Goals Since Day 1

Develop tools to **rigorously** study what programs mean.

*semantics*

- equivalence, termination, determinism, ...

Develop tools for studying program behavior

  *inductive defns, structural induction, inference rules*

Investigate core PL concepts

  *types, functions, scope, mutation, iteration*
Cruising to Victory
Covered Serious Ground

- Functional Programming
- Formal Definitions, Structural Induction, Semantics
- Various Lambda Calculi
- Types, Progress, Preservation
- Evaluation Contexts and Continuation Passing Style
- Subtyping, Parametric Polymorphism
Developed Sweet Skills

• Writing Formal Proofs
• Language Implementation
• Extending Languages
• Taste for Design Tradeoffs
• Appreciating Deep Connections (e.g. Curry-Howard)

• *Enduring Long Exams*
Developed Sweet Skills

• *Keeping a Straight Face*
Today: Review & Review

• Extending Progress and Preservation Proofs
• Quick Look Back at Evaluation Contexts
• Putting Terms into Continuation Passing Style
• Subtyping: LSP, Covariance, Contravariance
• Type Derivations with Parametric Polymorphism
• Course Evaluations
Today: Review & Review

- Extending Progress and Preservation Proofs
- Quick Look Back at Evaluation Contexts
- Putting Terms into Continuation Passing Style
- Subtyping: LSP, Covariance, Contravariance
- Type Derivations with Parametric Polymorphism
- Course Evaluations
Extensions and Type Safety

Need to establish two properties:

1. **Progress**
   
   If $\vdash e : T$, then either (A) $e$ is a value or (B) there exists $e'$ such that $e \rightarrow e'$.

2. **Preservation**
   
   If $\vdash e : T$ and $e \rightarrow e'$, then $\vdash e' : T$. 
Proof generally has this shape:

induction on $* |- e : T$

base cases either:

1. value (done)
2. not typable in empty context (contradiction, done)

inductive cases:

- inversion on typing provides types for subexpressions
- $\text{IH} +$ subexpr type implies they are values or can step
- if subexpression steps, big expression steps
- $\text{NOTE}$: canonical forms provides shape of typed values
Product Progress

Case * |- (e1, e2) : T1 * T2
- inversion provides * |- e1 : T1 and * |- e2 : T2
- if e1 not a value
  - by IH and typing e1 can step to e1'
  - then (e1, e2) can step to (e1', e2)
- else e1 a value, if e2 not a value
  - by IH and typing e2 can step to e2'
  - then (e1, e2) can step to (e1, e2')
- else e2 a value
  - both values, whole thing value, not stuck, done
Preservation

Proof generally has this shape:

base cases all contradictions, either
(A) not typable in empty context (bogus)
(B) cannot step (bogus)

inductive cases:
- inversion on typing provides types for subexprs
- case analysis on step + inversion provides subexpr step
- \textbf{IH} + subexpr type + subexpr step provides new
  subexpr still well typed
- stitch back together to show big expr still well typed
- \textit{NOTE}: use substitution lemma for app, match, etc.
Product Preservation

Case $\vdash (e_1, e_2) : T_1 \times T_2 \text{ and } (e_1, e_2) \rightarrow e'$
- inversion provides $\vdash e_1 : T_1 \text{ and } \vdash e_2 : T_2$
- case analysis on step
  - $e_1 \rightarrow e_1'$ and $e' = (e_1', e_2)$
    - by IH and typing $e_1' : T_1$
    - then $(e_1', e_2)$ still has type $T_1 \times T_2$
  - $e_2 \rightarrow e_2'$ and $e' = (e_1, e_2')$
    - by IH and typing $e_2' : T_2$
    - then $(e_1, e_2')$ still has type $T_1 \times T_2$
Today: Review & Review

- Extending Progress and Preservation Proofs
- Quick Look Back at Evaluation Contexts
- Putting Terms into Continuation Passing Style
- Subtyping: LSP, Covariance, Contravariance
- Type Derivations with Parametric Polymorphism
- Course Evaluations
Today: Review & Review

- Extending Progress and Preservation Proofs
- **Quick Look Back at Evaluation Contexts**
- Putting Terms into Continuation Passing Style
- Subtyping: LSP, Covariance, Contravariance
- Type Derivations with Parametric Polymorphism
- **Course Evaluations**
Evaluation Contexts

Evaluation contexts define where interesting work can happen:

\[
E ::= [\cdot] \mid E \ e \mid v \ E \mid (E, e) \mid (v, E) \mid E.1 \mid E.2 \\
\mid A(E) \mid B(E) \mid (\text{match } E \text{ with } Ax. \ e_1 \mid By. \ e_2)
\]

\[
e \rightarrow e' \text{ with 1 rule: } \frac{e \xrightarrow{P} e'}{E[e] \rightarrow E[e']}
\]

\[
e \xrightarrow{P} e' \text{ does all the "interesting work":}
\]

\[
\begin{align*}
(\lambda x. \ e) \ v \xrightarrow{P} e[v/x] & \quad (v_1, v_2).1 \xrightarrow{P} v_1 & \quad (v_1, v_2).2 \xrightarrow{P} v_2 \\
\xrightarrow{\text{match } A(v) \text{ with } Ax. \ e_1 \mid By. \ e_2} e_1[v/x] & \quad \xrightarrow{\text{match } B(v) \text{ with } Ay. \ e_1 \mid Bx. \ e_2} e_2[v/x]
\end{align*}
\]
Today: Review & Review

