Disjoint-set Data Structure for Equality Theory

Thierry Martinez

Équipe–Projet Contraintes

Contraintes’ Lab talk, 26 May 2008
Outline

1. Introduction

2. Disjoint-sets Abstract Machine

3. Data Structure

4. Concurrent Constraints

5. Conclusion
Constraint System for the Equality Theory.

Let $\mathcal{V}$ be a set of variables.

For every term $\phi$:

- $\text{fv}(\phi)$ denotes the set of free variables of $\phi$;
- $\text{bv}(\phi)$ denotes the set of bound variables of $\phi$.

### Constraints

$\mathcal{C} := 1 | \mathcal{C} \land \mathcal{C} | \exists x(\mathcal{C}) | x = y$

(common to all constraint systems)

For $\phi \in \mathcal{C}$, we can always suppose that:

- $\text{fv}(\phi) \cap \text{bv}(\phi) = \emptyset$;
- each variable is bound at most one time.

### Axioms

In addition to logical axioms for $1$ and $\land$ and $\exists$:

- $\vdash_{\mathcal{C}} x = x$;
- $x = y \vdash_{\mathcal{C}} y = x$;
- $x = y, y = z \vdash_{\mathcal{C}} x = z$.

For every variable $\phi$, let $\mathcal{C}$ enumerate $\text{fv}(\phi)$. We always have $\vdash_{\mathcal{C}} \exists \overrightarrow{x}(\phi)$. 
Equality Theory and Disjoint-sets.

For every constraint $c$, we denote $x \sim_c y$ when $x = y$ is a sub-term of $c$.

\[ \exists xyzt (a = x \land b = y \land x = y \land c = d \land z = t) \]

**Definition**

A partition $\mathcal{P}$ of $\text{fv}(c)$ is a model for a constraint $c$ if for every $x, y \in \text{fv}(c)$ such that $x \sim_c \cdots \sim_c y$, we have $x$ and $y$ in the same equivalence class of $\mathcal{P}$. 
Definition

\[ \mathcal{P}_1 \sqsubseteq \mathcal{P}_2 \text{ when for every } x \text{ and } y \text{ belonging to the same equivalence class of } \mathcal{P}_1, \text{ we have } x \text{ and } y \text{ in the same equivalence class of } \mathcal{P}_2. \]
**Smallest Model.**

For every constraint $c$, we denote $x \approx_c y$ when there exists $z$ such that $x \sim_c \cdots \sim_c z$ and $y \sim_c \cdots \sim_c z$.

**Proposition**

For every constraint $c$, $\approx_c$ is an equivalence relation and the induced partition is the smallest model of $c$.

**Proposition**

For every $x, y \in \text{fv}(c)$, $c \vdash x = y$ if and only if $x \approx_c y$. 
Constraint Entailment.

\[ \exists x y z t (a = x \land b = y \land x = y \land c = d \land z = t) \]

\[ \vdash_{\mathcal{C}} \exists u v (a = u \land b = u \land c = v \land d = v) \]

**Proposition**

\[ c_1 \vdash c_2 \text{ if and only if, for every } x, y \in \text{fv}(c_2) \text{ such that } x \approx_{c_2} y, \text{ we have } x \text{ and } y \text{ in the same equivalence class in every model of } c_1. \]

It suffices to check in the inclusion between smallest models restricted to free variables.
A Very Simple Logic Language.

