CS 320
Computer Language Processing
Exercise Set 4

March 26, 2025

Exercise 1 For each of the following pairs of grammars, show that they are equivalent by identifying them with inductive relations, and proving that the inductive relations contain the same elements.

1. $A_1 : S ::= S + S \mid \textbf{num}$
   $A_2 : R ::= \textbf{num}\ R'$ and $R' ::= +R\ R' \mid \epsilon$
2. $B_1 : S ::= S(S)S \mid \epsilon$
   $B_2 : R ::= RR \mid (R) \mid \epsilon$

Solution

1. A_2 is the result of left-recursion elimination on A_1. First, expressing them as inductive relations, with rules named as on the right:

$$\frac{}{\textbf{num} \in S}\ S_{num} \qquad \frac{w_1 \in S \qquad w_2 \in S}{w_1 + w_2 \in S}\ S_+$$

$$\frac{w \in S}{w \in A_1}\ A_1^{start}$$

$$\frac{w \in R'}{\textbf{num}\ w \in R}\ R_{num}$$

$$\frac{w \in R \qquad w' \in R'}{+w\ w' \in R'}\ R'_+ \qquad \frac{}{\epsilon \in R'}\ R'_\epsilon$$

$$\frac{w \in R}{w \in A_2}\ A_2^{start}$$

   We must show that for any word w, $w \in A_1$ if and only if $w \in A_2$. For this, it must be the case that there is a derivation tree for $w \in A_1$ (equivalently, $w \in S$) if and only if there is a derivation tree for $w \in A_2$ (equivalently, $w \in R$) according to the inference rules above.

   (a) $w \in S \implies w \in R$: we induct on the depth of the derivation tree.
   - Base case: derivation tree of depth 1. The tree must be

$$\frac{}{\textbf{num} \in S}\ S_{num}$$

We can show that there is a corresponding derivation tree for $w \in R$:

$$\dfrac{\dfrac{}{\epsilon \in R'}\ R'_\epsilon}{\textbf{num} \in R}\ R_{num}$$

- Inductive case: derivation tree of depth $n+1$, given that for every derivation of depth $\leq n$ of $w' \in S$ for any w', there is a corresponding derivation of $w' \in R$. The last rule applied in the derivation must be S_+:

$$\dfrac{\dfrac{\dots}{w_1 \in S} \quad \dfrac{\dots}{w_2 \in S}}{w_1 + w_2 \in S}\ S_+$$

By the inductive hypothesis, since $w_1 \in S$ and $w_2 \in S$ have a derivation tree of smaller depth, there are derivation trees for $w_1 \in R$ and $w_2 \in R$. In particular, the derivation for $w_1 \in R$ must end with the rule R_{num} (only case), so there must be a derivation tree for $\textbf{num}\ w'_1 \in R$ with $w'_1 \in R'$ and $num\ w'_1 = w_1$. We have the following pieces:

$$\dfrac{\dfrac{\dots}{w'_1 \in R'}}{\textbf{num}\ w'_1 \in R}\ R_{num} \qquad \dfrac{\dots}{w_2 \in R}$$

To show that $w_1 + w_2 \in R$, i.e. $\textbf{num}\ w'_1 + w_2 \in R$, we must first show that $w'_1 + w_2 \in R'$, as required by the rule R_{num}. Note that words in R' are of the form $(+\textbf{num})^*$. We will prove this separately for all pairs of words at the end (R'_{Lemma}). Knowing this, however, we can construct the derivation tree for $w_1 + w_2 \in R$:

$$\dfrac{\dfrac{\dfrac{\dots}{w'_1 \in R'} \quad \dfrac{\dots}{w_2 \in R}}{w'_1 + w_2 \in R'}\ R'_{Lemma}}{\textbf{num}\ w'_1 + w_2 \in R}\ R_{num}$$

$\textbf{num}\ w'_1 + w_2 = w_1 + w_2 = w$, as required.
Finally, we will show the required lemma. We will prove a stronger property R'_{concat} first, that for any pair of words $w_1, w_2 \in R'$, $w_1\ w_2 \in R'$ as well. We induct on the derivation of $w_1 \in R'$.
Base case: derivation ends with R'_ϵ. Then $w_1 = \epsilon$, and $w_1\ w_2 = w_2 \in R'$ by assumption.
Inductive case: derivation ends with R'_+. Then $w_1 = +vv'$ for some $v \in R$ and $v' \in R'$:

$$\dfrac{\dfrac{\dots}{v \in R} \quad \dfrac{\dots}{v' \in R'}}{+v\ v' \in R'}\ R'_+$$

