Operational Semantics

Hongfei Fu

John Hopcroft Center for Computer Science
Shanghai Jiao Tong University

Mar. 11th, 2021

Previous Lecture

- an introduction to operational semantics
- rules and derivation trees

Today's Topic

- equivalence of commands through derivations
- one-step operational semantics
- mathematical induction over derivations

Equivalence of Commands

textbook, Page 19 – 24

Equivalence of Commands

Definition

- definition: $\langle c, \sigma \rangle \to \sigma'$ iff there is a derivation tree with conclusion $\langle c, \sigma \rangle \to \sigma'$.
- equivalence: $c \sim c'$ iff $\forall \sigma, \sigma'. (\langle c, \sigma \rangle \to \sigma' \Leftrightarrow \langle c', \sigma \rangle \to \sigma')$

Equivalence of Commands

- $w =$ **while** b **do** c;
- $w \sim$ **if** b **then** c; w **else skip**

Equivalence of Commands

- $w =$ **while** b **do** c;
- for all states σ, σ',
$$\langle w, \sigma \rangle \to \sigma' \textit{ iff } \langle \textbf{if } b \textbf{ then } c; w \textbf{ else skip}, \sigma \rangle \to \sigma'.$$

Equivalence of Commands

Proof

- $\langle w, \sigma \rangle \to \sigma'$ implies $\langle \textbf{if } b \textbf{ then } c; w \textbf{ else skip}, \sigma \rangle \to \sigma'$.

Equivalence of Commands

- $\langle w, \sigma \rangle \to \sigma'$ implies $\langle \textbf{if } b \textbf{ then } c; w \textbf{ else skip}, \sigma \rangle \to \sigma'$.
- Case 1: $\langle b, \sigma \rangle \to \textbf{false}$

Equivalence of Commands

- $\langle w, \sigma \rangle \to \sigma'$ implies $\langle \textbf{if } b \textbf{ then } c; w \textbf{ else skip}, \sigma \rangle \to \sigma'$.
- Case 1: $\langle b, \sigma \rangle \to \textbf{false}$
- from the rule for while-loop:

$$\dfrac{\dfrac{\vdots}{\langle b, \sigma \rangle \to \textbf{false}}}{\langle w, \sigma \rangle \to \sigma} \quad (\sigma' = \sigma)$$

Equivalence of Commands

- $\langle w, \sigma \rangle \to \sigma'$ implies $\langle \textbf{if } b \textbf{ then } c; w \textbf{ else skip}, \sigma \rangle \to \sigma'$.
- Case 1: $\langle b, \sigma \rangle \to \textbf{false}$
- from the rule for while-loop:

$$\frac{\dfrac{\vdots}{\langle b, \sigma \rangle \to \textbf{false}}}{\langle w, \sigma \rangle \to \sigma} \quad (\sigma' = \sigma)$$

- thus:

$$\frac{\dfrac{\vdots}{\langle b, \sigma \rangle \to \textbf{false}} \quad \dfrac{}{\langle \textbf{skip}, \sigma \rangle \to \sigma}}{\langle \textbf{if } b \textbf{ then } c; w \textbf{ else skip}, \sigma \rangle \to \sigma}$$

Equivalence of Commands

- $\langle w, \sigma \rangle \to \sigma'$ implies $\langle \textbf{if } b \textbf{ then } c; w \textbf{ else skip}, \sigma \rangle \to \sigma'$.
- Case 2: $\langle b, \sigma \rangle \to \textbf{true}$

Equivalence of Commands

- $\langle w, \sigma \rangle \to \sigma'$ implies $\langle \textbf{if } b \textbf{ then } c; w \textbf{ else skip}, \sigma \rangle \to \sigma'$.
- Case 2: $\langle b, \sigma \rangle \to \textbf{true}$
- from the rule for while-loop:

$$\frac{\vdots \qquad \vdots \qquad \vdots}{\langle b, \sigma \rangle \to \textbf{true} \quad \langle c, \sigma \rangle \to \sigma'' \quad \langle w, \sigma'' \rangle \to \sigma'} \\ \langle w, \sigma \rangle \to \sigma'$$

Equivalence of Commands

- $\langle w, \sigma \rangle \to \sigma'$ implies $\langle \textbf{if } b \textbf{ then } c; w \textbf{ else skip}, \sigma \rangle \to \sigma'$.
- Case 2: $\langle b, \sigma \rangle \to \textbf{true}$
- from the rule for while-loop:

$$\frac{\vdots \quad \vdots \quad \vdots}{\langle b, \sigma \rangle \to \textbf{true} \quad \langle c, \sigma \rangle \to \sigma'' \quad \langle w, \sigma'' \rangle \to \sigma'} \\ \langle w, \sigma \rangle \to \sigma'$$

