Assignment 4: Sample Solution

Joshua Dunfield
March 14, 2018

Name(s):  Estimated time spent (per person):

**Note:** Questions marked with a + are bonus questions. You can receive full marks without doing them.
### Typing

#### Types

\[ S, T ::= \text{unit} \quad \text{unit type} \]
\[ | \text{int} \quad \text{type of integers} \]
\[ | \text{bool} \quad \text{type of booleans} \]
\[ | S \rightarrow T \quad \text{type of functions on } S \text{ that produce } T \]
\[ | S \times T \quad \text{type of pairs of one } S \text{ and one } T \]
\[ | S + T \quad \text{disjoint union or sum type: contains either an } S \text{ or a } T \]

#### Typing contexts

\[ \Gamma ::= \emptyset \quad \text{empty context} \]
\[ | \Gamma, x : S \quad x \text{ has type } S \]

Now consider the typing rules in Figure 1.

---

**Figure 1** Typing with functions, integers, booleans, sums (unions), and pairs (structs)
§1 Typing

**Question 1(a).** Complete the following typing derivation. You can write $\Gamma$ instead of $y : \text{bool}$.

$$
\begin{array}{c}
\frac{(y : \text{bool}) \in \Gamma}{\Gamma \vdash y : \text{bool}} \quad \text{type-assum} \\
\frac{\Gamma \vdash 1 : \text{int}}{\Gamma \vdash 1 : \text{int}} \quad \text{intIntro} \\
\frac{\Gamma \vdash 0 : \text{int}}{\Gamma \vdash 0 : \text{int}} \quad \text{intIntro} \\
\hline
y : \text{bool} \vdash (\text{Ite } y \ 1 \ 0) : \text{int} \quad \text{type-ite}
\end{array}
$$

**Question 1(b).** Complete the following typing derivation. You can write $\Gamma$ instead of $z : (\text{int + bool})$.

$$
\begin{array}{c}
\frac{(z : (\text{int + bool})) \in \Gamma}{\Gamma \vdash z : (\text{int + bool})} \quad \text{type-assum} \\
\frac{\Gamma, x : \text{int} \vdash \text{True} : \text{bool}}{\Gamma, x : \text{int} \vdash \text{True} : \text{bool}} \quad \text{type-true} \\
\frac{(y : \text{bool}) \in \Gamma, y : \text{bool}}{\Gamma, y : \text{bool} \vdash y : \text{bool}} \quad \text{type-assum} \\
\hline
\Gamma \vdash (\text{Case } z (x \Rightarrow \text{True}) (y \Rightarrow y)) : \text{bool} \quad +\text{Elim}
\end{array}
$$

**Question 1(c)+.** Booleans are not really necessary, because we can write $(\text{Inj}_1())$ instead of True, and $(\text{Inj}_2())$ instead of False, and use Case instead of Ite. Translate the expression from 1(a), $(\text{Ite } y \ 1 \ 0)$, into an expression that has type int under the typing context

$y : (\text{unit + unit})$

**Hint:** think about the derivation in 1(b).

Start with the expression from 1(a):

$$(\text{Ite } y \ 1 \ 0)$$

Neither True nor False appear in this expression, so the part about “writing $(\text{Inj}_1())$ instead of True, and $(\text{Inj}_2())$ instead of False” isn’t relevant. We only have to change Ite into Case:

$$(\text{Case } y (x_1 \Rightarrow 1) (x_2 \Rightarrow 0))$$

The question didn’t ask for a typing derivation, but here’s one anyway. Let $\Gamma = y : (\text{unit + unit})$.

$$
\begin{array}{c}
\frac{(y : (\text{unit + unit})) \in \Gamma}{\Gamma \vdash y : (\text{unit + unit})} \quad \text{type-assum} \\
\frac{\Gamma, x_1 : \text{unit} \vdash 1 : \text{int}}{\Gamma, x_1 : \text{unit} \vdash 1 : \text{int}} \quad \text{intIntro} \\
\frac{\Gamma, x_2 : \text{unit} \vdash 0 : \text{int}}{\Gamma, x_2 : \text{unit} \vdash 0 : \text{int}} \quad \text{intIntro} \\
\hline
\Gamma \vdash (\text{Case } y (x_1 \Rightarrow 1) (x_2 \Rightarrow 0)) : \text{int} \quad +\text{Elim}
\end{array}
$$
§1 Typing

