~announce past pape~

~mth sheet~

Call-by-name Vs. Call-by-value

Throughout the course, we have been discussing evaluation order and how our dynamics have strictly controlled it. I've also attluded to different options, promising to get to them at the end of the course. That time has come.

So far in the course, the evaluation strategy that we have used is

call-by-name (CBN)
= where variables represent terms
(as opposed to values as we will come on to)

We can see this evaluation strategy in the substitution lemma, and the dynamics of constructs that involve substitution

~open beq lemmata + STLC reference pointing out the unconstrained nature of both and how that means terms get represented by variables~

Example:

$$f(x) = x + x$$

$f(1+1)$
$\rightarrow (1+1) + (1+1)$
$\rightarrow 2 + (1+1)$
$\rightarrow 2 + 2$
$\rightarrow 4$

Notice how (1+1) gets subbed in whole-sale as a term.

There is another approach we could of taken, which for this example would reduce the number of evaluation steps.

Introducing

call-by-value
= where variables represent values

i.e. arguments to functions (or any terms that get bound) must be reduced to values before they get substituted in

Example:

$f(1+1)$
$\rightarrow f(2)$
$\rightarrow 2+2$
$\rightarrow 4$

3 eval steps vs. 4.

A key thing to highlight is that

Evaluation order does not effect pure results.

Note that our example evaluated to 4 in both cases, the only difference is the number of steps to get there.

But assuming that when we run programs we cannot noble the difference between eval steps

The difference is only noticable in the presence of effects

As we will come onto shortly, but first we need a CBV language to play with

You've actually met loads of CBV languages in your degree, because almost all languages with effects are CBV

CBV languages:
- C
- Java
- Scala
- JS
- OCaml
- Scheme
- ...

CBN is very strongly related to lazy eval of purely functional languages

Haskell = call-by-need
↑
optimised variant of CBN.

Call-by-value λ-calc

But as always in this course, we will keep things simple. We will use a CBV variant of STLC as our CBV lang.

To achieve this, we want to adjust our specification of STLC so that any time we perform a substitution, we only substitute in a value for a variable.

Q Do we need to change the statics?

No, because evaluation order is a matter for the dynamics: it is how the program runs.

Figure 1: Statics of the simply-typed λ-calculus (with numbers)

$$\textsc{Var}\ \frac{}{\Gamma, x : \sigma \vdash x : \sigma} \qquad \textsc{Num}\ \frac{n \in \mathbb{N}}{\Gamma \vdash \mathsf{num}[n] : \mathsf{Num}} \qquad \textsc{Plus}\ \frac{\Gamma \vdash e_1 : \mathsf{Num} \qquad \Gamma \vdash e_2 : \mathsf{Num}}{\Gamma \vdash \mathsf{plus}(e_1; e_2) : \mathsf{Num}}$$

$$\textsc{Times}\ \frac{\Gamma \vdash e_1 : \mathsf{Num} \qquad \Gamma \vdash e_2 : \mathsf{Num}}{\Gamma \vdash \mathsf{times}(e_1; e_2) : \mathsf{Num}} \qquad \textsc{Let}\ \frac{\Gamma \vdash e_1 : \sigma_1 \qquad \Gamma, x : \sigma_1 \vdash e_2 : \sigma_2}{\Gamma \vdash \mathsf{let}(e_1; x.\, e_2) : \sigma_2} \qquad \textsc{Unit}\ \frac{}{\Gamma \vdash \langle\rangle : \mathbf{1}}$$

