Slide 1

Normalisation

The evaluation relations $\longrightarrow_e$ and $\longrightarrow_b$ are normalising.

The execution relation $\longrightarrow_c$ is not normalising.

Specifically, we can have infinite loops. For example, the program $(\texttt{while true do skip})$ loops forever.

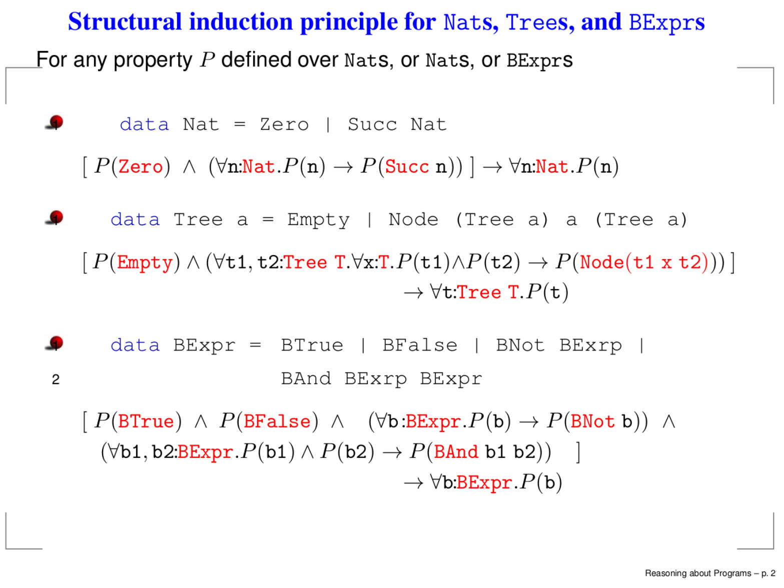
Let s be any state. We have the execution path

$$\langle \texttt{while true do skip}, s\rangle \longrightarrow_c^3 \langle \texttt{while true do skip}, s\rangle$$

where $\longrightarrow_c^3$ means three steps. Hence, we have an infinite execution path.

Slide 2

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

Structural Induction

Last year, in Course 141: 'Reasoning about Programs', you learned about using structural 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 3

What is structural 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*.

Slide 4

You can use structural 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 assertions like $E \Downarrow 4$: these derivations are finite trees of axioms and rules.

Mathematical Induction

The simplest form of induction, and probably the one most familiar to you, 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 5

Mathematical Induction

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

$$P(0) \wedge [\forall k \in \mathbb{N}.\ P(k) \Rightarrow P(k+1)]$$

holds then

$$\forall n \in \mathbb{N}.\ P(n)$$

holds.

Slide 6

Writing an Inductive Proof

Prove $\forall n \in \mathbb{N}.P(n)$ holds by induction on the natural numbers:

Base Case:

- The base case is $P(0)$.
- Prove $P(0)$ any way we like.

Inductive Case:

- The inductive case is $P(k+1)$.
- Assume $P(k)$ holds (this is called the **inductive hypothesis**, also sometimes called the **induction hypothesis**).
- Prove that $P(k+1)$ holds using the inductive hypothesis.

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.

Slide 7

A Property of the Natural Numbers

For all $n \in \mathbb{N}$,

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

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 is $P(0)$. The property $P(0)$ is: the sum of numbers from 0 to 0 inclusive is equal to $\frac{0^2+0}{2}$. This property obviously holds, so the base case is true.

Inductive Case: The inductive case is $P(k+1)$, with the inductive hypothesis $P(k)$ which states that: the sum of numbers from 0 to k inclusive is equal to $\frac{k^2+k}{2}$. Assuming this inductive hypothesis, we must prove that $P(k+1)$ holds: that is, 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:

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

Definition by Mathematical Induction

As well as using mathematical induction to prove properties of natural numbers, we can use it to define functions which operate on natural numbers. Just as a proof by mathematical induction proves a property $\forall n \in \mathbb{N}.\ P(n)$ by considering the base case $P(0)$ and the inductive case $P(k+1)$ assuming that the inductive hypothesis $P(k)$ holds, so the definition of a function f by induction works by giving the definition of $f(0)$ directly, and building the definition of $f(k+1)$ out of the definition 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.