Since $v' \in R'$ has a smaller derivation tree than w_1, by the inductive hypothesis, we can prove that $v'\ w_2 \in R'$. We get:

$$\dfrac{\dfrac{\dots}{v \in R} \quad \dfrac{\dfrac{\dots}{v' \in R'} \quad \dfrac{\dots}{w_2 \in R'}}{v'\ w_2 \in R'}\ R'_{concat}}{+v\ v'\ w_2 \in R'}\ R'_+$$

So, R'_{concat} is proven. We can show R'_{lemma}, i.e. $w_1' + w_2 \in R'$ if $w_1' \in R'$ and $w_2 \in R$ as:

$$\dfrac{\dfrac{\dots}{w_1' \in R'} \qquad \dfrac{\dfrac{\dots}{w_2 \in R} \qquad \dfrac{}{\epsilon \in R'}\,R'_\epsilon}{+w_2 \in R'}\,R'_+}{w_1' + w_2 \in R'}\,R'_{concat}$$

Thus, the proof is complete.

(b) $w \in R \implies w \in S$: we induct on the depth of the derivation tree for $w \in R$. This direction is simpler than the other, but the general method is similar.

- Base case: derivation tree of depth 2 (minimum). The tree must be

$$\dfrac{\dfrac{}{\epsilon \in R'}\,R'_\epsilon}{\mathbf{num} \in R}\,R_{num}$$

We have the corresponding derivation tree for $w \in S$:

$$\dfrac{}{\mathbf{num} \in S}\,S_{num}$$

- Inductive case: derivation tree of depth $n+1$, given that for every derivation of depth $\leq n$ of $w' \in R$ for any w', there is a corresponding derivation of $w' \in S$. The last rules applied must be R_{num} and R'_+ (otherwise the derivation would be of the base case):

$$\dfrac{\dfrac{\dfrac{\dots}{w_1 \in R} \qquad \dfrac{\dots}{w_2 \in R'}}{+w_1\ w_2 \in R'}\,R'_+}{\mathbf{num} + w_1\ w_2 \in R}\,R_{num}$$

where $w = \mathbf{num} + w_1\ w_2$. However, we are somewhat stuck here, as we have no way to relate R' and S. We will separately show that if $+w' \in R'$, then there is a derivation of $w' in S$ (lemma R'_S). This will allow us to complete the proof:

$$\dfrac{\dfrac{}{\mathbf{num} \in S}\,S_{num} \qquad \dfrac{\dfrac{\dots}{+w_1\ w_2 \in R'}}{w_1\ w_2 \in S}\,R'_S}{\mathbf{num} + w_1\ w_2 \in S}\,S_+$$

The proof of the lemma R'_S is by induction again, and not shown here. This completes the original proof.

2. Argument similar to Exercise Set 2 Problem 4 (same pair of grammars). $B_1 \subseteq B_2$ as relations can be seen by producing a derivation tree for each possible case in B_1. For the other direction, $B_2 \subseteq B_1$, it is first convenient to prove that B_1 is closed under concatenation, i.e., if $w_1, w_2 \in B_1$ then there is a derivation tree for $w_1\ w_2 \in B_1$.

$\square$

Exercise 2 Consider the following expression language over naturals, and a *halving* operator:

$$expr ::= \text{half}(expr) \mid expr + expr \mid \textbf{num}$$

where **num** is any natural number constant ≥ 0.

