightarrow$ contradiction!
Equivalence Relations

- Decision procedure for theory of equality known as congruence closure algorithm
- But need to understand what congruence closure is first ⇒ new terminology and concepts
- A binary relation \( R \) over a set \( S \) is an equivalence relation if
  1. reflexive: \( \forall s \in S.\ sRs \)
  2. symmetric: \( \forall s_1, s_2 \in S.\ s_1Rs_2 \iff s_2Rs_1 \)
  3. transitive: \( \forall s_1, s_2, s_3 \in S.\ s_1Rs_2 \land s_2Rs_3 \implies s_1Rs_3 \).

Equivalence Closure

- The equivalence closure \( RE \) of a binary relation \( R \) over \( S \) is the equivalence relation such that:
  1. \( R \) refines \( RE \), i.e., \( R \prec RE \);
  2. for all other equivalence relations \( R' \) s.t. \( R \prec R' \), either \( R' = RE \) or \( R' \prec RE \).
- Thus, \( RE \) is the smallest equivalence relation that includes \( R \).

Equivalence and Congruence Relations

- Equality predicate \( = \) is equivalence relation over real numbers
- The relation “has same birthday as” is an equivalence relation over set of people
- The relation \( \equiv \) is equivalence relation over \( \mathbb{Z} \)
- A relation \( R \) is congruence relation over set \( S \) if it is an equivalence relation and for every \( n \)-ary function \( f \):
  \[
  \forall \bar{s}, \bar{t}. \bigwedge_{i=1}^{n} s_iRt_i \implies f(\bar{s}) \approx f(\bar{t}).
  \]

Equivalence and Congruence Classes

- For a given equivalence relation over \( S \), every member of \( S \) belongs to an equivalence class
- The equivalence class of \( s \in S \) under \( R \) is the set:
  \[
  [s]_R \overset{\text{def}}{=} \{s' \in S : sRs'\}.
  \]
- If \( R \) is a congruence relation, then this set is called congruence class
- Example: What is the equivalence class of 1 under \( \equiv \)? odd numbers
- What is the equivalence class of 6 under \( \equiv \)? multiples of 3

Relation Refinements

- A binary relation \( R_1 \) is a refinement of another binary relation \( R_2 \), written \( R_1 \prec R_2 \), if
  \[
  \forall s_1, s_2 \in S.\ s_1R_1s_2 \implies s_1R_2s_2.
  \]
- Example 1: Consider set \( S = \{a, b\} \) and relations \( R_1 = \{(a, a), (b, b)\} \) and \( R_2 = \{(a, b), (b, b)\} \)
  - Do either of these hold? \( R_1 \prec R_2 \lor R_2 \prec R_1 \) \( \iff \) \( R_1 \prec R_2 \)
- Example 2: Consider set \( \mathbb{Z} \) and the relations:
  \( R_1 : \{xR_1y : x \mod 2 = y \mod 2\} \)
  \( R_2 : \{xR_2y : x \mod 4 = y \mod 4\} \)
  - What is the refinement relationship between \( R_1 \) and \( R_2 \)? \( R_2 \prec R_1 \)

Equivalence Closure Example

- Consider set \( S = \{a, b, c, d\} \) and binary relation \( R : \{(a, b), (b, c), (d, d)\} \)
  - Is \( R \) an equivalence relation? \( \neg \)
- We want to compute the equivalence closure \( RE \), i.e., smallest equivalence relation including \( R \)
  - Thus, \( RE \) needs to include all tuples in \( R \) and must obey reflexivity, symmetry, and transitivity.
Equivalence Closure Example, cont

- $R = \{(a, b), (b, c), (d, d)\}$

- Since $R^E$ must include $R$, which elements are in $R^E$?
  $\langle a, b \rangle, \langle b, c \rangle, \langle d, d \rangle$

- Since $R^E$ equivalence relation, it must obey reflexivity. What other elements in $R^E$ due to reflexivity? $\langle a, a \rangle, \langle b, b \rangle, \langle c, c \rangle$

- What elements in $R^E$ due to symmetry? $\langle b, a \rangle, \langle c, b \rangle$

- What elements in $R^E$ due to transitivity? $\langle a, c \rangle, \langle c, a \rangle$

- What is $R^E$?
  $R^E = \{(a, b), (b, c), (d, d), (a, b), (b, c), (d, d), (b, a), (c, b), (c, a), (c, a)\}$

Congruence Closure

- Given a set $S$ and binary relation $R$, we also define congruence closure of $R$

- Congruence closure is similar to equivalence closure, but it is the smallest congruence relation that covers $R$

- Formally, the congruence closure $R^C$ of a binary relation $R$ over $S$ is the congruence relation such that:
  1. $R$ refines $R^C$, i.e. $R \triangleleft R^C$;
  2. for all other congruence relations $R'$ s.t. $R \triangleleft R'$, either $R' = R^C$ or $R^C \triangleleft R'$

Satisfiability using Congruence Relations

- We can now define satisfiability of a $\Sigma_m$ formula in terms of congruence closure over subterm set

