Implementation of a Fast Congruence Closure Algorithm

Patrick Bahr
s0404888@inf.tu-dresden.de

Technische Universität Dresden

December 17, 2007

Outline

Outline

Word Problem of Equational Logic

- Given: Finite set of equations E, and a single equation $s \approx t$
- Question: $E \models s \approx t$, i.e., follows $s \approx t$ from the equations in E?

Definition

$E \models s \approx t$, also written $s \approx_E t$, iff every model of E is a model of $s \approx t$. That is, for all algebras $\mathcal{A}$ we have:

$$\mathcal{A} \models E \qquad \text{implies} \qquad \mathcal{A} \models s \approx t$$

Solving the Word Problem: Rewriting Systems

- Idea: Read E as rewriting rules, i.e., equations in E are only "applied" from left to right.
- If the resulting rewriting system $\rightarrow_E$ can be proven terminating and confluent $s \approx_E t$ is decidable by checking $s{\downarrow} = t{\downarrow}$.
- If it's not: Use Knuth-Bendix completion to create an equivalent rewriting system that can be proven terminating and confluent.
- Of course, this procedure does not always succeed!
- It does if all equations in E are ground!

Hence, $s \approx_E t$ is decidable if E is ground.

Solving the Word Problem: Congruence Closure

If E is ground, there is an alternative solution.

Theorem

Let Σ be a signature and E be a set of ground Σ-equations. $\approx_E$ is the smallest congruence relation containing E.

If we restrict ourselves to the subterms in E, s and t this congruence closure is computable.

Congruence

Definition

Let Σ be a signature and $\equiv$ an equivalence relation on T_Σ. $\equiv$ is called a congruence relation if the following condition holds:
If for some $k \geq 0$ we have $t_i \equiv s_i$ for all $1 \leq i \leq k$ and $f \in \Sigma^{(k)}$ then also $f(t_1, \ldots, t_k) \equiv f(s_1, \ldots, s_k)$.

Representing and Manipulating Equivalence Relations

General idea:

- Represent the equivalence relation as its quotient set, i.e. the set of all equivalence classes.
- Operation find to get the equivalence class of a given element.
- Operation union to combine two equivalence classes.

Representing and Manipulating Equivalence Relations

General idea:

- Represent the equivalence relation as its quotient set, i.e. the set of all equivalence classes.
- Operation find to get the equivalence class of a given element.
- Operation union to combine two equivalence classes.

Two different approaches:

- Represent the quotient set as a lookup table mapping from elements to equivalence class names.

  $\leadsto$ find: $\mathcal{O}(1)$; union: $\mathcal{O}(n)$.

Representing and Manipulating Equivalence Relations

General idea:

- Represent the equivalence relation as its quotient set, i.e. the set of all equivalence classes.
- Operation find to get the equivalence class of a given element.
- Operation union to combine two equivalence classes.

Two different approaches:

- Represent the quotient set as a lookup table mapping from elements to equivalence class names.
  $\leadsto$ find: $\mathcal{O}(1)$; union: $\mathcal{O}(n)$.
- Represent the quotient set as a forest, where each tree represents an equivalence class. The root of a tree is the canonical representative of the class.
  $\leadsto$ find: $\mathcal{O}(n)$; union: $\mathcal{O}(1)$.

Representing and Manipulating Equivalence Relations

General idea:

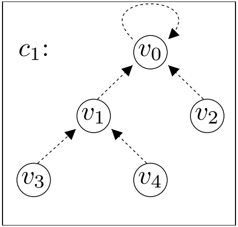
- Represent the equivalence relation as its quotient set, i.e. the set of all equivalence classes.
- Operation find to get the equivalence class of a given element.
- Operation union to combine two equivalence classes.

