Induction: in many forms

Massimo Merro

10 October 2017

Induction as a proof technique

- In the previous lecture we stated several "theorems"; but how can we formally prove them?
- Intuition is often wrong: we need formal proofs!
- Formal proofs are also useful for strengthening our intuition about subtle language features, and for debugging definitions: they help in examining all possible cases.
- Which proof technique technique should we adopt?
- Most of our definitions are inductively defined. To prove properties about them we need the use the adequate induction principle!

What is induction for?

Induction is a formal 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 the arbitrary complexity.

Structured and finite objects arise in many areas of computer science. Data structures, 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 of a certain language*.

Three forms of induction

In the following we will focus on three different forms of induction.

Mathematical induction

Prove facts/properties about all natural numbers.

Structural induction

Prove facts/properties about things that have an inductively defined structure: trees, lists, programs, etc.

Rule induction

Prove facts/things about all elements of a relation defined by means of inference rules.

We will see that all three forms of induction boils down to induction over certains trees.

The natural numbers $\mathbb{N}$

Two rules for constructing natural numbers

(a) base rule: 0 is in $\mathbb{N}$

(b) inductive rule: if k is in $\mathbb{N}$ then so is its successor $k+1$

Every natural number can be *constructed* using these two rules.

Use induction to define functions which operates on natural numbers.

Definition principle for $\mathbb{N}$

To define a function $f : \mathbb{N} \longrightarrow X$

(a) base rule: describe the result of applying f to 0

(b) inductive rule: assuming $f(k)$ has already been defined, define $f(k+1)$ in terms of $f(k)$

Result: with (a) and (b) function f is defined for every natural number.

Examples

Summation:

sum : $\mathbb{N} \longrightarrow \mathbb{N}$ is defined by:

(a) base rule: $\text{sum}(0) = 0$

(b) inductive rule: $\text{sum}(k+1) = \text{sum}(k) + (k+1)$

Factorial:

fac : $\mathbb{N} \longrightarrow \mathbb{N}$ is defined by:

(a) base rule: $\text{fac}(0) = 1$

(b) inductive rule: $\text{fac}(k+1) = \text{fac}(k) \times (k+1)$

Example

Multi-step reductions in Exp:

red : $\mathrm{Exp} \times \mathbb{N} \longrightarrow \mathrm{Exp}$ is defined by:

(a) base rule: $\mathrm{red}(\mathsf{E}, 0) = \mathsf{E}$, for every espression E

(b) inductive rule: $\mathrm{red}(\mathsf{E}, k+1) = \mathsf{E}''$
if there is E' such that $\mathrm{red}(\mathsf{E}, k) = \mathsf{E}'$ and $\mathsf{E}' \twoheadrightarrow \mathsf{E}''$.

Quite often, instead of writing $\mathrm{red}(\mathsf{E}, k) = \mathsf{E}'$ we write, more intuitively,

$$\mathsf{E} \twoheadrightarrow^{k} \mathsf{E}'$$

The intuition is that after k steps the evaluation of the expression E returns E'.

Are we sure the fuction red is correctly defined? Does it change anything if we replace $\twoheadrightarrow$ with $\twoheadrightarrow_{\mathrm{ch}}$?

Proof principle for $\mathbb{N}$ – Mathematical Induction

The simplest form of induction is *mathematical induction*, that is to say, induction over natural numbers. The principle can be described as follows.

Given a property $P(-)$ on natural numbers we want to prove that $P(n)$ holds for *every* natural number n:

(a) Base case: prove $P(0)$ is true (using some known mathematical facts)

(b) Inductive case:

- assume the *inductive hypothesis*, i.e., that $P(k)$ is true
- from the inductive hypothesis prove that $P(k+1)$ follows (using some known mathematical fact)

If (a) and (b) are established then $P(n)$ *is true for every natural number* n.

Mathematical induction is a valid principle because every natural number can be "built" using 0 as a starting point and the operation of adding one for building new numbers.

It should be clear why this principle is valid: if we can prove (a) and (b). 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
- and so on...

Therefore $P(n)$ holds for any n, regardeless of how big n is.

This conclusion can only be drawn because every natural number can be reached by starting at zero and adding one repeatedly.

Example 1

Lemma 1 $\text{sum}(n) = \frac{n*(n+1)}{2}$ for every natural number n.

Property $P(n)$: $\text{sum}(n) = \frac{n*(n+1)}{2}$

Proof We must show:

(a) *Base case*: $\text{sum}(0) = 0$ (it follows from the def. of $\text{sum}(-)$)

(b) *Inductive case*:

- Assume the inductive hypothesis (IH): $\text{sum}(k) = \frac{k*(k+1)}{2}$
- Use IH to deduce $P(k+1)$: $\text{sum}(k+1) = \frac{(k+1)*(k+2)}{2}$ (use some algebraic manipulations)

Result: $\text{sum}(n) = \frac{n*(n+1)}{2}$ for every natural number n.

Example 2

Lemma 2 For every $E \in Exp$, if $E \rightarrow^k F$ then $E + G \rightarrow^k F + G$ for any expression G.

Property $P(k)$: $E_1 \rightarrow^k E_2$ implies $E_1 + G \rightarrow^k E_2 + G$, for any E_1, E_2, G.

Proof We must show:

(a) *Base case*: $P(0)$ is true. For if $E_1 \rightarrow^0 E_2$ then $E_2 = E_1$ and therefore trivially $E_1 + G \rightarrow^0 E_2 + G$

(b) *Inductive case*: prove $P(k+1)$ by assuming $P(k)$.
Let $E_1 \xrightarrow{k+1} E_2$.
There must be E_3 s.t. $E_1 \rightarrow E_3 \rightarrow^k E_2$.

- By IH, $E_3 + G \rightarrow^k E_2 + G$.
- By an application of rule (S-Left), $E_1 + G \rightarrow E_3 + G$.

It follows that $E_1 + G \rightarrow^{k+1} E_2 + G$.

Inductively defined structures: Natural numbers

We said 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 for building new numbers.

Example: Natural numbers as inductive objects

We can turn mathematical induction into a form of structural induction by viewing natural numbers as elements of the following BNF grammar:

$$N \quad ::= \quad \mathbf{zero} \quad \Big| \quad \mathbf{succ}(N)$$

Here, **succ**$(-)$, short for *successor*, should be thought as the operation of adding one to its argument. Therefore, **zero** respresents 0 and, 3, for instance, is represented by **succ**(**succ**(**succ**(**zero**))).

Numbers, when thought of like this, are finite, structured objects.

Functions on natural numbers, revisited

The principle of defining functions by induction works for this representation of the natural numbers in exactly the same way as before:

Summation:

sum : $N \longrightarrow N$ is defined by:

(a) base rule: $\text{sum}(\mathbf{zero}) = \mathbf{zero}$

(b) inductive rule: $\text{sum}(\mathbf{succ}(N)) = \mathbf{succ}^{n+1}(\text{sum}(N))$
if $N = \mathbf{succ}(\ldots\mathbf{succ}(\mathbf{zero})) = \mathbf{succ}^{n}(\mathbf{zero})$, for some natural number n.