Program

\[
\mathcal{P} := \forall \overrightarrow{x} (\mathcal{C} \Rightarrow \mathcal{C}) ?
\]

Closed term

\[
\begin{aligned}
\text{hypotheses} & \\
\text{query} & \\
\end{aligned}
\]

Execution Scheme

1. Build the smallest model associated to the hypotheses.
2. Read the model to check whether the query is entailed.
Build the Smallest Model Associated to a Constraint.

\[ C \models \exists g(g = a \land a = b) \land \exists ij(d = i \land e = j \land \exists hkl(i = h \land j = k \land f = l \land c = h)) \land d = e \]
Build the Smallest Model Associated to a Constraint.

\[ C \equiv \exists g (g = a \land a = b) \land \exists ij (d = i \land e = j \land \exists hkl (i = h \land j = k \land f = l \land c = h)) \land d = e \]
Build the Smallest Model Associated to a Constraint.

\[ C \equiv \exists g (g = a \land a = b) \land \exists ij (d = i \land e = j \land \exists hkl (i = h \land j = k \land f = l \land c = h)) \land d = e \]
Build the Smallest Model Associated to a Constraint.

\[ C \equiv \exists g(g = a \land a = b) \land \exists ij(d = i \land e = j \land \exists hkl(i = h \land j = k \land f = l \land c = h)) \land d = e \]
Build the Smallest Model Associated to a Constraint.

\[ C \equiv \exists g (g = a \land a = b) \land \exists ij (d = i \land e = j \land \exists hkl (i = h \land j = k \land f = l \land c = h)) \land d = e \]
Build the Smallest Model Associated to a Constraint.

\[ C \triangleq \exists g(g = a \land a = b) \land \exists ij(d = i \land e = j \land \exists hkl(i = h \land j = k \land f = l \land c = h)) \land d = e \]
Build the Smallest Model Associated to a Constraint.

\[ C \models \exists g(g = a \land a = b) \land \exists ij(d = i \land e = j \land \exists hkl(i = h \land j = k \land f = l \land c = h)) \land d = e \]
Build the Smallest Model Associated to a Constraint.

\[ C \models \exists g (g = a \land a = b) \land \exists ij (d = i \land e = j \land \exists hkl (i = h \land j = k \land f = l \land c = h)) \land d = e \]
Build the Smallest Model Associated to a Constraint.

\[ \exists g (g = a \land a = b) \land \exists ij (d = i \land e = j \land \exists hkl (i = h \land j = k \land f = l \land c = h)) \land d = e \]

Two op-codes

\[ x \equiv y \text{ and } x \equiv y \]
Compiling Queries.

\[ C_0 ::= 1 \mid C_0 \land C_0 \mid x = y \]

**Proposition**

*For every* \( c \in C \), *there is a computable* \( J_{cK} \in C_0 \) *such that* \( c \vdash \models J_{cK} \).

**Proof.**

**Computation.** \( c \vdash \models a = b \land c = d \land c = e \)

\( c \vdash \models J_{cK} \). \( c \) *and* \( J_{cK} \) *have the same models.*
Compiling Queries.

\[ C_0 ::= 1 \mid C_0 \land C_0 \mid x = y \]

**Proposition**

For every \( c \in \mathcal{C} \), there is a computable \( J_{cK} \in C_0 \) such that \( c \models J_{cK} \).

**Proof.**

**Computation.** Let \( \mathfrak{P} \) be the smallest model of \( c \).

\[
J_{cK} \models \bigwedge_{P \in \mathfrak{P}} \bigwedge_{y \in V} x = y
\]

\( c \models J_{cK} \) and \( J_{cK} \) have the same models.
Extension to Rational Terms.

Constraints $\mathcal{C} ::= 1 | \mathcal{C} \land \mathcal{C} | \exists x(\mathcal{C}) | x = y | x = f(\overrightarrow{y})$

common to all constraint systems

New Op-codes  Labelled equivalence classes.

