① <var>

(<var>)

$\underbrace{(x) \equiv x}_{\text{as expressions}}$

② <var>

use instead of True & False

! <var>

$! x$

y

③ f $\quad \backslash x \to (s t)$

$f\,(s\ t)$ or $(f\ s)\ t$

Last lecture on reduction & evaluation 18/10

Done

big step & small step reduction for lam

big step " " IMP

we used substitution

$$\frac{f \Downarrow \lambda x \to t[x] \quad t[a] \Downarrow v}{f\,a \Downarrow v} \quad \text{CBN}$$

What is this

it is substitution

This lecture : Do big step reduction, without substitution

+ve : We won't need to implement substitution

-ve : Bit more work

Call By Value

[Reduce input before evaluating body of λ-term]

Judgement

$$\Theta \Vert t \Downarrow V$$

environment

term

values

contains values for the free variables of t

$$\langle val \rangle ::= \langle number \rangle$$
$$\quad | \ \boxed{env}\ \langle var \rangle \longrightarrow \langle term \rangle$$

$$\langle env \rangle ::= $$
$$\quad | \ \langle env \rangle,\ \langle var \rangle = \langle val \rangle$$

Some environments

<val> ::= <number>
| [<env>] <var> → <term>

<env> ::=
| <env>, <var> = <value>

example enviroments

$, x = 6$

$, x = 6, y = 5$

$, x = 6, y = 5, z = 7$

$, x = 6, y = 5, z = 7, w = [\quad] x \to x + 6$

$, x = 6, w = [, y = 7] x \to x + 6$

$, x = 6, w = \underbrace{[, y = [\,] z \to z]\ y \to y + 8}_{\text{value}}$

variable

square brackets

is this an environment

Big step Reduction for Lam with closures/enviroments

- one rule for every syntactic form

$\theta \mid t \Downarrow v$

$$\frac{}{\theta \mid n \Downarrow n} \qquad \frac{\theta \mid s \Downarrow n \quad \theta \mid t \Downarrow m}{\theta \mid s+t \Downarrow n+m}$$

this is a value, so we can just return it

$$\frac{}{\gamma, x=v, \gamma' \mid x \Downarrow v}$$

$$\frac{}{\gamma \mid \backslash x \to t \Downarrow [\gamma] x \to t}$$

NOT $\backslash x \to t$, not a value

$$\frac{\gamma \mid f \Downarrow [\gamma'] x \to t \quad \gamma \mid a \Downarrow v \quad \gamma', x=v \mid t \Downarrow v'}{\gamma \mid f a \Downarrow v'}$$

Example

$$\frac{}{\vdash ((\backslash x \to \backslash y \to x+y)\ 5)\ 6 \Downarrow}$$

$$\frac{\vdash \backslash x \to \backslash y \to x+y \Downarrow [\,]\ x \to \backslash y \to x+y \qquad \vdash 5 \Downarrow 5}{\vdash (\backslash x \to \backslash y \to x+y)\ 5 \Downarrow}$$

$$\frac{}{, x=5 \vdash \backslash y \to x+y \Downarrow [, x=5]\ y \to x+y}$$

$$\frac{\vdash (\backslash x \to \backslash y \to x+y)\ 5 \Downarrow [, x=5]\ y \to x+y \qquad \vdash 6 \Downarrow 6}{\vdash ((\backslash x \to \backslash y \to x+y)\ 5)\ 6 \Downarrow 11}$$

$$\frac{[, x=5, y=6] \vdash x \Downarrow 5 \qquad [, x=5, y=6] \vdash y \Downarrow 6}{[, x=5, y=6] \vdash x+y \Downarrow 11}$$

W/10

$$t[a] \Downarrow v' \qquad \text{CBN}$$

$$\frac{f \Downarrow \backslash x \to t[x] \quad a \Downarrow v \quad t[v] \Downarrow v'}{f\ a \Downarrow v'} \text{CBV}$$

<u>choice</u> (i) Shall we reduce a & substitute into t — CBV

(ii) Shall we substitute a as it <u>currently</u> is into t

— CBN

CBN
& CBV

$$
\begin{tikzcd}[column sep=small]
 & (\backslash x \to x + x)\ (5+5) \arrow[dl, "\text{CBN}"] \arrow[dr, "\text{CBV}"] & \\
(5+5) + (5+5) \arrow[d] & & (\backslash x \to x+x)\ 10 \arrow[d] \\
10 + 10 \arrow[d] & & 10+10 \arrow[d] \\
20 & & 20
\end{tikzcd}
$$

