Computation Answers 1: Expressions

1. The number is, of course, 9. The derivation tree is:

$$\text{(B-ADD)}\ \dfrac{\text{(B-ADD)}\ \dfrac{\text{(B-NUM)}\ \dfrac{}{4 \Downarrow 4} \quad \text{(B-NUM)}\ \dfrac{}{1 \Downarrow 1}}{(4+1) \Downarrow 5} \qquad \text{(B-ADD)}\ \dfrac{\text{(B-NUM)}\ \dfrac{}{2 \Downarrow 2} \quad \text{(B-NUM)}\ \dfrac{}{2 \Downarrow 2}}{(2+2) \Downarrow 4}}{((4+1)+(2+2)) \Downarrow 9}$$

2. To handle multiplication, we simply add a single rule to our existing system. In addition to the axiom (B-NUM) and the rule (B-ADD), we add the rule

$$\text{(B-MULT)}\ \dfrac{E_1 \Downarrow n_1 \qquad E_2 \Downarrow n_2}{(E_1 \times E_2) \Downarrow n_3}\ n_3 = n_1 \underline{\times} n_2$$

The proof is given by

$$\text{(B-MULT)}\ \dfrac{\text{(B-ADD)}\ \dfrac{\text{(B-NUM)}\ \dfrac{}{3 \Downarrow 3} \quad \text{(B-NUM)}\ \dfrac{}{2 \Downarrow 2}}{(3+2) \Downarrow 5} \qquad \text{(B-ADD)}\ \dfrac{\text{(B-NUM)}\ \dfrac{}{1 \Downarrow 1} \quad \text{(B-NUM)}\ \dfrac{}{4 \Downarrow 4}}{(1+4) \Downarrow 5}}{((3+2) \times (1+4)) \Downarrow 25}$$

3. An obvious first try is

$$\text{(B-SUB)}\ \dfrac{E_1 \Downarrow n_1 \qquad E_2 \Downarrow n_2}{(E_1 - E_2) \Downarrow n_3}\ n_3 = n_1 \underline{-} n_2$$

However, note that, since we do not have negative numbers, this affects the totality of $\Downarrow$. For example, if n_1 is 3 and n_2 is 7 then B-SUB does not apply because $-4 = 3 \underline{-} 7$ is not a natural number. With this approach, there is just no n for which $(3-7) \Downarrow n$, so this expression has *no* final answer at all. A more precise way of writing this rule would be:

$$\text{(B-SUB)}\ \dfrac{E_1 \Downarrow n_1 \qquad E_2 \Downarrow n_2}{(E_1 - E_2) \Downarrow n_3}\ n_3 = n_1 \underline{-} n_2,\ n_1 \geq n_2$$

If we want to have a total semantics, we need a rule that applies when the result would be negative. This is a design decision, and there are several ways in which we can handle this issue. One way would be to define the result to be zero. We can do this by having the following additional rule:

$$\text{(B-SUB-NEG)}\ \dfrac{E_1 \Downarrow n_1 \qquad E_2 \Downarrow n_2}{(E_1 - E_2) \Downarrow 0}\ n_1 < n_2$$

This choice has its own consequences, since it would allow for unusual behaviour, such as

$$4 + (3-7) \Downarrow 4$$

whilst, at the same time

$$(4+3) - 7 \Downarrow 0.$$

Alternatively, we could add a new result value, say `NaN` (for Not a Number), to signal an error in the computation. If we do this, we would have to add new rules for propagating this error condition (or perhaps recovering from it). Illustratively, we would need to have that, for all natural numbers n, $n + \texttt{NaN} = \texttt{NaN} + n = \texttt{NaN}$, and similarly for multiplication.

Note: The very exotic $\texttt{NaN} \neq \texttt{NaN}$ would also have to hold. Why do you think that is?

Another approach is to say that an implementation is allowed to do whatever it likes if the result would be negative. We can approximate this with the following rule:

$$\text{(B-SUB-NEG)}\ \frac{E_1 \Downarrow n_1 \qquad E_2 \Downarrow n_2}{(E_1 - E_2) \Downarrow n_3}\ n_1 < n_2$$

Since this rule does not constrain n_3, a subtraction that would go negative can return an arbitrary result. So, we have $(3 - 7) \Downarrow 15$, $(3 - 7) \Downarrow 42$, $(3 - 7) \Downarrow 0$, ...

Of course, it is completely reasonable to use a language that works with integers, instead of the natural numbers. This was just not our choice, and this is one of the ramifications of that choice. However, real-world programming languages need to make the same choice, albeit in a different setting. While they will surely support subtraction, they also need to be able to deal with mathematically undefined expressions such as 0/0. C++, Java, and JavaScript, for instance, all use the `NaN` approach.

