# Chapter 1

# Evaluating arithmetic expressions

In this introductory chapter we explain the idea of formal semantics for a programming language using as an example a very simple language for arithmetic expressions *Exp*, involving numerals and two operations, addition and multiplication. Anybody reading these notes will know very well how to evaluate these expressions. But our purpose is to use the language to explain the formalism we will use to give semantics to languages which are much more complicated than *Exp*.

## 1.1 Syntax

The syntax for a very simple language of *arithmetic expressions Exp* is given in Figure 1.1. It uses an auxiliary set of *numerals*, *Nums*, which are syntactic representations of the more abstract set of natural numbers $\mathbb{N}$. The natural numbers 0, 1, 2, ... are mathematical objects which exist in some abstract world of concepts. They have concrete representations in different languages. For example the natural number 5 is represented by the string of symbols *five* in English and the string *cinq* in French; the Romans represented it by the symbol *V*. In our language of arithmetic expressions it will be represented as the corresponding symbol in bold italic font `5`.

In addition to the numerals the BNF schema in Figure 1.1 also uses two extra symbols, + and ×. Once more most people would know that these symbols are representations for binary mathematical operations on natural numbers, namely *addition* and *multiplication*. Thus the first line of Figure 1.1 says that there are three ways to construct an arbitrary expression *E* in the language *Exp*:

(i) If `n` is an arbitrary numeral then it is also an arithmetic expression. From this we therefore already know that there an infinite number of arithmetic expressions, namely `0`, `1`, `2`, ...

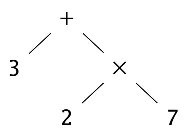
(ii) If we have already constructed two arithmetic expressions $E_1$ and $E_2$ then $E_1 + E_2$ is also an arithmetic expression in *Exp*.

$$E \in \mathit{Exp} ::= \mathtt{n} \mid E + E \mid E \times E$$
$$\mathtt{n} \in \mathit{Nums} ::= \mathtt{0} \mid \mathtt{1} \mid \mathtt{2} \mid \ldots$$

Figure 1.1: Syntax: arithmetic expressions

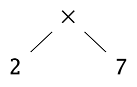
(iii) Similarly if $E_1$, $E_2$ are two expressions in *Exp* then $E_1 \times E_2$ is also an arithmetic expression in *Exp*.

Here we take the view that schemas such as that in Figure 1.1 specify the *abstract syntax* of a language, rather than its *concrete syntax*. The latter is concerned with the precise linear sequences of symbols which are valid terms of the language whereas the former describes terms purely in terms of their structure. Another way of saying this is that the schema in Figure 1.1 describes the valid *abstract syntax trees* of the language, rather than linear sequences of symbols. Thus the following is a valid tree in the language *Exp*:

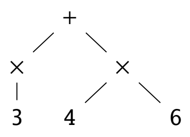


This is because it is formed by condition (ii) above because:

(a) 3 is a valid tree in *Exp*; this follows from condition (i)

(b) the object  is also in *Exp*. This in turn follows by condition (iii) above, because both the objects 2 and 7 are valid trees; these two statements are an instance of condition (i).

On the other hand a tree such as



is not in the language *Exp*; no matter how we try to apply the rules (i) - (iii) above we will not be able to construct it.

However it would be tedious to have to continually draw these syntax trees and therefore throughout the notes we use a convention for their linear representation; this consists of using brackets in order to indicate the structure of expressions. Thus in linear representation the valid tree above will be rendered as $3 + (2 \times 7)$. The linear representation $(3 + 2) \times 7$ on the other hand represents a different tree, namely

$$\begin{tikzcd}
 & & \times \arrow[dl, dash] \arrow[dr, dash] & \\
 & + \arrow[dl, dash] \arrow[dr, dash] & & 7 \\
3 & & 2 &
\end{tikzcd}$$

This linear representation of abstract tress will be rather informal; for example there are many linear expressions, such as $3 + 2 \times 7$, which represent no abstract syntax tree. The over-riding principle will be that given an expression we should always know its structure; how it is constructed using the rules (i) (ii) and (iii) above.