- Consider $\Sigma_m$ formula $F$:
  $F : s_1 = t_1 \land \ldots s_m = t_m \land s_{m+1} \neq t_{m+1} \land \ldots s_n \neq t_n$

- Theorem: $F$ is satisfiable iff there exists a congruence relation $\sim$ over the subterm set $S_F$ of $F$ such that:
  1. For each $i$ in $[1, m]$, $s_i \sim t_i$
  2. For each $i$ in $[m+1, n]$, $s_i \neq t_i$

Congruence Closure Algorithm

- The decision procedure for $T_e$ computes congruence closure of equality over the subterm set of formula

- Subterm set $S_F$ of $F$ is the set of all subterms of $F$

- Example: Consider formula $F : f(a, b) = a \land f(f(a, b), b) \neq a$

- What is $S_F$? $\{a, b, f(a, b), f(f(a, b), b)\}$

Equivalence Closure Example, cont

- $R = \{(a, b), (b, c), (d, d)\}$

- $R_E = \{(a, b), (b, c), (d, d), (a, b), (b, c), (d, d), (b, a), (c, b), (c, a), (c, a)\}$

- Consider relation
  $R' = R_E \cup \{(c, d), (d, c), (d, d), (d, a), (a, d)\}$

- $R'$ is also an equivalence relation and covers $R$

- Is $R'$ also an equivalence closure of $R$? No!

Congruence Closure Algorithm: Basic Idea

Congruence closure algorithm decide satisfiability of

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

1. Construct the congruence closure $\sim$ of
   $$\{s_1 = t_1, \ldots, s_m = t_m\}$$
   over the subterm set $S_F$.
2. If $s_i \sim t_i$ for any $i$ in $[m+1, n]$, $F$ is unsatisfiable
3. Otherwise, $F$ is satisfiable
Example

- Consider the formula $F : f(a, b) = a \land f(f(a, b), b) \neq a$
- We’ll represent $\sim$ as a set of congruence classes, i.e., if $t_1$ and $t_2$ are in the same set, this means $t_1 \sim t_2$, otherwise $t_1 \not\sim t_2$
- First, construct subterm set $S_F$ and place each subterm in a separate set:
  \[
  \{\{a\}, \{b\}, \{f(a, b)\}, \{f(f(a, b), b)\}\}
  \]
- Because of equality $f(a, b) = a$, merge congruence classes of $f(a, b)$ and $a$:
  \[
  \{\{a, f(a, b)\}, \{b\}, \{f(f(a, b), b)\}\}
  \]

Example, cont

- Formula $F : f(a, b) = a \land f(f(a, b), b) \neq a$
- Current congruence classes:
  \[
  \{\{a, f(a, b)\}, \{b\}, \{f(f(a, b), b)\}\}
  \]
- Is $F$ satisfiable? No
- Since $a$ and $f(f(a, b), b)$ are in same congruence class, we have $a \sim f(f(a, b), b)$
- This contradicts $f(f(a, b), b) \neq a$!

Another Example

- Formula $F : f^3(a) = a \land f^5(a) = a \land f(a) \neq a$
- Current congruence classes:
  \[
  \{\{a, f^3(a)\}, \{f(a)\}, \{f^2(a)\}, \{f^4(a)\}, \{f^5(a)\}\}
  \]
- From $a = f^3(a)$, what can we infer using function congruence? $f(a) = f^4(a)$ and $f^2(a) = f^5(a)$
- Resulting congruence classes:
  \[
  \{\{a, f^3(a)\}, \{f(a), f^4(a)\}, \{f^2(a), f^5(a)\}\}
  \]
Another Example, cont

- Formula $F : f^3(a) = a \land f^3(a) = a \land f(a) \neq a$
- Current congruence classes:
  $\{(a, f(a), f^2(a), f^3(a), f^4(a), f^5(a))\}$
- Is the formula satisfiable? No
- Since $f(a)$ and $a$ are in the same congruence class, this contradicts $f(a) \neq a$

One More Example

- Consider formula $F : f(x) = f(y) \land x \neq y$
- What is the subterm set? $\{x, y, f(x), f(y)\}$
- Each subterm starts in its own congruence class:
  $\{(x, y, f(x), f(y))\}$
- Process equality $f(x) = f(y) \Rightarrow \{(x, y, f(x), f(y))\}$
- What new equalities can we infer from congruence? None!
- Is the formula satisfiable? Yes

How to Compute Congruence Closure

- So far, we described how to decide satisfiability using congruence closure
- But we haven’t discussed an algorithm for efficiently computing congruence closure
- Next: Efficient algorithm for computing congruence closure

Representing Subterms

- To compute congruence closure efficiently, we’ll represent the subterm set of the formula as a DAG
- Each node corresponds to a subterm and has unique id
- Edges point from function symbol to arguments
- Question: What subterm does node labeled 1 represent? $f(f(a, b), b)$

Representative of Congruence Class

- To compute congruence closure, we need to merge congruence classes
- To do this efficiently, each congruence class has a representative: When merging two classes, only need to update the representative
  - Thus, for a given subterm, we need to be able to find the representative of its class
  - Each subterm contains a pointer that eventually leads to the representative of its congruence class
  - In this example, $a, f(a, b), f(f(a, b), b)$ are in the same congruence class; $a$ is the representative