Factorial:

fac : $N \longrightarrow N$ is defined by:

(a) base rule: $\text{fac}(\mathbf{zero}) = \mathbf{succ}(\mathbf{zero})$

(b) inductive rule: $\text{fac}(\mathbf{succ}(N)) = \ldots$ That's a good exercise!

Structural induction for natural numbers

The principle of induction now says that in order to prove $P(N)$ for all numbers N, it suffices to do two things:

(a) Base case: Prove that $P(\mathbf{zero})$ holds.

(b) Inductive case: Prove that $P(\mathbf{succ}(K))$ follows by assuming as IH that $P(K)$ holds for some number K.

Note that when trying to prove $P(\mathbf{succ}(K))$, the inductive hypothesis tells us that we may assume the propery holds for the *substructure* of $\mathbf{succ}(K)$, that is, we can assume $P(K)$ holds.

This structural viewpoint, and the associated form of induction, called **structural induction**, is widely applicable.

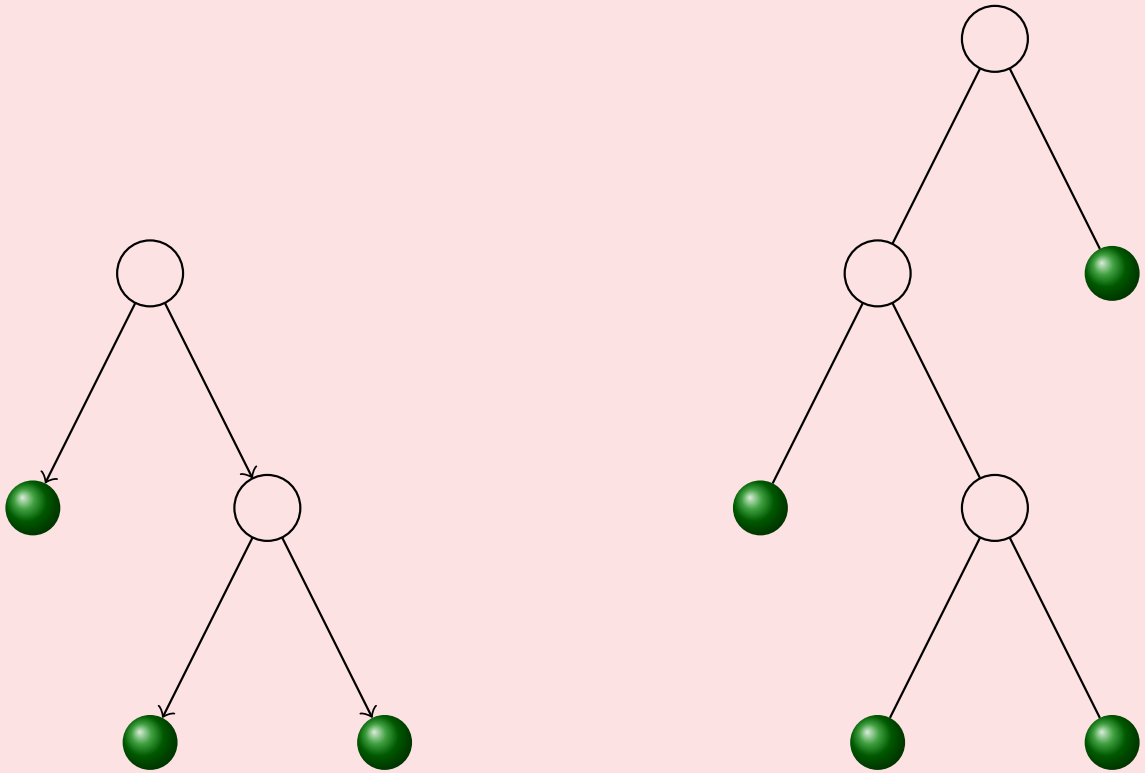
It should be clear why this principle is valid: if we can prove (a) and (b). then we know

- $P(\mathbf{zero})$ holds
- Since $P(\mathbf{zero})$ holds, $P(\mathbf{succ}(\mathbf{zero}))$ holds
- Since $P(\mathbf{succ}(\mathbf{zero}))$ holds, $P(\mathbf{succ}(\mathbf{succ}(\mathbf{zero})))$ holds
- Since $P(\mathbf{succ}(\mathbf{succ}(\mathbf{zero})))$ holds, $P(\mathbf{succ}(\mathbf{succ}(\mathbf{succ}(\mathbf{zero}))))$ holds
- and so on...

That is to say, we have shown that every way of building a number preserves the property P, and that if P is true of the basic building block **zero**, so P is true of every number.

Inductive structures: Binary Trees

Example: Binary trees



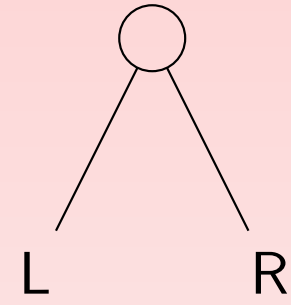
Each node is either

- a leaf: 
- or a node  with two siblings

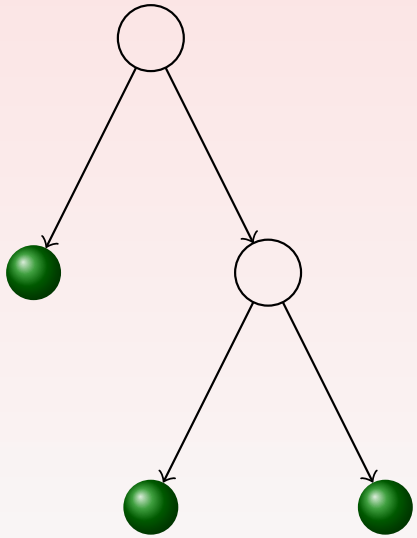
Constructing binary trees

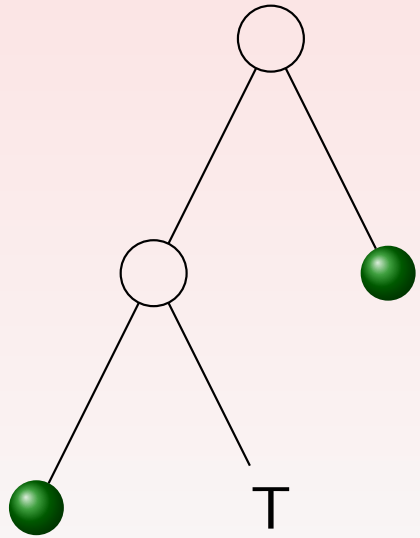
(a) Base case:  is a binary tree

(b) Inductive case: If L and R are binary trees then so is



T =





Syntax over binary trees

$$T \in bTree \quad ::= \quad \textbf{leaf} \quad \Big| \quad \textbf{tree}(T, T)$$

Construction rules:

(a) Base case: **leaf** is a binary tree

(b) Inductive case: if L and R are binary trees then so is $\textbf{tree}(L, R)$

Examples

$\textbf{tree}(\textbf{leaf}, \textbf{tree}(\textbf{leaf}, \textbf{leaf}))$
$\textbf{tree}(\textbf{tree}(\textbf{leaf}, \textbf{tree}(\textbf{leaf}, \textbf{leaf})), \textbf{tree}(\textbf{leaf}, \textbf{leaf}))$

Definition principle for binary trees

The principle of defining functions by induction works for this representation of the binary trees in exactly the same way as for natural numbers:

To define a function $f : bTree \longrightarrow X$:

(a) Base rule: describe the result of applying f to the terminal **leaf**

(b) Inductive rule: assuming $f(T_1)$ and $f(T_2)$ have already been defined, describe the result of applying f to the binary tree **tree**(T_1, T_2).

