CPSC 311: Definition of Programming Languages: Assignment 6: Bidirectional Typing

Joshua Dunfield and Khurram A. Jafery
University of British Columbia

November 27, 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 a6. 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/a6.rkt`

1.1 Important! (new in a3, still in a6)

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!

*Problem **2** must be handed in using `handin` by 22:00 (10:00pm) on Tuesday, December 6, 2016.*

*Problem **1** must be in the box in X235 by 22:00 (10:00pm) on Tuesday, December 6, 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 `a6`. Submit two files:

- `a6.rkt`
- `README.txt`

If you are working in a team, submit only one set of files. If you have both run handin for a6 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⟩`...}`

| `unit`
| `{+` ⟨Type⟩ ⟨Type⟩`}`

| `{all` ⟨symbol⟩ ⟨Type⟩`}`
| ⟨symbol⟩

| `{mu` ⟨symbol⟩ ⟨Type⟩`}`

Abstract syntax

Types A, B ::= rat
| int
| pos
| bool
| $A * B$
| $A \to B$

| unit
| $A + B$

| $\forall a.\ A$
| a

| $\mu a.\ A$

Sum types $A + B$ are new in this assignment. Sum types describe the Inl and Inr expressions from a4.

For example, all of the following expressions have the type bool + rat.

(Inl (Bfalse))
(Inl (Btrue))
(Inr (Num −0.5))
(Inr (Num −2))
(Inr (Num 7))

The first two expressions, (Inl (Bfalse)) and (Inl (Btrue)), have type bool + rat because they are inl applied to something of type bool. The other expressions have type bool + rat because they have inr applied to something of type rat. The "L" in inl refers to the left-hand type bool in bool + rat, and the "R" in inr refers to the right-hand type rat in bool + rat.

Sum types, along with the unit type unit, give us an alternative way to define booleans: Let

$$\textit{weird-bool} \;=\; (\mathsf{unit} + \mathsf{unit})$$

Now, instead of writing (Btrue), we write (Inl (Unit)); instead of (Bfalse), we write (Inr (Unit)); and instead of (Ite e $e1$ $e2$), we write (Sum-case e x1 $e1$ x2 $e2$). The variables x1 and x2 don't matter—they will always be (Unit), which carries no information, so we can't do anything useful with x1 and x2.

Polymorphic types $\forall a.\ A$ were introduced in the lecture notes "lec-bidir-poly".

Recursive types $\mu a.\ A$ are new in this assignment. Unlike all our other types, *there are no expression variants specific to recursive types*. Any expression form can have a recursive type. Recursive types capture the recursive nature of data definitions. For example, a natural number is either zero, or the successor of a natural number. The "either... or" part of this definition is captured by a sum type: we *either* have zero (represented as (Inl (Unit))), *or* the successor of a natural number N (represented as (Inr N)). For example, one is the successor of zero, so one is (Inr N) where N is zero, and zero is represented as (Inr (Unit)), so one is represented as (Inr (Inl (Unit))). Zero carries no further information, so we can express zero as the type unit. So we want to say that

$$\text{a natural number} \quad = \quad \underbrace{\mathsf{unit}}_{\text{I'm zero}} \quad + \quad \underbrace{\text{a natural number}}_{\text{I'm the successor of this number}}$$

This leaves the question of how to handle the recursive nature of this definition: natural numbers are being defined in terms of natural numbers. We will do this using a recursive type. First, we will replace "a natural number" with "*nat*".

$$\mathit{nat} \quad = \quad \underbrace{\mathsf{unit}}_{\text{I'm zero}} \quad + \quad \underbrace{\mathit{nat}}_{\text{I'm the successor of this number}}$$

Then we will do something similar to Rec expressions: we will bind a name and give the whole type that name, similar to how (Rec u e) binds an identifier u that represents the whole expression (Rec u e).

$$\mu \mathit{nat}.\ (\mathsf{unit} + \mathit{nat})$$

We can also use recursive types to define lists: a list of type A is either (Inl unit), representing the empty list, or (Inr (Pair h t)) where h has type A, and t has type *list-of-A*. Following the pattern of *nat*, we get

$$\mu \mathit{list\text{-}of\text{-}A}.\ \Big(\underbrace{\mathsf{unit}}_{\text{I'm empty}} \quad + \quad \underbrace{(A * \mathit{list\text{-}of\text{-}A})}_{\text{I'm a "cons" of an } A \text{ and another list}} \Big)$$