Parents of a Subterm

- In addition to efficiently finding representative, also need to efficiently find parents of terms
- Why? Because if $x_1 = y_1, \ldots, x_k = y_k$, function congruence implies $f(x) = f(y)$
- Thus, when each $x_i, y_i$ pair is in the same congruence class, need to merge congruence classes of their parents $f(x)$ and $f(y)$
- Thus, keep pointer from representative of congruence class to parents of all subterms in the congruence class
### Summary of Data Structure
- Represent subterms as a DAG
- Each node in the DAG corresponds to a subterm
- Each node stores its unique id, name of function or variable, and list of argument subterms
- Each node n has a `find` pointer field that leads to its representative
- The `find` field of a representative points to itself
- Each representative stores the set of `parents` for all subterms in that class
- If a term is not a representative, then its `parents` field is empty

### Finding Representative of Congruence Class
- Given a term t, we need to find representative for that term
- If t’s `find` field points to itself, then t is the representative of its congruence class
- Otherwise, we follow the chain of `find` references until we find a node t’ that points to itself
- In this case, t’ is t’s representative

### Merging Congruence Classes
- Using this data structure, how do we merge congruence classes of two terms t₁ and t₂?
- First find representatives of t₁ and t₂ as described earlier
- Want to make `Rep(t₂)` new representative for merged class
- Thus, change `find` field of `Rep(t₁)` to point to `Rep(t₂)`
- Update parents: add parent terms stored in `Rep(t₁)` to those of `Rep(t₂)`, and remove parents stored in `Rep(t₁)`

### Process Equalities
- How do we process an equality t₁ = t₂?
- Need to merge equivalence classes of t₁ and t₂
- Might potentially also need to merge t₁ and t₂’s parents due to function congruence
- Given parent p₁ of t₁ and p₂ of t₂, when do we merge p₁ and p₂’s congruence classes?
- If they have the same function name and all of their arguments are congruent (i.e., have same representative)

### Processing Equalities, cont
To process equality t₁ = t₂:
1. Find representatives of t₁ and t₂
2. Merge equivalence classes
3. Retrieve the set of parents P₁, P₂ stored in `Rep(t₁)`, `Rep(t₂)`
4. For each (p₁, p₂) ∈ P₁ × P₂, if p₁ and p₂ are congruent, process equality p₁ = p₂

**Observe:** Processing one equality creates new equalities, which in turn might generate other new equalities!

### Full Algorithm for Deciding Satisfiability
Algorithm to decide satisfiability of Tᵣ formula

\[ F : s₁ = t₁ ∧ ... sₘ = tₘ ∧ sₘ₊₁ ≠ tₘ₊₁ ∧ ... sₙ ≠ tₙ \]

1. Compute subterms and construct initial DAG (each node's representative is itself)
2. For each i ∈ [1, m], process equality sᵢ = tᵢ as described
3. For each i ∈ [m + 1, n], check if `Rep(sᵢ) = Rep(tᵢ)`
4. If there exists some i ∈ [m + 1, n] for which `Rep(sᵢ) = Rep(tᵢ)`, return UNSAT
5. If for all i, `Rep(sᵢ) ≠ Rep(tᵢ)`, return SAT
Example

- Consider formula $F : f(a, b) = a \land f(f(a, b), b) \neq a$
- Subterms: $a, b, f(a, b), f(f(a, b), b)$
  - Construct initial DAG
  - Process equality $f(a, b) = a$
  - Are parents $f(a, b)$ and $f(f(a, b), b)$ congruent? Yes
  - Yes, so process equality $f(a, b) = f(f(a, b), b)$
  - Formula unsatisfiable because $f(f(a, b), b)$ and $a$ have same representative!

Example II

- Consider formula: $F : f^3(a) = a \land f^5(a) = a \land f(a) \neq a$
  - Initial DAG:
  - Process equality $f^3(a) = a$:
  - Are parents congruent? Yes
  - Process equality $f^5(a) = f(a)$

Example II, cont

- After merging classes:
  - Are $f^4(a)$’s and $f(a)$’s parents congruent? Yes
  - Process equality $f^4(a) = f^2(a)$

Example II, cont

- Formula: $F : f^3(a) = a \land f^5(a) = a \land f(a) \neq a$
  - Process equality $f^3(a) = a$:
  - Now, parents $f^3(a)$ and $f(a)$ congruent; process equality $f^3(a) = f(a)$

Summary

- Congruence closure algorithm is used for determining satisfiability of $T_m$ formulas (without disjunction)
  - Our algorithm for computing congruence closures is called Union-Find, also used in other applications
  - Deciding conjunctive $T_m$ formulas is inexpensive: our algorithm is $O(e^2)$, but can be solved in
  - To decide satisfiability of formulas containing disjunctions, can either convert to DNF or use DPLL($T$) (more on this later)
  - Next lecture: Decision procedure for linear arithmetic over reals (Simplex algorithm)