Result: with (a) and (b) we know that function f is defined for every binary tree.

Example definitions

Number of leaves in a tree:

leaves : $bTree \longrightarrow \mathbb{N}$ define by:

(a) Base case: $leaves(\mathbf{leaf}) = 1$

(b) Inductive case: $leaves(\mathbf{tree}(T_1, T_2)) = leaves(T_1) + leaves(T_2)$

Number of branches in a tree:

branches : $bTree \longrightarrow \mathbb{N}$ define by:

(a) Base case: $branches(\mathbf{leaf}) = 0$

(b) Inductive case:
$branches(\mathbf{tree}(T_1, T_2)) = branches(T_1) + branches(T_2) + 1$

Structural induction for binary trees

To prove a property $P(T)$ for every binary tree $T \in bTree$.

(a) Base case: prove $P(\textbf{leaf})$ is true (using known mathematical facts)

(b) Inductive case:

– assume the *inductive hypothesis*: $P(T_1)$ and $P(T_2)$ are both true

– from this hypothesis prove that $P(\textbf{tree}(T_1, T_2))$ follows (using known mathematical facts)

If (a) and (b) are established it follows that $P(T)$ *is true for every binary tree* T.

Example proof

Let us prove

$$\mathit{leaves}(T) = \mathit{branches}(T) + 1 \text{ for every binary tree } T$$

Property $P(T)$ is: $\mathit{leaves}(T) = \mathit{branches}(T) + 1$

(a) Base case:
$P(\mathbf{leaf})$: $\mathit{leaves}(\mathbf{leaf}) = \mathit{branches}(\mathbf{leaf}) + 1$ (follows by definition)

(b) Inductive case:
assume $P(T_1)$ and $P(T_2)$ are true (IH). From IH prove $P(\mathbf{tree}(T_1, T_2))$ follows:

$$\begin{aligned}
\mathit{leaves}(\mathbf{tree}(T_1, T_2)) &= \mathit{leaves}(T_1) + \mathit{leaves}(T_2) \\
&= \mathit{branches}(T_1) + 1 + \mathit{branches}(T_2) + 1 \quad (\mathit{IH}) \\
&= (\mathit{branches}(T_1) + \mathit{branches}(T_2) + 1) + 1 \\
&= \mathit{branches}(\mathbf{tree}(T_1, T_2)) + 1
\end{aligned}$$

Inductive structures: Arithmetic espressions

Also arithmetic expressions were defined in terms of inductive definitions.

$$E \in Exp \ ::= \ \mathtt{n} \ \Big| \ E + E \ \Big| \ E \times E$$

Constructing arithmetic expressions:

(a) Base case: $\mathtt{n}$ is an arithmetic expression for every numeral $\mathtt{n} \in Num$

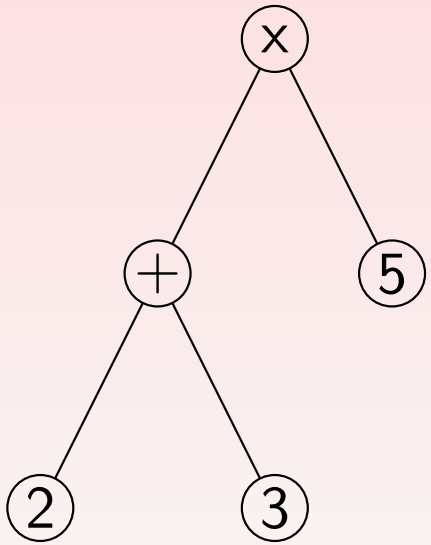
(b) Inductive case: if E_1 and E_2 are aritmethic expressions so are
- $E_1 + E_2$
- $E_1 \times E_2$

- an infinite number of base cases
- two inductive cases.

Abstract syntax 1/2

Here, we want to stress a bit that when proving properties on expressions defined by a grammar we are actually interested in the *abstract syntax*!
Q: Is the expression $(2 + 3) \times 5$:

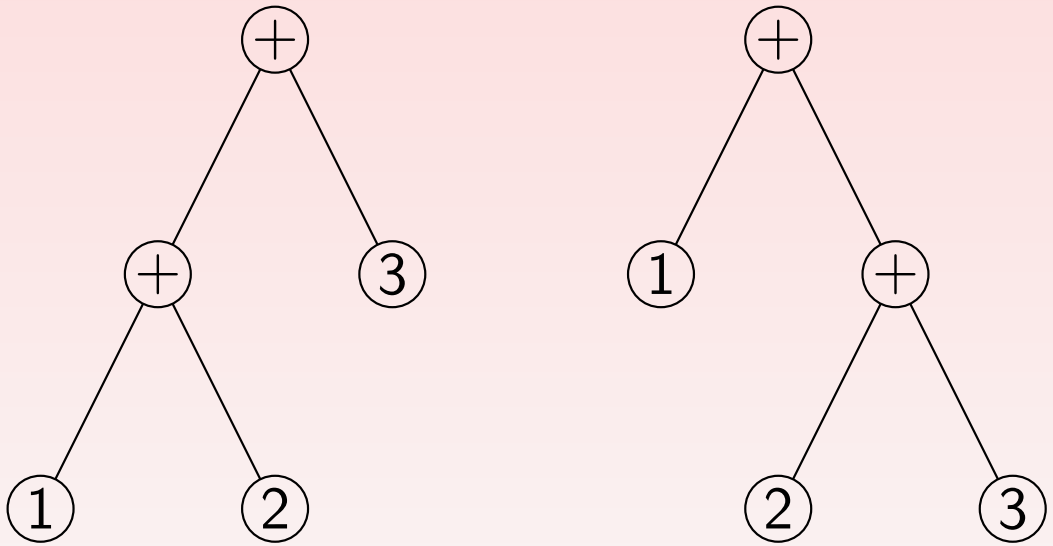
1. a list of characters?
2. or, a list of tokens?
3. or, an abstract syntax tree?



A: An abstract syntax tree!

