Foundations of Programming Languages

BASICS Lab

Shanghai Jiao Tong University

Spring, 2024

Course Materials

The Textbook

Glynn Winskel
The Formal Semantics of Programming Languages: An Introduction

The MIT Press, 1993

What is this course about?

Programming Languages

- imperative languages: C, C++, Java, Python, ...
- functional languages: Haskell, ML, OCaml, ...
- assembly languages: MASM, NASM, ...

But: Warning!

- This course is not about how to write programs!!!

Course Focus

Program Semantics

- programs as mathematical objects
- logical characterizations of programs
- mathematical properties for programs

Course Focus

Why study programs mathematically?

- fundamental components
- rising complexity
- main factor in system performance
- main reason for system failure

Potential Hazard of Program Error

- malfunction of codes
- crash of systems
- attack from hackers
- ...

Course Focus

Program Verification

Formal approaches for analysing programs:

- rigorous proof for absence of bugs
- rigorous proof for absence of vulnerabilities
- full enumeration of race situations
- timing analysis of programs
- . . .

Course Focus

Testing Approaches

- easy to conduct
- can detect normal bugs
- less coverage over codes
- more tendency to neglect critical bugs
- inapplicable in certain situations (e.g., concurrency)

Stuxnet

Stuxnet Computer Worm:

- targets Programmable Logic Controllers;
- hinders automation of electromechanical processes;
- damaged Iran's nuclear plants.

WannaCry

WannaCry Ransomware:

- utilizes a flaw in the Microsoft SMB protocol;
- enforces encryption of data and demands ransom;
- affected computers worldwide.

Timsort

Timsort Implementation Bug:

- is introduced by optimization on merge sort;
- is widely used;
- causes software crash;
- happens rarely.

Retrospect

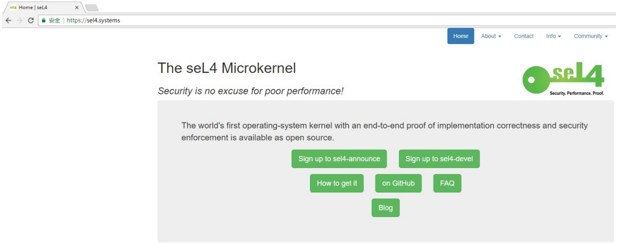
- Subtle critical bugs are hard to detect through testing.
- Subtle critical bugs can be devastating if they are triggered.
- Subtle critical bugs are vulnerable against adversaries.

Successful Stories

- functionally-correct operating system: SEL4
- hacker-free operating system: CertiKOS
- error-free compiler: CompCert, L2C
- race-free concurrency: Astrée
- ...

Successful Stories

SEL4 Operating System



The seL4 Microkernel

Security is no excuse for poor performance!

The world's first operating-system kernel with an end-to-end proof of implementation correctness and security enforcement is available as open source.

source: from internet

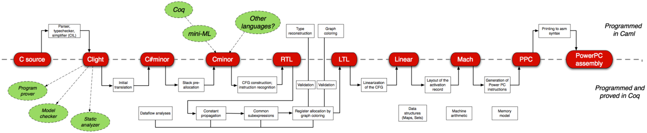
Successful Stories

CertiKOS Operating System

source: from internet

Successful Stories

CompCert Compiler

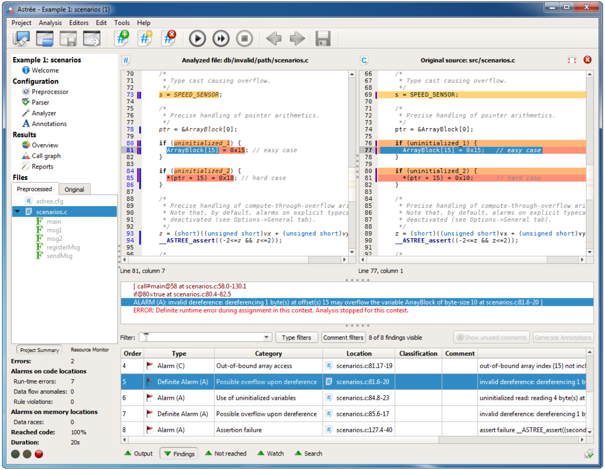


Coq
mini-ML
Other languages?
Parser, typechecker, simplifier (CIL)
Type reconstruction
Graph coloring
Printing to asm syntax
Programmed in Caml
C source
Clight
C#minor
Cminor
RTL
LTL
Linear
Mach
PPC
PowerPC assembly
Program prover
Model checker
Static analyzer
Initial translation
Stack pre-allocation
CFG construction; instruction recognition
Validation
Validation
Linearization of the CFG
Layout of the activation record
Generation of Power PC instructions
Programmed and proved in Coq
Dataflow analyses
Constant propagation
Common subexpressions
Register allocation by graph coloring
Data structures (Maps, Sets)
Machine arithmetic
Memory model

source: from internet

Successful Stories

Astrée Static Analyzer



Analyzed file: db/invalid/path/scenarios.c

```
    /*
     * Type cast causing overflow.
     */
    s = SPEED_SENSOR;

    /*
     * Precise handling of pointer arithmetics.
     */
    ptr = &ArrayBlock[0];

    if (uninitialized_1) {
      ArrayBlock[15] = 0x15; // easy case
    }

    if (uninitialized_2) {
      *(ptr + 15) = 0x10; // hard case
    }

    /*
     * Precise handling of compute-through-overflow ari
     * Note that, by default, alarms on explicit typeca
     * deactivated (see Options->General tab).
     */
    z = (short)((unsigned short)vx + (unsigned short)vy
    __ASTREE_assert((-2<=z && z<=2));

    /*
```

Line 81, column 7

Original source: src/scenarios.c

```
    /*
     * Type cast causing overflow.
     */
    s = SPEED_SENSOR;

    /*
     * Precise handling of pointer arithmetics.
     */
    ptr = &ArrayBlock[0];

    if (uninitialized_1) {
      ArrayBlock[15] = 0x15;   // easy case
    }

    if (uninitialized_2) {
      *(ptr + 15) = 0x10;      // hard case
    }

    /*
     * Precise handling of compute-through-overflow ari
     * Note that, by default, alarms on explicit typeca
     * deactivated (see Options->General tab).
     */
    z = (short)((unsigned short)vx + (unsigned short)vy
    __ASTREE_assert((-2<=z && z<=2));

    /*
```

Line 77, column 1

```
[ call#main@58 at scenarios.c:58.0-130.1
if@80=true at scenarios.c:80.4-82.5
ALARM (A): invalid dereference: dereferencing 1 byte(s) at offset(s) 15 may overflow the variable ArrayBlock of byte-size 10 at scenarios.c:81.6-20 ]
ERROR: Definite runtime error during assignment in this context. Analysis stopped for this context.
```

Filter: 8 of 8 findings visible

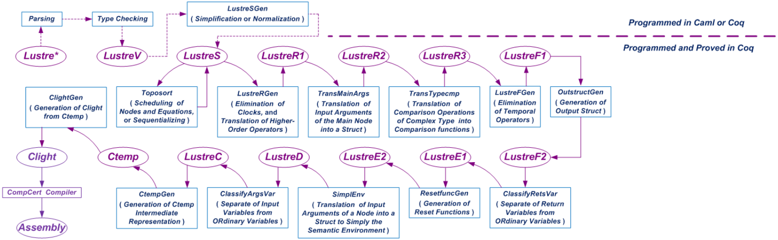
Order	Type	Category	Location	Classification	Comment	
4	Alarm (C)	Out-of-bound array access	scenarios.c:81.17-19			out-of-bound array index {15} not incl
5	Definite Alarm (A)	Possible overflow upon dereference	scenarios.c:81.6-20			invalid dereference: dereferencing 1 b
6	Alarm (A)	Use of uninitialized variables	scenarios.c:84.8-23			uninitialized read: reading 4 byte(s) at
7	Definite Alarm (A)	Possible overflow upon dereference	scenarios.c:85.6-17			invalid dereference: dereferencing 1 b
8	Alarm (A)	Assertion failure	scenarios.c:127.4-40			assert failure __ASTREE_assert((second

Errors:	2
Alarms on code locations	
Run-time errors:	7
Data flow anomalies:	0
Rule violations:	0
Alarms on memory locations	
Data races:	0
Reached code:	100%
Duration:	20s

source: from internet

Successful Stories

L2C: A Formally Verified Compiler From Lustre To C



Parsing

Type Checking

LustreSGen
(Simplification or Normalization)

Programmed in Caml or Coq

Programmed and Proved in Coq

Lustre*

LustreV

LustreS

LustreR1

LustreR2

LustreR3

LustreF1

ClightGen
(Generation of Clight from Ctemp)

Toposort
(Scheduling of Nodes and Equations, or Sequentializing)

LustreRGen
(Elimination of Clocks, and Translation of Higher-Order Operators)

TransMainArgs
(Translation of Input Arguments of the Main Node into a Struct)

TransTypecmp
(Translation of Comparison Operations of Complex Type into Comparison functions)

LustreFGen
(Elimination of Temporal Operators)

OutstructGen
(Generation of Output Struct)

Clight

Ctemp

LustreC

LustreD

LustreE2

LustreE1

LustreF2

CompCert Compiler

CtempGen
(Generation of Ctemp Intermediate Representation)

ClassifyArgsVar
(Separate of Input Variables from ORdinary Variables)

SimplEnv
(Translation of Input Arguments of a Node into a Struct to Simply the Semantic Environment)

ResetfuncGen
(Generation of Reset Functions)

ClassifyRetsVar
(Separate of Return Variables from ORdinary Variables)

Assembly

source: from internet

Successful Stories

Amazon Web Services

source: from internet

Successful Stories

Huawei Harmony OS

source: from internet

Perspective

Program Verification

- is difficult, but not infeasible.
- has much better guarantee than testing.
- is necessary in critical systems.

In Our Course:

Program semantics provides a solid theoretical foundation for program verification.

What will the course cover?

Course Content

Program Semantics

- operational semantics: how do programs execute ?
- denotational semantics: what do programs output ?
- axiomatic semantics: which requirements do programs meet ?

Course Content

Types of Programs

- (mostly) imperative programs
- (some) functional programs

Technical Content

- logical definitions for programs
- mathematical background behind program semantics

Course Content

- Chapter 1: set theory
- Chapter 2,3,4: operational semantics
- Chapter 5: denotational semantics
- Chapter 6,7: axiomatic semantics
- Chapter 8: domain theory
- Chapter 11: typed languages

An Example

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

Another Example

```
while ($X \le 100$) do
    if ($X \le 0$)
        then $X := X - 1$
        else $X := X + 2$
```

What can we gain from this course?

Course Benefits

- a rigorous thinking of programs
- a comprehensive start to program analysis and verification

Chapter 1

Basic set theory

Topics

An Informal Introduction on Set Theory

- What are sets?
- How can one reason about sets?
- How can one construct sets?
- How relations and functions are defined in set theory?

Textbook Content

- Chapter 1: Basic Set Theory

Set Theory: An Intuitive Description

Why Set Theory?

- a rigorous language for a logical world
- a solid foundation for mathematics
- a solid foundation for programming languages

Why Set Theory?

- Reasoning without a solid foundation is error-prone.
- Reasoning with a solid foundation is precise.

Set Theory

What is a Set?

A set is a collection of objects that acts as a single entity.

Set Theory: An Overview

- an abstract world: a world of sets as objects

Set Theory: An Overview

- set reasoning: for any object a and set X, either $a \in X$ or $a \notin X$ but not both.