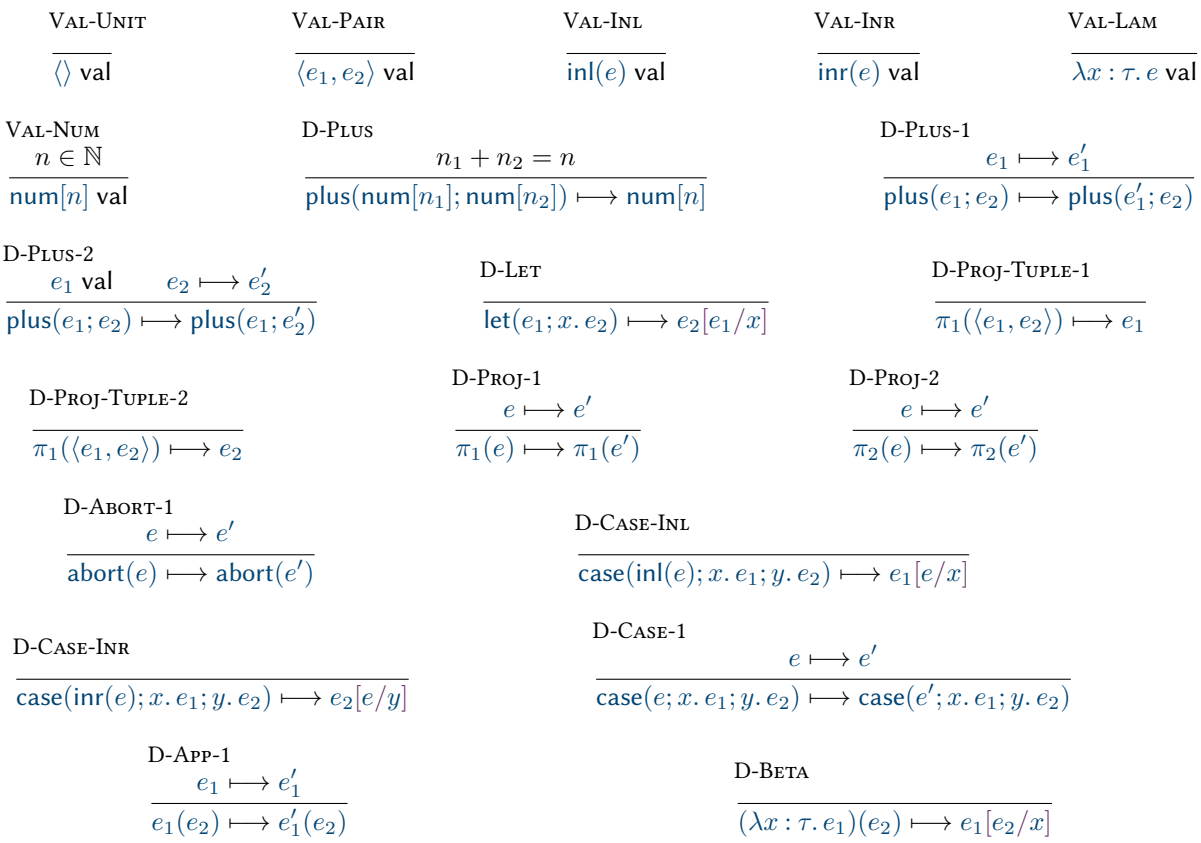
$$\textsc{Prod}\ \frac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash \langle e_1, e_2 \rangle : \tau_1 \times \tau_2} \qquad \textsc{Proj-1}\ \frac{\Gamma \vdash e : \tau_1 \times \tau_2}{\Gamma \vdash \pi_1(e) : \tau_1} \qquad \textsc{Proj-2}\ \frac{\Gamma \vdash e : \tau_1 \times \tau_2}{\Gamma \vdash \pi_2(e) : \tau_2} \qquad \textsc{Abort}\ \frac{\Gamma \vdash e : \mathbf{0}}{\Gamma \vdash \mathsf{abort}(e) : \tau}$$

$$\textsc{Inl}\ \frac{\Gamma \vdash e : \tau_1}{\Gamma \vdash \mathsf{inl}(e) : \tau_1 + \tau_2} \qquad \textsc{Inr}\ \frac{\Gamma \vdash e : \tau_2}{\Gamma \vdash \mathsf{inr}(e) : \tau_1 + \tau_2}$$