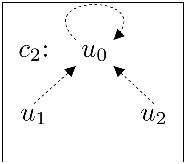
Two different approaches:

- Represent the quotient set as a lookup table mapping from elements to equivalence class names.
  $\rightsquigarrow$ find: $\mathcal{O}(1)$; union: $\mathcal{O}(n)$.
- Represent the quotient set as a forest, where each tree represents an equivalence class. The root of a tree is the canonical representative of the class.
  $\rightsquigarrow$ find: $\mathcal{O}(n)$; union: $\mathcal{O}(1)$.

Equivalence Classes as Trees (1)

Equivalence class $c_1 = \{v_0, v_1, v_2, v_3, v_4\}$
Equivalence class $c_2 = \{u_0, u_1, u_2\}$

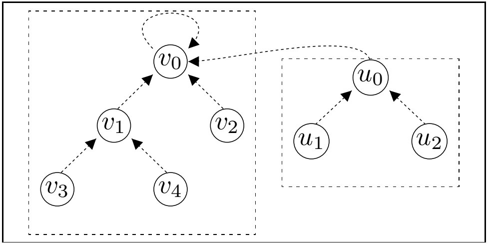




- operation find traverses up the tree until the root is reached.
- e.g. $\text{find}(R, v_4) = \text{find}(R, v_2) = v_0$
- operation union takes two roots of some trees, and makes one of them child of the other.

Equivalence Classes as Trees (2)

E.g. union(R, v_0, u_0) produces the following:



We now have only one class $c = c_1 \cup c_2 = \{v_0, v_1, v_2, v_3, v_4, u_0, u_1, u_2\}$ represented by the node v_0

Terms as Digraphs

Terms will be interpreted as labelled graphs in the following.

Definition

Let Σ be a signature, $t \in T_\Sigma$ and E a set of ground Σ-equations.

(i) The labelled digraph $G_t = (V_t, E_t, l_t)$, where $V_t = \mathcal{P}os(t)$, $l(p) = t(p)$ for all $p \in V_t$, $E_t = \{(p, p') \in V_t^2 \mid \exists i \in \mathbb{N}.p' = pi\}$ and for each node $p \in V_t$ the set of successors of p is ordered by $p1 < \cdots < pk$, is called the graph of t. Note that G_t is a tree. By $v_t \in V_t$ we denote the root of this tree.

(ii) The graph $G_E = \biguplus_{s \approx s'} G_s \uplus G_{s'}$ is called the graph of E.

Achieving Congruence

Definition

Let Σ be a signature, $G = (V, E, l : V \to \Sigma)$ a labelled digraph, v a node in G, $v_1 < \cdots < v_k$ its successors and $R \in Eq_{G,C}$. The Σ, C-signature of v w.r.t. R is the $(k+1)$-tuple
$sig(R, v) = (l(v), \mathrm{find}(R, v_1), \ldots, \mathrm{find}(R, v_k))$.

Nodes having the same Σ, C-signature should be in the same equivalence class to achieve congruence!

Outline

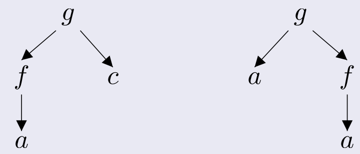
1. Introduction
   - Motivation
   - Preliminaries
2. Congruence Closure and Decision Algorithm
   - Congruence Closure Algorithm
   - Decision Algorithm
3. Conclusive Remarks

The Idea of the Algorithm

- Every term of the equations is interpreted as a graph.
- Every node is put into a singleton class.
- Roots of terms that are equal are put in the same equivalence class using union.
- Until no further changes can be derived the following is repeated:
  - Find nodes that have the same Σ, C-signature but are in different equivalence classes.
  - Combine the equivalence classes of these two nodes.