2.2 Expressions

Concrete syntax

```
⟨E⟩ ::= ⋮
      | {Unit}
      | {Inl ⟨E⟩}
      | {Inr ⟨E⟩}
      | {Sum-case ⟨E⟩ ⟨symbol⟩ ⟨E⟩ ⟨symbol⟩ ⟨E⟩}
      | {Anno ⟨E⟩ ⟨Type⟩}
      | {At ⟨E⟩ ⟨Type⟩}
      | {All ⟨symbol⟩ ⟨E⟩}
```

Abstract syntax

```
Expressions  e ::= ⋮
                 | (Unit)
                 | (Inl e)
                 | (Inr e)
                 | (Sum-case e x e x e)
                 | (Anno e A)
                 | (At e A)
                 | (All a e)
```

3 Evaluation semantics

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

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

$$\frac{}{(\mathsf{Btrue}) \Downarrow (\mathsf{Btrue})}\ \text{Eval-true} \qquad \frac{}{(\mathsf{Bfalse}) \Downarrow (\mathsf{Bfalse})}\ \text{Eval-false}$$

$$\frac{e \Downarrow (\mathsf{Btrue}) \qquad e\mathsf{Then} \Downarrow v}{(\mathsf{Ite}\ e\ e\mathsf{Then}\ e\mathsf{Else}) \Downarrow v}\ \text{Eval-ite-true} \qquad \frac{e \Downarrow (\mathsf{Bfalse}) \qquad e\mathsf{Else} \Downarrow v}{(\mathsf{Ite}\ e\ e\mathsf{Then}\ e\mathsf{Else}) \Downarrow v}\ \text{Eval-ite-false}$$

$$\frac{}{(\mathsf{Id}\ x)\ \text{free-variable-error}}\ \text{Eval-id} \qquad \frac{e1 \Downarrow v1 \qquad [v1/x]e2 \Downarrow v2}{(\mathsf{Let}\ x\ e1\ e2) \Downarrow v2}\ \text{Eval-let} \qquad \frac{[eB/u](\mathsf{Rec}\ u\ eB) \Downarrow v}{(\mathsf{Rec}\ u\ eB) \Downarrow v}\ \text{Eval-rec}$$

$$\frac{}{(\mathsf{Lam}\ x\ eB) \Downarrow (\mathsf{Lam}\ x\ eB)}\ \text{Eval-lam} \qquad \frac{e1 \Downarrow (\mathsf{Lam}\ x\ eB) \qquad e2 \Downarrow v2 \qquad [v2/x]eB \Downarrow v}{(\mathsf{App}\ e1\ e2) \Downarrow v}\ \text{Eval-app-value}$$

$$\frac{e1 \Downarrow v1 \qquad e2 \Downarrow v2}{(\mathsf{Pair}\ e1\ e2) \Downarrow (\mathsf{Pair}\ v1\ v2)}\ \text{Eval-pair} \qquad \frac{e \Downarrow (\mathsf{Pair}\ v1\ v2) \qquad [v2/x2][v1/x1]e\mathsf{Body} \Downarrow v}{(\mathsf{Pair\text{-}case}\ e\ x1\ x2\ e\mathsf{Body}) \Downarrow v}\ \text{Eval-pair-case}$$

$$\frac{e \Downarrow v}{(\mathsf{Inl}\ e) \Downarrow (\mathsf{Inl}\ v)}\ \text{Eval-inl} \qquad \frac{e \Downarrow v}{(\mathsf{Inr}\ e) \Downarrow (\mathsf{Inr}\ v)}\ \text{Eval-inr}$$

$$\frac{e \Downarrow (\mathsf{Inl}\ vL) \qquad [vL/xL]eL \Downarrow v}{(\mathsf{Sum\text{-}case}\ e\ xL\ eL\ xR\ eR) \Downarrow v}\ \text{Eval-sum-case-left} \qquad \frac{e \Downarrow (\mathsf{Inr}\ vR) \qquad [vR/xR]eR \Downarrow v}{(\mathsf{Sum\text{-}case}\ e\ xL\ eL\ xR\ eR) \Downarrow v}\ \text{Eval-sum-case-right}$$

$$\frac{e \Downarrow v}{(\mathsf{Anno}\ e\ A) \Downarrow v}\ \text{Eval-anno} \qquad \frac{e \Downarrow v}{(\mathsf{All}\ a\ e) \Downarrow v}\ \text{Eval-all} \qquad \frac{e \Downarrow v}{(\mathsf{At}\ e\ A) \Downarrow v}\ \text{Eval-at}$$

