Disjoint-set Data Structure for Equality Theory

Thierry Martinez

Équipe–Projet Contraintes

Contraintes' Lab talk, 26 May 2008

Outline

Constraint System for the Equality Theory.

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

For every term ϕ:

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

Constraints $\mathscr{C} ::= \underbrace{\mathbf{1} \mid \mathscr{C} \wedge \mathscr{C} \mid \exists x(\mathscr{C})}_{\text{common to all constraint systems}} \mid x = y$

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

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

Axioms In addition to logical axioms for $\mathbf{1}$ and $\wedge$ and $\exists$:

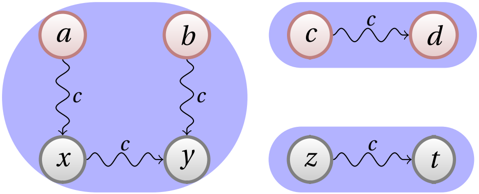
- $\vdash_{\mathscr{C}} x = x$;
- $x = y \vdash_{\mathscr{C}} y = x$;
- $x = y, y = z \vdash_{\mathscr{C}} x = z$.

For every variable ϕ, let $\overrightarrow{x}$ enumerate $\mathrm{fv}(\phi)$. We always have $\vdash_{\mathscr{C}} \exists \overrightarrow{x}(\phi)$.

Equality Theory and Disjoint-sets.

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

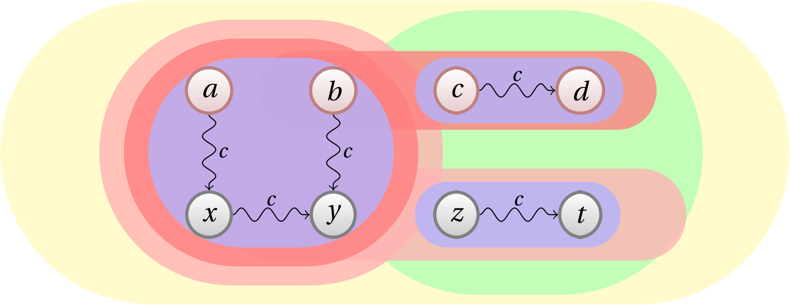
$$\exists xyzt(a = x \wedge b = y \wedge x = y \wedge c = d \wedge z = t)$$



Definition

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

Model Ordering.

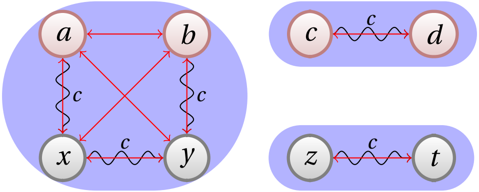


Definition

$\mathfrak{P}_1 \sqsubseteq \mathfrak{P}_2$ when for every x and y belonging to the same equivalence class of $\mathfrak{P}_1$, we have x and y in the same equivalence class of $\mathfrak{P}_2$.

Smallest Model.

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



Proposition

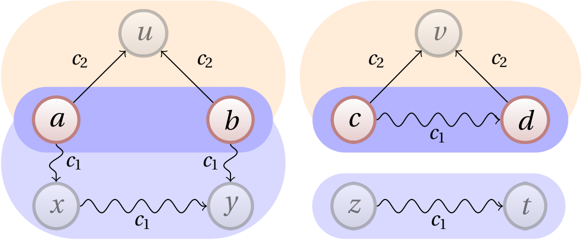
For every constraint c, $\approx_c$ is an equivalence relation and the induced partition is the smallest model of c.

Proposition

For every $x, y \in \mathrm{fv}(c)$, $c \vdash_{\mathscr{C}} x = y$ if and only if $x \approx_c y$.