- it follows that:

$$\frac{\vdots \quad \vdots}{\langle c, \sigma \rangle \to \sigma'' \quad \langle w, \sigma'' \rangle \to \sigma'} \\ \langle c; w, \sigma \rangle \to \sigma'$$

Equivalence of Commands

- $\langle w, \sigma \rangle \to \sigma'$ implies $\langle \textbf{if } b \textbf{ then } c; w \textbf{ else skip}, \sigma \rangle \to \sigma'$.
- Case 2: $\langle b, \sigma \rangle \to \textbf{true}$
- it follows that:

$$\frac{\vdots \qquad \vdots}{\langle c, \sigma \rangle \to \sigma'' \quad \langle w, \sigma'' \rangle \to \sigma'} \\ \langle c; w, \sigma \rangle \to \sigma'$$

Equivalence of Commands

- $\langle w, \sigma \rangle \to \sigma'$ implies $\langle \textbf{if } b \textbf{ then } c; w \textbf{ else skip}, \sigma \rangle \to \sigma'$.
- Case 2: $\langle b, \sigma \rangle \to \textbf{true}$
- it follows that:

$$\dfrac{\dfrac{\vdots}{\langle c, \sigma \rangle \to \sigma''} \quad \dfrac{\vdots}{\langle w, \sigma'' \rangle \to \sigma'}}{\langle c; w, \sigma \rangle \to \sigma'}$$

- hence:

$$\dfrac{\dfrac{\vdots}{\langle b, \sigma \rangle \to \textbf{true}} \quad \dfrac{\dfrac{\vdots}{\langle c, \sigma \rangle \to \sigma''} \quad \dfrac{\vdots}{\langle w, \sigma'' \rangle \to \sigma'}}{\langle c; w, \sigma \rangle \to \sigma'}}{\langle \textbf{if } b \textbf{ then } c; w \textbf{ else skip}, \sigma \rangle \to \sigma'}$$

Equivalence of Commands

- $\langle \textbf{if } b \textbf{ then } c; w \textbf{ else skip}, \sigma \rangle \to \sigma'$ implies $\langle w, \sigma \rangle \to \sigma'$.
- Case 1:

$$\dfrac{\dfrac{\vdots}{\langle b, \sigma \rangle \to \textbf{false}} \quad \dfrac{}{\langle \textbf{skip}, \sigma \rangle \to \sigma}}{\langle \textbf{if } b \textbf{ then } c; w \textbf{ else skip}, \sigma \rangle \to \sigma}$$

- Case 2:

$$\dfrac{\dfrac{\vdots}{\langle b, \sigma \rangle \to \textbf{true}} \quad \dfrac{\vdots}{\langle c; w, \sigma \rangle \to \sigma'}}{\langle \textbf{if } b \textbf{ then } c; w \textbf{ else skip}, \sigma \rangle \to \sigma'}$$

Equivalence of Commands

- $\langle \textbf{if } b \textbf{ then } c; w \textbf{ else skip}, \sigma \rangle \to \sigma'$ implies $\langle w, \sigma \rangle \to \sigma'$.
- Case 1:

$$\dfrac{\dfrac{\vdots}{\langle b, \sigma \rangle \to \textbf{false}} \quad \dfrac{}{\langle \textbf{skip}, \sigma \rangle \to \sigma}}{\langle \textbf{if } b \textbf{ then } c; w \textbf{ else skip}, \sigma \rangle \to \sigma}$$

$$\dfrac{\dfrac{\vdots}{\langle b, \sigma \rangle \to \textbf{false}}}{\langle w, \sigma \rangle \to \sigma}$$

Equivalence of Commands

- $\langle \textbf{if } b \textbf{ then } c; w \textbf{ else skip}, \sigma \rangle \to \sigma'$ implies $\langle w, \sigma \rangle \to \sigma'$.
- Case 2:

$$\dfrac{\dfrac{\vdots}{\langle b, \sigma \rangle \to \textbf{true}} \quad \dfrac{\dfrac{\vdots}{\langle c, \sigma \rangle \to \sigma''} \quad \dfrac{\vdots}{\langle w, \sigma'' \rangle \to \sigma'}}{\langle c; w, \sigma \rangle \to \sigma'}}{\langle \textbf{if } b \textbf{ then } c; w \textbf{ else skip}, \sigma \rangle \to \sigma'}$$

$$\dfrac{\dfrac{\vdots}{\langle b, \sigma \rangle \to \textbf{true}} \quad \dfrac{\vdots}{\langle c, \sigma \rangle \to \sigma''} \quad \dfrac{\vdots}{\langle w, \sigma'' \rangle \to \sigma'}}{\langle \textbf{while } b \textbf{ do } c, \sigma \rangle \to \sigma'}$$

Small-Step Operational Semantics

textbook, Page 24 – 26

One-Step Operational Semantics

Motivation

- Full-step operational semantics ignores internal execution.
- Single-step execution are important in parallel environments.