Here CBV better

$$\begin{tikzcd}
 & (\lambda x \to 5)\ (10+10) \arrow[dl, "\mathrm{CBN}"'] \arrow[dr, "\mathrm{CBV}"] & \\
5 & & (\lambda x \to 5)(20 \arrow[d] \\
 & & 5
\end{tikzcd}$$

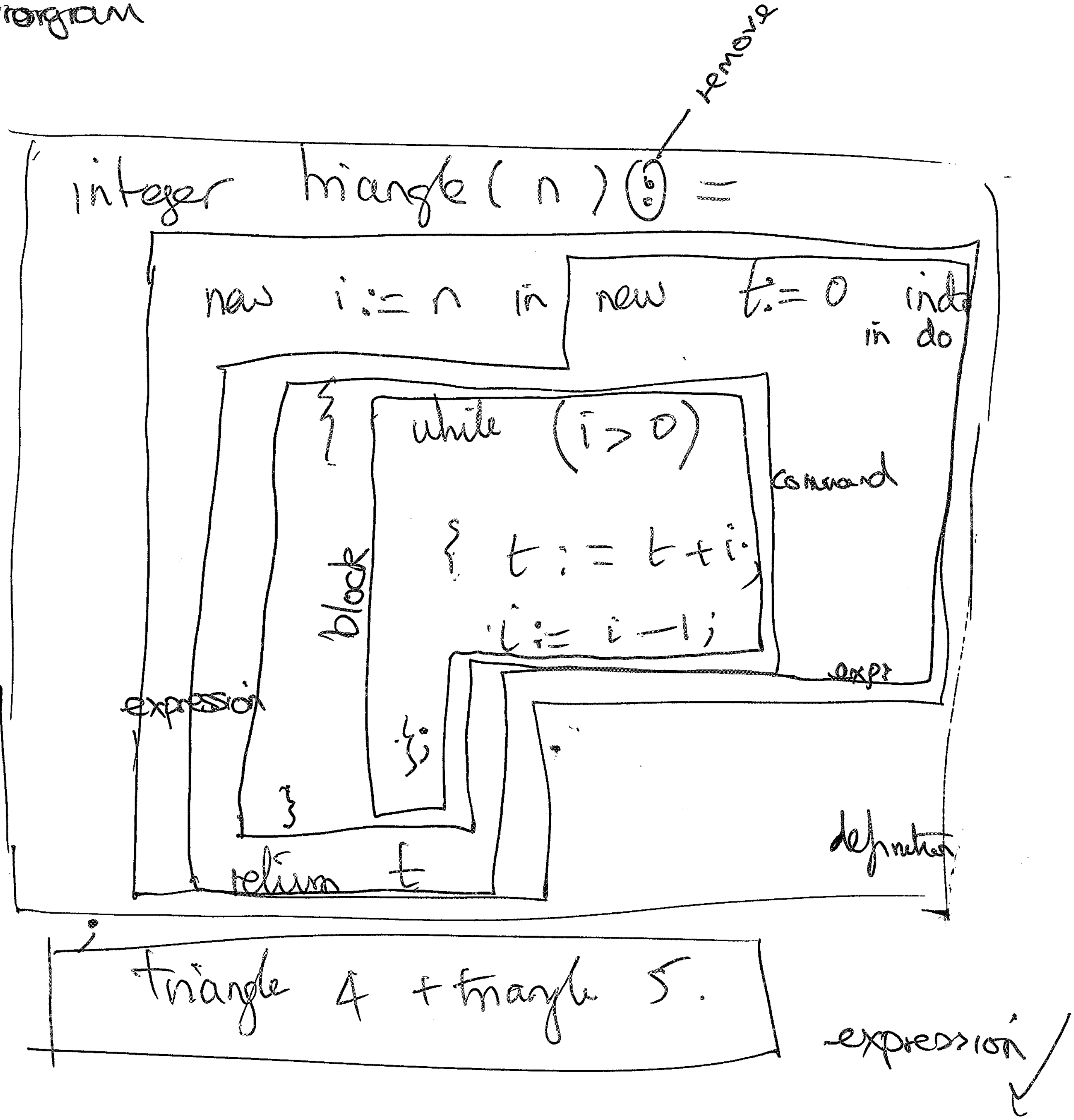
This time CBN
better than
CBV

$$\frac{t \Downarrow n \qquad s \Downarrow m}{(t+s) \Downarrow n+m}$$

$$\frac{t \Downarrow 3 \qquad \backslash x \to 5 \Downarrow \backslash x \to 5}{t + (\backslash x \to 5)}$$

doesnt reduce to anything

program


remove
integer triangle(n) : =
new i := n in new t := 0 in do
{
while (i > 0)
command
{ t := t + i;
block
i := i − 1;
expr
expression
;}
}
return t
definition
;
triangle 4 + triangle 5.
expression