Figure 1 Evaluation rules

4 Type substitution

$$
\begin{aligned}
[A/a]\mathsf{rat} &= \mathsf{rat} \\
[A/a]\mathsf{int} &= \mathsf{int} \\
[A/a]\mathsf{pos} &= \mathsf{pos} \\
[A/a]\mathsf{bool} &= \mathsf{bool} \\
[A/a]\mathsf{unit} &= \mathsf{unit} \\
[A/a](B1 * B2) &= ([A/a]B1) * ([A/a]B2) \\
[A/a](B1 \to B2) &= ([A/a]B1) \to ([A/a]B2) \\
[A/a](B1 + B2) &= ([A/a]B1) + ([A/a]B2) \\
[A/a]a &= A \\
[A/a]b &= b \quad \text{if } a \neq b \\
[A/a](\forall b.\, B) &= \forall b.\, ([A/a]B) \quad \text{if } a \neq b \\
[A/a](\forall a.\, B) &= (\forall a.\, B) \\
[A/a](\mu b.\, B) &= \mu b.\, ([A/a]B) \quad \text{if } a \neq b \\
[A/a](\mu a.\, B) &= (\mu a.\, B)
\end{aligned}
$$

5 Subtyping

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

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

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

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

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

$$\frac{[(\mu a.\, A)/a]A <: B}{(\mu a.\, A) <: B}\ \text{Sub-mu-left} \qquad \frac{A <: [(\mu b.\, B)/b]B}{A <: (\mu b.\, B)}\ \text{Sub-mu-right} \qquad \frac{[b/a]A <: B}{(\mu a.\, A) <: (\mu b.\, B)}\ \text{Sub-mu-both}$$

Figure 2 Subtyping rules

6 Typing

$$\frac{\Gamma(x) = A}{\Gamma \vdash (\mathsf{Id}\ x) \Rightarrow A}\ \text{Synth-var} \qquad \frac{\Gamma \vdash e \Rightarrow A \qquad A <: B}{\Gamma \vdash e \Leftarrow B}\ \text{Check-sub} \qquad \frac{\Gamma \vdash e \Leftarrow A}{\Gamma \vdash (\mathsf{Anno}\ e\ A) \Rightarrow A}\ \text{Synth-anno}$$

$$\frac{\Gamma \vdash e1 \Rightarrow A \to B \qquad \Gamma \vdash e2 \Leftarrow A}{\Gamma \vdash (\mathsf{App}\ e1\ e2) \Rightarrow B}\ \text{Synth-app} \qquad \frac{x : A1, \Gamma \vdash e \Leftarrow A2}{\Gamma \vdash (\mathsf{Lam}\ x\ e) \Leftarrow A1 \to A2}\ \text{Check-lam}$$

$$\frac{u : A, \Gamma \vdash e \Leftarrow A}{\Gamma \vdash (\mathsf{Rec}\ u\ e) \Leftarrow A}\ \text{Check-rec}$$

$$\frac{\Gamma \vdash e1 \Rightarrow A1 \qquad x : A1, \Gamma \vdash e2 \Leftarrow A}{\Gamma \vdash (\mathsf{Let}\ x\ e1\ e2) \Leftarrow A}\ \text{Check-let} \qquad \frac{\Gamma \vdash e1 \Rightarrow A1 \qquad x : A1, \Gamma \vdash e2 \Rightarrow A}{\Gamma \vdash (\mathsf{Let}\ x\ e1\ e2) \Rightarrow A}\ \text{Synth-let}$$