2 Mirror World

Consider the sequent calculus rules in Figure 2.

\[
\begin{array}{ll}
\Gamma \vdash A \text{ true} & \text{Under assumptions } \Gamma, \text{ formula } A \text{ is true} \\
\frac{\frac{x[A \text{ true}] \in \Gamma}{\Gamma \vdash A \text{ true}}}{\text{sc-assum}} & \frac{\frac{\Gamma, x[A \text{ true}] \vdash B \text{ true}}{\Gamma \vdash (A \supset B) \text{ true}}}{\text{sc-}\supset \text{Intro}} & \frac{\frac{\Gamma \vdash A \supset B \text{ true}}{\Gamma \vdash B \text{ true}}}{\Gamma \vdash A \text{ true}} & \frac{\frac{\Gamma \vdash A \text{ true}}{\Gamma \vdash A \supset B \text{ true}}}{\text{sc-}\supset \text{Elim}} \\
\frac{\frac{\Gamma \vdash A \text{ true}}{\Gamma \vdash \text{ true}}}{\text{sc-trueIntro}} & \frac{\frac{\frac{\Gamma \vdash A \text{ true}}{\Gamma \vdash A \lor B \text{ true}}}{\text{sc-}\lor \text{Intro1}}}{\frac{\frac{\Gamma \vdash B \text{ true}}{\Gamma \vdash A \lor B \text{ true}}}{\text{sc-}\lor \text{Intro2}}} & \frac{\frac{\frac{\Gamma \vdash A \lor B \text{ true}}{\Gamma, x[A \text{ true}] \vdash C \text{ true}}}{\text{sc-}\lor \text{Elim}}}{\Gamma, y[B \text{ true}] \vdash C \text{ true} & \Gamma \vdash C \text{ true}} \\
\frac{\frac{\Gamma, x[A \text{ true}] \vdash B \text{ true}}{\Gamma \vdash A \lor B \text{ true}}}{\text{sc-}\& \text{Intro}} & \frac{\frac{\Gamma \vdash A \text{ true}}{\Gamma \vdash A \& B \text{ true}}}{\text{sc-}\& \text{Elim1}} & \frac{\frac{\Gamma \vdash B \text{ true}}{\Gamma \vdash A \lor B \text{ true}}}{\text{sc-}\& \text{Elim2}} \\
\end{array}
\]

**Figure 2** Sequent calculus, with \( \supset \), \text{ True} \( \lor \), and \( \& \)

Some of the typing rules from Figure 1 have a curious property: if we change some of the metavariables, remove the expression and colon, translate the types, and add the word \text{ true}, we get a rule in Figure 2.

For example, the rule \( \times \text{Elim1} \) becomes the rule \( \text{sc-}\& \text{Elim1} \):

\[
\begin{array}{ll}
\Gamma \vdash e : S_1 \times S_2 & \Gamma \vdash e : A_1 \times A_2 \\
\Gamma \vdash \text{ proj}_1 e : S_1 \rightarrow A_1 & \Gamma \vdash \text{ proj}_1 e : A_1 \\
\end{array}
\]

Here we translated the type \( S_1 \times S_2 \) to \( A_1 \& A_2 \) by replacing \( \times \) with \( \& \). The types can be translated as follows:

\[
\begin{array}{ccc}
\text{unit} & \rightarrow & \text{True} \\
\times & \rightarrow & \& \\
+ & \rightarrow & \lor \\
\rightarrow & \rightarrow & \supset \\
\end{array}
\]

Assumptions need some extra work; for example, in translating \( \rightarrow \text{Intro} \), \( x : S \) becomes \( x[S \text{ true}] \):

\[
\begin{array}{ll}
\Gamma, x : S \vdash e : T & \text{instead of } \Gamma, x : A \vdash e : B \\
\Gamma \vdash (\text{ Lam } x e) : S \rightarrow T & \Gamma \vdash (\text{ Lam } x e) : A \rightarrow B \\
\text{sc-}\supset \text{Intro} & \Gamma \vdash (A \supset B) \text{ true}
\end{array}
\]

