Verification

Lecture 32

Martin Zimmermann

UNIVERSITÄT
DES
SAARLANDES

Plan for today

- Deductive verification
  - Congruence closure DAG method
  - Recursive Data Structures

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

The algorithm performs the following 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 unsatisfiable.
3. Otherwise, $\sim \models F$, so return satisfiable.

How do we actually 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 each term of 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

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

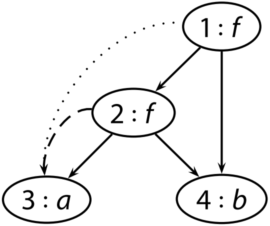
to form a new congruence relation $\sim_i$. To accomplish this merging,

- form the union of $[s_i]_{\sim_{i-1}}$ and $[t_i]_{\sim_{i-1}}$
- propagate any new congruences that arise within this union.

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

Directed Acyclic Graph (DAG)

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



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

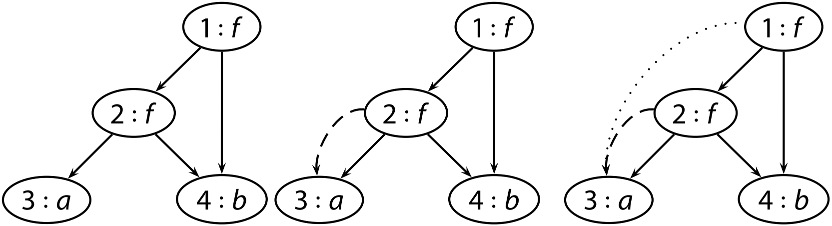
$f(a, b)$

$a \qquad b$

Efficient way for computing the congruence closure algorithm.

T_E-Satisfiability (Summary of idea)

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



Initial DAG

$f(a,b) = a \Rightarrow$
merge $f(a,b)$ a

$f(a,b) \sim a,\ b \sim b \Rightarrow$
$f(f(a,b),b) \sim f(a,b)$
merge $f(f(a,b),b)$
$f(a,b)$

_ _ explicit equation ..... by congruence

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

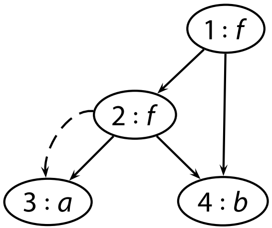
DAG representation

```
type node = {
    id           : id
                   node's unique identification number
    fn           : string
                   constant or function name
    args         : id list
                   list of function arguments
    mutable find : id
                   the representative of the congruence class
    mutable ccpar : id set
                   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
}
```

`ccpar` is initialized with the set containing the parents of the node (if it has any), `find` with the `id` of the node.

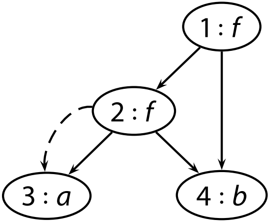
DAG Representation of node 2

$$
\begin{array}{llll}
\multicolumn{4}{l}{\texttt{type}\ \textbf{node}\ =\ \{} \\
\quad \texttt{id} & : & \textbf{id} & \ldots 2 \\
\quad \texttt{fn} & : & \textbf{string} & \ldots f \\
\quad \texttt{args} & : & \textbf{idlist} & \ldots [3, 4] \\
\quad \texttt{mutable find} & : & \textbf{id} & \ldots 3 \\
\quad \texttt{mutable ccpar} & : & \textbf{idset} & \ldots \emptyset \\
\}
\end{array}
$$



DAG Representation of node 3

$$
\begin{array}{llll}
\multicolumn{4}{l}{\texttt{type}\ \textbf{node}\ =\ \{} \\
\quad \texttt{id} & : & \textbf{id} & \ldots 3 \\
\quad \texttt{fn} & : & \textbf{string} & \ldots a \\
\quad \texttt{args} & : & \textbf{idlist} & \ldots [\,] \\
\quad \texttt{mutable find} & : & \textbf{id} & \ldots 3 \\
\quad \texttt{mutable ccpar} & : & \textbf{idset} & \ldots \{1, 2\} \\
\}
\end{array}
$$

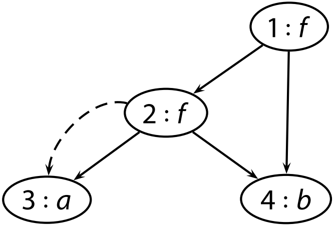


The Implementation

find function

returns the representative of node's congruence class

```
let rec find i =
  let n = node i in
  if n.find = i then i else find n.find
```



