

TaPL Cheat Sheet (TODO: hand-write!)

- Type Safety = Progress + Preservation
 - Progress: If $\vdash e : t$, then $e = v$ or $\exists e' : e \rightarrow e'$
 - Preservation: If $\Gamma \vdash e : t$ and $e \rightarrow e'$, then $\Gamma \vdash e' : t$

Untyped λ -Calculus

- $e_1 e_2 e_3 = (e_1 e_2) e_3$
- $\lambda x. e_1 e_2 = \lambda x. (e_1 e_2)$
- Church encodings:
 - True: $\lambda x. \lambda y. x$
 - False: $\lambda x. \lambda y. y$
 - Not: $\lambda a. a f t t$
 - And: $\lambda a. \lambda b. a b a$
 - Or: $\lambda a. \lambda b. a a b$
- Free variables:
 - $FV(x) = \{x\}$
 - $FV(e_1 e_2) = FV(e_1) \cup FV(e_2)$
 - $FV(\lambda x. e) = FV(e) \setminus \{x\}$
- α -renaming:
 - $(x \mapsto y) x = y$
 - $(x \mapsto y) z = z$
 - $(x \mapsto y) (e_1 e_2) = ((x \mapsto y) e_1) ((x \mapsto y) e_2)$
 - $(x \mapsto y) \lambda x. e = \lambda y. ((x \mapsto y) e)$
 - $(x \mapsto y) \lambda z. e = \lambda z. ((x \mapsto y) e)$
- Substitution:
 - $[x \mapsto e] x = e$
 - $[x \mapsto e] y = y$
 - $[x \mapsto e] (e_1 e_2) = ([x \mapsto e] e_1) ([x \mapsto e] e_2)$
 - $[x \mapsto e] \lambda x. e_b = \lambda x. e_b$
 - $[x \mapsto e] \lambda y. e_b = \lambda y. ([x \mapsto e] e_b)$ if $y \notin FV(e)$
 - $[x \mapsto e] \lambda y. e_b = \lambda y'. ([x \mapsto e] (y \mapsto y') e_b)$ if $y \in FV(e)$
- β -reduction: $(\lambda x. e_b) e \xrightarrow{\beta} [x \mapsto e] e_b$
- η -reduction: $\lambda x. e x \xrightarrow{\eta} e$ if $x \notin FV(e)$
- Diamond Property: if $a \rightarrow a'$ and $a \rightarrow a''$, then there exists $a^\dagger$ s.t. $a' \rightarrow a^\dagger$ and $a'' \rightarrow a^\dagger$
 - Church-Rosser Property: $\xrightarrow[\beta]{*}$ satisfies the diamond property
- De Bruijn Indices: replace named variables by numbers, where k stands for the variable bound by the k th enclosing λ

Simply Typed λ -Calculus

- simple types and function types
- Type Erasure:
 - $E(x) = x$
 - $E(e_1 e_2) = E(e_1) E(e_2)$
 - $E(\lambda x : t. e) = \lambda x. E(e)$
- Tuples:
 - Expressions: $\{\bar{e}\}, e.i$
 - Type: $\{\bar{t}\}$

- Evaluation: $\{\overline{v}\}.i \rightarrow v_i$
- Records:
 - Expressions: $\{\overline{l = e}\}, e.l$
 - Type: $\{\overline{l : t}\}$
 - Evaluation: $\{\overline{l = v}\}.l_i \rightarrow v_i$
- Sums:
 - Expressions:
 - $\text{inl } e \text{ as } t$
 - $\text{inr } e \text{ as } t$
 - $\text{case } e \text{ of } \text{inl } x \Rightarrow e \mid \text{inr } x \Rightarrow e$
 - Type: $t \cup t$
 - Evaluation:
 - $\text{case } (\text{inl } v \text{ as } t) \text{ of } \text{inl } x_1 \Rightarrow e_1 \mid \text{inr } x_2 \Rightarrow e_2 \rightarrow [x_1 \mapsto v]e_1$
 - $\text{case } (\text{inr } v \text{ as } t) \text{ of } \text{inl } x_1 \Rightarrow e_1 \mid \text{inr } x_2 \Rightarrow e_2 \rightarrow [x_2 \mapsto v]e_2$
- Variants:
 - Expressions:
 - $\langle l = e \rangle \text{ as } t$
 - $\text{case } e \text{ of } \overline{\langle l = x \rangle \Rightarrow e}$
 - Type: $\overline{\langle l : t \rangle}$
 - Evaluation: $\text{case } (\langle l_i = v \rangle \text{ as } t) \text{ of } \overline{\langle l = x \rangle \Rightarrow e} \rightarrow [x_i \mapsto v]e_i$
- General recursion:
 - Expressions: $\text{fix } e$
 - Evaluation: $\text{fix } (\lambda x : t. e) \rightarrow [x \mapsto \text{fix } (\lambda x : t. e)]e$
- Simply typed λ -calculus + Nat + fix is Turing-complete
- Normalization:
 - $R_{\text{simple}}(e)$ iff e halts
 - $R_{t_d \rightarrow t_c}(e)$ iff e halts and $R_{t_d}(e_2) \implies R_{t_c}(e_1 e_2)$

