Nameless Representation of Terms

CIS500: Software Foundations

First, some review . . .

A Proof on λ-Terms

Proof (1)

We want to prove that if

$$z \in FV([x \mapsto v]u)$$

then

$$z \in (FV(u) \setminus \{x\}) \cup FV(v)$$

In other words,

$$FV([x \mapsto v]u) \subseteq (FV(u) \setminus \{x\}) \cup FV(v)$$

Proof by induction on the structure of u.

Proof (2)

- Case $u = x$: Then $[x \mapsto v]u = v$, and

$$FV(v) \subseteq FV(u) \setminus \{x\} \cup FV(v)$$

- Case $u = y$, where $y \neq x$: Then $[x \mapsto v]u = y$, and

$$\begin{array}{rcl} FV(u) & = & FV(y) \\ & = & \{y\} \\ & \subseteq & (\{y\} \setminus \{x\}) \cup FV(v) \\ & = & (FV(u) \setminus \{x\}) \cup FV(v) \end{array}$$

Proof (3)

- Case $u = \lambda y.\ t$, where $y \neq x$: Then

$$[x \mapsto v]u = \lambda y.\ [x \mapsto v]t$$

By the IH, $FV([x \mapsto v]t) \subseteq (FV(t) \setminus \{x\}) \cup FV(v)$. So

$$\begin{array}{rcl} FV([x \mapsto v]u) & = & FV(\lambda y.\ [x \mapsto v]\ t) \\ & = & FV([x \mapsto v]\ t) \setminus \{y\} \\ & \subseteq & ((FV(t) \setminus \{x\}) \cup FV(v)) \setminus \{y\} \\ & \subseteq & (FV(t) \setminus \{x\} \setminus \{y\}) \cup FV(v) \\ & = & (FV(t) \setminus \{y\} \setminus \{x\}) \cup FV(v) \\ & = & (FV(u) \setminus \{x\}) \cup FV(v) \end{array}$$

Proof (4)

- Case $u = t_1\ t_2$: Exercise.

Now on to the main topic . . .

Nameless Representation of Terms

Representing Terms

$$
\begin{array}{rcl}
t & ::= & x \\
  & | & \lambda x.\, t \\
  & | & t_1\ t_2
\end{array}
$$

Choosing a concrete way to represent terms is necessary when using computers to work with λ-terms.

- Implementing programming language evaluators.
- Writing machine-checkable definitions and proofs of theorems.

Variable Capture

$$[x \mapsto \lambda y.\ z](\lambda z.\ x) \neq \lambda z.\ \lambda y.\ z$$

How can we be sure that our implementation doesn't make this mistake?

Idea: Rename During Substitution

Rename z to z' before applying substitution.

$$[x \mapsto \lambda y.\ z](\lambda z.\ x) = \lambda z'.\ \lambda y.\ z$$

Idea: “Barendregt Convention”

We can make sure our terms never use the same variable name twice. So we must always start with

$$[x \mapsto \lambda y.\ z](\lambda z'.\ x)$$

Idea: “Barendregt Convention”

We can make sure our terms never use the same variable name twice. So we must always start with

$$[x \mapsto \lambda y.\ z](\lambda z'.\ x)$$

But then what happens here?

$$[x \mapsto \lambda y.\ z](\lambda z.\ x\ x)$$

More Extreme Proposals

- Explicit Substitutions: Make substitutions part of the syntax and encode renaming into the evaluation rules.
- Combinators: Find a language with applications but no variables or binding, and translate terms to this langauge.

Devise Canonical Representation

Maybe we can think of a unique representation for α-equivalent terms.

Devise Canonical Representation

Maybe we can think of a unique representation for α-equivalent terms.
For

$$\lambda x.\ \lambda y.\ x\ (y\ x)$$

we could write

$$\lambda.\ \lambda.\ 1\ (0\ 1)$$

- Is this representation unique?

Devise Canonical Representation

Maybe we can think of a unique representation for α-equivalent terms.
For

$$\lambda x.\ \lambda y.\ x\ (y\ x)$$

we could write

$$\lambda.\ \lambda.\ 1\ (0\ 1)$$

- Is this representation unique?
- What about free variables?

Formal Definition of de Bruijn Terms

We will define a family of sets $\mathcal{T}_n$ so that the set $\mathcal{T}_i$ can represent terms with at most i free variables.

$$\frac{0 \leq k < n}{k \in \mathcal{T}_n} \qquad \frac{t \in \mathcal{T}_n \quad n > 0}{\lambda.t \in \mathcal{T}_{n-1}}$$

$$\frac{t_1 \in \mathcal{T}_n \quad t_2 \in \mathcal{T}_n}{(t_1\ t_2) \in \mathcal{T}_n}$$

Free Variables

What do we do with y?

$$\lambda x.\ y\ x$$

Free Variables

What do we do with y?

$$\lambda x.\ y\ x$$

We need some sort of context of definitions, for example

$$\Gamma = x \mapsto 4, y \mapsto 3, z \mapsto 2, a \mapsto 1, b \mapsto 0$$

Then we should be able to define a function db_Γ, such that

$$db_\Gamma(x\ (y\ z)) = 4\ (3\ 2) \qquad db_\Gamma(\lambda x.\ y\ x) = \lambda.\ 4\ 0$$