Constraint Entailment.

$$\exists xyzt(a = x \wedge b = y \wedge x = y \wedge c = d \wedge z = t)$$
$$\overset{?}{\vdash_{\mathscr{C}}}$$
$$\exists uv(a = u \wedge b = u \wedge c = v \wedge d = v)$$



Proposition

$c_1 \vdash c_2$ if and only if, for every $x, y \in \mathrm{fv}(c_2)$ such that $x \approx_{c_2} y$, we have x and y 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

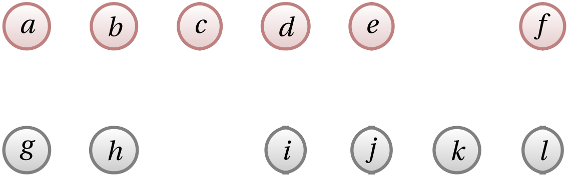
$$\mathscr{P} ::= \overbrace{\forall \overrightarrow{x} \; (\underbrace{\mathscr{C}}_{\text{hypotheses}} \Rightarrow \underbrace{\mathscr{C}}_{\text{query}})}^{\text{closed term}} ?$$

Execution Scheme

1. Build the smallest model associated to the hypotheses.
2. Read the model to check whether the query is entailed.

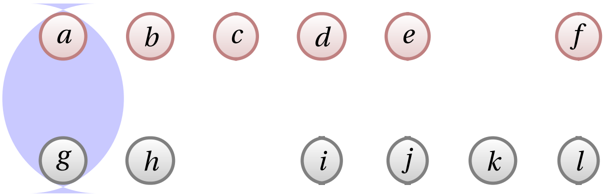
Build the Smallest Model Associated to a Constraint.

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



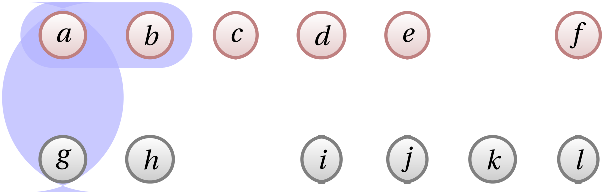
Build the Smallest Model Associated to a Constraint.

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



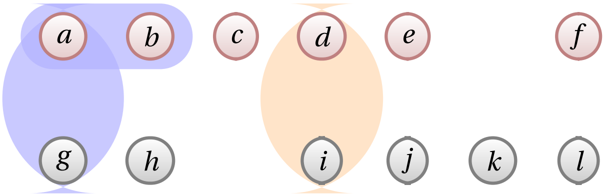
Build the Smallest Model Associated to a Constraint.

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



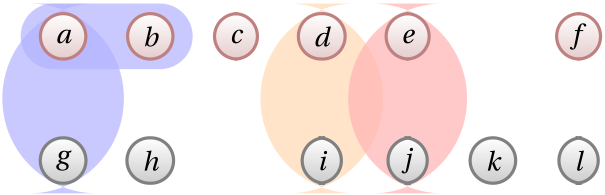
Build the Smallest Model Associated to a Constraint.

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



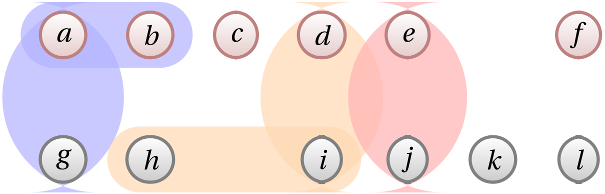
Build the Smallest Model Associated to a Constraint.

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



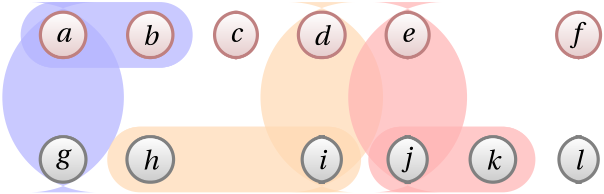
Build the Smallest Model Associated to a Constraint.

$$C \mathrel{\hat{=}} \exists g(g = a \wedge a = b) \wedge \exists ij(d = i \wedge e = j \wedge \exists hkl(\color{red}{i = h} \wedge j = k \wedge f = l \wedge c = h)) \wedge d = e$$



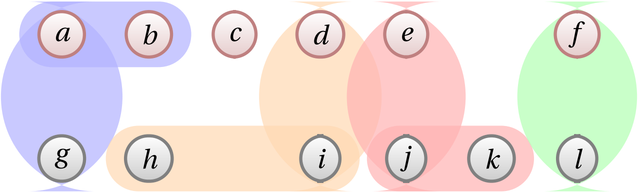
Build the Smallest Model Associated to a Constraint.

