Proof Techniques Based on Operational Semantics

Lecture 3
CS263

Plan

- We’ll study various flavors of induction
  - mathematical induction
  - well-founded induction
  - structural induction

- We’ll see how to apply induction to language analysis

Induction

- Probably the single most important technique for the study of formal semantics of programming languages

- Of several kinds
  - mathematical induction (the simplest)
  - well-founded induction (the most general)
  - structural induction (the most widely used in PL)

Mathematical Induction

- Goal: prove that ∀n ∈ N, P(n)

- Strategy: (2 steps)
  1. Base case: prove that P(0)
  2. Inductive case:
     - pick an arbitrary n ∈ N
     - assume that P(n) holds
     - prove that P(n + 1)

- or, formally prove that ∀n ∈ N. P(n) → P(n+1)
Mathematical Induction. Notes.

- The inductive case looks similar to the overall goal \( \forall n \in \mathbb{N}. P(n) \Rightarrow P(n+1) \) vs. \( \forall n \in \mathbb{N}. P(n) \)
- Easier to prove because of the assumption that \( P(n) \) holds

- Why does mathematical induction work?
  - The key property of \( \mathbb{N} \) is that there are no infinite descending chains of naturals. It has to stop somewhere.
  - For each \( n \), \( P(n) \) can be obtained from the base case and \( n \) uses of the inductive case

Example of Mathematical Induction

- Recall the evaluation rules for IMP commands
- Prove that if \( \sigma(x) \leq 6 \) then <while \( x \leq 5 \) do \( x := x + 1 \), \( \sigma \) > \( \sigma[ x := 6 ] \)

- Reformulate the claim:
  - Let \( W = \) while \( x \leq 5 \) do \( x := x + 1 \)
  - Let \( \sigma_i = \sigma[ x := 6 - i ] \)
  - Claim: \( \forall i \in \mathbb{N}, <W, \sigma_i > \Downarrow \sigma_0 \)
  - Related to loop invariants

- Now the claim looks provable by mathematical induction on \( i \)

Example of Mathematical Induction (Base Case)

- Base case: for \( i = 0 \) prove <\( W, \sigma_i > \Downarrow \sigma_0 \)
  - To prove an evaluation judgment, construct a derivation tree:
    \[
    \sigma_0(x) = 6 \\
    <x, \sigma_0 > \Downarrow 6 \\
    \vdash 6 \leq 5, \sigma_0 \Downarrow \text{false} \\
    <x, \sigma_0 > \Downarrow \text{false} \\
    \vdash \text{while } x \leq 5 \text{ do } x := x + 1, \sigma_0 \Downarrow \text{false} \\
    \vdash 5 - i \leq 5 \\
    \vdash x \leq 5 \text{ do } x := x + 1, \sigma_i \Downarrow \sigma_0 \\
    \vdash <x + 1, \sigma_i > \Downarrow \text{false}
    \]

- This completes the base case

Example of Mathematical Induction (Inductive Case)

- Must prove \( \forall i \in \mathbb{N}, <W, \sigma_i > \Downarrow \sigma_0 \Rightarrow <W, \sigma_{i+1} > \Downarrow \sigma_0 \)
- The beginning of the proof is straightforward
  - Pick an arbitrary \( i \in \mathbb{N} \)
  - Assume that <\( W, \sigma_i > \Downarrow \sigma_0 \)
  - Now prove that <\( W, \sigma_{i+1} > \Downarrow \sigma_0 \)
  - Must construct a derivation tree with conclusion <\( W, \sigma_{i+1} > \Downarrow \sigma_0 \)
Discussion

• A proof is more powerful than running the code and observing the result. Why?

• The proof relied on a loop invariant
  - \( x \leq 6 \) in all iterations
  - Or, the equivalent variant \( x = 6 - i \) \( \forall i \geq 0 \)

• ... and a loop variant
  - \( 6 - x \) is positive and decreasing

• Picking the loop invariant and variant is typically the hardest part of a proof

Well-Founded Induction

• A relation \( \prec \subseteq A \times A \) is well-founded if there are no infinite descending chains by \( \prec \) in \( A \)
  - Example: \( \prec_1 = \{ (x, x+1) \mid x \in \mathbb{N} \} \)
    - the predecessor relation
  - Example: \( \prec = \{ (x, y) \mid x, y \in \mathbb{N} \) and \( x < y \} \)

• Well-founded induction:
  - To prove \( \forall x \in A. P(x) \) it is enough to prove
    \( \forall x \in A. (\forall y \prec x \Rightarrow P(y)) \Rightarrow P(x) \)

• If \( \prec \) is \( \prec_1 \), then we obtain mathematical induction as a special case

Discussion

• We proved termination and correctness. This is called total correctness

• Mathematical induction is good when we prove properties of natural numbers
  - In PL analysis we most often prove properties of expressions, commands, programs, input data, etc.
  - We often need a more powerful induction principle

Well-Founded Induction. Notes.