We will design the operational semantics of this language. The semantics should define rules that apply to as many expressions as possible, while being subjected to the following safety conditions:

- the semantics should *not* permit halving unless the argument is even
- they should evaluate operands from left-to-right

Of the given rules below, choose a *minimal* set that satisfies the conditions above. A set is *not* minimal if removing any rule does not change the set of expressions that can be evaluated by the semantics, i.e. the domain of $\rightsquigarrow$, $\{x \mid \exists y.\ x \rightsquigarrow y\}$, remains unchanged. The removed rule is said to be *redundant*.

$$\frac{e \rightsquigarrow e'}{\text{half}(e) \rightsquigarrow e'} \tag{A}$$

$$\frac{n \text{ is a value} \qquad n = 2k}{\text{half}(n) \rightsquigarrow k} \tag{B}$$

$$\frac{n \text{ is a value}}{\text{half}(n) \rightsquigarrow \lfloor \frac{n}{2} \rfloor} \tag{C}$$

$$\frac{\text{half}(e) \rightsquigarrow \text{half}(e')}{\text{half}(e) \rightsquigarrow e'} \tag{D}$$

$$\frac{e \rightsquigarrow e'}{\text{half}(e) \rightsquigarrow \text{half}(e')} \tag{E}$$

$$\frac{e' \rightsquigarrow \text{half}(e)}{half(e) \rightsquigarrow e'} \tag{F}$$

$$\frac{n_1 \text{ is a value} \qquad n_2 \text{ is a value} \qquad n_1 + n_2 = k \qquad n_1 \text{ is odd}}{n_1 + n_2 \rightsquigarrow k} \tag{G}$$

$$\frac{e \rightsquigarrow e' \qquad n \text{ is a value}}{n + e \rightsquigarrow n + e'} \tag{H}$$

$$\frac{e_2 \rightsquigarrow e_2'}{e_1 + e_2 \rightsquigarrow e_1 + e_2'} \tag{I}$$

$$\frac{n_1 \text{ is a value} \qquad n_2 \text{ is a value} \qquad n_1 + n_2 = k \qquad n_1, n_2 \text{ are even}}{n_1 + n_2 \rightsquigarrow k} \tag{J}$$

$$\frac{n_1 \text{ is a value} \qquad n_2 \text{ is a value} \qquad n_1 + n_2 = k}{n_1 + n_2 \rightsquigarrow k} \tag{K}$$

$$\frac{e_1 \leadsto e_1'}{e_1 + e_2 \leadsto e_1' + e_2} \tag{L}$$

Solution A possible such minimal set of rules is $\{B, E, H, K, L\}$.

On what happens when the other rules are added to this set:

- A: incorrect; allows deducing half(half(10)) $\leadsto$ 5 with rule B.
- C: incorrect; allows deducing half(3) $\leadsto$ 1.
- D: incorrect; allows deducing half(half(10)) $\leadsto$ 5 with rules B and E.
- F: redundant; reverses a reduction.
- G: redundant; special case of rule K.
- I: incorrect; does not reduce the expression left-to-right.
- J: redundant; special case of rule K.

□

Exercise 3 Consider a simple programming language with integer arithmetic, boolean expressions, and user-defined functions:

$$\begin{aligned} expr ::= &\ true \mid false \mid \mathbf{num} \\ &\ expr == expr \mid expr + expr \\ &\ expr\ \&\&\ expr \mid if\ (expr)\ expr\ else\ expr \\ &\ f(expr, \ldots, expr) \mid x \end{aligned}$$

where f represents a (user-defined) function, x represents a variable, and **num** represents an integer.

1. Inductively define a substitution operation for the terms in this language, which replaces every free occurrence of a variable x with a given expression e.

   The rule for substitution in an addition is provided as an example. Here, $t[x := e]$ represents the term t, with every free occurrence of x simultaneously replaced by e.

   $$\frac{t_1[x := e] \to t_1' \qquad t_2[x := e] \to t_2'}{t_1 + t_2[x := e] \to t_1' + t_2'}$$

2. Write the rules for the operational semantics for this language, assuming *call-by-name* semantics for function calls. In call-by-name semantics, function arguments are not evaluated before the call. Instead, the parameters are merely substituted into the function body. You may assume that function parameters are named distinctly from variables in the program.

3. Under the following environment (with function names, parameters, and bodies):

$$(sum, [x], if\ (x == 0)\ then\ 0\ else\ x + sum(x + (-1)))$$
$$(rec, [\ ], rec())$$
$$(default, [b, x], if\ b\ then\ x\ else\ 0)$$

evaluate each of the following expressions, showing the derivations:

   (a) $sum(2)$
   (b) $if\ (1 == 2)\ then\ 3\ else\ 4$
   (c) $sum(sum(0))$
   (d) $rec()$
   (e) $default(false, rec())$

How would the evaluations in each case change if we used *call-by-value* semantics instead?