One-Step Operational Semantics

Arithmetic Expressions

- big-step semantics: $\langle a, \sigma \rangle \to n$
- small-step semantics: $\langle a, \sigma \rangle \to_1 \langle a', \sigma \rangle$

One-Step Operational Semantics

Arithmetic Expressions

$$\frac{\langle a_0, \sigma\rangle \to_1 \langle a_0', \sigma\rangle}{\langle a_0 + a_1, \sigma\rangle \to_1 \langle a_0' + a_1, \sigma\rangle}$$

$$\frac{\langle a_1, \sigma\rangle \to_1 \langle a_1', \sigma\rangle}{\langle n + a_1, \sigma\rangle \to_1 \langle n + a_1', \sigma\rangle}$$

$$\frac{}{\langle n + m, \sigma\rangle \to_1 \langle p, \sigma\rangle} \quad (p = m + n)$$

One-Step Operational Semantics

Commands

- big-step semantics: $\langle c, \sigma \rangle \to \sigma'$
- small-step semantics: $\langle c, \sigma \rangle \to_1 \langle c', \sigma' \rangle$

One-Step Operational Semantics

Commands

$$\frac{\langle a, \sigma \rangle \to_1 \langle a', \sigma \rangle}{\langle X := a, \sigma \rangle \to_1 \langle X := a', \sigma \rangle} \quad (a' \notin \mathbb{Z})$$

$$\frac{\langle a, \sigma \rangle \to_1 \langle n, \sigma \rangle}{\langle X := a, \sigma \rangle \to_1 \sigma\,[n/X]} \quad (n \in \mathbb{Z})$$

One-Step Operational Semantics

Commands

$$\frac{\langle c_1, \sigma \rangle \to_1 \langle c_1', \sigma' \rangle}{\langle c_1; c_2, \sigma \rangle \to_1 \langle c_1'; c_2, \sigma' \rangle}$$

$$\frac{\langle c_1, \sigma \rangle \to_1 \sigma'}{\langle c_1; c_2, \sigma \rangle \to_1 \langle c_2, \sigma' \rangle}$$

One-Step Operational Semantics

Question

What about if-branches and while-loops?

Principles of Induction

textbook, Page 27 – 38

Principles of Induction

- mathematical induction
- structural induction
- induction on derivation trees
- well-founded induction

Mathematical Induction

Description

- P: a property (or predicate, assertion, formula) over natural numbers

Mathematical Induction

Description

- P: a property (or predicate, assertion, formula) over natural numbers
- illustration: if $P(0)$ and $P(n) \Rightarrow P(n+1)$ for all natural numbers n, then it holds that $P(n)$ for all natural numbers n.

Mathematical Induction

Description

- P: a property (or predicate, assertion, formula) over natural numbers
- illustration: if $P(0)$ and $P(n) \Rightarrow P(n+1)$ for all natural numbers n, then it holds that $P(n)$ for all natural numbers n.
- formal statement:

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

Mathematical Induction

Proof

Mathematical Induction

Proof

- $P = \{n \in \mathbb{N} \mid P(n)\} \subseteq \mathbb{N}$;

Mathematical Induction

Proof

- $P = \{n \in \mathbb{N} \mid P(n)\} \subseteq \mathbb{N}$;
- P is an inductive set: $0 \in P$ and $n \in P \Rightarrow n + 1 \in P$;

Mathematical Induction

Proof

- $P = \{n \in \mathbb{N} \mid P(n)\} \subseteq \mathbb{N}$;
- P is an inductive set: $0 \in P$ and $n \in P \Rightarrow n + 1 \in P$;
- $\mathbb{N}$ is the smallest inductive set: $\mathbb{N} \subseteq P$;

Mathematical Induction

Proof

- $P = \{n \in \mathbb{N} \mid P(n)\} \subseteq \mathbb{N}$;
- P is an inductive set: $0 \in P$ and $n \in P \Rightarrow n + 1 \in P$;
- $\mathbb{N}$ is the smallest inductive set: $\mathbb{N} \subseteq P$;
- $P = \mathbb{N}$: i.e., $\forall n \in \mathbb{N}.P(n)$;

Mathematical Induction

Course-of-Values Induction

Mathematical Induction

Course-of-Values Induction

- target: $\forall n.P(n)$;

Mathematical Induction

Course-of-Values Induction

- target: $\forall n.P(n)$;
- variant form: $Q(n) := \forall k < n.P(k)$;

