

CS 421 Final Exam Reference

1 λ -calculus

Syntax:

$$\text{Expr} ::= \text{Var} \mid \lambda \text{Var} . \text{Expr} \mid \text{Expr Expr}$$

Free variables:

$$\begin{aligned} \text{fv}(x) &= \{x\} \\ \text{fv}(\lambda x . e) &= \text{fv}(e) \setminus \{x\} \\ \text{fv}(e_1 e_2) &= \text{fv}(e_1) \cup \text{fv}(e_2) \end{aligned}$$

Substitution:

$$\begin{aligned} x[x \mapsto e'] &= e' \\ y[x \mapsto e'] &= y && \text{where } x \neq y \\ (\lambda x . e)[x \mapsto e'] &= (\lambda x . e) \\ (\lambda y . e)[x \mapsto e'] &= (\lambda y . e[x \mapsto e']) && \text{where } x \neq y \text{ and } y \notin \text{fv}(e') \\ (e_1 e_2)[x \mapsto e'] &= (e_1[x \mapsto e'])(e_2[x \mapsto e']) \end{aligned}$$

Equivalences:

$$\begin{aligned} \lambda x . e &\equiv_{\alpha} \lambda y . e[x \mapsto y] && \text{where } y \notin \text{fv}(e) \\ (\lambda x . e) e' &\equiv_{\beta} e[x \mapsto e'] && \text{where } x \notin \text{fv}(e) \\ e &\equiv_{\eta} \lambda x . ex && \text{where } x \notin \text{fv}(e) \end{aligned}$$

2 Combinatory Logic

$$\begin{aligned} S x y z &= x z (y z) \\ K x y &= x \\ I &= S K K \end{aligned}$$

Bracket abstraction:

$$\begin{aligned} \lambda x . \rho &\Rightarrow [x]\rho \\ [x]x &\Rightarrow S K K \\ [x]e &\Rightarrow K (e) && \text{where } x \notin \text{fv}(e) \\ [x](\rho \rho') &\Rightarrow S ([x]\rho) ([x]\rho') \end{aligned}$$

3 Church Booleans

$$\begin{aligned} \text{true} &= \lambda \text{then} . \lambda \text{else} . \text{then} \\ \text{false} &= \lambda \text{then} . \lambda \text{else} . \text{else} \\ \text{and} &= \lambda x . \lambda y . x y x \\ \text{not} &= \lambda x . \lambda \text{then} . \lambda \text{else} . x \text{ else then} \end{aligned}$$

4 Church Numerals

$$\begin{aligned} \text{zero} &= \lambda s . \lambda z . z \\ \text{succ} &= \lambda n . \lambda s . \lambda z . s (n s z) \\ \text{add} &= \lambda n . \lambda m . \lambda s . \lambda z . n s (m s z) \end{aligned}$$

5 Transition Semantics (for strict/eager OCaml subset)

e 's are expressions; v 's are values (results of evaluation expressions which cannot be further simplified); n 's are numbers (one kind of value)

	$\overline{(E, x) \rightarrow x}$ where $x \notin E$	$\overline{(E, x) \rightarrow E(x)}$	
			(constants)
			(variables)
(let ₁)	$\frac{(E, e_1) \rightarrow (E, e'_1)}{\overline{(E, \text{let } x = e_1 \text{ in } e_2) \rightarrow (E, \text{let } x = e'_1 \text{ in } e_2)}}$	$\overline{(E, \text{fun } x \rightarrow e') \rightarrow \text{Cl}(x, e', E)}$	(fun)
(let ₂)	$\frac{(E, e_1) \rightarrow v}{\overline{(E, \text{let } x = e_1 \text{ in } e_2) \rightarrow (E[x \mapsto v], e_2)}}$		
(β'_1)	$\frac{(E, e_1) \rightarrow (E, e'_1)}{\overline{(E, e_1 e_2) \rightarrow (E, e'_1 e_2)}}$	$\frac{(E, e_1) \rightarrow (E, e'_1)}{\overline{(E, e_1 + e_2) \rightarrow (E, e'_1 + e_2)}}$	(+ ₁)
(β_1)	$\frac{(E, e_1) \rightarrow v_1}{\overline{(E, e_1 e_2) \rightarrow (E, v_1 e_2)}}$	$\frac{(E, e_1) \rightarrow n_1}{\overline{(E, e_1 + e_2) \rightarrow (E, n_1 + e_2)}}$	(+ ₁)
(β'_2)	$\frac{(E, e_2) \rightarrow (E, e'_2)}{\overline{(E, v_1 e_2) \rightarrow (E, v_1 e'_2)}}$	$\frac{(E, e_2) \rightarrow (E, e'_2)}{\overline{(E, n_1 + e_2) \rightarrow (E, n_1 + e'_2)}}$	(+ ₂)
(β_2)	$\frac{(E, e_2) \rightarrow v_2}{\overline{(E, v_1 e_2) \rightarrow (E, v_1 v_2)}}$	$\frac{(E, e_2) \rightarrow n_2}{\overline{(E, n_1 + e_2) \rightarrow (E, n_1 + n_2)}}$	(+ ₂)
(β_3)	$\overline{(E, \text{Cl}(x, e', E') v_2) \rightarrow (E[x \mapsto v_2], e')}$	$\frac{n = n_1 + n_2}{\overline{(E, n_1 + n_2) \rightarrow n}}$	(+ ₃)