Solution

1. Substitution rules:

$$\frac{}{true[x := e] \to true}$$
$$\frac{}{false[x := e] \to false}$$
$$\frac{}{\mathbf{num}[x := e] \to \mathbf{num}}$$
$$\frac{t_1[x := e] \to t_1' \qquad t_2[x := e] \to t_2'}{t_1 == t_2[x := e] \to t_1' == t_2'}$$
$$\frac{t_1[x := e] \to t_1' \qquad t_2[x := e] \to t_2'}{t_1 + t_2[x := e] \to t_1' + t_2'}$$
$$\frac{t_1[x := e] \to t_1' \qquad t_2[x := e] \to t_2'}{t_1\ \&\&\ t_2[x := e] \to t_1'\ \&\&\ t_2'}$$
$$\frac{t_1[x := e] \to t_1' \qquad t_2[x := e] \to t_2' \qquad t_3[x := e] \to t_3'}{if\ (t_1)\ t_2\ else\ t_3[x := e] \to if\ (t_1')\ t_2'\ else\ t_3'}$$
$$\frac{t_1[x := e] \to t_1' \qquad \ldots \qquad t_n[x := e] \to t_3'}{f(t_1, \ldots, t_n)[x := e] \to f(t_1', \ldots, t_n')}$$
$$\frac{}{x[x := e] \to e}$$
$$\frac{x \neq y}{y[x := e] \to y}$$

2. Operational semantics:

   - Equality:

$$\frac{t_1 \rightsquigarrow t_1'}{t_1 == t_2 \rightsquigarrow t_1' == t_2}$$

$$\frac{n_1 \text{ is an integer value} \qquad t_2 \rightsquigarrow t_2'}{n_1 == t_2 \rightsquigarrow n_1 == t_2'}$$

$$\frac{n_1, n_2 \text{ are integer values} \qquad n_1 = n_2}{n_1 == n_2 \rightsquigarrow true}$$

$$\frac{n_1, n_2 \text{ are integer values} \qquad n_1 \neq n_2}{n_1 == n_2 \rightsquigarrow false}$$

- Addition:

$$\frac{t_1 \rightsquigarrow t_1'}{t_1 + t_2 \rightsquigarrow t_1' + t_2}$$

$$\frac{n \text{ is an integer value} \qquad t_2 \rightsquigarrow t_2'}{n + t_2 \rightsquigarrow n + t_2'}$$

$$\frac{n_1, n_2 \text{ are integer values} \qquad n_1 + n_2 = k}{n_1 + n_2 \rightsquigarrow k}$$

- Conjunction:

$$\frac{t_1 \rightsquigarrow t_1'}{t_1 \ \&\& \ t_2 \rightsquigarrow t_1' \ \&\& \ t_2}$$

$$\frac{}{true \ \&\& \ t \rightsquigarrow t}$$

$$\frac{}{false \ \&\& \ t \rightsquigarrow false}$$

- Conditionals:

$$\frac{t_1 \rightsquigarrow t_1'}{if \ (t_1) \ t_2 \ else \ t_3 \rightsquigarrow if \ (t_1') \ t_2 \ else \ t_3}$$

$$\frac{}{if \ (true) \ t_2 \ else \ t_3 \rightsquigarrow t_2}$$

$$\frac{}{if \ (false) \ t_2 \ else \ t_3 \rightsquigarrow t_3}$$

- Function call:

$$\frac{\begin{array}{c} b_0 \text{ is the body of } f \\ (x_1, \ldots, x_n) \text{ are parameters of } f \\ b_0[x_1 := t_1] \to b_1 \quad \ldots \quad b_{n-1}[x_n := t_n] \to b_n \end{array}}{f(t_1, \ldots, t_n) \rightsquigarrow b_n}$$

