CPSC 311: Definition of Programming Languages: Review

16-review

DRAFT

Joshua Dunfield
University of British Columbia

October 28, 2015

1 Fundamental advice

- Algorithm to find lost glasses: Step 1: are you wearing them?

  Exam version: *Always* check whether an answer is already in front of you.

  Values version: What does a value evaluate to? Itself.

- In case of “I forgot Racket syntax” panic:
  1. Look on the same page. The syntax you’re trying to remember may be used in the given code already.
  2. Look on other pages.
  3. Still can’t find it? Make something up, but write a few words explaining what it does.

- In case of “is this derivation tree *precise enough*?” panic:
  1. Using concrete syntax instead of abstract syntax, or mixing them, is okay. (I don’t *encourage* this, but you won’t lose marks unless we really can’t figure out what you were trying to do.)
  2. Going outside the box is okay.
  3. Writing the whole derivation tree on a worksheet and saying “see top of worksheet (ii)” is totally okay.

2 Showing work

Exam instructions often direct you to "show your work" by writing intermediate steps of a computation. In 311, however:

- For a derivation, the derivation tree *is* the work.
- For a coding question, you're not required to show work.

If you are unsure of your answer, and the answer itself turns out to be wrong, additional explanations may help us give partial credit. But be as clear and concise as you can.

I have seen students find a good answer—after ample intermediate work on a worksheet—but then they become confused and manage to get a wrong answer instead. Occasionally, however, a student who *tried* to erase or cross out correct intermediate work failed to cross it out completely, and I gave partial credit based on the intermediate work they tried to hide, because that work demonstrated understanding of the material.

Simply writing a bunch of answers (without intermediate work) and crossing most of them out, however, will generally not lead to correct marks. (Tell the story of the two envelopes?)

3 Substitution

You can answer the substitution question on the exam by mechanically following the definition. It probably helps to keep the following in mind:

- An inner binding *shadows* an outer binding. If we're evaluating

  $$\big(\mathsf{with}\ x\ (\mathsf{num}\ 3)\ (\mathsf{with}\ x\ (\mathsf{num}\ 4)\ (\mathsf{id}\ x))\big)$$

  then we'll substitute $(\mathsf{num}\ 3)$ for x in the body of the outer with. Following the definition of *subst*, however, since the expression we are substituting into—$(\mathsf{with}\ x\ (\mathsf{num}\ 4)\ (\mathsf{id}\ x))$—is binding the same symbol we are substituting, we leave the body $(\mathsf{id}\ x)$ alone: it refers to the nearest enclosing binder, which is the inner x.

  $$\begin{aligned} & \mathit{subst}\big((\mathsf{with}\ x\ (\mathsf{num}\ 4)\ (\mathsf{id}\ x)),\ x,\ (\mathsf{num}\ 3)\big) \\ = \ & (\mathsf{with}\ x\ (\mathsf{num}\ 4)\ (\mathsf{id}\ x)) \end{aligned}$$

  If, however, the inner binding were of a different symbol, say z, then

  $$\begin{aligned} & \mathit{subst}\big((\mathsf{with}\ z\ (\mathsf{num}\ 4)\ (\mathsf{id}\ x)),\ x,\ (\mathsf{num}\ 3)\big) \\ = \ & (\mathsf{with}\ z\ (\mathsf{num}\ 4)\ \mathit{subst}((\mathsf{id}\ x), x, (\mathsf{num}\ 3))) \\ = \ & (\mathsf{with}\ z\ (\mathsf{num}\ 4)\ (\mathsf{num}\ 3)) \end{aligned}$$

4 Derivations

Method of hope: Work upwards from the (partial) conclusion you want. Match up meta-variables in the rule with the actual judgment you're trying to derive, then replace in premises.