• Why does well-founded induction work?
  - Assume we proved that \( \forall x \in A. (\forall y \prec x \Rightarrow P(y)) \Rightarrow P(x) \)
  - Define \( D_n \) to be the subset of all elements of \( A \) whose longest descending chain is of length at most \( n \)
    \( \forall x \in A. \exists n \geq 1. x \in D_n \) (no infinite descending chains)
  - Show that \( \forall n \geq 1. (\forall x \in D_n. P(x)) \) (by mathematical induction)
  - Base case: Show \( \forall x \in D_1. P(x) \)
    - It is vacuously true that if \( x \in D_1 \) then \( \forall y \prec x \Rightarrow P(y) \)
  - Inductive case:
    - Assume \( \forall x \in D_n. P(x) \)
    - Show that \( \forall x \in D_{n+1}. P(x) \)
      - But for any \( y \) if \( y \prec x \) then \( y \in D_n \)
      - Thus \( \forall y \prec x \Rightarrow P(y) \)
  - See Winskel (Chapter 3) for another proof (the correct one!)
Well-Founded Induction. Examples.

- Consider $\subseteq \mathbb{N} \times \mathbb{N}$ with $x \prec y$ iff $x + 2 = y$
  - $\forall x \in \mathbb{N}. (\forall y (x \prec y \rightarrow P(y)) \rightarrow P(x)$ is equivalent to
  - $P(0) \land \forall x \in \mathbb{N}. (P(x) \rightarrow P(x + 2))$

- Consider $\subseteq \mathbb{Z} \times \mathbb{Z}$ with $x \prec y$ iff
  - $(y < 0$ and $y = x - 1)$ or $(y > 0$ and $y = x + 1)$
  - $P(0) \land \forall x \leq 0. P(x) \rightarrow P(x - 1) \land \forall x \geq 0. P(x) \rightarrow P(x + 1)$

- Consider $\subseteq (\mathbb{N} \times \mathbb{N}) \times (\mathbb{N} \times \mathbb{N})$ and $(x_1, y_1) \prec (x_2, y_2)$ iff
  - $x_2 = x_1 + 1$ $\lor$ $(x_1 = x_2$ and $y_2 = y_1 + 1)$
  - This leads to the induction principle
  - $P(0,0) \land \forall x,y,y'. (P(x,y) \rightarrow \{ P(x+1,y') \land P(x,y+1) )
  - This is sometimes called lexicographic induction

Structural Induction

- Recall $e ::= n \mid e_1 + e_2 \mid e_1 \cdot e_2 \mid x$
- Define $\preceq \mathit{Aexp} \times \mathit{Aexp}$ such that
  - $e_1 \preceq e_2$
  - $e_1 = e_2$
  - $e_1 \neq e_2$
  - and no other elements of $\mathit{Aexp} \times \mathit{Aexp}$ are related by $\preceq$

- To prove $\forall e \in \mathit{Aexp}. P(e)$
  1. Prove $\forall n \in \mathbb{Z}. P(n)$
  2. Prove $\forall x \in L. P(x)$
  3. Prove $\forall e_1, e_2 \in \mathit{Aexp}. P(e_1) \land P(e_2) \rightarrow P(e_1 + e_2)$
  4. Prove $\forall e_1, e_2 \in \mathit{Aexp}. P(e_1) \land P(e_2) \rightarrow P(e_1 \cdot e_2)$

Structural Induction. Notes.

- Called structural induction because the proof is guided by the structure of the expression
- As many cases as there are expression forms
  - Atomic expressions (with no subexpressions) are all base cases
  - Composite expressions are the inductive case
- This is the most useful form of induction in PL

Example of Induction on Structure of Expressions

- Let
  - $L(e)$ be the number of literals and variable occurrences in $e$
  - $O(e)$ be the number of operators in $e$
- Prove that $\forall e \in \mathit{Aexp}. L(e) = O(e) + 1$
- By induction on the structure of $e$
  - Case $e = n$. $L(e) = 1$ and $O(e) = 0$
  - Case $e = x$. $L(e) = 1$ and $O(e) = 0$
  - Case $e = e_1 + e_2$.
    - $L(e) = L(e_1) + L(e_2)$ and $O(e) = O(e_1) + O(e_2) + 1$
    - By induction hypothesis $L(e_1) \cdot O(e_1) + 1$ and $L(e_2) = O(e_2) + 1$
    - Thus $L(e) = O(e) + 1$
  - Case $e = e_1 \cdot e_2$. Same as the case for +
Other Proofs by Structural Induction on Expressions