While each of the options that we have given is valid, the important thing to note is that these choices have consequences for the people using the language as well as those implementing it. Choosing to make $\Downarrow$ have no outcome for less-than-zero subtractions places a burden on the implementer to reject programs that require a less-than-zero subtraction. On the other hand, allowing an arbitrary result for less-than-zero subtractions gives the implementer a lot of freedom, but requires more care on the part of the user, who must avoid such subtractions, or deal with the consequences—such code would be extremely difficult to debug. The other options represent more of a middle ground between the obligations of the parties.

4. The full derivation of the first step is

$$\text{(S-LEFT)}\ \frac{\text{(S-ADD)}\ \dfrac{}{(1 + 2) \rightarrow 3}}{((1 + 2) + (4 + 3)) \rightarrow (3 + (4 + 3))}$$

All the steps of the evaluation are given by

$$((1 + 2) + (4 + 3)) \rightarrow (3 + (4 + 3)) \rightarrow (3 + 7) \rightarrow 10$$

The derivations of the other steps are given by

$$\text{(S-RIGHT)}\ \frac{\text{(S-ADD)}\ \dfrac{}{(4 + 3) \rightarrow 7}}{(3 + (4 + 3)) \rightarrow (3 + 7)}$$

$$\text{(S-ADD)}\ \frac{}{(3 + 7) \rightarrow 10}$$

5. (a) A small-step operational semantics for *Bool* is

$$\frac{B_1 \to B_1'}{B_1 \mathbin{\&} B_2 \to B_1' \mathbin{\&} B_2} \qquad \frac{B_2 \to B_2'}{\texttt{true} \mathbin{\&} B_2 \to \texttt{true} \mathbin{\&} B_2'} \qquad \frac{B_2 \to B_2'}{\texttt{false} \mathbin{\&} B_2 \to \texttt{false} \mathbin{\&} B_2'}$$

$$\frac{}{\texttt{true} \mathbin{\&} \texttt{true} \to \texttt{true}} \qquad \frac{}{\texttt{true} \mathbin{\&} \texttt{false} \to \texttt{false}}$$

$$\frac{}{\texttt{false} \mathbin{\&} \texttt{true} \to \texttt{false}} \qquad \frac{}{\texttt{false} \mathbin{\&} \texttt{false} \to \texttt{false}}$$

$$\frac{B \to B'}{\neg B \to \neg B'} \qquad \frac{}{\neg\texttt{true} \to \texttt{false}} \qquad \frac{}{\neg\texttt{false} \to \texttt{true}}$$

$$\frac{B \to B'}{\texttt{if } B \texttt{ then } B_1 \texttt{ else } B_2 \to \texttt{if } B' \texttt{ then } B_1 \texttt{ else } B_2}$$

$$\frac{}{\texttt{if true then } B_1 \texttt{ else } B_2 \to B_1} \qquad \frac{}{\texttt{if false then } B_1 \texttt{ else } B_2 \to B_2}$$

Take a close look at how the expressions are evaluated and note again the contrast between small-step and big-step semantics. In big-step, if an expression (such as B_1 & B_2) depends on two sub-expressions, we can think of evaluating those expressions in parallel. In small-step, we need to evaluate them *one at a time.*

For B_1 & B_2, we first fully evaluate B_1 to B_1', which we know must, in the end, be equal to either `true` or `false`. Only then does it make sense to talk about the evaluation of B_2. We follow the same approach for the `if` statement: we first evaluate the condition B, obtaining a value, and then move on to either B_1 or B_2 depending on that obtained value.

Note: We could have chosen to evaluate B_1 & B_2 right-to-left, i.e. to evaluate B_2 first, which would preserve the intuitive meaning of the conjunction. A more interesting choice would be not to evaluate B_2 at all if B_1 evaluated to `false`. Would that choice make a difference in our very simple *Bool* language? Would it make a difference in a broader context of a language with errors and side effects?

