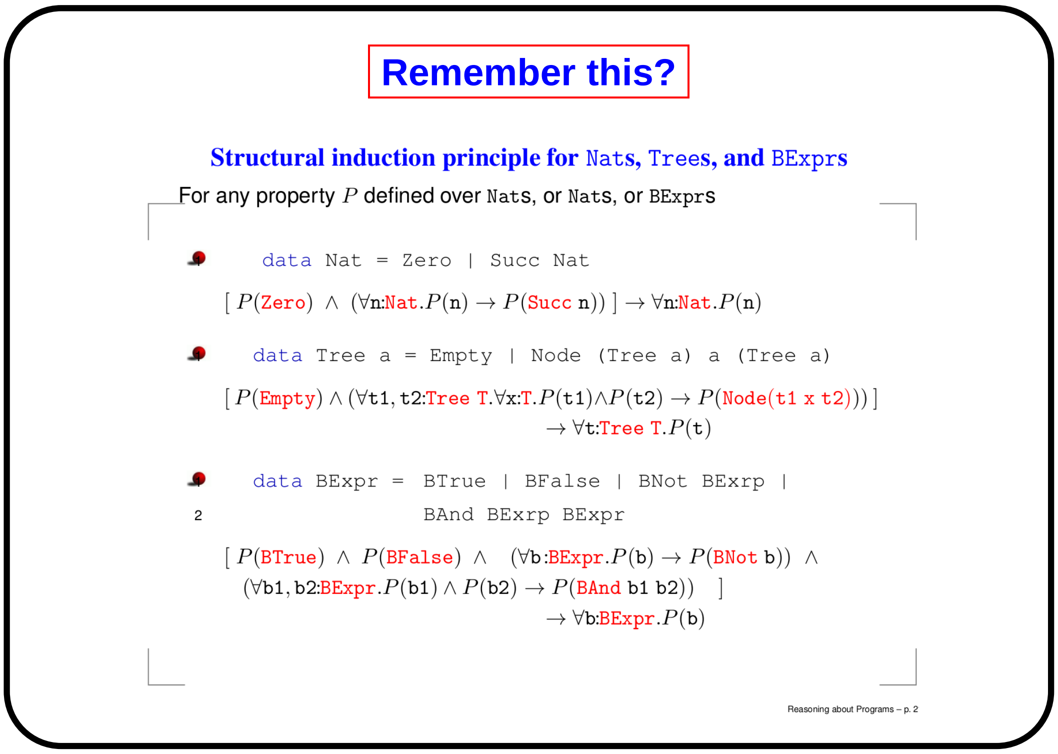
Slide 1



Remember this?

Structural induction principle for `Nats`, `Trees`, and `BExprs`

For any property P defined over `Nats`, or `Nats`, or `BExprs`

-
  ```
  data Nat = Zero | Succ Nat
  ```
  $[\, P(\texttt{Zero}) \;\wedge\; (\forall \texttt{n}{:}\texttt{Nat}.P(\texttt{n}) \rightarrow P(\texttt{Succ n}))\,] \rightarrow \forall \texttt{n}{:}\texttt{Nat}.P(\texttt{n})$

-
  ```
  data Tree a = Empty | Node (Tree a) a (Tree a)
  ```
  $[\, P(\texttt{Empty}) \wedge (\forall \texttt{t1}, \texttt{t2}{:}\texttt{Tree T}.\forall \texttt{x}{:}\texttt{T}.P(\texttt{t1}) \wedge P(\texttt{t2}) \rightarrow P(\texttt{Node(t1 x t2)}))\,]$
  $\rightarrow \forall \texttt{t}{:}\texttt{Tree T}.P(\texttt{t})$

-
  ```
  data BExpr = BTrue | BFalse | BNot BExrp |
  2              BAnd BExrp BExpr
  ```
  $[\, P(\texttt{BTrue}) \;\wedge\; P(\texttt{BFalse}) \;\wedge\; (\forall \texttt{b}{:}\texttt{BExpr}.P(\texttt{b}) \rightarrow P(\texttt{BNot b})) \;\wedge$
  $(\forall \texttt{b1}, \texttt{b2}{:}\texttt{BExpr}.P(\texttt{b1}) \wedge P(\texttt{b2}) \rightarrow P(\texttt{BAnd b1 b2}))\;\;]$
  $\rightarrow \forall \texttt{b}{:}\texttt{BExpr}.P(\texttt{b})$

Reasoning about Programs – p. 2

Induction

Last year, in Course 141: ‘Reasoning about Programs’, you learned about using induction to prove properties about Haskell programs.

