

Evaluation rules

The single-step evaluation relation $t \longrightarrow t'$ is the smallest relation closed under the following rules:

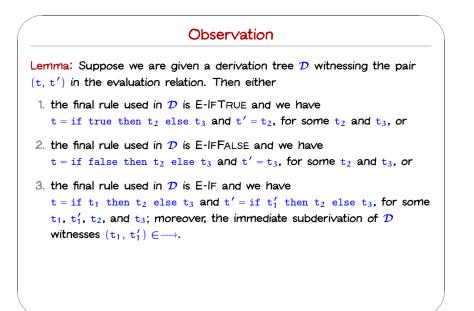
if true then
$$t_2$$
 else $t_3 \longrightarrow t_2$ (E-IFTRUE)

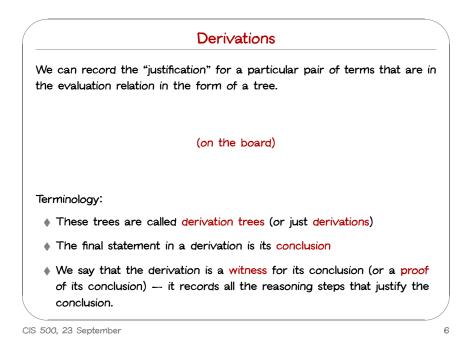
if false then t_2 else $t_3 \longrightarrow t_3$ (E-IFFALSE)

$$t_1 \longrightarrow t'_1$$
 (E-IF)

if
$$t_1$$
 then t_2 else $t_3 \longrightarrow$ if t_1' then t_2 else t_3

CIS 500, 23 September





Induction on Derivations

Combining the previous ideas, we can write proofs about evaluation "by induction on derivation trees." E.g....

 $\label{eq:theorem: for the size} \mbox{Theorem: If } t \longrightarrow t' \mbox{ - i.e., if } (t,\,t') \in \mbox{ - then } \mbox{size}(t) \mbox{ > size}(t').$

Proof: By induction on a derivation of $t \longrightarrow t'$.

For each step of the induction, we assume the desired result for all smaller derivations and proceed by a case analysis (using the previous lemma) of the final evaluation rule used in constructing the derivation tree.

CIS 500, 23 September

7

Aside Aside Q: Why are we bothering to prove all these completely obvious facts about terms and evaluation? Q: Why are we bothering to prove all these completely obvious facts about terms and evaluation? A: Suppose you told one of these facts to someone and they replied, "I don't believe it!" How would you convince them, aside from just saying, "Well, look at it again... isn't it obvious?" I.e., we're trying to draw out why it is obvious. Cts 500, 23 September 9 Cts 500, 23 September

Aside

Q: Why are we bothering to prove all these completely obvious facts about terms and evaluation?

A: Suppose you told one of these facts to someone and they replied, "I don't believe it!" How would you convince them, aside from just saying, "Well, look at it again... isn't it obvious?"

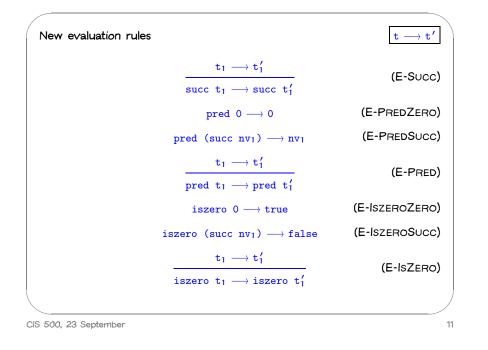
I.e., we're trying to draw out why it is obvious.

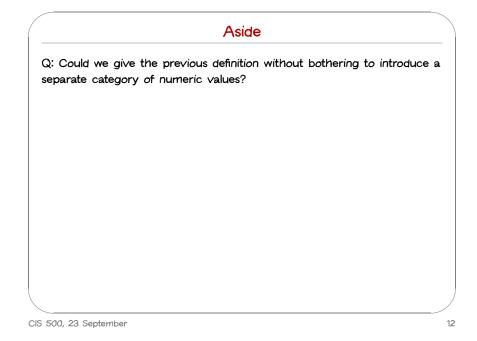
A': Facts almost this obvious have a habit of being false.

