Efficiency of a good but not linear nominal unification algorithm

Weixi Ma, Jeremy G. Siek, David Thrane Christiansen, Daniel P. Friedman

July 4, 2018

Nominal Unification [Urban et al., 2004]

$$\lambda\, a.\lambda\, b.X\; Y \stackrel{?}{=} \lambda\, a.\lambda\, a.a\, Z$$

Unifier:
$\{X/b,\; Y/(a \leftrightarrow b)\cdot Z\}$ and $\{b \# Z\}$

Nominal Unification [Urban et al., 2004]

- The fastest existing nominal unification algorithms are $O(n^2)$.
- The quadratic comes from swappings
  e.g., $X \stackrel{?}{=} (a_0 \leftrightarrow a'_0,\ \ldots,\ a_n \leftrightarrow a'_n)a$.
- Lists cannot be replaced by structures with better lookup efficiency, because of $\pi \cdot (\pi' \cdot X) \equiv (\pi @ \pi') \cdot X$.

To lookup or to append?

	LinkedList	Hashtable	?
lookup	$O(n)$	$O(1)$	$O(1)$
append	$O(1)$	$O(n)$	$O(1)$

It's straightforward to check α-equivalence

using de Bruijn indices [de Bruijn, 1972]

$$\lambda\, a.\lambda\, b.a \stackrel{\alpha}{=} \lambda\, c.\lambda\, d.c$$

In de Bruijn indices:

$$\lambda\,\lambda\, 1 \stackrel{\alpha}{=} \lambda\,\lambda\, 1$$

But in unification it's different

name shadowing is lost

$$\lambda a.\lambda a.X \stackrel{\alpha}{=} \lambda c.\lambda d.c$$

In de Bruijn indices:

$$\lambda \lambda X \stackrel{\alpha}{=} \lambda \lambda 1$$

De Bruijn indices should coexist with names

A *scope* is a list of names.
A *static closure* consists of a term and its enclosing scope.

term: $\lambda\, a.\lambda\, b.a$
closure: $\langle (b\, a); a\rangle$

To decide α-equivalence, we need Free and Bound

$$\frac{a \notin \Phi}{\Phi \vdash \texttt{Fr}\ a}\ [\textsc{Free}]$$

$$\frac{\begin{array}{l}(\texttt{name}{\rightarrow}\texttt{index}\,\Phi\, a) = i \\ (\texttt{index}{\rightarrow}\texttt{name}\,\Phi\, i) = a\end{array}}{\Phi \vdash \texttt{Bd}\ a\ i}\ [\textsc{Bound}]$$

The core of name management

$$\frac{\begin{array}{c} a_1 = a_2 \\ \Phi_1 \vdash \texttt{Fr}\ a_1 \\ \Phi_2 \vdash \texttt{Fr}\ a_2 \end{array}}{\langle \Phi_1; a_1 \rangle \approx \langle \Phi_2; a_2 \rangle}\ [\textsc{Same-Free}]$$

$$\frac{\begin{array}{c} i_1 = i_2 \\ \Phi_1 \vdash \texttt{Bd}\ a_1\ i_1 \\ \Phi_2 \vdash \texttt{Bd}\ a_2\ i_2 \end{array}}{\langle \Phi_1; a_1 \rangle \approx \langle \Phi_2; a_2 \rangle}\ [\textsc{Same-Bound}]$$

Unifying name-closures

$$\lambda a.\lambda b.a \stackrel{?}{=} \lambda c.\lambda d.c$$

$$\sigma \vdash \langle (b\,a); a \rangle \stackrel{\alpha}{=} \langle (d\,c); c \rangle \Rightarrow \sigma$$

By Same-Bound

Unifying name-closure versus var-closure

$$\lambda a.\lambda b.X \stackrel{?}{=} \lambda c.\lambda d.e$$

$$\sigma \vdash \langle (b\, a); X \rangle \stackrel{\alpha}{=} \langle (d\, c); e \rangle \Rightarrow \{X/e\} \cup \sigma$$

By Same-Free

Unifying name-closure versus var-closure

another example

$$\lambda a.\lambda a.X \stackrel{?}{=} \lambda c.\lambda d.c$$

$$\sigma \vdash \langle (a\,a); X \rangle \stackrel{\alpha}{=} \langle (d\,c); c \rangle \Rightarrow \bot$$

No rules apply

Unifying var-closures

$$\lambda a.\lambda b.X \stackrel{?}{=} \lambda c.\lambda d.Y$$

$$\sigma \vdash \langle (b\,a); X \rangle \stackrel{\alpha}{=} \langle (d\,c); Y \rangle \Rightarrow ?$$

σ is not enough.