Slide 8

Definition by mathematical Induction

Define a function f on natural numbers, using mathematical induction, by:

Base Case: define $f(0)$ directly, any way we can.

Inductive case: define $f(k+1)$ using the definition of $f(k)$.

Slide 9

Inductive Definition of Factorial

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

Base case: $\texttt{fact}(0) = 1$;

Inductive case: $\texttt{fact}(k+1) = (k+1) \times \texttt{fact}(k)$.

$\texttt{fact}(n)$ is often written $n!$.

Exercise: Prove that $n! > 2^n$ for $n \geq 4$ by mathematical induction.

For example, slide 9 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 write and prove things about!

Exercise: Prove that $n! > 2^n$ for $n \geq 4$ by mathematical induction.

Proof: We need to prove the property $P(n) = n! > 2^n$ for $n \geq 4$.

First, notice that the property does *not* hold for $n = 0, 1, 2, 3$. The base case is $P(4)$. We need to calculate $4!$ and 2^n and compare them: $4! = 24$; $2^4 = 16$; and $24 > 16$. Hence, $P(4)$ holds.

The inductive case is $P(k+1)$. We can assume the inductive hypothesis that:

$$P(k) \text{ is } k! > 2^k \text{ for } k \geq 4.$$

Now to show that $P(k+1)$ holds, multiply both sides of the inductive hypothesis by $k+1$:

$$k!(k+1) > 2^k(k+1).$$

The left side is equal to (k + 1)!. For $k \geq 4$, we have $k+1 > 2$. Multiply both sides of this inequality by 2^k to obtain

$$2^k(k+1) > 2(2^k).$$

The above inequality can be written $2^k(k+1) > 2^{k+1}$. We have proved that $(k+1)! > 2^k(k+1)$ and $2^k(k+1) > 2^{k+1}$. We therefore have $(k+1)! > 2^{k+1}$, as required.
Slide 29 contains another definitional use of mathematical induction, one that some students find quite hard. We have already defined the one-step operational semantics on simple expressions, written $E \rightarrow E'$. 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 $\rightarrow^n$, one for each natural number n. Intuitively, $E \rightarrow^n E'$ is supposed to mean that by applying exactly n computation steps to E we obtain E'.

Slide 10

Many Steps of Evaluation

Given a relation $\rightarrow$, we define a new relation $\rightarrow^*$ by:

$E \rightarrow^* E'$ holds if and only if either $E = E'$ (so no steps of evaluation are needed to get from E to E') or there is a finite sequence

$$E \rightarrow E_1 \rightarrow E_2 \ldots \rightarrow E_k \rightarrow E'.$$

This relation $\rightarrow^*$ is called the *reflexive transitive closure* of $\rightarrow$.

For our expressions, we say that number n is the final answer of E if $E \rightarrow^* n$.

Slide 11

Multi-step Reductions in *SimpleExp*

The relation $E \rightarrow^n E'$ is defined using mathematical induction by:

Base case: $E \rightarrow^0 E$ for every simple expression $E \in$ *SimpleExp*;

Inductive Case: For every $E, E' \in$ *SimpleExp*, $E \rightarrow^{k+1} E'$ if and only if there is some E'' such that

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

Definition For every $E, E' \in$ *SimpleExp*, define $E \rightarrow^* E'$ by:

$$E \rightarrow^* E' \Leftrightarrow \exists n.\, E \rightarrow^n E'.$$

In slide 29, the base case defines the relation $\rightarrow^0$ outright. In zero steps an expression remains untouched, so $E \rightarrow^0 E$ for every expression E. In the inductive case, the relation $\rightarrow^{k+1}$ is defined in terms of $\rightarrow^k$. It says that E reduces to E' in $k+1$ steps if and only 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, for all $n \in \mathbb{N}$, the relation $\rightarrow^n$ is well-defined. This means that the relation $E \rightarrow^* E'$ is well-defined. Compare this definition with the informal defintion we gave earlier in the course.