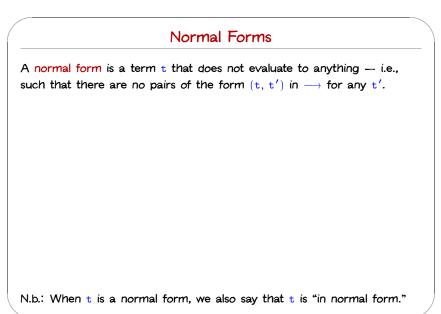
Doing the proofs is a methodology for debugging definitions.

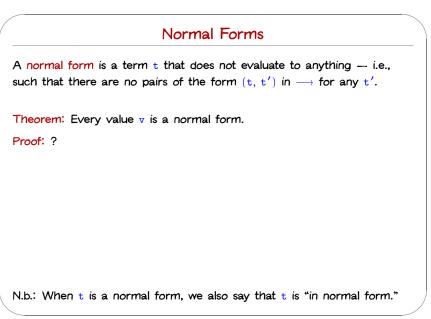
Numbers		
	terms	
	constant zer	
	successo	
	predecesso	
	zero tes	
	values	
	numeric valu	
	numeric values	
	zero valu	
	successor valu	

CIS 500, 23 September









CIS 500, 23 September

13

CIS 500, 23 September

Is the converse true? I.e., is every normal form a value?

Stuck terms

Is the converse true? I.e., is every normal form a value?

No: some terms are stuck.

Formally, a stuck term is one that is a normal form but not a value. Stuck terms model run-time errors.

CIS 500, 23 September

14

Multi-step evaluation.

The multi-step evaluation relation, written \rightarrow^* , is the reflexive, transitive closure of one-step evaluation.

That is, it is the smallest relation such that

1. if $t \longrightarrow t'$ then $t \longrightarrow^* t'$,

2. $t \longrightarrow^* t$ for all t, and

3. if $t \longrightarrow t'$ and $t' \longrightarrow t''$, then $t \longrightarrow t''$.

Termination of evaluation

Theorem: For every t there is some normal form t' such that t \longrightarrow^* t'. Proof:

CIS 500, 23 September

CIS 500, 23 September

14-a

Termination of evaluation

Theorem: For every t there is some normal form t' such that $t \longrightarrow^* t'$. Proof:

First, recall that single-step evaluation strictly reduces the size of the term:

if $t \longrightarrow t'$, then size(t) > size(t')

Now, assume (for a contradiction) that

$t_0, t_1, t_2, t_3, t_4, \ldots$

is an infinite-length sequence such that

```
\mathtt{t}_0, \longrightarrow \mathtt{t}_1, \longrightarrow \mathtt{t}_2, \longrightarrow \mathtt{t}_3, \longrightarrow \mathtt{t}_4 \longrightarrow \boldsymbol{\cdots},
```

Then

$\text{size}(\texttt{t}_0),\,\text{size}(\texttt{t}_1),\,\text{size}(\texttt{t}_2),\,\text{size}(\texttt{t}_3),\,\text{size}(\texttt{t}_4),\,\ldots$

is an infinite, strictly decreasing sequence of natural numbers.

```
• But such a sequence cannot exist - contradiction!
```

CIS 500, 23 September

16-a

Termination Proofs

Most termination proofs have the same basic form:

Theorem: The relation $R \subseteq X \times X$ is terminating — i.e., there are no infinite sequences x_0 , x_1 , x_2 , etc. such that $(x_i, x_{i+1}) \in R$ for each i.

Proof:

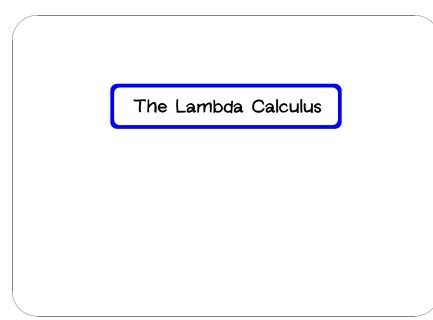
1. Choose

 \blacklozenge a well-founded set (W,<) — i.e., a set W with a partial order < such that there are no infinite descending chains