- $\text{struct}(x, f(\overrightarrow{y}))$ to label an equivalence class, possible previous label must match and equality constraints on arguments are added.
- $\overrightarrow{y} \leftarrow \text{struct?}(x, f)$ to check that $x$ has the functor $f$, then extract arguments.
Build the Code Associated to a Rational Tree Query.

\[ C \equiv \exists xyz (a = f(x, y) \land x = f(z, y) \land z = f(x, y)) \]

1. \((x, y) \leftarrow \text{struct?}(a, f)\)
2. \((z, y_1) \leftarrow \text{struct?}(x, f)\)
3. \(y_1 \equiv y\)
4. \((x_1, y_2) \leftarrow \text{struct?}(z, f)\)
5. \(x_1 \equiv x\)
6. \(y_2 \equiv y\)
State Minimization.

\[ C \equiv \exists xyz (a = f(x, y) \land x = f(z, y) \land z = f(x, y)) \]

1. \((a_1, y) \leftarrow \text{struct?}(a, f)\)
2. \(? a_1 = a\)
Equivalence Classes as Variable Trees.

Op-codes implementation:

\[ x \dot{=} y \quad \text{“link(find(x), find(y))”} \]

\[ x \overset{?}{=} y \quad \text{check whether “find(x) = find(y)”} \]

where:

- \( \text{find}(x) \): Follows the arcs from \( x \) up to find the root. Root is returned.
- \( \text{link}(x, y) \): Adds an arc from \( x \) to \( y \). Prerequisite: \( x \) former root, \( y \) root.

\[ C \models \exists g (g = a \land b = g) \land \exists ij (d = i \land e = j \land \exists hkl (i = h \land j = k \land f = l \land c = h)) \land d = e \]

Proposition

For a sequence of \( m \) operations, among which \( n \) are “link” operations, the total complexity is \( \Theta(mn) \).
Equivalence Classes as Variable Trees.

Op-codes implementation:

\[ x \doteq y \quad \text{“link(find(x), find(y))”} \]

\[ x \equiv y \quad \text{check whether “find(x) = find(y)”} \]

where:

- \( \text{find}(x) \) Follows the arcs from \( x \) up to find the root. Root is returned.
- \( \text{link}(x, y) \) Adds an arc from \( x \) to \( y \). Prerequisite: \( x \) former root, \( y \) root.

\[ C \models \exists g (g = a \land b = g) \land \exists ij (d = i \land e = j \land \exists hkl (i = h \land j = k \land f = l \land c = h)) \land d = e \]

Proposition

For a sequence of \( m \) operations, among which \( n \) are “link” operations, the total complexity is \( \Theta(mn) \).
Equivalence Classes as Variable Trees.

Op-codes implementation:

\[ x \doteq y \quad \text{"link(find(x), find(y))"} \]
\[ x \overset{?}{=} y \quad \text{check whether } \text{"find}(x) = \text{find}(y)" \]

where:

- \text{find}(x) \quad \text{Follows the arcs from } x \text{ up to find the root. Root is returned.}
- \text{link}(x, y) \quad \text{Adds an arc from } x \text{ to } y. \text{ Prerequisite: } x \text{ former root, } y \text{ root.}

\[
C \doteq \exists g (g = a \land b = g) \land \exists ij (d = i \land e = j \land \exists hkl (i = h \land j = k \land f = l \land c = h)) \land d = e
\]

Proposition

For a sequence of \( m \) operations, among which \( n \) are “link” operations, the total complexity is \( \Theta(mn) \).
Equivalence Classes as Variable Trees.

Op-codes implementation:

\[ \dot{x} = y \quad \text{“link(find(x), find(y))”} \]

\[ x \equiv y \quad \text{check whether “find(x) = find(y)”} \]

where:

- \( \text{find}(x) \) follows the arcs from \( x \) up to find the root. Root is returned.
- \( \text{link}(x, y) \) adds an arc from \( x \) to \( y \). Prerequisite: \( x \) former root, \( y \) root.

\[
C \models \exists g (g = a \land b = g) \land \exists ij (d = i \land e = j \land \exists hkl (i = h \land j = k \land f = l \land c = h)) \land d = e
\]

Proposition

For a sequence of \( m \) operations, among which \( n \) are “link” operations, the total complexity is \( \Theta(mn) \).
Equivalence Classes as Variable Trees.

