Assignment 4: Sample Solution

Joshua Dunfield

March 18, 2019

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

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

1 Typing

Types

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

Typing contexts

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

Now consider the typing rules in Figure 1.

\[
\Gamma \vdash e : T \ \text{Under assumptions } \Gamma, \text{ expression } e \text{ has type } T
\]

\[\vdash (x : S) \in \Gamma \quad \text{type-assum} \quad \Gamma, x : S \vdash e : T \quad \text{→Intro} \quad \Gamma \vdash (\text{Lam } x \ e) : (S \rightarrow T) \quad \text{→Elim} \]

\[
\Gamma \vdash () : \text{unit} \quad \text{unitIntro} \quad \Gamma \vdash \text{True} : \text{bool} \quad \text{type-true} \quad \Gamma \vdash \text{False} : \text{bool} \quad \text{type-false}
\]

\[\Gamma \vdash n : \text{int} \quad \text{intIntro} \quad \Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \text{int} \quad \text{type-add} \quad \Gamma \vdash (e_1 + e_2) : \text{int} \]

\[\Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \text{int} \quad \text{type-equals} \quad \Gamma \vdash (e_1 = e_2) : \text{bool} \]

\[\Gamma \vdash e_1 : \text{int} \quad \Gamma \vdash e_2 : \text{int} \quad \text{type-sub} \quad \Gamma \vdash (e_1 - e_2) : \text{int} \quad \Gamma \vdash (e_1 < e_2) : \text{bool} \quad \text{type-lt}
\]

\[\Gamma \vdash e : \text{bool} \quad \text{type-ite} \quad \Gamma \vdash e_\text{then} : T \quad \Gamma \vdash e_\text{else} : T \quad \Gamma \vdash (\text{Ite } e e_\text{then} e_\text{else}) : T
\]

\[\Gamma \vdash e_1 : S_1 \quad \text{Intro1} \quad \Gamma \vdash (\text{Inj}_1 e_1) : (S_1 + S_2) \]

\[\Gamma \vdash e_2 : S_2 \quad \text{Intro2} \quad \Gamma \vdash (\text{Inj}_2 e_2) : (S_1 + S_2) \]

\[\Gamma, x_1 : S_1 \vdash e_1 : T \quad \Gamma, x_2 : S_2 \vdash e_2 : T \quad \text{→Elim} \quad \Gamma \vdash (\text{Case } e (x_1 \Rightarrow e_1) (x_2 \Rightarrow e_2)) : T
\]

\[\Gamma \vdash e_1 : S_1 \quad \Gamma \vdash e_2 : S_2 \quad \text{×Intro} \quad \Gamma \vdash e : (S_1 \times S_2) \quad \text{×Elim1} \quad \Gamma \vdash e_1 : (S_1 \times S_2) \quad \text{×Elim2} \]

\[\Gamma \vdash e_2 : S_2 \quad \Gamma \vdash (\text{Proj}_1 e) : S_1 \quad \Gamma \vdash (\text{Proj}_2 e) : S_2
\]

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 $y : \text{bool}$.

\[
\frac{(y : \text{bool}) \in \Gamma}{\Gamma \vdash y : \text{bool}} \quad \text{type-assum} \quad \frac{}{\Gamma \vdash 1 : \text{int}} \quad \text{intIntro} \quad \frac{}{\Gamma \vdash 0 : \text{int}} \quad \text{intIntro} \quad \frac{}{y : \text{bool} \vdash (\text{Ite } y 1 0) : \text{int}} \quad \text{type-ite}
\]

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

\[
\frac{(z : (\text{int} + \text{bool})) \in \Gamma}{\Gamma \vdash z : (\text{int} + \text{bool})} \quad \text{type-assum} \quad \frac{}{\Gamma, x : \text{int} \vdash \text{True} : \text{bool}} \quad \text{type-true} \quad \frac{(y : \text{bool}) \in \Gamma, y : \text{bool}}{\Gamma, y : \text{bool} \vdash y : \text{bool}} \quad \text{type-assum} \quad \frac{}{z : (\text{int} + \text{bool}) \vdash (\text{Case } z (x \Rightarrow \text{True}) (y \Rightarrow y)) : \text{bool}} \quad \text{+Elim}
\]

Question 1(c). Booleans are not really necessary, because we can write $(\text{Inj}_1 (\cdot))$ instead of True, and $(\text{Inj}_2 (\cdot))$ 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 $y : (\text{unit} + \text{unit})$.

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; the part about “writing $(\text{Inj}_1 (\cdot))$ instead of True and $(\text{Inj}_2 (\cdot))$ instead of False” is meant to tell you how your expression needs to behave.) 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 $\Gamma = y : (\text{unit} + \text{unit})$.

\[
\frac{(y : (\text{unit} + \text{unit})) \in \Gamma}{\Gamma \vdash y : (\text{unit} + \text{unit})} \quad \text{type-assum} \quad \frac{}{\Gamma, x_1 : \text{unit} \vdash 1 : \text{int}} \quad \text{intIntro} \quad \frac{}{\Gamma, x_2 : \text{unit} \vdash 0 : \text{int}} \quad \text{intIntro} \quad \frac{}{\Gamma \vdash (\text{Case } y (x_1 \Rightarrow 1) (x_2 \Rightarrow 0)) : \text{int}} \quad \text{+Elim}
\]
§1 Typing

2 Mirror World

Consider the sequent calculus rules in Figure 2.

