Computation Exercises 1: Expressions

1. Consider the **big-step** operational semantics for the language *SimpleExp* given in the lectures. Find a number n such that

$$(4+1)+(2+2) \Downarrow n.$$

   Give the full derivation tree.

2. The big-step operational semantics for *SimpleExp* was only given for addition. Extend it to include *multiplication*. Give a proof that $((3+2)\times(1+4)) \Downarrow 25$. [In this context, 'give a proof' means 'give the full derivation tree'.]

3. Extend the **big-step** semantics further to include *subtraction*. Remember that the numbers in the syntax of the language are $0, 1, 2, \ldots$: that is to say, there are no negative numbers.

   How is an expression such as $(3-7)$ handled in your semantics? Have you made any arbitrary decisions about this? If so, what other options were available?

For the following questions, ignore the multiplication and subtraction extensions.

4. Recall the **small-step** operational semantics of *SimpleExp*.
   (a) Give the full derivation of the first step of evaluation of $((1+2)+(4+3))$. In other words, give the derivation tree of the step

$$((1+2)+(4+3)) \to E$$

   for some expression E.
   (b) Write down all the steps of evaluation needed to reduce the above expression to 10. Give the full derivation for each of these steps.

5. Here is the abstract syntax for a simple language *Bool* of boolean expressions:

$$\begin{array}{rcl} B \in \mathit{Bool} & ::= & \texttt{true} \mid \texttt{false} \mid B \,\&\, B \mid \\ & & \neg B \mid \texttt{if}\ B\ \texttt{then}\ B\ \texttt{else}\ B \end{array}$$

   Intuitively, every expression evaluates to either `true` or `false`.
   (a) Give a **small-step** operational semantics for *Bool*.
   (b) Write down all the steps of evaluation needed to reduce the following expression to a result:

$$\begin{array}{l} \neg\,(\texttt{if}\ (\texttt{false} \,\&\, \texttt{true}) \\ \ \texttt{then}\ (\texttt{if}\ \texttt{true}\ \texttt{then}\ (\texttt{false} \,\&\, \texttt{true})\ \texttt{else}\ \texttt{false}) \\ \ \texttt{else}\ \neg\texttt{true} \\ \quad) \end{array}$$

For questions 6 and 7, suppose that the syntax of *SimpleExp* is extended with a new operator ? as follows:

$$E \in \mathit{SimpleExp} ::= \ldots \;\big|\; (E \mathbin{?} E)$$

The intended meaning of this operator is that the implementation can choose[1] given $E_1 \mathbin{?} E_2$ whether to give the result of E_1 or the result of E_2. In particular, we would like both $(1+2)?4 \Downarrow 3$ and $(1+2) \mathbin{?} 4 \Downarrow 4$.

6. (a) Extend the **big-step** operational semantics with rules for ? that capture this meaning.
   (b) For what values of n does $(0 \mathbin{?} 1) + (2 \mathbin{?} 3) \Downarrow n$? Give all of the possible derivation trees.
   (c) Is the semantics deterministic? Is it total? Explain your answer.

7. (a) Extend the **small-step** semantics for *SimpleExp* to handle the ? operator by adding appropriate derivation rules for $\rightarrow$. (Note that there is more than one valid way to do this. Is your choice a good one?)
   (b) Give all possible derivations of the first step of evaluation of $(0 \mathbin{?} 1) + (2 \mathbin{?} 3)$.
   (c) Give all of the possible evaluation paths for $(0 \mathbin{?} 1) + (2 \mathbin{?} 3)$. You do not need to give the derivation for each step.
   (d) Is the semantics confluent? Explain your answer.
   (e) Is the semantics normalising?

8. Suppose that instead of the *SimpleExp* small-step rule S-RIGHT we had the following rule:

$$(\textsc{s-right'})\ \frac{E_2 \rightarrow E_2'}{(E_1 + E_2) \rightarrow (E_1 + E_2')}$$

   (a) Given an evaluation path using the S-RIGHT rule, is it also an evaluation path using the S-RIGHT' rule? Explain your answer.
   (b) Find an expression that has an evaluation path using the S-RIGHT' rule that it did not have with the S-RIGHT rule. Give the evaluation path.
   (c) Is $\rightarrow$ deterministic?
   (d) Is $\rightarrow$ confluent? (You need not prove this formally.)

9. In the notes, we considered numbers to be part of the abstract syntax of expressions. An alternative is to define a concrete syntactic representation of numbers, which we will call *numerals*. A very simple syntax, which represents numerals in unary, is the following:

$$n \in \mathit{Num} ::= \mathtt{0} \;\big|\; \mathtt{S}\, n$$

   Let us define the syntax of numeral expressions using these numerals as the basis:

$$E \in \mathit{NumExp} ::= n \;\big|\; E + E \;\big|\; E - E$$

[1] How it chooses — randomly, by asking the user, with some strategy — is not important here.

(a) Denotational semantics maps syntax directly into mathematical entities (with a function that is defined inductively on the syntax). An appropriate denotational semantics of numerals as natural numbers may be given by the following function:

$$
\begin{aligned}
[\![\mathtt{0}]\!] &= 0 \\
[\![\mathtt{S}\,n]\!] &= 1 \,\underline{+}\, [\![n]\!]
\end{aligned}
$$

What is $[\![\mathtt{S\,S\,S\,S\,0}]\!]$? What numeral n is such that $[\![n]\!] = 7$?

(b) The small-step operational semantics for addition may be given by the following rules:

$$
\textsc{s-add-left}\ \frac{E_1 \to E_1'}{E_1 + E_2 \to E_1' + E_2}
\qquad
\textsc{s-add-right}\ \frac{E_2 \to E_2'}{n_1 + E_2 \to n_1 + E_2'}
$$

$$
\textsc{s-add-zero}\ \frac{}{\mathtt{0} + n \to n}
\qquad
\textsc{s-add-step}\ \frac{}{\mathtt{S}\,n_1 + n_2 \to n_1 + \mathtt{S}\,n_2}
$$

Give the evaluation sequence for the expression $\mathtt{S\,S\,0}+(\mathtt{S\,S\,0}+\mathtt{S\,0})$, and the derivation tree for the first step.

(c) Define a small-step operational semantics for subtraction that is deterministic, such that the result of any subtraction that would go negative is $\mathtt{0}$.