CS520, Handout 04, 14 September 2006

Programming in the Lambda Calculus

Reading: [Pierce, Section 5.2]

Syntactic Sugar

Multiple arguments: instead of $\lambda(x,y).\,t$ write $\lambda x.\,\lambda y.\,t$, i.e., function definitions are **curried**.

Church booleans:

$\mathtt{tru}$	$\triangleq \lambda\mathtt{x}.\,\lambda\mathtt{y}.\,\mathtt{x}$	*true*
$\mathtt{fls}$	$\triangleq \lambda\mathtt{x}.\,\lambda\mathtt{y}.\,\mathtt{y}$	*false*
$\mathtt{test}$	$\triangleq \lambda\mathtt{l}.\,\lambda\mathtt{m}.\,\lambda\mathtt{n}.\,\mathtt{l\ m\ n}$	simulates *if-then-else*
$\mathtt{and}$	$\triangleq \lambda\mathtt{x}.\,\lambda\mathtt{y}.\,\mathtt{test\ x\ (test\ y\ tru\ fls)\ fls}$	*and* defined with *if-then-else*
$\mathtt{or}$	$\triangleq \lambda\mathtt{x}.\,\lambda\mathtt{y}.\,\mathtt{test\ x\ tru\ (test\ y\ tru\ fls)}$	*or* defined with *if-then-else*
$\mathtt{not}$	$\triangleq \lambda\mathtt{x}.\,\mathtt{test\ x\ fls\ tru}$	*not* defined with *if-then-else*

Checking "$\mathtt{test}$" behaves correctly:

$$\begin{aligned}
& \mathtt{test\ fls}\ t\ u \\
=\ & (\lambda\mathtt{l}.\,\lambda\mathtt{m}.\,\lambda\mathtt{n}.\,\mathtt{l\ m\ n}\ )\ \mathtt{fls}\ t\ u \\
\Rightarrow\ & (\lambda\mathtt{m}.\,\lambda\mathtt{n}.\,\mathtt{fls\ m\ n}\ )\ t\ u \\
\Rightarrow\ & (\lambda\mathtt{n}.\,\mathtt{fls}\ t\ \mathtt{n}\,)\ u \\
\Rightarrow\ & \mathtt{fls}\ t\ u \\
=\ & (\lambda\mathtt{x}.\,\lambda\mathtt{y}.\,\mathtt{y}\ )\ t\ u \\
\Rightarrow\ & (\lambda\mathtt{y}.\,\mathtt{y}\,)\ u \\
\Rightarrow\ & u
\end{aligned}$$

Similarly, we check

$$\mathtt{test\ tru}\ t\ u \Rightarrow^* t$$

Details in [Pierce, p. 59].

More syntactic sugar

Pairing:

$$\mathtt{pair} \triangleq \lambda\mathtt{x}.\,\lambda\mathtt{y}.\,\lambda\mathtt{m}.\,\mathtt{m\ x\ y}$$

Projecting:

$$\begin{aligned}
\mathtt{fst} &\triangleq \lambda\mathtt{z}.\,\mathtt{z\ tru} \\
\mathtt{snd} &\triangleq \lambda\mathtt{z}.\,\mathtt{z\ fls}
\end{aligned}$$

Checking:

$$\begin{aligned}
\mathtt{fst}\,(\mathtt{pair}\ t\ u) &\Rightarrow^* t \\
\mathtt{snd}\,(\mathtt{pair}\ t\ u) &\Rightarrow^* u
\end{aligned}$$

Church numerals

c_0	$\triangleq$	$\lambda x.\,\lambda y.\,y$	0
c_1	$\triangleq$	$\lambda x.\,\lambda y.\,x\;y$	1
c_2	$\triangleq$	$\lambda x.\,\lambda y.\,x\;(x\;y)$	2
c_3	$\triangleq$	$\lambda x.\,\lambda y.\,x\;(x\;(x\;y))$	3
...		...	...
scc	$\triangleq$	$\lambda n.\,\lambda x.\,\lambda y.\,x\;(n\;x\;y)$	simulates *successor*
plus	$\triangleq$	$\lambda m.\,\lambda n.\,\lambda x.\,\lambda y.\,m\;x\;(n\;x\;y)$	simulates *addition*
times	$\triangleq$	$\lambda m.\,\lambda n.\,m\;(\texttt{plus}\;n)\;c_0$	*multiplication* defined with *addition*
prd	$\triangleq$	???	*predecessor* defined with ???
iszero	$\triangleq$	???	*test-for-zero* defined with ???
div	$\triangleq$	???	*integer-division* defined with ???
...		...	...

