# CSE 505
# Graduate PL

Fall 2013

## 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 `* |- e : T`, then either (A) `e` is a value or (B) there exists `e'` such that `e -> e'`.

2. ***Preservation***
   If `* |- e : T` and `e -> e'`, then `* |- e' : T`.

# Progress

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 subexprs
      - `IH` + subexpr type implies they are values or can step
      - if subexpression steps, big expression steps
      - *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
      - `IH` + subexpr type + subexpr step provides new
        subexpr still well typed
      - stitch back together to show big expr still well typed
      - *NOTE*: use substitution lemma for app, match, etc.

# Product Preservation

Case `* |- (e1, e2) : T1 * T2` and `(e1, e2) -> e'`

- inversion provides `* |- e1 : T1` and `* |- e2 : T2`
- case analysis on step
- `e1 -> e1'` and `e' = (e1', e2)`
  - by `IH` and typing `e1' : T1`
  - then `(e1', e2)` still has type `T1 * T2`
- `e2 -> e2'` and `e' = (e1, e2')`
  - by `IH` and typing `e2' : T2`
  - still has type `(e1, e2')` still has type `T1 * T2`

# 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:

$$E ::= [\cdot] \mid E\ e \mid v\ E \mid (E, e) \mid (v, E) \mid E.1 \mid E.2$$
$$\mid \mathsf{A}(E) \mid \mathsf{B}(E) \mid (\textsf{match}\ E\ \textsf{with}\ \mathsf{A}x.\ e_1 \mid \mathsf{B}y.\ e_2)$$

$e \to e'$ with 1 rule: $\dfrac{e \xrightarrow{\text{P}} e'}{E[e] \to E[e']}$

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

$$\dfrac{}{(\lambda x.\ e)\ v \xrightarrow{\text{P}} e[v/x]} \qquad \dfrac{}{(v_1, v_2).1 \xrightarrow{\text{P}} v_1} \qquad \dfrac{}{(v_1, v_2).2 \xrightarrow{\text{P}} v_2}$$

$$\dfrac{}{\textsf{match}\ \mathsf{A}(v)\ \textsf{with}\ \mathsf{A}x.\ e_1 \mid \mathsf{B}y.\ e_2 \xrightarrow{\text{P}} e_1[v/x]}$$

$$\dfrac{}{\textsf{match}\ \mathsf{B}(v)\ \textsf{with}\ \mathsf{A}y.\ e_1 \mid \mathsf{B}x.\ e_2 \xrightarrow{\text{P}} e_2[v/x]}$$

# 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

# CPS

***Everything takes a continuation, all the time!***

```
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

$$\frac{???}{\tau_1 \to \tau_2 \leq \tau_3 \to \tau_4}$$

# 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*

$$\frac{}{\Delta;\Gamma \vdash x : \Gamma(x)} \qquad \frac{}{\Delta;\Gamma \vdash c : \mathsf{int}}$$

$$\frac{\Delta;\Gamma, x{:}\tau_1 \vdash e : \tau_2 \qquad \Delta \vdash \tau_1}{\Delta;\Gamma \vdash \lambda x{:}\tau_1.\ e : \tau_1 \to \tau_2} \qquad \frac{\Delta;\Gamma \vdash e_1 : \tau_2 \to \tau_1 \qquad \Delta;\Gamma \vdash e_2 : \tau_2}{\Delta;\Gamma \vdash e_1\ e_2 : \tau_1}$$

$$\frac{\Delta, \alpha;\Gamma \vdash e : \tau_1}{\Delta;\Gamma \vdash \Lambda\alpha.\ e : \forall\alpha.\tau_1} \qquad \frac{\Delta;\Gamma \vdash e : \forall\alpha.\tau_1 \qquad \Delta \vdash \tau_2}{\Delta;\Gamma \vdash e[\tau_2] : \tau_1[\tau_2/\alpha]}$$

$$(\Lambda\alpha.\ \Lambda\beta.\ \lambda x : \alpha.\ \lambda f{:}\alpha \to \beta.\ f\ x)\ [\mathsf{int}]\ [\mathsf{int}]\ 3\ (\lambda y : \mathsf{int}.\ y + y)$$


# 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