6 Natural Semantics (for strict/eager λ -calculus)

$\overline{(E, x) \Downarrow x}$ where $x \notin E$	(constants)
$\overline{(E, x) \Downarrow E(x)}$	(variables)
$\overline{(E, \lambda x . e') \Downarrow \text{Cl}(x, e', E)}$	(closures)
$\frac{(E, e_1) \Downarrow \text{Cl}(x, e', E') \quad (E, e_2) \Downarrow v \quad ([x \mapsto v]E', e') \Downarrow v'}{\overline{(E, e_1 e_2) \Downarrow v'}}$	(β -reduction)
$\frac{(E, e_1) \Downarrow x \quad (E, e_2) \Downarrow v}{\overline{(E, e_1 e_2) \Downarrow x v}}$ where $x \notin E$	(term construction)

7 The Y Combinator

$$Y F = F (Y F)$$

$$Y = \lambda F . (\lambda x . F (x x))(\lambda x . F (x x))$$

8 Simply-typed λ -calculus

Syntax:

$$\begin{aligned} \text{Type} &::= \bullet \mid \text{Type} \rightarrow \text{Type} \\ \text{Expr} &::= \text{Var} \mid \lambda \text{Var} : \text{Type} . \text{Expr} \mid \text{Expr Expr} \end{aligned}$$

Typing rules:

$$\begin{array}{c} \frac{\Gamma \vdash e : \tau}{\Gamma \vdash e : \bullet} \qquad \frac{}{\Gamma, x : \tau \vdash x : \tau} \\ \\ \frac{\Gamma, x : \sigma \vdash e : \tau}{\Gamma \vdash \lambda x : \sigma . e : \sigma \rightarrow \tau} \qquad \frac{\Gamma \vdash e : \sigma \rightarrow \tau \quad \Gamma \vdash e' : \sigma}{\Gamma \vdash e e' : \tau} \end{array}$$

9 Continuations

CPS transform:

$$\begin{aligned} \text{CPS}(x) &= \lambda k . k x \\ \text{CPS}(\lambda x . e) &= \lambda k . k (\lambda k' . \lambda x . \text{CPS}(e) k') \\ \text{CPS}(e_1 e_2) &= \lambda k . \text{CPS}(e_2) (\lambda x . \text{CPS}(e_1) (\lambda f . f k x)) \end{aligned}$$

10 Pseudo-assembly Language

$R_i = K$	load constant K into register R_i
$R_i = R_j$	copy the contents of register R_j into register R_i
$R_i = *R_j$	load the value at the memory address stored in R_j into register R_i
$*R_i = R_j$	store the contents of register R_j into memory at the address stored in R_i
$R_i = R_j \text{ op } R_k$	where op is one of $+$, $-$, $*$, $/$, and , or , xor
push R_i	push the contents of register R_i onto the stack
pop R_i	pop the next value off the stack, into register R_i
jmp $label$	unconditional jump to $label$
jmp $*R_i$	unconditional indirect jump to address stored in R_i
jq $R_i, R_j, label$	jump to $label$ if $R_i > R_j$
jeq $R_i, R_j, label$	jump to $label$ if $R_i = R_j$
$R_i = \phi(R_j, R_k)$	SSA ϕ operator: R_i gets the value of R_j or R_k depending on the control flow

SSA form: each logical register is assigned a value at exactly one point in the code.