Subtyping

- Top and Bottom Type
 - $t <: \top, \perp <: t$ for all types t
- If $t_1 <: s_1$ and $s_2 <: t_2$, then $s_1 \rightarrow s_2 <: t_1 \rightarrow t_2$
- Records:
 - $\{\overline{l : t}, \overline{k : u}\} <: \{\overline{l : t}\}$
 - if $s <: t$, then $\{\overline{l : s}\} <: \{\overline{l : t}\}$
 - Permutations of records are subtypes
- In variants, it's the other way:
 - $\overline{\langle l : t \rangle} <: \overline{\langle l : t, k : u \rangle}$

Recursive Types

- Y-Combinator: $Y = \lambda f : T \rightarrow T. (\lambda x : (\mu A. A \rightarrow T). f(xx))(\lambda x : (\mu A. A \rightarrow T). f(xx))$
- $A = \text{NatList } B = \langle \text{nil} : (), \text{const} : (\text{Nat}, \text{Natlist}) \rangle$
 - Equi-recursive types: $A = B$, undecidable
 - Iso-recursive types: $A \cong B$, explicitly annotate unfolding/folding
- Iso-recursive types:
 - Expressions: $\text{fold } [t]e, \text{unfold } [t]e$
 - Types:
 - Type variable X

- Recursive type $\mu X.t$
- Evaluation: $\text{unfold}[x](\text{fold}[t]v) \mapsto v$
- unfold: $\mu X.t \mapsto [X \mapsto \mu X.t]t$
- fold: unfold^{-1}

Polymorphism

- Ad-hoc polymorphism
- Parametric polymorphism
- Subtype polymorphism
- System F:
 - Expressions: $e[t], \Lambda \alpha. e$
 - Types: type variables α , universal type $\forall \alpha. t$
 - Evaluation: $(\Lambda a. e)[t] \rightarrow [\alpha \mapsto t]e$
- Existential types:
 - Expressions: $\text{pack}[t](t, e), \text{let } (\alpha, x) = e; e$
 - Types: existential type $\exists \alpha. t$
 - Evaluation: $\text{let } (\alpha, x) = \text{pack}[t](t, v); e \rightarrow [\alpha \mapsto t][x \mapsto v]e$
- Skolemization: $\forall \alpha. \exists \beta. P(\alpha, \beta) \iff \forall \alpha. P(\alpha, f(\alpha))$, where f maps each $\alpha \rightarrow$ a corresponding β (which must exist)
- Curry-Howard Isomorphism: System F is isomorphic $\rightarrow$ second order logic
- Let-polymorphism:
 - Expressions: $\text{let } x = e; e$
 - Monotypes: type variables α , type constructors $Ct \dots t$
 - Polytypes: monotypes, universal type $(\forall \alpha. \sigma)$
- Free type variables:
 - $\text{FV}(\alpha) = (\alpha)$
 - $\text{FV}(Ct_1 \dots t_n) = \bigcup_{i=1}^n \text{FV}(t_i)$
 - $\text{FV}(\Gamma) = \bigcup_{x:\sigma \in \Gamma} \text{FV}(\sigma)$
 - $\text{FV}(\forall \alpha. \sigma) = \text{FV}(\sigma) \setminus (\alpha)$
 - $\text{FV}(\Gamma \vdash e : \sigma) = \text{FV}(\sigma) \setminus \text{FV}(\Gamma)$
- Specialization: if $t' = [\alpha_i \mapsto t_i]t$ and $\beta_i \notin \text{FV}(\forall \alpha_1 \dots \forall \alpha_n. t)$, then $\forall \alpha_1 \dots \forall \alpha_n. t \sqsubseteq \forall \beta_1 \dots \forall \beta_m. t'$
- System $F_{<}$: has subtyping
 - Expressions: $\Lambda \alpha <: t. e$
 - Types: top type $\top$, universal type $\forall \alpha <: t. t$
- System F_ω has type-level functions and higher-order polymorphism
 - Expressions:
 - Type application et
 - Type abstraction $\lambda \alpha : k. e$
 - Kinds:
 - Kind of proper types $*$
 - Kind of operators $k \rightarrow k$
 - Types:
 - Universal type $\forall \alpha : k. t$
 - Operator abstraction $\lambda \alpha : k. t$
 - Operator application tt
 - Evaluation: $(\lambda \alpha : k. e)t \rightarrow [\alpha \mapsto t]e$

Dependent Types

- λ -cube:
 - Expressions:
 - Abstraction: $\lambda x : e.e$
 - Quantification: $\Pi x : e.e$
 - Sorts
 - Sorts: $*$, $\square$
 - Syntax sugar:
 - $\Lambda x.e := \lambda x : *.e$
 - $e_1 \rightarrow e_2 := \Pi x.e_1.e_2$
 - $\forall x.e := \Pi x : *.e$
- TODO copy table
- Curry-Howard correspondence:
 - proposition $\cong$ type
 - proof $\cong$ term