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



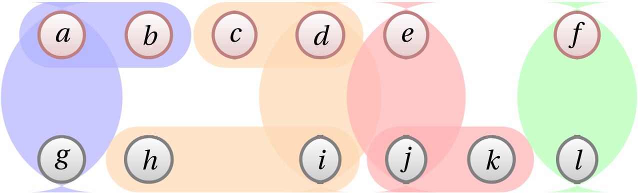
Build the Smallest Model Associated to a Constraint.

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



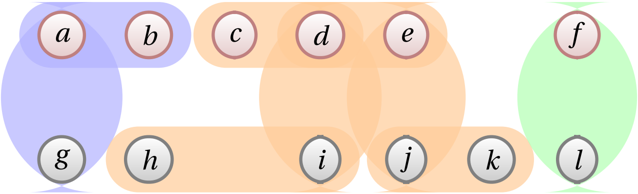
Build the Smallest Model Associated to a Constraint.

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



Build the Smallest Model Associated to a Constraint.

$$C \hat{=} \exists g(g = a \wedge a = b) \wedge \exists ij(d = i \wedge e = j \wedge \exists hkl(i = h \wedge j = k \wedge f = l \wedge c = h)) \wedge {\color{red} d = e}$$



Two op-codes

$$x \doteq y \text{ and } x \overset{?}{\doteq} y$$

Compiling Queries.

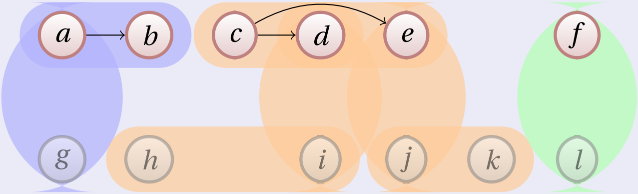
$$\mathscr{C}_0 ::= \mathbf{1} \mid \mathscr{C}_0 \wedge \mathscr{C}_0 \mid x = y$$

Proposition

For every $c \in \mathscr{C}$, there is a computable $[\![c]\!] \in \mathscr{C}_0$ such that $c \dashv\vdash [\![c]\!]$.

Proof.

Computation. $c \dashv\vdash a = b \wedge c = d \wedge c = e$



$c \dashv\vdash [\![c]\!]$. c and $[\![c]\!]$ have the same models.

□

Compiling Queries.

$$\mathscr{C}_0 ::= \mathbf{1} \mid \mathscr{C}_0 \wedge \mathscr{C}_0 \mid x = y$$

Proposition

For every $c \in \mathscr{C}$, there is a computable $[\![c]\!] \in \mathscr{C}_0$ such that $c \dashv\vdash [\![c]\!]$.

Proof.

Computation. Let $\mathfrak{P}$ be the smallest model of c.

$$[\![c]\!] \mathrel{\hat{=}} \bigwedge_{\substack{P \in \mathfrak{P} \\ \#(P \cap \mathrm{fv}(c)) \geqslant 2 \\ \{x\} \uplus V \mathrel{\hat{=}} P \cap \mathrm{fv}(c)}} \bigwedge_{y \in V} x = y$$

$c \dashv\vdash [\![c]\!]$. c and $[\![c]\!]$ have the same models.

□

Extension to Rational Terms.