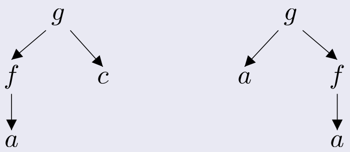
An Example (1)

Graph of the set $\{g(f(a), c) \approx g(a, f(a))\}$

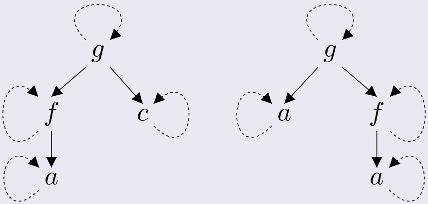


An Example (1)

Graph of the set $\{g(f(a), c) \approx g(a, f(a))\}$

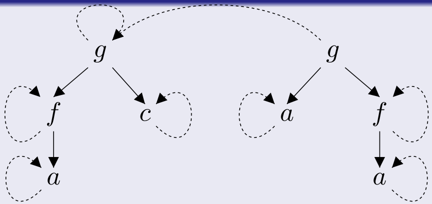


Singleton equivalence classes



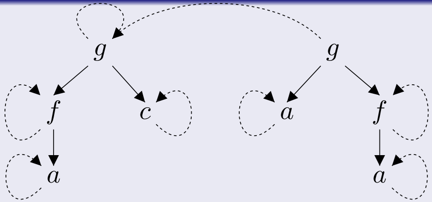
An Example (2)

union of the roots' equivalence classes

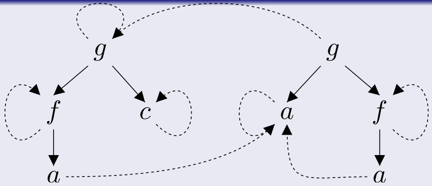


An Example (2)

union of the roots' equivalence classes

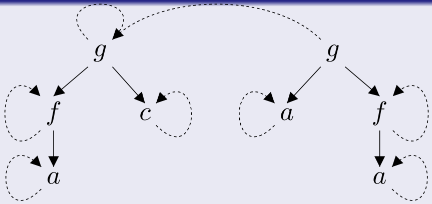


Establish congruence

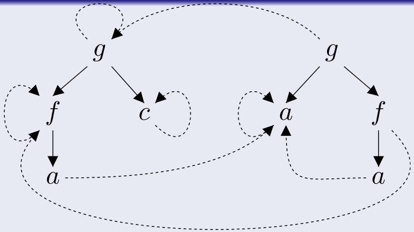


An Example (2)

union of the roots' equivalence classes



Establish congruence



How to Keep Track of Changes of Equivalence Classes?

- Maintain for each equivalence class c the set $\mathrm{pred}(R, c)$ of nodes that have a successor that is in c.
  $\rightsquigarrow$ helps to find nodes that have to be checked for their Σ, C-signatures again
- Maintain a set *pending* of nodes that have to be checked for their Σ, C-signatures.
- Maintain a signature table τ, that maps from Σ, C-signatures to equivalence classes that have a node having this Σ, C-signature.
  $\rightsquigarrow$ helps to find equivalence classes that have to be merged.
- Maintain a set *combine* of pairs of equivalence classes that have to be merged.

The Algorithm

input: set E of ground Σ-equations
$\tau \leftarrow \varepsilon$ $\quad \triangleright$ Initialise Σ, C-signature table τ as empty
set R s.t. $\text{find}(R, v) = v$ for all $v \in V_E$.
for $s \approx t \in E$ **do** $\quad \triangleright$ Impose the desired equalities.
    $R \leftarrow \text{union}(R, v_s, v_t)$
end for
$\mathit{pending} \leftarrow V_E$
while $\mathit{pending} \neq \emptyset$ **do**
    $\mathit{combine} \leftarrow \emptyset$
    $\textsc{CheckSignature}(R, \tau, \mathit{pending}, \mathit{combine})$
    $\mathit{pending} \leftarrow \emptyset$
    $\textsc{Combine}(R, \tau, \mathit{pending}, \mathit{combine})$
end while
output: τ

The CHECKSIGNATURE Subprocedure

procedure $\textsc{CheckSignature}(R, \tau, \mathit{pending}, \mathit{combine})$
  for $v \in \mathit{pending}$ **do**
    if $\tau(\mathit{sig}(v)) = \perp$ **then**
      $\tau \leftarrow \tau[\mathit{sig}(v) \mapsto \mathrm{find}(R, v)]$
    else if $\mathrm{find}(R, v) \neq \tau(\mathit{sig}(v))$ **then**
      $\mathit{combine} \leftarrow \mathit{combine} \cup \{(\mathrm{find}(R, v), \tau(\mathit{sig}(v)))\}$
    end if
  end for
end procedure

