Induction

We use a lot of inductive techniques in this course, both to give definitions and to prove facts about our semantics. So, it's worth taking a little while to set out exactly what a proof by induction is, what a definition by induction is, and so on.

When designing an algorithm to solve a problem, we want to know that the result produced by the algorithm is correct, regardless of the input. For example, the quicksort algorithm takes a list of numbers and puts them into ascending order. In this example, we know that the algorithm operates on a *list of numbers*, but we do not know how long that list is or exactly what numbers it contains. Similarly, one may raise questions about depth-first search of a tree: how do we know it always visits all the nodes in a tree if we do not know the exact size and shape of the tree?

In examples such as these, there are two important facts about the input data which allows us to reason about arbitrary inputs:

- the input is *structured*: for example, a non-empty list has a first element and a 'tail', which is the rest of the list, and the binary tree has a root node and two subtrees;
- the input is finite.

In this situation, the technique of *structural induction* provides a principle by which we may formally reason about arbitrary lists, trees, and so on.

Slide 1

What is induction for?

Induction is a technique for reasoning about and working with collections of objects (things!) which are

- *structured* in some well-defined way;
- *finite* but *arbitrarily large and complex*.

Induction exploits the finite, structured nature of these objects to overcome their arbitrary complexity.

These kinds of structured, finite objects arise in many areas of computer science. Data structures such as lists and trees are common, but in fact programs themselves can be seen as structured finite objects. This means that induction can be used to prove facts about *all programs in a certain language*. In semantics, we use this very frequently. We will also make use of induction to reason about purely semantic notions, such as *derivations* of assertions in the operational semantics of a language.

Mathematical Induction

The simplest form of induction is mathematical induction: that is to say, induction over the natural numbers. The principle can be described as follows: given a property $P(\_)$ of natural numbers, to prove that $P(n)$ holds for *all* natural numbers n, it is enough to:

- prove that $P(0)$ holds; and
- prove that if $P(k)$ holds for arbitrary natural number k, then $P(k+1)$ holds too.

Slide 2

You can use induction...

... to reason about things like

- *natural numbers:* each one is finite, but a natural number could be arbitrary big;
- *data structures* such as lists, trees and so on;
- *programs* in a programming language: again, you can write arbitrarily large programs, but they are always finite;
- *derivations* of semantic assertions like $E \Downarrow 4$: these derivations are finite trees of axioms and rules.

Slide 3

Proof by Mathematical Induction

Let $P(\_)$ be a property of natural numbers. The **principle of mathematical induction** states that if

$$P(0) \wedge [\forall k.P(k) \Rightarrow P(k+1)]$$

holds then

$$\forall n.P(n)$$

holds. The number k is called the induction parameter.

Slide 4

Writing an Inductive Proof

To prove that $P(n)$ holds for all natural numbers n, we must do two things:

Base Case: prove that $P(0)$ holds, any way we like.

Inductive Step: let k be an arbitrary number, and assume that $P(k)$ holds. This assumption is called the **induction hypothesis (or IH)**, with parameter k. Using this assumption, prove that $P(k+1)$ holds.

It should be clear why this principle is valid: if we can prove the two things above, then we know:

- $P(0)$ holds;
- since $P(0)$ holds, $P(1)$ holds;
- since $P(1)$ holds, $P(2)$ holds;
- since $P(2)$ holds, $P(3)$ holds;
- ...

Therefore, $P(n)$ holds for any n, regardless of how big n is. This conclusion can *only* be be drawn because every natural number can be reached by starting at zero and adding one repeatedly. The two elements of the induction can be read as saying:

- Prove that P is true at the place where you start: that is, at zero.
- Prove that the operation of adding one *preserves* P: that is, if $P(k)$ is true then $P(k+1)$ is true.

Since every natural number can be 'built' by starting at zero and adding one repeatedly, every natural number has the property P: as you build the number, P is true of everything you build along the way, and it's still true when you've build the number you're really interested in.

Example

Here is perhaps the simplest example of a proof by mathematical induction. We shall show that

$$\sum_{i=0}^{n} i = \frac{n^2+n}{2}$$

So here the property $P(n)$ is

the sum of numbers from 0 to n inclusive is equal to $\frac{n^2+n}{2}$.