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

Figure 2 Sequent calculus, with $\supset$, True, $\lor$, and $\land$

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 true, we get a rule in Figure 2.

For example, the rule $\times$Elim1 becomes the rule sc-$\land$Elim1:

\[
\begin{align*}
\Gamma \vdash e : S_1 \times S_2 & \Rightarrow \Gamma \vdash \text{proj}_1 e : S_1 \\
\Gamma \vdash \text{proj}_1 e : A_1 & \Rightarrow \Gamma \vdash A_1 = \Rightarrow \Gamma \vdash A_1 \\
\end{align*}
\]

\[
\begin{align*}
\Gamma \vdash e : A_1 \land A_2 & \Rightarrow \Gamma \vdash A_1 = \Rightarrow \Gamma \vdash A_1 \\
\end{align*}
\]

\[
\begin{align*}
\Gamma \vdash A_1 \land A_2 & \Rightarrow \Gamma \vdash A_1 = \Rightarrow \Gamma \vdash A_1 \\
\end{align*}
\]

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

- unit becomes True
- $\times$ becomes $\land$
- $+$ becomes $\lor$
- $\rightarrow$ becomes $\supset$

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

\[
\begin{align*}
\Gamma, x : S \vdash e : T & \Rightarrow \Gamma, x : A \vdash e : B \\
\Gamma \vdash (\text{Lam } x \ e) : S \rightarrow T & \Rightarrow \Gamma \vdash (\text{Lam } x \ e) : A \rightarrow B \\
\Gamma, x[A \text{ true}] \vdash B \text{ true} & \Rightarrow \Gamma \vdash (A \supset B) \text{ true} \quad \text{sc-\supset\text{Intro}} \\
\end{align*}
\]

Not all the typing rules in Figure 1 have meaningful translations. The rules involving arithmetic, among others, do not lead to anything interesting.
§2 Mirror World

**Question 2(a).** Following the procedure above, translate \( \rightarrow \text{Elim} \). Indicate which rule from Figure 2 was “rediscovered” by this translation. (For example, by translating \( \times \text{Elim1} \), I rediscovered sc-&Elim1.)

\[
\begin{align*}
\Gamma \vdash e_1 : S &\rightarrow T & \Gamma \vdash e_2 : S &\implies \Gamma \vdash (\text{Call } e_1, e_2) : S \rightarrow T \\
\Gamma \vdash e_1 : A &\rightarrow B & \Gamma \vdash e_2 : A &\implies \Gamma \vdash (\text{Call } e_1, e_2) : A \rightarrow B \\
\Gamma \vdash A &\supset B \text{ true} & \Gamma \vdash A &\text{ true sc-\&Elim}
\end{align*}
\]

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

\[
\begin{align*}
&\text{Let } \Gamma \\
&\text{ you may want to experiment with the expression in Racket (lambda.rkt on the course website). The}
\end{align*}
\]

The following is a typing derivation for the expression.

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

\[
\begin{align*}
(f : S_2 \rightarrow S_3) &\in \Gamma_{f gx} \text{ type-assum} & (g : S_1 \rightarrow S_2) &\in \Gamma_{f gx} \text{ type-assum} & (x : S_1) &\in \Gamma_{f gx} \text{ type-assum} \\
\Gamma_{f gx} \vdash f : S_2 \rightarrow S_3 &\implies \Gamma_{f gx} \vdash g : S_1 \rightarrow S_2 & \Gamma_{f gx} \vdash x : S_1 &\rightarrow \text{Elim} & \Gamma_{f gx} \vdash (\text{Call } g \ x) : S_2 &\rightarrow \text{Elim} & \Gamma_{f gx} \vdash (\text{Call } f \ (\text{Call } g \ x)) : S_3 &\rightarrow \text{Intro} \\
&\implies \Gamma_{f gx} \vdash (\text{Call } f \ (\text{Call } g \ x)) : S_3 &\rightarrow \text{Intro} & \Gamma_{f gx} \vdash (\text{Call } f \ (\text{Call } g \ x)) : S_3 &\rightarrow \text{Intro} & \Gamma_{f gx} \vdash (\text{Call } f \ (\text{Call } g \ x)) : S_3 &\rightarrow \text{Intro} & \Gamma_{f gx} \vdash (\text{Call } f \ (\text{Call } g \ x)) : S_3 &\rightarrow \text{Intro} \\
\end{align*}
\]

Using the rules 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 \( \Gamma_{f gx} \) instead of \( f \left[ (A_2 \supset A_3) \text{ true} \right], g \left[ (A_1 \supset A_2) \text{ true} \right], x \left[ A_1 \text{ true} \right] \).