3. Evaluations:

   (a) $sum(2) \rightsquigarrow 3$:

$$\begin{array}{l} sum(2) \\ \rightsquigarrow if \ (2 == 0) \ then \ 0 \ else \ 2 + sum(2 + (-1)) \\ \rightsquigarrow if \ (false) \ then \ 0 \ else \ 2 + sum(2 + (-1)) \\ \rightsquigarrow 2 + sum(2 + (-1)) \\ \rightsquigarrow 2 + if \ ((2 + (-1)) == 0) \ then \ 0 \ else \ 2 + (-1) + sum(2 + (-1) + (-1)) \\ \rightsquigarrow 2 + if \ (1 == 0) \ then \ 0 \ else \ 2 + (-1) + sum(2 + (-1) + (-1)) \end{array}$$

$$\begin{aligned}
&\rightsquigarrow 2 + if\ (false)\ then\ 0\ else\ 2 + (-1) + sum(2 + (-1) + (-1))\\
&\rightsquigarrow 2 + (2 + (-1) + sum(2 + (-1) + (-1)))\\
&\ldots\\
&\rightsquigarrow 2 + (1 + 0)\\
&\rightsquigarrow 2 + 1\\
&\rightsquigarrow 3
\end{aligned}$$

(b) $sum(sum(0)) \rightsquigarrow 0$:

$$\begin{aligned}
&sum(sum(0))\\
&\rightsquigarrow if\ (sum(0) == 0)\ then\ 0\ else\ 0 + sum(sum(0) + (-1))\\
&\rightsquigarrow \ldots\ \textit{(expand } sum(0) \textit{ in the conditional)}\\
&\rightsquigarrow if\ (0 == 0)\ then\ 0\ else\ 0 + sum(sum(0) + (-1))\\
&\rightsquigarrow if\ (true)\ then\ 0\ else\ 0 + sum(sum(0) + (-1))\\
&\rightsquigarrow 0
\end{aligned}$$

(c) $if\ (1 == 2)\ then\ 3\ else\ 4 \rightsquigarrow 4$.

(d) $rec() \rightsquigarrow rec()$ (infinite loop).

(e) $default(false, rec()) \rightsquigarrow 0$.

Under call-by-value-semantics, the structure of the evaluations would be different. In $sum(sum(0))$, we would evaluate the inner $sum(0)$ to 0 before evaluating the outer $sum(\cdot)$. In $default(false, rec())$, we would need to evaluate $rec()$, which would lead to an infinite loop.

□

Exercise 4 Consider the following type system for a language with integers, conditionals, pairs, and functions:

$$\frac{n \text{ is an integer literal}}{\Gamma \vdash n : \texttt{Int}} \qquad \frac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau}$$

$$\frac{\Gamma \vdash e_1 : \texttt{Int} \qquad \Gamma \vdash e_2 : \texttt{Int}}{\Gamma \vdash e_1 + e_2 : \texttt{Int}}$$

$$\frac{\Gamma \vdash e_1 : \texttt{Int} \qquad \Gamma \vdash e_2 : \texttt{Int}}{\Gamma \vdash e_1 \times e_2 : \texttt{Int}}$$

$$\frac{b \text{ is a boolean literal}}{\Gamma \vdash b : \texttt{Bool}} \qquad \frac{\Gamma \vdash e : \texttt{Bool}}{\Gamma \vdash \texttt{not e} : \texttt{Bool}}$$

$$\frac{\Gamma \vdash e_1 : \texttt{Bool} \qquad \Gamma \vdash e_2 : \texttt{Bool}}{\Gamma \vdash e_1 \wedge e_2 : \texttt{Bool}} \qquad \frac{\Gamma \vdash e_1 : \texttt{Bool} \qquad \Gamma \vdash e_2 : \texttt{Bool}}{\Gamma \vdash e_1 \vee e_2 : \texttt{Bool}}$$

$$\frac{\Gamma \vdash e_1 : \texttt{Bool} \qquad \Gamma \vdash e_2 : \tau \qquad \Gamma \vdash e_3 : \tau}{\Gamma \vdash if\ e_1\ then\ e_2\ else\ e_3 : \tau}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash (e_1, e_2) : (\tau_1, \tau_2)}$$

$$\frac{\Gamma \vdash e : (\tau_1, \tau_2)}{\Gamma \vdash fst(e) : \tau_1} \quad \frac{\Gamma \vdash e : (\tau_1, \tau_2)}{\Gamma \vdash snd(e) : \tau_2}$$

$$\frac{\Gamma \oplus \{x : \tau_1\} \vdash e : \tau_2}{\Gamma \vdash x \Rightarrow e : \tau_1 \to \tau_2} \quad \frac{\Gamma \vdash e_1 : \tau_1 \to \tau_2 \qquad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash e_1 e_2 : \tau_2}$$