Base Case: The base case, $P(0)$, is

the sum of numbers from 0 to 0 inclusive is equal to $\frac{0^2+0}{2}$.

This is obviously true, so the base case holds.

Inductive Step: Here the inductive hypothesis, IH for parameter k, is the statement $P(k)$:

the sum of numbers from 0 to k inclusive is equal to $\frac{k^2+k}{2}$.

From this inductive hypothesis, with parameter k, we must prove that

the sum of numbers from 0 to $k+1$ inclusive is equal to $\frac{(k+1)^2+(k+1)}{2}$.

The proof is a simple calculation:

$$\begin{aligned}
\sum_{i=0}^{k+1} i &= (\sum_{i=0}^{k} i) + (k+1) \\
&= \frac{k^2+k}{2} + (k+1) \quad \text{using IH for } k \\
&= \frac{k^2+k+2k+2}{2} \\
&= \frac{(k^2+2k+1)+(k+1)}{2} \\
&= \frac{(k+1)^2+(k+1)}{2}
\end{aligned}$$

which is what we had to prove.

Defining Functions and Relations over Natural Numbers

As well as using induction to prove properties of natural numbers, we can use it to define functions which operate on natural numbers.
Just as proof by induction proves a property $P(n)$ by considering the case of zero and the case of adding one to a number known to satisfy P, so definition of a function f by induction works by giving the definition of $f(0)$ directly, and building the value $f(k+1)$ out of $f(k)$.
All this is saying is that if you define the value of a function at zero, by giving some value a, and you show how to calculate the value at $k+1$ from that at k, then this does indeed define a function. This function is 'unique' , meaning that it is completely defined by the information you have given; there is no choice about what f can be.
Roughly, the fact that we use $f(k)$ to define $f(k+1)$ in this definition corresponds to the fact that we assume $P(k)$ to prove $P(k+1)$ in a proof by induction.

Slide 5

Definition by induction

We can define a function f on natural numbers by:

Base Case: giving a value for $f(0)$ directly.

Inductive step: giving a value for $f(k+1)$ in terms of $f(k)$.

Slide 6

Inductive Definition of Factorial

The factorial function `fact` is defined inductively on the natural numbers:

- $\texttt{fact}(0) = 1$;
- $\texttt{fact}(k+1) = (k+1) \times \texttt{fact}(k)$.

For example, slide 6 gives an inductive definition of the factorial function over the natural numbers. Slide 7 contains another definitional use of induction. We have already defined the one-step operational semantics on expressions E from $SimpleExp$. This is represented as a relation $E \rightarrow E'$ over expressions. Suppose we wanted to define what is the effect of k reduction steps, for any natural number k. This would mean defining a family of relations $\rightarrow^k$, one for each natural number k. Intuitively, $E \rightarrow^k E'$ is supposed to meant that by applying exactly k computation rules to E we obtain E'.

Slide 7

Multi-step Reductions in *SimpleExp*

The relation $E \rightarrow^n E'$ is defined inductively by:

- $E \rightarrow^0 E$ for every simple expression E in *SimpleExp*;
- $E \rightarrow^{k+1} E'$ if there is some E'' such that

$$E \rightarrow^k E'' \text{ and } E'' \rightarrow E'$$

In slide 7, the first point defines the relation $\rightarrow^0$ outright. In zero steps an expression remains untouched, so $E \rightarrow^0 E$ for every expression E. In the second clause, the relation $\rightarrow^{(k+1)}$ is defined in terms of $\rightarrow^k$. It says that E reduces to E' in $(k+1)$ steps if

- there is some intermediary expression E'' to which E reduces to in k steps;
- this intermediary expression E'' reduces to E' in one step.

The principle of induction now says that each of the infinite collection of relations $\rightarrow^n$ are well-defined.

A Structural View of Mathematical Induction

We said in the last section that mathematical induction is a valid principle because every natural number can be 'built' using zero as a starting point and the operation of adding one as a method of building new numbers from old. We can turn mathematical induction into a form of structural induction by viewing numbers as elements in the following grammar:

$$N ::= \texttt{zero} \mid \texttt{succ}(N).$$

Here succ, short for *successor*, should be thought of as the operation of adding one. Therefore, the number 0 is represented by zero and 3 is represented by

$$\mathtt{succ}(\mathtt{succ}(\mathtt{succ}(\mathtt{zero})))$$

With this view, it really is the case that a number is built by starting from zero and repeatedly applying succ. Numbers, when thought of like this, are finite, structured objects. The principle of induction now says that, to prove $P(N)$ for all numbers N, it suffices to do two things:

Base Case: prove that $P(0)$ holds.

