1 Language Extension

Consider the following language, defined by a grammar and a big-step evaluation judgment. The
big-step evaluation given is incomplete, in the informal sense that it has no rules saying how to
evaluate \((\text{Square } e)\).

\[
\begin{align*}
\text{integers} & \quad n \\
\text{values} & \quad v ::= n \\
\text{expressions} & \quad e ::= n \quad \mid (+ e_1 e_2) \quad \mid (\text{Square } e)
\end{align*}
\]

\[\text{expression } e \text{ evaluates to value } v\]

\[
\begin{align*}
&\quad n \Downarrow n \quad \text{eval-const} \quad e_1 \Downarrow n_1 \quad e_2 \Downarrow n_2 \\ &\quad (+ e_1 e_2) \Downarrow n_1 + n_2 \quad \text{eval-add}
\end{align*}
\]

Question 1(a). Roughly following the structure of eval-add, design a rule “eval-sq” such that
\((\text{Square } e)\) will compute the square of \(e\). Similar to how we used the standard notation for addition,
\(n_1 + n_2\), in eval-add, you may use the notation \(n^2\) for the square of \(n\).

\[
\begin{align*}
&\quad e \Downarrow n \\
&\quad (\text{Square } e) \Downarrow n^2 \quad \text{eval-sq}
\end{align*}
\]

2 Proof techniques

These questions are not about complete proofs. In some of the questions, the conjecture is not even
true, or you have not been given enough information to do a complete proof. Instead, they ask you
to make progress on several different proofs by using a specific proof technique.

In all of these questions, the grammar of expressions is the extended grammar (Section 1), and
the system of rules deriving \(e \Downarrow v\) includes the three rules eval-const, eval-add, eval-sq.

Question 2(a). Using the extended grammar of expressions (Section 1), list the cases produced
by case analysis on \(e\). The cases must correspond to the grammar. Do not attempt to complete the
cases to show \(e' = e''\).

Conjecture.

For all expressions \(e, e'\) and \(e''\),

if \(e \Rightarrow e'\) and \(e \Rightarrow e''\)

then \(e' = e''\).

Proof. Consider cases of \(e\).
§1 Language Extension

- Case $e = n$
- Case $e = (+ e_1 e_2)$
- Case $e = (\text{Square } e_1)$

[Writing $e = (\text{Square } e)$ doesn’t make sense: it’s like saying that $x + 1 = x$. The name $e$ can’t be used for two different things at the same time. We need to rename the $e$ from the grammar so it doesn’t clash with the $e$ mentioned in the conjecture. □

**Question 2(b).** In this question, your goal is to derive

$$(+ 3 (+ e_{21} e_{22})) \Downarrow 3$$

The following are given. Use equations (and the fact that $0 + 0 = 0$) and apply the rules eval-const, eval-add to derive the goal.

<table>
<thead>
<tr>
<th>Equation</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$e_{21} \Downarrow n_{21}$</td>
<td>Given</td>
</tr>
<tr>
<td>$e_{22} \Downarrow n_{22}$</td>
<td>Given</td>
</tr>
<tr>
<td>$n_{21} = 0$</td>
<td>Given</td>
</tr>
<tr>
<td>$n_{22} = 0$</td>
<td>Given</td>
</tr>
<tr>
<td>$3 \Downarrow 3$</td>
<td>By eval-const</td>
</tr>
<tr>
<td>$(+ e_{21} e_{22}) \Downarrow n_{21} + n_{22}$</td>
<td>By eval-add</td>
</tr>
<tr>
<td>$(+ 3 (+ e_{21} e_{22})) \Downarrow 3 + (n_{21} + n_{22})$</td>
<td>By eval-add</td>
</tr>
<tr>
<td>$n_{21} = 0$</td>
<td>Above</td>
</tr>
<tr>
<td>$n_{22} = 0$</td>
<td>Above</td>
</tr>
<tr>
<td>$3 + (n_{21} + n_{22}) = 3$</td>
<td>By arithmetic</td>
</tr>
<tr>
<td>$(+ 3 (+ e_{21} e_{22})) \Downarrow 3$</td>
<td>By above equation $[3 + (n_{21} + n_{22}) = 3]$</td>
</tr>
</tbody>
</table>
§2 Proof techniques

**Question 2(c).** In this question, use inversion: write down all the facts given by inverting on rule eval-sq. (I can’t give you a specific goal because what you get depends on your rule, eval-sq.)

\[(\text{Square } e_1) \downarrow n_1 \quad \text{Given} \]

\[n_1 = n^2 \quad \text{By inversion on rule eval-sq} \]

**Question 2(d).**

Conjecture.
For all $C$, $M_1$, $M_2$ and $D_1$ such that $D_1$ derives $M_1 \rightarrow M_2$,
there exists $D_2$ such that $D_2$ derives $C[M_1] \rightarrow C[M_2]$.

