TaPL Cheat Sheet (TODO: hand-write!)

- Type Safety = Progress + Preservation
  - Progress: If $\vdash e : t$, then $e = v$ or $\exists e' : e \to e'$
  - Preservation: If $\Gamma \vdash e : t$ and $e \to e'$, 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: $\lambda a.a f f t t$
  - And: $\lambda a.\lambda b.aba$
  - Or: $\lambda a.\lambda b.aab$
- Free variables:
  - $\mathrm{FV}(x) = \{x\}$
  - $\mathrm{FV}(e_1 e_2) = \mathrm{FV}(e_1) \cup \mathrm{FV}(e_2)$
  - $\mathrm{FV}(\lambda x.e) = \mathrm{FV}(e) \setminus \{x\}$
- 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 $y \notin \mathrm{FV}(e)$
  - $[x \mapsto e]\lambda y.e_b = \lambda y'.([x \mapsto e](y \mapsto y')e_b)$ if $y \in \mathrm{FV}(e)$
- β-reduction: $(\lambda x.e_b)e \underset{\beta}{\to} [x \mapsto e]e_b$
- η-reduction: $\lambda x.ex \underset{\eta}{\to} e$ if $x \notin \mathrm{FV}(e)$
- Diamond Property: if $a \to a'$ and $a \to a''$, then there exists $a^\dagger$ s.t. $a' \to a^\dagger$ and $a'' \to a^\dagger$
  - Church-Rosser Property: $\overset{*}{\underset{\beta}{\to}}$ satisfies the diamond property
- De Bruijn Indices: replace named variables by numbers, where k stands for the variable bound by the kth 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: $\{\overline{e}\}$, $e.i$
  - Type: $\{\overline{t}\}$

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

Subtyping

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

Recursive Types

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

    - Recursive type $\mu X.t$
  - Evaluation: unfold$[x]$(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: $(\Lambda a.e)[t] \to [\alpha \mapsto t]e$
- Existential types:
  - Expressions: pack$[t](t, e)$, let $(\alpha, x) = e; e$
  - Types: existential type $\exists\alpha.t$
  - Evaluation: let $(\alpha, x) =$ pack$[t](t, v); e \to [\alpha \mapsto t][x \mapsto v]e$
- Skolemization: $\forall\alpha.\exists\beta.P(\alpha, \beta) \iff \forall\alpha.P(\alpha, f(\alpha))$, where f maps each α -> a corresponding β (which must exist)
- Curry-Howard Isomorphism: System F is isomorphic -> second order logic
- Let-polymorphism:
  - Expressions: let $x = e; e$
  - Monotypes: type variables α, type constructors $Ct...t$
  - Polytypes: monotypes, universal type ($\forall\alpha.\sigma$)
- Free type variables:
  - $\mathrm{FV}(\alpha) = (\alpha)$
  - $\mathrm{FV}(Ct_1...t_n) = \bigcup_{i=1}^{n} \mathrm{FV}(t_i)$
  - $\mathrm{FV}(\Gamma) = \bigcup_{x:\sigma\in\Gamma} \mathrm{FV}(\sigma)$
  - $\mathrm{FV}(\forall\alpha.\sigma) = \mathrm{FV}(\sigma) \setminus (\alpha)$
  - $\mathrm{FV}(\Gamma \vdash e : \sigma) = \mathrm{FV}(\sigma) \setminus \mathrm{FV}(\Gamma)$
- Specialization: if $t' = [\alpha_i \mapsto t_i]t$ and $\beta_i \notin \mathrm{FV}(\forall\alpha_1...\forall\alpha_n.t)$, then $\forall\alpha_1...\forall\alpha_n.t \sqsubseteq \forall\beta_1...\forall\beta_m.t'$
- System $\mathrm{F}_{<:}$ 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 $k \to k$
  - Types:
    - Universal type $\forall\alpha : k.t$
    - Operator abstraction $\lambda\alpha : k.t$
    - Operator application tt
  - Evaluation: $(\lambda\alpha : k.e)t \to [\alpha \mapsto t]e$

Dependent Types

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