Abstract syntax 2/2

- Notice that parentheses are not part of the grammar: they are only used for disambiguation!
- The expression $1 + 2 + 3$ is ambiguous.
- In fact $(1 + 2) + 3 \neq 1 + (2 + 3)$: their corresponding abstract syntax trees are different!



- For semantics purposes it's easier to work with abstract syntax and abstract syntax trees rather than concrete syntax!

Definition principle for arithmetic expressions

To define a function $f : Exp \longrightarrow X$:

(a) Base case: describe the result of applying f to $\mathtt{n}$, for every $\mathtt{n} \in Num$.

(b) Inductive case: assuming $f(E_1)$ and $f(E_2)$ have both already been defined, describe the result of

- applying f to $E_1 + E_2$
- applying f to $E_1 \times E_2$

Result: with (a) (b) we know function f is defined for every arithmetic expression.

Structural induction over arithmetic expression

To prove a property $P(E)$ for every arithmetic expression $E \in Exp$.

(a) Base case: prove $P(\mathtt{n})$ is true for every numeral $\mathtt{n} \in Num$ (using known mathematical facts)

(b) Inductive case:

- assume the *inductive hypothesis*: $P(E_1)$ and $P(E_2)$ are both true
- from this hypothesis prove that
  - $P(E_1 + E_2)$ follows (using known mathematical facts)
  - $P(E_1 \times E_2)$ follows (using known mathematical facts)

If (a) and (b) are established it follows that $P(E)$ *is true for every arithmetic expression* E.

Example: normalisation of big-step semantics

Property: "For every arithmetic expression E there exists some numeral $\mathtt{k}$ such that $E \Downarrow \mathtt{k}$."

This property says that all programs in our *Exp* language have a final answer or so-called "*normal form*".

Formally, the property $P(E)$ is: $E \Downarrow \mathtt{k}$ for some numeral $\mathtt{k}$

Proof by structural induction:

(a) Base case: We have to show $P(\mathtt{n})$ for every numeral $\mathtt{n}$

(b) Inductive case: Assume $P(E_1)$ and $P(E_2)$ are true. We have to show

- $P(E_1 + E_2)$ is true
- $P(E_1 \times E_2)$ is true.

Example: small-step semantics

Property: "$E \rightarrow F$ implies $E \rightarrow_{ch} F$ for all arithmetic expressions E and F"

Formally, $P(E)$ is: $E \rightarrow F$ implies $E \rightarrow_{ch} F$

Proof by induction on the structure of E:

(a) Base case: We have to show $\mathtt{n} \rightarrow F$ implies $\mathtt{n} \rightarrow_{ch} F$, for every numeral $\mathtt{n}$

(b) Inductive case: Assume the inductive hypothesis (IH)

- $E_1 \rightarrow F_1$ implies $E_1 \rightarrow_{ch} F_1$
- $E_2 \rightarrow F_2$ implies $E_2 \rightarrow_{ch} F_2$

From (IH) we have to show

- $E_1 + E_2 \rightarrow F$ implies $E_1 + E_2 \rightarrow_{ch} F$
- $E_1 \times E_2 \rightarrow F$ implies $E_1 \times E_2 \rightarrow_{ch} F$.

Q: Does $E \rightarrow_{ch} F$ imply $E \rightarrow F$ for all arithmetic expressions E and F?

More properties

Normalisation goes hand in hand with *determinacy*.

Determinacy for big-step semantics

$E \Downarrow \mathtt{m}$ and $E \Downarrow \mathtt{n}$ implies $\mathtt{m} = \mathtt{n}$

Determinacy for small-step semantics

- (strong) $E \rightarrowtriangle F$ and $E \rightarrowtriangle G$ implies $F = G$
- (weak) $E \rightarrowtriangle^* \mathtt{m}$ and $E \rightarrowtriangle^* \mathtt{n}$ implies $\mathtt{m} = \mathtt{n}$.

Any relation between the weak and the strong form?

Any idea on how to prove these properties?

What about by induction on the structure of E?

Rule induction

Assumo la presenza di un sistema di inferenze

Lemma: $\Gamma \vdash E \cdot T \Rightarrow \Gamma(E)$

Per rule induction sulla profondità dell'albero di derivazione di $\Gamma \vdash E \; T$.

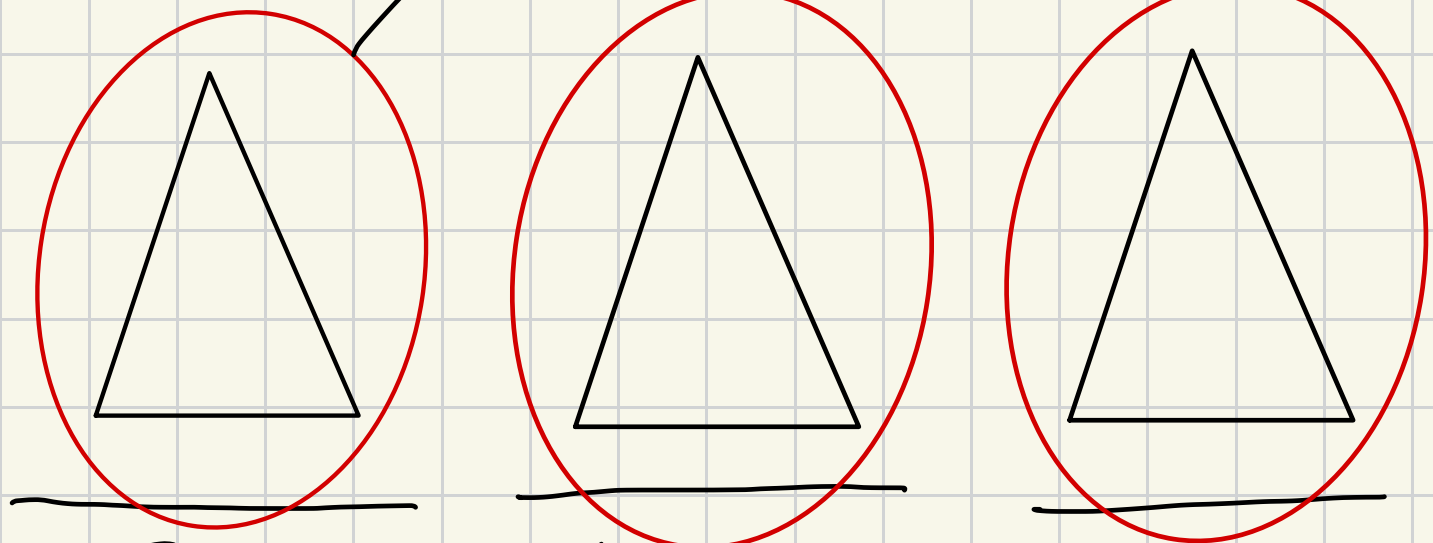
- The behaviour of an arithmetic expression E is completely determined by the behaviour of its components
- For this reason structural induction is sufficiently powerful to prove properties for the different semantics of *Exp*
- However, in more complicated languages, with recursive or inductive control operators, we need more sophisticated instruments
- Idea: The basic idea of rule induction is to ignore the structure of objects and instead concentrate on the size of the derivations of judgements.

non guardiamo la struttura dell'oggetto (E) ma la dimensione della profondità dell'albero di derivazione (in questo caso ftpaggio)

utilizzati per induzione

Albero originale di dimensione più grande

Alberi di profondità più piccola



$\Gamma \vdash E_1 : Bool$ $\quad \Gamma \vdash E_2 : T$ $\quad \Gamma \vdash E_3 : T$

$\Gamma \vdash E : T$

What is the size of a derivation?