Mathematical Induction

Course-of-Values Induction

- target: $\forall n.P(n)$;
- variant form: $Q(n) := \forall k < n.P(k)$;
- equivalence: $\forall n.P(n)$ is equivalent to $\forall n.Q(n)$;

Mathematical Induction

Course-of-Values Induction

- target: $\forall n.P(n)$;
- variant form: $Q(n) := \forall k < n.P(k)$;
- equivalence: $\forall n.P(n)$ is equivalent to $\forall n.Q(n)$;
- base step: $Q(0)$ is vacuously true;

Mathematical Induction

Course-of-Values Induction

- target: $\forall n.P(n)$;
- variant form: $Q(n) := \forall k < n.P(k)$;
- equivalence: $\forall n.P(n)$ is equivalent to $\forall n.Q(n)$;
- base step: $Q(0)$ is vacuously true;
- the induction step: $Q(n) \Rightarrow Q(n+1)$ for all n;
- the induction step: $(\forall k < n.P(k)) \Rightarrow P(n)$ for all n;

Mathematical Induction

Course-of-Values Induction

- target: $\forall n.P(n)$;
- variant form: $Q(n) := \forall k < n.P(k)$;
- equivalence: $\forall n.P(n)$ is equivalent to $\forall n.Q(n)$;
- base step: $Q(0)$ is vacuously true;
- the induction step: $Q(n) \Rightarrow Q(n+1)$ for all n;
- the induction step: $(\forall k < n.P(k)) \Rightarrow P(n)$ for all n;

Question

Where do we require that $P(0)$ holds?

Well-Founded Induction

Definition

- A: a set
- $\prec \subseteq A \times A$: a binary relation on A

The relation $\prec$ is well-founded if:

- there is no infinite descending sequence $\ldots \prec a_n \prec \ldots \prec a_1 \prec a_0$ in A;

Well-Founded Induction

Definition

- A: a set
- $\prec \subseteq A \times A$: a binary relation on A

The relation $\prec$ is well-founded if:

- there is no infinite descending sequence $\ldots \prec a_n \prec \ldots \prec a_1 \prec a_0$ in A;
- well-foundedness implies irreflexibility: $\forall a \in A. a \not\prec a$.

Well-Founded Induction

Minimal Elements

- A: a set
- $\prec \subseteq A \times A$: a binary relation on A
- $Q \subseteq A$: a subset of A
- $u \in Q$: an element of Q

The element u is a minimal element in Q if $\forall v \in Q. (v \not\prec u)$.

Well-Founded Induction

Minimal Elements

- A: a set
- $\prec \subseteq A \times A$: a binary relation on A
- $Q \subseteq A$: a subset of A
- $u \in Q$: an element of Q

The element u is a minimal element in Q if $\forall v \in Q. (v \not\prec u)$.

Proposition

The relation $\prec$ is well-founded iff any nonempty subset $Q \subseteq A$ has a minimal element.

Well-Founded Induction

Proposition

- A: a set
- $\prec$: a binary relation on A

The relation $\prec$ is well-founded iff any nonempty subset $Q \subseteq A$ has a minimal element.

Proof for "$\Leftarrow$" (by contradiction)

Well-Founded Induction

Proposition

- A: a set
- $\prec$: a binary relation on A

The relation $\prec$ is well-founded iff any nonempty subset $Q \subseteq A$ has a minimal element.

Proof for "$\Leftarrow$" (by contradiction)

- Suppose that $\prec$ is not well-founded.

Well-Founded Induction

Proposition

- A: a set
- $\prec$: a binary relation on A

The relation $\prec$ is well-founded iff any nonempty subset $Q \subseteq A$ has a minimal element.

Proof for "$\Leftarrow$" (by contradiction)

- Suppose that $\prec$ is not well-founded.
- There exists an infinite sequence $\ldots \prec a_n \prec \ldots \prec a_1 \prec a_0$.

Well-Founded Induction

Proposition

- A: a set
- $\prec$: a binary relation on A

The relation $\prec$ is well-founded iff any nonempty subset $Q \subseteq A$ has a minimal element.

Proof for "$\Leftarrow$" (by contradiction)

- Suppose that $\prec$ is not well-founded.
- There exists an infinite sequence $\ldots \prec a_n \prec \ldots \prec a_1 \prec a_0$.
- The set $\{a_0, a_1, \ldots, a_n, \ldots\}$ does not have a minimal element.

Well-Founded Induction

Proposition

- A: a set
- $\prec$: a binary relation on A

The relation $\prec$ is well-founded iff any nonempty subset $Q \subseteq A$ has a minimal element.

Proof for "$\Rightarrow$" (by contradiction)

Well-Founded Induction

Proposition

- A: a set
- $\prec$: a binary relation on A

The relation $\prec$ is well-founded iff any nonempty subset $Q \subseteq A$ has a minimal element.

