CPSC 311: Definition of Programming Languages: Assignment 5

Joshua Dunfield
University of British Columbia

November 20, 2016

1 Logistics

You may work in teams of **up to 2** on this assignment. (You *can* work individually. But we recommend that you collaborate, mostly for your benefit, but—I'll be honest—also for ours: there are many students taking 311, but not very many TAs, and working individually means one more assignment to mark. **Repeating this reminder for a5. Also, you can submit as a team even if you complete the assignment separately and meet only to decide on a combined solution. We don't particularly recommend that, but it's still one less assignment to mark, and you'll almost certainly learn something from the other person's solution.**)

You must include a README.txt file based on this template:

`http://www.ugrad.cs.ubc.ca/~cs311/2016W1/assignments/support/README.txt`

For your final submission, be sure you have replaced **all** of the "`TODO`"s in `README.txt`.

handin has not yet been set up for this assignment. When handin has been set up, we will make an announcement on Piazza.

Download `http://www.ugrad.cs.ubc.ca/~cs311/2016W1/assignments/a5.rkt`

1.1 Important! (new in a3, still in a5)

If your code is rejected by "Check Syntax", or the `handin` script can't run your code, you will receive a mark of 0 **for the entire coding portion of the assignment**.

So, if you get stuck on one problem, **comment out the code that doesn't work**, and try to explain in a comment what you were trying to do. **Make sure that your final handin, at minimum, prints test results**—even if all the tests fail!

*Problems **3** and **4** must be handed in using `handin` by 01:00 (1:00am) on Friday, November 25, 2016.*

*Problems **1** and **2** must be in the box in X235 by 8:45 a.m. on **Friday**, November 25, 2016.*

1.2 Handin

You must turn in the coding part of the assignment using the **command-line version** of the handin program.

For handin, this assignment is called `a5`. Submit two files:

- `a5.rkt`
- `README.txt`

§1 Logistics

If you are working in a team, submit only one set of files. If you have both run handin for a4 already, have one person overwrite their handin with a directory containing only a file called `please-mark-aaaaa` where `aaaaa` is the CS ugrad username of your partner.

Just a reminder, late assignments are not accepted (except for the "grace period" of a few minutes, which you shouldn't rely on), and (basically) no excuses will be entertained. So, hand in your assignments early and often!

Avoid using DrRacket comment boxes, because handin is still afraid of them. Comments using "`;`" and "`#| ... |#`" are fine.

2 Syntax

2.1 Types

Concrete syntax

```
⟨Type⟩ ::= rat
         | int
         | pos
         | bool
         | {* ⟨Type⟩ ⟨Type⟩}
         | {-> ⟨Type⟩ ⟨Type⟩...}
         | {ref ⟨Type⟩}
         | {record ⟨FieldTypes⟩...}

⟨FieldTypes⟩ ::= {⟨id⟩ ⟨id⟩... ⟨Type⟩}
```

Abstract syntax

$$
\begin{array}{llcll}
\text{Types} & A, B & ::= & \mathsf{rat} & \\
 & & & \mid \mathsf{int} & \\
 & & & \mid \mathsf{pos} & \\
 & & & \mid \mathsf{bool} & \\
 & & & \mid A * B & \\
 & & & \mid A \rightarrow B & \\
 & & & \mid \mathsf{ref}\ A & \\
 & & & \mid \mathsf{record}\ \mathit{FldType}1, \cdots, \mathit{FldType}n & n \geq 0 \\
\text{FieldType} & \mathit{FldType} & ::= & f : A &
\end{array}
$$

The record type is new in this assignment. A record type has a list of zero or more occurrences of ⟨FieldTypes⟩. Each occurrence of ⟨FieldTypes⟩ declares the type of one or more fields (the ⟨id⟩s), each of type ⟨Type⟩.