Examination advice:

- Consider using pencil for derivation trees.
- A question might ask you to derive something that's not derivable. Start writing the derivation tree anyway. If you become convinced you can't complete it, do *not* erase it all; circle the premise you can't derive and (briefly) explain why no derivation exists. **See the next page for an example.**

not a lam — not derivable

$$\dfrac{\dfrac{}{(\text{lam } x\ (\text{id } x)) \Downarrow \circlearrowleft}\,\text{Eval-lam} \qquad \dfrac{\dfrac{}{(\text{num } 5) \Downarrow \circlearrowleft}\,\underline{\text{Eval-num}} \qquad \dfrac{}{(\text{num } 2) \Downarrow \circlearrowleft}\,\text{Eval-num}}{(\text{app } (\text{num } 5)\ (\text{num } 2)) \Downarrow}\,\text{Eval-app-value}}{(\underline{\text{app}}\ (\text{lam } x\ (\text{id } x))\ \underline{(\text{app } (\text{num } 5)(\text{num } 2))}) \Downarrow}\,\text{Eval-app-value}$$

4.1 Evaluation

$\boxed{e \Downarrow v}$ Expression e evaluates to value v

$$\frac{}{(\mathsf{num}\ n) \Downarrow (\mathsf{num}\ n)}\ \text{Eval-num} \qquad \frac{e1 \Downarrow v1 \qquad \mathit{subst}(e2, x, v1) \Downarrow v2}{(\mathsf{with}\ x\ e1\ e2) \Downarrow v2}\ \text{Eval-with}$$

$$\frac{}{(\mathsf{lam}\ x\ A\ e1) \Downarrow (\mathsf{lam}\ x\ A\ e1)}\ \text{Eval-lam}$$

$$\frac{e1 \Downarrow (\mathsf{lam}\ x\ A\ eB) \qquad e2 \Downarrow v2 \qquad \mathit{subst}(eB, x, v2) \Downarrow v}{(\mathsf{app}\ e1\ e2) \Downarrow v}\ \text{Eval-app-value}$$

$$\frac{e1 \Downarrow v1 \qquad e2 \Downarrow v2 \qquad v1\ \mathsf{op}\ v2\ =\ v}{(\mathsf{binop}\ \mathsf{op}\ e1\ e2) \Downarrow v}\ \text{Eval-binop}$$

$$\frac{e1 \Downarrow v1 \qquad e2 \Downarrow v2}{(\mathsf{pair}\ e1\ e2) \Downarrow (\mathsf{pair}\ v1\ v2)}\ \text{Eval-pair} \qquad \frac{ePair \Downarrow (\mathsf{pair}\ v1\ v2) \qquad \mathit{subst}(\mathit{subst}(eB, x1, v1), x2, v2) \Downarrow v}{(\mathsf{pair\text{-}case}\ \ ePair\ \ x1\ \ x2\ \ eB) \Downarrow v}\ \text{Eval-pair-case}$$

$$\frac{}{(\mathsf{btrue}) \Downarrow (\mathsf{btrue})}\ \text{Eval-btrue} \qquad \frac{}{(\mathsf{bfalse}) \Downarrow (\mathsf{bfalse})}\ \text{Eval-bfalse}$$

$$\frac{e \Downarrow (\mathsf{btrue}) \qquad eThen \Downarrow v}{(\mathsf{ite}\ e\ eThen\ eElse) \Downarrow v}\ \text{Eval-ite-true} \qquad \frac{e \Downarrow (\mathsf{bfalse}) \qquad eElse \Downarrow v}{(\mathsf{ite}\ e\ eThen\ eElse) \Downarrow v}\ \text{Eval-ite-false}$$

$$\frac{\mathit{subst}(e,\ u,\ (\mathsf{rec}\ u\ B\ e)) \Downarrow v}{(\mathsf{rec}\ u\ B\ e) \Downarrow v}\ \text{Eval-rec}$$

4.1.1 Evaluation: rec, app, ...

$$\frac{}{\Big(\mathsf{app}\ \ (\mathsf{rec}\ u\ (\mathsf{lam}\ x\ eB))\ \ (\mathsf{num}\ 2)\Big) \Downarrow}$$