Proof for "$\Rightarrow$" (by contradiction)

- Suppose that there exists a nonempty subset $Q \subseteq A$ having no minimal elements, i.e., $\forall u \in Q. \exists v \in Q. v \prec u$.

Well-Founded Induction

Proposition

- A: a set
- $\prec$: a binary relation on A

The relation $\prec$ is well-founded iff any nonempty subset $Q \subseteq A$ has a minimal element.

Proof for "$\Rightarrow$" (by contradiction)

- Suppose that there exists a nonempty subset $Q \subseteq A$ having no minimal elements, i.e., $\forall u \in Q. \exists v \in Q. v \prec u$.
- Then starting from any u_0, one can construct a sequence $u_0, u_1, \ldots$ of infinite descending elements in A.

Well-Founded Induction

Statement

- $\prec$: a well-founded binary relation on a set A
- P: a property on elements of A (a subset of A)
- the principle:

$$\forall a \in A.P(a) \text{ iff } \forall a \in A.\,[(\forall b \prec a.P(b)) \Rightarrow P(a)]$$

Proof for "$\Rightarrow$"

Straightforward.

Well-Founded Induction

Statement

- $\prec$: a well-founded binary relation on a set A
- P: a property on elements of A (a subset of A)
- the principle:

$$\forall a \in A.P(a) \text{ iff } \forall a \in A.\,[(\forall b \prec a.P(b)) \Rightarrow P(a)]$$

Proof for "$\Leftarrow$" (by contradiction)

- Suppose that $\exists a.\neg P(a)$ and define $Q := \{a \in A \mid \neg P(a)\}$.
- Q is nonempty and hence has a minimal element a^*.
- $(\forall b \prec a^*.b \notin Q)$, and hence $(\forall b \prec a^*.P(b))$.
- From $(\forall b \prec a^*.P(b)) \Rightarrow P(a^*)$, we have $P(a^*)$.

Well-Founded Induction

Example

- $A = \mathbb{N}$, $\prec = \{(n, n+1) \mid n \in \mathbb{N}\}$: mathematical induction
- $A = \mathbb{N}$, $\prec = \{(m, n) \in \mathbb{N} \times \mathbb{N} \mid m < n\}$: course-of-value induction

Structural Induction

Motivation

- mathematical induction: inductive proofs on natural numbers
- structural induction: inductive proofs on syntactic structures

Structural Induction

Arithmetic Expressions

Structural Induction

Arithmetic Expressions

- **Aexp**: the set of all arithmetic expressions

Structural Induction

Arithmetic Expressions

- **Aexp**: the set of all arithmetic expressions
- $\prec$: $a_0 \prec a_1$ iff a_0 is an immediate syntactical child of a_1

Structural Induction

Arithmetic Expressions

- **Aexp**: the set of all arithmetic expressions
- $\prec$: $a_0 \prec a_1$ iff a_0 is an immediate syntactical child of a_1
- P: a property on arithmetic expressions

Structural Induction

Arithmetic Expressions

- **Aexp**: the set of all arithmetic expressions
- $\prec$: $a_0 \prec a_1$ iff a_0 is an immediate syntactical child of a_1
- P: a property on arithmetic expressions
- well-founded induction:

$$\forall a \in \textbf{Aexp}.\, [(\forall b \prec a.P(b)) \Rightarrow P(a)] \text{ implies } \forall a \in \textbf{Aexp}.P(a)$$

Structural Induction

Arithmetic Expressions

Structural Induction

Arithmetic Expressions

- bases step: P holds at atomic arithmetic expressions n, X.

Structural Induction

Arithmetic Expressions

- bases step: P holds at atomic arithmetic expressions n, X.
- inductive step: if P holds at arithmetic expressions a_0, a_1, then P also holds at $a_0 + a_1, a_0 - a_1, a_0 \times a_1$.

Structural Induction

Arithmetic Expressions

- bases step: P holds at atomic arithmetic expressions n, X.
- inductive step: if P holds at arithmetic expressions a_0, a_1, then P also holds at $a_0 + a_1, a_0 - a_1, a_0 \times a_1$.
- consequence: P holds at all arithmetic expressions.

Structural Induction

Example

For all arithmetic expressions a, states σ and integers m, m',

$$\langle a, \sigma \rangle \to m \wedge \langle a, \sigma \rangle \to m' \Rightarrow m = m' \quad .$$

Structural Induction

The Inductive Proof

- base step: $\langle n, \sigma \rangle \to n$, $\langle X, \sigma \rangle \to \sigma(X)$

Structural Induction

The Inductive Proof

- base step: $\langle n, \sigma \rangle \to n, \quad \langle X, \sigma \rangle \to \sigma(X)$
- inductive step:

$$\frac{\langle a_0, \sigma \rangle \to n_0, \quad \langle a_1, \sigma \rangle \to n_1}{\langle a_0 + a_1, \sigma \rangle \to n_0 + n_1}$$

Structural Induction

Boolean Expressions

- $\forall b, \sigma, t, t'. [(\langle b, \sigma \rangle \to t \ \& \ \langle b, \sigma \rangle \to t') \Rightarrow t = t']$.