For example, the following is concrete syntax for the type of a record with three fields: a positive integer `route-number`, and Booleans `electric` and `express`:

```
{record {route-number Pos} {electric express Bool}}
```

The corresponding abstract syntax is

$$(\mathsf{Record}\ \texttt{route-number} : \mathsf{pos}, \texttt{electric} : \mathsf{bool}, \texttt{express} : \mathsf{bool})$$

where `route-number` : pos is a $\mathit{FldType}$.

2.2 Expressions

Concrete syntax

```
⟨E⟩ ::=
        ⋮
      | {Record ⟨Field⟩...}
      | {Dot ⟨E⟩ ⟨id⟩}

⟨Fields⟩ ::= {⟨id⟩ ⟨E⟩}
```

Abstract syntax

$$
\begin{array}{llcl}
\text{Expressions} & e & ::= & \\
 & & & \vdots \\
 & & & \mid (\mathsf{Record}\ \mathit{Field}1, \ldots, \mathit{Field}n) \\
 & & & \mid (\mathsf{Dot}\ e\ \mathsf{f}) \\
\text{Fields} & \mathit{Field} & ::= & \mathsf{f}{=}e
\end{array}
$$

An example expression with the above record type is

```
{Record {express Bfalse} {route-number 14} {electric Btrue}}
```

If the identifier r is bound to the above expression, then

```
{Dot e route-number}
```

evaluates to 14 (in abstract syntax, (Num 14)).

3 Environment-based semantics

In addition to new features like records, rules from previous assignments have been updated to be environment-based, rather than substitution-based.

$\boxed{env; S1 \vdash e \Downarrow v; S2}$ Under environment env and initial store S1, expression e evaluates to value v with new store S2

$$\frac{}{env; S \vdash (\mathsf{Num}\ n) \Downarrow (\mathsf{Num}\ n); S}\ \text{SEnv-num} \qquad \frac{env; S \vdash e1 \Downarrow v1; S1 \quad env; S1 \vdash e2 \Downarrow v2; S2 \quad v1\ op\ v2 = v}{env; S \vdash (\mathsf{Binop}\ op\ e1\ e2) \Downarrow v; S2}\ \text{SEnv-binop}$$

$$\frac{}{env; S \vdash (\mathsf{Btrue}) \Downarrow (\mathsf{Btrue}); S}\ \text{SEnv-true} \qquad \frac{}{env; S \vdash (\mathsf{Bfalse}) \Downarrow (\mathsf{Bfalse}); S}\ \text{SEnv-false}$$

$$\frac{env; S \vdash e \Downarrow (\mathsf{Btrue}); S1 \quad env; S1 \vdash eThen \Downarrow v; S2}{env; S \vdash (\mathsf{Ite}\ e\ eThen\ eElse) \Downarrow v; S2}\ \text{SEnv-ite-true} \qquad \frac{env; S \vdash e \Downarrow (\mathsf{Bfalse}); S1 \quad env; S1 \vdash eElse \Downarrow v; S2}{env; S \vdash (\mathsf{Ite}\ e\ eThen\ eElse) \Downarrow v; S2}\ \text{SEnv-ite-false}$$

$$\frac{lookup(env, x) = e \quad env; S \vdash e \Downarrow v; S1}{env; S \vdash (\mathsf{Id}\ x) \Downarrow v; S1}\ \text{SEnv-id} \qquad \frac{env; S \vdash e1 \Downarrow v1; S1 \quad x{=}v1, env; S1 \vdash e2 \Downarrow v2; S2}{env; S \vdash (\mathsf{Let}\ x\ e1\ e2) \Downarrow v2; S2}\ \text{SEnv-let}$$

$$\frac{}{env; S \vdash (\mathsf{Lam}\ x\ eB) \Downarrow (\mathsf{Clo}\ env\ (\mathsf{Lam}\ x\ eB)); S}\ \text{SEnv-lam} \qquad \frac{env_{old}; S1 \vdash e \Downarrow v; S2}{env; S1 \vdash (\mathsf{Clo}\ env_{old}\ e) \Downarrow v; S2}\ \text{SEnv-clo}$$