Structural Induction for Natural Numbers

We said in the last section that mathematical induction is a valid principle because every natural number can be 'built' using zero 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:

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

Here $\mathtt{succ}$, short for *successor*, should be thought of as the operation of adding one. Therefore, the number 0 is represented by $\mathtt{zero}$ 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 $\mathtt{zero}$ and repeatedly applying $\mathtt{succ}$. 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 steps:

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

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

Slide 12

Structural Induction for Natural Numbers

The natural numbers can be defined as elements of the following grammar:

$$N \in \mathit{Nat} ::= \mathtt{zero} \,|\, \mathtt{succ}(N).$$

Slide 13

Writing an Inductive Proof: Revisited

Prove that property $P(N)$ holds, for every number $N \in$ *Nat*, by induction on the structure of N:

Base Case:

- The base case is $P(\text{zero})$.
- Prove $P(\text{zero})$ holds any way we like.

Inductive Case:

- The inductive case is $P(\mathtt{succ}(K))$.
- Assume the inductive hypothesis that $P(K)$ holds.
- Prove that $P(\mathtt{succ}(K))$ holds using the inductive hypothesis.

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 by structural induction on the natural numbers defined structurally, we must

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

No go back to the factorial example. The inductive definition and reasoning are very similar.

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 14

A Syntax for Binary Trees

Binary trees are defined as elements of the following grammar:

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

Note the similarity with the structural definition of the natural numbers.

Slide 15

Inductive Definitions over Trees

Define the function, **leaves**, which take a binary tree as an argument and returns the number of leaf nodes in a tree, by:

Base Case: $\texttt{leaves}(\texttt{Node}) = 1$

Inductive Case:

$\texttt{leaves}(\texttt{Branch}(T_1, T_2)) = \texttt{leaves}(T_1) + \texttt{leaves}(T_2)$.

Define function, **branches**, which counts the number of $\texttt{Branch}(\_,\_)$ nodes in a tree, by:

Base Case: $\texttt{branches}(\texttt{Node}) = 0$

Inductive Case:

$\texttt{branches}(\texttt{Branch}(T_1, T_2)) = \texttt{branches}(T_1) + \texttt{branches}(T_2) + 1$.

Slide 16

Exercise

Prove by induction on the structure of trees that, for any tree T,

$$\texttt{leaves}(T) = \texttt{branches}(T) + 1.$$

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

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

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

We give a full proof of this exercise in the answers to the induction exercise sheet.

Slide 17

Writing an Inductive Proof: Revisited

Prove $P(X)$ holds, for all $X \in$ *SomeSet*, by induction on the structure of X:

Base Cases

- Identify the base cases $P(X)$.
- For each base case , prove that $P(X)$ holds any way we like.

Inductive Cases

- Identify the inductive cases $P(X)$
- For each inductive case, say what the inductive hypotheses are.
- For each inductive case $P(X)$, prove that $P(X)$ holds assuming the inductive hypothesis.

Slide 18

Writing an Inductive Proof About Trees

Prove that $P(T)$ holds, for all binary trees $T \in$ *BinaryTree*, by induction on the structure of binary trees:

Base Case

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

Inductive Case

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

Structural Induction for Simple Expressions

The syntax of our illustrative language *SimpleExp* also gives a collection of structured, finite, but arbitrarily large objects over which structural induction may be used. The syntax is repeated below:

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

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 $P(E_1 \times E_2)$.

Slide 19

Syntax of Simple Expressions

Simple expressions are defined as elements of the following grammar:

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

where $n \in \mathbb{N}$ ranges over the natural numbers $0, 1, 2, \ldots$.

We can add more operations if we need to.

Notice the similarity with the structural definitions of the natural numbers and binary trees.

Slide 20

Writing an Inductive Proof About *SimpleExp* (1)

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 $P(n)$, for all 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 21

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 22

Some Properties of $\Downarrow$ for *SimpleExp*

Determinacy says that a simple expression cannot evaluate to more than one answer: for any expression E and natural numbers n_1, n_2,

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 natural number 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 answers to exercise sheet on induction).

Theorem 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:

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

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. □

(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.)

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

Slide 23

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 $\mathtt{den}(E)$ which is the 'meaning' or the 'final answer' for E.

Slide 24

The function den

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

- $\mathtt{den}(n) = n$ for each number n;
- $\mathtt{den}(E_1 + E_2) = \mathtt{den}(E_1)\underline{+}\mathtt{den}(E_2)$;
- $\mathtt{den}(E_1 \times E_2) = \mathtt{den}(E_1)\underline{\times}\mathtt{den}(E_2)$;

Exercise For every simple expression E and number n,

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

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

Structural induction on derivations will not be examined.