- set construction: any set can be constructed from the empty set through a finite number of axioms

Sets: Examples

- $\{a\}$, $\{a, b\}$, $\{a_1, \dots, a_n\}$
- $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$, $\mathbb{C}$
- $\mathbb{R} \times \mathbb{R}$
- $[a, b] = \{x \in \mathbb{R} \mid a \leq x \;\&\; x \leq b\}$
- $\{p \in \mathbb{N} \mid \forall q, r \in \mathbb{N}. [p = q \cdot r \Rightarrow (q = 1 \vee r = 1)]\}$

The Language of Set Theory

The Language of Set Theory

- a formal language for sets
- a formal language for mathematical objects, i.e., numbers, functions, graphs, . . .
  (as they can be defined as sets in set theory)

The Language of Set Theory

- names: $a_0, a_1, \ldots, A_0, A_1, \ldots$
- variables: $x_0, x_1, \ldots$
- logic connectives: $\neg$, & , or , $\Rightarrow$, $\Leftrightarrow$, $\exists$, $\forall$

The Language of Set Theory

Formulas (Clauses, Sentences)

- atomic formulas: $x = y$, $x \in y$ (x, y are names or variables);
- boolean connectives:
  - if ϕ is a formula, then so is $\neg\phi$;
  - if ϕ_1, ϕ_2 are formulas, then so too are

$$\phi_1 \mathbin{\&} \phi_2 \ , \ \phi_1 \text{ or } \phi_2 \ , \ \phi_1 \Rightarrow \phi_2 \ , \ \phi_1 \Leftrightarrow \phi_2 \ ;$$

- quantifiers: if ϕ is a formula and x is a variable, then $\forall x.\phi$, $\exists x.\phi$ are formulas.

Atomic Formulae

- (equality) $x = y$
  (meaning the assertion "x, y name the same object (set)")
- (membership) $x \in y$
  (meaning the assertion "x is an element of y")
- evaluated to
  - either **true** (i.e. the formula holds),
  - or **false** (i.e., the formula does not hold)

  when the meaning of x, y (i.e., which sets x, y name) is clear

Boolean Connectives

- $\neg$: negation (“not”)
- &: conjunction (“and”)
- or: disjunction (“or”)
- $\Rightarrow$: implication (“imply”)
- $\Leftrightarrow$: double implication (“iff”)

Truth Table

Negation ("not")

ϕ	$\neg\phi$
true	**false**
false	**true**

Truth Table

Conjunction (“and”)

ϕ_1	ϕ_2	ϕ_1 & ϕ_2
true	**true**	**true**
true	**false**	**false**
false	**true**	**false**
false	**false**	**false**

Truth Table

Disjunction ("or")

ϕ_1	ϕ_2	ϕ_1 or ϕ_2
true	**true**	**true**
true	**false**	**true**
false	**true**	**true**
false	**false**	**false**

Truth Table

Implication (“imply”)

ϕ_1	ϕ_2	$\phi_1 \Rightarrow \phi_2$
true	**true**	**true**
true	**false**	**false**
false	**true**	**true**
false	**false**	**true**

Truth Table

Double Implication (“iff”)

ϕ_1	ϕ_2	$\phi_1 \Leftrightarrow \phi_2$
true	**true**	**true**
true	**false**	**false**
false	**true**	**false**
false	**false**	**true**

Boolean Expressibility

Exercise

Prove (through truth table) that the following formulas are logically equivalent:

- ϕ_1 or ϕ_2 and $\neg(\neg\phi_1 \,\&\, \neg\phi_2)$;
- $\phi_1 \Rightarrow \phi_2$ and $(\neg\phi_1)$ or ϕ_2;
- $\phi_1 \Leftrightarrow \phi_2$ and $(\phi_1 \,\&\, \phi_2)$ or $(\neg\phi_1 \,\&\, \neg\phi_2)$.
- $\phi_1 \Leftrightarrow \phi_2$ and $(\phi_1 \Rightarrow \phi_2) \,\&\, (\phi_2 \Rightarrow \phi_1)$

Quantifiers

Universal Quantification

$\forall x.\phi(x)$ holds (or simply written as $\forall x.\phi(x)$) if

- (intuition) for any set x, $\phi(x)$ holds;
- (meaning) $\phi(x)$ is true (i.e., holds) no matter what x names in the universe of all sets;

Example

- A, B : sets
- A equals B: $\forall x.\,(x \in A \Leftrightarrow x \in B)$

Quantifiers

Existential Quantification

$\exists x.\phi(x)$ holds (or simply written as $\exists x.\phi(x)$) if

- (intuition) there exists a set x such that $\phi(x)$ holds;
- (meaning) there exists a set such that $\phi(x)$ is true (i.e., holds) when x names that set.

Quantifiers

- no clear thinking (e.g., no truth table)
- $\forall x.\phi$ is logically equivalent to $\neg(\exists x.\neg\phi)$.

Exercise

Prove that $\forall x.\phi$ is logically equivalent to $\neg(\exists x.\neg\phi)$.

Quantifiers

Conditioned Quantifiers

- A: a set
- $\forall x \in A.\phi(x) \ := \ \forall x.\,(x \in A \Rightarrow \phi(x))$
- $\exists x \in A.\phi(x) \ := \ \exists x \in A.\,(x \in A \ \& \ \phi(x))$

Exercise

Prove that $\forall x \in A.\phi$ is logically equivalent to $\neg\,(\exists x \in A.\neg\phi)$.

Axioms

- formulas assumed to be correct
- formulas for asserting properties of sets
- formulas for constructing sets

Axioms for Set Reasoning

Set Reasoning

Extensionality Axiom

- statement: $\forall A \forall B. [\forall x.(x \in A \Leftrightarrow x \in B) \Rightarrow A = B]$
- meaning: if two sets A, B have exactly the same members, then they are equal.

Set Reasoning

Set Inclusion

- definition: Given any two sets A, B, we write

$$A \subseteq B \text{ if } \forall x.(x \in A \Rightarrow x \in B).$$

- property: For any two sets A, B, $A = B$ iff $A \subseteq B$ and $B \subseteq A$.

Exercise

Prove from Extensionality Axiom that $A = B$ iff $A \subseteq B$ and $B \subseteq A$.

Set Reasoning

The Axiom of Foundation (Regularity)

- statement: $\forall A. [A \neq \emptyset \Rightarrow \exists B. (B \in A \; \& \; B \cap A = \emptyset)]$.
- corollary: for any set A, $A \notin A$.
- corollary: there is no infinite set sequence $A_0, A_1, \cdots$ such that $\cdots \in A_{n+1} \in A_n \in \cdots \in A_1 \in A_0$.

Axioms for Set Construction

Set Construction

Empty Set Axiom

- statement: $\exists B.\,(\forall x.x \notin B)$;
- uniqueness: $\forall A \forall B.\,[(\forall x.x \notin A)\ \&\ (\forall x.x \notin B) \Rightarrow A = B]$
- notation: $\emptyset$ is the set without any member.

Question

Why do we need uniqueness?

Set Construction

Pairing Axiom

- statement: $\forall u.\forall v.\exists B.\,[\forall x.\,(x \in B \Leftrightarrow x = u \text{ or } x = v)]$;
- uniqueness: from Extensionality Axiom
- notation: $B = \{u, v\}$ ($\{u\} := \{u, u\}$).

Set Construction

Pairing Axiom

- ordered pairs: $(x, y) := \{\{x\}, \{x, y\}\}$;
- property: $(x_1, y_1) = (x_2, y_2)$ iff $x_1 = x_2$ and $y_1 = y_2$.

Set Construction

Union Axiom (Preliminary Version)

- statement: $\forall A.\forall B.\exists C.\,[\forall x.\,(x \in C \Leftrightarrow x \in A \text{ or } x \in B)]$;
- uniqueness: from Extensionality Axiom
- notation: $C = A \cup B$

Set Construction

Union Axiom

- statement: $\forall \mathcal{A}. \exists B. [\forall x. (x \in B \Leftrightarrow \exists A \in \mathcal{A}. x \in A)]$;
- uniqueness: from Extensionality Axiom
- notation: $B = \bigcup \mathcal{A}$
- example: $A \cup B = \bigcup \{A, B\}$

Set Construction

Power Set Axiom

- statement: $\forall A.\exists B.[\forall x.x \in B \Leftrightarrow x \subseteq A]$;
- uniqueness: from Extensionality Axiom
- notation: $B = 2^A$, $B = Pow(A)$ (textbook) or informally $B = \{Y \mid Y \subseteq A\}$

Example

$2^{\{0,1\}} = \{\emptyset, \{0\}, \{1\}, \{0,1\}\}$.

Set Construction

Subset Axiom

- statement: for any set A, for any sets $t_1, \ldots, t_n$, for any formula $\phi(x, y_1, \ldots, y_n)$, there exists a set B such that

$$\forall x. (x \in B \Leftrightarrow x \in A \,\&\, \phi(x, t_1, \ldots, t_n)) \;\;;$$

- uniqueness: from Extensionality Axiom
- notation: $B = \{x \in A \mid \phi(x, t_1, \ldots, t_n)\}$

Set Construction

Some Set Operations

- intersection: $A \cap B := \{x \in A \cup B \mid x \in A \mathbin{\&} x \in B\}$;
- set difference: $A \setminus B := \{x \in A \cup B \mid x \in A \mathbin{\&} x \notin B\}$;
- general intersection: if $\mathcal{A} \neq \emptyset$,

$$\bigcap \mathcal{A} := \{x \in \bigcup \mathcal{A} \mid \forall B.\,(B \in \mathcal{A} \Rightarrow x \in B)\};$$

Set Construction

Cartesian Product

- A, B: sets
- informal definition: $A \times B := \{(x, y) \mid x \in A \ \& \ y \in B\}$;
- formal definition:

$$A \times B := \{w \in 2^{2^{A \cup B}} \mid \exists x.\exists y.\,(w = (x, y) \ \& \ x \in A \ \& \ y \in B)\};$$

Set Construction

Disjoint Union

- A, B: sets
- $A \uplus B := (\{0\} \times A) \cup (\{1\} \times B)$

Set Construction

Why subset axiom requires a super set?

- Russell's Paradox: $X := \{x \mid x \notin x\}$;
- the paradox:
  - $X \in X \Rightarrow X \notin X$
  - $X \notin X \Rightarrow X \in X$
- explanation: $\{x \mid x \notin x\}$ conceptually exists, but is not a set.

Natural Numbers

- $0 := \emptyset$;
- $1 := \emptyset \cup \{\emptyset\}$;
- $n^+ := n \cup \{n\}$ for any natural number n;

Natural Numbers

Inductive Sets

A set A is inductive if $\emptyset \in A$ and for any $a \in A$, $a^+ := a \cup \{a\} \in A$.

Infinity Axiom

There exists an inductive set: $\exists A.\,[\emptyset \in A \;\&\; (\forall a.a \in A \Rightarrow a^+ \in A)]$.

Natural Numbers

Definition

- A : an inductive set;
- natural numbers:

$$\omega := \mathbb{N} := \{n \in A \mid \forall B. (B \text{ is inductive} \Rightarrow n \in B)\}$$

- $\omega = \mathbb{N} = \{0, 1, 2, \dots\}$

Numbers

- the set of integers
- the set of rational numbers
- the set of real numbers
- the set of complex numbers

The Overview of Set Theory

Relations and Functions

Relations

Definition

- x, y: sets (objects)
- ordered pairs: $(x, y) := \{\{x\}, \{x, y\}\}$;

A relation $\mathcal{R}$ is a set of ordered pairs.

Intuition

$(x, y) \in \mathcal{R}$ means x, y are related by $\mathcal{R}$ in order.

Notation

$x\mathcal{R}y : (x, y) \in \mathcal{R}$

Examples

- $\mathcal{R} = \emptyset$;
- $\mathcal{R} = \{(0,1),(0,2),(2,1),(1,2),(4,1)\}$;
- $\mathcal{R} = \mathbb{N} \times \mathbb{N}$;
- $\mathcal{R} = \{(n,m) \in \mathbb{N} \times \mathbb{N} \mid m = 2 \cdot n\}$;
- $\mathcal{R} = \{(x,y) \in \mathbb{R} \times \mathbb{R} \mid x^2 + y^2 = 1\}$;

Binary Relations

Definition

- X, Y: sets

A binary relation $\mathcal{R}$ between X and Y is a subset $\mathcal{R} \subseteq X \times Y$ of $X \times Y$.