4.1.2 Evaluation: pair-case

$$\frac{ePair \Downarrow (\text{pair } v1\ v2) \qquad \mathit{subst}(\mathit{subst}(eB, x1, v1), x2, v2) \Downarrow v}{(\text{pair-case } \ ePair \ \ x1 \ \ x2 \ \ eB) \Downarrow v}\ \text{Eval-pair-case}$$

$$\frac{\big(\text{app } (\text{lam } x\ (\text{pair } (\text{id } x)\ (\text{id } x)))\ (\text{num } 3)\big) \Downarrow \qquad\qquad \mathit{subst}(\mathit{subst}((\text{id } x2), x1, \qquad), x2, \qquad) \Downarrow}{\Big(\text{pair-case } \ \big(\text{app } (\text{lam } x\ (\text{pair } (\text{id } x)\ (\text{id } x)))\ (\text{num } 3)\big) \ \ x1 \ \ x2 \ \ (\text{id } x2)\Big) \Downarrow}$$

4.2 Some typing rules

$\boxed{\Gamma \vdash e : A}$ Under assumptions Γ, expression e has type A

$$\frac{(x : A) \in \Gamma}{\Gamma \vdash (\mathsf{id}\ x) : A}\ \text{Type-var}$$

$$\frac{}{\Gamma \vdash (\mathsf{num}\ n) : \mathsf{Num}}\ \text{Type-num} \qquad \frac{op : A1 * A2 \to B \qquad \Gamma \vdash e1 : A1 \qquad \Gamma \vdash e2 : A2}{\Gamma \vdash (\mathsf{binop}\ op\ e1\ e2) : B}\ \text{Type-binop}$$

$$\frac{}{\Gamma \vdash (\mathsf{bfalse}) : \mathsf{Bool}}\ \text{Type-false} \qquad \frac{}{\Gamma \vdash (\mathsf{btrue}) : \mathsf{Bool}}\ \text{Type-true}$$

$$\frac{\Gamma \vdash e : \mathsf{Bool} \qquad \Gamma \vdash eThen : A \qquad \Gamma \vdash eElse : A}{\Gamma \vdash (\mathsf{ite}\ e\ eThen\ eElse) : A}\ \text{Type-ite}$$

$$\frac{x : A, \Gamma \vdash eBody : B}{\Gamma \vdash (\mathsf{lam}\ x\ A\ eBody) : A \to B}\ \text{Type-lam} \qquad \frac{\Gamma \vdash e1 : A \to B \qquad \Gamma \vdash e2 : A}{\Gamma \vdash (\mathsf{app}\ e1\ e2) : B}\ \text{Type-app}$$

$$\frac{\Gamma \vdash e1 : A1 \qquad \Gamma \vdash e2 : A2}{\Gamma \vdash (\mathsf{pair}\ e1\ e2) : A1 * A2}\ \text{Type-pair} \qquad \frac{\Gamma \vdash e : A1 * A2 \qquad x1 : A1, x2 : A2, \Gamma \vdash eBody : B}{\Gamma \vdash (\mathsf{pair\text{-}case}\ e\ x1\ x2\ eBody) : B}\ \text{Type-pair-case}$$

$$\frac{\Gamma \vdash e : A \qquad x : A, \Gamma \vdash eBody : B}{\Gamma \vdash (\mathsf{with}\ x\ e\ eBody) : B}\ \text{Type-with} \qquad \frac{u : B, \Gamma \vdash e : B}{\Gamma \vdash (\mathsf{rec}\ u\ B\ e) : B}\ \text{Type-rec}$$

$$\frac{}{\Gamma \vdash (\mathsf{list\text{-}empty}\ A) : \mathsf{List}\ A}\ \text{Type-empty} \qquad \frac{\Gamma \vdash e1 : A \qquad \Gamma \vdash e2 : \mathsf{List}\ A}{\Gamma \vdash (\mathsf{list\text{-}cons}\ e1\ e2) : \mathsf{List}\ A}\ \text{Type-cons}$$

$$\frac{\Gamma \vdash e : \mathsf{List}\ A \qquad \Gamma \vdash eEmpty : B \qquad xh : A, xt : \mathsf{List}\ A, \Gamma \vdash eCons : B}{\Gamma \vdash (\mathsf{list\text{-}case}\ e\ eEmpty\ xh\ xt\ eCons) : B}\ \text{Type-list-case}$$