$f \ \backslash x \to s\ t$

$f\ (\backslash x \to s t)$

$f((\backslash x \to s)\ \ t)$

$(f\ (\backslash x \to s))\ t$

convention

$3+5+7$

$f\ s\ t \Rightarrow f\ (s\ \ t)$

$f\ s\ t \xrightarrow{\text{means}} (fs)(t)$

Application associates to the left.

<bool> ::= <var>

| True

| False

| <bool> & <bool>

| <bool> || <bool>

| ~~not~~ !<bool>

| <bool> => <bool>

alternative

| <const>

alternative

<bool> <bop> <bool>

<const> ::= True

| False

<bop> ::= &

| ||

| =>

ok to have dnf

<dnf> ::= <conj> v <dnf>

| <conj>

<conj> ::= <lit> $\wedge$ <conj>

| <lit>

could be <conj>

<lit> ::= ! <var>

| <var>

brackets ??

$\boxed{A\&B}$ + C&D

A & B s.

def

$\boxed{AB}$ + CD

$\boxed{A, B}$

A + B

$\boxed{A\&B}$ ~~False~~

Normal forms

<nf> ::= ~~<ne>~~ <var>
| \x → <nf>
~~| <ne> <nf>~~
| <ne> <nf>

normal forms

<ne> ::= <var>
| <ne> <nf>

neutral term $\triangleq$ normal form that is not a λ-abstraction

Also

<nf> ::= <ne>
| \x → <nf>

<ne> ::= <var>
| <ne> <nf>

$\langle x \rangle ::= ~~a~~ \; a \; \langle y \rangle \; b$

$\quad\quad\quad \mid \; c$

$\quad\quad\quad \mid \; b \; \langle x \rangle \; a$

$y ::= c \; ~~\langle\ldots\rangle~~ \langle x \rangle \; c$

$\quad\quad \mid \; a$

All $\langle x \rangle$-expressions of length 6 or less.

$\langle x \rangle \longrightarrow ~~a~~ a \langle y \rangle b \longrightarrow ac \langle x \rangle cb$

$a \langle y \rangle b \to aab$

$ac \langle x \rangle cb \to acccb$

$\langle x \rangle \to c$

$\langle x \rangle \to b \langle x \rangle a \longrightarrow ba \langle y \rangle ba$

$b \langle x \rangle a \to bca$

$ba \langle y \rangle ba \to baaba$

$b \langle x \rangle a \to bb \langle x \rangle aa$

$bb \langle x \rangle aa \to bbcaa$

9/10

$\backslash x \to t$

$\lambda x \to t$ ..... means

$\backslash x \to t$

$\backslash x .\ t$ ... means

$$\lambda x. \left((x + y) + (\lambda z. x + z) \right)$$

free variable

$$\lambda x. \left(x + (\lambda x. x) \right)$$

$$(\lambda x. t[x])s \rightsquigarrow t[s]$$

MAKE SURE NO FREE VARIABLE OF S BECOMES BOUND IN t

$\Rightarrow$ IF THIS MIGHT OCCUR, CHANGE $\lambda x. t[x]$ to $\lambda z. t[z]$

to ensure no variables become bound

From Small step
to Big step $\Big\}$ for LAM

Small step Judgement form

$$\boxed{s \rightsquigarrow t}$$

$$\frac{}{(\backslash x \to t[x])s \rightsquigarrow t[s]}$$

$$\frac{}{(m+n) \rightsquigarrow \overline{m+n}} \qquad \frac{t[x] \rightsquigarrow t'[x]}{\backslash x \to t[x] \rightsquigarrow \backslash x \to t'[x]}$$

$$\frac{f \rightsquigarrow f'}{fa \rightsquigarrow f'a} \qquad \frac{a \rightsquigarrow a'}{fa \rightsquigarrow fa'}$$

$$\frac{s \rightsquigarrow s'}{(s+t) \rightsquigarrow (s'+t)} \qquad \frac{t \rightsquigarrow t'}{(s+t) \rightsquigarrow (s+t')}$$