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 refresh our memories about this technique.

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 2

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*.

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.

Last year, in ‘Reasoning about Programs’, this principle was written like this:

$$[P(0) \wedge (\forall k : \mathbb{N}.(P(k) \rightarrow P(k+1)))] \rightarrow \forall n : \mathbb{N}.P(n)$$

Slide 3

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 4

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 5

Writing an Inductive Proof

To prove $\forall n.P(n)$ by induction on the natural numbers:

Base Case: $n = 0$

- Prove that $P(0)$ holds, any way we like.

Inductive Case: $n = k + 1$ for some k.

- Assume $P(k)$ (The Inductive Hypothesis)
- Prove $P(k + 1)$ using that assumption.

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 built the number you're really interested in.

Example

Here is an 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 Case: Here the inductive hypothesis for parameter k, written IH for 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 the 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 $f(k)$. This function is 'unique' in the sense that 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 6

Definition by induction

We can define a function f on natural numbers by:

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

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

Slide 7

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 7 gives an inductive definition of the factorial function over the natural numbers. This is exactly like the sort of Haskell program that you already know how to prove things about! Slide 8 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 $E \longrightarrow E'$ over expressions. Suppose we wanted to define what is the effect of n reduction steps, for any natural number n. This would mean defining a family of relations $\longrightarrow^n$, one for each natural number n. Intuitively, $E \longrightarrow^n E'$ is supposed to mean that by applying exactly n computation steps to E we obtain E'.

Slide 8

Multi-step Reductions in *SimpleExp*

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

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

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

In slide 8, the first point defines the relation $\longrightarrow^0$ outright. In zero steps an expression remains untouched, so $E \longrightarrow^0 E$ for every expression E. In the second clause, the relation $\longrightarrow^{(k+1)}$ is defined in terms of $\longrightarrow^k$. It says that E reduces to E' in $(k+1)$ steps if

- E reduces in k steps to an intermediary expression E'';
- this intermediary expression E'' reduces to E' in one step.

The principle of induction now says that each of the infinite collection of relations $\longrightarrow^n$ is 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

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

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 $P(\texttt{zero})$ holds.

Inductive Case: prove that, for all numbers K, $P(\texttt{succ}(K))$ holds, assuming the inductive hypothesis that $P(K)$ holds.
This is summarized in slide 9.

Slide 9

Structural view of Mathematical Induction

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

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

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

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

Inductive Case: prove that, for all numbers K, $P(\texttt{succ}(K))$ holds, assuming the inductive hypothesis that $P(K)$ holds.

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 10

A Syntax for Binary Trees

Binary trees are defined as elements of the following grammar:

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

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 $P(\mathtt{Node})$ holds;

Inductive Case: prove that, for all binary trees T_1 and T_2,

$$P(\mathtt{Branch}(T_1, T_2))$$

holds, assuming the inductive hypotheses that $P(T_1)$ and $P(T_2)$ hold.

Slide 11

Writing an Inductive Proof: Revisited

Say what you want to prove, and what you're doing induction over.

Base Case(s) For each base case X:

- Say what the base case X is.
- Prove that $P(X)$ holds, any way we like.

Inductive Case(s) For each inductive case I:

- Say what the inductive case I is.
- Say what the inductive hypotheses are (there may be more than one!).
- Prove $P(I)$, assuming the inductive hypotheses.

Slide 12

Writing an Inductive Proof About Trees

To prove $\forall T.P(T)$ by induction on binary trees:

Base Case

- The base case is where $T = \texttt{Node}$
- Prove that $P(\texttt{Node})$ holds, any way we like.