$$\frac{env; S \vdash e1 \Downarrow (\mathsf{Clo}\ env_{old}\ (\mathsf{Lam}\ x\ eB)); S1 \quad env; S1 \vdash e2 \Downarrow v2; S2 \quad x{=}v2, env_{old} \vdash eB \Downarrow v; S2}{env; S \vdash (\mathsf{App}\ e1\ e2) \Downarrow v; S2}\ \text{SEnv-app-value}$$

$$\frac{\big(\mathsf{Env\text{-}Rec}\ \mathcal{E}\ (u{=}(\mathsf{Clo}\ \mathcal{E}\ eB), env)\big); S1 \vdash eB \Downarrow v; S2 \quad \mathcal{E} \notin env}{env; S1 \vdash (\mathsf{Rec}\ u\ eB) \Downarrow v; S2}\ \text{SEnv-rec}$$

$$\frac{}{env; S \vdash (\mathsf{Location}\ \ell) \Downarrow (\mathsf{Location}\ \ell); S}\ \text{SEnv-location} \qquad \frac{\ell \text{ fresh for } S1 \quad env; S \vdash e \Downarrow v; S1}{env; S \vdash (\mathsf{Ref}\ e) \Downarrow (\mathsf{Location}\ \ell); \ell \triangleright v,\ S1}\ \text{SEnv-ref}$$

$$\frac{env; S \vdash e \Downarrow (\mathsf{Location}\ \ell); S2 \quad \textit{lookup-loc}(S2, \ell) = v}{env; S \vdash (\mathsf{Deref}\ e) \Downarrow v; S2}\ \text{SEnv-deref}$$

$$\frac{env; S \vdash e1 \Downarrow (\mathsf{Location}\ \ell); S1 \quad env; S1 \vdash e2 \Downarrow v2; S2 \quad \textit{update-loc}(S2, \ell, v2) = S3}{env; S \vdash (\mathsf{Setref}\ e1\ e2) \Downarrow v2; S3}\ \text{SEnv-setref}$$

$$\frac{env; S1 \vdash e \Downarrow v; S2 \quad \emptyset \vdash v : A}{env; S1 \vdash (\mathsf{Downcast}\ A\ e) \Downarrow v; S2}\ \text{SEnv-downcast}$$

$$\frac{n \geq 0 \quad env; S \vdash e1 \Downarrow v1; S1 \quad \cdots \quad env; S(n-1) \vdash en \Downarrow vn; Sn}{env; S \vdash (\mathsf{Record}\ f1{=}e1, \ldots, fn{=}en) \Downarrow (\mathsf{Record}\ f1{=}v1, \ldots, fn{=}vn); Sn}\ \text{SEnv-record}$$

$$\frac{env; S1 \vdash e \Downarrow (\mathsf{Record}\ f1{=}v1, \ldots, fn{=}vn); S2 \quad (f{=}v) \in \{f1{=}e1, \ldots, fn{=}vn\}}{env; S1 \vdash (\mathsf{Dot}\ e\ f) \Downarrow v; S2}\ \text{SEnv-dot}$$

$$\frac{n \geq 1 \quad env; S \vdash e1 \Downarrow v1; S1 \quad \cdots \quad env; S(n-1) \vdash en \Downarrow vn; Sn}{env; S \vdash (\mathsf{Begin}\ e1\ \cdots\ en) \Downarrow vn; Sn}\ \text{SEnv-begin}$$