Structural Induction

Proposition

$\forall c, \sigma, \sigma', \sigma''. [(\langle c, \sigma \rangle \to \sigma' \ \& \ \langle c, \sigma \rangle \to \sigma'') \Rightarrow \sigma' = \sigma'']$.

Structural Induction

Proposition

$\forall c, \sigma, \sigma', \sigma''. [(\langle c, \sigma \rangle \to \sigma' \ \& \ \langle c, \sigma \rangle \to \sigma'') \Rightarrow \sigma' = \sigma'']$.

Question

Can we prove this proposition through structural induction?

Question

Proposition

$$\forall c, \sigma, \sigma', \sigma''. [(\langle c, \sigma \rangle \to \sigma' \ \& \ \langle c, \sigma \rangle \to \sigma'') \Rightarrow \sigma' = \sigma''] \ .$$

Rules for While Loops

$$\frac{\langle b, \sigma \rangle \to \textbf{false}}{\langle \textbf{while } b \textbf{ do } c, \sigma \rangle \to \sigma}$$

$$\frac{\langle b, \sigma \rangle \to \textbf{true}, \ \langle c, \sigma \rangle \to \sigma'', \ \langle \textbf{while } b \textbf{ do } c, \sigma'' \rangle \to \sigma'}{\langle \textbf{while } b \textbf{ do } c, \sigma \rangle \to \sigma'}$$

Induction on Derivation Trees

- A: the set of all derivation trees (or derivations)
- $\prec$: $r_0 \prec r_1$ iff r_0 is a proper sub-derivation tree of r_1
- well-founded induction:

$$\forall r \in A. \left[\left(\forall r' \prec r. P(r') \right) \Rightarrow P(r) \right] \text{ implies } \forall r \in A. P(r)$$

Induction on Derivation Trees

Rule Instance (X/y)

- X: premise (a finite set of elements)
- y: conclusion (a single element)

Induction on Derivation Trees

Axiom Instances: $\emptyset/y$

$$\frac{}{y}$$

Other Rule Instances: $\{x_1, \ldots, x_n\}/y$

$$\frac{x_1, \ldots, x_n}{y}$$

Induction on Derivation Trees

Derivation Trees

- R: a set of rule instances
- y: an element

An R-derivation of y is

- either a rule instance $(\emptyset / y)$
- or $(\{d_1, \ldots, d_n\} / y)$ such that
  - $(\{x_1, \ldots, x_n\} / y)$ is a rule instance
  - each d_i is a (smaller) R-derivation of x_i

Induction on Derivation Trees

Notations

- R: a set of rule instances
- d: an R-derivation
- y: an element

Then we write

- $d \Vdash_R y$: d is an R-derivation of y.
- $\Vdash_R y$: $d \Vdash_R y$ for some derivation d.
- $d \Vdash y, \Vdash y$: omission of R

Induction on Derivation Trees

Notations

- R: a set of rule instances
- d: an R-derivation
- y: an element

Then we have

- $(\emptyset/y) \Vdash_R y$ if $(\emptyset/y) \in R$
- $(\{d_1, \ldots, d_n\}/y) \Vdash_R y$ if $(\{x_1, \ldots, x_n\}/y) \in R$ and $d_1 \Vdash_R x_1, \ldots, d_n \Vdash_R x_n$

Induction on Derivation Trees

The Well-Founded Relation on Derivation Trees

- d, d': derivations

$d' \prec d$ if d' is a proper sub-derivation of d.

Induction on Derivation Trees

Proposition

- $\forall c, \sigma, \sigma', \sigma''. [(\langle c, \sigma \rangle \to \sigma' \ \& \ \langle c, \sigma \rangle \to \sigma'') \Rightarrow \sigma' = \sigma'']$.