① Reduction doesn't always terminate

$$\Omega = (\backslash x \to \underbrace{x\,x}_{t[x]})\ \underbrace{(\backslash x \to x\,x)}_{s}$$

$$\rightsquigarrow \quad (\backslash x \to x\,x)(\backslash x \to x\,x)$$

② Reduction order matters!!

$$(\backslash x \to \underbrace{5}_{t[x]})\ \underbrace{\Omega}_{s}$$

$$\begin{tikzcd}
& (\backslash x \to 5)\ \Omega \arrow[dl] \arrow[dr] & \\
5 & & (\backslash x \to 5)\ \Omega \arrow[d] \\
& & (\backslash x \to 5)\ \Omega \arrow[d, phantom, "\vdots"] \\
& & {}
\end{tikzcd}$$

$$(\lambda x \to \underbrace{x+x}_{t[x]})\ \underbrace{(2+7)}_{s}$$

$$(2+7) + (2+7) \;\to\; 9 + (2+7) \;\to\; 9+9 \;\to\; 18$$

$$(\lambda x \to \underbrace{x+x}_{t[x]})\ \underbrace{9}_{s} \;\to\; 9+9 \;\to\; 18$$

good news

- same answer
- badnews : the LHS duplicated work

Big step Reduction for LAM

Judgement form $\boxed{t \Downarrow v}$

values $\langle value \rangle ::= \ | \ \cancel{\langle var \rangle} \to \langle term \rangle$
$\qquad\qquad\qquad\qquad | \ \langle number \rangle$

In a λ-abstraction, the body can be anything

Rules

$$\frac{}{n \Downarrow n}$$

$$\frac{t \Downarrow v \qquad s \Downarrow v'}{(t + s) \Downarrow v + v'}$$

$$\frac{}{\backslash x \to t \Downarrow \backslash x \to t}$$

$$\frac{f \Downarrow \backslash x \to t[x] \qquad t[a] \Downarrow v}{f\,a \Downarrow v}$$

chose

to not reduce a first,

call-by-name

I could have writen

$$\frac{f \Downarrow \backslash x \to t[x] \qquad a \Downarrow v \qquad t[v] \Downarrow v'}{f\,a \Downarrow v'}$$

call-by-value

WRITE A BIG STEP REDUCTION

IN ND - tree style for

$$(\lambda x. ((\lambda x. \ x+3)(4+2)))5$$

by call by name reduction

ANS 9.

$$\frac{}{(\lambda x \to t[x])\ s \leadsto t[s]}\ \beta$$

$$\frac{f \leadsto f'}{f\ s \leadsto f'\ s}$$

$$\frac{s \leadsto s'}{f\ s \leadsto f\ s'}$$

?

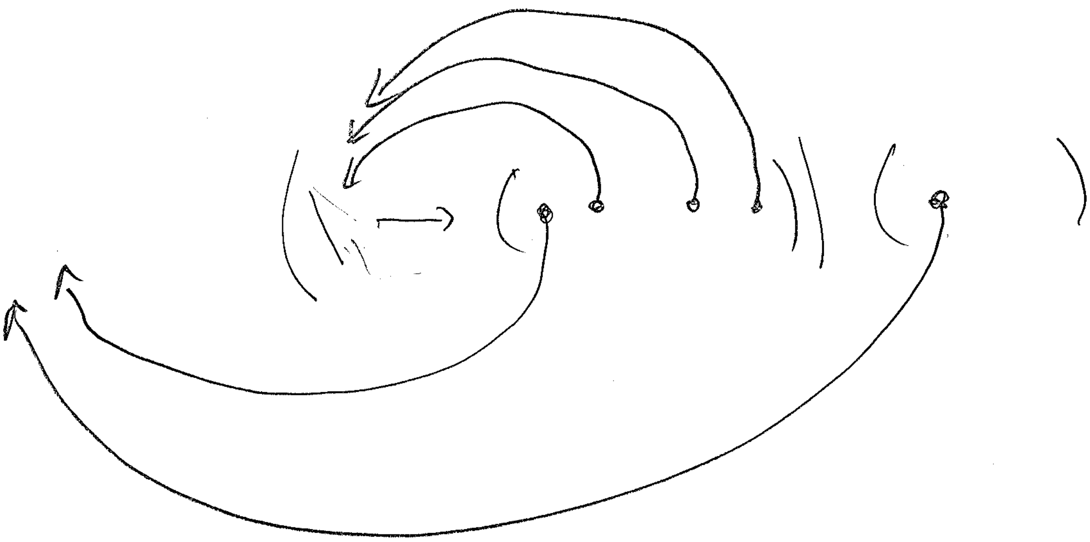
$$\frac{x \vdash t[x] \leadsto t'[x]}{\lambda x \to t[x] \leadsto \lambda x \to t'[x]}$$

$$\frac{}{\underset{\substack{\uparrow \\ \text{syntax}}}{(m + n)} \leadsto \underset{\substack{\uparrow \\ \text{numeric}}}{m+n}}$$