Op-codes implementation:

\[ \overset{\text{link}}{=} \quad \text{“link(find(x), find(y))”} \]

\[ \overset{\text{?}}{=} \quad \text{check whether “find(x) = find(y)”} \]

where:

- \text{find}(x) \quad \text{Follows the arcs from } x \text{ up to find the root. Root is returned.}
- \text{link}(x, y) \quad \text{Adds an arc from } x \text{ to } y. \text{ Prerequisite: } x \text{ former root, } y \text{ root.}

\[ C \hat{=} \exists g (g = a \land b = g) \land \exists ij (d = i \land e = j) \land \exists hkl (i = h \land j = k \land f = l \land c = h) \land d = e \]

**Proposition**

For a sequence of \( m \) operations, among which \( n \) are “link” operations, the total complexity is \( \Theta(mn) \).
Equivalence Classes as Variable Trees.

Op-codes implementation:

\[ \bar{x} \doteq \bar{y} \ 	ext{“link(find(x), find(y))”} \]
\[ x \doteq y \ 	ext{check whether “find(x) = find(y)”} \]

where:

find(x) Follows the arcs from x up to find the root. Root is returned.

link(x, y) Adds an arc from x to y. Prerequisite: x former root, y root.

\[ C \models \exists g (g = a \land b = g) \land \exists ij (d = i \land e = j \land \exists hkl (i = h \land j = k \land f = l \land c = h)) \land d = e \]

Proposition

For a sequence of m operations, among which n are “link” operations, the total complexity is \( \Theta(mn) \).
Equivalence Classes as Variable Trees.

Op-codes implementation:

\[ x \overset{\text{link}}{=} y \quad \text{“link(find(x), find(y))”} \]

\[ x \overset{?}{=} y \quad \text{check whether “find(x) = find(y)”} \]

where:

- \( \text{find}(x) \) Follows the arcs from \( x \) up to find the root. Root is returned.
- \( \text{link}(x, y) \) Adds an arc from \( x \) to \( y \). Prerequisite: \( x \) former root, \( y \) root.

\[ C \models \exists g (g = a \land b = g) \land \exists ij (d = i \land e = j \land \exists hkl (i = h \land j = k \land f = l \land c = h)) \land d = e \]

Proposition

For a sequence of \( m \) operations, among which \( n \) are “link” operations, the total complexity is \( \Theta(mn) \).
Equivalence Classes as Variable Trees.

Op-codes implementation:

\[ x \doteq y \quad \text{“link(find}(x), \text{find}(y))” \]

\[ x \doteq y \quad \text{check whether “find}(x) = \text{find}(y)” \]

where:

- \( \text{find}(x) \) Follows the arcs from \( x \) up to find the root. Root is returned.
- \( \text{link}(x, y) \) Adds an arc from \( x \) to \( y \). Prerequisite: \( x \) former root, \( y \) root.

\[
C \doteq \exists g (g = a \land b = g) \land \exists ij (d = i \land e = j \land \exists hkl (i = h \land j = k \land f = l \land c = h)) \land d = e
\]

Proposition

For a sequence of \( m \) operations, among which \( n \) are “link” operations, the total complexity is \( \Theta(mn) \).
Equivalence Classes as Variable Trees.

Op-codes implementation:

\[ \begin{align*}
  x \doteq y & \quad \text{“link(find(x), find(y))”} \\
  x \mathbin{\#} y & \quad \text{check whether “find(x) = find(y)”}
\end{align*} \]

where:

- \( \text{find}(x) \) Follows the arcs from \( x \) up to find the root. Root is returned.
- \( \text{link}(x, y) \) Adds an arc from \( x \) to \( y \). Prerequisite: \( x \) former root, \( y \) root.

\[
C \models \exists g (g = a \land b = g) \land \exists ij (d = i \land e = j \land \exists hkl (i = h \land j = k \land f = l \land c = h)) \land d = e
\]

Proposition

For a sequence of \( m \) operations, among which \( n \) are “link” operations, the total complexity is \( \mathcal{O}(mn) \).
Equivalence Classes as Variable Trees.