 $w_0 > w_1 > w_2 > \dots \text{ in } W$

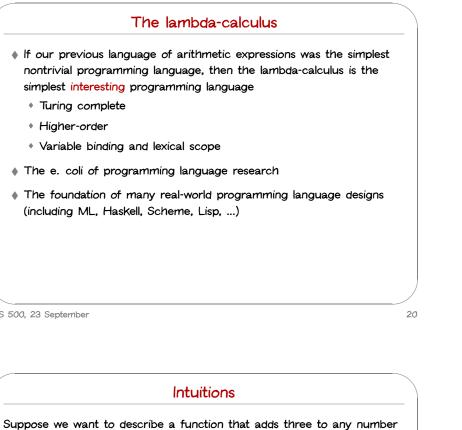
- ♦ a function f from X to W
- 2. Show f(x) > f(y) for all $(x, y) \in R$
- 3. Conclude that there are no infinite sequences x_0 , x_1 , x_2 , etc. such that $(x_i, x_{i+1}) \in R$ for each i), since, if there were, we could construct an infinite descending chain in W.

CIS 500, 23 September



More examples (time permitting)

- Nondeterministic choice
- Simple parallel composition
- ♦ A one-element memory



we pass it. We might write
plus3 x = succ (succ (succ x))
That is, "plus3 x yields succ (succ (succ x))."

Intuitions

Suppose we want to describe a function that adds three to any number

CIS 500, 23 September

CIS 500, 23 September

we pass it. We might write

```
plus3 x = succ (succ (succ x))
```

That is, "plus3 x yields succ (succ (succ x))."

Q: What is plus3 itself?

Intuitions

Suppose we want to describe a function that adds three to any number we pass it. We might write

```
plus3 x = succ (succ (succ x))
```

That is, "plus3 x yields succ (succ (succ x))."

Q: What is plus3 itself?

A: plus3 is the function that, given x, yields succ (succ (succ x)).

CIS 500, 23 September

Intuitions

Suppose we want to describe a function that adds three to any number we pass it. We might write

```
plus3 x = succ (succ (succ x))
```

That is, "plus3 x yields succ (succ (succ x))."

Q: What is plus3 itself?

A: plus3 is the function that, given x, yields succ (succ (succ x)).

plus3 = λx . succ (succ (succ x))

This function exists independent of the name plus3.

CIS 500, 23 September

21-c

What's new?

We have introduced two new primitive syntactic forms:

 \blacklozenge abstraction of a term t on some subterm x:

$\lambda \texttt{x. t}$

"The function that, when applied to a value v, yields t with v in place of x."

application of a function to an argument:

$\mathtt{t}_1 \ \mathtt{t}_2$

"the function t_1 applied to the argument t_2 "

Intuitions

Suppose we want to describe a function that adds three to any number we pass it. We might write

plus3 x = succ (succ (succ x))

That is, "plus3 x yields succ (succ (succ x))."

Q: What is plus3 itself?

A: plus3 is the function that, given x, yields succ (succ (succ x)).

plus3 = λx . succ (succ (succ x))

This function exists independent of the name plus3.

On this view, plus3 (succ 0) is just a convenient shorthand for "the function that, given x, yields succ (succ (succ x)), applied to succ 0."

plus3 (succ 0) = $(\lambda x. \text{ succ } (\text{succ } x)))$ (succ 0)

CIS 500, 23 September

21-d

What's new?

We have introduced two new primitive syntactic forms:

abstraction of a term t on some subterm x:

$\lambda x. t$

"The function that, when applied to a value v, yields t with v in place of x."

application of a function to an argument:

t1 t2

"the function t_1 applied to the argument t_2 "

Note that abstractions are anonymous. For convenience in examples, we will sometimes write things like

Let plus3 be λx . succ (succ (succ x)) and consider the term plus3 (succ 0)

But the naming here is a metalanguage operation — the names are not part of the object language under discussion.

CIS 500, 23 September

Abstractions over Functions

Consider the λ -abstraction

 $g = \lambda f. f (f (succ 0))$

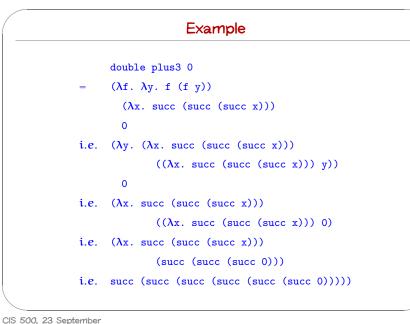
Note that the parameter variable f is used in the function position in the body of g. Terms like g are called higher-order functions.