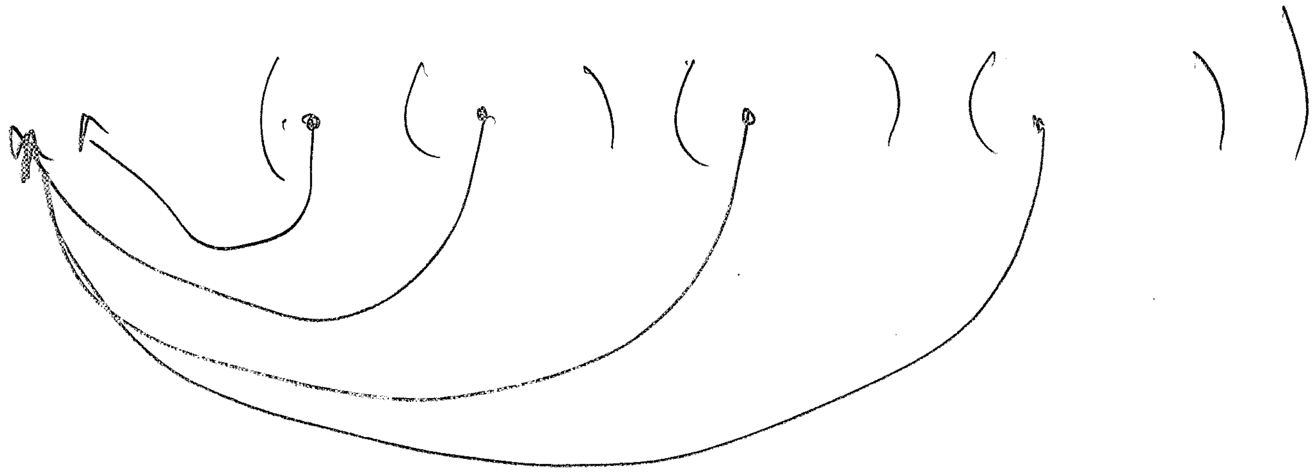
$$\frac{s \leadsto s'}{(s+t) \leadsto (s'+t)} \qquad \frac{t \leadsto t'}{(s+t) \leadsto (s+t')}$$

\f → \x → f (f x)

\ → \ → 1 (1 0)



~>



~~σ|if p then~~

$$\frac{\sigma | p \Downarrow \sigma' | 0}{\sigma | p \,\&\, q \Downarrow \sigma' | 0}$$

Small step semantics for LAM

incremental little steps

Thur 4/10

<term> ::= <var>
| <term> <term>
| $\boxed{\backslash\ \langle var \rangle \to \langle term \rangle}$
| <number>
| <term> + <term>

$\backslash\, x \to t$

mean

the program that takes x as input & feeds it into t

SCOPE

$\lambda x.\ x+5$

is the program that takes x as input & returns $x+5$

The scope of x in $\lambda x \to t$ is all of t

all x in t are "bound" by the x in the Lambda term

example

$\lambda x. ((\lambda x.\ x+5)\ x)$

So $\lambda x \to x+x$ is the same as $\lambda y \to y+y$

Computation in lam
is given by

$$\frac{}{(\backslash x \to t[x])\, s \rightsquigarrow t[s]}$$

symbol for "reduces to"

This is a <u>redex</u>, ie a lambda term
applied to another term

The reduction intuitively reduces
the program that supplies s as
input to the program that takes x
as input & does t
to
the program t where x is replaced by s