Inductive Step: the inductive hypothesis IH is that $P(K)$ holds for some number K; from this IH, prove that $P(\mathtt{succ}(K))$ also holds.
This is summarized in slide 8.

Slide 8

Structural view of Mathematical Induction

We can view the natural numbers as elements of the following grammar:

$$N ::= \mathtt{zero} \mid \mathtt{succ}(N).$$

To prove that property $P(N)$ holds for every number N:

Base Case: prove $P(\mathtt{zero})$ holds.

Inductive Step: the IH is that $P(K)$ holds for some K; assuming IH, prove that $P(\mathtt{succ}(K))$ follows.

Defining Functions

The principle of defining functions by induction works for this representation of the natural numbers in exactly the same way as before. To define a function f which operates on these numbers, we must

- define $f(\texttt{zero})$ directly;
- define $f(\texttt{succ}(K))$ in terms of $f(K)$.

Structural Induction for Binary Trees

Binary trees are a commonly used data structure. Roughly, a binary tree is either a single *leaf node*, or a *branch node* which has two *subtrees*.

Slide 9

A Syntax for Binary Trees

Binary trees are defined as elements of the following grammar:

$$\texttt{bTree} ::= \texttt{Node} \mid \texttt{Branch}(\texttt{bTree}, \texttt{bTree})$$

Note the similarity with arithmetic expressions.

The principle of *structural induction over binary trees* states that to prove a property $P(T)$ for all trees T, it is sufficient to do the following two things:

Base Case: prove that $P(\texttt{Node})$ holds;

Inductive Step: the inductive hypothesis IH is that $P(T_1)$ and $P(T_2)$ hold for some arbitrary binary trees T_1 and T_2; then from this assumption prove that $P(\texttt{Branch}(T_1, T_2))$ also holds.

Structural Induction over Simple Expressions

The syntax of our illustrative language *SimpleExp* also gives a collection of structured, finite, but arbitrarily large objects over which induction may be

used. The syntax is repeated below:

$$E \in SimpleExp ::= \mathsf{n} \mid (E + E) \mid (E \times E)$$

Recall that n ranges over the numerals $\mathtt{0}, \mathtt{1}, \mathtt{2}, \ldots$ This means that, in this language, there are in fact an infinite number of indecomposable expressions; contrast this with the cases above, where 0 is the only indecomposable natural number, and $\mathtt{Node}$ is the only indecomposable binary tree. Also, note that we can build new expressions from old in two ways, by using $+$ and $\times$. The principle of induction for expressions reflects these differences as follows. If P is a property of expressions, then to prove that $P(E)$ holds for any E, we must do the following:

Base Cases: prove that $P(\mathsf{n})$ holds for every numeral n.

Inductive Step: here the inductive hypothesis IH is that $P(E_1)$ and $P(E_1)$ hold for some E_1 and E_2; assuming IH, prove that both $P((E_1 + E_2))$ and $P((E_1 \times E_2))$ follow.

The conclusion will then be that $P(E)$ is true for *every* expression E. Again, this induction principle can be seen as a case analysis: expressions come in two forms:

- **numerals**, which cannot be decomposed, so we have to prove $P(\mathsf{n})$ directly for each of them; and
- **composite expressions** $(E_1 + E_2)$ and $(E_1 \times E_2)$, which can be decomposed into subexpressions E_1 and E_2. In this case, the induction hypothesis says that we may assume $P(E_1)$ and $P(E_2)$ when trying to prove $P((E_1 + E_2))$ and $P((E_1 \times E_2))$.

Slide 10

Structural Induction for Terms of *SimpleExp*

To prove that property $P(\_)$ holds for all terms in *SimpleExp*, it suffices to prove:

Base Cases: $P(\mathsf{n})$ holds for all n;

Inductive Step: the IH is that $P(E_1)$ and $P(E_2)$ both hold for some arbitrary expression E_1 and E_2; from IH we must prove that $P(E_1 + E_2)$ and $P(E_1 \times E_2)$ follow.

