Last lecture: reduction rules for the lambda calculus
This lecture: reduction strategies
As a computer scientist, we can view nondeterminism in the rules as a challenge—it is easier to implement deterministic machines.
As a computer scientist, we can view nondeterminism in the rules as a challenge—it is easier to implement deterministic machines.

\[
((\text{lambda } (x) \; x) \; ((\text{lambda } (z) \; z) \; y))
\]

\[
((\text{lambda } (x) \; x) \; y) \quad \text{β} \quad \text{β} \\
\text{β} \quad \text{β} \\
y 
\]

\[
((\text{lambda } (z) \; z) \; y)
\]
We will assume a few basic, but **important**, choices:
- Evaluation of a term will occur **top-down**
We will assume a few basic, but **important**, choices:
- Evaluation of a term will occur **top-down**
- We will never reduce **under a lambda**
We will assume a few basic, but important, choices:
- Evaluation of a term will occur top-down
- We will never reduce under a lambda

\[
\text{(lambda (x) ((lambda (y) (y y)) (lambda (y) (y y))))}
\]

We say that lambda expressions are in **Weak Head Normal Form (WHNF)**

Even though a potential redex exists under the lambda, we will not evaluate it (until application)
Two popular strategies:
- Call by value, reduce arguments *early* as possible
- Call by name, reduce arguments *late* as possible
Two popular strategies:
- Call by value, reduce arguments **early** as possible
  - Applicative order (innermost), but **not under lambdas**
- Call by name, reduce arguments **late** as possible
  - Normal order, but **not under lambdas**
Whenever you get to an application of a lambda, you have a choice:
- Attempt to evaluate argument?
- Perform application immediately

\[
(((\text{lambda } (x) \ x)) \ ((\text{lambda } (z) \ z) \ y))
\]

\[
(((\text{lambda } (x) \ x) \ y) \ ((\text{lambda } (z) \ z) \ y))
\]
Church-Rosser Theorem

For any expression e,
If \( e \rightarrow^* e_0 \) and \( e \rightarrow^* e_1 \)
Then, both \( e_0 \) and \( e_1 \) step to some \textit{common} term \( e' \)
Church-Rosser Theorem

For any expression $e$, if $e \rightarrow^* e_0$ and $e \rightarrow^* e_1$, then both $e_0$ and $e_1$ step to some common term $e'$.

Corollary: all terminating paths result in same normal form!
Give the **reduction sequences** using…
- Call-by-Name
- Call-by-Value

```
((lambda (x) x) ((lambda (y) y) (lambda (y) y)))
```
Give the **reduction sequences** using…
- Call-by-Name
- Call-by-Value

\[
((\lambda (x) x) ((\lambda (y) y) (\lambda (y) y)))
\]

- CBN

\[
((\lambda (y) y) (\lambda (y) y))
\]

- CBV

\[
(\lambda (y) y)
\]
Up to alpha equivalence, evaluate this term using:
- Call-by-Name
- Call-by-Value

\[
((\lambda x (\lambda y y))
((\lambda x (x x)) (\lambda x (x x))))
\]
Up to alpha equivalence, evaluate this term using:
- Call-by-Name
- Call-by-Value

(((\lambda (x) (\lambda (y) y))
  ((\lambda (x) (x x)) (\lambda (x) (x x))))
  (\lambda (y) y))

CBN
Up to alpha equivalence, evaluate this term using:
- Call-by-Name
- Call-by-Value

\[
(((\lambda (x) (\lambda (y) y))
  ((\lambda (x) (x x)) (\lambda (x) (x x))))

((\lambda (x) (\lambda (y) y))
  ((\lambda (x) (x x)) (\lambda (x) (x x))))
\]

CBV

CBN
**Standardization theorem**

If an expression can be evaluated to WHNF (i.e., it doesn’t loop), then it has a normal-order reduction sequence.

In other words: the lazy semantics is most permissive, in terms of termination.