For example, consider the following pair of rules , defining an infix binary relation D between numbers in $\mathbb{N}$.

premessa

$$(\text{Ax})\ \frac{}{n\ \mathit{Div}\ 0} \qquad (\text{Plus})\ \frac{n\ \mathit{Div}\ m}{n\ \mathit{Div}\ (m+n)}$$

divisore numero da dividere

Derivations:

$$(\text{Plus})\ \frac{(\text{Plus})\ \dfrac{(\text{Plus})\ \dfrac{(\text{Ax})\ \dfrac{}{7\ \mathit{Div}\ 0}}{7\ \mathit{Div}\ 7}}{7\ \mathit{Div}\ 14}}{7\ \mathit{Div}\ 21} \qquad (\text{Plus})\ \frac{(\text{Plus})\ \dfrac{(\text{Ax})\ \dfrac{}{2\ \mathit{Div}\ 0}}{2\ \mathit{Div}\ 2}}{2\ \mathit{Div}\ 4}$$

Devo usare sempre una delle due regole

Size of the derivation of $2\ \mathit{Div}\ 4$ is smaller that that of $7\ \mathit{Div}\ 21$.

Example of rule induction 1/3

- Suppose we want to prove a statement of the form

$$n \ \ Div \ \ m \text{ implies } P(n, m)$$

  then we can use induction on the size of the derivation of $n \ \ Div \ \ m$.
- As an example, suppose $P(n, m)$ be: $m = n \times k$ for some natural number k
- This actually means that the rules (Ax) and (Plus) correctly capture the notion of *division*
- So, let us prove

$$n \ \ Div \ \ m \text{ implies } P(n, m)$$

  by mathematical induction on the size of derivation of the judgement $n \ \ Div \ \ m$ from the rules (Ax) and (Plus).

Example of rule induction 2/3

- Suppose we have a derivation of $n \ \mathit{Div} \ m$
- Using mathematical induction means that we have as inductive hypothesis saying that $P(k_1, k_2)$ is *true* for any k_1, k_2 for which there is a derivation $k_1 \ \mathit{Div} \ k_2$ whose size is less than the size of the derivation for $n \ \mathit{Div} \ m$.
- $n \ \mathit{Div} \ m$ can be derived only using axioms (Ax) and (Plus).
- What does this derivation look like? There are only two possibilities:

(a) It is an application of axiom (Ax): $(\mathrm{Ax})\ \dfrac{}{n \ \mathit{Div} \ 0}$.

Only if m is actually 0. $P(n, 0)$ is trivially true, the required k being 0.

Example of rule induction 3/3

(b) It is an application of rule (Plus):

$$\text{(Plus)}\ \frac{(\ldots)\quad \dfrac{\ldots}{n\ \ Div\ \ m_1}}{n\ \ Div\ \ (m_1 + n)}$$

se uso questa regola ho una qualche informazione sulla struttura, poiché $m = m_1 + n$, informazione derivata dal fatto che ho usato quella regola

where $m = m_1 + n$.

- But this means that the judgement $n\ \ Div\ \ m_1$ also has a derivation from the rules.
- Moreover, the size of this derivation is strictly less than that of $n\ \ Div\ \ m$.
- So, (IH) applies and we know there is k_1 such that $m_1 = n \times k_1$.
- Now, $P(n, m)$ is an immediate consequence as $m = n \times (k_1 + 1)$.

Formally, Rule induction

To prove a property $P(D)$ for every derivation D, it is enough to do the following.

(a) Base case: prove $P(A)$ is true for every axiom A (using known mathematical facts)

(b) Inductive case:

for each rule of the form

$$(\text{rule})\ \frac{h_1 \ldots h_n}{c}$$

prove that any derivation ending with a use of this rule satisfies the property. Such derivation has *subderivations* $D_1, \ldots, D_n$ with conclusions $h_1, \ldots, h_n$. By *inductive hypothesis* we assume that $P(D_i)$ holds for each subderivation D_i, $1 \leq i \leq n$.

Proving Progress (Outline)

se ho un programma ben tipato il programma non va in deadlock

Theorem 3 (Progress) If $\Gamma \vdash e : T$ and $\mathrm{dom}(\Gamma) \subseteq \mathrm{dom}(s)$ then either e is a value or there exist e', s' such that $\langle e, s\rangle \rightarrow \langle e', s'\rangle$.

Proof Let ϕ be the ternary relation defined as follows:

$$\phi(\Gamma, e, T) \stackrel{\text{def}}{=} \forall s.\, \mathrm{dom}(\Gamma) \subseteq \mathrm{dom}(s) \Rightarrow \mathrm{value}(e) \vee (\exists e', s'.\, \langle e, s\rangle \rightarrow \langle e', s'\rangle)$$

per ogni stato e — è valore o — ha una transizione

We prove that for all

$$\Gamma, e, T,\ \text{if } \Gamma \vdash e : T \text{ then } \phi(\Gamma, e, T)$$

by rule induction on why $\Gamma \vdash e : T$.

This means that we do a case analysis on the "last" typing rule applied to derive $\Gamma \vdash e : T$. There are

4 base cases: axioms (int), (bool), (deref), (skip);

6 inductive cases: rules (op +), (op $\geq$), (if), (assign), (seq) and (while).

Let us see in detail a few of them.