1. Given the following type derivation with type variables $\tau_1, \ldots, \tau_5$, choose the correct options:

$$\dfrac{\dfrac{\dfrac{\dfrac{(x, \tau_4) \in \Gamma}{\Gamma \vdash x : \tau_4}}{\Gamma \vdash fst(x) : \tau_3} \quad \dfrac{\dfrac{(x, \tau_4) \in \Gamma}{\Gamma \vdash x : \tau_4}}{\Gamma \vdash snd(x) : \tau_5}}{\Gamma \vdash fst(x)(snd(x)) : \tau_2}}{\Gamma' \vdash x \Rightarrow fst(x)(snd(x)) : \tau_1}$$

   (a) There are no valid assignments to the type variables such that the above derivation is valid.

   (b) In all valid derivations, $\tau_2 = \tau_5$.

   (c) There are *no* valid derivations where $\tau_2 = \texttt{Int}$.

   (d) In all valid derivations, $\tau_4 = (\tau_3, \tau_5)$.

   (e) In all valid derivations, $\tau_1 = \tau_4 \rightarrow \tau_2$.

   (f) There is a valid derivation where $\tau_1 = \tau_2$.

2. For each of the following pairs of terms and types, provide a valid type derivation or briefly argue why the typing is incorrect:

   (a) $x \Rightarrow x + 5$: $\texttt{Int} \rightarrow \texttt{Int}$

   (b) $x \Rightarrow y \Rightarrow x + y$: $\texttt{Int} \rightarrow \texttt{Int} \rightarrow \texttt{Int}$

   (c) $x \Rightarrow y \Rightarrow y(2) \times x$: $\texttt{Int} \rightarrow \texttt{Int} \rightarrow \texttt{Int}$

   (d) $x \Rightarrow (x, x)$: $\texttt{Int} \rightarrow \texttt{(Int, Int)}$

   (e) $x \Rightarrow y \Rightarrow if\ fst(x)\ then\ snd(x)\ else\ y$: $\texttt{(Bool, Int)} \rightarrow \texttt{(Int, Int)} \rightarrow \texttt{Int}$

   (f) $x \Rightarrow y \Rightarrow if\ y\ then\ (z \Rightarrow y)\ else\ x$: $(\texttt{Bool} \rightarrow \texttt{Bool}) \rightarrow \texttt{Bool} \rightarrow (\texttt{Bool} \rightarrow \texttt{Bool})$

   (g) $x \Rightarrow y \Rightarrow if\ y\ then\ (z \Rightarrow y)\ else\ x$: $(\texttt{Int} \rightarrow \texttt{Bool}) \rightarrow \texttt{Bool} \rightarrow (\texttt{Int} \rightarrow \texttt{Bool})$

3. Prove that there is *no* valid type derivation for the term

$$x \Rightarrow if\ fst(x)\ then\ snd(x)\ else\ x$$

Solution

1. The correct statements are d and e. For the remaining:

   - **a**: set $\tau_2 = \texttt{Int}$, $\tau_5 = \texttt{Bool}$, $\tau_3 = \tau_5 \rightarrow \tau_2$, $\tau_4 = (\tau_3, \tau_5)$, and $\tau_1 = \tau_4 \rightarrow \tau_2$.
   - **b**: see (a).
   - **c**: see (a).
   - **f**: given $\tau_1 = \tau_2$, we also know from the rule for lambda abstraction that $\tau_1 = \tau_4 \rightarrow \tau_2$, and hence $\tau_2 = \tau_4 \rightarrow \tau_2$ recursively, which is a contradiction.

2. For the given terms and types:

   (a) $x \Rightarrow x + 5$: $\texttt{Int} \to \texttt{Int}$: ✓

$$\dfrac{\dfrac{\dfrac{x : \texttt{Int} \in \Gamma}{\Gamma \vdash x : \texttt{Int}} \qquad \dfrac{}{\Gamma \vdash 5 : \texttt{Int}}}{\Gamma \vdash x + 5 : \texttt{Int}}}{\Gamma' \vdash x \Rightarrow x + 5 : \texttt{Int} \to \texttt{Int}}$$

   (b) $x \Rightarrow y \Rightarrow x + y$: $\texttt{Int} \to \texttt{Int} \to \texttt{Int}$: ✓

   (c) $x \Rightarrow y \Rightarrow y(2) \times x$: $\texttt{Int} \to \texttt{Int} \to \texttt{Int}$: ✗. If y has type $\texttt{Int}$, then $y(2)$ cannot not well-typed, as the function application rule is not applicable.