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* $\texttt{3+(2+1)} \Downarrow \texttt{6}$, and the two *subderivations*, which are

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

The first subderivation is an axiom which cannot be broken down any further. The second subderivation has its own conclusion $2+1 \Downarrow 3$, and subderivations

$$\text{(B-NUM)}\;\dfrac{}{2 \Downarrow 2} \qquad \text{(B-NUM)}\;\dfrac{}{1 \Downarrow 1}$$

which cannot be broken down further.
We can give notation for these derivations. For example, the derivation of $3+(2+1) \Downarrow 6$ can be annotated by *derivation expressions* as follows (removing rule names for space reasons):

$$\dfrac{\dfrac{}{\texttt{num}(3) : 3 \Downarrow 3} \quad \dfrac{\dfrac{}{\texttt{num}(2) : 2 \Downarrow 2} \quad \dfrac{}{\texttt{num}(1) : 1 \Downarrow 1}}{\texttt{add}(\texttt{num}(2), \texttt{num}(1)) : 2+1 \Downarrow 3}}{\texttt{add}(\texttt{num}(3), \texttt{add}(\texttt{num}(2), \texttt{num}(1))) : \texttt{3+(2+1)} \Downarrow \texttt{6}}$$

More formally, we can define *derivation expressions* for our big-step operational semantics for simple expressions by:

$$d \in \textit{SimpleExp} ::= \texttt{num}(n) \mid \texttt{add}(d, d) \mid \texttt{mult}(d, d) \mid \ldots$$

where $n \in \mathbb{N}$ ranges over the natural numbers $0, 1, 2, \ldots$. We annotate the axioms and rules of the big-step operational semantics for simple expressions with these derivation expressions:

$$\text{(B-NUM)}\ \frac{}{\mathtt{num}(n) : n \Downarrow n}$$

$$\text{(B-ADD)}\ \frac{d_1 : E_1 \Downarrow n_1 \quad d_2 : E_2 \Downarrow n_2}{\mathtt{add}(d_1, d_2) : E_1 + E_2 \Downarrow n_3} n_1 \underline{+} n_2 = n_3$$

$$\text{(B-MULT)}\ \frac{d_1 : E_1 \Downarrow n_1 \quad d_2 : E_2 \Downarrow n_2}{\mathtt{mult}(d_1, d_2) : E_1 \times E_2 \Downarrow n_3} n_1 \underline{\times} n_2 = n_3$$

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 the property holds for every derivation expression that cannot be broken into smaller parts: that is, the derivation expressions used for the axioms. In the case of the big-step semantics for simple expressions, we must prove that $P(\mathtt{num}(n))$ holds for every number n.

Inductive Cases: For every compound derivation expression, assume that the property holds for the subderivations and prove that the property holds for the compound expression. In the case of the big-step semantics for simple expressions, we must prove that $P(\mathtt{add}(d1, d2))$ and $P(\mathtt{mult}(d_1, d_2))$ hold, in each case assuming the induction hypothesis that $P(d_1)$ and $P(d_2)$ hold.
We already gave a proof of determinacy of the big-step semantics of simple expressions 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 $P(d)$ for all derivation expressions d, where

$$P(d) \equiv \forall E, n_1, n_2, d_2 .\, d : E \Downarrow n_1 \wedge d_2 : E \Downarrow n_2 \implies d = d_2 \wedge n_1 = n_2$$

Proceed by induction on the structure of the derivation expressions.
Base Case: We must prove that $P(\mathtt{num}(n))$ holds for arbitrary number n, so assume that $\mathtt{num}(n) : E \Downarrow n_1$ and $d_2 : E \Downarrow n_2$ for arbitrary n_1, n_2, d_2. The only way that $d_2 : n \Downarrow n_2$ is if $d_2 = \mathtt{num}(n)$ and $n = n_2$. Thus, $P(\mathtt{num}(n))$ holds.