Inductive Case

- The inductive case is where $T = \texttt{Branch}(T_1, T_2)$
- There are two inductive hypotheses: IH1 is $P(T_1)$. IH2 is $P(T_2)$.
- Prove $P(\texttt{Branch}(T_1, T_2))$, assuming that $P(T_1)$ and $P(T_2)$ hold.

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:

$$E \in \mathit{SimpleExp} ::= n \mid E + E \mid E \times E$$

Recall that n ranges over the natural numbers $0, 1, 2, \ldots$ This means that, in this language, there are in fact an infinite number of indecomposable expressions; contrast this with the cases above, where `zero` is the only indecomposable natural number, and `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 $P(n)$ holds for every number n.

Inductive Case 1: prove that, for all E_1 and E_2, $P(E_1 + E_2)$ holds, assuming the inductive hypotheses that $P(E_1)$ and $P(E_2)$ hold.

Inductive Case 2: prove that, for all E_1 and E_2, $P(E_1 \times E_2)$ holds, assuming the inductive hypotheses that $P(E_1)$ and $P(E_2)$ hold.

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 three forms: numbers, sums and products.

- **numbers**, cannot be decomposed, so we have to prove $P(n)$ directly for each of them. This is the base case.
- **composite expressions** E_1+E_2 and $E_1 \times E_2$, can be decomposed into subexpressions E_1 and E_2. These are inductive cases: 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 $P(E_1 \times E_2)$.

The reason this principle holds is similar to the reason the principle of mathematical induction (over natural numbers) holds. If we can prove the base case for numbers, and the inductive cases for composite expressions then we know, for example:

- $P(12)$ and $P(7)$ and $P(3)$ and $P(8)$ and $\ldots$ all hold;

- therefore $P(12+7)$ and $P(3+8)$ and ... all hold;
- therefore $P((12+7)\times(3+8))$ and ... all hold;
- ...

Slide 13

Writing an Inductive Proof About *SimpleExp* (1)

To prove that $P(E)$ holds for all expressions $E \in$ *SimpleExp* by induction on the structure of expressions:

Base Cases:

- The base cases are $P(n)$ for all n.
- Prove $P(n)$ any way we like.

Inductive Case 1:

- The first inductive case is $P(E_1 + E_2)$.
- The inductive hypotheses are $P(E_1)$ and $P(E_2)$.
- Prove $P(E_1 + E_2)$ assuming $P(E_1)$ and $P(E_2)$.

Slide 14

Writing an Inductive Proof About *SimpleExp* (2)

Inductive Case 2:

- The second inductive case is $P(E_1 \times E_2)$.
- The inductive hypotheses are $P(E_1)$ and $P(E_2)$.
- Prove $P(E_1 \times E_2)$ assuming $P(E_1)$ and $P(E_2)$.

Slide 15

Some Properties of $\Downarrow$ for *SimpleExp*

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

for any expression E, if $E \Downarrow n_1$ and $E \Downarrow n_2$ then $n_1 = n_2$.

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

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

Both of these properties can be proved by induction on the structure of expressions.

We give the proof of determinacy here. Totality is left as an exercise (see Exercises: Induction I).

Proposition (Determinacy of $\Downarrow$) For every simple expression E and all numbers n_1, n_2 with $E \Downarrow n_1$ and $E \Downarrow n_2$, $n_1 = n_2$.

Proof. We wish to show $P(E)$ for all E, where

$$P(E) \equiv \forall n_1, n_2.\, E \Downarrow n_1 \wedge E \Downarrow n_2 \implies n_1 = n_2$$

Proceed by induction on the structure of the expresion E.

Base Case: We must prove that $P(n)$ holds for arbitrary number n. Assume that n_1 and n_2 are such that $n \Downarrow n_1$ and $n \Downarrow n_2$. By the derivation rules for $\Downarrow$, it must be that $n_1 = n$ and $n_2 = n$. Thus, $n_1 = n_2$ as required.