## 1.2 Big-step semantics

Anybody with the least exposure to mathematics will know how to evaluate expressions in the language *Exp*; for example $3+(2\times7)$ evaluates to $17$ while $(3+2)\times7$ evaluates to $35$. However this might not be the case for more complicated languages, and therefore we need general methods for specifying how expressions are to be evaluated, or more abstractly what should be the result of evaluating an expression. We will illustrate these methods using the simple language *Exp*.

One approach would simply be to write a computer programme, an *evaluator* or *interpreter*, which inputs an arithmetic expression and outputs the correct result. However this is unsatisfactory for a number of reasons:

(i) As an explanation it is unnecessarily complicated. Writing the programme would involve all kinds of superfluous decisions about data-structures, and control flow.

(ii) It would also be overly prescriptive; the program would essentially give a specific algorithm for evaluating expressions, thereby offering a bias against other possibilities.

Suppose instead we merely wanted to *specify* what the result should be, rather than how the evaluation should proceed. One way to do this would be to publish a table consisting of all the possible expressions together with the numeral to which they should evaluate. Apart from being incredibly tedious this approach is doomed to failure as there are an infinite number of possible expressions. But as is made clear in the BNF description of the language in Figure 1.1, there is a simple structure to all expressions; this can be exploited to give a simple specification of what the result should be from any algorithm designed to evaluate an arbitrary expression.

But any such specification can only be understood by somebody who is familiar with the abstract arithmetic operations of addition and subtraction. Note that this is also true of *evaluators* or *interpreters*; it would be impossible to implement a program to evaluate expressions if the target language had no way to execute these arithmetic operations.

Suppose we want to evaluate an arbitrary expression $E \in Exp$. According to the description of *Exp* in Figure 1.1 there are three possibilities for the structure of $E$:

$$\textsc{(b-num)}\ \frac{}{\mathtt{n} \Downarrow \mathtt{n}} \qquad \textsc{(b-add)}\ \frac{E_1 \Downarrow \mathtt{n}_1 \quad E_2 \Downarrow \mathtt{n}_2}{E_1 + E_2 \Downarrow \mathtt{n}_3}\ n_3 = \mathsf{add}(n_1, n_2)$$

Figure 1.2: Big-step semantics

(i) $E$ is some numeral $\mathtt{n}$: In this case the result of evaluation should obviously be the numeral $\mathtt{n}$ itself.

(ii) $E$ has the structure $E_1 + E_2$ for some (sub)-expressions $E_1$ and $E_2$. In this case the result of evaluating $E$ should be the numeral obtained by applying the binary addition operator to the results obtained from $E_1$ and $E_2$. Spelled out in more detail, if $\mathtt{n}_1$ is the result of evaluating $E_1$ and $\mathtt{n}_2$ is the result of evaluating $E_2$ then the result of evaluating $E$ should be the numeral $\mathtt{n}_3$ where $\mathsf{add}(n_1, n_2) = n_3$.

(iii) $E$ has the structure $E_1 \times E_2$ for some (sub)-expressions $E_1$ and $E_2$. In this case we proceed as in case (ii) but using the multiplication operator $\mathsf{mult}(-,-)$ in place of addition.

Note the use of numbers versus numerals in (ii) and (iii). Both $\mathsf{add}(-,-)$ and $\mathsf{mult}(-,-)$ are abstract mathematical operations on natural numbers; so in (ii) they are applied to the numbers $n_1, n_2$, to obtain the number $n_3$, and the result of the valuation is the corresponding numeral $\mathtt{n}_3$.

The specification given in (i)-(iii) above does not necessarily constitute a precise algorithm for evaluating expressions but it can be used by any reasonably intelligent person to calculate the prescribed result. For example the result of evaluating $(\mathtt{2} + \mathtt{6}) + (\mathtt{2} \times \mathtt{7})$ should be the numeral $\mathtt{22}$. This follows by an application of (ii) because:

(a) $(\mathtt{2} + \mathtt{6}) + (\mathtt{2} \times \mathtt{7})$ has the form $E_1 + E_2$ where $E_1$ is $\mathtt{2} + \mathtt{6}$ and $E_2$ is $\mathtt{2} \times \mathtt{7}$

(b) the result of evaluating $\mathtt{2} + \mathtt{6}$ should be $\mathtt{8}$

(c) the result of evaluating $\mathtt{2} \times \mathtt{7}$ should be $\mathtt{14}$

(d) and $\mathsf{add}(8, 14)$ is the number 22.

