Interpreting and Typechecking Simply Typed Lambda Calculus!

Solomon Bothwell

2020-03-02

Untyped Lambda Calculus

A formal system for expressing computation using function abstraction, application, and reduction.

Syntax

- Variables: x
- Lambda Abstractions: $\lambda x.x$
- Application: $t_1 t_2$

Reduction

- Alpha Conversion (name changes)
- Beta Reduction (applying functions to their arguments)
- Eta Reduction (reducing abstraction around a function)

Example Terms

- $id := \lambda x.x$
- $const := \lambda a.\lambda b.a$
- $Z := \lambda f.\lambda x.x$
- $S\,Z := \lambda f.\lambda x.f\ x$
- $S\,S\,Z := \lambda f.\lambda x.f\ f\ x$
- $True := \lambda p.\lambda q.p$
- $False := \lambda p.\lambda q.q$
- $Not := \lambda p.p\,False\,True$
- $Or := \lambda p.\lambda q.p\,p\,q$

Sample reduction

Not True

$$(\lambda p.p(\lambda p.\lambda q.q)(\lambda p.\lambda q.p))\,(\lambda p.\lambda q.p)$$
$$(\lambda p.p(\lambda x.\lambda y.y)(\lambda f.\lambda g.f))\,(\lambda a.\lambda b.a)$$
$$(\lambda a.\lambda b.a)(\lambda x.\lambda y.y)\,(\lambda f.\lambda g.f)$$
$$(\lambda b.(\lambda x.\lambda y.y))\,(\lambda f.\lambda g.f)$$
$$\lambda x.\lambda y.y$$

How powerful is this?

Very powerful. Fully isomorphic to turning machines.

Evaluation Semantics

Operational Semantics - Intensional | How

Create an abstract state machine consisting of terms as state and reduction rules for terms which can be followed in sequence to reach some halting state.

Denotational Semantics - Extensional | What

Create a mapping to a mathematical domain that denotes the meanings of terms.

Small Step/Big Step

Operational Semantics comes in two flavors:

- Small Step: describe individual steps of computation.
- Big Step: describe the overall result of execution.

Untyped Lambda Calculus

Syntax

$$
\begin{aligned}
t :=\ & x \\
& \lambda x.t \\
& t_1\ t_2 \\
v :=\ & \lambda x.t
\end{aligned}
$$

Evaluation

$$\frac{t_1 \rightarrow t_1'}{t_1\, t_2 \rightarrow t_1'\, t_2} \text{ E-App1}$$

$$\frac{t_2 \rightarrow t_2'}{v_1\, t_2 \rightarrow v_1\, t_2'} \text{ E-App2}$$

$$(\lambda x.t_{12})v_2 \rightarrow [x \mapsto v_2]t_{12} \text{ E-AppAbs}$$

Substitution Rules

- $[x \mapsto s]x = s$
- $[x \mapsto s]y = y$
- $[x \mapsto s](\lambda y.t_1) = \lambda y.[x \mapsto s]t_1$
- $[x \mapsto s](t_1 t_2) = ([x \mapsto s]t_1)\,([x \mapsto s]t_2)$

Avoiding Name Collisions In Substitution

Given Expression:

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

Avoiding Name Collisions In Substitution

Given Expression:

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

Evaluation Rule: E-AppAbs

$(\lambda x.t_{12})v_2 \to [x \mapsto v_2]t_{12}$

Avoiding Name Collisions In Substitution

Given Expression:

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

Evaluation Rule:

$E - AppAbs: (\lambda x.t_{12})v_2 \rightarrow [x \mapsto v_2]t_{12}$

Our substitution:

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

Avoiding Name Collisions In Substitution

Given Expression:

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

Evaluation Rule:

$E - AppAbs : (\lambda x.t_{12})v_2 \to [x \mapsto v_2]t_{12}$

Our substitution:

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

Our Desired Final Value:

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

Avoiding Name Collisions In Substitution

How do we perform this substitution without capturing free variables?

Avoiding Name Collisions In Substitution

How do we perform this substitution without capturing free variables?

Two Options:

1. Identify free variables and use Alpha Conversion prevent shadowing
2. Convert our Lambda Terms to Nameless Form using DeBruijn Indices.

Avoiding Name Collisions In Substitution

Capture Avoiding Substitution

1. Given the substitution $[x \mapsto v_2]t_{12}$
2. Identiy all the bound variables in t_{12}.
3. Rename all bound variables inside t_{12} with *fresh* variables.
4. Perform the substitution of v_2 for x in t_{12}.

