Evaluation

Rule 1: Do not write rules for things that do not evaluate
Rule 2: Each rule should capture exactly one step in the evaluation process
Rule 3: Evaluate subterms before terms
Rule 4: No term should have more than one matching inference rule

$t ::= true \mid false \mid (if\ t\ t\ t)$
$v ::= true | false$

$$\frac{}{(if\ true\ t_t\ t_e) \rightarrow t_t}$$

$$\frac{}{(if\ false\ t_t\ t_e) \rightarrow t_e}$$

$$\frac{t_c \rightarrow t_c'}{(if\ t_c t_t t_e) \rightarrow (if\ t_c' t_t t_e)}$$

A term is in normal form if no evaluation rule applies to it
A term is a value if it represents the conclusion of a computation
Every value is a normal form
Satisfies - for each instance of a rule either the conclusion is in a relation or one of its preconditions is not
One-step evaluation relating ($t \rightarrow t'$) - the smallest operation satisfying all rules
Derivable - result of evaluation

Multi step:

$$\frac{}{t \rightarrow^* t}\ ERefl$$

$$\frac{t \rightarrow t'}{t \rightarrow^* t'}\ EStep$$

$$\frac{t \rightarrow^* t' \quad t' \rightarrow t''}{t \rightarrow^* t''}\ ETrans$$

$$\frac{}{pred\ 0 \rightarrow 0}\ EPredZero$$

$$\frac{}{pred\ (succ\ nv_1) \rightarrow nv_1}\ EPredSucc$$

$$\frac{t_1 \rightarrow t_1'}{pred\ t_1 \rightarrow pred\ t_1'}\ EPred$$

Binding instance - where variable is declared
Bound instance - variable used in scope
Free instance - variable used outside of scope
Combinator - lambda with no free variables
Redux - lambda that can be reduced (value)
Reducible form - evaluable

$((\lambda w.\,w)(\lambda x.\,x))((\lambda y.\,y)(\lambda z.\,z))$
$\rightarrow ((\lambda w.\,w)(\lambda x.\,x))(\lambda z.\,z)$
$\rightarrow (\lambda x.\,x)(\lambda z.\,z)$
$\rightarrow (\lambda z.\,z)$

Church Encoding

$true = \lambda x.\,\lambda y.\,x$
$false = \lambda x.\,\lambda y.\,y$
$and = \lambda x.\,\lambda y.\,xyx$
$or = \lambda x.\,\lambda y.\,xxy$
$not = \lambda x.\,x\,false\,true$
$cons = \lambda l.\,\lambda r.\,\lambda c.\,c\;l\;r$
$((cons\;v_l)v_r) = \lambda l.\,\lambda r.\,\lambda c.\,c\;l\;r$
$\rightarrow \lambda r.\,\lambda c.\,c\;v_l\;r$
$\rightarrow \lambda c.\,c\;v_l v_r$
$succ = \lambda n.\,\lambda s.\,\lambda z\;s\;(n\;s\;z)$
$0 = \lambda s.\,\lambda z.\,z$
$1 = \lambda s.\,\lambda z.\,s\;z$
$2 = \lambda s.\,\lambda z.\,s\;(s\;z)$
$plus = \lambda m.\,\lambda n.\,\lambda s.\,\lambda z.\,(m\;S\;(n\;S)\;Z)$
$(\lambda s.\,\lambda z.\,(s(s(sz))))(\lambda s.\,\lambda z.\,(s(sz)))$
$\rightarrow \lambda s.\,\lambda z.\,((\lambda z.\,(s(sz))))((((s(s(sz))))s)s)$
$\rightarrow \lambda s.\,\lambda z.\,((\lambda z.\,(s(sz)))(((s(s(sz))))$
$\rightarrow \lambda s.\,\lambda z.\,(s(s(s(s(sz)))))$
$times = \lambda m.\,\lambda n.\,\lambda s.\,\lambda z.\,m\;(n\;s\;z)\;z$
$[\![t_1 + t_2]\!] = [\![t_1]\!] + [\![t_2]\!]$
$[\![true]\!] = \lambda a.\,\lambda b.\,a$
$[\![if\;t_1\;then\;t_2\;else\;t_3]\!] = (([\![t_1]\!][\![t_2]\!])[\![t_3]\!])$
$[\![a + b]\!] = \lambda s.\,\lambda z.\,([\![a]\!]S\;([\![b]\!]\;S)\;Z)$
$F = \lambda g.\,\lambda z.\,if\;z = 0\;then\;1\;else\;z * (g(z - 1))$
$Y = (\lambda f.\,(\lambda x.\,(f(xx)))(\lambda x.\,(f(xx))))$
$YF\;3 = F(YF)3 =$
$\lambda f.\,\lambda x.\,(if\;x == 0\;then\;1\;else\;x * f(x - 1))(YF)3$
$\lambda x.\,(if\;x == 0\;then\;1\;else\;x * (YF)(x - 1))3$
$if\;3 == 0\;then\;1\;else\;3 * (YF)(3 - 1) = 3 * (YF)2$