Of course this is not the complete justification of why $(\mathtt{2}+\mathtt{6})+(\mathtt{2}\times\mathtt{7})$ should evaluate to $\mathtt{22}$. In addition we need to justify steps (b) and (c) above; these in turn can be justified using applications of the principles (ii) and (iii) respectively.

With some thought the reader should be convinced that these principles, (i), (ii), and (iii), are sufficient to determine the value of any expression from *Exp* no matter how complicated. However they are expressed in natural language (English), which is notoriously prone to mis-interpretation and mis-understanding. For *Exp*, a very simple language, this is not the case, but for more complicated languages it is better to avoid the vagaries of natural language. So instead we propose to replace specifications such as (i) - (iii) above with formal logical systems which do not suffer from the defects of natural language.

The idea is to use logical rules whose general format is given by:

$$\frac{\overset{\text{name}}{\text{hypothesis}} \quad \ldots \text{hypothesis}}{\text{conclusion}} \; \text{(side-condition)} \tag{1.1}$$

Each rule has

- at least one conclusion, written underneath the line
- a list, possibly empty, of hypotheses, written above the line
- a side-condition, again possibly empty
- a name with which we can refer to the rule.

The intuition is that if all the hypotheses hold, and the side-condition holds, then the conclusion also holds.

Let us now see how we can recast the informal specification of the semantics above using this form of logical rules. The predicate in which we are interested is: *the expression E should evaluate to the numeral* $\mathtt{n}$. Let us denote this English phrase with a mathematical predicate or *judgement*

$$E \Downarrow \mathtt{n}$$

Now what we want is a set of rules which determine valid instances of this predicate. Two such rules are given in Figure 3.2, corresponding to the informal specifications (i) and (ii) above; the missing third rule can be supplied by the reader to correspond with clause (iii). The first rule, (B-NUM), has no hypothesis and no side condition; such rules are refered to as *axioms*. Thus it says that $\mathtt{n} \Downarrow \mathtt{n}$ for every numeral $\mathtt{n}$; thus it corresponds to the informal specification (i) above. The second rule, (B-ADD), corresponds to the informal specification (ii); it has two hypotheses, namely that $E_1 \Downarrow \mathtt{n}_1$ and $E_2 \Downarrow \mathtt{n}_2$ and one side-condition about natural numbers, $n_3 = \mathsf{add}(n_1, n_2)$. If these hypotheses are known to hold and the side-condition is true then the conclusion $E_1 + E_2 \Downarrow \mathtt{n}_3$ is also true.

These rules can now be used formally to determine when, for a particular expression $E$ and numeral $\mathtt{n}$, the judgement $E \Downarrow \mathtt{n}$ is valid. Valid judgements are those which can be derived by any sequence of applications of the defining rules. Here is an example of such a derivation, which determines that the judgement $3 + (2 + 1) \Downarrow 6$ is valid, that is, the evaluation of the expression $3 + (2 + 1)$ should evaluate to the numeral $6$.

$$\dfrac{\dfrac{}{3 \Downarrow 3}\,\text{(B-NUM)} \qquad \dfrac{\dfrac{}{2 \Downarrow 2}\,\text{(B-NUM)} \quad \dfrac{}{1 \Downarrow 1}\,\text{(B-NUM)}}{(2+1) \Downarrow 3}\,\text{(B-ADD)}}{3 + (2+1) \Downarrow 6}\,\text{(B-ADD)}$$

The derivation is presented as an inverted tree, with the required judgement to be verified, $3 + (2 + 1) \Downarrow 6$, at the root. The tree is generated by applications of the defining

$$\dfrac{\dfrac{\dfrac{}{2 \Downarrow 2}\,(\textsc{b-num}) \qquad \dfrac{}{6 \Downarrow 6}\,(\textsc{b-num})}{(2+6) \Downarrow 8}\,(\textsc{b-add}) \qquad \dfrac{\dfrac{}{2 \Downarrow 2}\,(\textsc{b-num}) \qquad \dfrac{}{7 \Downarrow 7}\,(\textsc{b-num})}{(2\times 7) \Downarrow 14}\,(\textsc{b-mult})}{(2+6)+(2\times 7) \Downarrow 22}\,(\textsc{b-add})$$

Figure 1.3: An example derivation in the big-step semantics

rules, with the terminating leaves being generated by axioms. In this example we have three applications of the axiom (b-num) and two applications of the rule (b-add).