Proof By rule induction on hone $\Gamma \vdash E : T$

Questo significa fare induzione con un analisi per casi sull'ultima regola di tipaggio usata

↓ quella che sta più in basso

(1) Caso base: ho usato la regola (int). Se $\Gamma \vdash E : T$ ne consegue che $E = n$ per qualche n e $T = int$

Possiamo concludere che vale $\Phi(\Gamma, n, int)$ perché value $(n) = true$ e quindi Φ è vero

$$\frac{-}{\Gamma \vdash n : int}\ (int)$$

(2) Caso base 2: ho usato deref: $\dfrac{-}{\Gamma \vdash !\ell : int}\ (\Gamma(\ell) = intref)$

Ne consegue che $E \equiv !\ell$ per qualche ℓ, e $T \equiv int$. Posso concludere $\Phi(\Gamma, !\ell, int)$ perché usando la regola semantica (deref) ho

$\langle E, s \rangle \equiv \langle !\ell, s \rangle \rightarrow \langle n, s \rangle$ se $s(\ell) = n$ dove $dom(s) \supseteq dom(\Gamma) \supseteq \{\ell\}$

Gli altri due casi base sono skip e bool

- Caso induttivo (i casi induttivi sono 6): ho usato la regola assign: $\dfrac{\Gamma \vdash E_1 : int \qquad \Gamma[l] = intref}{\Gamma \vdash l := E_1 : unit}$

ne consegue che $E \equiv l := E_1$, per qualche E_1, e $\tau \equiv unit$

Per ipotesi induttiva (IH) ho $\Gamma \vdash E_1 : int$

vale che $\Phi(\Gamma, E_1, int)$ è vera

ho un albero più piccolo rispetto a quello da cui sono partito cioè quello di $\Gamma \vdash E : T$ perciò vale la proprietà

Ne consegue che ci sono due casi

(1) $value(E_1) = true$. Cioè un intero, visto che $\Gamma = int$

Cioè $E \equiv l := n$ per qualche n

Ma allora per uno store s, t.c $dom(s) \equiv dom(\Gamma)$, uso la regola assign 1 per dedurre

$$\langle E, s \rangle \equiv \langle l := n, s \rangle \Rightarrow \langle skip, s[l \mapsto n] \rangle \equiv \langle E', s' \rangle$$

Quindi $\Phi(\Gamma, E, T)$ è vera

(2) $value(E_1) = false$, cioè E_1 non è un terminale!
Da $\Phi(\Gamma, E_1, int)$ deriva che $\langle E_1, s\rangle \to \langle E_1', s'\rangle$ per qualche E_1', s' Usando la regola (assign 2)

$$\langle E, s\rangle \equiv \langle l := E_1, s\rangle \to \langle l := E_1', s\rangle \equiv \langle E', s'\rangle$$

Quindi $\Phi(\Gamma, E, T)$ è vera

- Caso induttivo (2): ho usato la regola di tipaggio seq:

$$\frac{\Gamma \vdash E_1 : unit \qquad \Gamma \vdash E_2 : T}{\Gamma \vdash E_1 ; E_2 : T}$$

Ne consegue che $E = E_1; E_2$, $\Gamma \vdash E_1 : unit$, $\Gamma \vdash E_2 : T$
Per (IH) da $\Gamma \vdash E_1 : unit$ ne deriva che $\Phi(\Gamma, E_1, unit)$. Ne consegue che ci sono 2 casi:

(1) $value(E_1) = true$ ne consegue che $E_1 \equiv skip$, quindi $E \equiv skip; E_2$
Ma allora usando la regola semantica (seq. skip). Per uno store tale $dom(s) \supseteq dom(\Gamma)$
Abbiamo $\langle E_1 s\rangle \equiv \langle skip; E_2, s\rangle \to \langle E_2, s\rangle \equiv \langle E', s'\rangle$. Quindi vale $\Phi(\Gamma, E, T)$.

(2) valve (E_1) = false. ne consegue che $\exists\ E_1', s'$ t.c $\langle E_1, s\rangle \to \langle E_1', s'\rangle$

Usando la regola semantica

$$\langle E_1\ s\rangle \equiv \langle E_1;\ E_2\ ,\ s\rangle \longrightarrow \langle E_1';\ E_2, s'\rangle \equiv \langle E', s'\rangle$$

Ne consegue che $\Phi(\Gamma, E, T)$ è vera

Caso induttivo (3): ho usato la regola di tipaggio if:

$$\frac{\Gamma \vdash E_1 : \text{bool} \quad \Gamma \vdash E_2 : T \quad \Gamma \vdash E_3 : T}{\Gamma \vdash \text{if } E_1 \text{ then } E_2 \text{ else } E_3 : T}$$

Ne consegue che $E \equiv$ if E_1 then E_2 else E_3 con $\Gamma \vdash E_1 : \text{bool}$, $\Gamma \vdash E_2 : T$, $\Gamma \vdash E_3 : T$ Poiché l'albero di derivazione $\Gamma \vdash E_1 : \text{bool}$ ha profondità inferiore rispetto a quello di $\Gamma \vdash E : T$ posso usare l'induzione e concludere che $\Phi(\Gamma, E_1, \text{bool})$ è vera

Ci sono allora 2 casi:

① value (E_1) = valore

$$\exists\, E_1', s', \text{ t.c } \langle E_1, s\rangle \longrightarrow \langle E_1', s'\rangle$$

Ne consegue che E_1 è un booleano ma allora ci sono 2 ulteriori sottocasi:

① $E_1 \equiv$ true. In tal caso

$\langle E, s\rangle \equiv \langle$ if E_1 then E_2 else $E_3, s\rangle \longrightarrow \langle E_2, s\rangle \equiv \langle E', s'\rangle$

usando la regola semantica if

② $E_1 \equiv$ false. In tal caso

$\langle E, s\rangle \equiv \langle$ if E_1 then E_2 else $E_3, s\rangle \longrightarrow \langle E_3, s\rangle \equiv \langle E', s'\rangle$

usando la regola semantica if

Perciò $\exists (\Gamma, E, T)$ è vera!!!

② $\exists E_1', s'$ t.c $\langle E_1, s \rangle \to \langle E_1', s' \rangle$.

Ma allora usando la regola semantica (IF) ne consegue che

$\langle E, s \rangle \to \langle \text{if } E_1 \text{ then } E_2 \text{ else } E_3, s \rangle \to \langle \text{if } E_1' \text{ then } E_2 \text{ else } E_3, s' \rangle \equiv \langle E', s' \rangle$

Quindi $\Phi(\Gamma, E, \tau)$ è vera

④ caso induttivo 4: ha le regole di tipaggio (while):

$$\frac{\Gamma \vdash E_1 : \text{bool} \quad \Gamma \vdash E_2 : \text{unit}}{\Gamma \vdash \text{while } E_1 \text{ do } E_2 : \text{unit}}$$

e consegue $E = \text{while } E_1 \text{ do } E_2$. Ma allora usando la regola semantica while ho:

$\langle E_1, s \rangle$

$\langle \text{while } E_1 \text{ do } E_2, s \rangle \longrightarrow$

$\langle \text{if } E_1 \text{ then } E_2 \text{ while } E_1 \text{ do } E_2 \text{ else skip}, s \rangle = \langle E', s' \rangle$

$\langle E', s' \rangle$

Quindi E fa un passo!