Slide 11

Determinacy and Normalization

Determinacy says that a simple expression cannot evaluate to more than one answer:

for any expression E, if $E \Downarrow \mathsf{n}$ and $E \Downarrow \mathsf{n}'$ then $\mathsf{n} = \mathsf{n}'$.

This proof is a little tricky. See the notes.

Normalization says that a simple expression evaluates to at least one answer:

for every expression E, there is some n such that $E \Downarrow \mathsf{n}$.

This is proved by induction on the structure of E.

It is not difficult to show by induction on the structure of expressions that, for any expression E, there is some numeral n for which $E \Downarrow \mathsf{n}$. Recall that this property is called *normalization:* it says that all programs in our language have a final answer or so-called 'normal form'. It goes hand in hand with another property, called *determinacy*, which states that the final answer is unique.

Exercise (Normalization) For every expression E, there is some n such that $E \Downarrow \mathsf{n}$.

Defining Functions over Simple Expressions

We may also use the principle of induction to define functions which operate on simple expressions.

Slide 12

Definition by Induction for *SimpleExp*

To define a function on all expressions in *SimpleExp*, it suffices to do the following:

- define $f(\mathsf{n})$ directly, for each numeral n;
- define $f((E_1 + E_2))$ in terms of $f(E_1)$ and $f(E_2)$; and
- define $f((E_1 \times E_2))$ in terms of $f(E_1)$ and $f(E_2)$.

For example, we will soon define the *denotational semantics* of simple expressions and programs as a function defined inductively on simple expressions and programs. As a precursor to this, we define, for each expression E, a number $\mathsf{den}(E)$ which is the 'meaning' or the 'final answer' for E.

Slide 13

The function den

For each simple expression E, a number $\mathsf{den}(E)$ is defined inductively on the structure of E by:

- $\mathsf{den}(\mathsf{n}) = n$ for each numeral n;
- $\mathsf{den}((E_1 + E_2)) = \mathsf{den}(E_1) + \mathsf{den}(E_2)$;
- $\mathsf{den}((E_1 \times E_2)) = \mathsf{den}(E_1) \times \mathsf{den}(E_2)$;

Exercise For every simple expression E and number n,

$$\mathsf{den}(E) = n \text{ if and only if } E \Downarrow \mathsf{n}.$$

Again, this definition should be regarded as showing how to build up the 'meaning' of a complex expression, as the expression itself is built up from numberals and uses of $+$ and $\times$.

Structural Induction over Derivations

Another example of a collection of finite, structured objects which we have seen is the collection of *proofs* of statements $E \Downarrow \mathsf{n}$ in the big-step semantics of *SimpleExp*. In general, an operational semantics given by axioms and proof rules defines a collection of proofs of this kind, and induction is available to us for reasoning about them. [To clarify the presentation, we will refer to such proofs as *derivations* in this section.]
Recall the derivation of $\mathtt{(3+(2+1))} \Downarrow \mathtt{6}$:

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

This derivation has three key elements: the *conclusion* $\texttt{(3+(2+1))} \Downarrow \texttt{6}$, and the two *subderivations*, which are

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

We can think of a complex derivation like this as a structured object:

$$\dfrac{\begin{array}{cc} \vdots & \vdots \\ D_1 & D_2 \\ \vdots & \vdots \\ h_1 & h_2 \end{array}}{c}$$

Here, we see a derivation whose last line is

$$\dfrac{h_1 \quad h_2}{c}$$

where h_1 and h_2 are the *hypothesis (or premises)* of the rule and c is the *conclusion* of the rule; c is also the conclusion of the whole derivation. Since the hypotheses themselves must be derived, there are *subderivations* D_1 and D_2 with conclusions h_1 and h_2.

The only derivations which do not decompose into a last rule and a collection of subderivations are those which are simply axioms. Our principle of induction will therefore treat the axioms as the base cases, and the more complex proof as the inductive step.

The principle of structural induction for derivations says that, to prove a property $P(D)$ for every derivation D, it is enough to do the following:

Base Cases: Prove that $P(A)$ holds for every axiom. In the case of the big-step semantics, we must prove that every derivation

$$\dfrac{}{\texttt{n} \Downarrow \texttt{n}}$$

satisfies property P.