Another example derivation is given in Figure 1.3; it makes reference to the (obvious) missing rule (b-mult) for dealing with expressions of the form $E_1 \times E_2$. This is a formal justification of the valid judgement $(2 + 6) + (2 \times 7) \Downarrow 22$ corresponding to the informal justification given in natural language in the clauses (a)-(d) on page 6.

We now sum up what has been achieved in this section. To do so let us introduce the notation

$$\vdash_{big} E \Downarrow \mathtt{n} \tag{1.2}$$

to mean that there is some derivation of the judgement $E \Downarrow \mathtt{n}$ using the three rules (b-num), (b-add) and (b-mult). For example, because Figure 1.3 exhibits a derivation of the judgement $2 + 6) + (2 \times 7) \Downarrow 22$, we can conclude $\vdash_{big} 2 + 6) + (2 \times 7) \Downarrow 22$. Then we can say that we have given a formal semantics to the language *Exp*. By this we mean that if somebody asks the question: *To what value should the expression E evaluate?* we can answer: $E$ should evaluate to a numeral $\mathtt{n}$ such that $\vdash_{big} E \Downarrow \mathtt{n}$.

Before moving on we should say a few words about the format of the logical rules which we use, in (1.1) above. We have not been very specific about the contents of the various components, *hypothesis*, *conclusion* and *side-condition*. In general the purpose of a rule is to constrain some predicate, the focus of the semantic definition. In this case the predicate is $\Downarrow$, a binary infix predicate between expressions and numerals. Consequently it is natural that the *conclusion*, and very often the *hypotheses*, be particular instances of this predicate; this is the case in the rules (b-add) and (b-num) in Figure 3.2. On the other hand *side-condition* should concern auxiliary predicates and functions which play a role, but a minor role, in the definition of the main predicate. We have seen that it is not possible to understand the semantics of *Exp* without knowing that the symbols $+$ and $\times$ refer to the mathematical functions $\mathsf{add}(-,-)$ and $\mathsf{mult}(-,-)$ on natural numbers; and in our rules the side-conditions refer to properties of these auxiliary functions. Thus although one might consider an alternative rule such as

$$\overset{(\textsc{b-add.alt})}{\dfrac{E_1 \Downarrow \mathtt{n}_1 \qquad n_3 = \mathsf{add}(n_1, n_2)}{E_1 + E_2 \Downarrow \mathtt{n}_3}} \quad E_2 \Downarrow \mathtt{n}_2$$

the original rule (b-add) in Figure 3.2 is to be preferred.

(S-LEFT)
$$\frac{E_1 \rightarrow E_1'}{(E_1 + E_2) \rightarrow (E_1' + E_2)}$$

(S-N.RIGHT)
$$\frac{E_2 \rightarrow E_2'}{(\mathtt{n} + E_2) \rightarrow (\mathtt{n} + E_2')}$$

(S-ADD)
$$\frac{}{(\mathtt{n}_1 + \mathtt{n}_2) \rightarrow \mathtt{n}_3} \quad n_3 = \mathsf{add}(n_1, n_2)$$

Figure 1.4: Small-step semantics

We should also point out that a rule such as (B-ADD) is actually a *meta-rule*, that is formally represents an infinite number of concrete rules, obtained by instantiating the *meta-variables* $E_1$, $E_2$, $\mathtt{n}_1$, $\mathtt{n}_2$ and $\mathtt{n}_1$. Thus among the many instances of (B-ADD) are

$$\frac{3 \times 7 \Downarrow 4 \qquad 8 \Downarrow 2}{(3 \times 7) + 8 \Downarrow 6} \; 6 = \mathsf{add}(4, 2) \qquad \frac{4 + 2 \Downarrow 9 \qquad 8 + 1 \Downarrow 3}{(3 \times 7) + (8 + 1) \Downarrow 12} \; 12 = \mathsf{add}(9, 3)$$

However the vast majority of these concrete instances are useless; if the premises can not be established then they can not be employed in any valid derivation.