Inductive Case: We must prove $P(E_1 + E_2)$, assuming $P(E_1)$ and $P(E_2)$ as inductive hypotheses. Specifically, the inductive hypotheses are:

$$\begin{array}{l} \forall n_{1,1}, n_{1,2}.\, E_1 \Downarrow n_{1,1} \wedge E_1 \Downarrow n_{1,2} \implies n_{1,1} = n_{1,2} \\ \forall n_{2,1}, n_{2,2}.\, E_2 \Downarrow n_{2,1} \wedge E_2 \Downarrow n_{2,2} \implies n_{2,1} = n_{2,2} \end{array}$$

Suppose that $E_1 + E_2 \Downarrow n_1$ and $E_1 + E_2 \Downarrow n_2$. The big step rules for deriving these require that $E_1 \Downarrow n_{1,1}$ and $E_2 \Downarrow n_{2,1}$ for some $n_{1,1}$ and $n_{2,1}$ with $n_1 = n_{1,1} \underline{+} n_{2,1}$. Similarly, we have $E_1 \Downarrow n_{1,2}$ and $E_2 \Downarrow n_{2,2}$ for some $n_{1,2}$ and $n_{2,2}$ with $n_2 = n_{1,2} \underline{+} n_{2,2}$. Now the inductive hypotheses imply that $n_{1,1} = n_{1,2}$ and $n_{2,1} = n_{2,2}$. Therefore, $n_1 = n_2$, as required.

Inductive Case: We must prove $P(E_1 \times E_2)$, assuming $P(E_1)$ and $P(E_2)$ as inductive hypotheses. This case follows the same pattern as the previous case, so we omit the details. $\square$

(Later, we shall see how to do this proof by induction on the structure of *derivations*, which, in some cases, leads to more concise proofs.)

Defining Functions over Simple Expressions

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

Slide 16

Definition by Induction for *SimpleExp*

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

- define $f(n)$ directly, for each number 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 17

The function den

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

- $\mathsf{den}(n) = n$ for each number 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,

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

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 numbers 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 $E \Downarrow n$ 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 $\texttt{3+(2+1)} \Downarrow \texttt{6}$:

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

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

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

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

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

Here, we see a derivation whose last line is

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

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 case.
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

$$\frac{}{n \Downarrow n}$$

satisfies property P.

Inductive Cases: For each rule of the form

$$\frac{h_1 \quad \cdots \quad h_n}{c}$$

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 already gave a proof of determinacy of $\Downarrow$ by induction on the structure of expressions. Now let's do the proof by induction on the structure of derivations.
Proposition (Determinacy of $\Downarrow$) For every simple expression E and all numbers n_1, n_2 with $E \Downarrow n_1$ and $E \Downarrow n_2$, $n_1 = n_2$.

Proof. We wish to show $P(E, n_1)$ for all E, n_1 with $E \Downarrow n_1$, where

$$P(E, n_1) \equiv \forall n_2. E \Downarrow n_2 \implies n_1 = n_2$$

Proceed by induction on the structure of the derivation of $E \Downarrow n_1$.
Base Case: The derivation has the form

$$(\textsc{b-num})\ \frac{}{n_1 \Downarrow n_1}$$

with $E = n_1$. We must prove $P(n_1, n_1)$. Suppose that $n_1 \Downarrow n_2$ also. The only rule that gives such a derivation is B-NUM, which ensures that $n_1 = n_2$ as required.
Inductive Case: The derivation has the form

$$(\textsc{b-add})\ \frac{E_1 \Downarrow n_{1,1} \qquad E_2 \Downarrow n_{2,1}}{E_1 + E_2 \Downarrow n_1}$$

with $E = E_1 + E_2$ and $n_1 = n_{1,1} + n_{2,1}$. For the inductive hypotheses, we have $P(E_1, n_{1,1})$ and $P(E_2, n_{2,1})$, namely:

$$\forall n_{1,2}. E_1 \Downarrow n_{1,2} \implies n_{1,1} = n_{1,2}$$
$$\forall n_{2,2}. E_2 \Downarrow n_{2,2} \implies n_{2,1} = n_{2,2}$$

Suppose that $E_1 + E_2 \Downarrow n_2$. The only rule that gives such a derivation is B-ADD:

$$(\textsc{b-add})\ \frac{E_1 \Downarrow n_{1,2} \qquad E_2 \Downarrow n_{2,2}}{E_1 + E_2 \Downarrow n_2}\ n_2 = n_{1,2} \underline{+} n_{2,2}$$

Now the inductive hypotheses imply that $n_{1,1} = n_{1,2}$ and $n_{2,1} = n_{2,2}$. Therefore, $n_1 = n_2$, as required.
Inductive Case: The last rule in the derivation is B-MULT. This case follows the same pattern as for B-ADD, so we omit the details. □

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 18

Some properties of $\longrightarrow$ for *SimpleExp*

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

Confluence If $E \longrightarrow^* E_1$ and $E \longrightarrow^* E_2$ then there exists E' such that $E_1 \longrightarrow^* E'$ and $E_2 \longrightarrow^* E'$.