Example reductions

$$\beta: (\backslash x \to t[x])\, s \rightsquigarrow \underline{t[s]}$$

this is not "t[s]" but t with x replaced by s

$$(\backslash x \to \underbrace{x+x}_{t})\, \underbrace{5}_{s} \rightsquigarrow 5+5$$

$$\rightsquigarrow 10$$

$$(\backslash y \to \underbrace{x+y}_{t})\, \underbrace{10}_{s}$$

$$\rightsquigarrow x+10$$

$$(\backslash x \to t[x])\ s \rightsquigarrow t[s]$$

$$\Big(\underbrace{\big(\backslash z \to \underbrace{(\backslash k \to z+k)}_{\text{a lambda term}}\big)}_{\text{lambda term}}\ \underbrace{5}_{\text{applied to}}\Big)\ 10$$

$$t \equiv \backslash k \to z+k$$

$$s \equiv 5$$

$$t[s] \equiv \backslash k \to 5+k$$

$$\rightsquigarrow (\backslash k \to 5+k)\ 10$$

$$\rightsquigarrow 5+10$$

$$\rightsquigarrow 15$$

$$(\lambda x \to t[x])\,s \rightsquigarrow t[s]$$

applied

$$\lambda x \to (t + ((\lambda x \to 7)\;9))$$

is this the start of a redex

is this the start of a redex $\Rightarrow$ Yes

- variable bound is x
- t is 7
- s is 9

so $(\lambda x \to 7)\;9 \rightsquigarrow 7$

$$(\backslash x \to t[x]) \leadsto t[s]$$

$$\underbrace{(\backslash x \to (\backslash y \to x+y))}_{\text{lambda term}} \underbrace{(y+2)}_{\text{applied to something}}$$

bound variable is x

t is $\backslash y \to x+y$

s is $y+2$

So what is $t[s]$

take $\backslash y \to x+y$ & replace x with $y+2$

WRONG AS $\backslash y \to (y+2) + y$

THIS y becomes bound by this $\backslash$

$$(\backslash x \to (\backslash y \to x+y))\,(y+2)$$

$$\rightsquigarrow \backslash z \to (y+2)+z$$

MORAL check in t if there are variables bound that also occur in s

If so ... rename the bound variable in t

$$(\backslash x \to \boxed{(\backslash y \to x+y)})\ (y+2)$$

$$\equiv$$

$$(\backslash x \to \boxed{(\backslash z \to x+z)})\ (y+2)$$

Thurs 2/Oct

$$\dfrac{\dfrac{5 \Downarrow 5 \qquad 4 \Downarrow 4}{(5+4) \Downarrow 9} \qquad 3 \Downarrow 3}{(5+4) - 3 \Downarrow 6}$$

$$\dfrac{e_1 \Downarrow n_1 \qquad e_2 \Downarrow n_2}{e_1 - e_2 \Downarrow n_1 - n_2}$$

- on expressions ... its syntax

- on numbers ... so do the subtraction

$\sigma | e \Downarrow \sigma' | v$

$\sigma | c \Downarrow \sigma'$ etc.

$\vdots$

Consider $\quad$ new $x := 6$ in (do $x := x+1$ return x)

new $x := 37$ in

(new $x := 42$ in $\underline{\text{do } x := x+1 \text{ return } x}$)

this x has scope

this x has scope

To understand this, you need to understand SCOPE

Key point

in a memory

$$x_1 = n_1, \; x_2 = n_2 \; .... \quad x_5 = n_5$$

some of these x's are the same.

eg x_1 & x_3 could be the same variable

So the x listed is in scope is the right most x

Use of this in assignment

x does not occur in σ'

$$\frac{\sigma \mid e \Downarrow \sigma_1, x := v', \sigma' \mid v}{\sigma \mid x := e \Downarrow \sigma_1, x := v, \sigma'}$$

Judgement for assignment

Judgement for new

$$\frac{\sigma_0 \mid e \Downarrow \sigma_1 \mid v \qquad \sigma_1, x := v \mid e' \Downarrow \sigma_2 \mid v_2}{\sigma_0 \mid \text{new } x := e \text{ in } e' \Downarrow \sigma_3 \mid v_2}$$

where $\sigma_2 = \sigma', x := v_3, x \backslash \sigma''$

$\sigma_3 = \sigma', \sigma''$

boolean expressions

$$\frac{\sigma_0 \mid p \Downarrow \sigma_1 \mid b_1 \qquad \sigma_1 \mid p_1 \Downarrow \sigma_2 \mid b_2}{\sigma_0 \mid p \,\&\, p_1 \Downarrow \sigma_2 \mid \begin{cases} T & \text{if } b_1 = T \text{ and } b_2 = T \\ F & \text{otw} \end{cases}}$$

$$\boxed{\sigma \mid p \Downarrow \sigma' \Downarrow b}$$

judgement form.

true or false

What is b_1 is False.

