CPSC 311: Definition of Programming Languages: Assignment 4: Subtyping and Stepping

Joshua Dunfield
University of British Columbia

November 27, 2015

The course outline said there would be 3 assignments and a project, and your lowest assignment mark (or 1/3 of the project mark, if lower than the assignments) would be dropped.

This is an **extra** assignment. Because it was not in the course outline, we can't reasonably penalize you for not doing it. Therefore, we will drop your **two** lowest assignment marks (potentially, 2/3 of the project mark, if it is the lowest).

1 Logistics

You may work in teams of **up to 3** 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 a3. 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/~cs311/2015W1/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/~cs311/2015W1/assignments/a4.rkt`

and

`http://www.ugrad.cs.ubc.ca/~cs311/2015W1/assignments/a4-step.rkt`

1.1 Important! (new in a3, still in a4)

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 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!

This assignment is due at 22:30 (10:30pm) on Tuesday, December 1, 2015.

1.2 Handin

You must turn in the assignment using the handin program. For handin, this assignment is called `a4`. Submit **three** files:

- `a4.rkt`
- `a4-step.rkt`
- `README.txt`

(If you choose to try one or more bonus problems, also include a file `bonus.rkt`). Include your name(s) at the top of each file.

If you are working in a team, submit only one set of files. If you have both run handin for a3 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, handin your assignments early and often!

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

2 Overview

This assignment has two separate parts. They are meant to be entirely independent, so you can do them in any order.

- In the first part, you'll extend subtyping to handle *record types*.
- In the second part, you'll extend a *small-step interpreter* with *angelic* and *demonic* nondeterminism. (Angelic nondeterminism is parallel computation!)

3 Syntax

Parts of this assignment build on a3.rkt. Please check the a3 handout to remind yourself of that assignment, and/or consult a3-solution.rkt.

3.1 Types

Concrete syntax

```
⟨Type⟩ ::= Rat
         | Int
         | Pos
         | Bool
         | {List ⟨Type⟩}
         | {* ⟨Type⟩ ⟨Type⟩}
         | {-> ⟨Type⟩ ⟨Type⟩...}
         | {Ref ⟨Type⟩}
         | {Record ⟨FieldTypes⟩...}

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

Abstract syntax

$$
\begin{array}{lrcll}
\text{Types} & A, B & ::= & \mathsf{Rat} & \\
 & & & \mid \mathsf{Int} & \\
 & & & \mid \mathsf{Pos} & \\
 & & & \mid \mathsf{Bool} & \\
 & & & \mid \mathsf{List}\ A & \\
 & & & \mid A * B & \\
 & & & \mid A \rightarrow B & \\
 & & & \mid \mathsf{Ref}\ A & \\
 & & & \mid \mathsf{Record}\ FldType1, \cdots, FldTypen & n \geq 0 \\
\text{FieldTypes} & FldType & ::= & f : A &
\end{array}
$$

The `Ref` type was introduced in `22-subtyping.pdf`.

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 $FldType$.

3.2 Expressions

Concrete syntax

```
⟨E⟩ ::=
      ⋮
      | {downcast ⟨Type⟩ ⟨E⟩}
      | {record ⟨Field⟩...}
      | {dot ⟨E⟩ ⟨id⟩}