(b)

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

6. (a) The appropriate rules are:

$$(\textsc{b-choice-1})\ \frac{E_1 \Downarrow n_1}{E_1 \mathbin{?} E_2 \Downarrow n_1} \qquad (\textsc{b-choice-2})\ \frac{E_2 \Downarrow n_2}{E_1 \mathbin{?} E_2 \Downarrow n_2}$$

(b) There are three values of n: 2, 3 and 4. However, there are two different derivations which give the value 3. The derivations are as follows:

$$(\textsc{b-add})\ \dfrac{(\textsc{b-choice-1})\ \dfrac{(\textsc{b-num})\ \dfrac{}{0 \Downarrow 0}}{0 \mathbin{?} 1 \Downarrow 0} \qquad (\textsc{b-choice-1})\ \dfrac{(\textsc{b-num})\ \dfrac{}{2 \Downarrow 2}}{2 \mathbin{?} 3 \Downarrow 2}}{(0 \mathbin{?} 1) + (2 \mathbin{?} 3) \Downarrow 2}\ 0 \underline{+} 2 = 2$$

$$\text{(B-ADD)}\ \dfrac{\text{(B-CHOICE-2)}\ \dfrac{\text{(B-NUM)}\ \dfrac{}{1 \Downarrow 1}}{0 \mathbin{?} 1 \Downarrow 1} \qquad \text{(B-CHOICE-1)}\ \dfrac{\text{(B-NUM)}\ \dfrac{}{2 \Downarrow 2}}{2 \mathbin{?} 3 \Downarrow 2}}{(0 \mathbin{?} 1) + (2 \mathbin{?} 3) \Downarrow 3}\ 1 \underline{+} 2 = 3$$

$$\text{(B-ADD)}\ \dfrac{\text{(B-CHOICE-1)}\ \dfrac{\text{(B-NUM)}\ \dfrac{}{0 \Downarrow 0}}{0 \mathbin{?} 1 \Downarrow 0} \qquad \text{(B-CHOICE-2)}\ \dfrac{\text{(B-NUM)}\ \dfrac{}{3 \Downarrow 3}}{2 \mathbin{?} 3 \Downarrow 3}}{(0 \mathbin{?} 1) + (2 \mathbin{?} 3) \Downarrow 3}\ 0 \underline{+} 3 = 3$$

$$\text{(B-ADD)}\ \dfrac{\text{(B-CHOICE-2)}\ \dfrac{\text{(B-NUM)}\ \dfrac{}{1 \Downarrow 1}}{0 \mathbin{?} 1 \Downarrow 1} \qquad \text{(B-CHOICE-2)}\ \dfrac{\text{(B-NUM)}\ \dfrac{}{3 \Downarrow 3}}{2 \mathbin{?} 3 \Downarrow 3}}{(0 \mathbin{?} 1) + (2 \mathbin{?} 3) \Downarrow 4}\ 1 \underline{+} 3 = 4$$

(c) Recall that $\Downarrow$ is deterministic if and only if every expression gives at most one result. That is, for every expression E and numbers n_1, n_2, if $E \Downarrow n_1$ and $E \Downarrow n_2$ then $n_1 = n_2$. However, this is not the case: as we have seen, $(0 \mathbin{?} 1) + (2 \mathbin{?} 3) \Downarrow 2$ and $(0 \mathbin{?} 1) + (2 \mathbin{?} 3) \Downarrow 3$, but $2 \neq 3$. Therefore, $\Downarrow$ is not deterministic.

Recall that $\Downarrow$ is total if and only if for every expression E there is some number n such that $E \Downarrow n$. This is the case, since there is a derivation rule that applies to every expression.

We could formalise the proof by induction on the structure of expressions. Do not worry if you did not give such a formal proof—we will cover induction in more depth later in the course.

7. (a) The choice for defining the semantics of ? depends on whether or not expressions should be evaluated before choosing between them. If we decide to choose between the expressions before evaluating, we can simply have the rules:

$$\text{(S-CHOICE-1)}\ \dfrac{}{E_1 \mathbin{?} E_2 \to E_1} \qquad \text{(S-CHOICE-2)}\ \dfrac{}{E_1 \mathbin{?} E_2 \to E_2}$$

Alternatively, we could require both expressions to be fully evaluated (to numbers) before choosing between them. In this case we can give a similar semantics to the one for +:

$$\text{(S-LEFTC)}\ \dfrac{E_1 \to E_1'}{E_1 \mathbin{?} E_2 \to E_1' \mathbin{?} E_2} \qquad \text{(S-RIGHTC)}\ \dfrac{E_2 \to E_2'}{n_1 \mathbin{?} E_2 \to n_1 \mathbin{?} E_2'}$$

$$\text{(S-CHOICE-1)}\ \dfrac{}{n_1 \mathbin{?} n_2 \to n_1} \qquad \text{(S-CHOICE-2)}\ \dfrac{}{n_1 \mathbin{?} n_2 \to n_2}$$

If we wanted the rules for ? to look a little more like those for +, we could condense these last two into a single one with a side-condition:

$$\text{(S-CHOICE)}\ \dfrac{}{n_1 \mathbin{?} n_2 \to n}\ n \in \{n_1, n_2\}$$

These are probably the most appropriate choices, however, there are others. Note that a rule like the following is *not* in keeping with the spirit of small-step semantics:

$$(\textsc{bad-choice})\ \frac{E_1 \to E_1'}{E_1 \mathbin{?} E_2 \to E_1'}$$

This is because the rule is effectively doing two steps in one: making the choice of which expression to evaluate, and evaluating it for one step. (Furthermore, if E_1 is a number, this rule does not apply since numbers are normal forms — they do not have further evaluation steps.)