Binary Relations

Images

- X, Y: sets
- $\mathcal{R} \subseteq X \times Y$: a binary relation
- direct image: for any $A \subseteq X$,

$$\mathcal{R}(A) := \{y \in Y \mid \exists x \in A. x\mathcal{R}y\}$$

- inverse image: for any $B \subseteq Y$,

$$\mathcal{R}^{-1}(B) := \{x \in X \mid \exists y \in B. x\mathcal{R}y\}$$

Binary Relations

Examples

- $\mathcal{R} = \{(x, y) \in \mathbb{R} \times \mathbb{R} \mid x^2 + y^2 = 1\}$
    - $\mathcal{R}(\{-1\}) = \{0\}$
    - $\mathcal{R}^{-1}(\{0\}) = \{-1, 1\}$
- $\mathcal{R} = \{(x, y) \in \mathbb{R} \times \mathbb{R} \mid x \leq y\}$
    - $\mathcal{R}(\{1\}) = [1, \infty)$
    - $\mathcal{R}^{-1}(\{1\}) = (-\infty, 1]$

Composition

Definition

- $\mathcal{R}$: a binary relation between X and Y
- $\mathcal{S}$: a binary relation between Y and Z

$\mathcal{S} \circ \mathcal{R}$ is the binary relation between X and Z

$$\mathcal{S} \circ \mathcal{R} := \{(x, z) \in X \times Z \mid \exists y. (x\mathcal{R}y \;\&\; y\mathcal{S}z)\}$$

Composition

- $\mathcal{R} \subseteq X \times X$

Repeated Composition

- $\mathcal{R}^0 := \mathrm{Id}_{\mathcal{R}} := \{(x, y) \in X \times X \mid x = y\}$;
- $\mathcal{R}^{n+1} := \mathcal{R} \circ \mathcal{R}^n$

Closures

- $\mathcal{R}^+ := \bigcup_n \mathcal{R}^{n+1}$;
- $\mathcal{R}^* := \bigcup_n \mathcal{R}^n$;

Composition

Closures

- transitive closure: $\mathcal{R}^{+} := \bigcup_n \mathcal{R}^{n+1}$;
- reflexive transitive closure: $\mathcal{R}^{*} := \bigcup_n \mathcal{R}^{n}$;

Properties

- transitivity: for any $x, y, z \in X$,

$$x\mathcal{R}^{+}y \ \& \ y\mathcal{R}^{+}z \Rightarrow x\mathcal{R}^{+}z$$

- $\mathcal{R}^{*}$ is in addition reflexive: for any $x \in X$, $x\mathcal{R}^{*}x$;

Equivalence Relations

Definition

- X: a set
- $\mathcal{R} \subseteq X \times X$: a binary relation

$\mathcal{R}$ is an equivalence relation on X if:

- reflexibility: for any $x \in X$, $x\mathcal{R}x$;
- symmetry: for any $x, y \in X$, $x\mathcal{R}y \Leftrightarrow y\mathcal{R}x$;
- transitivity: for any $x, y, z \in X$, $x\mathcal{R}y \;\&\; y\mathcal{R}z \Rightarrow x\mathcal{R}z$.

Equivalence Relations

Examples

- $\{(x, y) \in X \times X \mid x = y\}$;
- $\{(x, y) \in X \times X \mid f(x) = f(y)\}$ (f is a function on X);
- $\{(m, n) \in \mathbb{N} \times \mathbb{N} \mid 7|m - n\}$;

Equivalence Relations

Definition

- X: a set
- $\mathcal{R} \subseteq X \times X$: an equivalence relation on X

For any $a \in X$, define the equivalence class of a by

$$[a]_{\mathcal{R}} := \{x \in X \mid x\mathcal{R}a\}$$

Equivalence Relations

Examples

- $\mathcal{R} = \{(x, y) \in X \times X \mid x = y\}$, $[a]_{\mathcal{R}} = \{a\}$;
- $\{(x, y) \in X \times X \mid f(x) = f(y)\}$, $[a]_{\mathcal{R}} = \{x \mid f(x) = f(a)\}$;
- $\{(m, n) \in \mathbb{N} \times \mathbb{N} \mid 7 | m - n\}$, $[6]_{\mathcal{R}} = \{n \mid \exists k \in \mathbb{N}. n = 7 \cdot k + 6\}$;

Partial Orders

Definition

- X: a set
- $\mathcal{R} \subseteq X \times X$: a binary relation

$\mathcal{R}$ is a partial order on X if:

- reflexibility: for any $x \in X$, $x\mathcal{R}x$;
- antisymmetry: for any $x, y \in X$, $x\mathcal{R}y \ \& \ y\mathcal{R}x \Rightarrow x = y$;
- transitivity: for any $x, y, z \in X$, $x\mathcal{R}y \ \& \ y\mathcal{R}z \Rightarrow x\mathcal{R}z$.

Partial Orders

Examples

- $\{(x,y) \in X \times X \mid x = y\}$;
- $\leq$ on $\mathbb{N}, \mathbb{Q}, \mathbb{R}$;
- $\{(m,n) \in \mathbb{N} \times \mathbb{N} \mid m,n \geq 1, m|n\}$;

Functions

- informal vs. set-theoretic definitions

Functions

Intuition

- X, Y: sets

A function from X to Y is a mapping that assigns to each element in X a unique element in Y.

Functions

Intuition

- A single map is of the form $a \mapsto b$ ($a \in X$, $b \in Y$).
- A function is a collection of such maps.
- It will never happen that there exist two maps $a \mapsto b, a \mapsto c$ such that $b \neq c$.

Functions

Characterization

- a single map $a \mapsto b$: an ordered pair $(a, b) \in X \times Y$
- a collection of maps: a binary relation $F \subseteq X \times Y$
- no $a \mapsto b, a \mapsto c$ satisfying $b \neq c$:

$$\text{for any } (a, b), (a, c) \in F, \text{ we have } b = c$$

Example

- $F(x) = x^2,\ x \in \mathbb{R}$
- $F = \{(x, y) \in \mathbb{R} \times \mathbb{R} \mid y = x^2\}$

Functions

Set-Theoretic Definition

- X, Y: sets

A partial function F from X to Y is a binary relation $F \subseteq X \times Y$ such that

$$\forall x \in X . \forall y, y' \in Y . [xFy \ \& \ xFy' \Rightarrow y = y']$$

Notation

- A partial function F from X to Y is stressed by $F : X \rightharpoonup Y$.
- $F(x)$ is define as the unique y such that xFy if such y exists.

Functions

Set-Theoretic Definition

- X, Y: sets

A (total) function F from X to Y is a partial function from X to Y such that for any $x \in X$ there exists $y \in Y$ such that xFy.

Notation

- For each $x \in X$, $F(x)$ is the unique element such that $(x, F(x)) \in F$.
- A function F from X to Y is stressed by $F : X \to Y$.

Functions

- range: $F(X) = \{y \in Y \mid \exists x . y = F(x)\}$
- domain: $F^{-1}(Y) = \{x \in X \mid \exists y . y = F(x)\} = X$

Functions

Examples

- $F : \mathbb{R} \rightharpoonup \mathbb{R}$, $F(x) = \frac{1}{x}$:

$$F = \{(x, y) \in \mathbb{R} \times \mathbb{R} \mid x \cdot y = 1\} \ ;$$

- $F : \mathbb{R} \to \mathbb{R}$, $F(x) = \sin x$:

$$F = \{(x, y) \in \mathbb{R} \times \mathbb{R} \mid y = \sin x\};$$

- $F : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$, $F(x, y) = x + y$:

$$F = \{((x, y), z) \in (\mathbb{R} \times \mathbb{R}) \times \mathbb{R} \mid z = x + y\};$$

Functions

λ-Notation

- X, Y: sets
- $f : X \to Y$: a function
- e: an expression representing f (e.g., $e = x + 1$ and $f(x) = x + 1$)

Then we denote also by $\lambda x \in X.e$ the function f.

Example

- $\lambda x \in \omega.(x + 1)$: the function $f(x) = x + 1$
- $\lambda x \in \mathbb{R}. \sin x$: the function $f(x) = \sin x$

Composition (Recall)

Definition

- $\mathcal{R}$: a binary relation between X and Y
- $\mathcal{S}$: a binary relation between Y and Z

$\mathcal{S} \circ \mathcal{R}$ is the binary relation between X and Z defined by

$$\mathcal{S} \circ \mathcal{R} := \{(x, z) \in X \times Z \mid \exists y.\,(x\mathcal{R}y \;\&\; y\mathcal{S}z)\}$$

Composition (Recall)

Definition

- $\mathcal{R}$: a binary relation between X and Y
- $\mathcal{S}$: a binary relation between Y and Z

$\mathcal{S} \circ \mathcal{R}$ is the binary relation between X and Z defined by

$$\mathcal{S} \circ \mathcal{R} := \{(x, z) \in X \times Z \mid \exists y. (x\mathcal{R}y \ \& \ y\mathcal{S}z)\}$$

Example

- $F : X \to Y$
- $G : Y \to Z$

$G \circ F : X \to Z$ satisfies that $(G \circ F)(x) = G(F(x))$

Cantor's Diagonal Argument

Inverse

- X, Y: sets

A function $F : X \to Y$ has an inverse $G : Y \to X$ if

- $G(F(x)) = x$ for all $x \in X$, and
- $F(G(y)) = y$ for all $y \in Y$.

If there exists a function $F : X \to Y$ with its inverse $G : Y \to X$, then X, Y are in 1-1 correspondence.

Cantor's Diagonal Argument

Theorem

- X: a set

X and 2^X are not in 1-1 correspondence.

Cantor's Diagonal Argument

Theorem

X and 2^X are not in 1-1 correspondence.

Proof (by Contradiction)

$\theta : X \to 2^X$ with an inverse

	$\theta(x_0)$	$\theta(x_1)$	$\theta(x_2)$	$\ldots$	$\theta(x_j)$	$\ldots$
x_0	0	1	1	$\ldots$	1	$\ldots$
x_1	1	1	1	$\ldots$	0	$\ldots$
x_2	0	0	1	$\ldots$	0	$\ldots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$		$\vdots$	
x_i	0	1	0	$\ldots$	1	$\ldots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$		$\vdots$	

Cantor's Diagonal Argument

Theorem

X and 2^X are not in 1-1 correspondence.

Proof

$\theta : X \to 2^X$ with an inverse

$$\begin{array}{c|ccc}
 & \ldots & Y = \theta(y) & \ldots \\
\hline
\ldots & \ldots & \ldots & \ldots \\
y & \ldots & y \in \theta(y)? & \ldots \\
\ldots & \ldots & \ldots & \ldots
\end{array}$$

Cantor's Diagonal Argument

Theorem

X and 2^X are not in 1-1 correspondence.

Proof

- Suppose that there is a function $\theta : X \to 2^X$ with an inverse.
- Define the set

$$Y := \{x \in X \mid x \notin \theta(x)\} \in 2^X \quad .$$

- Let y be the unique element in X such that $\theta(y) = Y$.
- $y \in Y \Rightarrow y \notin \theta(y)(= Y)$
- $y \notin Y \Rightarrow y \in \theta(y)(= Y)$

Summary

Basic Set Theory

- a deeper understanding of sets
- axioms for set reasoning and construction
- set-theoretic definitions for relations and functions
- rigorous reasoning with relations and functions

Introduction to operational semantics

Topic

Operational Semantics

- a simple imperative language as a minimal language
- a set of rules as building blocks for the semantics
- rule-based derivations as the operational semantics

Topic

After the lecture, we will be able to . . .

- know the logical background of operational semantics.
- know the necessary ingredients to construct operational semantics.

A Simple Imperative Language **IMP**

Textbook, Page 11 – Page 13

A Simple Imperative Language **IMP**

- data type: integers **N**
  (e.g., $0, 1, 2, \ldots, -1, -2, \ldots$)
- truth value: boolean values $\mathbf{T} = \{\mathbf{true}, \mathbf{false}\}$
- locations: **Loc** (identifiers or program variables)
  (e.g., x, y, i, j, a, b, $flag, \ldots$)
- arithmetic expressions: **Aexp**
  (e.g., $x + y$, $z - 3$, $x \times y$, $\ldots$)
- boolean expressions: **Bexp**
  (e.g., $(x > 0) \wedge (y < 0)$, $(x > 0) \vee (y < 0)$, $\neg(x > y)$, $\ldots$)
- commands: statements **Com**
  (e.g., assignment, if branch, while loop, ...)

Program Syntax

Arithmetic Expressions **Aexp**

Arithmetic expressions are built from

- integers,
- locations (identifiers),
- arithmetic operations including $+, -, \times$.

The syntax:

$$a ::= \; n \;\mid\; X \;\mid\; a_0 + a_1 \;\mid\; a_0 - a_1 \;\mid\; a_0 \times a_1$$

where n is any integer and X is any location.