## 1.3 Small-step semantics

The big-step semantics of the previous section is not very constraining; it prescribes what the answer should be when an expression is evaluated but says nothing about how the actual evaluation is to proceed. For example, to evaluate $(3 + 7) + (8 \times 1)$ we know that two additions have to preformed and one multiplication; but the big-step semantics does not decree in what order these are to be carried out. For some languages, for example those with *side-effects*, the order of evaluation is important. In this section we see an alternative semantics for *Exp* in which constraints on the order of the basic operations can be made. In particular it will prescribe, indirectly, that the order of evaluations should be from left to right.

The idea is to design a predicate on expressions which decrees which operation is to be performed first, and then describes the result of performing this operation. This is achieved indirectly by defining judgements of the form

$$E_1 \rightarrow E_2$$

to be read as: *after performing one step of evaluation* of the expression $E_1$ the expression $E_2$ remains to be evaluated; thus this judgement prescribes

- the first operation to be performed, transforming $E_1$ into $E_2$
- the remaining operations to be performed, embodied indirectly in the the residual $E_1$.

The rules defining this small step relation $\rightarrow$ are given in Figure 1.4, although we leave it to the reader to design the two rules, similar to (S-LEFT) and (S-N.RIGHT), for dealing with expressions of the form $E_1 \times E_2$. Let us write

$$\vdash_{sm} E_1 \rightarrow E_2$$

to mean that there is a derivation of the judgement $E_1 \rightarrow E_2$ using these rules. Thus we have

$$\vdash_{sm} (3 + 7) + (8 + 1) \rightarrow 10 + (8 + 1)$$

because of the following derivation:

$$\dfrac{\dfrac{}{3 + 7 \rightarrow 10}\ (\text{S-ADD})}{(3 + 7) + (8 + 1) \rightarrow 10 + (8 + 1)}\ (\text{S-LEFT})$$

As another example we have

$$\vdash_{sm} 10 + (8 + 1) \rightarrow 10 + 9$$

because the following is a valid derivation:

$$\dfrac{\dfrac{}{8 + 1 \rightarrow 9}\ (\text{S-ADD})}{10 + (8 + 1) \rightarrow 10 + 9}\ (\text{S-N.RIGHT})$$

On the other hand we do *not* have

$$\vdash_{sm} (3 + 7) + (8 + 1) \rightarrow (3 + 7) + 9$$

because no matter how inventive we are with the rules in Figure 1.4 we will not be able to construct a derivation of the judgement $(3 + 7) + (8 + 1) \rightarrow (3 + 7) + 9$; the reader is invited to try.

By trying various examples readers should be able to convince themselves that if $\vdash_{sm} E_1 \rightarrow E_2$ then $E_2$ is obtained from $E_1$ by executing the left-most occurrence of an operator, $+, \times$, which has both its operands already evaluated. For example we have

$$\begin{aligned}
&\vdash_{sm} (3 + 4) + (5 + 6) \rightarrow 7 + (5 + 6)\\
&\vdash_{sm} 3 + (4 + (5 + 6)) \rightarrow (3 + (4 + 11)\\
&\vdash_{sm} (3 + (4 + 5)) + 6 \rightarrow (3 + 9) + 6)
\end{aligned}$$

How do we use the small-step semantics to evaluate an expression, as in the previous section? We construct derivations again and again until a numeral is obtained. For example we have seen that $\vdash_{sm} (3+7)+(8+1) \rightarrow 10+(8+1)$ and $\vdash_{sm} 10+(8+1) \rightarrow 10+9$. In other words in two steps the expression $(3 + 7) + (8 + 1)$ can be reduced to $10 + 9$;

this we write as $\vdash_{sm} (3+7)+(8+1) \rightarrow^2 \mathtt{10}+9$. More generally for any natural number $k \geq 0$ we write

$$E_0 \rightarrow^k E_k$$

if $E_0$ can be reduced to $E_k$ in $k$ steps; that is, there are intermediate expressions $E_i$ such that

$$\vdash_{sm} E_o \rightarrow E_1 \qquad \vdash_{sm} E_1 \rightarrow E_2 \ldots\ldots \vdash_{sm} E_{k-1} \rightarrow E_k$$