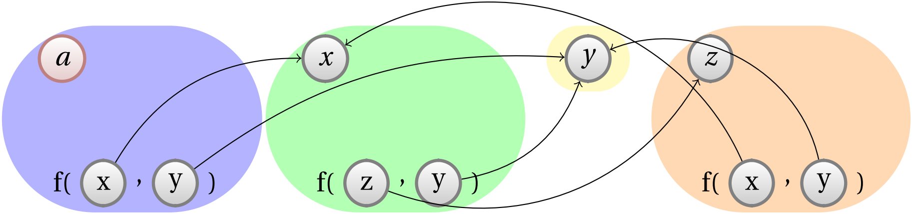
Constraints $\mathscr{C} ::= \underbrace{\mathbf{1} \mid \mathscr{C} \wedge \mathscr{C} \mid \exists x(\mathscr{C})}_{\text{common to all constraint systems}} \mid x = y \mid x = f(\overrightarrow{y})$

New Op-codes Labelled equivalence classes.

- $\mathrm{struct}(x, f(\overrightarrow{y}))$ to label an equivalence class, possible previous label must match and equality constraints on arguments are added.
- $\overrightarrow{y} \leftarrow \mathrm{struct}_?(x, f)$ to check that x has the functor f, then extract arguments.

Build the Code Associated to a Rational Tree Query.

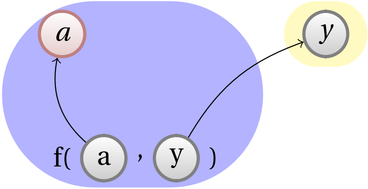
$$C \hat{=} \exists xyz(a = f(x,y) \wedge x = f(z,y) \wedge z = f(x,y))$$



1. $(x, y) \leftarrow \text{struct}_?(a, f)$
2. $(z, y_1) \leftarrow \text{struct}_?(x, f)$
3. $y_1 \stackrel{?}{=} y$
4. $(x_1, y_2) \leftarrow \text{struct}_?(z, f)$
5. $x_1 \stackrel{?}{=} x$
6. $y_2 \stackrel{?}{=} y$

State Minimization.

$$C \mathrel{\hat{=}} \exists xyz(a = f(x,y) \land x = f(z,y) \land z = f(x,y))$$



1. $(a_1, y) \leftarrow \text{struct}_?(a, f)$
2. $a_1 \stackrel{?}{=} a$

Equivalence Classes as Variable Trees.

Op-codes implementation:

$x \doteq y$ “link(find(x), find(y))”

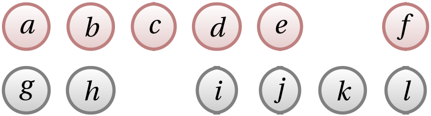
$x \overset{?}{\doteq} y$ check whether “find(x) = find(y)”

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.

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



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:

$x \doteq y$ "link(find(x), find(y))"

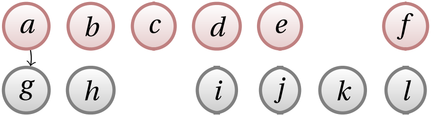
$x \overset{?}{\doteq} y$ check whether "find(x) = find(y)"

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.

$$C \mathrel{\hat{=}} \exists g(g = a \wedge b = g) \wedge \exists ij(d = i \wedge e = j \wedge \exists hkl(i = h \wedge j = k \wedge f = l \wedge c = h)) \wedge d = e$$



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:

$x \doteq y$ "link(find(x), find(y))"

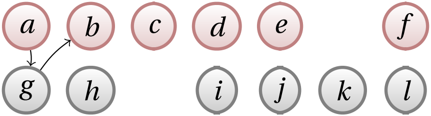
$x \overset{?}{\doteq} y$ check whether "find(x) = find(y)"

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.

$$C \mathrel{\hat{=}} \exists g(g = a \wedge b = g) \wedge \exists ij(d = i \wedge e = j \wedge \exists hkl(i = h \wedge j = k \wedge f = l \wedge c = h)) \wedge d = e$$



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:

$x \doteq y$ "link(find(x), find(y))"

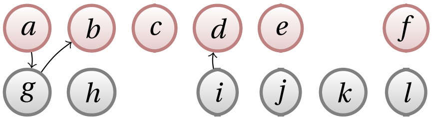
$x \stackrel{?}{\doteq} y$ check whether "find(x) = find(y)"

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.