Induction on Derivation Trees

Proposition

- $\forall c, \sigma, \sigma', \sigma''. [(\langle c, \sigma \rangle \to \sigma' \ \& \ \langle c, \sigma \rangle \to \sigma'') \Rightarrow \sigma' = \sigma'']$.
- $P(d) := \forall c, \sigma, \sigma', \sigma''. [(d \Vdash \langle c, \sigma \rangle \to \sigma' \wedge \langle c, \sigma \rangle \to \sigma'') \Rightarrow \sigma' = \sigma'']$

Induction on Derivation Trees

Proposition

- $\forall c, \sigma, \sigma', \sigma''. [(\langle c, \sigma \rangle \to \sigma' \ \& \ \langle c, \sigma \rangle \to \sigma'') \Rightarrow \sigma' = \sigma'']$.
- $P(d) := \forall c, \sigma, \sigma', \sigma''. [(d \Vdash \langle c, \sigma \rangle \to \sigma' \wedge \langle c, \sigma \rangle \to \sigma'') \Rightarrow \sigma' = \sigma'']$
- the goal: $\forall d' \prec d. P(d')$ implies $P(d)$

Induction on Derivation Trees

- $\forall c, \sigma, \sigma', \sigma''. [(\langle c, \sigma \rangle \to \sigma' \ \& \ \langle c, \sigma \rangle \to \sigma'') \Rightarrow \sigma' = \sigma'']$.
- base step:

$$\frac{}{\langle \textbf{skip}, \sigma \rangle \to \sigma} \qquad \frac{\dfrac{\vdots}{\langle a, \sigma \rangle \to m}}{\langle X := a, \sigma \rangle \to \sigma\,[m/X]}$$

- inductive step:

$$\frac{\dfrac{\vdots}{\langle b, \sigma \rangle \to \textbf{true}} \qquad \dfrac{\vdots}{\langle c, \sigma \rangle \to \sigma''} \qquad \dfrac{\vdots}{\langle \textbf{while}\ b\ \textbf{do}\ c, \sigma'' \rangle \to \sigma'}}{\langle \textbf{while}\ b\ \textbf{do}\ c, \sigma \rangle \to \sigma'}$$

Program Termination

A Variant of Euclidean's Algorithm

$$\begin{array}{l}\textbf{while}\ \ \neg(M = N)\ \ \textbf{do}\\ \quad \textbf{if}\ \ M \leq N\\ \qquad \textbf{then}\ \ N := N - M\\ \qquad \textbf{else}\ \ M := M - N\end{array}$$

Program Termination

A Variant of Euclidean's Algorithm

$$
\begin{array}{l}
\text{Euclid} \ = \\
\quad \textbf{while} \ \ \neg(M = N) \ \ \textbf{do} \\
\quad\quad \textbf{if} \ \ M \leq N \\
\quad\quad\quad \textbf{then} \ \ N := N - M \\
\quad\quad\quad \textbf{else} \ \ M := M - N
\end{array}
$$

Termination Property

$$\forall \sigma. \left[(\sigma(M) \geq 1 \wedge \sigma(N) \geq 1) \Rightarrow (\exists \sigma'. \langle \text{Euclid}, \sigma \rangle \rightarrow \sigma') \right]$$

Program Termination

Termination Property

$$\forall \sigma. [(\sigma(M) \geq 1 \wedge \sigma(N) \geq 1) \Rightarrow (\exists \sigma'. \langle \text{Euclid}, \sigma \rangle \rightarrow \sigma')]$$

Proof

Program Termination

Termination Property

$$\forall \sigma. [(\sigma(M) \geq 1 \land \sigma(N) \geq 1) \Rightarrow (\exists \sigma'. \langle \text{Euclid}, \sigma \rangle \to \sigma')]$$

Proof

- $A := \{\sigma \in \Sigma \mid \sigma(M) \geq 1 \land \sigma(N) \geq 1\}$.

Program Termination

Termination Property

$\forall \sigma. [(\sigma(M) \geq 1 \wedge \sigma(N) \geq 1) \Rightarrow (\exists \sigma'. \langle \text{Euclid}, \sigma \rangle \to \sigma')]$

Proof

- $A := \{\sigma \in \Sigma \mid \sigma(M) \geq 1 \wedge \sigma(N) \geq 1\}$.
- $\sigma \prec \sigma'$ iff the followings hold:
  1. $\sigma(M) \leq \sigma'(M)$ and $\sigma(N) \leq \sigma'(N)$;
  2. $\sigma \neq \sigma'$;

Program Termination

Termination Property

$\forall \sigma. [(\sigma(M) \geq 1 \wedge \sigma(N) \geq 1) \Rightarrow (\exists \sigma'. \langle \text{Euclid}, \sigma \rangle \rightarrow \sigma')]$

Proof

- $A := \{\sigma \in \Sigma \mid \sigma(M) \geq 1 \wedge \sigma(N) \geq 1\}$.
- $\sigma \prec \sigma'$ iff the followings hold:
  1. $\sigma(M) \leq \sigma'(M)$ and $\sigma(N) \leq \sigma'(N)$;
  2. $\sigma \neq \sigma'$;
- $P(\sigma) := \exists \sigma'. \langle \text{Euclid}, \sigma \rangle \rightarrow \sigma'$.