Program Syntax

Boolean Expressions **Bexp**

Boolean expressions are built from

- truth values: **true**, **false**
- comparison: $=, \leq, \geq, <, >$
- propositional logical connectives: $\neg, \wedge, \vee$

The syntax

$$b ::= \textbf{true} \mid \textbf{false} \mid a \bowtie a' \mid \neg b \mid b \wedge b' \mid b \vee b'$$

where $\bowtie \in \{=, \leq, \geq, <, >\}$ and a, a' are arithmetic expressions.

Program Syntax

Commands **Com**

- assignment statements
- sequential composition
- if branches
- while loops

Program Syntax

Commands **Com**

The syntax of commands:

$$
\begin{aligned}
c ::= {} & \textbf{skip} \\
& \mid X := a \\
& \mid c_0;\ c_1 \\
& \mid \textbf{if } b \textbf{ then } c_0 \textbf{ else } c_1 \\
& \mid \textbf{while } b \textbf{ do } c'
\end{aligned}
$$

Program Syntax

A Variant of the Euclidean Algorithm

```
while $\neg(M = N)$ do
  if $M \leq N$
    then $N := N - M$
    else $M := M - N$
```

Program Syntax

IMP allows

- integer type,
- assignment,
- sequential composition,
- conditional branch,
- while loop.

Program Syntax

IMP does not allow

- data structures,
- floating numbers,
- recursion,
- pointers,
- . . .

The Operational Semantics of **IMP**

Textbook, Page 15 – 20

Operational Semantics

Overview

- rules for arithmetic/boolean expressions
- rules for commands (statements)
- derivations for the final operational semantics

Operational Semantics

States

- A state is a function $\sigma : \mathbf{Loc} \to \mathbf{N}$.
- The set of states is denote by Σ.

Intuition

A state specifies values held by locations.

Operational Semantics

Our Goal

A relation $\mathcal{R} \subseteq (\textbf{Com} \times \Sigma) \times \Sigma$ such that $(c, \sigma)\, \mathcal{R}\, \sigma'$ iff

> *when executing c with initial state σ, c terminates and we eventually get σ' after the execution.*

We often write $(c, \sigma) \to \sigma'$ instead of $(c, \sigma)\, \mathcal{R}\, \sigma'$.

Operational Semantics

Question

How can we construct such a relation?

The Methodology

- from rules to derivations
- from arithmetic expressions to commands

Arithmetic Expressions

Configurations (for **Aexp**)

A configuration is a pair $\langle a, \sigma \rangle$ where $a \in$ **Aexp** and $\sigma \in \Sigma$.

Sub-goal

a relation for $\langle a, \sigma \rangle \to n$:

> *an arithmetic expression a is evaluated to an integer n when locations in a are substituted by their values from σ.*

Arithmetic Expressions

Question

How can we define "$\langle a, \sigma \rangle \to n$" rigorously?

Principles

- The definition should be syntactical.
- The definition should be correct.

Arithmetic Expressions

The Intuition

How can we evaluate $a_0 + a_1$ under a state σ ?

- first evaluate a_0, a_1 correctly: $\langle a_0, \sigma \rangle \to n_0$, $\langle a_1, \sigma \rangle \to n_1$
- then evaluate $a_0 + a_1$ correctly: $\langle a_0 + a_1, \sigma \rangle \to n_0 + n_1$

Implementation: rules and derivations!

Arithmetic Expressions

The Rule for Addition

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

- if $\langle a_0, \sigma\rangle \to n_0$ and $\langle a_1, \sigma\rangle \to n_1$, then $\langle a_0 + a_1, \sigma\rangle \to n_0 + n_1$;
- premise: $\langle a_0, \sigma\rangle \to n_0$ and $\langle a_1, \sigma\rangle \to n_1$;
- conclusion: $\langle a_0 + a_1, \sigma\rangle \to n_0 + n_1$;

Arithmetic Expressions

How to build rules?

- Establish rules for each arithmetic operation.
  (e.g., addition, subtraction, multiplication)
- Prove correctness for each rule.
  (i.e., proving that the premise implies the conclusion)

Arithmetic Expressions

Numbers and Locations

$$\frac{}{\langle n, \sigma \rangle \rightarrow n} \qquad \frac{}{\langle X, \sigma \rangle \rightarrow \sigma(X)}$$

- axioms: rules without premise
- metavariables: n, X, σ

Arithmetic Expressions

Arithmetic Operations

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

$$\frac{\langle a_0, \sigma \rangle \to n_0, \ \langle a_1, \sigma \rangle \to n_1}{\langle a_0 \times a_1, \sigma \rangle \to n_0 \cdot n_1}$$

- $n_0, n_1, a_0, a_1, \sigma$: metavariables

Arithmetic Expressions

Rule Instances

A rule instance is obtained from substituting metavariables by concrete elements.

Examples

$$\frac{}{\langle 5, \sigma \rangle \to 5} \qquad \frac{}{\langle X, \{X \mapsto 4, Y \mapsto 5\} \rangle \to 4}$$

$$\frac{\langle a_0, \sigma \rangle \to 2, \ \langle a_1, \sigma \rangle \to 3}{\langle a_0 \times a_1, \sigma \rangle \to 6}$$ (a_i's are concrete arithmetic expressions)

Arithmetic Expressions

Question

How can we organize rules for compound arithmetic expressions?

Arithmetic Expressions

Derivation Trees

- $\sigma(X) = 1, \sigma(Y) = -1$
- the evaluation of $\langle (X+5)-(Y\times 2), \sigma\rangle$:

$$\dfrac{\dfrac{\overline{\langle X,\sigma\rangle \to 1} \quad \overline{\langle 5,\sigma\rangle \to 5}}{\langle X+5,\sigma\rangle \to 6} \qquad \dfrac{\overline{\langle Y,\sigma\rangle \to -1} \quad \overline{\langle 2,\sigma\rangle \to 2}}{\langle Y\times 2,\sigma\rangle \to -2}}{\langle (X+5)-(Y\times 2),\sigma\rangle \to 8}$$

- conclusion: $\langle (X+5)-(Y\times 2), \sigma\rangle \to 8$

Arithmetic Expressions

Derivation Tree

A derivation tree (derivation) is a finite tree such that every parent-children substructure in the tree is a rule instance.

Definitions

- definition: $\langle a, \sigma \rangle \to n$ iff there is a derivation tree with conclusion $\langle a, \sigma \rangle \to n$.
- property: $\forall a.\forall \sigma.\exists n.\langle a, \sigma \rangle \to n$
- equivalence: $a \sim a'$ iff $\forall n.\forall \sigma.\,(\langle a, \sigma \rangle \to n \Leftrightarrow \langle a', \sigma \rangle \to n)$
- big-step semantics: internal computation is omitted.
- missing rigor: derivation trees

Boolean Expressions

Truth Values

$$\frac{}{\langle \textbf{true}, \sigma \rangle \to \textbf{true}} \qquad \frac{}{\langle \textbf{false}, \sigma \rangle \to \textbf{false}}$$

Boolean Expressions

Comparison

$$\frac{\langle a_0, \sigma \rangle \rightarrow n_0, \;\; \langle a_1, \sigma \rangle \rightarrow n_1}{\langle a_0 = a_1, \sigma \rangle \rightarrow \mathbf{true}} \text{ if } n_0 = n_1$$

$$\frac{\langle a_0, \sigma \rangle \rightarrow n_0, \;\; \langle a_1, \sigma \rangle \rightarrow n_1}{\langle a_0 = a_1, \sigma \rangle \rightarrow \mathbf{false}} \text{ if } n_0 \neq n_1$$

Boolean Expressions

Comparison

$$\frac{\langle a_0, \sigma\rangle \to n_0, \ \ \langle a_1, \sigma\rangle \to n_1}{\langle a_0 \leq a_1, \sigma\rangle \to \textbf{true}} \text{ if } n_0 \leq n_1$$

$$\frac{\langle a_0, \sigma\rangle \to n_0, \ \ \langle a_1, \sigma\rangle \to n_1}{\langle a_0 \leq a_1, \sigma\rangle \to \textbf{false}} \text{ if } n_0 > n_1$$

Boolean Expressions

Negation

$$\frac{\langle b, \sigma \rangle \rightarrow \textbf{true}}{\langle \neg b, \sigma \rangle \rightarrow \textbf{false}} \qquad \frac{\langle b, \sigma \rangle \rightarrow \textbf{false}}{\langle \neg b, \sigma \rangle \rightarrow \textbf{true}}$$

Boolean Expressions

Disjunction and Conjunction

$$\frac{\langle b_0, \sigma \rangle \rightarrow t_0, \ \ \langle b_1, \sigma \rangle \rightarrow t_1}{\langle b_0 \wedge b_1, \sigma \rangle \rightarrow t_0 \wedge t_1}$$

$$\frac{\langle b_0, \sigma \rangle \rightarrow t_0, \ \ \langle b_1, \sigma \rangle \rightarrow t_1}{\langle b_0 \vee b_1, \sigma \rangle \rightarrow t_0 \vee t_1}$$

Boolean Expressions

Definition

- definition: $\langle b, \sigma \rangle \to t$ iff there is a derivation tree with conclusion $\langle b, \sigma \rangle \to t$.
- property: $\forall b. \forall \sigma. \exists t. \langle b, \sigma \rangle \to t$
- equivalence: $b \sim b'$ iff $\forall t \in \{\mathbf{true}, \mathbf{false}\}. \forall \sigma. (\langle b, \sigma \rangle \to t \Leftrightarrow \langle b', \sigma \rangle \to t)$
- big-step semantics: Internal computation is omitted.
- missing rigor: derivation trees

Commands

Skip

$$\frac{}{\langle \textbf{skip}, \sigma \rangle \to \sigma}$$

Commands

Substitution over States

- σ: a state
- m: an integer
- X: a location (program variable)

We define $\sigma[m/X]$ by

$$\sigma[m/X](Y) := \begin{cases} m & \text{if } Y = X \\ \sigma(Y) & \text{otherwise} \end{cases}$$

Commands

Assignment Statements

$$\frac{\langle a, \sigma\rangle \to m}{\langle X := a, \sigma\rangle \to \sigma\,[m/X]}$$

Commands

Sequential Composition

$$\frac{\langle c_0, \sigma\rangle \rightarrow \sigma'', \ \langle c_1, \sigma''\rangle \rightarrow \sigma'}{\langle c_0;\ c_1, \sigma\rangle \rightarrow \sigma'}$$

Commands

Conditional Branches

$$\frac{\langle b, \sigma\rangle \to \textbf{true}, \ \langle c_0, \sigma\rangle \to \sigma'}{\langle \textbf{if } b \textbf{ then } c_0 \textbf{ else } c_1, \sigma\rangle \to \sigma'}$$

$$\frac{\langle b, \sigma\rangle \to \textbf{false}, \ \langle c_1, \sigma\rangle \to \sigma'}{\langle \textbf{if } b \textbf{ then } c_0 \textbf{ else } c_1, \sigma\rangle \to \sigma'}$$

Commands

While Loops

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

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

Commands

Definitions

- 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 $\forall \sigma, \sigma'. (\langle c, \sigma \rangle \to \sigma' \Leftrightarrow \langle c', \sigma \rangle \to \sigma')$
- big-step semantics: Internal computation is omitted.
- missing rigour: derivation trees

Commands

An Example

```
while $\neg(M = N)$ do
  if $M \leq N$
    then $N := N - M$
    else $M := M - N$
```

Commands

Another Example

while true **do** **skip**

Commands

Question

What if there is no σ' such that $\langle c, \sigma \rangle \to \sigma'$?

Summary

- a simple imperative language **IMP**
- a first look at operational semantics
- rules and derivations

Exercise

Let X, Y be locations (i.e., program variables). Let the state σ be given by $\sigma(X) = 3$ and $\sigma(Y) = 5$. Solve the following problems through derivation trees.

(a) For $a = X - 1$, determine the integer n such that $\langle a, \sigma\rangle \to n$.

(b) For $b = Y - X \leq 2$, determine the truth value t such that $\langle b, \sigma\rangle \to t$.

(c) For $c = \textbf{if } Y - X \leq 2 \textbf{ then } Y := X - 1 \textbf{ else skip}$, determine the state σ' such that $\langle c, \sigma\rangle \to \sigma'$.