- Most proofs for Aexp sublanguage of IMP
- Small-step and natural semantics are equivalent

\[ \forall e \in \text{Exp}. \forall n \in \mathbb{N}. e \rightarrow^* n \iff e \downarrow^* n \]

- Structural induction on expressions works here because the semantics rules themselves are syntax directed

Induction on the Structure of Derivations

- Key idea: The hypothesis does not assume just a \( c \in \text{Comm} \) but the existence of a derivation of \( \langle c, c' \rangle \)
- Derivation trees are also defined inductively, just like expression trees
- A derivation is built of subderivations:

\[
\begin{align*}
\langle x, o_2 \rangle \downarrow 5 - i & \quad 5 - i = x \\
\langle x + 1, o_2 \rangle \downarrow 6 - i & \\
\langle x + 1, o_2 \rangle \downarrow o_1 & \quad W, o_2 \downarrow o_0
\end{align*}
\]

- While \( x \times 5 \) do \( x := x + 1, o_2 \) \( \downarrow o_0 \)
- Adapt the structural induction principle to work on the structure of derivations

Another Proof

- Prove that IMP is deterministic

\[ \forall e \in \text{Aexp}. \forall \sigma \in \Sigma. \forall n, n' \in \mathbb{N}. \langle e, \sigma \rangle \downarrow^* n \iff \langle e, \sigma \rangle \downarrow^* n' \]

- No immediate way to use mathematical induction
- For commands we cannot use induction on the structure of the command

\[
\begin{align*}
\langle b, c, o_2 \rangle \downarrow o_1 & \quad \langle c, o \rangle \downarrow o' & \quad \langle b, o \rangle \downarrow o'' \iff o = o''
\end{align*}
\]

Induction on Derivations (analogy with induction on exp)

<table>
<thead>
<tr>
<th>Struct. Ind. On exp</th>
<th>Struct. Ind. On derivations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prove: ( P(e) ) for any expression ( e )</td>
<td>Prove: ( P(D) ) for any derivation ( D )</td>
</tr>
<tr>
<td>For each ( e ) constructor (e.g., ( + ))</td>
<td>For each derivation constructor (derivation rule)</td>
</tr>
</tbody>
</table>

1. Assume \( P(e_i) \) holds
2. Prove \( P(e_i + e_j) \) holds
1. Assume \( P(D_i) \) holds for derivation \( D_i \) of \( H_i (i = 1 \ldots n) \)
2. Prove that \( P(D) \) holds, where \( D \) is obtained by applying "rule" to \( D_i \)
Example of Induction on Derivations (I)

- Prove that evaluation of commands is deterministic:
  \[ \text{\texttt{<c, }\sigma} \Downarrow \sigma' \implies \forall \sigma'' \in \Sigma. \text{\texttt{<c, }\sigma} \Downarrow \sigma'' \implies \sigma'' = \sigma' \]

- Pick arbitrary \( c, \sigma, \sigma' \) and \( D := \text{\texttt{<c, }\sigma} \Downarrow \sigma' \)
- To prove: \( \forall \sigma'' \in \Sigma. \text{\texttt{<c, }\sigma} \Downarrow \sigma'' \implies \sigma'' = \sigma' \)
- Proof by induction on the structure of the derivation \( D \)
  - Case: last rule used in \( D \) was the one for skip
    - This means that \( c = \text{\texttt{skip}} \), and \( \sigma'' = \sigma \)
    - By inversion \( \text{\texttt{<c, }\sigma} \Downarrow \sigma'' \) uses the rule for skip. Thus \( \sigma'' = \sigma \)
    - This is a base case in the structural induction

Example of Induction on Derivations (II)

- Case: the last rule used in \( D \) was the one for sequencing
  - Pick arbitrary \( \sigma'' \) such that \( D'' := \text{\texttt{<c_1; c_2}, }\sigma'' \Downarrow \sigma'' \)
    - by inversion \( D'' \) uses the rule for sequencing
    - and has subder. \( D''_1 := \text{\texttt{<c_1}, }\sigma''_1 \Downarrow \sigma''_1 \)
    - By induction hypothesis on \( D'' \) (with \( D''_1 \)): \( \sigma_1 = \sigma''_1 \)
    - Now \( D''_2 := \text{\texttt{<c_2, }\sigma''_2} \Downarrow \sigma''_2 \)
    - By induction hypothesis on \( D_2 \) (with \( D''_2 \)): \( \sigma''_2 = \sigma''_1 \)
    - This is a simple inductive case