Op-codes implementation:

\[ x \doteq y \quad \text{“} \text{link(find}(x), \text{find}(y)) \text{”} \]

\[ x \equiv y \quad \text{check whether “} \text{find}(x) = \text{find}(y) \text{”} \]

where:

find\( (x) \) Follows the arcs from \( x \) up to find the root. Root is returned.

link\( (x, y) \) Adds an arc from \( x \) to \( y \). Prerequisite: \( x \) former root, \( y \) root.

\[ C \doteq \exists g (g = a \land b = g) \land \exists ij (d = i \land e = j \land \exists hkl (i = h \land j = k \land f = l \land c = h)) \land d = e \]

### Proposition

For a sequence of \( m \) operations, among which \( n \) are “link” operations, the total complexity is \( \Theta(mn) \).
**Heuristic 1: Union by Rank.**

Roots keep track (of an upper-bound) of tree heights.

\[
C \uparrow a = e \land b = e \land c = f \land d = f \land f = e
\]

**Proposition**

For a sequence of \( m \) operations, among which \( n \) are “link” operations, the total complexity is \( \Theta(m \log n) \).
Heuristic 1: Union by Rank.

Roots keep track (of an upper-bound) of tree heights.

\[ C \triangleq a = e \land b = e \land c = f \land d = f \land f = e \]

Proposition

For a sequence of \( m \) operations, among which \( n \) are “link” operations, the total complexity is \( \Theta(m \log n) \).
Heuristic 1: Union by Rank.

Roots keep track (of an upper-bound) of tree heights.

\[ C \triangleq a = e \land b = e \land c = f \land d = f \land f = e \]

Proposition

For a sequence of \( m \) operations, among which \( n \) are “link” operations, the total complexity is \( \Theta(m \log n) \).
Heuristic 1: Union by Rank.

Roots keep track (of an upper-bound) of tree heights.

\[ C \triangleq a = e \wedge b = e \wedge c = f \wedge d = f \wedge f = e \]

Proposition

For a sequence of \( m \) operations, among which \( n \) are “link” operations, the total complexity is \( \Theta(m \log n) \).
Heuristic 1: Union by Rank.

Roots keep track (of an upper-bound) of tree heights.

\[ C \triangleq a = e \land b = e \land c = f \land d = f \land f = e \]

**Proposition**

For a sequence of \( m \) operations, among which \( n \) are “link” operations, the total complexity is \( \Theta(m \log n) \).
Heuristic 1: Union by Rank.

Roots keep track (of an upper-bound) of tree heights.

\[ C \equiv a = e \land b = e \land c = f \land d = f \land f = e \]

Proposition

For a sequence of \( m \) operations, among which \( n \) are “link” operations, the total complexity is \( \Theta(m \log n) \).
Heuristic 2: Path Compression.

Once “find” obtains the actual root, update the arc to directly point to it.

\[ C \hat{=} b = a \land c = a \land d = a \land e = a \land f = a \]

Proposition

For a sequence of \( f \) “find” operations and \( n \) “link” operations, the total complexity is \( O(n + f \cdot (1 + \log_{2+f/n} n)) \).
Heuristic 2: Path Compression.

Once “find” obtains the actual root, update the arc to directly point to it.

\[ C \triangleq b = a \land c = a \land d = a \land e = a \land f = a \]

Proposition

For a sequence of \( f \) “find” operations and \( n \) “link” operations, the total complexity is \( O(n + f \cdot (1 + \log_{2^{+f/n}} n)) \).
Heuristic 2: Path Compression.