Inductive Case for +: For $d = \mathtt{add}(d_1', d_2')$, we must prove that $P(\mathtt{add}(d_1', d_2'))$ holds assuming the inductive hypothesis that $P(d_1')$ and

$P(d'_2)$ hold. Assume $\mathtt{add}(d'_1, d'_2) : E \Downarrow n_1$ and $d_2 : E \Downarrow n_2$. By inspecting the derivation rules we know that

$$\text{(B-ADD)}\ \frac{d'_1 : E_1 \Downarrow n_{1,1} \quad d'_2 : E_2 \Downarrow n_{1,2}}{\mathtt{add}(d'_1, d'_2) : E_1 + E_2 \Downarrow n_1}$$

with $E = E_1 + E_2$ and $n_1 = n_{1,1} \underline{+} n_{1,2}$.
Now given $d_2 : E \Downarrow n_2$ and $E = E_1 + E_2$, we know that there exists $d''_1, d''_2, n_{2,1}, n_{2,2}$ such that $d_2 = \mathtt{add}(d''_1, d''_2)$, $d''_1 : E_1 \Downarrow n_{2,1}$ and $d''_2 : E_2 \Downarrow n_{2,2}$ and $n_{2,1} \underline{+} n_{2,2} = n_2$. So we have $d'1 : E_1 \Downarrow n_{1,1}$ and $d'' : E_1 \Downarrow n_{2,1}$ and $P(d'_1)$ holds by the induction hypothesis. This means that $d'_1 = d''_1$ and $n_{1,1} = n_{2,1}$.
Similarly, we have $d'_2 : E_2 \Downarrow n_{1,2}$ and $d''_2 : E_2 \Downarrow n_{2,2}$ and $P(d'_2)$ holds by the induction hypothesis. This means that $d'_2 = d''_2$ and $n_{1,2} = n_{2,2}$.
Together, it follows that $d = \mathtt{add}(d'_1, d'_2) = \mathtt{add}(d''_1, d''_2) = d_2$, and $n_1 = n_{1,1} \underline{+} n_{1,2} = n_{2,1} \underline{+} n2, 2 = n_2$ as required.

Inductive Case for $\times$: For $d = \mathtt{mult}(d'_1, d'_2)$, the result is proved similarly. $\square$

Aside: Notice that we have strengthened the property $P(d)$ by also requiring that the derivation expressions are equal. This strengthening is sometimes necessary for the induction to go through. Why is it necessary in this case?

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 25

Some properties of $\rightarrow$ for *SimpleExp*

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

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

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

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

Normalization There are no infinite sequences of expressions $E_1, E_2, E_3, \ldots$ such that, for all i, $E_i \rightarrow E_{i+1}$. (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, $\rightarrow$ has the stronger *determinacy* property, which ensures that it is confluent.
Here, we give a proof of determinacy using structural induction on expressions. It is also possible to do a proof by induction on the structure of derivations, analogous to the big-step case.

(Determinacy of $\rightarrow$) If $E \rightarrow E_1$ and $E \rightarrow E_2$ then $E_1 = E_2$.

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

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

Proceed by induction on the structure of E.
Base Case: When $E = n$ for arbitrary n, there is nothing to do as there is no evaluation step for n.

Inductive Case for $+$: When $E = E_1' + E_2'$, we need to show that $P(E_1' + E_2')$ holds, assuming the inductive hypothesis that $P(E_1')$ and $P(E_2')$ hold. Assume that $E_1' + E_2' \to E_1$ and $E_1' + E_2' \to E_2$ for arbitrary E_1 and E_2. Now there are three cases for how $E_1' + E_2' \to E_1$ is derived.
Case 1: Let $E_1' = n_1$ and $E_2' = n_2$. By inspecting the rules, we know that:

$$(\text{S-ADD})\ \frac{}{n_1 + n_2 \to n_3}$$

where $E_1 = n_3$ and $n_3 = n_1 \underline{+} n_2$. Only the S-ADD rule applies to evaluate $E = n_1 + n_2$, so if $n_1 + n_2 \to E_2$ 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.
Case 2: Let $E_1' = n$ and $E_2' = F$. By inspecting the rules, we know that:

$$(\text{S-RIGHT})\ \frac{F \to F'}{n + F \to n + F'}$$

where $E_1 = n + F'$. Only the S-RIGHT rule applies to evaluate $E = n + F$ so, if $n + F \to E_2$, it must be that $E_2 = n + F''$ for some F'' with $F \to F''$. We have $F \to F'$ and $F \to F''$. Since $E_2' = F$, by the inductive hypothesis, we know that $F' = F''$. Hence, $E_1 = n + F' = n + F'' = E_2$ as required.
Case 3: Let $E_1' = F_1$ and $E_2' = F_2$. By inspecting the rules, we know that