- Suppose that I suggest you use structural induction on $D_1$. Write the appropriate induction hypothesis:
  
  For all $C'$, $M'_1$, $M'_2$ and $D'_1$ such that $D'_1 \prec D_1$ and $D'_1$ derives $M'_1 \rightarrow M'_2$,
  there exists $D'_2$ such that $D'_2$ derives $C'[M'_1] \rightarrow C'[M'_2]$.

- Suppose that I suggest you use structural induction on $M$. Write the appropriate induction hypothesis (using the subexpression ordering $\prec$ for $M$):
  
  For all $C'$, $M'_1$, $M'_2$ and $D'_1$ such that $M'_1 \prec M$ and $D'_1$ derives $M'_1 \rightarrow M'_2$,
  there exists $D'_2$ such that $D'_2$ derives $C'[M'_1] \rightarrow C'[M'_2]$.

[The derivation names don’t actually matter here, so they could safely be omitted:

For all $C'$, $M'_1$ and $M'_2$ such that $M'_1 \prec M$ and $M'_1 \rightarrow M'_2$,
$C'[M'_1] \rightarrow C'[M'_2]$.]
§2 Proof techniques

3 Typing

types $\Lambda ::= \text{int}$

$[e : \Lambda]$ expression $e$ has type $\Lambda$

\[
\begin{align*}
\text{type-const} & : n : \text{int} \\
\text{type-sq} & : (\text{Square } e) : \text{int} \\
\text{type-add} & : (e_1 + e_2) : \text{int}
\end{align*}
\]

This is not a terribly interesting type system: every possible expression has the same type, int. Prove the following conjecture:

**Conjecture 3.1.**

*For all expressions $e$, it is the case that $e : \text{int}$.*

**Proof.** By structural induction on $e$. [The only thing given is $e$; we are trying to construct the derivation of $e : \text{int}$, but it doesn't exist yet. So we have to induct on $e$.]

**Induction hypothesis:** For all expressions $e'$ such that $e' \prec e$, it is the case that $e' : \text{int}$.

**Consider cases of $e$.**

- **Case: $e = n$**

  \[
  \begin{align*}
  n & : \text{int} \quad \text{By applying rule type-const} \\
  e & : \text{int} \quad \text{By above equation [$e = n$]}
  \end{align*}
  \]

  [I wrote “By applying” to make it more clear that this is not inversion. We can use inversion when we already have a derivation. Here, we have no derivation; we are trying to build a derivation (of $e : \text{int}$).]

- **Case: $e = (\text{Square } e_1)$**

  \[
  \begin{align*}
  e_1 & \prec e \quad \text{[By above equation $e = (\text{Square } e_1)$]} \\
  e_1 & : \text{int} \quad \text{By IH [with $e_1$ as $e'$]} \\
  (\text{Square } e_1) & : \text{int} \quad \text{By applying rule type-sq} \\
  e & : \text{int} \quad \text{By above equation [$e = (\text{Square } e_1)$]}
  \end{align*}
  \]

- **Case: $e = (+ e_1 e_2)$**

  \[
  \begin{align*}
  e_1 & \prec e \quad \text{[By above equation $e = (+ e_1 e_2)$]} \\
  e_1 & : \text{int} \quad \text{By IH [with $e_1$ as $e'$]} \\
  e_2 & \prec e \quad \text{[By above equation $e = (+ e_1 e_2)$]} \\
  e_2 & : \text{int} \quad \text{By IH [with $e_2$ as $e'$]} \\
  (+ e_1 e_2) & : \text{int} \quad \text{By applying rule type-add} \\
  e & : \text{int} \quad \text{By above equation [$e = (+ e_1 e_2)$]}
  \end{align*}
  \]

\[\square\]
§3 Typing

Part 1(a). Derive \((P \land Q) \supset (P \lor Q)\) true.

\[
\begin{align*}
  x &\quad [P \land Q \text{ true}] \\
  (P \land Q) &\quad \text{true} &\quad x \\
  P &\quad \text{true} &\quad \land \text{Elim1} \\
  (P \lor Q) &\quad \text{true} &\quad \lor \text{Intro1} \\
  (P \land Q) &\quad \supset (P \lor Q) \quad \supset \text{Intro}
\end{align*}
\]

Alternate solution: use \(\land \text{Elim2}\) to derive \(Q \text{ true}\), with \(\lor \text{Intro2}\) instead of \(\lor \text{Intro1}\) to get \((P \lor Q) \text{ true}\).