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



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:

$x \doteq y$ "link(find(x), find(y))"

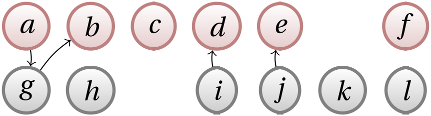
$x \overset{?}{\doteq} y$ check whether "find(x) = find(y)"

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.

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



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:

$x \doteq y$ "link(find(x), find(y))"

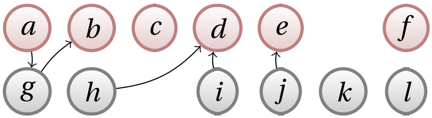
$x \overset{?}{\doteq} y$ check whether "find(x) = find(y)"

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.

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



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:

$x \doteq y$ “link(find(x), find(y))”

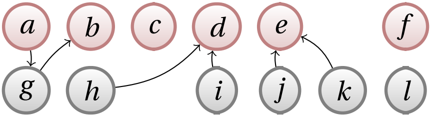
$x \stackrel{?}{\doteq} y$ check whether “find(x) = find(y)”

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.

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



Proposition

For a sequence of m operations, among which n are “link” operations, the total complexity is $\mathbf{\Theta}(mn)$.

Equivalence Classes as Variable Trees.

Op-codes implementation:

$x \doteq y$ "link(find(x), find(y))"

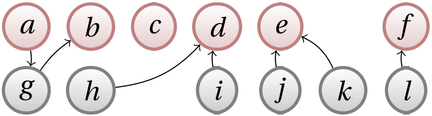
$x \stackrel{?}{\doteq} y$ check whether "find(x) = find(y)"

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.

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



Proposition

For a sequence of m operations, among which n are "link" operations, the total complexity is $\mathbf{\Theta}(mn)$.

Equivalence Classes as Variable Trees.

Op-codes implementation:

$x \doteq y$ "link(find(x), find(y))"

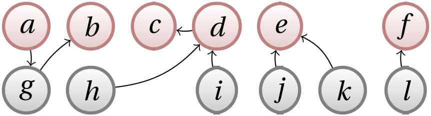
$x \overset{?}{\doteq} y$ check whether "find(x) = find(y)"

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.

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



Proposition

For a sequence of m operations, among which n are "link" operations, the total complexity is $\boldsymbol{\Theta}(mn)$.

Equivalence Classes as Variable Trees.

Op-codes implementation:

$x \doteq y$ "link(find(x), find(y))"

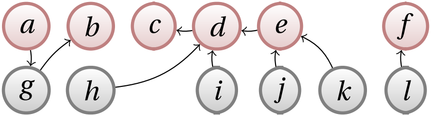
$x \overset{?}{\doteq} y$ check whether "find(x) = find(y)"

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.

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



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:

$x \doteq y$ “link(find(x), find(y))”

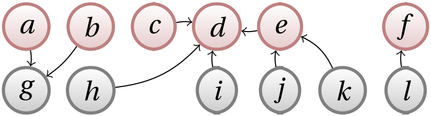
$x \stackrel{?}{\doteq} y$ check whether “find(x) = find(y)”

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.

$$C \mathrel{\hat{=}} \exists g(g = a \wedge g = b) \wedge \exists ij(d = i \wedge e = j \wedge \exists hkl(i = h \wedge j = k \wedge f = l \wedge h = c)) \wedge d = e$$



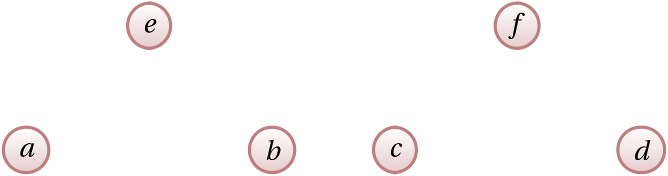
Proposition

For a sequence of m operations, among which n are “link” operations, the total complexity is $\boldsymbol{\Theta}(mn)$.