Nameless Form (DeBruijn Indices)

In nameless form variable names are replaced by natural numbers representing the number of lambda abstractions between the variable and its binder.
Examples:

- $\lambda x.\lambda y.x \longrightarrow \lambda\lambda 1$
- $\lambda x.\lambda y.x \longrightarrow \lambda\lambda 0$
- $(\lambda x.\lambda y.(\lambda x.x)\, y\, x)\,(\lambda y.y\,(\lambda x.x)) \longrightarrow (\lambda\lambda(\lambda 0)\, 0\, 1)\,(\lambda 0(\lambda 0))$

A haskell implementation

```haskell
data Term = Var String
          | Abs String Term
          | App Term Term

singleEval :: Term -> Maybe Term
singleEval t =
  case t of
    (App (Abs x t12) v2) | isVal v2 -> Just $ subst x v2 t12
    (App v1@(Abs _ _) t2)           ->      App v1 <$> singleEval t2
    (App t1 t2)                     -> flip App t2 <$> singleEval t1
    _ -> Nothing

multiStepEval :: Term -> Term
multiStepEval t = maybe t multiStepEval (singleEval t)
```

Simply Typed Lambda Calculus

Syntax

$$
\begin{aligned}
t &::= x \\
&\quad \lambda x : T . t \\
&\quad t_1 \; t_2 \\
v &::= \lambda x : T . t \\
T &::= T \to T \\
\Gamma &::= \varnothing \\
&\quad \Gamma, x : T
\end{aligned}
$$

Evaluation

$$\frac{t_1 \to t_1'}{t_1 \, t_2 \to t_1' \, t_2} \; \text{E-App1}$$

$$\frac{t_2 \to t_2'}{v_1 \, t_2 \to v_1 \, t_2'} \; \text{E-App2}$$

$$(\lambda x : T_{11} . t_{12}) v_2 \to [x \mapsto v_2] t_{12} \; \text{E-AppAbs}$$

Typing

$$\frac{x : T \in \Gamma}{\Gamma \vdash x : T} \; \text{T-Var}$$

$$\frac{\Gamma, x : T_1 \vdash t_2 : T_2}{\Gamma \vdash \lambda x : T_1 . t_2 : T_1 \to T_2} \; \text{T-Abs}$$

$$\frac{\Gamma \vdash t_1 : T_{11} \to T_{12} \qquad \Gamma \vdash t_2 : T_{11}}{\Gamma \vdash t_1 \, t_2 : T_{12}} \; \text{T-App}$$

Simply Typed Lambda Calculus

Syntax

$$
\begin{aligned}
t ::= \;& x \\
& \lambda x : T.t \\
& t_1 \; t_2 \\
& Z \\
& S \; t \\
& \mathit{Case} \; t_0 \; \mathit{of} \; 0 \to t_1 \mid Sm \to t_2 \\
v ::= \;& \lambda x : T.t \\
& Z \\
& S \; v \\
T ::= \;& T \to T \\
& \mathit{Nat} \\
\Gamma ::= \;& \varnothing \\
& \Gamma, x : T
\end{aligned}
$$

Evaluation

$$\frac{t_1 \to t_1'}{t_1 \, t_2 \to t_1' \, t_2} \text{ E-App1}$$

$$\frac{t_2 \to t_2'}{v_1 \, t_2 \to v_1 \, t_2'} \text{ E-App2}$$

$$(\lambda x : T_{11}.t_{12}) v_2 \to [x \mapsto v_2] t_{12} \text{ E-AppAbs}$$

Typing

$$\frac{x : T \in \Gamma}{\Gamma \vdash x : T} \text{ T-Var}$$

$$\frac{\Gamma, x : T_1 \vdash t_2 : T_2}{\Gamma \vdash \lambda x : T_1.t_2 : T_1 \to T_2} \text{ T-Abs}$$

$$\frac{\Gamma \vdash t_1 : T_{11} \to T_{12} \qquad \Gamma \vdash t_2 : T_{11}}{\Gamma \vdash t_1 \, t_2 : T_{12}} \text{ T-App}$$

New Evaluation Rules

$$\frac{t_1 \rightarrow t_1'}{S\ t1'}\ \textbf{E-Succ}$$

$$(Case\ Z\ of\ 0 \rightarrow t_1 \mid S\ m \rightarrow t_2) \longrightarrow t1\ \textbf{E-CaseZ}$$

$$(Case\ (S\ n)\ of\ 0 \rightarrow t_1 \mid S\ m \rightarrow t_2) \longrightarrow [m \mapsto n]t_2\ \textbf{E-CaseS}$$

$$\frac{t_0 \rightarrow t_0'}{\begin{array}{c}(Case\ t_0\ of\ 0 \rightarrow t_1 \mid S\ m \rightarrow t_2) \\ \longrightarrow (Case\ t_0'\ of\ 0 \rightarrow t_1 \mid S\ m \rightarrow t_2)\end{array}}\ \textbf{E-Case}$$

New Typing Rules

$$\frac{}{Z : Nat}\ \text{T-NatZ}$$

$$\frac{\Gamma \vdash t_1 : Nat}{S\ t_1 : Nat}\ \text{T-NatS}$$

$$\frac{\Gamma \vdash\ t_0 : Nat \qquad \Gamma \vdash\ t_1 : T_1 \qquad \Gamma \vdash\ t_2 : T_1}{\Gamma \vdash (Case\ t_0\ of\ 0 \rightarrow t_1 \mid (S\ m) \rightarrow t_2) : T_1}\ \text{T-Case}$$

Typechecking

Implementation: Typechecker

Implementation: Evaluator

Thank You!