If we apply g to an argument like plus3, the "substitution rule" yields a nontrivial computation:

g plus3 = $(\lambda f. f (f (succ 0))) (\lambda x. succ (succ (succ x)))$

- i.e. $(\lambda x. \text{ succ } (\text{succ } x)))$ $((\lambda x. \text{ succ } (\text{succ } x))) (\text{succ } 0))$ i.e. $(\lambda x. \text{ succ } (\text{succ } x)))$
 - (succ (succ (succ (succ 0))))
- i.e. succ (succ (succ (succ (succ (succ 0)))))

CIS 500, 23 September



Abstractions Returning Functions

Consider the following variant of g:

double = λf . λy . f (f y)

I.e., double is the function that, when applied to a function f, yields a function that, when applied to an argument y, yields f(f y).

CIS 500, 23 September

24

The Pure Lambda-Calculus

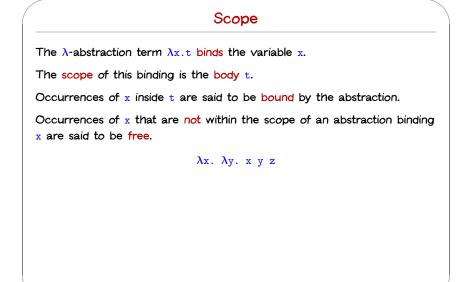
As the preceding examples suggest, once we have λ -abstraction and application, we can throw away all the other language primitives and still have left a rich and powerful programming language.

In this language - the "pure lambda-calculus" - everything is a function.

- Variables always denote functions
- Functions always take other functions as parameters
- The result of a function is always a function

25

		Syntax	
		t ::=	terms:
		x	variable
Formalities		$\lambda x.t$ t t	abstraction application
CIS 500, 23 September	27	Terminology: terms in the pure λ-calculus are often called terms of the form λx. t are called λ-abstra 	



Scope

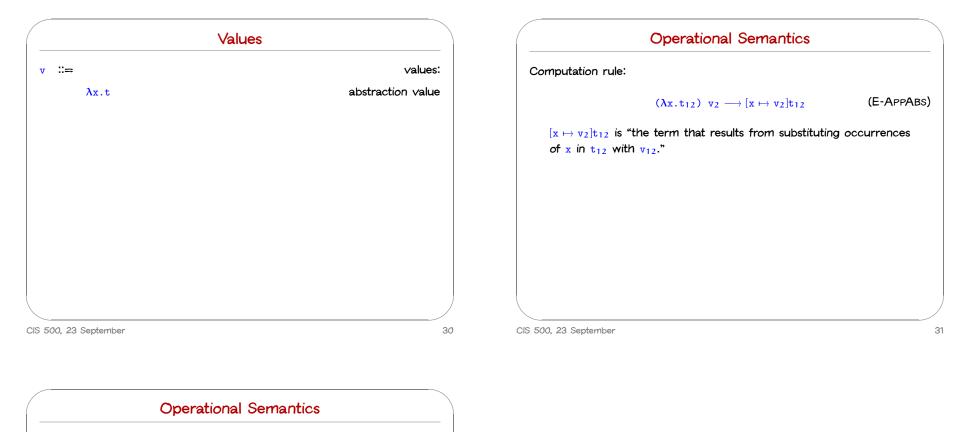
The λ -abstraction term $\lambda x.t$ binds the variable x.

The scope of this binding is the body t.

Occurrences of x inside t are said to be bound by the abstraction.

Occurrences of x that are not within the scope of an abstraction binding x are said to be free.

 $\lambda x. \lambda y. x y z$ $\lambda x. (\lambda y. z y) y$



Computation I	rule:
---------------	-------

 $(\lambda \mathtt{x}.\mathtt{t}_{12}) \hspace{0.1cm} \mathtt{v}_{2} \longrightarrow [\mathtt{x} \mapsto \mathtt{v}_{2}]\mathtt{t}_{12}$

(E-APPABS)

 $[\mathtt{x} \mapsto \mathtt{v}_2]\mathtt{t}_{12}$ is "the term that results from substituting occurrences of x in t_{12} with v_{12} ."

Congruence rules:

$$\frac{t_1 \longrightarrow t'_1}{t_1 \ t_2 \longrightarrow t'_1 \ t_2}$$
(E-App1)

$$\frac{t_2 \longrightarrow t'_2}{v_1 \ t_2 \longrightarrow v_1 \ t'_2}$$
(E-App2)

$$\mathtt{t}_2 \longrightarrow \mathtt{v}_1 \ \mathtt{t}_2'$$