• Extending Progress and Preservation Proofs

• **Quick Look Back at Evaluation Contexts**

• Putting Terms into Continuation Passing Style

• Subtyping: LSP, Covariance, Contravariance

• Type Derivations with Parametric Polymorphism

• **Course Evaluations**
Today: Review & Review

• Extending Progress and Preservation Proofs
• Quick Look Back at Evaluation Contexts
• Putting Terms into Continuation Passing Style
  • Subtyping: LSP, Covariance, Contravariance
  • Type Derivations with Parametric Polymorphism
• Course Evaluations
let rec fact n =
  if n = 0 then
    1
  else
    n * fact (n - 1)

let rec fact' n k =
  (eq' n 0 (fun b ->
    (if b then
      (k 1)
    else
      (sub' n 1 (fun m ->
        (fact' m (fun p ->
          (mult' n p k))))))))
Today: Review & Review

- Extending Progress and Preservation Proofs
- Quick Look Back at Evaluation Contexts
- **Putting Terms into Continuation Passing Style**
- Subtyping: LSP, Covariance, Contravariance
- Type Derivations with Parametric Polymorphism
- **Course Evaluations**
Today: Review & Review

- Extending Progress and Preservation Proofs
- Quick Look Back at Evaluation Contexts
- Putting Terms into Continuation Passing Style
- Subtyping: LSP, Covariance, Contravariance
- Type Derivations with Parametric Polymorphism
- Course Evaluations
Subtyping: Follow LSP

Liskov Substitution Principle:

If \( A \) is a subtype of \( B \) (written \( A <: B \)), then we can safely use a value of type \( A \) anywhere a value of type \( B \) is expected.
Subtyping Smaller Parts

- *Covariance*: same direction as bigger type
- *Contravariance*: opposite direction of bigger type

\[ \tau_1 \rightarrow \tau_2 \leq \tau_3 \rightarrow \tau_4 \]
Today: Review & Review

• Extending Progress and Preservation Proofs
• Quick Look Back at Evaluation Contexts
• Putting Terms into Continuation Passing Style

• **Subtyping: LSP, Covariance, Contravariance**
• Type Derivations with Parametric Polymorphism
• **Course Evaluations**
Today: Review & Review

- Extending Progress and Preservation Proofs
- Quick Look Back at Evaluation Contexts
- Putting Terms into Continuation Passing Style
- Subtyping: LSP, Covariance, Contravariance

  - **Type Derivations with Parametric Polymorphism**

- Course Evaluations
Typing Bambdas

• Look at AST, look at typing rules, pattern match

• *Try to think as little as possible*

\[
\begin{align*}
\Delta; \Gamma \vdash x : \Gamma(x) \\
\Delta; \Gamma, x: \tau_1 \vdash e : \tau_2 \quad \Delta \vdash \tau_1 \\
\Delta; \Gamma \vdash \lambda x: \tau_1. \ e : \tau_1 \to \tau_2 \\
\Delta, \alpha; \Gamma \vdash e : \tau_1 \\
\Delta; \Gamma \vdash \Lambda \alpha. \ e : \forall \alpha. \tau_1 \\
\Delta; \Gamma \vdash e_{\tau_2} : \tau_1[\tau_2/\alpha] \\
\end{align*}
\]

\[(\Lambda \alpha. \Lambda \beta. \lambda x : \alpha. \lambda f : \alpha \to \beta. \ f \ x) \ [\text{int}] \ [\text{int}] \ 3 \ (\lambda y : \text{int}. \ y + y)\]
Today: Review & Review

- Extending Progress and Preservation Proofs
- Quick Look Back at Evaluation Contexts
- Putting Terms into Continuation Passing Style
- Subtyping: LSP, Covariance, Contravariance

- Type Derivations with Parametric Polymorphism
- Course Evaluations
Today: Review & Review

- Extending Progress and Preservation Proofs
- Quick Look Back at Evaluation Contexts
- Putting Terms into Continuation Passing Style
- Subtyping: LSP, Covariance, Contravariance
- Type Derivations with Parametric Polymorphism

- Course Evaluations
Thanks!

- Really enjoyed our discussions during lecture
- Learned a lot about teaching vs. giving a lecture
- Y’all are incredibly bright, very promising futures
- Remember tricks:
  - Have one question for each topic.
  - “That’s a great question. What do you think?”
Course Feedback

• Voluntary

• Confidential

• Grade Independent

• No. 2 pencil ONLY on scan forms