Figure 1 **Evaluation rules**

4 Subtyping

$\boxed{A <: B}$ Type A is a subtype of type B

$$\frac{}{A <: A}\ \text{Sub-refl} \qquad \frac{A1 <: A2 \qquad A2 <: A3}{A1 <: A3}\ \text{Sub-trans}$$

$$\frac{}{\mathsf{pos} <: \mathsf{int}}\ \text{Sub-pos-int} \qquad \frac{}{\mathsf{int} <: \mathsf{rat}}\ \text{Sub-int-rat}$$

$$\frac{A1 <: B1 \qquad A2 <: B2}{(A1 * A2) <: (B1 * B2)}\ \text{Sub-product}$$

$$\frac{B1 <: A1 \qquad A2 <: B2}{(A1 \rightarrow A2) <: (B1 \rightarrow B2)}\ \text{Sub-arr} \qquad \frac{A <: B \qquad B <: A}{(\mathsf{ref}\ A) <: (\mathsf{ref}\ B)}\ \text{Sub-ref}$$

Figure 2 Subtyping rules; for record subtyping, see Problem 4

5 Typing

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

$$\frac{\Gamma \vdash e : A \qquad A <: B}{\Gamma \vdash e : B}\ \text{Type-sub} \qquad \frac{\Gamma(x) = A}{\Gamma \vdash (\mathsf{Id}\ x) : A}\ \text{Type-var}$$

$$\frac{n \in \mathbb{Z} \qquad n \geq 0}{\Gamma \vdash (\mathsf{Num}\ n) : \mathsf{pos}}\ \text{Type-pos} \qquad \frac{n \in \mathbb{Z}}{\Gamma \vdash (\mathsf{Num}\ n) : \mathsf{int}}\ \text{Type-int} \qquad \frac{n \in \mathbb{Q}}{\Gamma \vdash (\mathsf{Num}\ n) : \mathsf{rat}}\ \text{Type-rat}$$

$$\frac{op : A1 * A2 \rightarrow B \qquad \Gamma \vdash e1 : A1 \qquad \Gamma \vdash e2 : A2}{\Gamma \vdash (\mathsf{Binop}\ op\ e1\ e2) : B}\ \text{Type-binop}$$

$$\frac{}{\Gamma \vdash (\mathsf{Bfalse}) : \mathsf{bool}}\ \text{Type-false} \qquad \frac{}{\Gamma \vdash (\mathsf{Btrue}) : \mathsf{bool}}\ \text{Type-true}$$

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

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

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

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

$$\frac{\Gamma \vdash e : A}{\Gamma \vdash (\mathsf{Ref}\ A\ e) : \mathsf{ref}\ A}\ \text{Type-ref} \qquad \frac{\Gamma \vdash e : \mathsf{ref}\ A}{\Gamma \vdash (\mathsf{Deref}\ e) : A}\ \text{Type-deref} \qquad \frac{\Gamma \vdash e1 : \mathsf{ref}\ A \qquad \Gamma \vdash e2 : A}{\Gamma \vdash (\mathsf{Setref}\ e1\ e2) : A}\ \text{Type-setref}$$

$$\frac{A <: B \qquad \Gamma \vdash e : B}{\Gamma \vdash (\mathsf{Downcast}\ A\ e) : A}\ \text{Type-downcast}$$

$$\frac{n \geq 0 \qquad \Gamma \vdash e1 : A1 \qquad \cdots \qquad \Gamma \vdash en : An}{\Gamma \vdash (\mathsf{Record}\ Fld1{=}e1, \ldots, Fldn{=}en)\ :\ (\mathsf{record}\ \ (f1 : A1), \ldots, (fn : An))}\ \text{Type-record}$$

$$\frac{\Gamma \vdash e : (\mathsf{record}\ \ FldType1, \ldots, FldTypen) \qquad (x : A) \in \{FldType1, \ldots, FldTypen\}}{\Gamma \vdash (\mathsf{Dot}\ e\ x) : A}\ \text{Type-dot}$$

$$\frac{n \geq 1 \qquad \Gamma \vdash e1 : A1 \qquad \cdots \qquad \Gamma \vdash en : An}{\Gamma \vdash (\mathsf{Begin}\ e1\ \cdots\ en) : An}\ \text{Type-begin}$$

$$\frac{\Gamma \vdash eG : \mathsf{bool} \qquad \Gamma \vdash eB : A}{\Gamma \vdash (\mathsf{While}\ eG\ eB) : \mathsf{bool}}\ \text{Type-while}$$