(b) For both (good) choices of semantics described above, the derivations are the same, since ? is only applied to numbers. There are two derivations, and they are:

$$(\textsc{s-left})\ \frac{(\textsc{s-choice-1})\ \frac{}{0 \mathbin{?} 1 \to 0}}{(0 \mathbin{?} 1) + (2 \mathbin{?} 3) \to 0 + (2 \mathbin{?} 3)}$$

$$(\textsc{s-left})\ \frac{(\textsc{s-choice-2})\ \frac{}{0 \mathbin{?} 1 \to 1}}{(0 \mathbin{?} 1) + (2 \mathbin{?} 3) \to 1 + (2 \mathbin{?} 3)}$$

(c) There are four evaluation paths, giving three different results. They are:

$$\begin{aligned}
(0 \mathbin{?} 1) + (2 \mathbin{?} 3) &\to 0 + (2 \mathbin{?} 3) \to 0 + 2 \to 2\\
(0 \mathbin{?} 1) + (2 \mathbin{?} 3) &\to 0 + (2 \mathbin{?} 3) \to 0 + 3 \to 3\\
(0 \mathbin{?} 1) + (2 \mathbin{?} 3) &\to 1 + (2 \mathbin{?} 3) \to 1 + 2 \to 3\\
(0 \mathbin{?} 1) + (2 \mathbin{?} 3) &\to 1 + (2 \mathbin{?} 3) \to 1 + 3 \to 4
\end{aligned}$$

(d) Recall that for the semantics to be confluent, for all expressions E, E_1, E_2 with $E \to^* E_1$ and $E \to^* E_2$ there must exist some E' such that $E_1 \to^* E'$ and $E_2 \to^* E'$. If E_1 and E_2 are both normal forms, then this can only hold if they are equal. However, note that 2 and 3 are both normal forms, and $(0 \mathbin{?} 1) + (2 \mathbin{?} 3) \to^* 2$ and $(0 \mathbin{?} 1) + (2 \mathbin{?} 3) \to^* 3$, as we have seen, yet $2 \neq 3$. Therefore, the semantics is not confluent.

(e) Recall that the semantics is normalising if there are no infinite sequences of evaluation steps (*i.e.* every evaluation path eventually terminates). This is the case here. Intuitively, this is because each evaluation step reduces the number of operators in the expression — including the evaluation steps given by the new rules. Since every expression has a finite and non-negative number of operators in it, there can be no infinite evaluation paths.

This argument is similar to the reasoning for proving that a loop terminates that you learned in *Reasoning about Programs*. Effectively, we have a loop that continues to evaluate the expression until it is no longer possible to evaluate it any further, and wish to know if this loop always terminates. The loop variant is the number of operators in the expression. To show termination of the loop, we require that the variant is always at least 0 (easy, since an expression cannot have a negative number of operators) and that it decreases on each iteration of the loop.

To formally prove that each evaluation step reduces the number of operators we can use induction on the structure of the derivation of evaluation steps. We shall see how to do such inductions later in the course. (If you are already confident in your induction abilities, you may wish to try this.)

8. (a) Yes. The S-RIGHT rule is a special case of the S-RIGHT' rule: anything that is derived with the first can be derived with the second.

   (b) Such an expression must contain the sum of two expressions that are not themselves numbers, such as $((1+2)+(3+4))$. With S-RIGHT', the right side of the expression can be evaluated before the left, which was not possible with S-RIGHT. Thus, we have the evaluation path:

$$((1+2)+(3+4)) \to ((1+2)+7) \to (3+7) \to 10$$

   (c) No. Given what we have already seen, the expression $((1+2)+(3+4))$ has two possible evaluation steps:

$$((1+2)+(3+4)) \to ((1+2)+7)$$
$$((1+2)+(3+4)) \to (3+(3+4))$$

   (d) Yes. (Note that if a system is deterministic then it is necessarily confluent, but it may be confluent without being deterministic!) You should probably have a strong intuition that $\to$ is indeed confluent, but may be unsure how to prove it. We shall see how to later in the course.