Naming Contexts

Let's simplify Γ to be a sequence of variable names.

$$\Gamma = x_{n-1}, \ldots, x_1, x_0$$

Then we'll define

$$dom(\Gamma) = \{x_{n-1}, \ldots, x_1, x_0\}$$

And

$$\Gamma(x) = \text{rightmost index of } x \text{ in } \Gamma$$

Converting to Nameless Representation

$$
\begin{array}{rcl}
db_\Gamma(x) & = & \Gamma(x) \\
db_\Gamma(\lambda x.\ t) & = & \lambda . db_{\Gamma,x}(t) \\
db_\Gamma(t_1\ t_2) & = & db_\Gamma(t_1)\ db_\Gamma(t_2)
\end{array}
$$

Converting to Nameless Representation

$$
\begin{array}{rcl}
db_\Gamma(x) & = & \Gamma(x) \\
db_\Gamma(\lambda x.\ t) & = & \lambda.db_{\Gamma,x}(t) \\
db_\Gamma(t_1\ t_2) & = & db_\Gamma(t_1)\ db_\Gamma(t_2)
\end{array}
$$

What is the type of db_Γ?

Converting to Nameless Representation

$$
\begin{array}{rcl}
db_\Gamma(x) & = & \Gamma(x) \\
db_\Gamma(\lambda x.\ t) & = & \lambda . db_{\Gamma,x}(t) \\
db_\Gamma(t_1\ t_2) & = & db_\Gamma(t_1)\ db_\Gamma(t_2)
\end{array}
$$

What is the type of db_Γ?

$$db_\Gamma : \mathcal{T}_\lambda \rightarrow \mathcal{T}_{len(\Gamma)}$$

Conversion Example

We will work with $\Gamma = x, y, z$ and will convert the term

$$\lambda x.\ y\ x$$

Then we have

$$\begin{array}{rcl} db_{x,y,z}(\lambda x.\ y\ x) & = & \lambda.\ db_{x,y,z,x}(y\ x) \\ & = & \lambda.\ db_{x,y,z,x}(y)\ db_{x,y,z,x}(x) \\ & = & \lambda.\ 2\ 0 \end{array}$$

Defining Substitution

Substitution on Nameless Terms

We must define

$$[k \mapsto s]t$$

for terms in $\mathcal{T}_n$. But how?

Substitution on Nameless Terms

We must define

$$[k \mapsto s]t$$

for terms in $\mathcal{T}_n$. But how? We want to guarantee

$$db_\Gamma([x \mapsto s]t) = [\Gamma(x) \mapsto db_\Gamma(s)]db_\Gamma(t)$$

for all Γ such that

$$FV(s) \cup FV(t) \cup \{x\} \subseteq dom(\Gamma)$$

First Attempt

$$
\begin{aligned}
[j \mapsto s]k &= \begin{cases} s & \text{if } k = j \\ k & \text{otherwise} \end{cases} \\
[j \mapsto s](\lambda.t) &= \lambda.\, [j \mapsto s]t \\
[j \mapsto s](t_1\ t_2) &= ([j \mapsto s]t_1)\ ([j \mapsto s]t_2)
\end{aligned}
$$

Counter-Example

$$[x \mapsto \lambda z.\ z](x\ (\lambda y.\ y))$$

Let $\Gamma = x$.

$$\begin{aligned} db_\Gamma([x \mapsto \lambda z.\ z](x\ (\lambda y.\ y))) &= db_\Gamma((\lambda z.\ z)\ (\lambda y.\ y)) \\ &= (\lambda.\ 0)\ (\lambda.\ 0) \end{aligned}$$

but

$$\begin{aligned} [\Gamma(x) \mapsto db_\Gamma(\lambda z.\ z)]db_\Gamma(x\ (\lambda y.\ y)) &= [0 \mapsto \lambda.\ 0](0\ (\lambda.0)) \\ &= (\lambda.\ 0)\ (\lambda.\ [0 \mapsto \lambda.\ 0]0) \\ &= (\lambda.\ 0)\ (\lambda.\ \lambda.\ 0) \end{aligned}$$

Second Attempt

$$
\begin{aligned}
[j \mapsto s]k &= \begin{cases} s & \text{if } k = j \\ k & \text{otherwise} \end{cases} \\
[j \mapsto s](\lambda.t) &= \lambda.\ [j+1 \mapsto s]t \\
[j \mapsto s](t_1\ t_2) &= ([j \mapsto s]t_1)\ ([j \mapsto s]t_2)
\end{aligned}
$$

Counter-Example

$$[x \mapsto \lambda y.\ w]\lambda z.\ x$$

Let $\Gamma = x, w$.