Unifying var-closures

$$\lambda\, a.\lambda\, b.X \stackrel{?}{=} \lambda\, c.\lambda\, d.Y$$

$$\delta;\ \sigma \vdash \langle (b\, a); X \rangle \stackrel{\alpha}{=} \langle (d\, c); Y \rangle \Rightarrow \delta';\ \sigma$$

$$\delta' = \{\langle (b\, a); X \rangle \stackrel{\alpha}{=} \langle (d\, c); Y \rangle\} \cup \delta$$

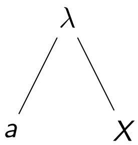
To sum up:

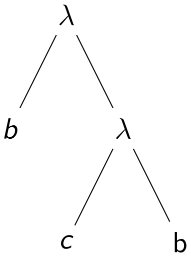
- De Bruijn indices coexist with names in closures.
- σ is a substitution.
- δ contains unsolved pairs of var-closures.
- We have solved the simple case where unification variables have only been instantiated with names.
- Next, let's look at $X \stackrel{?}{=} \lambda\, a.Y$ and $X \stackrel{?}{=} Y\, Z$.

To generalize the algorithm:

Need: a shared shape for two arbitrary terms

$$\lambda\, a.X \stackrel{?}{=} \lambda\, b.\lambda\, c.b$$





To generalize the algorithm:

Need: a shared shape for two arbitrary terms

$$\lambda\, a.X \stackrel{?}{=} \lambda\, b.\lambda\, c.b$$

Also, $\sigma = \{X/\lambda\, c'.Y\} \cup \sigma$

Replacing lists with hashtables

$$\lambda\, a.\lambda\, b.\lambda\, b.\,\lambda\, c\ldots$$

indices with list:
$(c\; b\; b\; a)$

levels with two hashtables:

$$\begin{array}{lcl} a \mapsto 0 & \quad & 0 \mapsto a \\ b \mapsto 1 & & 1 \mapsto b \\ b \mapsto 2 & & 2 \mapsto b \\ c \mapsto 3 & & 3 \mapsto c \end{array}$$

Where Φ changes

$$\frac{\delta_0; \sigma_0 \vdash \langle a_1, \Phi_1; t_1 \rangle \stackrel{\alpha}{=} \langle a_2, \Phi_2; t_2 \rangle \Rightarrow \delta_1; \sigma_1}{\delta_0; \sigma_0 \vdash \langle \Phi_1; \lambda\, a_1 . t_1 \rangle \stackrel{\alpha}{=} \langle \Phi_2; \lambda\, a_2 . t_2 \rangle \Rightarrow \delta_1; \sigma_1}\ [\textsc{Abs}]$$

$$\frac{\begin{array}{c}\delta_0; \sigma_0 \vdash \langle \Phi_1; l_1 \rangle \stackrel{\alpha}{=} \langle \Phi_2; l_2 \rangle \Rightarrow \delta_0'; \sigma_0' \\ \delta_0'; \sigma_0' \vdash \langle \Phi_1; r_1 \rangle \stackrel{\alpha}{=} \langle \Phi_2; r_2 \rangle \Rightarrow \delta_1; \sigma_1\end{array}}{\delta_0; \sigma_0 \vdash \langle \Phi_1; l_1\ r_1 \rangle \stackrel{\alpha}{=} \langle \Phi_2; l_2\ r_2 \rangle \Rightarrow \delta_1; \sigma_1}\ [\textsc{App}]$$

Lookup only

	LinkedList	Ordinary Hashtable	Persistent Hashtable
lookup	$O(n)$	$O(1)$	$O(log(n))$
append	$O(1)$	$O(n)$	whatever

Conclusion

- Replacing swappings by scopes
- Replacing the need of composing by the need of sharing
- An $O(n \cdot G(n) \cdot log(n))$ nominal unification algorithm

Primary references

Nicolaas G. de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. *Indagationes Mathematicae (Proceedings)*, 75(5):381–392, January 1972.

Christian Urban, Andrew M. Pitts, and Murdoch J. Gabbay. Nominal unification. *Theoretical Computer Science*, 323(1-3): 473–497, September 2004.

Thank you!