Simply Typed Lambda

$$\frac{}{0{:}Nat}\ TZero$$

$$\frac{}{true{:}Bool}\ TTrue$$

$$\frac{}{false{:}Bool}\ TFalse$$

$$\frac{t{:}Nat}{succ\ t{:}Nat}\ TSucc$$

$$\frac{t_1{:}Bool\ \ t_2{:}T\ \ t_3{:}T}{(if\ t_1 t_2 t_3){:}T}\ TIf$$

$T ::= T \to T | Bool$

$(\lambda x: Bool. x): Bool \to Bool$

$(((\lambda x: Bool. x): Bool \to Bool) true): Bool$

$$\frac{t_1{:}T_1 \to T_2\ \ t_2{:}T_1}{(t_1 t_2){:}T_2}\ TApp$$

The context of a term Γ is a list of all bound variables and their types

$\Gamma = [(x \mapsto Bool \mapsto Bool), (y \mapsto Bool)]$

$$\frac{(x \mapsto T) \in \Gamma}{\Gamma \vdash x{:}T}\ TVar$$

$$\frac{((x \mapsto T_1), \Gamma \vdash t_2{:}T_2}{\Gamma \vdash \lambda x{:}T_1.t_2{:}T_1 \to T_2}\ TLambda$$

$$\dfrac{\dfrac{\dfrac{\dfrac{x \mapsto Bool \to Bool \in \Gamma}{\Gamma \vdash x : Bool \to Bool} \qquad \dfrac{y \mapsto Bool \in \Gamma}{\Gamma \vdash y : Bool}}{[y \mapsto Bool, x \mapsto (Bool \to Bool)] \vdash (x\ y) : Bool}}{[x, \mapsto (Bool \to Bool)] \vdash \lambda y : Bool. (x\ y) : Bool \to Bool}}{\emptyset \vdash \lambda x : Bool \to Bool. \lambda y : Bool. (x\ y) : (Bool \to Bool) \to (Bool \to Bool)}$$

$t ::= \lambda x: T. t \mid (tt) \mid x$

$v ::= \lambda x: T. t$

$T ::= T \to T$

$\Gamma ::= \varnothing \mid (x \mapsto T), \Gamma$

$$\frac{}{(\lambda{:}T_{11}.t_{12} v_2) \to [x \mapsto v_2] t_{12}}\ Beta$$ # Beta reduction

$$\frac{t_1 \to t_1'}{t_1 t_2 \to t_1' t_2}\ LambdaT1$$

$$\frac{t_2 \to t_2'}{v_1 t_2 \to v_1 t_2'}\ LambdaT2$$

$$\frac{z{:}T\ \ \lambda x{:}T.t{:}T \to R}{((\lambda x.T.t)z){:}R}$$

Progress:

$\Gamma \vdash (if\ t_1\ t_2\ t_3): T$ - assume progress for t_1, t_2, t_3. $\Gamma \vdash t_1: Bool$ by type inversion. Canonical forms gives:

$t_1 = true$ then EIfTrue applies and we take step

$t_1 = false$ then EIfFalse applies and we take step

$t_1 \mapsto t_1'$ then EIf applies and we take a step

Proof tools:
Induction over type derivations (looking at each type rule)
Canonical forms (type tells us values)
Unique types (only one type per term)
Induction hypothesis (parts imply the whole)
Type inversion (whole implies the parts)
$if\ (if\ t_1\ t_2\ t_3)\colon\ R\ then\ t_1\colon Bool,\ t_2\colon R,\ and\ t_3\colon R$
$if\ \Gamma\vdash(if\ t_1 t_2 t_3)\colon R\ then\ \Gamma\vdash t_1\colon Bool\ and\ \Gamma\vdash t_2, t_3\colon R$
Progress + Preservation = Safety (soundness)
Progress - a well-typed term is value or reducible
Preservation - type does not change in evaluation for well-typed
$t\colon T\ {-}{>}\ t'\colon T$

		Evaluation + Induction Hyp		
	$t_1 : T_1, t_2 : T_2, \ldots$	$\longrightarrow$	$t_1' : T_1, t_2' : T_2, \ldots$	
Type Inversion	$\uparrow$		$\downarrow$	Type Rule
	$t : T$	$\longrightarrow$	$t' : T$	
		Evaluation		

$t_1\colon T_2 \to T_1 \quad t_2\colon T_2 \quad \Rightarrow t_1'\colon T_2 \to T_1 \quad t_2'\colon T_2 \Downarrow$
$\Uparrow \quad (t_1,\ t_2\ )\colon T \quad \Rightarrow \quad t\colon T \quad (t_1'\ t_2)\colon T_1$

Extension

There are three primary ways to enrich a language:
2. Derived Forms - Add new syntax and new types defined in terms of existing constructs.
3. Libraries - Add new functionality by using the language to define new reusable constructs.
1. Extension - Add new syntax, new values, new types, and associated inference rules. (eval, type, progress, preservation, determinicity)
Syntax
$T ::= \ldots\ |A|T \to T$
$t ::= \ldots\ |unit$
$v ::= \ldots\ |unit$
$T ::= \ldots\ |Unit$
$t ::= ..\ |let\ x\ =\ t\ in\ t$
Type rules
$\Gamma\vdash unit\colon Unit$
$\dfrac{\Gamma\vdash t_1:Unit\ \ \Gamma\vdash t_2:T}{\Gamma\vdash t_1;t_2:T}\ TSeq$
$\dfrac{\Gamma\vdash t_1:T_1\ \ (x\mapsto T_1),\Gamma\vdash t_2:T_2}{\Gamma\vdash let\ x=t_1\ in\ t_2:T}\ TLet$
Evaluation Rules

$$\frac{t_1 \to t_1'}{t_1; t_2 \to t_1'; t_2}\ ESeq1$$

$$\frac{}{unit; t_2 \to t_2}\ ESeqUnit$$

$$\frac{}{let\ x = v_1\ in\ t_2 \to [x \mapsto v_1] t_2}\ ELetV$$

$$\frac{t_1 \to t_1'}{let\ x = t_1\ in\ t_2 \to let\ x = t_1'\ in\ t_2}\ ELet$$

Derived Forms

$$let\ x\ =\ t_1\ in\ t_1 \equiv ((\lambda : T.\, t_2)\ t1)$$

$$\frac{a}{b} \Rightarrow a \vdash b$$