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

- $w = \textbf{while } b \textbf{ do } c$;
- for all states σ, σ',

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

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: $\langle b, \sigma \rangle \to \textbf{false}$
- from the rule for while-loop:

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

- thus:

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

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:

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

- it follows that:

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

Equivalence of Commands

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

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

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:

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

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

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

$$\frac{\langle a_0, \sigma\rangle \rightarrow_1 \langle a_0', \sigma\rangle}{\langle a_0 + a_1, \sigma\rangle \rightarrow_1 \langle a_0' + a_1, \sigma\rangle}$$

$$\frac{\langle a_1, \sigma\rangle \rightarrow_1 \langle a_1', \sigma\rangle}{\langle n + a_1, \sigma\rangle \rightarrow_1 \langle n + a_1', \sigma\rangle}$$

$$\frac{}{\langle n + m, \sigma\rangle \rightarrow_1 \langle p, \sigma\rangle} \quad (p = m + n)$$

One-Step Operational Semantics

Commands

- big-step semantics: $\langle c, \sigma \rangle \rightarrow \sigma'$
- small-step semantics: $\langle c, \sigma \rangle \rightarrow_1 \langle c', \sigma' \rangle$

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

$$\frac{\langle c_1, \sigma\rangle \rightarrow_1 \langle c_1', \sigma'\rangle}{\langle c_1; c_2, \sigma\rangle \rightarrow_1 \langle c_1'; c_2, \sigma'\rangle}$$

$$\frac{\langle c_1, \sigma\rangle \rightarrow_1 \sigma'}{\langle c_1; c_2, \sigma\rangle \rightarrow_1 \langle c_2, \sigma'\rangle}$$

One-Step Operational Semantics

Question

What about if-branches and while-loops?

Some principles of induction

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
- 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

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

- 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-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 $\forall v \in Q. (v \not\prec u)$.

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)

- 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)

- 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

- **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:

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

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: $\langle n, \sigma \rangle \rightarrow n$, $\langle X, \sigma \rangle \rightarrow \sigma(X)$
- inductive step:

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

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

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

Question

Can we prove this proposition through structural induction?

Question

Proposition

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

Rules for While Loops

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

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

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:

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

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

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

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

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

- inductive step:

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

Program Termination

A Variant of Euclidean's Algorithm

```
while  $\neg(M = N)$  do
   if  $M \leq N$
      then  $N := N - M$
      else  $M := M - N$
```

Program Termination

A Variant of Euclidean's Algorithm

```
Euclid =
while ¬($M = N$) do
  if $M \leq N$
    then $N := N - M$
    else $M := M - N$
```

Termination Property

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

Program Termination

Termination Property

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

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

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

Well-founded Induction

Proof

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

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

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

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

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

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

- 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

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

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

Exercise 1

Problem

Consider the command

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

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.

Exercise 2

Problem

Consider the command

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

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)

Inductive definitions

Topics

- rule induction
- inductive definitions

Rule Induction

textbook, Page 42 – 51

Rule Induction

Rule Instances (X/y)

- E: a set of elements
- $X \subseteq E$: premise (a finite set of elements)
- $y \in E$: conclusion (a single element)

Rule Induction

Axiom Instances: $\emptyset/y$

$$\frac{}{y}$$

Non-axiom Rule Instances: $\{x_1, \ldots, x_n\}/y$

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

Rule Induction

Recall: Derivation Trees

- E: a set of elements
- R: a set of rule instances
- $y \in E$: 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

Rule Induction

Recall: Notations

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

Then we denote

- $d \Vdash_R y$: 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

Rule Induction

Recall: Properties

- 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 $d_1 \Vdash_R x_1, \ldots, d_n \Vdash_R x_n$;

Rule Induction

Notation

- E: a set of elements
- R: a set of rule instances

We define $I_R := \{y \in E \mid \Vdash_R y\}$.

Rule Induction

The Principle

- E: a set of elements
- R: a set of rule instances
- $I_R = \{y \in E \mid \Vdash_R y\}$
- P: a predicate over I_R

Then we have that $\forall x \in I_R . P(x)$ iff

$$\forall (X/y) \in R. [(X \subseteq I_R \ \& \ \forall x \in X. P(x)) \Rightarrow P(y)] \quad .$$

Rule Induction

The Principle

$$\forall (X/y) \in R. [(X \subseteq I_R \ \& \ \forall x \in X. P(x)) \Rightarrow P(y)]$$

- base step: $X = \emptyset$ (axioms)
- inductive step: $X \neq \emptyset$

Rule Induction

Theorem

- E: a set of elements
- R: a set of rule instances
- $I_R = \{y \in E \mid \Vdash_R y\}$
- P: a predicate over I_R

Then we have that $\forall x \in I_R . P(x)$ iff

$$\forall (X/y) \in R . [(X \subseteq I_R \ \& \ \forall x \in X . P(x)) \Rightarrow P(y)] \quad .$$

Rule Induction

Closedness

- R: a set of rule instances
- Q: a set of elements

We say that Q is closed under R (or R-closed) if

$$\forall (X/y) \in R.\, (X \subseteq Q \Rightarrow y \in Q) \quad .$$

Rule Induction

Proposition

- R: a set of rule instances
- Q: a set of elements

Then we have:

- I_R is R-closed;
- if Q is R-closed, then $I_R \subseteq Q$.

Proof

- from definition of I_R
- by induction on derivation trees:

$$P(d) := \forall y. \, [d \Vdash_R y \Rightarrow y \in Q]$$

Rule Induction

Theorem

- E: a set of elements
- R: a set of rule instances
- $I_R = \{y \in E \mid \Vdash_R y\}$
- P: a predicate over I_R

Then we have that $\forall x \in I_R . P(x)$ iff

$$\forall (X/y) \in R . \left[(X \subseteq I_R \ \& \ \forall x \in X . P(x)) \Rightarrow P(y) \right] \quad .$$

Proof

- $Q := \{x \in I_R \mid P(x)\}$ and $Q \subseteq I_R$;
- Q is R-closed and $I_R \subseteq Q$;
- $I_R = Q$ and $\forall x \in I_R . P(x)$.

Rule Induction

Example: Induction on Derivation Trees

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

We define:

- $P_1(c, \sigma, \sigma') := \forall \sigma''. [\langle c, \sigma\rangle \to \sigma'' \Rightarrow \sigma' = \sigma''].$
- $P_2(a, \sigma, n') := \forall n''. [\langle a, \sigma\rangle \to n'' \Rightarrow n' = n''].$
- $P_3(b, \sigma, t') := \forall t''. [\langle b, \sigma\rangle \to t'' \Rightarrow t' = t''].$
- $P := (\text{“}\textbf{Aexp}\text{”} \Rightarrow P_1) \ \& \ (\text{“}\textbf{Bexp}\text{”} \Rightarrow P_2) \ \& \ (\text{“}\textbf{Com}\text{”} \Rightarrow P_3)$
  (i.e., we aggregate all the three cases)

Rule Induction

Special Rule Induction

- general rule induction: a property for all elements
- special rule induction: a property for a part of elements

Rule Induction

Special Rule Induction

- R: a set of rule instances
- $A \subseteq I_R$: a subset
- Q: a predicate over I_R

Then we have that $\forall a \in A. Q(a)$ iff

$$\forall (X/y) \in R. [(X \subseteq I_R \ \&\ y \in A \ \&\ (\forall x \in X \cap A. Q(x))) \Rightarrow Q(y)] \quad .$$

Rule Induction

Special Rule Induction

We have that $\forall a \in A.Q(a)$ iff

$$\forall(X/y) \in R.\,[(X \subseteq I_R \;\&\; y \in A \;\&\; (\forall x \in X \cap A.Q(x))) \Rightarrow Q(y)] \quad .$$

Proof

- $P(x) := x \in A \Rightarrow Q(x)$ and $\forall a \in I_R.P(a) \Leftrightarrow \forall a \in A.Q(a)$;
- $\forall x \in I_R.P(x)$ iff $\forall(X/y) \in R.\,[(X \subseteq I_R \;\&\; \forall x \in X.P(x)) \Rightarrow P(y)]$;
- $\forall(X/y) \in R.\,[(X \subseteq I_R \;\&\; \forall x \in X.(x \in A \Rightarrow Q(x))) \Rightarrow (y \in A \Rightarrow Q(y))]$;
- $\forall(X/y) \in R.\,[(X \subseteq I_R \;\&\; \forall x \in X \cap A.Q(x)) \Rightarrow (y \in A \Rightarrow Q(y))]$;
- $\forall(X/y) \in R.\,[(X \subseteq I_R \;\&\; y \in A \;\&\; \forall x \in X \cap A.Q(x)) \Rightarrow Q(y)]$;

Rule Induction

Example

- Y: a location (program variable)

Then $\forall c, \sigma, \sigma'. [(Y \notin \mathrm{loc}(c)\ \&\ \langle c, \sigma\rangle \to \sigma') \Rightarrow \sigma(Y) = \sigma'(Y)]$.

Proof

- $A := \{\langle c, \sigma\rangle \to \sigma' \mid Y \notin \mathrm{loc}(c)\}$.
- $Q(c, \sigma, \sigma') := \sigma(Y) = \sigma'(Y)$.
- $\forall \langle c, \sigma\rangle \to \sigma' \in A.\ Q(c, \sigma, \sigma')$.
- $\forall (X/y) \in R. [(X \subseteq I_R\ \&\ y \in A\ \&\ (\forall x \in X \cap A. Q(x))) \Rightarrow Q(y)]$

Rule Induction

Proof

- $A := \{\langle c, \sigma\rangle \to \sigma' \mid Y \notin \mathrm{loc}(c)\}$.
- $Q(c, \sigma, \sigma') := \sigma(Y) = \sigma'(Y)$.
- $\forall \langle c, \sigma\rangle \to \sigma' \in A.\ Q(c, \sigma, \sigma')$.
- $\forall (X/y) \in R. [(X \subseteq I_R \ \&\ y \in A \ \&\ (\forall x \in X \cap A. Q(x))) \Rightarrow Q(y)]$

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

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

Rule Induction

Another Example

- w := **while true do skip**

We prove that $\forall \sigma, \sigma'. \langle w, \sigma \rangle \not\to \sigma'$.

Proof

- $A := \{(c, \sigma, \sigma') \mid \langle c, \sigma \rangle \to \sigma' \;\&\; c = w\}$;
- $Q :=$ **false**;
- $A = \emptyset \Leftrightarrow \forall a \in A.\ Q(a)$;
- $\forall (X/y) \in R. [(X \subseteq I_R \;\&\; y \in A \;\&\; (\forall x \in X \cap A. Q(x))) \Rightarrow Q(y)]$

Rule Induction

Proof

- w := **while true do skip**
- $A := \{(c, \sigma, \sigma') \mid \langle c, \sigma \rangle \to \sigma' \ \&\ c = w\};$
- $Q := \textbf{false};$
- $A = \emptyset \Leftrightarrow \forall a \in A.\ Q(a);$
- $\forall (X/y) \in R. [(X \subseteq I_R \ \&\ y \in A \ \&\ (\forall x \in X \cap A. Q(x))) \Rightarrow Q(y)]$

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

Inductive Definitions

textbook, Page 39 – 40

Inductive Definitions

Intuition

- A: a nonempty set
- $h : A \to A$: a function
- $a \in A$: an initial element

There is an infinite sequence $a_0, a_1, \ldots$ such that $a_0 = a$ and $a_{n+1} = h(a_n)$.

Inductive Definitions

The Recursion Theorem

- A: a nonempty set
- $h : A \to A$: a function
- $a \in A$: an initial element

There exists a unique function $f : \mathbb{N} \to A$ such that $f(0) = a$ and $f(n+1) = h(f(n))$.

Inductive Definitions

The Proof Sketch

- T: the set of all functions $g : \{0, \ldots, n\} \to A$ such that
    - $g(0) = a$;
    - $g(k + 1) = h(g(k))$ for all $0 \leq k < n$;
- existence: $f := \{(n, a') \in \mathbb{N} \times A \mid \exists g \in T.g(n) = a'\}$
    - for all n, there is $g \in T$ such that $g(n) = a$ for some $a \in A$;
    - for all n, there exists a unique $a \in A$ such that $(n, a) \in f$;
- uniqueness: for $f, g : \mathbb{N} \to A$, if $f(0) = g(0) = a$, $f(n+1) = h(f(n))$ and $g(n + 1) = h(g(n))$, then we have that $f = g$.