Figure 3 **Typing rules, including records**

6 Problems

Problems 1–2 are not coding problems. Turn them in on paper in the box in room X235. You can print this part of the assignment and write on it, or write out your answers separately. **Be sure to write your name, student number, and CS userid.** If you are working with someone else, turn in **one** solution.

	Name	Student ID#	CS userid
Student #1:			
Student #2:			

Problem 1: All Turned Around

Part 1a: Derivation tree

Consider this proposed subtyping rule that purportedly allows programmers to not worry about the order in which they write a pair:

$$\frac{\mathsf{A1} <: \mathsf{B1} \qquad \mathsf{A2} <: \mathsf{B2}}{(\mathsf{A1} * \mathsf{A2}) <: (\mathsf{B2} * \mathsf{B1})}\ \text{??Sub-twist}$$

Use the above rule (and the subtyping rules in Figure 2) to complete the following derivation:

$$(\mathsf{int} * \mathsf{bool}) <: (\mathsf{bool} * \mathsf{rat})$$

Part 1b: "If it can do anything, he thought, then let it do a flip."

Subtyping rules should preserve a "substitution principle": roughly, if $\mathsf{A} <: \mathsf{B}$ then an expression expecting a B must accept an A.

Find an expression $eBody$ such that evaluating

$$ePaircase = \Big(\mathsf{Pair\text{-}case}\ (\mathsf{Pair}\ (\mathsf{Num}\ {-3})\ (\mathsf{Bfalse}))\ x1\ x2\ eBody\Big)$$

causes an error, even though $ePaircase$ would pass the type checker (supposing that ??Sub-twist were used).

$$eBody = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_$$

Problem 2: While

Consider the following environment-based evaluation rules for a `while` construct. The first expression, eG, is the condition or *guard*; the second expression is the body of the `while`.

$$\frac{env \vdash eG \Downarrow (\mathsf{Btrue}) \qquad env \vdash eBody \Downarrow v1 \qquad env \vdash (\mathsf{While}\ eG\ eBody) \Downarrow v2}{env \vdash (\mathsf{While}\ eG\ eBody) \Downarrow v2}\ \text{Env-while-true}$$

$$\frac{env \vdash eG \Downarrow (\mathsf{Bfalse})}{env \vdash (\mathsf{While}\ eG\ eBody) \Downarrow (\mathsf{Bfalse})}\ \text{Env-while-false}$$

In early versions of Fun, this `while` construct would have been completely useless. In those languages, an expression e that evaluates to a value would *always* evaluate to the same value. Assuming eG has type bool, if eG converges then it evaluates to either (Btrue) or (Bfalse). If eG evaluates to (Btrue), it will evaluate to (Btrue) again the second time around, so the While would always loop forever. If eG evaluates to (Bfalse), then the loop doesn't execute at all.

However, While might be useful in the version of Fun on a4, because Par and Choose allow a single expression to evaluate to different values.

Now that we have a store, While is more clearly useful, because we can use Setref to change the state.

(Having (While (Bfalse) $eBody$) evaluate to (Bfalse) is arbitrary. We would have liked to have it evaluate to unit, but we were trying to keep the language on this assignment from getting too big.)