\[
\begin{align*}
&f \left[ (A_2 \supset A_3) \text{ true} \right] \in \Gamma_{f gx} \text{ sc-assum} & g \left[ (A_1 \supset A_2) \text{ true} \right] \in \Gamma_{f gx} \text{ sc-assum} & x \left[ A_1 \text{ true} \right] \in \Gamma_{f gx} \text{ sc-assum} \\
&\Gamma_{f gx} \vdash A_2 \supset A_3 & \Gamma_{f gx} \vdash A_1 \supset A_2 \text{ true} & \Gamma_{f gx} \vdash A_1 \text{ true} & \Gamma_{f gx} \vdash A_2 \text{ true} \text{ sc-\&Elim} & \text{ sc-\&Elim} \\
&\Gamma_{f gx} \vdash A_3 \text{ true sc-Intro} & \Gamma_{f gx} \vdash (A_1 \supset A_2) \supset (A_1 \supset A_3) \text{ true} \text{ sc-Intro} & \Gamma_{f gx} \vdash (A_1 \supset A_2) \supset (A_1 \supset A_3) \text{ true} \text{ sc-Intro} & \Gamma_{f gx} \vdash (A_1 \supset A_2) \supset (A_1 \supset A_3) \text{ true} \text{ sc-Intro} \\
\end{align*}
\]
§2  Mirror World

Question 2(c)+.

A single &-elimination: We have two elimination rules for &, which separately extract a sub-formula. 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 \(\emptyset \vdash P_1 \supset (P_2 \supset Q)\): 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:

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

A single \(\times\)-elimination rule: 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 \(\Gamma\):

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

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:

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

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:

\[
\frac{\Gamma \vdash e_0 : (S_1 \times S_2) \quad \Gamma, x : S_1, y : S_2 \vdash e_{12} : T}{\Gamma \vdash (\text{Split } e_{0} x y e_{12}) : T} \quad \text{type-}&\text{Elim-new}
\]

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

\[
(Split \ (Pair \ v_1 \ v_2) \ x_1 \ x_2 \ e_{12}) \rightarrow_{R} [v_1/x_1][v_2/x_2]e_{12}
\]

\[
C : = \ldots | (\text{Split } C \times x \times e)
\]
**Question 3(a).** Fill in the four listed cases.

Consider cases of the rule concluding $\emptyset \vdash v : (S_1 \times S_2)$. [Either explain why the case is impossible, even if you are only repeating what I gave for Part 1, or show the goal for Part 4 as stated by the lemma.]

- **type-assum:**
  
  In type-assum, we have $x$ (which is a value) but the context is $\emptyset$, so the premise is $(x : S_1 \times S_2) \in \emptyset$ which is impossible.

- **$\to$Intro:**
  
  By inversion on $\to$Intro, $(S_1 \times S_2) = (T_1 \to T_2)$ which is impossible.

- **$\times$Elim:** [There is no rule with this name! I should have listed $\times$Elim1 and $\times$Elim2.]
  
  By inversion on $\times$Elim1, $v = \text{proj}_1 e$ which is impossible: a Proj is never a value, according to the grammar of values.
  
  The $\times$Elim2 case is similar.

- **$\times$Intro:**
  
  [This is the only possible case. There is no need to explain why the case is possible: our “starting position” is that every case is possible. If a case is impossible, explain why; otherwise, do the case, which means: “show the goal for Part 4 as stated by the lemma”.]
  
  [Our goal is the goal for Part 4, namely: “there exist $v_1$ and $v_2$ such that $v = (\text{Pair } v_1 v_2)$ and $\emptyset \vdash v_1 : S_1$ and $\emptyset \vdash v_2 : S_2$.”]
  
  $v = (\text{Pair } v_1 v_2) \quad \text{By inversion on rule } \times\text{Intro}$
  
  $\emptyset \vdash v_1 : S_1 \quad "$
  
  $\emptyset \vdash v_2 : S_2 \quad "$
  
  [These are exactly the three goals we need, so we’re done! A valid but unnecessarily long proof would be:]
  
  $v = (\text{Pair } e_1 e_2) \quad \text{By inversion on rule } \times\text{Intro}$
  
  $\emptyset \vdash v_1 : S_1 \quad "$
  
  $\emptyset \vdash v_2 : S_2 \quad "$
  
  $e_1$ is a value, that is, $e_1 = v_1$ By the grammar of values
  
  $e_2$ is a value, that is, $e_2 = v_2$ By the grammar of values
  
  $v = (\text{Pair } v_1 v_2) \quad \text{By above equations}$

- The remaining cases are impossible for reasons similar to those given in Part 1.

**Part 5:** **Question 3(b).** Complete the missing case.

- **$\to$Intro:**
  
  $v = (\text{Lam } x e) \quad \text{By inversion on rule } \to\text{Intro}$
  
  $x : S_1 \vdash e : S_2 \quad "$
  
  [The $S_1$ in $(\text{Lam } x S_1 e)$ was a mistake; it didn’t match the grammar or the rules.]
  
  [These are exactly the two things we need for Part 5, so we’re done.]
§2 Mirror World

- The remaining cases are impossible for reasons similar to those given in Part 1.