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

$$\frac{\mathsf{op} : A1 * A2 \to B \qquad \Gamma \vdash e1 \Leftarrow A1 \qquad \Gamma \vdash e2 \Leftarrow A2}{\Gamma \vdash (\mathsf{Binop}\ \mathsf{op}\ e1\ e2) \Rightarrow B}\ \text{Synth-binop}$$

$$\frac{a\ \mathsf{type}, \Gamma \vdash e \Leftarrow A}{\Gamma \vdash (\mathsf{All}\ a\ e) \Leftarrow \forall a.\ A}\ \text{Check-all} \qquad \frac{\Gamma \vdash e \Rightarrow \forall b.\ B1}{\Gamma \vdash (\mathsf{At}\ e\ A) \Rightarrow [A/b]B1}\ \text{Synth-at}$$

$$\frac{}{\Gamma \vdash (\mathsf{Btrue}) \Rightarrow \mathsf{bool}}\ \text{Synth-btrue} \qquad \frac{}{\Gamma \vdash (\mathsf{Bfalse}) \Rightarrow \mathsf{bool}}\ \text{Synth-bfalse}$$

$$\frac{\Gamma \vdash e \Leftarrow \mathsf{bool} \qquad \Gamma \vdash e1 \Leftarrow A \qquad \Gamma \vdash e2 \Leftarrow A}{\Gamma \vdash (\mathsf{Ite}\ e\ e1\ e2) \Leftarrow A}\ \text{Check-ite}$$

$$\frac{\Gamma \vdash e1 \Leftarrow A1 \qquad \Gamma \vdash e2 \Leftarrow A2}{\Gamma \vdash (\mathsf{Pair}\ e1\ e2) \Leftarrow (A1 * A2)}\ \text{Check-pair} \qquad \frac{\Gamma \vdash e \Rightarrow (A1 * A2) \qquad x1 : A1, x2 : A2, \Gamma \vdash e\mathsf{Body} \Leftarrow A}{\Gamma \vdash (\mathsf{Pair\text{-}case}\ e\ x1\ x2\ e\mathsf{Body}) \Leftarrow A}\ \text{Check-pair-case}$$

$$\frac{\Gamma \vdash e1 \Rightarrow A1 \qquad \Gamma \vdash e2 \Rightarrow A2}{\Gamma \vdash (\mathsf{Pair}\ e1\ e2) \Rightarrow (A1 * A2)}\ \text{Synth-pair} \qquad \frac{}{\Gamma \vdash (\mathsf{Unit}) \Rightarrow \mathsf{unit}}\ \text{Synth-unit}$$

$$\frac{\Gamma \vdash e1 \Leftarrow A1}{\Gamma \vdash (\mathsf{Inl}\ e1) \Leftarrow (A1 + A2)}\ \text{Check-inl} \qquad \frac{\Gamma \vdash e2 \Leftarrow A2}{\Gamma \vdash (\mathsf{Inr}\ e2) \Leftarrow (A1 + A2)}\ \text{Check-inr}$$

$$\frac{\Gamma \vdash e \Rightarrow (A1 + A2) \qquad x1 : A1, \Gamma \vdash e1 \Leftarrow B \qquad x2 : A2, \Gamma \vdash e2 \Leftarrow B}{\Gamma \vdash (\mathsf{Sum\text{-}case}\ e\ x1\ e1\ x2\ e2) \Leftarrow B}\ \text{Check-sum-case}$$

$$\frac{\Gamma \vdash e \Leftarrow [(\mu a.\ A)/a]A}{\Gamma \vdash e \Leftarrow \mu a.\ A}\ \text{Check-mu} \qquad \frac{\Gamma \vdash e \Rightarrow \mu a.\ A}{\Gamma \vdash e \Rightarrow [(\mu a.\ A)/a]A}\ \text{Synth-mu}$$

Figure 3 **Bidirectional typing rules**

7 Problems

Problem 1 is not a coding problem. Turn it 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.

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

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

Problem 1: Un type sans lumière

Part 1a. Complete the following derivation.

$$\dfrac{\dfrac{\Gamma \vdash (\mathsf{Inl}\ (\mathsf{Unit})) \Leftarrow \mu a.\ (\mathsf{unit} + a)}{\Gamma \vdash (\mathsf{Inr}\ (\mathsf{Inl}\ (\mathsf{Unit}))) \Leftarrow \mathsf{unit} + \mu a.\ (\mathsf{unit} + a)}\ \text{Check-inr}}{\underbrace{f : \forall b.\ (\mu a.\ (b + a) \to \mathsf{pos}), \emptyset}_{\Gamma} \vdash (\mathsf{Inr}\ (\mathsf{Inl}\ (\mathsf{Unit}))) \Leftarrow \mu a.\ (\mathsf{unit} + a)}\ \text{Check-mu}$$

