

Aspects to formalize

- Syntax: what's a syntactically well-formed program?
 EBNF notation for a context-free grammar
- Static semantics: which syntactically well-formed programs are semantically well-formed? which programs type-check?
- typing rules, well-formedness judgments
 Dynamic semantics: what does a program
- evaluate to or do when it runs?operational, denotational, or axiomatic semantics
- Metatheory: properties of the formalization itself
- E.g. do the static and dynamic semantics match? i.e., is the static semantics **sound** w.r.t. the dynamic semantics?

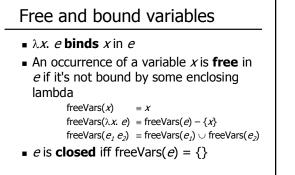
Approach

- Formalizing full-sized languages is very hard, tedious
 - many cases to consider
 - lots of interacting features
- Better: boil full-sized language down into essential core, then formalize and study the core
 - cut out as much complication as possible, without losing the key parts that need formal study
 - hope that insights gained about core will carry back to full-sized language

The lambda calculus

- The essential core of a (functional) programming language
 - The tiniest Turing-complete programming language
- Outline:
 - Untyped: syntax, dynamic semantics, cool properties
 - Simply typed: static semantics, soundness, more cool properties
 - Polymorphic: fancier static semantics

Untyped λ -calculus: syntax• (Abstract) syntax:e ::= x variable $|\lambda x. e$ function/abstraction $(\cong fn \ x => e)$ $|e_1 e_2$ call/application• Freely parenthesize in concrete syntax to imply
the right abstract syntax• The trees described by this grammar are
called **term trees**



α -renaming

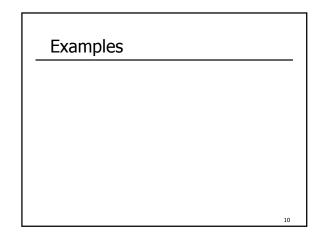
- First semantic property of lambda calculus: bound variables in a term tree can be renamed (properly) without affecting the semantics of the term tree
 - α-equivalent term trees
 - $(\lambda X_1, X_2, X_1) \Leftrightarrow_a (\lambda X_3, X_2, X_3)$ cannot rename free variables
- term e: e and all α-equivalent term trees
 - Can freely rename bound vars whenever helpful

Evaluation: β -reduction

- Define what it means to "run" a lambdacalculus program by giving simple reduction/rewriting/simplification rules

 - " $e_1 \rightarrow_{\beta} e_2$ " means " e_1 evaluates to e_2 in one step"
- One case:

 - $(\lambda x, e_I) e_2 \rightarrow_{\beta} [x \rightarrow e_2] e_I$ "if you see a lambda applied to an argument expression, rewrite it into the lambda body where all free occurrences of the formal in the body have been replaced by the argument expression"



Substitution

When doing substitution, must avoid changing the meaning of a variable occurrence

$$[X \rightarrow e]X = e$$

$$[X \rightarrow e]Y = Y, \text{ if } X \neq Y$$

$$[x \rightarrow e](\lambda x, e_{\lambda}) = (\lambda x, e_{\lambda})$$

$$[X \rightarrow e](\lambda x. e_2) = (\lambda x. e_2)$$

$$[X \rightarrow e](\lambda y. e_2) = (\lambda y. [X \rightarrow e]e_2), \text{ if }$$

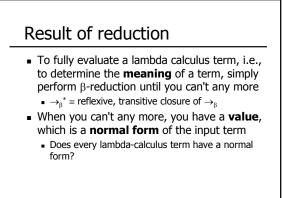
$$[x \rightarrow e](\lambda y. e_2) = (\lambda y. [x \rightarrow e]e_2), \text{ if } x \neq y$$

and y not free in e

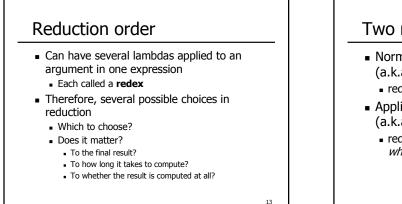
- $[X \rightarrow e](e_1 e_2) = ([X \rightarrow e]e_1)([X \rightarrow e]e_2)$
- use α-renaming to ensure "y not free in e"

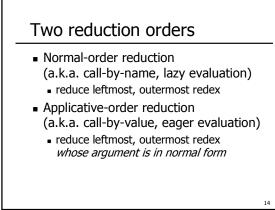
11

q



12





Amazing fact #1: Church-Rosser Theorem, Part 1 • Thm. If $e_1 \rightarrow_{\beta}^{*} e_2$ and $e_1 \rightarrow_{\beta}^{*} e_3$, then $\exists e_4$ such that $e_2 \rightarrow_{\beta}^{*} e_4$ and $e_3 \rightarrow_{\beta}^{*} e_4$ • Corollary. Every term has a unique normal form, if it has one • No matter what reduction order is used! • Wow!

Existence of normal forms?

- Does every term have a normal form?
- Consider: (λx. x x) (λx. x x)

Amazing fact #2: Church-Rosser Theorem, Part 2 • If a term has a normal form, then

- If a term has a normal form, then normal-order reduction will find it!
 Applicative order reduction might active
 - Applicative-order reduction might not!
- Example:
 - $\bullet (\lambda X_1 \cdot (\lambda X_2 \cdot X_2)) ((\lambda X \cdot X \cdot X) (\lambda X \cdot X \cdot X))$

17

16