Checking:

$$
\begin{array}{lll}
 & \texttt{scc}\ c_k & \text{where } k \geqslant 0 \\
= & (\,\lambda n.\,\lambda x.\,\lambda y.\,x\;(n\;x\;y)\,)\ c_k & \\
\Rightarrow & \lambda x.\,\lambda y.\,x\;(c_k\;x\;y) & \\
\Rightarrow^* & \lambda x.\,\lambda y.\,x\;\underbrace{(x\;(x\cdots(x}_{k}\;y))) & \text{notice reduction under } \lambda \\
= & c_{k+1} &
\end{array}
$$

Checking:

$$
\begin{array}{lll}
 & \texttt{plus}\ c_k\ c_\ell & \text{where } k,\ell \geqslant 0 \\
= & (\,\lambda m.\,\lambda n.\,\lambda x.\,\lambda y.\,m\;x\;(n\;x\;y)\,)\ c_k\ c_\ell & \\
\Rightarrow^* & \lambda x.\,\lambda y.\,c_k\;x\;(c_\ell\;x\;y) & \\
\Rightarrow^* & \lambda x.\,\lambda y.\,c_k\;x\;t & \text{where } t = \underbrace{x\;(\cdots(x}_{\ell}\;y)) \\
\Rightarrow^* & \lambda x.\,\lambda y.\,\underbrace{x\;(\cdots(x}_{k}\;t)) & \\
= & \lambda x.\,\lambda y.\,\underbrace{x\;(\cdots(x}_{k+\ell}\;y)) & \\
= & c_{k+\ell} &
\end{array}
$$

Enriching the lambda calculus [Pierce, pp. 63-65]

$\boxed{\boldsymbol{\lambda}}$ – untyped lambda-calculus [Pierce, Fig. 5-3].

$\boxed{\mathbf{NB}}$ – arithmetical expressions [Pierce, Fig. 3-2].

$\boxed{\boldsymbol{\lambda}\mathbf{NB}}$ – untyped lambda-calculus + arithmetical expressions.

Syntax and semantics of $\boldsymbol{\lambda}\mathbf{NB}$ are not formally presented by Pierce, straightforward details left to the reader.

Recursion [Pierce,pp. 65-67]

The **divergent combinator** is an always-diverging term:

$$\boxed{\mathtt{omega} \triangleq (\lambda x.\, x\; x\,)(\lambda x.\, x\; x\,)}$$

Fixed-point combinators generalize the divergent combinator:

$$\boxed{\mathtt{Y} \triangleq \lambda f.\,(\lambda x.\, f\;(x\; x)\,)(\lambda x.\, f\;(x\; x)\,)}$$

$$\boxed{\mathtt{fix} \triangleq \lambda f.\,(\lambda x.\, f\;(\lambda y.\, x\; x\; y\,)\,)(\lambda x.\, f\;(\lambda y.\, x\; x\; y\,)\,)}$$

`Y` suitable for call-by-name evaluation, not for call-by-value.

`fix` suitable for both call-by-name and call-by-value.

The factorial function [Pierce,pp. 52, 66]

$$\mathtt{fact(n)} \triangleq \mathtt{if}\ \mathtt{n} = 0\ \mathtt{then}\ 1\ \mathtt{else}\ \mathtt{n} \times \mathtt{fact(n-1)}$$

Alternatively:

$$\mathtt{fact} \triangleq \lambda n.\, \mathtt{if}\ \mathtt{n} = 0\ \mathtt{then}\ 1\ \mathtt{else}\ \mathtt{n} \times \mathtt{fact(n-1)}$$

From the RHS of the preceding declaration, define:

$$\mathtt{F} \triangleq \lambda f.\, \lambda n.\, \mathtt{if}\ \mathtt{n} = 0\ \mathtt{then}\ 1\ \mathtt{else}\ \mathtt{n} \times \mathtt{f(n-1)}$$

Checking:

$\mathtt{Y\,F}\;3 \Rightarrow^* 3 \times 2 \times 1$ using call-by-name.

$\mathtt{Y\,F}\;3 \Rightarrow^* \cdots$ (diverges) using call-by-value.

$\mathtt{fix\,F}\;3 \Rightarrow^* 3 \times 2 \times 1$ using call-by-name.

$\mathtt{fix\,F}\;3 \Rightarrow^* 3 \times 2 \times 1$ using call-by-value.