Unique Answer If $E \longrightarrow^* n_1$ and $E \longrightarrow^* n_2$ then $n_1 = n_2$.

Normal forms The normal forms are exactly the numbers: either $E = n$ for some n or $E \longrightarrow E'$ for some E'.

Normalization There are no infinite sequences of expressions $E_1, E_2, E_3, \ldots$ such that, for all i, $E_i \longrightarrow E_{i+1}$. (Every evaluation path eventually reaches a normal form.)

An important property of the small-step semantics is that it is *confluent*: any two evaluation paths can eventually converge to the same state. This implies that an expression can be reduced to at most one number (since different numbers can certainly not be reduced to each other). In fact, $\longrightarrow$ has the stronger *determinacy* property, which ensures that it is confluent.
Here, we give a proof of determinacy using structural induction on derivations. It is also possible to do a proof by induction on the structure of expressions.
(Determinacy of $\longrightarrow$) If $E \longrightarrow E_1$ and $E \longrightarrow E_2$ then $E_1 = E_2$.

Proof. We wish to show $P(E, E_1)$ for all E, E_1 with $E \longrightarrow E_1$, where

$$P(E, E_1) \equiv \forall E_2 . E \rightarrow E_2 \implies E_1 = E_2$$

Proceed by induction on the structure of the derivation of $E \rightarrow E_1$.
Base Case: The derivation has the form

$$(\textsc{s-add})\ \frac{}{n_1 + n_2 \rightarrow n_3}$$

with $E = n_1 + n_2$, $E_1 = n_3$ and $n_3 = n_1 \underline{+} n_2$. Only the S-ADD rule applies to evaluate E, so if $E \rightarrow E_2$ then it must be that $E_2 = n_3'$ for some n_3' with $n_3' = n_1 \underline{+} n_2$. But there is only one such number, so $E_2 = n_3 = E_1$, as required.
Inductive Case: The derivation has the form

$$(\textsc{s-left})\ \frac{F_1 \rightarrow F_1'}{F_1 + F_2 \rightarrow F_1' + F_2}$$

with $E = F_1 + F_2$ and $E_1 = F_1' + F_2$. For the inductive hypothesis, we have $P(F_1, F_1')$, namely:

$$\forall F_1''.\, F_1 \rightarrow F_1'' \implies F_1' = F_1''$$

Suppose that E_2 is such that $E \rightarrow E_2$. It must be that F_1 is not a number (since $F_1 \rightarrow F_1'$), so $E \rightarrow E_2$ must be derived by the S-LEFT rule as follows:

$$(\textsc{s-left})\ \frac{F_1 \rightarrow F_1''}{F_1 + F_2 \rightarrow F_1'' + F_2}$$

for some F_1'' with $E_2 = F_1'' + F_2$. Now, since the derivation requires $F_1 \rightarrow F_1''$, the inductive hypothesis gives us that $F_1' = F_1''$. Therefore $E_1 = F_1' + F_2 = F_1'' + F_2 = E_2$, as required.
Inductive Case: The derivation has the form

$$(\textsc{s-right})\ \frac{F \rightarrow F'}{n + F \rightarrow n + F}$$

with $E = n + F$ and $E_1 = n + F'$. For the inductive hypothesis, we have $P(F, F')$, namely:

$$\forall F''.\, F \rightarrow F'' \implies F' = F''$$

Suppose that E_2 is such that $E \rightarrow E_2$. It must be that F is not a number (since $F \rightarrow F'$), so $E \rightarrow E_2$ must be derived by the S-RIGHT rule as follows:

$$(\textsc{s-right})\ \frac{F \rightarrow F''}{n + F \rightarrow n + F''}$$

for some F'' with $E_2 = F_1'' + F_2$. Now, since the derivation requires $F \longrightarrow F''$, the inductive hypothesis gives us that $F' = F''$. Therefore $E_1 = n + F' = n + F'' = E_2$, as required.