Example of Induction on Derivations (III)

- Case: the last rule used in \( D \) was the one for while true
  - Pick arbitrary \( \sigma'' \) such that \( D'' := \text{\texttt{<while b do c}, }\sigma'' \Downarrow \sigma'' \)
    - by inversion and determinism of boolean expressions, \( D'' \) also uses the rule for while true
    - and has subder. \( D''_1 := \text{\texttt{<b, }\sigma''_1} \Downarrow \text{true} \)
    - Now \( D''_2 := \text{\texttt{<while b do c, }\sigma''_2} \Downarrow \sigma''_2 \)
    - By induction hypothesis on \( D_2 \) (with \( D''_2 \)): \( \sigma''_2 = \sigma''_1 \)
    - By induction hypothesis on \( D \) (with \( D'' \)): \( \sigma'' = \sigma''_1 \)

Induction on Derivations. Notes.

- If we have to prove \( \forall x \in A. P(x) \Rightarrow Q(x) \)
  with \( A \) inductively defined and \( P(x) \) rule-defined
  - we could do induction on both facts
  - except when \( D \) is itself defined in a syntax-directed manner
  - In many situations there are several choices for induction
    - choosing the right one is a trial-and-error process
    - a bit of practice can help a lot
**Summary of Operational Semantics**

- Precise specification of dynamic semantics
  - order of evaluation (or that it doesn’t matter)
  - error conditions (sometimes implicitly, by rule applicability)
- Simple and abstract (vs. implementations)
  - no low-level details such as stack and memory management, data layout, etc.
- Often not compositional (see while)
- Basis for some proofs about the language
- Basis for some reasoning about programs
- Point of reference for other semantics

---

**Equivalence**

- Two expressions (commands) are equivalent if they yield the same result from all states

$$e_1 = e_2 \text{ iff } \forall \sigma \in \Sigma: \forall n \in \mathbb{N}, <e_1, \sigma> \Downarrow n \text{ iff } <e_2, \sigma> \Downarrow n$$

and for commands

$$c_1 = c_2 \text{ iff } \forall \sigma, \sigma' \in \Sigma: <c_1, \sigma> \Downarrow \sigma' \text{ iff } <c_2, \sigma> \Downarrow \sigma'$$

---

**Notes on Equivalence**

- Equivalence is like validity
  - must hold in all states
  - $2 = 1 + 1$ is like "$2 = 1 + 1$ is valid"
  - $2 = 1 + x$ might or might not hold.
    - So, $2$ is not equivalent to $1 + x$

- Equivalence (for IMP) is undecidable
  - If it were we could solve the halting problem for IMP. How?

- Equivalence justifies code transformations
  - compiler optimizations
  - code instrumentation
  - abstract modeling

- Semantics is the basis for proving equivalence.
**Equivalence Examples**

- skip; \(c = c\)
- \((x := e_1; x := e_2) = x := e_2\). When is this true?
- while \(b\) do \(c = \text{if } b \text{ then } c\) while \(b\) do \(c = \text{skip}\) when?
- If \(e_1 = e_2\) then \(x := e_1 = x := e_2\)
- While true do \(c = \text{while true do } x := x + 1\)
- If \(c\) is
  - while \(x \cdot y\) do
    - if \(x \cdot y\) then \(x := x \cdot y\) else \(y := y \cdot x\)
  - then \(x := 221; y := 527; c = (x := 17; y := 17)\)

**Proving An Equivalence**

- Prove that “\(\text{skip; } c = c\)” for all \(c\)
- Assume that \(D :: \langle\text{skip; } c, \sigma\rangle \Downarrow \sigma'\)
- By inversion (twice) we have that
  - \(D_1 :: \langle c, \sigma\rangle \Downarrow \sigma'\)
- Thus, we have \(D_1 :: \langle c, \sigma\rangle \Downarrow \sigma'\)
- The other direction is similar

**Proving An Inequivalence**

- Prove that \(x := y \neq x := z\) when \(y \neq z\)
- It suffices to exhibit a \(\alpha\) in which the two commands yield different results
  - Let \(\alpha(y) = 0\) and \(\alpha(z) = 1\)
  - Then \(\langle x := y, \sigma\rangle \Downarrow \alpha(x := 0)\)
  - and \(\langle x := z, \sigma\rangle \Downarrow \alpha(x := 1)\)