Inductive Definition

Application

- the set of all **IMP** programs
- the set of all derivation trees

Inductive Definitions

Derivation Trees

- R: a set of rule instances

Then we have

- $D_0 := \{(X/y) \in R \mid X = \emptyset\}$
- $d \in D_{n+1}$ iff
  - either $d \in D_0$,
  - or $d = \{d_1, \ldots, d_n\}/y$ for some $d_1, \ldots, d_n \in D_n$ and $(x_1, \ldots, x_n)/y \in R$ such that d_i is rooted at x_i $(1 \leq i \leq n)$
- $D := \bigcup_n D_n$

Inductive Definitions

Definition of I_R

- E: a set of elements
- R: a set of rule instances where all elements are from E

Then we have

- $\widehat{R} : 2^E \to 2^E$: $\widehat{R}(B) := \{y \in E \mid \exists X \subseteq B. (X/y) \in R\}$
- $A \subseteq B \Rightarrow \widehat{R}(A) \subseteq \widehat{R}(B)$
- $A_0 := \emptyset$, $A_{n+1} := \widehat{R}(A_n)$
- $A_0 \subseteq A_1 \subseteq \cdots \subseteq A_n \subseteq \ldots$
- $I_R = \bigcup_n A_n$.

Proposition

- I_R is R-closed;
- if Q is R-closed, then $I_R \subseteq Q$.

Inductive Definitions

Proposition

- I_R is R-closed;
- if Q is R-closed, then $I_R \subseteq Q$.

Proof

- our goal: I_R is R-closed.
- $(X/y) \in R$ and $X \subseteq I_R$
- $X \subseteq A_n$ for some n
- $y \in A_{n+1} \subseteq I_R$

Inductive Definitions

Proposition

- I_R is R-closed;
- if Q is R-closed, then $I_R \subseteq Q$.

Proof

- our goal: if Q is R-closed, then $I_R \subseteq Q$.
- proof by induction on n: $A_n \subseteq Q$

Inductive Definitions

The Example Again (textbook, Page 39)

- $w :=$ **while true do skip**

We prove that $\forall \sigma, \sigma'. \langle w, \sigma \rangle \not\to \sigma'$.

Proof (by Contradiction)

- Suppose that $\exists \sigma, \sigma'. \langle w, \sigma \rangle \to \sigma'$.
- $(w, \sigma, \sigma') \in A_n$ for some n.
- Let n^* be the least such that $(w, \sigma, \sigma') \in A_{n^*}$ for some w, σ, σ'.
- $n^* > 0$ and $(w, \sigma, \sigma') \in A_{n^*-1}$.
- Contradiction to the minimality of n^*.

Inductive Definitions

Well-Founded Recursion (Chapter 10.4)

- B: a set
- $\prec$: a well-founded binary relation on B
- for $b \in B$: $\prec^{-1}\{b\} := \{b' \in B \mid b' \prec b\}$
- for $B' \subseteq B$ and $f : B \to C$: $f{\restriction}B' : B' \to C$ is defined by

$$f{\restriction}B' := \{(b, f(b)) \mid b \in B'\}$$

Inductive Definitions

Well-Founded Recursion (Chapter 10.4)

- B, C: sets
- $\prec$: a well-founded binary relation on B
- $\prec^{-1}\{b\} := \{b' \in B \mid b' \prec b\}$
- $f{\restriction}B' := \{(b, f(b)) \mid b \in B'\}$

Then for any function

$$F : \{(b, h) \mid b \in B, h : \prec^{-1}\{b\} \to C\} \to C$$

there exists a unique function $f : B \to C$ such that

$$\forall b \in B. f(b) = F(b, f{\restriction}\prec^{-1}\{b\}).$$

Inductive Definitions

The loc Function

- $\mathrm{loc}(\textbf{skip}) := \emptyset$
- $\mathrm{loc}(X := a) := \{X\}$
- $\mathrm{loc}(c_0; c_1) := \mathrm{loc}(c_0) \cup \mathrm{loc}(c_1)$
- $\mathrm{loc}(\textbf{if } b \textbf{ then } c_0 \textbf{ else } c_1) := \mathrm{loc}(c_0) \cup \mathrm{loc}(c_1)$
- $\mathrm{loc}(\textbf{while } b \textbf{ do } c) := \mathrm{loc}(c)$

Summary

- rule induction
- inductive definitions
- end of operational semantics (Chapter 2 to Chapter 4)

Exercise 3

Problem

Consider $w := \textbf{while}\ X \leq 1000\ \textbf{do}\ X := (2 \times X) + 1$. Determine the set M of all states σ such that $\exists \sigma'. \langle w, \sigma \rangle \to \sigma'$, and prove that

- $\forall \sigma \in M. \exists \sigma'. \langle w, \sigma \rangle \to \sigma'$;
- $\forall \sigma \in \Sigma \backslash M. \forall \sigma'. \langle w, \sigma \rangle \not\to \sigma'$.

The denotational semantics of IMP

Topics

Denotational Semantics

- complete partial orders
- continuous functions
- a least-fixed-point theorem
- rigorous definition for denotational semantics

Denotational Semantics: An Informal View

Textbook, Page 55 – Page 61

Motivation

Denotational Semantics

- a functional viewpoint for programs
- programs as input-output transformers

Motivation

Equivalence over Commands

- c_0, c_1: two commands

$$
\begin{aligned}
c_0 \sim c_1 \quad & \text{iff} \quad \forall \sigma, \sigma'. (\langle c_0, \sigma \rangle \to \sigma' \Leftrightarrow \langle c_1, \sigma \rangle \to \sigma') \\
& \text{iff} \quad \{(\sigma, \sigma') \mid \langle c_0, \sigma \rangle \to \sigma'\} = \{(\sigma, \sigma') \mid \langle c_1, \sigma \rangle \to \sigma'\}
\end{aligned}
$$

Motivation

- c is represented by $\{(\sigma, \sigma') \mid \langle c, \sigma \rangle \to \sigma'\}$.
- commands as partial functions from inputs to outputs

Motivation

The Mathematical Layout

- arithmetic expressions a: $\mathcal{A}[\![a]\!] : \Sigma \to \mathbb{Z}$
- boolean expressions b: $\mathcal{B}[\![b]\!] : \Sigma \to \{\mathbf{true}, \mathbf{false}\}$
- commands c: $\mathcal{C}[\![c]\!] : \Sigma \rightharpoonup \Sigma$

Arithmetic Expressions

Definition through Well-Founded Recursion

- $\mathcal{A}[\![n]\!](\sigma) := n$ for any state σ;
- $\mathcal{A}[\![X]\!](\sigma) := \sigma(X)$ for any state σ;
- $\mathcal{A}[\![a_0 + a_1]\!](\sigma) := \mathcal{A}[\![a_0]\!](\sigma) + \mathcal{A}[\![a_1]\!](\sigma)$ for any state σ;
- $\mathcal{A}[\![a_0 - a_1]\!](\sigma) := \mathcal{A}[\![a_0]\!](\sigma) - \mathcal{A}[\![a_1]\!](\sigma)$ for any state σ;
- $\mathcal{A}[\![a_0 \times a_1]\!](\sigma) := \mathcal{A}[\![a_0]\!](\sigma) \times \mathcal{A}[\![a_1]\!](\sigma)$ for any state σ;

Boolean Expressions

Definition through Well-Founded Recursion

- $\mathcal{B}[\![\textbf{true}]\!](\sigma) := \textbf{true}$;
- $\mathcal{B}[\![\textbf{false}]\!](\sigma) := \textbf{false}$;

Boolean Expressions

Definition through Well-Founded Recursion

- $$\mathcal{B}[\![a_0 = a_1]\!](\sigma) := \begin{cases} \textbf{true} & \text{if } \mathcal{A}[\![a_0]\!](\sigma) = \mathcal{A}[\![a_1]\!](\sigma) \\ \textbf{false} & \text{if } \mathcal{A}[\![a_0]\!](\sigma) \neq \mathcal{A}[\![a_1]\!](\sigma) \end{cases}$$
- $$\mathcal{B}[\![a_0 \leq a_1]\!](\sigma) := \begin{cases} \textbf{true} & \text{if } \mathcal{A}[\![a_0]\!](\sigma) \leq \mathcal{A}[\![a_1]\!](\sigma) \\ \textbf{false} & \text{if } \mathcal{A}[\![a_0]\!](\sigma) > \mathcal{A}[\![a_1]\!](\sigma) \end{cases}$$

Boolean Expressions

Definition through Well-Founded Recursion

- $\mathcal{B}[\![\neg b]\!](\sigma) := \neg\mathcal{B}[\![b]\!](\sigma)$
- $\mathcal{B}[\![b_0 \wedge b_1]\!](\sigma) := \mathcal{B}[\![b_0]\!](\sigma)\ \&\ \mathcal{B}[\![b_1]\!](\sigma)$
- $\mathcal{B}[\![b_0 \vee b_1]\!](\sigma) := \mathcal{B}[\![b_0]\!](\sigma)$ or $\mathcal{B}[\![b_1]\!](\sigma)$

Denotational Semantics

Exercise

Prove by structural induction that for all arithmetic expressions a and boolean expressions b, $\mathcal{A}[\![a]\!]$ and $\mathcal{B}[\![b]\!]$ are indeed functions.

Commands

Skip and Assignment

- $\mathcal{C}[\![\textbf{skip}]\!] := \{(\sigma, \sigma) \mid \sigma \in \Sigma\}$;
- $\mathcal{C}[\![X := a]\!] := \{(\sigma, \sigma\,[\mathcal{A}[\![a]\!](\sigma)/X]) \mid \sigma \in \Sigma\}$;

Commands

Sequential Composition

- $\mathcal{C}[\![c_0; c_1]\!] := \mathcal{C}[\![c_1]\!] \circ \mathcal{C}[\![c_0]\!]$;

Commands

Conditional Branch

- $\mathcal{C}[\![\textbf{if } b \textbf{ then } c_0 \textbf{ else } c_1]\!]$ is the union of the following two sets:
    - $\{(\sigma, \sigma') \mid \mathcal{B}[\![b]\!](\sigma) = \textbf{true} \text{ and } (\sigma, \sigma') \in \mathcal{C}[\![c_0]\!]\}$
    - $\{(\sigma, \sigma') \mid \mathcal{B}[\![b]\!](\sigma) = \textbf{false} \text{ and } (\sigma, \sigma') \in \mathcal{C}[\![c_1]\!]\}$

Commands

While Loop

- w = **while** b **do** c;
- How??

Commands

A First Attempt

- $w = \textbf{while } b \textbf{ do } c;$
- $w \sim \textbf{if } b \textbf{ then } c; w \textbf{ else } \textbf{skip};$
- $$\begin{aligned}\mathcal{C}[\![w]\!] = &\{(\sigma, \sigma) \mid \mathcal{B}[\![b]\!](\sigma) = \textbf{false}\} \cup \\ &\{(\sigma, \sigma') \mid \mathcal{B}[\![b]\!](\sigma) = \textbf{true} \text{ and } (\sigma, \sigma') \in \mathcal{C}[\![w]\!] \circ \mathcal{C}[\![c]\!]\}\end{aligned}$$

Commands

The Problem

- $$\begin{aligned}\mathcal{C}[\![w]\!] = \{(\sigma,\sigma) \mid \mathcal{B}[\![b]\!](\sigma) = \textbf{false}\} \cup \\ \{(\sigma,\sigma') \mid \mathcal{B}[\![b]\!](\sigma) = \textbf{true} \text{ and } (\sigma,\sigma') \in \mathcal{C}[\![w]\!] \circ \mathcal{C}[\![c]\!]\}\end{aligned}$$
- $\mathcal{C}[\![w]\!]$ is not recursively defined.