   (d) $x \Rightarrow (x, x)$: $\texttt{Int} \to \texttt{(Int, Int)}$: ✓

   (e) $x \Rightarrow y \Rightarrow if\ fst(x)\ then\ snd(x)\ else\ y$: $\texttt{(Bool, Int)} \to \texttt{(Int, Int)} \to \texttt{Int}$: ✗. The type of the two branches of a conditional must match, but here they are $\texttt{Int}$ and $(\texttt{Int}, \texttt{Int})$ respectively.

   (f) $x \Rightarrow y \Rightarrow if\ y\ then\ (z \Rightarrow y)\ else\ x$: $(\texttt{Bool} \to \texttt{Bool}) \to \texttt{Bool} \to (\texttt{Bool} \to \texttt{Bool})$: ✓

$$\dfrac{\dfrac{\dfrac{\dfrac{(y, \texttt{Bool}) \in \Gamma}{\Gamma \vdash y : \texttt{Bool}} \qquad \dfrac{(x, \texttt{Bool} \to \texttt{Bool}) \in \Gamma}{\Gamma \vdash x : \texttt{Bool} \to \texttt{Bool}} \qquad \dfrac{\dfrac{(y, \texttt{Bool}) \in \Gamma \oplus \{(z, \texttt{Bool})\}}{\Gamma \oplus \{(z, \texttt{Bool})\} \vdash y : \texttt{Bool}}}{\Gamma \vdash z \Rightarrow y : \texttt{Bool} \to \texttt{Bool}}}{\Gamma \vdash if\ y\ then\ (z \Rightarrow y)\ else\ x : \texttt{Bool} \to \texttt{Bool}}}{\Gamma' \vdash y \Rightarrow if\ y\ then\ (z \Rightarrow y)\ else\ x : (\texttt{Bool} \to \texttt{Bool}) \to \texttt{Bool} \to (\texttt{Bool} \to \texttt{Bool})}}{\Gamma'' \vdash x \Rightarrow y \Rightarrow if\ y\ then\ (z \Rightarrow y)\ else\ x : (\texttt{Bool} \to \texttt{Bool}) \to \texttt{Bool} \to (\texttt{Bool} \to \texttt{Bool})}$$

   Note that the choice of type of z (and of the argument of x) is arbitrary. Hence, the next typing is also valid.

   (g) $x \Rightarrow y \Rightarrow if\ y\ then\ (z \Rightarrow y)\ else\ x$: $(\texttt{Int} \to \texttt{Bool}) \to \texttt{Bool} \to (\texttt{Int} \to \texttt{Bool})$: ✓

3. Non-existence of a valid type derivation for the term:

$$t = x \Rightarrow if\ fst(x)\ then\ snd(x)\ else\ x$$

   Assume that there is a valid type derivation for the term. We will attempt to derive a contradiction. We use the fact that if there exists a type derivation, every step must use one of the rules above, and that the types assigned to each variable must be consistent across the derivation.

   First, t has a type derivation *if and only if* $t_1 = if\ fst(x)\ then\ snd(x)\ else\ x$ has a type derivation, by using the function abstraction rule. We will work with t_1 directly. The function abstraction rule here does not give us more information.

   Any type derivation for t_1 must end in the conditional rule. For this rule to be applicable, we must have that the following are derivable:

   (a) $\Gamma \vdash fst(x) : \texttt{Bool}$

   (b) $\Gamma \vdash snd(x) : \tau$

(c) $\Gamma \vdash x : \tau$

where the type variable τ is also the type of t_1.

By using the projection rule on (a) and (b), we learn that the type of x must be $(\texttt{Bool}, \tau_1)$ and (τ_2, τ) for two fresh variables τ_1 and τ_2 respectively. Matching the two, as x may only have one type, we must have $\tau_1 = \tau$, $\tau_2 = \texttt{Bool}$, and thus the type of x is $(\texttt{Bool}, \tau)$.

However, from (c), we learn that the type of x is τ. It must be the case that $\tau = (\texttt{Bool}, \tau)$. This is not possible for any type τ, and we have a contradiction.

Hence, there is no valid type derivation for the term t.

$\square$