The COMBINE Subprocedure

procedure COMBINE($R, \mathcal{T}, \mathit{pending}, \mathit{combine}$)
    for $(e_1, e_2) \in \mathit{combine}$ **do**
        if both e_1 and e_2 are used in R **then**
            if $\mathrm{weight}(R, e_1) < \mathrm{weight}(R, e_2)$ **then**
                $(e_1, e_2) \leftarrow (e_2, e_1)$
            end if
            for $u \in \mathrm{pred}(R, e_2)$ **do**
                $\mathcal{T} \leftarrow \mathcal{T}\backslash_{\mathit{sig}(u)}$
                $\mathit{pending} \leftarrow \mathit{pending} \cup \{u\}$
            end for
            $R \leftarrow \mathrm{union}(R, e_1, e_2)$
        end if
    end for
end procedure

Using the signature table to decide equations

- Output of the congruence closure algorithm is the final signature table τ.
- Equivalence class of a term can be recursively computed using the signature table.
- If a Σ, C-signature s is computed that is not considered in τ, s is put into τ mapping it to a fresh equivalence class name from C.
- Therefore τ is transformed beforehand such that equivalence class names are integers.

The Algorithm

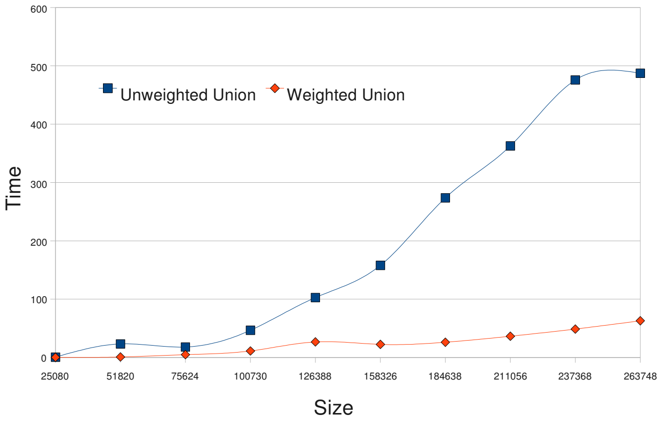
input: ground Σ-equation $s \approx t$, Σ, C-signature table τ
output: true if $\textsc{Class}(s) = \textsc{Class}(t)$; otherwise false
function $\textsc{Class}(t = f(t_1, \dots t_k) \in T_\Sigma)$
    for $i \in \{1, \dots k\}$ **do**
        $c_i = \textsc{Class}(t_i)$
    end for
    if $\tau((f, c_1, \dots, c_k)) = \perp$ **then**
        take some $c \in C \setminus dom(\tau)$
        $\tau \leftarrow \tau[(f, c_1, \dots, c_k) \mapsto c]$.
    else
        $c \leftarrow \tau((f, c_1, \dots, c_k))$
    end if
    return c
end function

Outline

Complexity

- The congruence closure algorithm takes $\mathcal{O}(n \log n)$ time for input equations of an overall size of n.
- The decision algorithm takes $\mathcal{O}(m)$ time for an input equation of size m.
- I.e., in particular the runtime of the decision algorithm is independent of the size of the equations defining the equational theory.

Runtime Comparison to Downey et al.



Bibliography

Alfred V. Aho and John E. Hopcroft.
The Design and Analysis of Computer Algorithms.
Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1974.

Franz Baader and Tobias Nipkow.
Term Rewriting and All That.
Cambridge University Press, 1999.

Peter J. Downey, Ravi Sethi, and Robert Endre Tarjan.
Variations on the common subexpression problem.
J. ACM, 27(4):758–771, 1980.