Program Termination

Termination Property

$\forall \sigma. [(\sigma(M) \geq 1 \wedge \sigma(N) \geq 1) \Rightarrow (\exists \sigma'. \langle \text{Euclid}, \sigma \rangle \rightarrow \sigma')]$

Proof

- $A := \{\sigma \in \Sigma \mid \sigma(M) \geq 1 \wedge \sigma(N) \geq 1\}$.
- $\sigma \prec \sigma'$ iff the followings hold:
  1. $\sigma(M) \leq \sigma'(M)$ and $\sigma(N) \leq \sigma'(N)$;
  2. $\sigma \neq \sigma'$;
- $P(\sigma) := \exists \sigma'. \langle \text{Euclid}, \sigma \rangle \rightarrow \sigma'$.
- our goal: prove $\forall \sigma \in A. P(\sigma)$ by

$$\forall \sigma \in A. \left[\left(\forall \sigma' \prec \sigma. P(\sigma') \right) \Rightarrow P(\sigma) \right]$$

Well-founded Induction

Proof

- $P(\sigma) := \exists \sigma'. \langle \text{Euclid}, \sigma \rangle \to \sigma'$.
- our goal: prove $\forall \sigma \in A. P(\sigma)$ by

$$\forall \sigma \in A. \left[\left(\forall \sigma' \prec \sigma. P(\sigma') \right) \Rightarrow P(\sigma) \right]$$

- Suppose that $\forall \sigma' \prec \sigma. P(\sigma')$.
- Case $\sigma(M) = \sigma(N)$:

$$\dfrac{\dfrac{\vdots}{\langle \neg M = N, \sigma \rangle \to \textbf{false}}}{\langle \text{Euclid}, \sigma \rangle \to \sigma}$$

Well-founded Induction

Proof

- $P(\sigma) := \exists \sigma'. \langle \text{Euclid}, \sigma \rangle \to \sigma'$.
- our goal: prove $\forall \sigma \in A. P(\sigma)$ by

$$\forall \sigma \in A. \left[\left(\forall \sigma' \prec \sigma. P(\sigma') \right) \Rightarrow P(\sigma) \right]$$

- Suppose that $\forall \sigma' \prec \sigma. P(\sigma')$.
- Case $\sigma(M) \neq \sigma(N)$:

$$\langle \textbf{if } M \leq N \textbf{ then } N := N - M \textbf{ else } M := M - N, \sigma \rangle \to \sigma''$$

where

$$\sigma'' = \begin{cases} \sigma \left[\sigma(N) - \sigma(M) / N \right] & \text{if } \sigma(N) \geq \sigma(M) \\ \sigma \left[\sigma(M) - \sigma(N) / M \right] & \text{otherwise} \end{cases}$$

and $\sigma'' \prec \sigma$;

Well-founded Induction

Proof

- $P(\sigma) := \exists \sigma'. \langle \text{Euclid}, \sigma \rangle \to \sigma'$.
- Prove $\forall \sigma \in A. [(\forall \sigma' \prec \sigma. P(\sigma')) \Rightarrow P(\sigma)]$.
- Suppose that $\forall \sigma' \prec \sigma. P(\sigma')$.
- Case $\sigma(M) \neq \sigma(N)$:
  - $\langle \neg(M = N), \sigma \rangle \to$ **true**;
  - $\langle$ **if** $M < N$ **then** $N := N - M$ **else** $M := M - N, \sigma \rangle \to \sigma''$ and $\sigma'' \prec \sigma$;
  - $\langle \text{Euclid}, \sigma'' \rangle \to \sigma'$ for some σ';
- Conclusion: $\langle \text{Euclid}, \sigma \rangle \to \sigma'$

Summary

- equivalence reasoning using rules
- small-step semantics
- principles of induction
  - mathematical induction
  - induction on derivation trees
  - well-founded induction
- proving program property through induction
- proving program termination through induction

Homework Problem 1

Problem

Consider the command

$$c = \textbf{while } X \leq 100 \textbf{ do } X := X + 2$$

where X is a location (program variable). For each initial state σ, determine through induction principle the state σ' such that $\langle c, \sigma \rangle \to \sigma'$ and verify your answer.

Homework Problem 2

Problem

Consider the command

$$c = \textbf{while } (X \geq 0 \wedge Y \geq 0) \textbf{ do}$$
$$\textbf{if } b \textbf{ then } Y := Y - 1$$
$$\textbf{else } (X := X - 1; Y := a)$$

where X, Y are locations (program variables), b is an arbitrary boolean expression and a is an arbitrary arithmetic expression. Prove through well-founded induction that the program always terminates, no matter what the initial state is and what b, a are. (**Hint**: Use lexicographic ordering)