Qn should we evaluate p_1

$$\frac{\sigma_0 \mid p \Downarrow \sigma_1 \mid 0}{\sigma_0 \mid p \,\&\, p_1 \Downarrow \sigma_1 \mid 0}$$

$$\frac{\sigma_0 \mid p \Downarrow \sigma_1 \mid 1 \qquad \sigma_1 \mid p_1 \Downarrow \sigma_2 \mid b}{\sigma_0 \mid p \,\&\, p_1 \Downarrow \sigma_2 \mid b}$$

RULES

Thu 27 Sep

$$\frac{e_1 \Downarrow n_1 \qquad e_2 \Downarrow n_2}{e_1 + e_2 \Downarrow n_1 + n_2}$$

the are different pluses

$$\frac{}{n \Downarrow n}$$

$$\frac{e_1 \Downarrow n_1 \qquad e_2 \Downarrow n_2}{e_1 - e_2 \Downarrow n_1 - n_2}$$

Ex

$$\cfrac{\cfrac{\cfrac{\cfrac{}{5 \Downarrow 5} \quad \cfrac{}{4 \Downarrow 4}}{5+4 \Downarrow 9} \qquad \cfrac{}{3 \Downarrow 3}}{(5+4)-3 \Downarrow 6} \qquad \cfrac{\cfrac{}{6 \Downarrow 6} \quad \cfrac{}{4 \Downarrow 4}}{6+4 \Downarrow 10}}{((5+4)-3) + (6+4) \Downarrow 16}$$

A SIMPLE FRAGMENT

$$\langle iexp \rangle ::= \langle num \rangle$$
$$\mid \langle iexp \rangle \; \langle iop \rangle \; \langle iexp \rangle$$

$$\langle iop \rangle ::= + \mid -$$

TOSAY WANT A PROGRAM DOES

(1) SAY WHAT THE OUTPUT VALUE IS   VALUES ARE INTEGERS

(2) HOW TO TURN A PROGRAM DEFINED BY EACH PRODUCTION RULE ... INTO A VALUE

USE JUDGEMENTS

$$e \Downarrow n$$

The program e evaluates to value n

YOU KNOW

SYNTAX of A PROGRAMMING LANGUAGE

&

PARSING / LEXING

WHAT YOU DONT KNOW

WHAT DOES A PROGRAM DO.

$\Rightarrow$ BIG STEP SEMANTICS.

Now we have judgement forms we can evaluate <u>all</u> the production rules.

In IMP, for iexp, there are 6 production rules .... so we need 6 rules

For <u>DO</u>

$$\frac{\sigma \mid c \Downarrow \sigma' \qquad \sigma' \mid e \Downarrow \sigma'' \mid n}{\sigma \mid \text{do } c \text{ return } e \Downarrow \sigma'' \mid n}$$

expecting new memory

integer

Also need judgement for evaluating <u>commands</u>

$$\sigma \mid c \Downarrow \sigma'$$

don't get anything else

<u>boolean expressions</u>

$$\sigma \mid p \Downarrow \sigma' \mid b$$

values for boolean expressions are bits

<u>blocks</u>

$$\sigma \mid ss \Downarrow \sigma'$$

* In IMP, we

  (1) more production rules

  (2) we need to deal with memory

  (3) we need to deal with scope

Memory

Consider

$$\text{do } x := x+1 \text{ return } x$$

Need memory so judgement form becomes

$$\sigma \mid e \Downarrow \sigma' / n$$

expression e in memory σ | evaluates to | integer n in memory σ'

x3	1
3x	2
x,4	3
->	1
- >	2
37	1
x4	1

lexing

- chopping text into tokens

our rules

- these are tokens by themselves
  ( ) [ ] { } , ;
- a digit followed by as many more digits as are present makes a numeric token
- an alphabetical char followed by as many more alphanumeric chars as possible is an identifier token
- all other ~~tokens~~ chars with no space between form one token, e.g. ->

```
<command>
  ::= {<block>}
    | <var> := <iexp>
    | if (<bexp>) <command> else <command>
    | while (<bexp>) <command>
    | new <var> := <iexp> in <command>
    | <var>(<arguments>)

<block>
  ::=
    | <command>; <block>


<arguments>
  ::=
    | <iexp> <commaargs>

<commaargs>
  ::=
    | , <iexp> <commaargs>

<iexp>
  ::= <number>
    | <var>
    | <iexp> <iop> <iexp>
    | new <var> := <iexp> in <iexp>
    | do <command> return <iexp>
    | <var>(<arguments>)

<iop>
  ::= +
    | -

<bexp>
  ::= <bit>
    | <bexp> & <bexp>
    | <bexp> \| <bexp>
    | ! <bexp>
    | <iexp> <comparator> <iexp>

<comparator>
  ::= ==
    | !=
    | <
    | >
    | <=
```