Part 2a. Design environment-based rules **with a store** that correspond to Env-while-true and Env-while-false.

$$\frac{}{env; S1 \vdash (\mathsf{While}\ eG\ eBody) \Downarrow v2;\ \_\_\_\_\_\_\_}\ \text{SEnv-while-true}$$

$$\frac{}{env; S1 \vdash (\mathsf{While}\ eG\ eBody) \Downarrow (\mathsf{Bfalse});\ \_\_\_\_\_\_\_}\ \text{SEnv-while-false}$$

Part 2b. Complete the following derivations.

$$\frac{}{\underbrace{x : (\mathsf{Location}\ \ell)}_{env}; \underbrace{\ell \triangleright (\mathsf{Num}\ 2)}_{S} \vdash \big(\mathsf{Binop} < (\mathsf{Num}\ 1)\ (\mathsf{Deref}\ (\mathsf{Id}\ x))\big) \Downarrow\ \_\_\_\_\_\_\_\_\_\_;\ \_\_\_\_\_\_\_\_}\ \text{SEnv-binop}$$

$$\frac{}{\underbrace{x : (\mathsf{Location}\ \ell)}_{env}; \underbrace{\ell \triangleright (\mathsf{Num}\ 2)}_{S} \vdash \big(\mathsf{Setref}\ (\mathsf{Id}\ x)\ (\mathsf{Num}\ 0)\big) \Downarrow\ \_\_\_\_\_\_\_\_\_\_;\ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_}\ \text{SEnv-setref}$$

Problem 3: Subtyping

For records, *depth subtyping* says that a record type whose fields are $\mathsf{F}1, \ldots, \mathsf{F}n$ is a subtype of another with the same fields if the field types are "pointwise" subtypes. For example:

$$(\mathsf{record}\ x : \mathsf{pos}, y : \mathsf{int}) \mathrel{<:} (\mathsf{record}\ x : \mathsf{int}, y : \mathsf{rat})$$

because (for the field x) $\mathsf{pos} \mathrel{<:} \mathsf{int}$, and (for the field y) $\mathsf{int} \mathrel{<:} \mathsf{rat}$.

If you think of a pair $(\mathsf{Pair}\ e1\ e2)$ as a record with two (anonymous) fields, depth subtyping for records behaves like the rule Sub-product for pair types.

In *width subtyping*, a record type with fields F is a subtype of a record type with fields G if F is a *superset* of G. For example, a record with 3 fields $x : \mathsf{int}$, $y : \mathsf{int}$, $z : \mathsf{int}$ is a subtype of a record with 2 fields $x : \mathsf{int}, y : \mathsf{int}$.

You can think of the record with 3 fields as a *subclass* of the record with 2 fields: it "inherits" the fields x and y, but adds a new field z.

The following two rules do depth and width subtyping. Sub-record handles the case where the supertype has exactly one field. Implement these rules by extending the function `subtype?` in `a5.rkt`.

$$\frac{\text{there exists } (f : A) \in \{f1 : A1, \cdots, fn : An\} : \qquad f = g \qquad A \mathrel{<:} B}{(\mathsf{record}\ f1 : A1, \cdots, fn : An) \mathrel{<:} (\mathsf{record}\ g : B)}\ \text{Sub-record}$$

$$\frac{m \geq 0 \qquad A \mathrel{<:} (\mathsf{record}\ g1 : B1) \qquad \cdots \qquad A \mathrel{<:} (\mathsf{record}\ gm : Bm)}{A \mathrel{<:} (\mathsf{record}\ g1 : B1, \cdots, gm : Bm)}\ \text{Sub-record-split}$$

Note the condition $m \geq 0$: the grammar allows a Record with *zero* fields, and so does Sub-record-split!

$$\frac{0 \geq 0 \qquad \textit{[zero premises here]}}{A \mathrel{<:} (\mathsf{record}\ )}\ \text{Sub-record-split}$$

Note: You can assume there are **no** duplicate field names, that is, `subtype?` will **never** be called with a type like $(\mathsf{record}\ x : \mathsf{int}, x : \mathsf{bool})$.

Hint: The function `get-field-type` is useful for implementing Sub-record.

Problem 4: Begin and While

Add code to `env-interp` to implement (1) the rule SEnv-begin and (2) the rules for While from Problem 2.