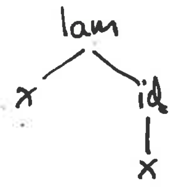
4.2.1 Typing: application

$$\frac{\dfrac{(f : \mathsf{Num} \to \mathsf{Bool}) \in \Gamma)}{\Gamma \vdash (\mathsf{id}\ f) : \mathsf{Num} \to \mathsf{Bool}}\ \text{Type-var} \qquad \dfrac{(y : \mathsf{Num}) \in \Gamma)}{\Gamma \vdash (\mathsf{id}\ y) : \mathsf{Num}}\ \text{Type-var}}{\underbrace{y : \mathsf{Num}, f : \mathsf{Num} \to \mathsf{Bool}, \emptyset}_{\Gamma} \vdash \big(\mathsf{app}\ (\mathsf{id}\ f)\ (\mathsf{id}\ y)\big) : \mathsf{Bool}}\ \text{Type-app}$$

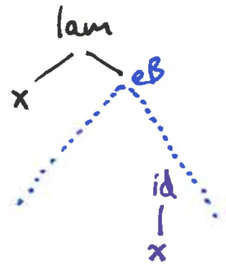
4.2.2 Typing: ite

4.2.3 Typing: rec

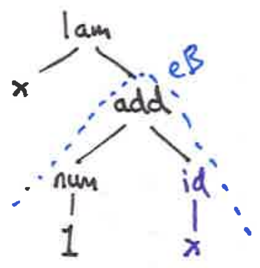
(lam x (id x))



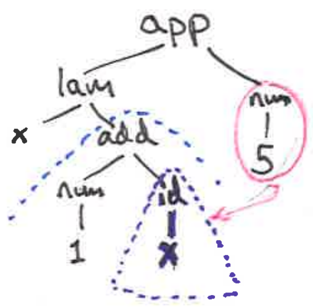
(lam x eB)

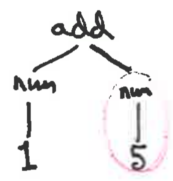


(lam x (add (num 1)
(id x)))

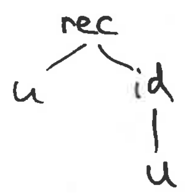


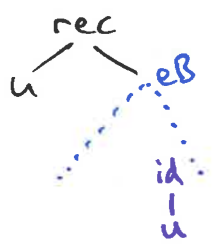
(app (lam x (add (num 1)
(id x)))
(num 5))





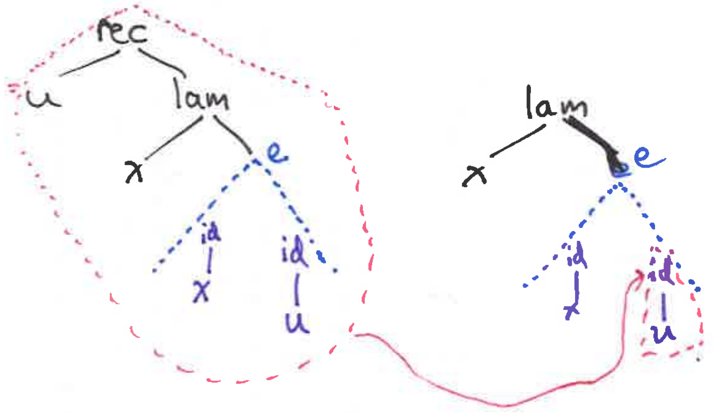
(rec u (id u))