→ lithium

helium

go condition by Thu

repo

lexing – the local rules

left recursion

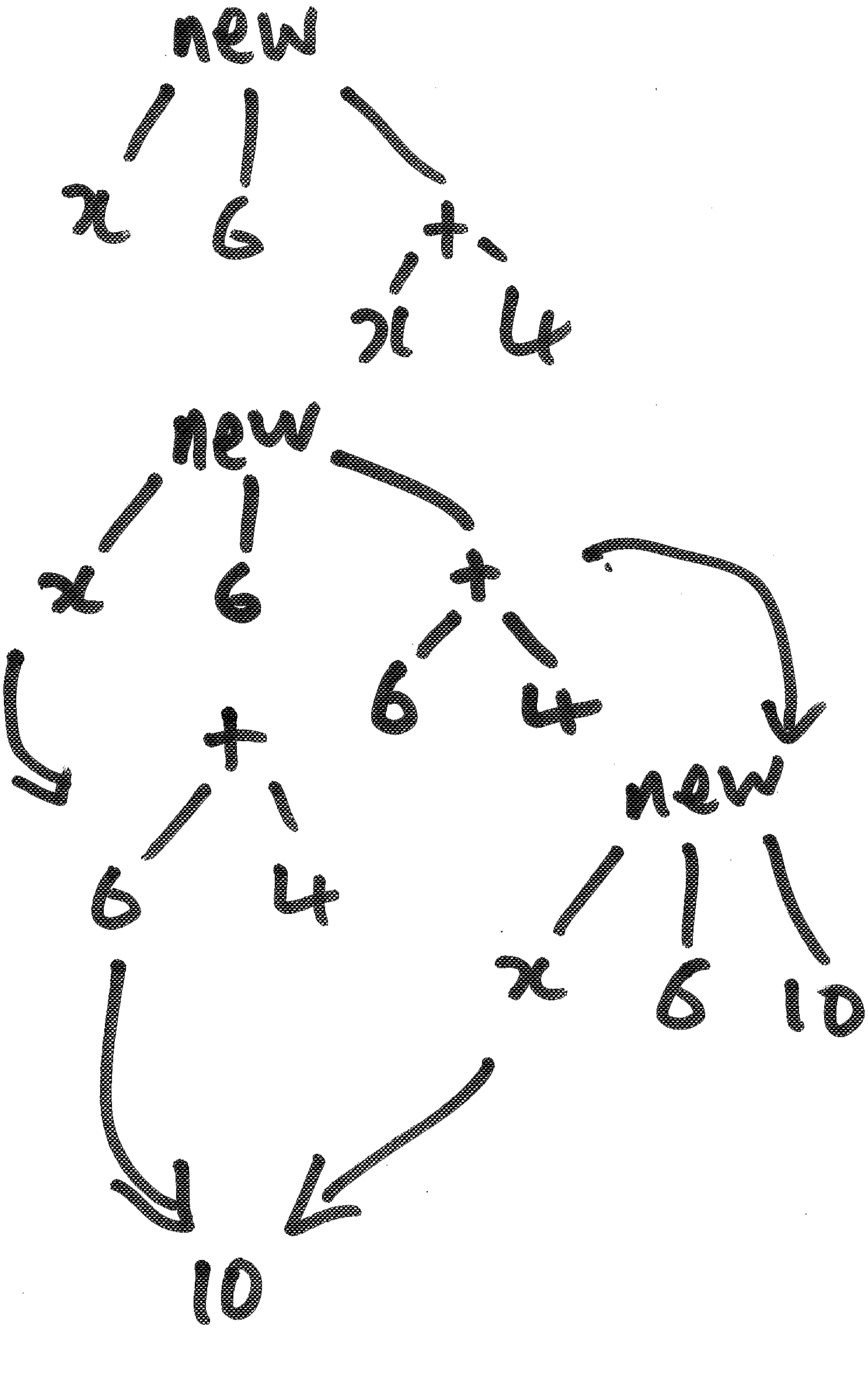
(Hutton's razor with –)

resolving ambiguity – big & little

grammars versus types

getting from a grammar
to a parser can be tricky!

new $x := 6$ in $x + 4$



$$\langle n \rangle ::= \langle \text{number} \rangle$$
$$\mid \langle n \rangle - \langle n \rangle$$

3 - 2 - 1

4 - 3 - 2 - 1