⟨Fields⟩ ::= {⟨id⟩ ⟨E⟩}
```

Abstract syntax

$$
\begin{array}{lrcl}
\text{Expressions} & e & ::= & \vdots \\
 & & & \mid (\mathsf{downcast}\ A\ e) \\
 & & & \mid (\mathsf{record}\ Field1, \ldots, Fieldn) \\
 & & & \mid (\mathsf{dot}\ e\ f) \\
\text{Fields} & Field & ::= & f{=}e
\end{array}
$$

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)).

4 Environment-based semantics

In addition to new features like records, rules from a3 have been updated to be environment-based, rather than substitution-based.

Due to time constraints, we have only typeset the rules for downcasts and records. However, we have already implemented all the environment-based evaluation rules, including these:

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

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

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

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

5 Typing

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

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

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

$$\frac{A1 <: B1 \qquad A2 <: B2}{(A1 * A2) <: (B1 * B2)}\ \text{Sub-product} \qquad \frac{A <: B}{(\mathsf{List}\ A) <: (\mathsf{List}\ B)}\ \text{Sub-list}$$

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

Figure 1 Subtyping rules; for record subtyping, see Problem 2

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Figure 2 Typing rules, including references and records

6 Problems

IMPORTANT: Read this before you start coding!

- Don't worry about making your code fast; clarity and correctness are much more important.

  You must implement code that follows the rules; you should also make your code similar to the rules.

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:

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

In the indicated space in a4.rkt (search for "Problem 1a"), use the above rule (and the subtyping rules in Figure 1) to write a derivation tree for

$$(\mathsf{Int} * \mathsf{Bool}) <: (\mathsf{Bool} * \mathsf{Rat})$$

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

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

Find an expression e such that evaluating

$$ePaircase = (\mathsf{pair\text{-}case}\ (\mathsf{pair}\ (\mathsf{num}\ {-3})\ (\mathsf{bfalse}))\ x1\ x2\ e)$$

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

Problem 2: The Width of a Circle

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 `a4.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 3: Angels and Demons

This problem uses a different file, a4-step.rkt.

In `a4-step.rkt` we have implemented a small-step interpreter and added **two** new features, "angelic nondeterminism" par (better known as parallelism) and "demonic nondeterminism" choose. ("Demonic nondeterminism" already made an appearance on the practice midterm.)

Small-step semantics was introduced in `10-contexts`:

`http://www.ugrad.cs.ubc.ca/~cs311/2015W1/notes/10-contexts.pdf`

In the function `reduce`, implement the rules Step-par-left, Step-par-right, Step-choose-left, and Step-choose-right; see Figure 3.

The rules for choose overlap, so you have a choice about which half of a choose to step. To resolve this choice, call the Racket function `random` with argument 2. This will (pseudo-)randomly return 0 or 1. If it returns 0, do Step-choose-left; if it returns 1, do Step-choose-right.

You're now done with choose! However, you aren't done with this problem, because the rule Step-context depends on the definition of evaluation contexts, which has been extended for par. So:

In the function `step`, implement the two additional possibilities for $\mathcal{C}$. It's often a good idea to try to follow the pattern of the existing branches, and that should hold for this problem as well.

Problem 4: Nontermination

This problem is also for `a4-step.rkt`. **Search for "Problem 4" to see where to write your solutions.**

Part 4a

Find Fun expressions $e1$ and $e2$ such that repeatedly stepping

$$(\text{par } e1\ e2)$$

always terminates (results in a value), but repeatedly stepping

$$(\text{choose } e1\ e2)$$

does not always terminate.

Or, explain why no such expressions exist.

Part 4b

Find Fun expressions $e3$ and $e4$ such that repeatedly stepping

$$(\text{choose } e3\ e4)$$

always terminates (results in a value), but repeatedly stepping

$$(\text{par } e3\ e4)$$

does not always terminate.

Or, explain why no such expressions exist.

$\boxed{e1 \longrightarrow e2}$ Expression $e1$ steps to $e2$

Reduction rules:

$$\frac{}{(\mathsf{add}\ (\mathsf{num}\ n_1)\ (\mathsf{num}\ n_2)) \longrightarrow (\mathsf{num}\ n_1{+}n_2)}\ \text{Step-add} \qquad \frac{}{(\mathsf{sub}\ (\mathsf{num}\ n_1)\ (\mathsf{num}\ n_2)) \longrightarrow (\mathsf{num}\ n_1{-}n_2)}\ \text{Step-sub}$$

$$\frac{}{(\mathsf{app}\ (\mathsf{lam}\ x\ eB)\ v) \longrightarrow \mathit{subst}(eB, x, v)}\ \text{Step-app-value}$$

$$\frac{}{(\mathsf{if0}\ (\mathsf{num}\ 0)\ eZ\ eNZ) \longrightarrow eZ}\ \text{Step-if0-zero} \qquad \frac{n \neq 0}{(\mathsf{if0}\ (\mathsf{num}\ n)\ eZ\ eNZ) \longrightarrow eNZ}\ \text{Step-if0-nonzero}$$

$$\frac{}{(\mathsf{with}\ x\ v1\ e2) \longrightarrow \mathit{subst}(e2, x, v1)}\ \text{Step-with} \qquad \frac{}{(\mathsf{rec}\ u\ e) \longrightarrow \mathit{subst}(e, u, (\mathsf{rec}\ u\ e))}\ \text{Step-rec}$$

$$\frac{}{(\mathsf{par}\ v1\ e2) \longrightarrow v1}\ \text{Step-par-left} \qquad \frac{}{(\mathsf{par}\ e1\ v2) \longrightarrow v2}\ \text{Step-par-right}$$

$$\frac{}{(\mathsf{choose}\ e1\ e2) \longrightarrow e1}\ \text{Step-choose-left} \qquad \frac{}{(\mathsf{choose}\ e1\ e2) \longrightarrow e2}\ \text{Step-choose-right}$$

Context rule:

$$\frac{e \longrightarrow e'}{\mathcal{C}[e] \longrightarrow \mathcal{C}[e']}\ \text{Step-context}$$

Definitions of values and evaluation contexts:

$$\begin{array}{lrcl}
\text{Values} & v & ::= & (\mathsf{num}\ n) \\
 & & \mid & (\mathsf{lam}\ x\ e) \\
 & & & \\
\text{Evaluation contexts} & \mathcal{C} & ::= & [\,] \\
 & & \mid & (\mathsf{add}\ \mathcal{C}\ e) \\
 & & \mid & (\mathsf{add}\ v\ \mathcal{C}) \\
 & & \mid & (\mathsf{sub}\ \mathcal{C}\ e) \\
 & & \mid & (\mathsf{sub}\ v\ \mathcal{C}) \\
 & & \mid & (\mathsf{app}\ \mathcal{C}\ e) \\
 & & \mid & (\mathsf{app}\ v\ \mathcal{C}) \\
 & & \mid & (\mathsf{with}\ x\ \mathcal{C}\ e) \\
 & & \mid & (\mathsf{if0}\ \mathcal{C}\ eZ\ eNZ) \\
 & & \mid & (\mathsf{par}\ \mathcal{C}\ e) \\
 & & \mid & (\mathsf{par}\ e\ \mathcal{C})
\end{array}$$

Figure 3 **Small-step semantics (for Problems 3–4)**