Not all the typing rules in Figure 1 have meaningful translations. The rules involving arithmetic, among others, do not lead to anything interesting.
Question 2(a). Following the procedure above, translate $\rightarrow$Elim. Indicate which rule from Figure 2 was “rediscovered” by this translation. (For example, by translating $\times$Elim1, I rediscovered sc-$\rightarrow$Elim1.)

$$
\Gamma \vdash e_1 : S \rightarrow T \quad \Gamma \vdash e_2 : S \\
\Gamma \vdash (\text{Call } e_1 \ e_2) : S \rightarrow T
\quad \Rightarrow
\Gamma \vdash e_1 : A \rightarrow B \quad \Gamma \vdash e_2 : A \\
\Gamma \vdash (\text{Call } e_1 \ e_2) : A \rightarrow B
\quad \Rightarrow
\Gamma \vdash A \supset B \ true \quad \Gamma \vdash A \ true
\quad \text{sc-$\rightarrow$Elim}
$$

Question 2(b). In addition to translating rules, we can translate derivations.

$$
(Lam \ f \ (Lam \ g \ (Lam \ x \ (\text{Call } f \ (\text{Call } g \ x)))))
$$

This expression is a function that creates a new function, $(Lam \ x \ (\text{Call } f \ (\text{Call } g \ x)))$, which *composes* $f$ and $g$: its body, in less Racket syntax, is

$$
f(g(x))
$$

You may want to experiment with the expression in Racket (*lambda.rkt* on the course website).

The following is a typing derivation for the expression.

Let $\Gamma_{fgx} = f : (S_2 \rightarrow S_3), g : (S_1 \rightarrow S_2), x : S_1$.

$$
\begin{array}{c}
(f : S_2 \rightarrow S_3) \in \Gamma_{fgx} \quad (g : S_1 \rightarrow S_2) \in \Gamma_{fgx} \quad (x : S_1) \in \Gamma_{fgx} \\
\Gamma_{fgx} \vdash f : S_2 \rightarrow S_3 \\
\Gamma_{fgx} \vdash g : S_1 \rightarrow S_2 \\
\Gamma_{fgx} \vdash x : S_1
\end{array}
\quad \Rightarrow
\begin{array}{c}
\Gamma_{fgx} \vdash (\text{Call } f \ (\text{Call } g \ x)) : S_3 \\
\Gamma_{fgx} \vdash (\text{Call } f \ (\text{Call } g \ x)) \ : \ S_3
\end{array}
\quad \text{→Elim}

\begin{array}{c}
(f : S_2 \rightarrow S_3), g : (S_1 \rightarrow S_2) \vdash (\text{Lam } x \ (\text{Call } f \ (\text{Call } g \ x))) : (S_1 \rightarrow S_3) \\
\end{array}
\quad \text{→Intro}

\begin{array}{c}
\emptyset \vdash (\text{Lam } f \ (\text{Lam } g \ (\text{Lam } x \ (\text{Call } f \ (\text{Call } g \ x))))) : (S_2 \rightarrow S_3) \rightarrow (S_1 \rightarrow S_2) \rightarrow (S_1 \rightarrow S_3)
\end{array}
\quad \text{→Intro}

Using the rule you translated in 2(a), complete the following *sequent calculus* derivation. Above, I wrote the expressions in gray to make it easier to ignore them as you complete the derivation.