Heuristic 1: Union by Rank.

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

$$C \hat{=} a = e \wedge b = e \wedge c = f \wedge d = f \wedge f = e$$



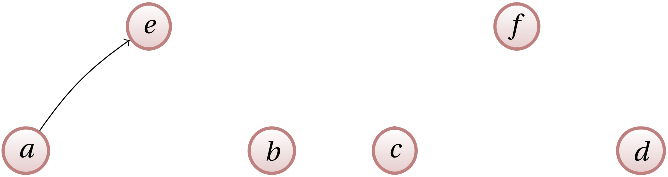
Proposition

For a sequence of m operations, among which n are "link" operations, the total complexity is $\mathbf{\Theta}(m \log n)$.

Heuristic 1: Union by Rank.

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

$$C \hat{=} a = e \wedge b = e \wedge c = f \wedge d = f \wedge f = e$$



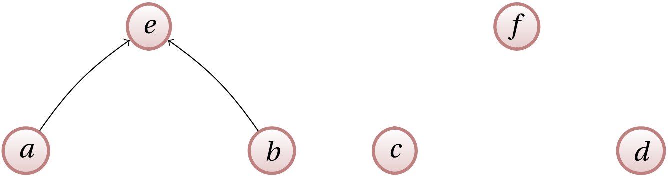
Proposition

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.

$$C \mathrel{\hat{=}} a = e \wedge \color{red}{b = e} \wedge c = f \wedge d = f \wedge f = e$$



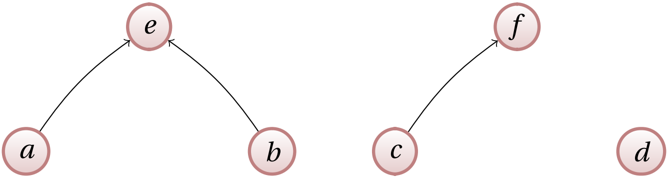
Proposition

For a sequence of m operations, among which n are "link" operations, the total complexity is $\mathbf{\Theta}(m \log n)$.

Heuristic 1: Union by Rank.

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

$$C \mathrel{\hat{=}} a = e \land b = e \land c = f \land d = f \land f = e$$



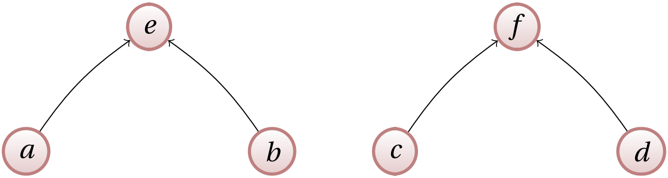
Proposition

For a sequence of m operations, among which n are "link" operations, the total complexity is $\boldsymbol{\Theta}(m \log n)$.

Heuristic 1: Union by Rank.

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

$$C \mathrel{\hat{=}} a = e \wedge b = e \wedge c = f \wedge d = f \wedge f = e$$



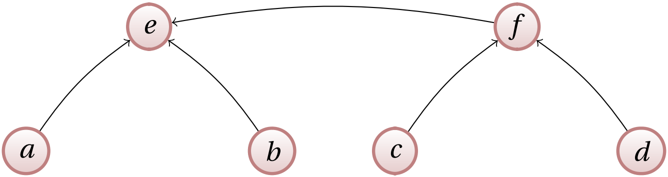
Proposition

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.

$$C \hat{=} a = e \wedge b = e \wedge c = f \wedge d = f \wedge f = e$$



Proposition

For a sequence of m operations, among which n are "link" operations, the total complexity is $\Theta(m \log n)$.