$$(\text{S-LEFT})\ \frac{F_1 \to F_1'}{F_1 + F_2 \to F_1' + F_2}$$

where $E_1 = F_1' + F_2$. Only the S-LEFT rule applies to evaluate $E = F_1 + F_2$ and F_1 is not a number as $F_1 \to F_1'$. It must therefore be the case that $E_2 = F_1'' + F_2$ and $F_1 \to F_1''$. We have $F_1 \to F_1'$ and $F_1 \to F_1''$ so, by the induction hypothesis for $E_1' = F_1$, we know that $F_1' = F_1''$. Hence, $E_1 = F_1' + F_2 = F_1'' + F_2 = E_2$, as required.

Inductive case for $\times$: Similar. $\square$

Aside: Notice that the inductive case for $+$ has a case analysis which highlights the three ways that $E_1' + E_2' \to E_1$ can be derived depending on the structure of $E_1' + E_2'$. It is possible to do a proof using structural induction on derivations, analogous to the proof given for the big-step semantics of simple expressions. With this proof, each derivation expression corresponds to one

rule, so we would not have this case split. Hence, the proof by induction on the structure of derivations is sometimes regarded as more natural.
Let us now see to prove confluence, making use of our result that $\rightarrow$ is deterministic.

Proposition (Confluence of $\rightarrow$) If $E \rightarrow^* E_1$ and $E \rightarrow^* E_2$ then there exists E' such that $E_1 \rightarrow^* E'$ and $E_2 \rightarrow^* E'$.
Propositions involving $\rightarrow^*$ are typically proved by induction on the number of evaluation steps.

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

$$P(n) \equiv \forall E, E_1, E_2. E \rightarrow^n E_1 \wedge E \rightarrow^* E_2 \implies \exists E'. E_1 \rightarrow^* E' \wedge E_2 \rightarrow^* E'$$

Proceed by mathematical induction on n.
Base Case: $n = 0$. Suppose $E \rightarrow^0 E_1$ and $E \rightarrow^* E_2$. Then $E = E_1$ and so $E_1 \rightarrow^* E_2$. Let $E' = E_2$. We have $E_1 \rightarrow^* E'$ and $E_2 \rightarrow^* E'$, as required.
Inductive Case: For the inductive hypothesis, we assume $P(k)$, namely:

$$\forall E, E_1', E_2. E \rightarrow^k E_1' \wedge E \rightarrow^* E_2 \implies \exists E''. E_1' \rightarrow^* E'' \wedge E_2 \rightarrow^* E''$$

Suppose that $E \rightarrow^{k+1} E_1$ and $E \rightarrow^* E_2$. We must have $E \rightarrow^k E_1' \rightarrow E_1$ for some E_1'. By the inductive hypothesis, there exists E'' with $E_1' \rightarrow^* E''$ and $E_2 \rightarrow^* E''$. It must be that either $E_1' = E''$ or $E_1' \rightarrow E_1'' \rightarrow^* E''$ for some E_1''. Consider each case.
● In the first case, let $E' = E_1$. We have that $E_1 \rightarrow^* E'$. We also have that $E_2 \rightarrow^* E'' = E_1' \rightarrow E_1 = E'$, so $E_2 \rightarrow^* E'$ as required.
● In the second case, let $E' = E''$. Since $E_1' \rightarrow E_1$ and $E_1' \rightarrow E_1''$, it must be that $E_1 = E_1''$ by determinacy of $\rightarrow$. Hence, $E_1 \rightarrow^* E'' = E'$. We also have that $E_2 \rightarrow^* E'' = E'$, as required. □

Of course, not every result needs to be proved by induction!
Corollary (Unique Answer for $\rightarrow$) If $E \rightarrow^* n_1$ and $E \rightarrow^* n_2$ then $n_1 = n_2$.