There are also base and inductive cases for $\times$, but they are similar to those for $+$, so we omit the details. $\square$

This result says that the one-step relation is deterministic. Let us now see how from this we can prove that the relation is confluent.

Proposition (Confluence of $\longrightarrow$) If $E \longrightarrow^* E_1$ and $E \longrightarrow^* E_2$ then there exists E' such that $E_1 \longrightarrow^* E'$ and $E_2 \longrightarrow^* E'$.

Propositions involving $\longrightarrow^*$ are typically proved by induction on the number of evaluation steps.

Proof. Note that $E \longrightarrow^* E_1$ if and only if $E \longrightarrow^n E_1$ for some n. Thus, it is sufficient to prove $P(n)$ for all natural numbers n, where

$$\begin{aligned} P(n) \equiv \forall E, E_1, E_2.\, E \longrightarrow^n E_1 \wedge E \longrightarrow^* E_2 \implies \\ \exists E'.\, E_1 \longrightarrow^* E' \wedge E_2 \longrightarrow^* E' \end{aligned}$$

Proceed by induction on n.

Base Case: $n = 0$. Suppose $E \longrightarrow^0 E_1$ and $E \longrightarrow^* E_2$. Then $E = E_1$ and so $E_1 \longrightarrow^* E_2$. Let $E' = E_2$. We have $E_1 \longrightarrow^* E'$ and $E_2 \longrightarrow^* E'$, as required.

Inductive Case: For the inductive hypothesis, we assume $P(k)$, namely:

$$\begin{aligned} \forall E, E_1', E_2.\, E \longrightarrow^k E_1' \wedge E \longrightarrow^* E_2 \implies \\ \exists E''.\, E_1' \longrightarrow^* E'' \wedge E_2 \longrightarrow^* E'' \end{aligned}$$

Suppose that $E \longrightarrow^{k+1} E_1$ and $E \longrightarrow^* E_2$. We must have $E \longrightarrow^k E_1' \longrightarrow E_1$ for some E_1'. By the inductive hypothesis, there exists E'' with $E_1' \longrightarrow^* E''$ and $E_2 \longrightarrow^* E''$. It must be that either $E_1' = E''$ or $E_1' \longrightarrow E_1'' \longrightarrow^* E''$ for some E_1''. Consider each case.

- In the first case, let $E' = E_1$. We have that $E_1 \longrightarrow^* E'$. We also have that $E_2 \longrightarrow^* E'' = E_1' \longrightarrow E_1 = E'$, so $E_2 \longrightarrow^* E'$ as required.
- In the second case, let $E' = E''$. Since $E_1' \longrightarrow E_1$ and $E_1' \longrightarrow E_1''$, it must be that $E_1 = E_1''$ by determinacy of $\longrightarrow$. Hence, $E_1 \longrightarrow^* E'' = E'$. We also have that $E_2 \longrightarrow^* E'' = E'$, as required. $\square$

Of course, not every result needs to be proved by induction!

Corollary (Unique Answer for $\longrightarrow$) If $E \longrightarrow^* n_1$ and $E \longrightarrow^* n_2$ then $n_1 = n_2$.

Proof. Suppose that $E \longrightarrow^* n_1$ and $E \longrightarrow^* n_2$. By confluence, there is some E' with $n_1 \longrightarrow^* E'$ and $n_2 \longrightarrow^* E'$. Since numbers have no reductions (they are normal forms) it must be that $n_1 = E'$ and $n_2 = E'$. Hence $n_1 = n_2$, as required. $\square$

Slide 19

Connecting $\Downarrow$ and $\longrightarrow^*$ for *SimpleExp*

Exercise (see sheet)

For all E and n, $E \Downarrow n$ if and only if $E \longrightarrow^* n$.

Normalization and the equivalence of the big-step and small-step semantics are for you to do in Exercises: Induction I.