Inductive Step: For each rule of the form

$$\dfrac{h_1 \quad h_n}{c}$$

prove that any derivation ending with a use of this rule satisfies the property. Such a derivation has *subderivations* with conclusions $h_1, ..., h_n$, and we may assume that property P holds for each of these subderivations. These assumptions form the inductive hypothesis.

We give a proof of determinacy for the big-step operational semantics on simple expressions, using structural induction on derivations. The proof is a little tricky. We also give a proof of determinacy for the small-step semantics using induction on derivations. We shall see that it is a little easier.

Proposition (Determinacy) For any simple expression E, if $E \Downarrow \mathsf{n}$ and $E \Downarrow \mathsf{n}'$ the $\mathsf{n} = \mathsf{n}'$.

Proof We prove this by induction on the structure of the *derivation* of $E \Downarrow \mathsf{n}$. This in itself requires a little thought. The property we wish to prove is:

> for any derivation D, *if* the conclusion of D is $E \Downarrow \mathsf{n}$, and it is also the case that $E \Downarrow \mathsf{n}'$ is derivable, then $\mathsf{n} = \mathsf{n}'$.

So, during this proof, we will consider

- a derivation D of a statement $E \Downarrow \mathsf{n}$; and
- another statement $E \Downarrow \mathsf{n}'$ which is *derivable*.

and try to show that $\mathsf{n} = \mathsf{n}'$. We apply induction to the derivation of D, and *not* to the derivation of $E \Downarrow \mathsf{n}'$.

Base Case: $E \Downarrow \mathsf{n}$ is an axiom. In this case, $E = \mathsf{n}$. We also have $E \Downarrow \mathsf{n}'$: that is, $\mathsf{n} \Downarrow \mathsf{n}'$. By examining the rules of the big-step semantics, it is clear that this can only be the case if $\mathsf{n} \Downarrow \mathsf{n}'$ is an axiom. It follows that $\mathsf{n} = \mathsf{n}'$.

Inductive step: If the derivation is not an axiom, it must have the form

$$\dfrac{\begin{array}{cc}\vdots & \vdots \\ D_1 & D_2 \\ \vdots & \vdots \\ E_1 \Downarrow \mathsf{n}_1 & E_2 \Downarrow \mathsf{n}_2\end{array}}{(E_1 + E_2) \Downarrow \mathsf{n}}$$

where $E = (E_1 + E_2)$ and $n = n_1 + n_2$. Call this whole derivation D. The inductive hypothesis applies to the subderivations D_1 and D_2. In the case of D_1, it says

since D_1 has conclusion $E_1 \Downarrow \mathsf{n}_1$, if a statement $E_1 \Downarrow \mathsf{n}''$ is derivable, then $\mathsf{n}_1 = \mathsf{n}''$.

We will use this in a moment.

We must show that if $E \Downarrow \mathsf{n}'$ then $\mathsf{n} = \mathsf{n}'$. So suppose that $E \Downarrow \mathsf{n}'$: that is, $(E_1 + E_2) \Downarrow \mathsf{n}'$ is derivable. This could not be derived using an axiom, so it must be the case that it was derived using the rule

$$\frac{E_1 \Downarrow \mathsf{n}_3 \quad E_2 \Downarrow \mathsf{n}_4}{(E_1 + E_2) \Downarrow \mathsf{n}'}$$

where $n' = n_3 + n_4$. This means that $E_1 \Downarrow \mathsf{n}_3$ and $E_2 \Downarrow \mathsf{n}_4$ is derivable. Using the induction hypothesis as spelled out above, we may assume that $\mathsf{n}_1 = \mathsf{n}_3$.

Similarly, applying the induction hypothesis to D_2, we may assume that $\mathsf{n}_2 = \mathsf{n}_4$. Since we have the equations $n = n_1 + n_2$ and $n' = n_3 + n_4$, it follows that $\mathsf{n} = \mathsf{n}'$ as required. ■

This is a tricky proof, because we not only do induction on the derivation of $E \Downarrow \mathsf{n}$, but we must also perform some analysis on the derivation of $E \Downarrow \mathsf{n}'$ too. Don't get too worried about understanding this proof, but do attempt to understand the technique because it crops up all the time in semantics.