Proof. Suppose that $E \rightarrow^* n_1$ and $E \rightarrow^* n_2$. By confluence, there is some E' with $n_1 \rightarrow^* E'$ and $n_2 \rightarrow^* E'$. 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 26

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

Exercise (see sheet)

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

Normalization and the equivalence of the big-step and small-step semantics are given in the answers to exercises.

Slide 27

Proof: one way

We prove each direction of implication separately. We first prove, by induction on the structure of E, the property $P(E)$:

$$E \Downarrow n \text{ implies } E \rightarrow^{*} n.$$

Base Case $E = n$ for arbitrary numeral n. We know that $n \Downarrow n$ and $n \rightarrow^{0} n$, so the base case is trivially true.

Slide 28

Inductive Case $E = E_1 + E_2$ for arbitrary expressions E_1 and E_2. The inductive hypotheses are:

$$E_1 \Downarrow n_1 \text{ implies } E_1 \rightarrow^{*} n_1$$
$$E_2 \Downarrow n_2 \text{ implies } E_2 \rightarrow^{*} n_2$$

From these assumptions, we must show that $P(E_1 + E_2)$: namely that

$$(E_1 + E_2) \Downarrow n \text{ implies } (E_1 + E_2) \rightarrow^{*} n.$$

Let $(E_1 + E_2) \Downarrow n$. From the rules, this can only happen if $E_1 \Downarrow n_1$ and $E_2 \Downarrow n_2$ for some n_1, n_2, with $n_1 \underline{+} n_2 = n$. By the inductive hypotheses, $E_1 \rightarrow^{*} n_1$ and $E_2 \rightarrow^{*} n_2$.

....

Slide 29

Multi-step Reductions in *SimpleExp*

The relation $E \rightarrow^r E'$ 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

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

Slide 30

Lemmas

Lemmas

1. $E_1 \rightarrow^r E_1'$ implies $(E_1 + E_2) \rightarrow^r (E_1' + E_2)$
2. $E_2 \rightarrow^r E_2'$ implies $(n + E_2) \rightarrow^r (n + E_2')$

Corollaries

1. $E_1 \rightarrow^* n_1$ implies $(E_1 + E_2) \rightarrow^* (n_1 + E_2)$.
2. $E_2 \rightarrow^* n_2$ implies $(n_1 + E_2) \rightarrow^* (n_1 + n_2)$.
3. $E_1 \rightarrow^* n_1$ and $E_2 \rightarrow^* n_2$ implies $(E_1 + E_2) \rightarrow^* n$ where $n = n_1 \underline{+} n_2$.

Slide 31

Inductive Case $E = E_1 + E_2$ for arbitrary expressions E_1 and E_2. The inductive hypotheses are:

$E_1 \Downarrow n_1$ implies $E_1 \rightarrow^* n_1$
$E_2 \Downarrow n_2$ implies $E_2 \rightarrow^* n_2$

From these assumptions, we must show that $P(E_1 + E_2)$: namely that

$(E_1 + E_2) \Downarrow n$ implies $(E_1 + E_2) \rightarrow^* n$.

Let $(E_1 + E_2) \Downarrow n$. From the rules, this can only happen if $E_1 \Downarrow n_1$ and $E_2 \Downarrow n_2$ for some n_1, n_2, with $n_1 \underline{+} n_2 = n$. By the inductive hypotheses, $E_1 \rightarrow^* n_1$ and $E_2 \rightarrow^* n_2$. By Corollary 3 on previous slide, $(E_1 + E_2) \rightarrow^* n$, as required.

Slide 32

Proof: the other way

Suppose that $E \rightarrow^* n$. By the totality of $\Downarrow$ (see the exercise sheet), we know that $E \Downarrow m$ for some m. By the first half of the result, it follows that $E \rightarrow^* m$. By the uniqueness of answers for $\rightarrow$ (in the lecture notes), it must be that $m = n$. Therefore $E \Downarrow n$, as required.

It is also possible to prove this result directly, without appealing to normalization and determinacy, by induction on the structure of E.

Slide 33

Lemma

$E_1 \to^r E_1'$ implies $(E_1 + E_2) \to^r (E_1' + E_2)$

Slide 34

Writing an Inductive Proof: Revisited

Say what you want to prove, and what you are 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 35

Writing an Inductive Proof About

Multi-step Reductions

This is simple induction on numbers. To prove $\forall r.P(r)$ by induction on r:

Base Case

- The base case is $r = 0$
- Prove that $P(0)$ holds, any way you like.