Heuristic 2: Path Compression.

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

$$C \mathrel{\hat{=}} b = a \wedge c = a \wedge d = a \wedge e = a \wedge f = a$$

Proposition

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

Heuristic 2: Path Compression.

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

$$C \mathrel{\hat{=}} \color{red}{b = a} \wedge c = a \wedge d = a \wedge e = a \wedge f = a$$

Proposition

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

Heuristic 2: Path Compression.

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

$$C \mathrel{\hat{=}} b = a \wedge \color{red}{c = a} \wedge d = a \wedge e = a \wedge f = a$$

Proposition

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

Heuristic 2: Path Compression.

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

$$C \triangleq b = a \wedge c = a \wedge \color{red}{d = a} \wedge e = a \wedge f = a$$

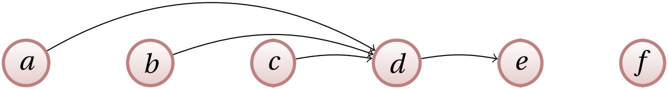
Proposition

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

Heuristic 2: Path Compression.

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

$$C \hat{=} b = a \wedge c = a \wedge d = a \wedge e = a \wedge f = a$$



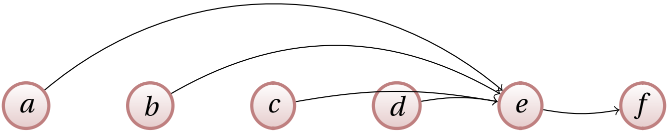
Proposition

For a sequence of f "find" operations and n "link" operations, the total complexity is $\mathbf{O}(n + f \cdot (1 + \log_{2+f/n} n))$.

Heuristic 2: Path Compression.

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

$$C \hat{=} b = a \wedge c = a \wedge d = a \wedge e = a \wedge {\color{red} f = a}$$



Proposition

For a sequence of f "find" operations and n "link" operations, the total complexity is $\mathbf{O}(n + f \cdot (1 + \log_{2+f/n} n))$.

A very quickly growing function…

Definition

For integers $m, n \geq 0$:

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

m	0	1	2	3	4	5
$A(m,1)$	2	3	5	13	65533	$\underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}_{65536} - 3 \gg 10^{80}$

…and its very slowly growing inverse.

Definition

For $n \geqslant 0$:

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

For all practical purposes

$$\alpha(n) \leqslant 5$$

Quasi-linear complexity.

Proposition

For a sequence of m operations, among which n are "link" operations, the total complexity is $\mathbf{O}(m\alpha(n))$.

cc(=)

Program

$$\mathscr{P} ::= \epsilon \mid p(\overrightarrow{x}) \text{ :- } \mathscr{A} \,.\, \mathscr{P}$$

$$\mathscr{A} ::= (\mathscr{A} \parallel \mathscr{A}) \mid \exists x(\mathscr{A}) \mid p(\overrightarrow{x}) \mid \text{tell}(\mathscr{C}) \mid \underbrace{\forall \overrightarrow{x}\,(\mathscr{C} \rightarrow \mathscr{A})}_{(\exists \overrightarrow{x}\,(c) \rightarrow \exists \overrightarrow{x}\,(\text{tell}(c) \parallel a)}$$

New Op-codes for Translating cc(=)

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

freeze($x \overset{?}{\doteq} y$, Code) Ask on atomic constraints.

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

$$\rightsquigarrow \mathrm{freeze}(a \overset{?}{\doteq} b, \mathrm{freeze}(b \overset{?}{\doteq} c, [\![A]\!]))$$

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 $(\mathrm{do}(f_1) \to a_1) \| \ldots \| (\mathrm{do}(f_n) \to a_n)$). If implemented naively, $\mathrm{tell}(\mathrm{do}(x))$ leads to freezes on $x \stackrel{?}{=} f_1, \ldots, x \stackrel{?}{=} f_n$.