Example: find 2 = 3

find 3 = 3

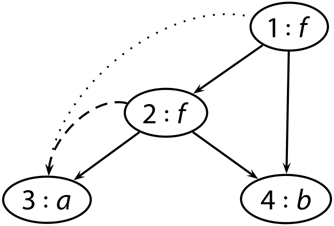
3 is the representative of 2.

union function

$$
\begin{array}{l}
\texttt{let}\ \text{union}\ i_1\ i_2 = \\
\quad \texttt{let}\ n_1 = \text{node}\ (\text{find}\ i_1)\ \texttt{in} \\
\quad \texttt{let}\ n_2 = \text{node}\ (\text{find}\ i_2)\ \texttt{in} \\
\quad n_1.\texttt{find} \leftarrow n_2.\texttt{find}; \\
\quad n_2.\texttt{ccpar} \leftarrow n_1.\texttt{ccpar} \cup n_2.\texttt{ccpar}; \\
\quad n_1.\texttt{ccpar} \leftarrow \emptyset
\end{array}
$$

n_2 is the representative of the union class

Example



union 1 2 $\qquad n_1 = 1 \qquad n_2 = 3$

$1.\texttt{find} \leftarrow 3$

$3.\texttt{ccpar} \leftarrow \{1, 2\}$

$1.\texttt{ccpar} \leftarrow \varnothing$

ccpar function

Returns parents of all nodes in *i*'s congruence class

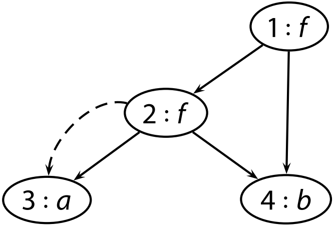
$$\texttt{let}\ \text{ccpar}\ i =$$
$$(\text{node}\ (\text{find}\ i)).\texttt{ccpar}$$

congruent predicate

Test whether i_1 and i_2 are congruent

$$\begin{aligned}
&\texttt{let}\ \text{congruent}\ i_1\ i_2 = \\
&\quad \texttt{let}\ n_1\ =\ \text{node}\ i_1\ \texttt{in} \\
&\quad \texttt{let}\ n_2\ =\ \text{node}\ i_2\ \texttt{in} \\
&\quad n_1.\texttt{fn} = n_2.\texttt{fn} \\
&\qquad \land\ |n_1.\texttt{args}| = |n_2.\texttt{args}| \\
&\qquad \land\ \forall i \in \{1, \ldots, |n_1.\texttt{args}|\}.\ \text{find}\ n_1.\texttt{args}[i] = \text{find}\ n_2.\texttt{args}[i]
\end{aligned}$$

Example:



Are 1 and 2 congruent?

`fn` fields	— both f
# of arguments	— same
left arguments $f(a, b)$ and a	— both congruent to 3
right arguments b and b	— both 4 (congruent)

Therefore 1 and 2 are congruent.

merge function

```
let rec merge i1 i2 =
  if find i1 ≠ find i2 then begin
    let P_i1 = ccpar i1 in
    let P_i2 = ccpar i2 in
    union i1 i2;
    foreach t1, t2 ∈ P_i1 × P_i2 do
      if find t1 ≠ find t2 ∧ congruent t1 t2
      then merge t1 t2
    done
  end
```

P_{i_1} and P_{i_2} store the current values of ccpar i_1 and ccpar i_2.

Decision Procedure: T_E-satisfiability

Given Σ_E-formula

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

with subterm set S_F, perform the following steps:

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

Theorem (Sound and Complete)

Quantifier-free conjunctive Σ_E-formula F is T_E-satisfiable iff the congruence closure algorithm returns satisfiable.

Recursive Data Structures

Recursive Data Structures

Quantifier-free Theory of Lists T_{cons}

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

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

Axioms of T_{cons}

- reflexivity, symmetry, transitivity
- congruence axioms:

$$\begin{array}{l}\forall x_1, x_2, y_1, y_2.\ x_1 = x_2 \ \wedge\ y_1 = y_2 \ \rightarrow\ \text{cons}(x_1, y_1) = \text{cons}(x_2, y_2)\\ \forall x, y.\ x = y \ \rightarrow\ \text{car}(x) = \text{car}(y)\\ \forall x, y.\ x = y \ \rightarrow\ \text{cdr}(x) = \text{cdr}(y)\end{array}$$

- equivalence axiom:

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

-

$$\begin{array}{llr}(A1) & \forall x, y.\ \text{car}(\text{cons}(x, y)) = x & \text{(left projection)}\\ (A2) & \forall x, y.\ \text{cdr}(\text{cons}(x, y)) = y & \text{(right projection)}\\ (A3) & \forall x.\ \neg\text{atom}(x) \rightarrow \text{cons}(\text{car}(x), \text{cdr}(x)) = x & \text{(construction)}\\ (A4) & \forall x, y.\ \neg\text{atom}(\text{cons}(x, y)) & \text{(atom)}\end{array}$$