e $\Phi(\Gamma, E, \tau)$ è vera

- (int): then $e = n \in \mathbb{Z}$ and $T = \text{int}$; this implies $\phi(\Gamma, n, \text{int})$.
- (deref): then $e = !l$, for some l, $\Gamma(l) = \text{intref}$ and $T = \text{int}$; this implies $\phi(\Gamma, !l, \text{int})$.
- (op +): then $e = e_1 + e_2$, $T = \text{int}$, $\Gamma \vdash e_1 : \text{int}$ and $\Gamma \vdash e_2 : \text{int}$; by inductive hypothesis $\phi(\Gamma, e_1, \text{int})$ and $\phi(\Gamma, e_2, \text{int})$. Thus,
  - if not value(e_1) then $\langle e_1, s \rangle$ progresses and hence, by an application of the small-step rule (op1), we have $\phi(\Gamma, e_1 + e_2, \text{int})$;
  - if value(e_1) and value(e_2) then, by an application of the small-step rule (op +), we have $\phi(\Gamma, e_1 + e_2, \text{int})$;
  - if value(e_1) and not value(e_2) then $\langle e_2, s \rangle$ progresses and hence, by an application of the small-step rule (op2), we have $\phi(\Gamma, e_1 + e_2, \text{int})$.
- (seq): then $e = e_1; e_2$, $T = \text{unit}$, $\Gamma \vdash e_1 : \text{unit}$ and $\Gamma \vdash e_2 : \text{unit}$; by inductive hypothesis $\phi(\Gamma, e_1, \text{unit})$ and $\phi(\Gamma, e_2, \text{unit})$. Thus,
  - if not value(e_1) then $\langle e_1, s \rangle$ progresses and hence, by an application of the small-step rule (Seq), we have $\phi(\Gamma, e_1; e_2, \text{unit})$;
  - if value(e_1) then by an application of the small-step rule (Seq.Skip), we have $\phi(\Gamma, e_1; e_2, \text{unit})$.
- ... do the remaining 2 base cases and 4 inductive cases.

Example

As an example, if $\Gamma \supseteq \{(l, \mathsf{intref})\}$, then $\Gamma \vdash (!l + 2) + 3 : \mathsf{int}$ implies $\phi(\Gamma, (!l + 2) + 3, \mathsf{int})$. In fact

$$\dfrac{(\mathsf{op}+)\ \dfrac{(\mathsf{op}+)\ \dfrac{(\mathsf{deref})\ \dfrac{-}{\Gamma !l : \mathsf{int}}\quad (\mathsf{int})\ \dfrac{-}{2 : \mathsf{int}}}{\Gamma \vdash (!+2) : \mathsf{int}}\quad (\mathsf{int})\ \dfrac{-}{\Gamma \vdash 3 : \mathsf{int}}}{\Gamma \vdash (!l + 2) + 3 : \mathsf{int}}$$

Proving Type Preservation (Outline)

Lemma 4 If $\langle e, s\rangle \rightarrowtriangle \langle e', s'\rangle$ then $\mathrm{dom}(s) = \mathrm{dom}(s')$.

Proof By rule induction on why $\langle e, s\rangle \rightarrowtriangle \langle e', s'\rangle$. Let $\Phi(e, s, e', s') = (\mathrm{dom}(s) = \mathrm{dom}(s'))$. All rules are immediate uses of the inductive hypothesis, except rule (assign1), for which we note that if $l \in \mathrm{dom}(s)$ then $\mathrm{dom})(s[l \mapsto n]) = \mathrm{dom}(s)$. □

Theorem 4 (Type Preservation) If $\Gamma \vdash e : T$ and $\mathrm{dom}(\Gamma) \subseteq \mathrm{dom}(s)$ and $\langle e, s\rangle \rightarrowtriangle \langle e', s'\rangle$ then $\Gamma \vdash e' : T$ and $\mathrm{dom}(\Gamma) \subseteq \mathrm{dom}(s')$.

Proof By Lemma 4 we only prove the first part. The proof is by rule induction on why $\langle e, s\rangle \rightarrowtriangle \langle e', s'\rangle$. Let

$$\Phi(e, s, e', s') = \forall \Gamma, T.\, (\Gamma \vdash e : T \ \wedge\ \mathrm{dom}(\Gamma) \subseteq \mathrm{dom}(s)) \Rightarrow \Gamma \vdash e' : T \ .$$

This means that we do a case analysis on the semantics rule applied to derive $\langle e, s\rangle \rightarrowtriangle \langle e', s'\rangle$. There are

8 base cases: (op +), (op $\geq$), (deref), (assign1), (seq1), (if1), (if2), (while);

5 inductive cases: rules (op 1), (op 2), (assign2), (seq2), (if3).

Let us see in detail a few of them. □

Hp

Theorem: $\Gamma \vdash e : T,\ dom(\Gamma) \supseteq dom(S),\ \langle e,s\rangle \to \langle e',s'\rangle \Rightarrow \Gamma \vdash e' : T \quad \Phi(e,s,e',s')$

Proof:

Per rule induction su come ho derivato il passo $\langle e,s\rangle \to \langle e',s'\rangle$

$\langle n + \emptyset, s\rangle \to \langle ff, s\rangle$

Caso base: La $\langle e,s\rangle \to \langle e',s'\rangle$ derivato usando la regola semantica (deref). Ne consegue da $e \equiv !l$, per qualche $l \in dom(s)$, inoltre $e' \equiv n$, t.c $s(l) = n$. Sappiamo che $\Gamma \vdash e.T$.
Per tiparlo abbiamo usato la regola di tipaggio della deferenziazione $\dfrac{\Gamma(l) = intref}{\Gamma \vdash e \equiv !l : int = T}$ Cioé $T \equiv int$ ma allora banalmente

$\Gamma \vdash e' \equiv n : T = int$

CASO BASE 1: La $\langle e,s\rangle \to \langle e',s\rangle$ derivato dall'applicazione della regola semantica (assign 1)

Ne consegue che $e \equiv l := n$, per qualche l, n, con $l \in dom(s)$. Inoltre $l' \equiv skip$. $dom(s') = dom(s)$

Essendo $\Gamma \vdash e : T$, vol dire che abbiamo usato le regole di tipaggio (assign)

$$\frac{\Gamma \vdash n : int \qquad \Gamma(l) = intref}{\Gamma \vdash e \equiv l := n : unit}\ (Assign)$$

cioé $T \equiv unit$. Ma ovviamente usando le regole di tipaggio (skip)

$\Gamma' \vdash l' \equiv skip : T \equiv unit$

CASO BASE 2: La $\langle e,s\rangle \to \langle e',s'\rangle$ derivato dall'applicazione della regola semantica (seq 1)

Ne consegue che $\ell \equiv skip; \ell_1$, per qualche e_1. Inoltre $e' \equiv e_1$. Poiché $\Gamma \vdash e : T$ ne consegue che ho applicato la regola di tipaggio (seq) $\dfrac{\Gamma \vdash skip : unit \quad \Gamma \vdash e_1 : T}{\Gamma \vdash e \equiv skip; \ell_1 : T}$ Notate che T è rimasto generica

Ma allora poiché $\ell' \equiv \ell_1$, $\Gamma \vdash e_1 : \ell$ ne deriva che $\Gamma \vdash \ell' \equiv \ell_1 : T$ come desiderato.

$$\frac{\langle \ell_1, s\rangle \to \langle e_1', s'\rangle}{\langle \ell := e_1, s\rangle \to \langle \ell := e_1', s'\rangle}$$

