Equivalence of expressions

- Last time: equivalence of two semantics for same language
- What about equivalence of two expressions in language?
  - IMP: expressions are commands, arithmetic, boolean exprs
  - Useful for program transformations
- Idea: programs observationally equivalent if they permit the same executions
- Example: \( x := y + y \) \( \sim \) \( x := 2 \times y \); \( z := z \)

Formalizing Equivalence

- Program equivalence:
  \[ c_1 \sim c_2 \iff \forall \sigma. \ (c_1, \sigma) \uparrow \sigma' \iff (c_2, \sigma) \uparrow \sigma' \]
- Expressions \( e_1, e_2 \) are observationally equivalent if every program containing one (e.g., \( e_1 \)) is equivalent to the same program with the other (e.g., \( e_2 \)) substituted for it
- Let \( C[\ ] \) be an expression context: any program with a hole \( [\] \) where an expression can go
- Example: \( x := 0; \) while \( x < 10 \) do \( [\] \)
- Let \( C[e_i] \) be the program with \( e_i \) instead of hole
- \( e_1 \sim e_2 \iff \forall C[\ ]. C[e_1] \sim C[e_2] \)
- IMP: two notions of equivalence identical for commands (not true for all languages)

Contexts

- To capture idea of “all contexts”, define command context \( C[\ ] \) with BNF:
  \[
  C[\ ] ::= [\ ] | C[\ ]; c | c; C[\ ]
  \left| \begin{array}{l}
  \text{if } b \text{ then } C[\ ] \text{ else } c \\
  \text{if } b \text{ then } c \text{ else } C[\ ] \\
  \text{while } b \text{ do } C[\ ]
  \end{array} \right.
  \]
- Can use inductive definition of context to construct proofs of expression equivalence

Inductive proofs

- Some things we’d like to prove
  - equivalence of different semantics
    - small-step vs. large-step
  - equivalence of different expressions
    - \( c; \) while \( \neg b \) do vs. \( c \) vs. do \( c \) until \( b \)
  - termination of expressions
  - deterministic evaluation of expressions, programs
- In general, need inductive proofs

Administration

- Homework 1 due September 11
### Proving termination

- **Assertion:** Arithmetic expressions always terminate: \( \exists n. (a, n) \rightarrow^* (n, \sigma) \)
- **An argument:**
  - Expressions of the form \( X \) or \( n \) always terminate in one step (evaluation defined by axioms)
  - Expressions of the form \( a_1 + a_2, a_1 \times a_2 \) or \( a_1 \) terminate if their constituent expressions \( a_1, a_2 \) terminate
- **Problem:** circular!

### Ordinary Induction

- **Mathematical induction:** a property \( P(n) \) holds for all \( n \geq 1 \) if
  - \( P(1) \) (base case)
  - \( \forall n \geq 1. P(n) \Rightarrow P(n+1) \) (inductive step)
- **Inductive hypothesis:** \( P(n) \)
- **Strategy:**
  1. prove base case
  2. show \( P(n+1) \) is true if inductive hypothesis \( P(n) \) holds

### Course-of-values induction

- **Course-of-values induction:** a property \( P(n) \) holds for all \( n \geq 1 \) if
  - \( P(1) \) (base case)
  - \( \forall n \geq 1. (\forall n \leq L, P(n')) \Rightarrow P(n+1) \) (inductive step)
- **Inductive hypothesis:** \( \forall n \leq L. P(n') \)
- **Often easier to prove**

### Soundness

- **Course-of-values rule:**
  - \( P(1) \)
  - \( \forall n \geq 1. (\forall n \leq L. P(n')) \Rightarrow P(n+1) \)
  - \( \forall n \geq 1. P(n) \)
- **Idea:** introduce new predicate \( P'(n) \):
  - \( P'(n) = \forall n \leq L. P(n') \)
- **Lemmas:** \( P'(n) \Rightarrow P(n), P(1) \Rightarrow P'(1) \)

### Structural Induction

- Property \( P(e) \) holds for all exprs \( e \) if
  - \( P(e) \) holds for all expression forms \( e \) with no sub-expressions (e.g. \( n, X \))
  - Given expression form \( e \) with sub-expressions \( e_i \)
    - \( (a_1 + a_2) \) or \( (a_1 \times a_2) \). \( P(e) \) holds assuming \( P(e_i) \) holds for all \( e_i \)
  - \( P'(a_1 + a_2) = \exists n. (a_1 + a_2, n) \rightarrow^* (n, \sigma) \)
  - **Assume:**
    - \( n \cdot (a_1 + a_2, n) \rightarrow^* (n, \sigma) \)
    - \( n \cdot (a_1 \cdot a_2, n) \rightarrow^* (n, \sigma) \)
    - \( n \cdot (a_1, a_2) \rightarrow^* (n, \sigma) \)
    - \( n \cdot (a_1, a_2) \rightarrow^* (n, \sigma) \)
      (axiom)
    - \( \exists n. (n \cdot (a_1, a_2), n) \rightarrow^* (n, \sigma) \)
    - \( (\Rightarrow) \) lemmas
**Induction Hypothesis**