   Intuitively, if any expression (say E) has an evaluation step to two different expressions (E_1 and E_2), then there must be some fourth expression (E') that these two expressions both evaluate to in one step.

$$\begin{tikzcd}
 & E \arrow[dl] \arrow[dr] & \\
E_1 \arrow[dr] & & E_2 \arrow[dl] \\
 & E' &
\end{tikzcd}$$

   So if two evaluation sequences get one step out with each other, they can get back together in one step. Because this diamond property holds everywhere, if they get two steps out then they can also get back together in (at most) two steps:

$$\begin{tikzcd}
 & & {} \arrow[dl] \arrow[dr] & & \\
 & {} \arrow[dl] \arrow[dr] & & {} \arrow[dl] \arrow[dr] & \\
{} \arrow[dr] & & {} \arrow[dl] \arrow[dr] & & {} \arrow[dl] \\
 & {} \arrow[dr] & & {} \arrow[dl] & \\
 & & {} & &
\end{tikzcd}$$

   So, after evaluating any expression for n steps in two different ways, the resulting expressions can each be evaluated to a fourth expression in at most n steps. Consequently, the semantics is confluent.

9. (a) $[\![\mathsf{S\,S\,S\,S\,0}]\!] = 4$. $[\![\mathsf{S\,S\,S\,S\,S\,S\,S\,0}]\!] = 7$.

   (b) The evaluation sequence is as follows: $\mathsf{S\,S\,0} + (\mathsf{S\,S\,0} + \mathsf{S\,0}) \to \mathsf{S\,S\,0} + (\mathsf{S\,0} + \mathsf{S\,S\,0}) \to \mathsf{S\,S\,0} + (\mathsf{0} + \mathsf{S\,S\,S\,0}) \to \mathsf{S\,S\,0} + \mathsf{S\,S\,S\,0} \to \mathsf{S\,0} + \mathsf{S\,S\,S\,S\,0} \to \mathsf{0} + \mathsf{S\,S\,S\,S\,S\,0} \to \mathsf{S\,S\,S\,S\,S\,0}$.

The derivation for the first step is:

$$\textsc{s-add-right}\ \dfrac{\textsc{s-add-step}\ \dfrac{}{\mathtt{S\,S\,0} + \mathtt{S\,0} \to \mathtt{S\,0} + \mathtt{S\,S\,0}}}{\mathtt{S\,S\,0} + (\mathtt{S\,S\,0} + \mathtt{S\,0}) \to \mathtt{S\,S\,0} + (\mathtt{S\,0} + \mathtt{S\,S\,0})}$$

(c) Following the style of rules for addition:

$$\textsc{s-sub-left}\ \dfrac{E_1 \to E_1'}{E_1 - E_2 \to E_1' - E_2} \qquad \textsc{s-sub-right}\ \dfrac{E_2 \to E_2'}{n_1 - E_2 \to n_1 - E_2'}$$

$$\textsc{s-sub-zero-l}\ \dfrac{}{\mathtt{0} - n \to \mathtt{0}} \qquad \textsc{s-sub-zero-r}\ \dfrac{}{n - \mathtt{0} \to n}$$

$$\textsc{s-sub-step}\ \dfrac{}{\mathtt{S}\,n_1 - \mathtt{S}\,n_2 \to n_1 - n_2}$$

We could also have used the following variant of s-sub-zero-l:

$$\textsc{s-sub-zero-l}\ \dfrac{}{\mathtt{0} - \mathtt{S}\,n \to \mathtt{0}}$$

(We could do a similar replacement for s-sub-zero-r, but not for both at the same time, as this would give no derivation for $\mathtt{0} - \mathtt{0}$.)
Another alternative is to make subtraction non-strict in the right operand, so that it is only evaluated when the left operand does not evaluate to $\mathtt{0}$:

$$\textsc{s-sub-left}\ \dfrac{E_1 \to E_1'}{E_1 - E_2 \to E_1' - E_2} \qquad \textsc{s-sub-right}\ \dfrac{E_2 \to E_2'}{\mathtt{S}\,n_1 - E_2 \to \mathtt{S}\,n_1 - E_2'}$$

$$\textsc{s-sub-zero-l}\ \dfrac{}{\mathtt{0} - E \to \mathtt{0}} \qquad \textsc{s-sub-zero-r}\ \dfrac{}{n - \mathtt{0} \to n}$$

$$\textsc{s-sub-step}\ \dfrac{}{\mathtt{S}\,n_1 - \mathtt{S}\,n_2 \to n_1 - n_2}$$

This semantics is deterministic because the only two overlapping cases are for $\mathtt{0} - \mathtt{0}$, which evaluates to $\mathtt{0}$ by both applicable rules.