$$\textsc{Case}\ \frac{\Gamma \vdash e : \tau_1 + \tau_2 \qquad \Gamma, x : \tau_1 \vdash e_1 : \tau \qquad \Gamma, y : \tau_2 \vdash e_2 : \tau}{\Gamma \vdash \mathsf{case}(e; x.\, e_1; y.\, e_2) : \tau} \qquad \textsc{Lam}\ \frac{\Gamma, x : \sigma \vdash e : \tau}{\Gamma \vdash \lambda x : \sigma.\, e : \sigma \to \tau}$$

$$\textsc{App}\ \frac{\Gamma \vdash e_1 : \sigma \to \tau \qquad \Gamma \vdash e_2 : \sigma}{\Gamma \vdash e_1(e_2) : \tau}$$

Figure 2: Dynamics of the simply-typed λ-calculus



Q what about the dynamics?

- val judgements with args need updated
- rules that perform subst need updated

Figure 2: Dynamics of the simply-typed λ-calculus

$$\textsc{Val-Unit}\ \frac{}{\langle\rangle\ \mathsf{val}} \qquad \textsc{Val-Pair}\ \frac{e_1\ \mathsf{val} \quad e_2\ \mathsf{val}}{\langle e_1, e_2\rangle\ \mathsf{val}} \qquad \textsc{Val-Inl}\ \frac{e\ \mathsf{val}}{\mathsf{inl}(e)\ \mathsf{val}} \qquad \textsc{Val-Inr}\ \frac{e\ \mathsf{val}}{\mathsf{inr}(e)\ \mathsf{val}} \qquad \textsc{Val-Lam}\ \frac{}{\lambda x:\tau.\, e\ \mathsf{val}}$$

$$\textsc{Val-Num}\ \frac{n \in \mathbb{N}}{\mathsf{num}[n]\ \mathsf{val}} \qquad \textsc{D-Plus}\ \frac{n_1 + n_2 = n}{\mathsf{plus}(\mathsf{num}[n_1]; \mathsf{num}[n_2]) \longmapsto \mathsf{num}[n]} \qquad \textsc{D-Plus-1}\ \frac{e_1 \longmapsto e_1'}{\mathsf{plus}(e_1; e_2) \longmapsto \mathsf{plus}(e_1'; e_2)}$$

$$\textsc{D-Plus-2}\ \frac{e_1\ \mathsf{val} \qquad e_2 \longmapsto e_2'}{\mathsf{plus}(e_1; e_2) \longmapsto \mathsf{plus}(e_1; e_2')} \qquad \textsc{D-Let}\ \frac{e_1\ \mathsf{val}}{\mathsf{let}(e_1; x.\, e_2) \longmapsto e_2[e_1/x]} \qquad \textsc{D-Proj-Tuple-1}\ \frac{}{\pi_1(\langle e_1, e_2\rangle) \longmapsto e_1}$$

$$\textsc{D-Proj-Tuple-2}\ \frac{}{\pi_1(\langle e_1, e_2\rangle) \longmapsto e_2} \qquad \textsc{D-Proj-1}\ \frac{e \longmapsto e'}{\pi_1(e) \longmapsto \pi_1(e')} \qquad \textsc{D-Proj-2}\ \frac{e \longmapsto e'}{\pi_2(e) \longmapsto \pi_2(e')}$$

$$\textsc{D-Abort-1}\ \frac{e \longmapsto e'}{\mathsf{abort}(e) \longmapsto \mathsf{abort}(e')} \qquad \textsc{D-Case-Inl}\ \frac{e\ \mathsf{val}}{\mathsf{case}(\mathsf{inl}(e); x.\, e_1; y.\, e_2) \longmapsto e_1[e/x]}$$

$$\textsc{D-Case-Inr}\ \frac{e\ \mathsf{val}}{\mathsf{case}(\mathsf{inr}(e); x.\, e_1; y.\, e_2) \longmapsto e_2[e/y]} \qquad \textsc{D-Case-1}\ \frac{e \longmapsto e'}{\mathsf{case}(e; x.\, e_1; y.\, e_2) \longmapsto \mathsf{case}(e'; x.\, e_1; y.\, e_2)}$$

$$\textsc{D-App-1}\ \frac{e_1 \longmapsto e_1'}{e_1(e_2) \longmapsto e_1'(e_2)} \qquad \textsc{D-Beta}\ \frac{e_2\ \mathsf{val}}{(\lambda x:\tau.\, e_1)(e_2) \longmapsto e_1[e_2/x]}$$