- Use course-of-values induction on size of expression (height of abstract syntax)
- P(n) is “all expressions of size n terminate”
- P(1) clearly true (n, X)
- Induction step: prove P(n) + P(n) of size n terminates
- Induction hypothesis: a₀, a₁ terminate (must be smaller than n)

**Induction on Derivations**

- Sometimes proof requires induction on height of derivation
- Example: commands in IMP are deterministic
- Want to show: ∀ n, σ₁, σ₂, σ₂’ ∈ C. (⟨ c, σ ⟩ ⊕ σ₁ & ⟨ c, σ ⟩ ⊕ σ₂’ ⇒ σ₁ = σ₂’)

**Proof of Determinism**

- Every command that terminates has a large-step semantics derivation (proof tree) with finite height
- Height of derivation tree is longest chain from conclusion (root) to any axiom (leaf)
- Let P(n) be statement “all commands whose derivation has height n are deterministic”
- P(n) = ∀ d, d’. (height(d) = n & d = ⟨ c, σ ⟩ ⊕ σ₁ & d’ = ⟨ c, σ ⟩ ⊕ σ₂ ⇒ σ₁ = σ₂’)

**Base Case**

- Statement about derivations (∀ n) implies desired statement about commands
- P(1): skip, X := a
- Inductive step: consider derivations of ⟨ c, σ ⟩ ⊕ σ₁ with height n for commands ; , if, while

**Inductive step for ;**

- Now suppose d is derivation of ⟨ c₀ ; c₁, σ ⟩ ⊕ σ₁ with height n, d’ derivation of ⟨ c₀ ; c₁, σ ⟩ ⊕ σ₂’
- Inductive hypothesis: σ₂ = σ₂’, then σ₁ = σ’₁

**Inductive step for if**

- Assume (if b then c₀ else c₁, σ) ⊕ σ₁’
  - (if b then c₀ else c₁, σ) ⊕ σ₂’
- Assume booleans are deterministic: b evaluates the same way for both
- WLOG derivations look like
- Sub-derivations:
  - ⟨ c₀, σ ⟩ ⊕ σ’
- (height < n)
- Therefore, σ’ = σ’’
Inductive step for while

- Assume \( \langle \text{while } b \text{ do } c, \sigma \rangle \cup \sigma_1 \)
- Derivations look like

\[
\begin{align*}
\langle b, \sigma \rangle \cup \text{true} & \quad \langle c, \sigma \rangle \cup \sigma_2 \quad \langle \text{while } b \text{ do } c, \sigma \rangle \cup \sigma_1 \\
\langle b, \sigma \rangle \cup \text{true} & \quad \langle c, \sigma \rangle \cup \sigma_2' \quad \langle \text{while } b \text{ do } c, \sigma \rangle \cup \sigma_1' \\
\end{align*}
\]

while

- Assume \( \langle \text{while } b \text{ do } c, \sigma \rangle \cup \sigma_1 \)
- Derivations look like

\[
\begin{align*}
\langle b, \sigma \rangle \cup \text{true} & \quad \langle c, \sigma \rangle \cup \sigma_2 \quad \langle \text{while } b \text{ do } c, \sigma \rangle \cup \sigma_1 \\
\langle b, \sigma \rangle \cup \text{true} & \quad \langle c, \sigma \rangle \cup \sigma_2' \quad \langle \text{while } b \text{ do } c, \sigma \rangle \cup \sigma_1' \\
\end{align*}
\]

Induction

- Structural induction:
  - prove that a property holds of all language atoms
  - prove that it holds for each kind of expression if it holds of the parts of the expression
  \( \Rightarrow \) property holds for all expressions
- Induction on derivations
  - prove it holds for derivations that are axioms
  - prove property holds if it holds for every derivation (evaluation) of parts of an expression
  \( \Rightarrow \) property holds for all derivations

Observation

- These two forms of induction are very similar — both operate on inductively defined sets (syntax, evaluations)

Expression inference rules

BNF spec for arithmetic expressions in IMP:

\[
a ::= n \mid X \mid a_0 + a_1 \mid a_0 - a_1 \mid a_0 \times a_1
\]

Let \( A \) be the set of all arithmetic expressions. **Inductive definition of \( A \)** via inference rules:

- **Axioms:**
  
  \[
  \begin{align*}
  n & \\
  X &
  \end{align*}
  \]

- **Rules:**
  
  \[
  \begin{align*}
  a_0 + a_1 & \\
  a_0 - a_1 & \\
  a_0 \times a_1 &
  \end{align*}
  \]

Expression derivation tree

- Every legal expression now has a derivation tree.
- Example: \((2+3) \times (4-x)\)

\[
\begin{align*}
2 & \quad 3 & \quad 4 & \quad x \\
2+3 & \quad 4 \times x \\
(2+3) \times (4-x) &
\end{align*}
\]

- Structural induction is induction on syntactic derivations!
Summary

- Any proof system (inference rules) is an inductive definition of a set
- Rule induction can be applied to any inductive definition
- Examples: structural induction, induction on derivations are both instances of this approach
- We will use rule induction for other proof systems in course (e.g., type-checking rules)