This includes the case when $k$ is 0, when $E_k$ must be the same as $E_0$; that is in 0 steps $E_0$ can only reduce to itself. For example the reader should check the following judgements, by showing that derivations can be obtained for appropriate intermediate expressions:

$$\begin{aligned}
&(3 + (4 + 5)) + 6 \rightarrow^2 12 + 6\\
&3 + (4 + (5 + 6)) \rightarrow^2 3 + 15\\
&(3 + 7) + (8 + 1) \rightarrow^3 19\\
&3 + (4 + (5 + 6)) \rightarrow^0 3 + (4 + (5 + 6))
\end{aligned}$$

To fully evaluate an expression we need to indefinitely apply the operations + and × until eventually a final numeral is obtained. Let us write

$$E \rightarrow^* \mathtt{n}$$

to mean that there is some natural number $k \geq 0$ such that $E \rightarrow^k \mathtt{n}$; in other words $E$ can be reduced to the numeral $\mathtt{n}$ in some number $k$ steps. The reader should verify that the following judgements are true, by instantiating the required number $k$:

$$\begin{aligned}
&(3 + 7) + (8 + 1) \rightarrow^* 19\\
&(3 + 4) + (5 + 6) \rightarrow^* 18\\
&3 + (4 + (5 + 6)) \rightarrow^* 18
\end{aligned}$$

So just as the big-step semantics associates a value $\mathtt{n}$ to an expression $E$, via the judgements $\vdash_{big} E \Downarrow \mathtt{n}$, the small-step semantics provides an alternative method for doing so, via the slightly more complicated judgements $\vdash_{sm} E \rightarrow^* \mathtt{n}$.

## 1.4 Parallel evaluation

As we have seen, the small-step semantics prescribes a particular order in which the operators in an expression are applied, namely *left-to-right*. Suppose we wish to relax this; suppose we just want to dictate that all the operators are applied but wish to leave the precise sequencing open. One of the roles of a formal semantics is to act as a reference for compiler writers or implementers. Leaving the order of evaluation open could then allow, for example, compiler writers to take advantage of technologies such as multi-core to increase the efficiency of an implementation.

(S-LEFT)
$$\frac{E_1 \rightarrow_{ch} E_1'}{(E_1 + E_2) \rightarrow_{ch} (E_1' + E_2)}$$

(S-RIGHT)
$$\frac{E_2 \rightarrow_{ch} E_2'}{(E_1 + E_2) \rightarrow_{ch} (E_1 + E_2')}$$

(S-ADD)
$$\frac{}{(\mathtt{n}_1 + \mathtt{n}_2) \rightarrow_{ch} \mathtt{n}_3} \quad n_3 = \mathsf{add}(n_1, n_1)$$

Figure 1.5: Parallel semantics

In Figure 1.5 we give an alternative small-step semantics, with judgements of the form $E_1 \rightarrow_{ch} E_2$, with the subscript referring to *choice*. Two rules are inherited from Figure 1.4 but the rule (S-N.RIGHT) is replaced with the less restrictive (S-RIGHT). The net effect of the presence of the two rules (S-LEFT) and (S-RIGHT) is that when evaluating an expression of the form $E_1 + E_2$ the compiler or interpreter may choose to work on either of $E_1$ or $E_2$. For example we have the derivation:

$$\frac{\dfrac{}{\mathtt{8 + 1} \rightarrow_{ch} \mathtt{9}}\;(\text{S-ADD})}{\mathtt{(3 + 7) + (8 + 1)} \rightarrow_{ch} \mathtt{(3 + 7) + 9}}\;(\text{S-RIGHT})$$

Using $\vdash_{ch} E_1 \rightarrow_{ch} E_2$ to denote the fact that the judgement $E_1 \rightarrow_{ch} E_2$ can be derived using the rules from Figure 1.5, we therefore have

$$\vdash_{ch} \mathtt{(3 + 7) + (8 + 1)} \rightarrow_{ch} \mathtt{(3 + 7) + 9} \tag{1.3}$$

in addition to

$$\vdash_{ch} \mathtt{(3 + 7) + (8 + 1)} \rightarrow_{ch} \mathtt{10 + (8 + 1)} \tag{1.4}$$