Some Proofs about the Small-step Semantics

We have seen how to use induction to prove some simple facts about the big-step semantics of *SimpleExp*. In this section, we will see how to carry out similar proofs for the small-step semantics, both to reassure ourselves that we are on the right course and to make some intuitively obvious facts about our language into formal theorems.

Slide 14

Some properties of $\longrightarrow$

Strong determinacy If $E \longrightarrow E_1$ and $E \longrightarrow E_2$ then $E_1 = E_2$.

Determinacy If $E \longrightarrow^* \mathsf{n}$ and $E \longrightarrow^* \mathsf{n}'$ then $\mathsf{n} = \mathsf{n}'$.

Normalization For all E, there is some n such that $E \longrightarrow^* \mathsf{n}$.

An important property of the small-step semantics is that it is *deterministic* in a very strong sense: not only does each expression have at most one final answer, as in the big-step semantics, but also each expression can be evaluated to its final answer in exactly one way. Here, we give a proof of strong determinacy using structural induction on derivations. It is also possible to do a proof based on structural induction on the structure of expressions.

(Strong Determinacy) If $E \longrightarrow E_1$ and $E \longrightarrow E_2$ then $E_1 = E_2$.

Proof We use structural induction on the *derivation* of $E \longrightarrow E_1$.

Base Case The axiom for this semantics is the case where E is $(\mathsf{n}_1 + \mathsf{n}_2)$ and E_1 is n_3, where $n_3 = n_1 + n_2$. Consider the derivation that $E \longrightarrow E_2$: that is, $(\mathsf{n}_1 + \mathsf{n}_2) \longrightarrow E_2$. If this derivation is just an axiom, then E_2 must be n_3 as required. Otherwise, the last rule of this derivation is either

$$\frac{\mathsf{n}_1 \longrightarrow E'}{(\mathsf{n}_1 + \mathsf{n}_2) \longrightarrow (E' + \mathsf{n}_2)}$$

or

$$\frac{\mathsf{n}_2 \to E'}{(\mathsf{n}_1 + \mathsf{n}_2) \to (\mathsf{n}_1 + E')}$$

This implies that there is a derivation of $\mathsf{n}_1 \to E'$ or $\mathsf{n}_2 \to E'$, but it is easy to see that no such derivation exists. Therefore this can't happen!

Inductive Step (We just do the cases for $+$; the cases for $\times$ are similar.) If $E \to E_1$ was established by a more complex derivation, we must consider two possible cases, one for each rule that may have been used last in the derivation.

1. For some E_3, E_4 and E_3', it is the case that $E = (E_3 + E_4)$ and the last line of the derivation is

$$\frac{E_3 \to E_3'}{(E_3 + E_4) \to (E_3' + E_4)}$$

where $E_1 = (E_3' + E_4)$. We also know that $E \to E_2$: that is, $(E_3 + E_4) \to E_2$. Since $E_3 \to E_3'$, E_3 cannot be a numeral, so the last line in the derivation of this reduction must have the form

$$\frac{E_3 \to E_3''}{(E_3 + E_4) \to (E_3'' + E_4)}$$

where $E_2 = (E_3'' + E_4)$. But the derivation of $E_3 \to E_3'$ is a subderivation of the derivation of $E \to E_1$, so our inductive hypothesis allows us to assume that $E_3' = E_3''$. It therefore follows that $E_1 = E_2$.

2. In the second case, $E = (\mathsf{n} + E_3)$ and the derivation of $E \to E_1$ has the shape

$$\frac{E_3 \to E_3'}{(\mathsf{n} + E_3) \to (\mathsf{n} + E_3')}$$

as its last line where $E_1 = (\mathsf{n} + E_3')$. Again we know that E_3 is not a numeral, so the derivation that $E \to E_2$ must also end with a rule of the form

$$\frac{E_3 \to E_3''}{(\mathsf{n} + E_3) \to (\mathsf{n} + E_3'')}$$

where $E_2 = (\mathsf{n} + E_3'')$. Again we may apply the inductive hypothesis to deduce that $E_3' = E_3''$, from which it follows that $E_1 = E_2$. ■

This result says that the one-step relation is deterministic. Let us now see how from this we can prove that there can be at most one final answer.

First, we show that the k-step reduction relation, defined in slide 7, is also deterministic.

