Assignment 2

J. Dunfield

due: Friday, 2020–02–14

If you wish, you may work in a group of 2 on this assignment.

ID number(s):

Name(s) (optional):

Estimated time spent:
(do not affect your mark)

Please fill in the “Estimated time spent” (above). This helps me to design reasonable assignments.

Late policy

Assignments may be submitted up to 48 hours late with a 20% penalty (that is, I multiply your mark by 0.8).

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 \((\text{Square } e)\).

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

\[
\begin{array}{c}
\text{Expression } e \text{ evaluates to value } v \\
\hline
n \Downarrow n \\
\text{eval-const} & e \Downarrow n_1 & e_2 \Downarrow n_2 \\
\text{eval-add} & (e_1 + e_2) \Downarrow n_1 + n_2
\end{array}
\]

**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 mathematical notation for addition, \(n_1 + n_2\), in eval-add, you may use the notation \(n^2\) for the square of \(n\).
§1 Language extension

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 \( e \).

- Case

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.

\[
\begin{align*}
e_{21} & \downarrow n_{21} & \text{Given} \\
e_{22} & \downarrow n_{22} & \text{Given} \\
n_{21} & = 0 & \text{Given} \\
n_{22} & = 0 & \text{Given}
\end{align*}
\]
\section*{\textbf{Proof techniques}}

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

\begin{equation*}
(Square\ e_1) \Downarrow n_1 \quad \text{Given}
\end{equation*}

\textbf{Question 2(d).}

\textbf{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:

- Suppose that I suggest you use structural induction on $M_1$. Write the appropriate induction hypothesis (using the subexpression ordering $\prec$ for $M_1$):
§2 Proof techniques

3 Typing

\[ A ::= \text{int} \]

expression \( e \) has type \( A \)

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

This is not a terribly interesting type system: every possible expression has the same type, \text{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 \( e' : \text{int} \).
A simplified system of natural deduction, omitting both kinds of quantification, is shown in Figure 1. This is the starting point for this question.

Figure 1  Natural deduction, without quantifiers

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

**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 $(A \supset B) \land (B \supset A)$.

**Part 1(c).** Complete the following derivation of $\neg P \supset (P \supset \text{False})$ true.

$x \neg P \text{ true}$

$y P \text{ true}$

\[
\ldots \quad \vdash (P \supset \text{False}) \text{ true} \quad \supset \text{Intro}^y
\]

\[
\ldots \quad \vdash (\neg P) \supset (P \supset \text{False}) \text{ true} \quad \supset \text{Intro}^x
\]

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