Once “find” obtains the actual root, update the arc to directly point to it.

\[ C \overset{\hat{=}}{\leftarrow} b = a \land c = a \land d = a \land e = a \land f = a \]

**Proposition**

For a sequence of \( f \) “find” operations and \( n \) “link” operations, the total complexity is \( O(n + f \cdot (1 + \log_2 f/n n)) \).
Heuristic 2: Path Compression.

Once “find” obtains the actual root, update the arc to directly point to it.

\[ C \hat{=} b = a \land c = a \land d = a \land e = a \land f = a \]

Proposition

For a sequence of \( f \) “find” operations and \( n \) “link” operations, the total complexity is \( O(n + f \cdot (1 + \log_2 f/n)) \).
Heuristic 2: Path Compression.

Once “find” obtains the actual root, update the arc to directly point to it.

\[ C \overset{\hat{\ }}{=} b = a \land c = a \land d = a \land e = a \land f = a \]

Proposition

For a sequence of \( f \) “find” operations and \( n \) “link” operations, the total complexity is \( O(n + f \cdot (1 + \log_2 f/n)) \).
Heuristic 2: Path Compression.

Once “find” obtains the actual root, update the arc to directly point to it.

\[ C \hat{=} b = a \land c = a \land d = a \land e = a \land f = a \]

Proposition

For a sequence of \( f \) “find” operations and \( n \) “link” operations, the total complexity is \( O(n + f \cdot (1 + \log_{2+f/n} n)) \).
A very quickly growing function...

**Definition**

For integers $m, n \geq 0$:

$$A(m, n) = \begin{cases} 
  n + 1 & \text{if } m = 0, \\
  A(m - 1, 1) & \text{if } m > 0 \text{ and } n = 0, \\
  A(m - 1, A(m, n - 1)) & \text{if } m > 0 \text{ and } n > 0.
\end{cases}$$

<table>
<thead>
<tr>
<th>$m$</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A(m, 1)$</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>13</td>
<td>65533</td>
<td>$2^{2^{65536}} - 3 \gg 10^{80}$</td>
</tr>
</tbody>
</table>
...and its very slowly growing inverse.

**Definition**

For $n \geq 0$:

\[
\alpha(n) = \min \{k | A(k, 1) \geq 1\}
\]

For all practical purposes

\[
\alpha(n) \leq 5
\]
Proposition

For a sequence of $m$ operations, among which $n$ are “link” operations, the total complexity is $O(m\alpha(n))$. 
Program

\[
P ::= e \mid p(\overrightarrow{x}) \triangleright A \cdot P
\]

\[
A ::= (A \parallel A) \mid \exists x(A) \mid p(\overrightarrow{x}) \mid \text{tell}(C) \mid \forall \overrightarrow{x}(\overrightarrow{c} \rightarrow A)
\]

\[
(\exists \overrightarrow{x}(c) \rightarrow \exists \overrightarrow{x}(\text{tell}(c) \parallel a)
\]
New Op-codes for Translating $cc(=)$

call $p(\overrightarrow{x})$ / return  Flow-control operations.

freeze($x \neq y$, Code)  Ask on atomic constraints.

$$(a = b \land b = c \rightarrow A) \equiv (a = b \rightarrow (b = c \rightarrow A))$$

$$\leadsto \text{freeze}(a \neq b, \text{freeze}(b \neq c, J A K))$$
Conclusion

Good things

- The entailment checker can be deduced from the smallest model of the constraint.
- Computing the smallest model offline allows to produce optimal code for the virtual machine (when used with minimization for rational terms).

Open questions

- Better representation for the freeze on equalities.
- Extension with linear tokens.
  Let $f_1, \ldots, f_n$ be closures in a lcc/RH program (with asks $(\text{do}(f_1) \rightarrow a_1) \parallel \ldots \parallel (\text{do}(f_n) \rightarrow a_n)$). If implemented naively, $\text{tell}(\text{do}(x))$ leads to freezes on $x = f_1, \ldots, x = f_n$. 