Simplifications

- Consider only quantifier-free conjunctive Σ_{cons}-formulae. Convert non-conjunctive formula to DNF and check each disjunct.
- $\neg\text{atom}(u_i)$ literals are removed:

$$\boxed{\text{replace} \quad \neg\text{atom}(u_i) \quad \text{with} \quad u_i = \text{cons}(u_i^1, u_i^2)}$$

  by the (construction) axiom.
- Because of similarity to Σ_{E}, we sometimes combine $\Sigma_{\text{cons}} \cup \Sigma_{\text{E}}$.

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

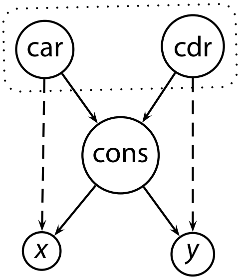
$$
\begin{array}{lll}
F : & & \underbrace{s_1 = t_1 \ \wedge \ \cdots \ \wedge \ s_m = t_m}_{\text{generate congruence closure}} \\
& \wedge & \underbrace{s_{m+1} \neq t_{m+1} \ \wedge \ \cdots \ \wedge \ s_n \neq t_n}_{\text{search for contradiction}} \\
& \wedge & \underbrace{\text{atom}(u_1) \ \wedge \ \cdots \ \wedge \ \text{atom}(u_l)}_{\text{search for contradiction}}
\end{array}
$$

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

Algorithm: T_{cons}-Satisfiability

1. Construct the initial DAG for S_F
2. for each node n with $n.\texttt{fn} = \text{cons}$
   - add $\text{car}(n)$ and merge $\text{car}(n)$ $n.\texttt{args}[1]$
   - add $\text{cdr}(n)$ and merge $\text{cdr}(n)$ $n.\texttt{args}[2]$

   by axioms (A1), (A2)
3. for $1 \leq i \leq m$, merge s_i t_i
4. for $m + 1 \leq i \leq n$, if find s_i = find t_i, return **unsatisfiable**
5. for $1 \leq i \leq l$, if $\exists v$. find v = find u_i $\wedge$ $v.\texttt{fn} = \text{cons}$, return **unsatisfiable**
6. Otherwise, return **satisfiable**



Example:

Given $(\Sigma_{\text{cons}} \cup \Sigma_{\text{E}})$-formula

$$F: \quad \begin{array}{l} \quad \text{car}(x) = \text{car}(y) \ \wedge \ \text{cdr}(x) = \text{cdr}(y) \\ \wedge \neg\text{atom}(x) \ \wedge \ \neg\text{atom}(y) \ \wedge \ f(x) \neq f(y) \end{array}$$

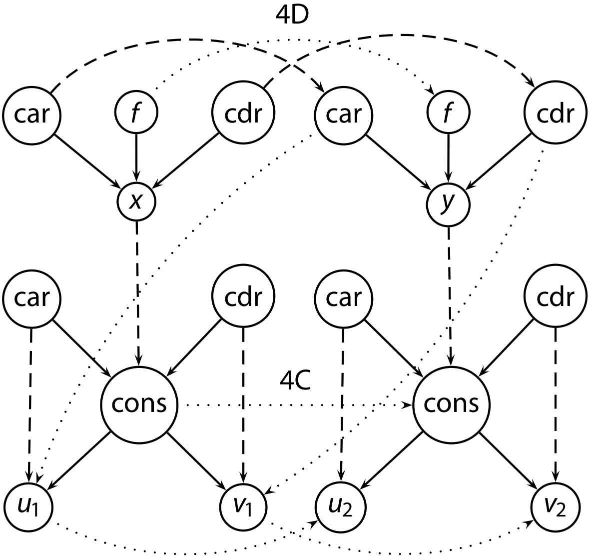
where the function symbol f is in Σ_{E}

$$F': \quad \begin{array}{lll} \text{car}(x) = \text{car}(y) & \wedge & (1) \\ \text{cdr}(x) = \text{cdr}(y) & \wedge & (2) \\ x = \text{cons}(u_1, v_1) & \wedge & (3) \\ y = \text{cons}(u_2, v_2) & \wedge & (4) \\ f(x) \neq f(y) & & (5) \end{array}$$

Recall the projection axioms:

$$\begin{array}{ll} (A1) & \forall x, y.\ \text{car}(\text{cons}(x, y)) = x \\ (A2) & \forall x, y.\ \text{cdr}(\text{cons}(x, y)) = y \end{array}$$

Example(cont): congruence



F is **unsatisfiable**