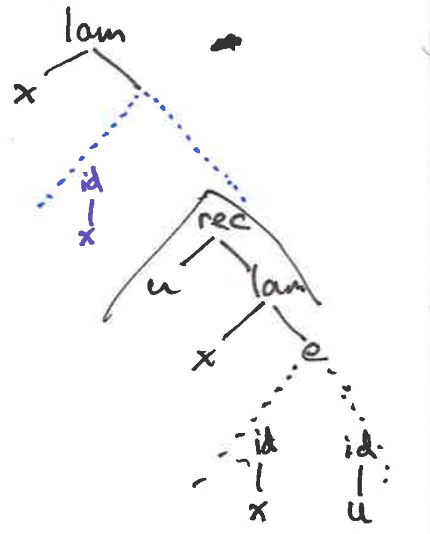


(rec u (lam x e))

(lam x e)



(lam x (subst(e, u, (rec u
(lam x e)))



$$\dfrac{\dfrac{}{\Gamma \vdash (\text{id } x) : \text{List Bool}}\,\text{Type-var} \qquad \dfrac{}{\Gamma \vdash (\text{num } 0) : \text{Num}}\,\text{Type-num} \qquad xh:\text{Bool}, xt:\text{List Bool}, \Gamma \vdash}{\underbrace{x : \text{List Bool}\, \emptyset}_{\Gamma} \vdash (\text{list-case } \underbrace{(\text{id } x)}_{e}\ \underbrace{(\text{num } 0)}_{e\text{Empty}}\ xh\ xt\ \underbrace{(\text{ite } \underset{(\text{id}}{} xh)\ (\text{num } 1)\ (\text{num } 2))}_{e\text{Cons}})) :}\,\text{Type-list-case}$$

$$\dfrac{\dfrac{}{\Gamma_2 \vdash (\text{id } xh) : \text{Bool}}\,\text{Type-var} \qquad \dfrac{}{\Gamma_2 \vdash (\text{num } 1) : \text{Num}} \qquad \dfrac{}{\Gamma_2 \vdash (\text{num } 2) : \text{Nu}}}{\underbrace{xh : \text{Bool}, xt : \text{List Bool}, \Gamma}_{\Gamma_2} \vdash (\text{ite } (\text{id } xh)\ (\text{num } 1)\ (\text{num } 2))}$$

$$x : \text{Bool}\underline{, \emptyset} \vdash$$

$$\dfrac{\dfrac{}{\emptyset \vdash \text{btrue} : \text{Bool}}\,\text{Type-true} \qquad \dfrac{(x : \text{Bool}) \in x : \text{Bool}}{x : \text{Bool} \vdash (\text{id } x) : \text{Bool}}\,\text{Type-var}}{\underbrace{\emptyset}_{\Gamma_1} \vdash (\text{with } x\ \underbrace{\text{btrue}}_{e}\ \underbrace{(\text{id } x)}_{e\text{Body}}) : \text{Bool}}\,\text{Type-with}$$

5 Syntactic sugar

6 Expression strategy vs. value strategy

`value-expr.rkt`

7 THE END IS NIGH

Some of the (higher-protein?) topics we've covered so far:

1. Syntax, BNFs, parsing, syntactic sugar
2. Identifiers, scope, substitution
3. Judgments and derivations
4. Evaluation (big-step) semantics
    - functions
    - recursion
    - pairs, lists
    - strings
5. Error handling and small-step semantics
6. Typing
7. Type safety

Some topics that we will **definitely** cover in the remaining ≈ 5 weeks:

1. Polymorphism (this Friday)
2. Environment-based evaluation
3. Mutable state
4. Subtyping
    - probably OO-style subtyping

I wish we could cover all of the following, but it's very, very unlikely:

4. Subtyping
    - . . . but "functional" (structural) subtyping, in addition to OO-style
5. Type inference
6. Bidirectional type checking
7. Refinement types
8. Contracts and/or gradual typing
9. Lazy evaluation
10. Type-directed translation
11. Continuations
12. Curry–Howard correspondence