You may write $\Gamma_{fgx}$ instead of $f \ [(A_2 \supset A_3) \ true], g \ [(A_1 \supset A_2) \ true], x \ [A_1 \ true]$.

$$
\begin{array}{c}
\Gamma_{fgx} \vdash A_2 \supset A_3 \\
\Gamma_{fgx} \vdash A_2 \ supset A_3 \\
\Gamma_{fgx} \vdash A_1 \ supset A_2 \ true \\
\Gamma_{fgx} \vdash A_1 \ supset A_2 \ true \\
\Gamma_{fgx} \vdash A_1 \ true \\
\Gamma_{fgx} \vdash A_1 \ true \\
\end{array}
\quad \text{sc-$\rightarrow$Elim}
\quad \text{sc-$\rightarrow$Intro}
\quad \text{sc-$\supset$Intro}
\quad \text{sc-$\supset$Intro}
\quad \text{sc-$\supset$Intro}
\quad \text{sc-$\supset$Intro}
\quad \text{sc-$\supset$Intro}
\quad \text{sc-$\supset$Intro}
\quad \text{sc-$\supset$Intro}
\quad \text{sc-$\supset$Intro}
\quad \text{sc-$\supset$Intro}
\quad \text{sc-$\supset$Intro}
\quad \text{sc-$\supset$Intro}
\quad \text{sc-$\supset$Intro}
\quad \text{sc-$\supset$Intro}
\quad \text{sc-$\supset$Intro}
\quad \text{sc-$\supset$Intro}
\quad \text{sc-$\supset$Intro}
\quad \text{sc-$\supset$Intro}
$$
Question 2(c)+.

Part the First. We have two elimination rules for &, which separately extract a subformula. Design a single sequent-calculus elimination rule for &. Hint: think about the structure of sc-∨Elim. Second hint: think about the connection between \((P_1 & P_2) \supset Q\) and \(P_1 \supset (P_2 \supset Q)\), and the shape of a derivation of \(\emptyset \vdash P_1 \supset (P_2 \supset Q)\) true: how many assumptions are needed within that derivation?

Rule sc-∨Elim case-analyzes \((A \lor B)\), leading to two cases (the second and third premises of sc-∨Elim), one assuming \(A\) true and one assuming \(B\) true. This mirrors the introduction rules for ∨, in which we “put in” either \(A\) true or \(B\) true.

The introduction rule for & puts in both \(A\) true and \(B\) true. If we case-analyze \((A & B)\), there is only one case—the case in which both \(A\) true and \(B\) true were used by sc-&Intro to derive \((A & B)\) true.

So a single elimination rule for & needs a premise with two assumptions:

\[
\Gamma \vdash (A & B) \text{ true} \quad \Gamma, x \vdash [A \text{ true}], y \vdash [B \text{ true}] \vdash C \text{ true} \quad \text{sc-&Elim-new}
\]

Part the Second. Translate your new &-elimination rule into a typing rule. You will need to extend the grammar of expressions e.

As an intermediate step, we turn & into \(\times\) and change the assumptions in \(\Gamma\):

\[
\Gamma \vdash (S_1 \times S_2) \text{ true} \quad \Gamma, x : S_1, y : S_2 \vdash T \text{ true} \quad \text{type-&Elim-new}
\]

We need to create a new expression form with a subexpression for each premise, which leads to a rule that must be shaped like this:

\[
\Gamma \vdash e_0 : (S_1 \times S_2) \quad \Gamma, x : S_1, y : S_2 \vdash e_{12} : T \quad \text{type-&Elim-new}
\]

Since \(x : S_1\) and \(y : S_2\) are assumptions in the second premise, the variables \(x\) and \(y\) can be used in \(e_{12}\). To allow programmers to choose those variable names, they also need to appear in the expression:

\[
e ::= \ldots | (\text{Split } e \times x \times e)
\]

With our new expression form, we can finish the typing rule:

\[
\Gamma \vdash e_0 : (S_1 \times S_2) \quad \Gamma, x : S_1, y : S_2 \vdash e_{12} : T \quad \Gamma \vdash (\text{Split } e_0 \times x \times e_{12}) : T \quad \text{type-&Elim-new}
\]

Part the Third. Extend the small-step semantics with a reduction rule for your new expression form, and extend the grammar \(C\) of contexts as appropriate.

\[
(\text{Split } (\text{Pair } v_1 \times v_2) \times_{12} e_{12}) \rightarrowR [v_1/x_1][v_2/x_2]e_{12} \quad \text{red-split}
\]

\[
C ::= \ldots | (\text{Split } C \times x \times e)
\]