Part 1b. Complete the following derivation.

$$\underbrace{f : \forall b.\ (\mu a.\ (b + a) \to \mathsf{pos}), \emptyset}_{\Gamma} \vdash \Big(\mathsf{App}\ \big(\mathsf{At}\ (\mathsf{Id}\ f)\ \mathsf{unit}\big)\ \big(\mathsf{Inr}\ (\mathsf{Inl}\ (\mathsf{Unit}))\big)\Big) \Rightarrow \_\_\_\_\_$$

Problem 2: Bidirectional typing

Part 2a. In `a6.rkt`, edit the functions `check` and/or `inner-synth` to implement the rule Check-sum-case (Figure 3).

Part 2b. In `a6.rkt`, edit the functions `check` and/or `synth` to implement the rules Check-mu and Synth-mu (Figure 3).

8 Bonus problems

Bonus problem X1: Intersection types

Recursive types $\mu\alpha.A$ are *property types*: they are not tied to a particular expression variant. Another example of a property type is the *intersection type*, written $A1 \cap A2$. If e has type $A1 \cap A2$, that means that e has, simultaneously, type $A1$ and type $A2$. For example, if we want to overload a function "plus" to work on pairs of numbers (with the usual meaning of addition) *and* pairs of booleans (with the meaning "or"), then plus would have the type

$$\big((\mathsf{rat} * \mathsf{rat}) \to \mathsf{rat}\big) \cap \big((\mathsf{bool} * \mathsf{bool}) \to \mathsf{bool}\big)$$

The "introduction rule" for intersection, like the rule Check-mu, works on any kind of expression:

$$\frac{\Gamma \vdash e \Leftarrow A1 \qquad \Gamma \vdash e \Leftarrow A2}{\Gamma \vdash e \Leftarrow (A1 \cap A2)}\ \text{Check-sect}$$

The "elimination rules" for intersection also work on any kind of expression:

$$\frac{\Gamma \vdash e \Rightarrow (A1 \cap A2)}{\Gamma \vdash e \Rightarrow A1}\ \text{Synth-sect-1} \qquad \frac{\Gamma \vdash e \Rightarrow (A1 \cap A2)}{\Gamma \vdash e \Rightarrow A2}\ \text{Synth-sect-2}$$

For example, assuming a function plus with the above type:

$$\Gamma \;=\; \mathtt{plus} : \big((\mathsf{rat} * \mathsf{rat}) \to \mathsf{rat}\big) \cap \big((\mathsf{bool} * \mathsf{bool}) \to \mathsf{bool}\big),\ \emptyset$$

we can apply plus to a pair of Booleans by using rule Synth-sect-2:

$$\frac{\dfrac{\dfrac{\Gamma(\mathtt{plus}) = \big((\mathsf{rat} * \mathsf{rat}) \to \mathsf{rat}\big) \cap \big((\mathsf{bool} * \mathsf{bool}) \to \mathsf{bool}\big)}{\Gamma \vdash (\mathsf{Id}\ \mathtt{plus}) \Rightarrow \big((\mathsf{rat} * \mathsf{rat}) \to \mathsf{rat}\big) \cap \big((\mathsf{bool} * \mathsf{bool}) \to \mathsf{bool}\big)}\ \text{Synth-var}}{\Gamma \vdash (\mathsf{Id}\ \mathtt{plus}) \Rightarrow \big((\mathsf{bool} * \mathsf{bool}) \to \mathsf{bool}\big)}\ \text{Synth-sect-2} \qquad \Gamma \vdash \big(\mathsf{Pair}\ (\mathsf{Bfalse})\ (\mathsf{Btrue})\big) \Leftarrow \mathsf{bool} * \mathsf{bool}}{\Gamma \vdash \Big(\mathsf{App}\ (\mathsf{Id}\ \mathtt{plus})\ \big(\mathsf{Pair}\ (\mathsf{Bfalse})\ (\mathsf{Btrue})\big)\Big) \Rightarrow \mathsf{bool}}\ \text{Synth-app}$$

Part X1a. Copy a6.rkt to bonus.rkt. Implement Check-sect, Synth-sect-1, and Synth-sect-2. You may want to read up on backtracking search algorithms.

Part X1b. Design and implement *subtyping* rules for intersection types.