$$\begin{aligned} db_\Gamma([x \mapsto \lambda y.\ w]\lambda z.\ x) &= db_\Gamma(\lambda z.\ \lambda y.\ w) \\ &= \lambda.\ db_{\Gamma,z}(\lambda y.\ w) \\ &= \lambda.\ \lambda.\ db_{\Gamma,z,y}(w) \\ &= \lambda.\ \lambda.\ 2 \end{aligned}$$

but

$$\begin{aligned} [\Gamma(x) \mapsto db_\Gamma(\lambda y.\ w)]db_\Gamma(\lambda z.\ x) &= [1 \mapsto \lambda.\ 1]\lambda.\ 2 \\ &= \lambda.\ [2 \mapsto \lambda.\ 1]2 \\ &= \lambda.\ \lambda.\ 1 \end{aligned}$$

Third Attempt (Shifting)

$$\begin{aligned}
\uparrow(k) &= k+1 \\
\uparrow(\lambda.\ t) &= \lambda.\ \uparrow(t) \\
\uparrow(t_1\ t_2) &= \uparrow(t_1)\ \uparrow(t_2)
\end{aligned}$$

$$\begin{aligned}
[j \mapsto s]k &= \begin{cases} s & \text{if } k = j \\ k & \text{otherwise} \end{cases} \\
[j \mapsto s](\lambda.t) &= \lambda.\ [j+1 \mapsto \uparrow(s)]t \\
[j \mapsto s](t_1\ t_2) &= ([j \mapsto s]t_1)\ ([j \mapsto s]t_2)
\end{aligned}$$

Counter-Example

$$[x \mapsto \lambda y.\ w\ y]\lambda z.\ x$$

Let $\Gamma = x, w$.

$$\begin{aligned} db_\Gamma([x \mapsto \lambda y.\ w\ y]\lambda z.\ x) &= db_\Gamma(\lambda z.\ \lambda y.\ w\ y) \\ &= \lambda.\ \lambda.\ db_{\Gamma,z,y}(w\ y) \\ &= \lambda.\ \lambda.\ 2\ 0 \end{aligned}$$

but

$$\begin{aligned} [\Gamma(x) \mapsto db_\Gamma(\lambda y.\ w\ y)]db_\Gamma(\lambda z.\ x) &= [1 \mapsto \lambda.\ 1\ 0]\lambda.\ 2 \\ &= \lambda.\ [2 \mapsto \uparrow (\lambda.\ 1\ 0)]2 \\ &= \lambda.\ [2 \mapsto \lambda.\ 2\ 1]2 \\ &= \lambda.\ \lambda.\ 2\ 1 \end{aligned}$$

Third Attempt (Shifting with Cut-Off)

$$
\begin{aligned}
\uparrow_c (k) &= \begin{cases} k & \text{if } k < c \\ k+1 & \text{if } k \geq c \end{cases} \\
\uparrow_c (\lambda.\ t) &= \lambda.\ \uparrow_{c+1} (t) \\
\uparrow_c (t_1\ t_2) &= \uparrow_c (t_1)\ \uparrow_c (t_2)
\end{aligned}
$$

$$
\begin{aligned}
[j \mapsto s]k &= \begin{cases} s & \text{if } k = j \\ k & \text{otherwise} \end{cases} \\
[j \mapsto s](\lambda.t) &= \lambda.\ [j+1 \mapsto \uparrow_0 (s)]t \\
[j \mapsto s](t_1\ t_2) &= ([j \mapsto s]t_1)\ ([j \mapsto s]t_2)
\end{aligned}
$$

Generalized Shifting

$$
\begin{aligned}
\uparrow_c^d (k) &= \begin{cases} k & \text{if } k < c \\ k + d & \text{if } k \geq c \end{cases} \\
\uparrow_c^d (\lambda.\, t) &= \lambda.\ \uparrow_{c+1}^d (t) \\
\uparrow_c^d (t_1\ t_2) &= \uparrow_c^d (t_1)\ \uparrow_c^d (t_2)
\end{aligned}
$$

Evaluation of de Bruijn Terms

The evaluation rule we want is

$$\frac{}{(\lambda.\ t_{12})\ v_2 \longrightarrow \uparrow_0^{-1} ([0 \mapsto \uparrow_0^1 (v_2)]t_{12})} \quad \textsc{E-AppAbs}$$

Evaluation of de Bruijn Terms

The evaluation rule we want is

$$\frac{}{(\lambda.\ t_{12})\ v_2 \longrightarrow \uparrow_0^{-1} ([0 \mapsto \uparrow_0^1 (v_2)]t_{12})} \quad \text{E-AppAbs}$$

Consider this example. Let's say our context is $\Gamma = z, y, x$.

$$\begin{array}{rcl}
db_\Gamma((\lambda w.\ w\ x\ y)\ x\ y\ z) & = & (\lambda.\ 0\ 1\ 2)\ 0\ 1\ 2 \\
& \longrightarrow & (\uparrow_0^{-1} ([0 \mapsto \uparrow_0^1 (0)](0\ 1\ 2)))\ 1\ 2 \\
& = & (\uparrow_0^{-1} ([0 \mapsto 1](0\ 1\ 2)))\ 1\ 2 \\
& = & (\uparrow_0^{-1} (1\ 1\ 2))\ 1\ 2 \\
& = & 0\ 0\ 1\ 1\ 2 \\
& = & db_\Gamma(x\ x\ y\ y\ z)
\end{array}$$