Larger solution: derive \((P \lor Q) \text{ true}\) as above, and use it as the first premise in \(\lor \text{Elim}\), with \(P \lor Q\) as \(C\). But this doesn’t buy you anything.

The question asked for a derivation of \((P \land Q) \supset (P \lor Q)\) true. You didn’t need to derive \(P \land Q\) separately, which is good, because there are no rules enabling you to do that. A derivation of an implication is like a function: think of it as taking a proof of \(P \land Q\) as an argument, and returning a proof of \(P \lor Q\). Building the proof of \(P \land Q\) is somebody else’s problem. For example, if you’re writing a method that takes a \(\text{Canvas}\) object and returns an integer, it’s the caller’s problem to construct the \(\text{Canvas}\); you only have to worry about returning the integer. (In the setting of implementing an entire graphics library, someone—perhaps you—will have to figure out how to construct a \(\text{Canvas}\), but that’s not relevant to that particular method.)

Part 1(b). Gentzen mentioned a symbol \(\supset \subset\), meaning “if and only if”. But, observing that instead of \(A \supset \subset B\) we can use \((A \supset B) \land (B \supset A)\), he didn’t include rules for \(\supset \subset\).

Design introduction and elimination rules for \(\supset \subset\). Hint: Think about the shape of derivations involving the formula \((A \supset B) \land (B \supset A)\).

Many possible solutions...

\[
\begin{align*}
  (A \supset B) &\quad \text{true} &\quad (B \supset A) &\quad \text{true} &\quad \supset \subset \text{Intro} &\quad (A \supset \subset B) &\quad \text{true} \\
  (A \supset B) &\quad \text{true} &\quad (A \supset \subset B) &\quad \text{true} &\quad \supset \subset \text{Elim1} &\quad (B \supset A) &\quad \text{true} \\
  (A \supset \subset B) &\quad \text{true} &\quad (A \supset B) &\quad \text{true} &\quad \supset \subset \text{Elim2} &\quad (B \supset A) &\quad \text{true}
\end{align*}
\]

These rules are not orthogonal, because they mention a connective \(\supset\) that is not the principal connective mentioned in the names of the rules. That is, the above rules defining \(\supset \subset\) depend on the presence of \(\supset\). We may find it implausible that a logic could exclude implication, but for maximum flexibility, we can avoid this dependence:

\[
\begin{align*}
  x &\quad [A \text{ true}] \\
  y &\quad [B \text{ true}] \\
  \vdots &\quad \vdots \\
  B &\quad \text{true} &\quad A &\quad \text{true} &\quad \supset \subset \text{Intro} &\quad (A \supset \subset B) &\quad \text{true} \\
  (A \supset \subset B) &\quad \text{true} &\quad A &\quad \text{true} &\quad \supset \subset \text{Elim1} &\quad (B \supset A) &\quad \text{true} \\
  (A \supset \subset B) &\quad \text{true} &\quad B &\quad \text{true} &\quad \supset \subset \text{Elim2} &\quad A &\quad \text{true}
\end{align*}
\]
Part 1(c). Complete the following derivation of \( (\neg P) \supset (P \supset \text{False}) \) true.

\[
\begin{align*}
&\ x [\neg P \text{ true}] \\
&\ y [P \text{ true}] \\
\end{align*}
\]

\[
\begin{array}{c}
P \text{ true} \\
\hline
\neg P \text{ true} \\
\text{false true} \\
\hline
(P \supset \text{False}) \text{ true} \\
\hline
(\neg P) \supset (P \supset \text{False}) \text{ true} \\
\end{array}
\]

\(\neg\text{Elim}\)

\(\supset\text{Intro}^y\)

\(\supset\text{Intro}^x\)

---

Part 1(d). Derive \( (P \supset \text{False}) \supset (\neg P) \) true.

\[
\begin{align*}
&\ x [(P \supset \text{False}) \text{ true}] \\
&\ y [P \text{ true}] \\
\end{align*}
\]

\[
\begin{array}{c}
(P \supset \text{False}) \text{ true} \\
\hline
\neg P \text{ true} \\
\text{false true} \\
\hline
(\neg P) \text{ true} \\
\hline
(P \supset \text{False}) \supset (\neg P) \text{ true} \\
\end{array}
\]

\(\supset\text{Intro}^y\)

\(\neg\text{Intro}^y\)

\(\supset\text{Intro}^x\)