Recall from the previous section that this reduction (1.4) is not possible in the standard *left-to-right* semantics. On the other hand note that every application of the rule (S-N.RIGHT) is also an application of the more general (S-RIGHT). This means that any derivation in the *left-to-right* semantics is also a derivation in the *parallel* semantics. It follows that

$$\vdash_{sm} E_1 \rightarrow E_2 \text{ implies } \vdash_{ch} E_1 \rightarrow_{ch} E_2 \tag{1.5}$$

In other words the *parallel* semantics is more general than the *left-to-right*; it allows all the derivations of the *left-to-right* semantics but in addition it allows others such as (1.4) above.

## 1.5 Questions questions

We have now seen three different semantics for the simple language of expressions *Exp*, and various questions arise naturally. For example, intuitively we expect every expression in *Exp* to have a corresponding value. In terms of the big-step semantics we expect the following to be true:

(Q1) For every expression $E$ in *Exp* there exists some numeral $\mathtt{n}$ such that $\vdash_{big} E \Downarrow \mathtt{n}$.

The advantage of a formal semantics is that statements such as (Q1) can be formally proved, or indeed disproved. The predicate $\Downarrow$ between expressions and numerals is formally defined using a set of logical rules, those in Figure 3.2, and therefore (Q1) amounts to a mathematical statement about the mathematical object $\Downarrow$. As such it is either mathematically true or false, which can be demonstrated using standard mathematical techniques. These techniques will be seen in the next chapter.

The same property, often refered to as **Normalisation**, can also be asked of the other two semantics we have seen. These amount to:

(Q2) For every expression $E$ in *Exp* there exists some numeral $\mathtt{n}$ such that $E \rightarrow^* \mathtt{n}$.

(Q3) For every expression $E$ in *Exp* there exists some numeral $\mathtt{n}$ such that $E \rightarrow^*_{ch} \mathtt{n}$.

Again because these are formal mathematical statements we will see how they can be demonstrated formally.

Another property we would naturally expect of a mechanism for evaluating expressions is a form of *internal consistency*. It would be unfortunate if there was some expression with multiple possible values; that is some expression $E$ such that the first time it is evaluated we would get $E \Downarrow \mathtt{n}_1$ while a subsequent evaluation gives $E \Downarrow \mathtt{n}_2$ where $n_2$ is different than $n_1$. The property which rules out this phenomenon is refered to as **Determinacy**. For each of the three semantics this is defined as follows:

$$\text{If } \vdash_{big} E \Downarrow \mathtt{n}_1 \text{ and } \vdash_{big} E \Downarrow \mathtt{n}_2 \text{ then } n_1 = n_2. \tag{Q4}$$

$$\text{If } E \rightarrow^* \mathtt{n}_1 \text{ and } E \rightarrow^* \mathtt{n}_2 \text{ then } n_1 = n_2. \tag{Q5}$$

$$\text{If } E \rightarrow^*_{ch} \mathtt{n}_1 \text{ and } E \rightarrow^*_{ch} \mathtt{n}_2 \text{ then } n_1 = n_2. \tag{Q6}$$

The combination of Normalisation and Determinacy means that each of the semantics we have developed for *Exp* determines one and only one value for every expression.

There are also interesting questions involving the consistency between the different semantics. For example it would be unfortunate if, for some some expression $E$, one semantics gave `20` as the resulting value, while another gave `25`. Ensuring that this can not arise amounts to proving mutual consistency of the different semantics. Specifically it would require proofs for the following mathematical statements:

$$\vdash_{big} E \Downarrow \mathtt{n} \text{ implies } E \rightarrow^* \mathtt{n} \tag{Q7}$$

$$E \rightarrow^*_{ch} \mathtt{n} \text{ implies } \vdash_{big} E \Downarrow \mathtt{n} \tag{Q8}$$

These, together with (1.5) above, will mean that each of the three different semantics will associate exactly the same value with a given expression $E$.