CASO INDUTTIVO 1 La $\langle e,s\rangle \to \langle e',s'\rangle$ derivato dall'applicazione della regola semantica (assign 2)

Ne consegue che $e \equiv \ell := e_1$, $\langle e_1, s\rangle \to \langle e_1', s'\rangle$, inoltre $e' \equiv \ell \equiv e_1$. Sappiamo $\Gamma \vdash e : T$,

Cioè ho usato le regole di tipaggio (Assign) $\dfrac{\Gamma \vdash e_1 : int \quad \Gamma(\ell) = intref}{\Gamma \vdash e \equiv \ell := e_1 : unit}$ Quindi $T \equiv unit$. Per (IH)

Poiché, l'albero di derivazione di $\Gamma \vdash e_1 : int$ ha profondità inferiore a quella dell'albero $\Gamma \vdash e : T$ e poiché $\langle e_1, s\rangle \to \langle e_1', s'\rangle$ ne deriva che $\Gamma \vdash e_1' : int$. Applicando la regola di tipaggio (Assign)

$\dfrac{\Gamma \vdash e_1' : int}{\Gamma \vdash e' \equiv \ell := e_1' : unit}$ quindi ℓ' ha lo stesso tipo T.

CASO INDUTTIVO 2: Per rule induction su come ho derivato il passo $\langle e_1, s\rangle \to \langle e_1', s'\rangle$

La $\langle e_1, s\rangle \to \langle e', s'\rangle$ derivato dall'applicazione della regola semantica (seq 2). Cioè $e \equiv e_1 ; e_2$, per qualche e_1, e_2, inoltre $\dfrac{\langle e_1, s\rangle \to \langle e_1', s'\rangle}{\langle e \equiv e_1 ; e_2, s\rangle \to \langle e_1', e_2, s'\rangle}$. Poiché $\Gamma \vdash e : T$ vuol dire che ho usato le regole di tipaggio (seq) $\dfrac{\Gamma \vdash e_1 : unit \quad \Gamma \vdash e_2 : T}{\Gamma \vdash e \equiv e_1 ; e_2 : T}$. Per (IH) avendo il passo $\langle e_1, s\rangle \to \langle e_1', s\rangle$ una derivazione con profondità inferiore di $\langle e_1, s\rangle \to \langle e', s'\rangle$, ed essendo $\Gamma \vdash e_1 : unit$, ne deriva che $\Gamma' \vdash e_1' : unit$. Allora, usando la regola di tipaggio (seq) ottengo: $\dfrac{\Gamma \vdash e_1' : unit \quad \Gamma \vdash e_2 : T}{\Gamma \vdash e' \equiv e_1' ; e_2 : T}$, cioè $\Gamma \vdash e' : T$, come derivato

CASO INDUTTIVO 3: Per rule induction su come ho derivato dall'applicazione della regola semantica (if 3). Cioè

$e \equiv$ if e_1 then e_2 else e_3 $\dfrac{\langle e_1, s\rangle \to \langle e_1', s'\rangle}{\langle e \equiv \text{if } e_1 \text{ then } e_2 \text{ else } e_3, s\rangle \to \langle \text{if } e_1' \text{ then } e_2 \text{ else } e_3, s'\rangle}$, $e' \equiv$ if e_1' then e_2 else e_3

Da $\Gamma \vdash e : T$ deriva che ho usato la regola di tipaggio if $\dfrac{\Gamma \vdash e_1 : bool \quad \Gamma \vdash e_2 : T \quad \Gamma \vdash e_3 : T}{\Gamma \vdash e \equiv \text{if } e_1 \text{ then } e_2 \text{ else } e_3 : T}$

Per (IH) poiché il passo $\langle e_1, s\rangle \to \langle e_1', s'\rangle$ ha un albero di derivazione con profondità inferiore di $\langle e_1, s\rangle \to \langle e', s'\rangle$ ne deriva che $\Gamma \vdash e_1' : bool$. Applicando nuovamente la regola di tipaggio dell'(if)

$\dfrac{\Gamma \vdash e_1' : bool \quad \Gamma \vdash e_2, e_3 : T}{\Gamma \vdash e' \equiv \text{if } e_1 \text{ then } e_2 \text{ else } e_3 : T}$. Cioè e' ho tipo bool

La $\langle e, s\rangle \to \langle e', s'\rangle$ derivato dall'applicazione della regola semantica (assign 2)
Cioé $e \equiv l := e_1$ con $\dfrac{\langle e_1, s\rangle \to \langle e_1', s'\rangle}{\langle e \equiv l := e_1, s\rangle \to \langle e' \equiv l := e_1', s'\rangle}$ Poiché $\Gamma \vdash e : T$, ho usato la regola di tipaggio (Assign)

$\dfrac{\Gamma \vdash e_1 : \text{int} \quad \Gamma(l) = \text{intref}}{\Gamma \vdash e \equiv l : e_1 : \text{unit} \equiv T}$ Per (IH) poiché il passo $\langle e_1, s\rangle \to \langle e_1, s\rangle$ ha profondità inferiore del $\langle e, s\rangle \to \langle e', s'\rangle$ ne deriva che $\Gamma \vdash e_1' : \text{int}$. Ma allora usando ancora la regola di tipaggio (Assign)

$\dfrac{\Gamma \vdash e_1' : \text{int} \quad \Gamma(l) = \text{intref}}{\Gamma \vdash e' \equiv l := e_1' : \text{unit} \equiv T}$. Cioé e' ha lo stesso tipo T.

- (op +):

$$(\text{op } +)\ \frac{-}{\langle n_1 + n_2, s\rangle \rightarrow \langle n, s\rangle} \quad \text{if } n = add(n_1, n_2)$$

Take arbitrary Γ, T. Suppose $\Gamma \vdash n_1 + n_2 : T$ and $\text{dom}(\Gamma) \subseteq \text{dom}(s)$. The last rule applied in the type derivation must be (op+), so must have $T = \text{int}$. Then we use the typing rule (int) to derive $\Gamma \vdash n : \text{int}$.

- (op 1):

$$(\text{op } 1)\ \frac{\langle e_1, s\rangle \rightarrow \langle e_1', s'\rangle}{\langle e_1 \text{ op } e_2, s\rangle \rightarrow \langle e_1' \text{ op } e_2, s'\rangle}$$

By induction $\Phi(e_1, s, e_1', s')$. Take arbitrary Γ, T. Suppose $\Gamma \vdash e_1 \text{ op } e_2 : T$ and $\text{dom}(\Gamma) \subseteq \text{dom}(s)$. There are 2 cases:

  - $op = +$. Must have $T = \text{int}$, $\Gamma \vdash e_1 : \text{int}$, $\Gamma \vdash e_2 : \text{int}$. By induction $\Gamma \vdash e_1' : \text{int}$, and by applying rule (op+) we have $\Gamma \vdash e_1' + e_2 : T$.
  - $op = \geq$. Similar.