Commands

The Fixed-Point Phenomenon

- $w = \textbf{while } b \textbf{ do } c;$
-
$$\begin{aligned}\mathcal{C}[\![w]\!] = \{(\sigma,\sigma) \mid \mathcal{B}[\![b]\!](\sigma) = \textbf{false}\} \cup \\ \{(\sigma,\sigma') \mid \mathcal{B}[\![b]\!](\sigma) = \textbf{true} \text{ and } (\sigma,\sigma') \in \mathcal{C}[\![w]\!] \circ \mathcal{C}[\![c]\!]\}\end{aligned}$$
- $\mathcal{C}[\![w]\!]$ should be a solution to the following set equation:
$$\begin{aligned}R = \{(\sigma,\sigma) \mid \mathcal{B}[\![b]\!](\sigma) = \textbf{false}\} \cup \\ \{(\sigma,\sigma') \mid \mathcal{B}[\![b]\!](\sigma) = \textbf{true} \text{ and } (\sigma,\sigma') \in R \circ \mathcal{C}[\![c]\!]\}\end{aligned}$$

Commands

The Fixed-Point Phenomenon

- $w = \textbf{while } b \textbf{ do } c$;
- $\mathcal{C}[\![w]\!]$ should be a solution to the following set equation:

$$R = \{(\sigma,\sigma) \mid \mathcal{B}[\![b]\!](\sigma) = \textbf{false}\} \cup \\ \{(\sigma,\sigma') \mid \mathcal{B}[\![b]\!](\sigma) = \textbf{true} \text{ and } (\sigma,\sigma') \in R \circ \mathcal{C}[\![c]\!]\}$$

- Does any solution R work ?
  - The set $R = \{(\sigma,\sigma) \mid \sigma \in \Sigma\}$ is a solution when $c = \textbf{skip}$.
  - However, we desire $\mathcal{C}[\![w]\!] = \emptyset$ when $c = \textbf{skip}$ and $b = \textbf{true}$.
- What do we desire about $\mathcal{C}[\![w]\!]$?
  - The set $R = \{(\sigma,\sigma) \mid \sigma \in \Sigma\}$ contains too much information.
  - $\mathcal{C}[\![w]\!]$ should be the solution with the least information.

Complete Partial Orders

Textbook, Page 68 – Page 70

Complete Partial Orders

Motivation

- a partial order to compare elements
- a complete property in infinitely ascending sequences
- a fundamental characterization with least fixed points

Complete Partial Orders

Recall: Partial Orders

A partial order is an ordered pair $(P, \sqsubseteq)$ such that P is a set and $\sqsubseteq$ is a binary relation $\sqsubseteq \ \subseteq P \times P$ satisfying the following conditions:

- (reflexibility) $\forall p \in P. p \sqsubseteq p$;
- (transitivity) $\forall p, q, r \in P. [(p \sqsubseteq q \ \& \ q \sqsubseteq r) \Rightarrow p \sqsubseteq r]$;
- (antisymmetry) $\forall p, q \in P. [(p \sqsubseteq q \ \& \ q \sqsubseteq p) \Rightarrow p = q]$.

Complete Partial Orders

Upper Bounds

- $(P, \sqsubseteq)$: a partial order
- X: a subset of P (i.e., that satisfies $X \subseteq P$)

$p \in P$ is an upper bound of X if $\forall q \in X. q \sqsubseteq p$.

Least Upper Bounds

$p \in P$ is a least upper bound (in short, lub) of X if

- p is an upper bound of X, and
- for all upper bounds q of X, $p \sqsubseteq q$

Exercise

For any $X \subseteq P$, X has at most one least upper bound.

Complete Partial Orders

Least Upper Bounds

$p \in P$ is a least upper bound (in short, lub) of X if

- p is an upper bound of X, and
- for all upper bounds q of X, $p \sqsubseteq q$

Notation

- The least upper bound of X (if exists) is denoted by $\bigsqcup X$.
- If $X = \{d_1, \ldots, d_n\}$, then $d_1 \sqcup \cdots \sqcup d_n := \bigsqcup X$.

Complete Partial Orders

ω-Chains

- $(P, \sqsubseteq)$: a partial order

An ω-chain in P is an infinite sequence $d_0, d_1, \ldots, d_n, \ldots$ in P such that $d_0 \sqsubseteq d_1 \sqsubseteq \cdots \sqsubseteq d_n \sqsubseteq \ldots$.

Complete Partial Orders (CPOs)

$(P, \sqsubseteq)$ is a complete partial order (cpo) if for any ω-chain

$$d_0 \sqsubseteq d_1 \sqsubseteq \cdots \sqsubseteq d_n \sqsubseteq \ldots$$

in P, the least upper bound

$$\bigsqcup_{n\in\omega} d_n := \bigsqcup\{d_n \mid n \in \omega\} = \bigsqcup\{d_0, d_1, \ldots, d_n, \ldots\}$$

exists in P.

Complete Partial Orders

Least Elements

- $(P, \sqsubseteq)$: a partial order

$p \in P$ is a least element if $\forall q \in P . p \sqsubseteq q$.

Exercise

Show that the least element, if exists, is unique.

CPOs with Bottom

- $(P, \sqsubseteq)$: a cpo

$(P, \sqsubseteq)$ is a cpo with bottom if P has a (unique) least element $\bot_P$.

Complete Partial Orders

Set Inclusion

- A: a set
- $D := 2^A$
- $\sqsubseteq := \{(X, Y) \in D \times D \mid X \subseteq Y\}$

Exercise

Verify that $(D, \sqsubseteq)$ is a cpo with bottom.

- $\bigsqcup_{n\in\omega} A_n = \bigcup_n A_n$ given $A_0 \subseteq A_1 \subseteq \ldots$
- $\bot_D = \emptyset$

Complete Partial Orders

Partial Functions

- B, C: sets
- $D := \{F \mid F : B \rightharpoonup C\}$
- $\sqsubseteq := \{(F, G) \in D \times D \mid F \subseteq G\}$

Exercise

Verify that $(D, \sqsubseteq)$ is a cpo with bottom.

- $\bigsqcup_{n\in\omega} F_n = \bigcup_n F_n$ given $F_0 \subseteq F_1 \subseteq \ldots$
- (important!) $\bigcup_n F_n$ is a function!
- $\perp_D = \emptyset$

Complete Partial Orders

Intervals

- $D := [0, \infty) \cup \{\infty\}$
- $\sqsubseteq := \{(x, y) \in D \times D \mid x \leq y\}$

Exercise

Verify that $(D, \sqsubseteq)$ is a cpo with bottom.

- $\bigsqcup_{n \in \omega} x_n = \sup_n x_n$ if $x_0 \leq x_1 \leq \ldots$
- $\bot_D = 0$

Complete Partial Orders

Intervals

- $D := [0,1)$
- $\sqsubseteq := \{(x,y) \in D \times D \mid x \leq y\}$

Exercise

Is $(D, \sqsubseteq)$ a cpo (with bottom) ?

Complete Partial Orders

Real Numbers

- $D := \mathbb{R}$
- $\sqsubseteq := \{(x, y) \in D \times D \mid x \leq y\}$

Exercise

Is $(D, \sqsubseteq)$ a cpo (with bottom) ?

Continuous Functions

Textbook, Page 71 – Page 72

Monotonic Functions

Definition

- $(D, \sqsubseteq_D)$ and $(E, \sqsubseteq_E)$: partial orders

A function $f : D \to E$ is monotonic if

$$\forall d, d' \in D. [d \sqsubseteq_D d' \Rightarrow f(d) \sqsubseteq_E f(d')]$$

Example

- partial order: $(\mathbb{R}, \leq)$
- $f(x) = 2 \cdot x$ is a monotonic function.

Continuous Functions

Definition

- $(D, \sqsubseteq_D)$ and $(E, \sqsubseteq_E)$: cpo's

A function $f : D \to E$ is continuous if the followings hold:

- f is monotonic;
- for all ω-chains $d_0 \sqsubseteq d_1 \sqsubseteq \cdots \sqsubseteq d_n \sqsubseteq \ldots$ in D, we have that

$$\bigsqcup_{n\in\omega} f(d_n) = f\left(\bigsqcup_{n\in\omega} d_n\right)$$

Example

- the cpo: $([0,1], \leq)$
- $f(x) = 2 \cdot x$ is a continuous function.

Continuous Functions

Definition

- $(D, \sqsubseteq_D)$ and $(E, \sqsubseteq_E)$: cpo's

A function $f : D \to E$ is continuous if the followings hold:

- f is monotonic;
- for all ω-chains $d_0 \sqsubseteq d_1 \sqsubseteq \cdots \sqsubseteq d_n \sqsubseteq \ldots$ in D, we have that

$$\bigsqcup_{n\in\omega} f(d_n) = f\left(\bigsqcup_{n\in\omega} d_n\right)$$

Question

Can one construct a monotonic function which is not continuous?

Fixed Points

Definition

- $(D, \sqsubseteq_D)$: a partial order
- $f : D \to D$: a function

An element $d \in D$ is:

- a fixed point of f if $f(d) = d$;
- a prefixed point of f if $f(d) \sqsubseteq d$;

Fixed Points

The Fixed-Point Theorem

Suppose

- $(D, \sqsubseteq_D)$: a cpo with bottom $\bot_D$
- $f : D \to D$: a continuous function
- $\bot_D \sqsubseteq_D f(\bot_D) \sqsubseteq_D \cdots \sqsubseteq_D f^n(\bot_D) \sqsubseteq_D \ldots$
- $\mathit{fix}(f) := \bigsqcup_{n \in \omega} f^n(\bot_D)$

Then

- $\mathit{fix}(f)$ is a fixed point of f: $f(\mathit{fix}(f)) = \mathit{fix}(f)$
- $\mathit{fix}(f)$ is the least prefixed point of f: $f(d) \sqsubseteq d \Rightarrow \mathit{fix}(f) \sqsubseteq d$
- $\mathit{fix}(f)$ is the least fixed point of f: $f(d) = d \Rightarrow \mathit{fix}(f) \sqsubseteq d$

Fixed Points

Proof

- $\mathit{fix}(f) := \bigsqcup_{n\in\omega} f^n(\bot_D)$
- $\mathit{fix}(f)$ is a fixed point of f: $f(\mathit{fix}(f)) = \mathit{fix}(f)$

$$\begin{array}{rcl}
f(\mathit{fix}(f)) & = & f(\bigsqcup_{n\in\omega} f^n(\bot_D)) \\
 & = & \bigsqcup_{n\in\omega} f^{n+1}(\bot_D) \\
 & = & \bigsqcup_{n\in\omega} f^{n+1}(\bot_D) \sqcup \bot_D \\
 & = & \bigsqcup_{n\in\omega} f^n(\bot_D) \\
 & = & \mathit{fix}(f)
\end{array}$$

Fixed Points

Fixed-Point Theorem: Proof

- $\mathit{fix}(f) := \bigsqcup_{n\in\omega} f^n(\bot_D)$
- $\mathit{fix}(f)$ is the least prefixed point of f: $f(d) \sqsubseteq d \Rightarrow \mathit{fix}(f) \sqsubseteq d$
- d: a prefixed point (i.e., $f(d) \sqsubseteq d$)
- $\bot_D \sqsubseteq d$
- $f^n(\bot_D) \sqsubseteq d \Rightarrow f^{n+1}(\bot_D) \sqsubseteq f(d) \sqsubseteq d$
- $\forall n. \left(f^n(\bot_D) \sqsubseteq d\right)$
- $\mathit{fix}(f) = \bigsqcup_{n\in\omega} f^n(\bot_D) \sqsubseteq d$

Denotational Semantics: Formal Definition

Textbook, Page 55 – Page 61

Recall: Denotational Semantics

- commands as partial functions from inputs to outputs
- c is represented by the partial function $\{(\sigma, \sigma') \mid \langle c, \sigma \rangle \to \sigma'\}$.