Additional rules:
(so we can fill v val constraints)

$$\textsc{D-Inl}\ \frac{e \mapsto e'}{\mathsf{inl}(e) \mapsto \mathsf{inl}(e')} \qquad \textsc{D-Inr}\ \frac{e \mapsto e'}{\mathsf{inr}(e) \mapsto \mathsf{inr}(e')}$$

$$\textsc{D-App-2}\ \frac{e_1\ \mathsf{val} \qquad e_2 \mapsto e_2'}{e_1(e_2) \mapsto e_1(e_2')}$$

$$\textsc{D-Let-1}\ \frac{e_1 \mapsto e_1'}{\mathsf{let}(e_1; x.\, e_2) \mapsto \mathsf{let}(e_1'; x.\, e_2)}$$

Call-by-value λ-calc-cont

Progress + Preservation hold for CBV-STLC.

(the sheet will explore this)

Effects

Like I said earlier, the difference between CBV and CBN is most apparent in the presence of effects

So lets introduce a simple printing effect to explore this

Statics:

$$\text{PRINT}\ \frac{s \in \Sigma^* \quad \Gamma \vdash e : \tau}{\Gamma \vdash \text{print}(s; e) : \tau}$$

If s is a string in our alphabet Σ, and the continuation $e : \tau$, then printing s and continuing with e has type τ

Dynamics:

To signify the side effect of printing happening, we will adjust our dynamic relation as follows:

$$e \xmapsto{s} e'$$

e prints s and steps to e'

$$\text{D-P-PRINT}\ \frac{}{\text{print}(s,e) \xmapsto{s} e}$$

If we have a print statement, it always prints s, and continues with e

We also need to adjust the rest of the dynamics to thread around, using ε to signify the empty string

e.g. "leaf" rules that would be at the top of the $\xmapsto{s}$ derivation need to explicitly print nothing

$$\text{D-P-BETA}\ \frac{v\ \text{val}}{(\lambda x{:}\tau.e)(v) \xmapsto{\varepsilon} e[v/x]}$$

only for CBV

and "node" rules need to propagate strings around (be them ε from D-P-BETA or s from D-P-PRINT)

$$\text{D-P-APP-1}\ \frac{e_1 \xmapsto{s} e_1'}{e_1(e_2) \xmapsto{s} e_1'(e_2)}$$

Examples with Effects

Now we have effects, lets see the difference on the term

$\vdash (\lambda x{:}Num.plus(x,x))(print(hi;num[1])) : Num$

~Do interactively~

CBN:

$(\lambda x{:}Num.plus(x,x))(print(hi;num[1]))$

$\mapsto_n$ (n for name, sub arg right in)

$plus(print(hi;num[1]);print(hi;num[1]))$

hi $\mapsto_n$ eval fst arg + print

$plus(num[1];print(hi;num[1]))$

hi $\mapsto_n$ eval snd arg + print

$plus(num[1];num[1])$

$\mapsto_n$ eval plus

$num[2]$

4 eval steps
2 "hi" prints

Q What do we expect the difference to be for CBV?

3 eval steps
1 "hi" print

$\lambda x{:}Num.plus(x;x)(print(hi;num[1]))$

hi $\mapsto_v$ must reduce arg first, causing first + only print

$\lambda x{:}Num.plus(x;x)(num[1])$

$\mapsto_v$ now we can sub.

$plus(num[1];num[1])$

$\mapsto_v$ eval plus + done.

$num[2]$

CBV vs CBN
- same output
- different eval steps
- different observationally in presence of effects.

~ remind + enthuse about guest lectures ~