Corollary For every natural number k and every expression E, if $E \to^k E_1$ and $E \to^k E_2$ then $E_1 = E_2$.

Proof We prove this by *mathematical induction* on the natural number k. Let $P(k)$ be the property:

$E \to^k E_1$ and $E \to^k E_2$ implies $E_1 = E_2$.

By mathematical induction, to prove $P(n)$ holds for every n, we need to establish two facts.

Base Case here we establish $P(0)$: namely that, if $E \to^0 E_1$ and $E \to^0 E_2$ then $E_1 = E_2$. But this is trivial. Looking at the definition of $\to^k$ in slide 7 we see that the only possibility for E_1 and E_2 is that they are E itself, and therefore must be equal.

Inductive Case Here we assume the inductive hypothesis , namely $P(k)$. From this, we must prove $P(k+1)$, namely that if $E \to^{(k+1)} E_1$ and $E \to^{(k+1)} E_2$ then $E_1 = E_2$.

Again looking at the definition of $\to^{(k+1)}$ in slide 7 we know that there must exist some expressions E_1' and E_2' such that

$$E \to^k E_1' \to E_1$$
$$E \to^k E_2' \to E_2$$

But the inductive hypothesis gives that $E_1' = E_2'$, and the determinism of the one-step relation, proved in the previous lemma, gives the required $E_1 = E_2$. ■

This corollary leads directly to the determinacy of the final result.

Lemma (Determinacy) If $E \to^* \mathsf{n}$ and $E \to^* \mathsf{n}'$ then $\mathsf{n} = \mathsf{n}'$.

Proof The statement $E \to^* \mathsf{n}$ means that E reduces to n in some finite number of steps. So there is some natural number k such that $E \to^k \mathsf{n}$. Similarly, we have some k' such that $E \to^{k'} \mathsf{n}'$. Now either $k \leq k'$ or $k' \leq k$. Let's assume the former; the proof in the latter case is completely

symmetric. Then these derivations take the form

$$E \to^k \mathsf{n}$$
$$E \to^k E' \to^{(k'-k)} \mathsf{n}'$$

for some intermediary expression E'.
By the previous corollary, E' must be the same as n. According to the rules in slide 7, no reductions can be made from numerals. So the reduction $E' \to^{(k'-k)} \mathsf{n}'$ must be the trivial one $\mathsf{n} \to^0 \mathsf{n}'$. In other words, $\mathsf{n} = \mathsf{n}'$. ■
We now know that every term reaches at most one final answer; of course, for this simple language, we can show that *normalization* for the one-step semantics also holds: that is, there is a final answer for every expression.

Lemma (Normalization) For all E, there is some n such that $E \to^* \mathsf{n}$.

Proof By induction (!!!) on the structure of E.

Base Case E is a numeral n. Then $\mathsf{n} \to^* \mathsf{n}$ as required.

Inductive Step (We just do the cases for $+$; the cases for $\times$ are similar.) E is $(E_1 + E_2)$. By the inductive hypothesis, we have numbers n_1 and n_2 such that $E_1 \to^* \mathsf{n}_1$ and $E_2 \to^* \mathsf{n}_2$. For each step in the reduction

$$E_1 \to E_1' \to E_1''... \to \mathsf{n}_1$$

applying the rule for reducing the left argument of an addition gives

$$(E_1 + E_2) \to (E_1' + E_2) \to (E_1'' + E_2)... \to (\mathsf{n}_1 + E_2).$$

Applying the other rule to the sequence for $E_2 \to^* \mathsf{n}_2$ allows us to deduce that

$$(\mathsf{n}_1 + E_2) \to^* (\mathsf{n}_1 + \mathsf{n}_2) \to \mathsf{n}_3$$

where $n_3 = n_1 + n_2$. Hence $(E_1 + E_2) \to^* \mathsf{n}_3$. ■

Corollary For every expression E, there is exactly one n such that $E \to^* \mathsf{n}$.

We now know that our one-step semantics computes exactly one final answer for any given expression. We expect that the final answers given by the one-step and big-step semantics should agree, and indeed they do.

Theorem For any expression E,

$$E \Downarrow \mathsf{n} \text{ if and only if } E \rightarrow^* \mathsf{n}.$$

Exercise Prove this theorem by induction on the structure of expressions.