Inductive Case

- The inductive case is $r = k + 1$ for arbitrary k.
- There is one inductive hypotheses: $P(k)$.
- Prove $P(k + 1)$, using the inductive hypothesis.

Slide 36

Proof of Lemma

Lemma

$E_1 \rightarrow^r E_1'$ implies $(E_1 + E_2) \rightarrow^r (E_1' + E_2)$

Proof

By induction on r. Let $P(r)$ be the property:

$$\forall E_2 . E_1 \rightarrow^r E_1' \text{ implies } (E_1 + E_2) \rightarrow^r (E_1' + E_2)$$

Base Case It must be the case that $E_1 \rightarrow^0 E_1$, and $(E_1 + E_2) \rightarrow^0 (E_1 + E_2)$ holds for arbitrary E_2. Hence, $P(0)$ is true.

Slide 37

Inductive Case the inductive hypothesis is that $P(k)$ holds: that is,

$$\forall E_2.E_1 \rightarrow^k E_1' \text{ implies } (E_1 + E_2) \rightarrow^k (E_1' + E_2)$$

From this assumption, we must show that $P(k+1)$: namely that,

$$\forall E_2.E_1 \rightarrow^{k+1} E_1' \text{ implies } (E_1 + E_2) \rightarrow^{k+1} (E_1' + E_2)$$

So consider $E_1 \rightarrow^{k+1} E_1'$ for arbitrary E_1 and E_1', which can be broken down into $E_1 \rightarrow^k E_1'' \rightarrow E_1'$. By the inductive hypothesis, $(E_1 + E_2) \rightarrow^k (E_1'' + E_2)$. Since $E_1'' \rightarrow E_1'$, we can apply the rule (S-LEFT) to obtain $(E_1'' + E_2) \rightarrow (E_1' + E_2)$, hence the result.

The proof of the other lemma is similar.

Slide 38

Syntax of Expressions

Recall that expressions of the *While* language are defined as elements of the following grammar:

$$E \in \mathit{Exp} ::= n \mid x \mid E + E \mid E \times E \mid \ldots$$

where $n \in \mathbb{N}$ is a natural number and x is a variable.

Exercicse: When doing a proof by induction on the structure of expressions, what are the base cases and what are the inductive cases?

Slide 39

Determinacy of $\rightarrow$ for *Exp*

All the results for simple expressions can be extended to expression configurations of the form $\langle E, s\rangle$ where variable store s describes the values of the variables.

Determinacy: For all E, s, E_1, s_1, E_2, s_2,

$$\langle E, s\rangle \rightarrow_e \langle E_1, s_1\rangle \wedge \langle E, s\rangle \rightarrow_e \langle E_2, s_2\rangle \Rightarrow \langle E_1, s_1\rangle = \langle E_2, s_2\rangle$$

Exercise: Prove this by induction on the structure of expression E.

Slide 40

Syntax of Commands

Recall that commands of the *While* language are defined as elements of the following grammar:

$$\begin{aligned} C \in \textit{Com} \ ::= \ & x := E \mid \texttt{if}\ B\ \texttt{then}\ C\ \texttt{else}\ C \\ & \mid C; C \mid \texttt{skip} \mid \texttt{while}\ B\ \texttt{do}\ C \end{aligned}$$

where x is a variable, $E \in$ *Exp* and $B \in$ *Bool*.

Exercise: When doing a proof by induction on the structure of commands, what are the base cases and what are the inductive cases?

Slide 41

Determinacy for *While*

Exercise

Prove, by induction on the structure of commands, that the small-step semantics for While is **deterministic**: that is, for any configurations $\langle C, s\rangle, \langle C_1, s_1\rangle, \langle C_2, s_2\rangle$,

$$\langle C, s\rangle \rightarrow_c \langle C_1, s_1\rangle \wedge \langle C, s\rangle \rightarrow_c \langle C_2, s_2\rangle \Rightarrow \langle C_1, s_1\rangle = \langle C_2, s_2\rangle$$

You may assume that the relations $\rightarrow_e$ and $\rightarrow_b$ on expressions and booleans are deterministic.

The proof of this result is given in the answer sheet to the exercises on induction.