Denotational Semantics

Recall: The Mathematical Layout

- arithmetic expressions a: $\mathcal{A}[\![a]\!] : \Sigma \to \mathbb{N}$
- boolean expressions b: $\mathcal{B}[\![b]\!] : \Sigma \to \{\textbf{true}, \textbf{false}\}$
- commands c: $\mathcal{C}[\![c]\!] : \Sigma \rightharpoonup \Sigma$

Arithmetic Expressions

Recall: Definition through Well-Founded Recursion

- $\mathcal{A}[\![n]\!](\sigma) := n$ for any state σ;
- $\mathcal{A}[\![X]\!](\sigma) := \sigma(X)$ for any state σ;
- $\mathcal{A}[\![a_0 + a_1]\!](\sigma) := \mathcal{A}[\![a_0]\!](\sigma) + \mathcal{A}[\![a_1]\!](\sigma)$ for any state σ;
- $\mathcal{A}[\![a_0 - a_1]\!](\sigma) := \mathcal{A}[\![a_0]\!](\sigma) - \mathcal{A}[\![a_1]\!](\sigma)$ for any state σ;
- $\mathcal{A}[\![a_0 \times a_1]\!](\sigma) := \mathcal{A}[\![a_0]\!](\sigma) \times \mathcal{A}[\![a_1]\!](\sigma)$ for any state σ;

Boolean Expressions

Recall: Definition through Well-Founded Recursion

- $\mathcal{B}[\![\textbf{true}]\!](\sigma) := \textbf{true}$;
- $\mathcal{B}[\![\textbf{false}]\!](\sigma) := \textbf{false}$;

Boolean Expressions

Recall: Definition through Well-Founded Recursion

- $$\mathcal{B}[\![a_0 = a_1]\!](\sigma) := \begin{cases} \textbf{true} & \text{if } \mathcal{A}[\![a_0]\!](\sigma) = \mathcal{A}[\![a_1]\!](\sigma) \\ \textbf{false} & \text{if } \mathcal{A}[\![a_0]\!](\sigma) \neq \mathcal{A}[\![a_1]\!](\sigma) \end{cases}$$
- $$\mathcal{B}[\![a_0 \leq a_1]\!](\sigma) := \begin{cases} \textbf{true} & \text{if } \mathcal{A}[\![a_0]\!](\sigma) \leq \mathcal{A}[\![a_1]\!](\sigma) \\ \textbf{false} & \text{if } \mathcal{A}[\![a_0]\!](\sigma) > \mathcal{A}[\![a_1]\!](\sigma) \end{cases}$$

Boolean Expressions

Recall: Definition through Well-Founded Recursion

- $\mathcal{B}[\![\neg b]\!](\sigma) := \neg\mathcal{B}[\![b]\!](\sigma)$
- $\mathcal{B}[\![b_0 \wedge b_1]\!](\sigma) := \mathcal{B}[\![b_0]\!](\sigma) \mathbin{\&} \mathcal{B}[\![b_1]\!](\sigma)$
- $\mathcal{B}[\![b_0 \vee b_1]\!](\sigma) := \mathcal{B}[\![b_0]\!](\sigma)$ or $\mathcal{B}[\![b_1]\!](\sigma)$

Commands

Recall: Assignment and Skip

- $\mathcal{C}[\![\textbf{skip}]\!] := \{(\sigma, \sigma) \mid \sigma \in \Sigma\};$
- $\mathcal{C}[\![X := a]\!] := \{(\sigma, \sigma\,[\mathcal{A}[\![a]\!](\sigma)/X]) \mid \sigma \in \Sigma\};$

Commands

Recall: Sequential Composition

- $\mathcal{C}[\![c_0; c_1]\!] := \mathcal{C}[\![c_1]\!] \circ \mathcal{C}[\![c_0]\!];$

Commands

If Branch

- $\mathcal{C}[\![\textbf{if } b \textbf{ then } c_0 \textbf{ else } c_1]\!]$ is the union of the following two sets:
  - $\{(\sigma, \sigma') \mid \mathcal{B}[\![b]\!](\sigma) = \textbf{true} \text{ and } (\sigma, \sigma') \in \mathcal{C}[\![c_0]\!]\}$;
  - $\{(\sigma, \sigma') \mid \mathcal{B}[\![b]\!](\sigma) = \textbf{false} \text{ and } (\sigma, \sigma') \in \mathcal{C}[\![c_1]\!]\}$

Commands

While Loop

- w = **while** b **do** c;
- $w \sim$ **if** b **then** c; w **else** **skip**;
- $$\begin{aligned}\mathcal{C}[\![w]\!] = \{(\sigma,\sigma) \mid \mathcal{B}[\![b]\!](\sigma) = \textbf{false}\}\cup \\ \{(\sigma,\sigma') \mid \mathcal{B}[\![b]\!](\sigma) = \textbf{true} \text{ and } (\sigma,\sigma') \in \mathcal{C}[\![w]\!] \circ \mathcal{C}[\![c]\!]\}\end{aligned}$$

Commands

The Fixed-Point Phenomenon

- $w = \textbf{while } b \textbf{ do } c$;
- $$\begin{aligned}\mathcal{C}[\![w]\!] = \{(\sigma,\sigma) \mid \mathcal{B}[\![b]\!](\sigma) = \textbf{false}\}\cup \\ \{(\sigma,\sigma') \mid \mathcal{B}[\![b]\!](\sigma) = \textbf{true} \text{ and } (\sigma,\sigma') \in \mathcal{C}[\![w]\!] \circ \mathcal{C}[\![c]\!]\}\end{aligned}$$
- $\mathcal{C}[\![w]\!]$ should be a solution to the following set equation:

$$\begin{aligned}R = \{(\sigma,\sigma) \mid \mathcal{B}[\![b]\!](\sigma) = \textbf{false}\}\cup \\ \{(\sigma,\sigma') \mid \mathcal{B}[\![b]\!](\sigma) = \textbf{true} \text{ and } (\sigma,\sigma') \in R \circ \mathcal{C}[\![c]\!]\}\end{aligned}$$

Commands

The Fixed-Point Phenomenon

- $w = \textbf{while } b \textbf{ do } c$;
- $$\begin{aligned} \mathcal{C}[\![w]\!] = \{(\sigma,\sigma) \mid \mathcal{B}[\![b]\!](\sigma) = \textbf{false}\} \cup \\ \{(\sigma,\sigma') \mid \mathcal{B}[\![b]\!](\sigma) = \textbf{true} \text{ and } (\sigma,\sigma') \in \mathcal{C}[\![w]\!] \circ \mathcal{C}[\![c]\!]\} \end{aligned}$$
- Define $\Gamma : (\Sigma \rightharpoonup \Sigma) \to (\Sigma \rightharpoonup \Sigma)$ by
$$\begin{aligned} \Gamma(F) := \{(\sigma,\sigma) \mid \mathcal{B}[\![b]\!](\sigma) = \textbf{false}\} \cup \\ \{(\sigma,\sigma') \mid \mathcal{B}[\![b]\!](\sigma) = \textbf{true} \text{ and } (\sigma,\sigma') \in F \circ \mathcal{C}[\![c]\!]\} \end{aligned}$$
- $\Gamma(\mathcal{C}[\![w]\!]) = \mathcal{C}[\![w]\!]$.

Commands

The Fixed-Point Phenomenon

- $w = \textbf{while } b \textbf{ do } c$;
- Define $\Gamma : (\Sigma \rightharpoonup \Sigma) \to (\Sigma \rightharpoonup \Sigma)$ by

$$\begin{aligned}\Gamma(F) := \{(\sigma,\sigma) \mid \mathcal{B}[\![b]\!](\sigma) = \textbf{false}\} \cup \\ \{(\sigma,\sigma') \mid \mathcal{B}[\![b]\!](\sigma) = \textbf{true} \text{ and } (\sigma,\sigma') \in F \circ \mathcal{C}[\![c]\!]\}\end{aligned}$$

- $((\Sigma \rightharpoonup \Sigma), \subseteq)$: the complete partial order
- Γ: a continuous function for $((\Sigma \rightharpoonup \Sigma), \subseteq)$

Exercise

- $((\Sigma \rightharpoonup \Sigma), \subseteq)$ is a complete partial order.
- Γ is a continuous function for $((\Sigma \rightharpoonup \Sigma), \subseteq)$.

Commands

The Fixed-Point Phenomenon

- $w =$ **while** b **do** c;
- Define $\Gamma : (\Sigma \rightharpoonup \Sigma) \to (\Sigma \rightharpoonup \Sigma)$ by

$$\begin{aligned}\Gamma(F) := \{(\sigma, \sigma) \mid \mathcal{B}[\![b]\!](\sigma) = \textbf{false}\} \cup \\ \{(\sigma, \sigma') \mid \mathcal{B}[\![b]\!](\sigma) = \textbf{true} \text{ and } (\sigma, \sigma') \in F \circ \mathcal{C}[\![c]\!]\}\end{aligned}$$

- $((\Sigma \rightharpoonup \Sigma), \subseteq)$: the complete partial order
- Γ: a continuous function for $((\Sigma \rightharpoonup \Sigma), \subseteq)$

Definition for $\mathcal{C}[\![w]\!]$

- $\mathcal{C}[\![w]\!] := \mathit{fix}(\Gamma) = \bigsqcup_{n \in \omega} \Gamma^n(\bot) = \bigcup_{n \in \omega} \Gamma^n(\emptyset)$;

Commands

The Intuition

- w = **while** b **do** c;
- $\Gamma : (\Sigma \rightharpoonup \Sigma) \to (\Sigma \rightharpoonup \Sigma)$ is given by

$$\begin{aligned}\Gamma(F) := \{(\sigma,\sigma) \mid \mathcal{B}[\![b]\!](\sigma) = \textbf{false}\} \cup \\ \{(\sigma,\sigma') \mid \mathcal{B}[\![b]\!](\sigma) = \textbf{true} \text{ and } (\sigma,\sigma') \in F \circ \mathcal{C}[\![c]\!]\}\end{aligned}$$

- $\mathcal{C}[\![w]\!] := \mathit{fix}(\Gamma) = \bigsqcup_{n\in\omega} \Gamma^n(\bot) = \bigcup_{n\in\omega} \Gamma^n(\emptyset)$;

Example

- w = **while true do skip**
- $\mathcal{C}[\![w]\!] = \emptyset$

Commands

Theorem

For all commands c, $\mathcal{C}[\![c]\!]$ is a partial function from Σ to Σ.

Proof

By structural induction.

Summary

- complete partial orders
- continuous functions
- a fixed-point theorem
- denotational semantics

Exercise

Problem 1

Let D be a non-empty set and $(D \rightharpoonup D)$ be the set of all partial functions from D to D. Prove that the partial order $((D \rightharpoonup D), \subseteq)$ (i.e., the set of partial functions ordered by set inclusion) is a complete partial order with bottom.

Exercise

Problem 2

- Prove that $(\mathbb{N}, \geq)$ is a cpo.
- Prove that $(\mathcal{P}(\mathbb{N})\backslash\{\emptyset\}, \subseteq)$ is a cpo.
- Determine whether the function $F : \mathcal{P}(\mathbb{N})\backslash\{\emptyset\} \to \mathbb{N}$ given by

  $$F(A) := \text{“the minimal number in } A\text{”}$$

  is a continuous function from $(\mathcal{P}(\mathbb{N})\backslash\{\emptyset\}, \subseteq)$ to $(\mathbb{N}, \geq)$. Prove your answer.

Topics

- equivalence with operational semantics
- Knaster-Tarski's Fixed-Point Theorem
- the bottom element

Equivalence with Operational Semantics

Textbook, Page 61 – 68

Equivalence with Operational Semantics

Denotational Semantics: Pros

- an elegant definition through fixed-point theory
- an operational-independent definition through partial functions

Key Question

- Does it really meet with operational semantics?

Equivalence with Operational Semantics

Equivalence Statement

- $\mathcal{A}[\![a]\!] = \{(\sigma, n) \in \Sigma \times \mathbb{Z} \mid \langle a, \sigma\rangle \to n\}$.
- $\mathcal{B}[\![b]\!] = \{(\sigma, t) \in \Sigma \times \{\textbf{true}, \textbf{false}\} \mid \langle b, \sigma\rangle \to t\}$.
- $\mathcal{C}[\![c]\!] = \{(\sigma, \sigma') \in \Sigma \times \Sigma \mid \langle c, \sigma\rangle \to \sigma'\}$.

Equivalence with Operational Semantics

Arithmetic Expressions

Prove by structural induction that

$$\forall a \in \textbf{Aexp}. \forall \sigma \in \Sigma. \forall n \in \mathbb{Z}. \left(\mathcal{A}[\![a]\!](\sigma) = n \Leftrightarrow \langle a, \sigma \rangle \rightarrow n \right)$$

Recall: Operational Semantics

Numbers and Locations

$$\frac{}{\langle n, \sigma \rangle \to n} \qquad \frac{}{\langle X, \sigma \rangle \to \sigma(X)}$$

- rules without premise: axioms
- n, X, σ: metavariables

Recall: Operational Semantics

Arithmetic Operations

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

$$\frac{\langle a_0, \sigma\rangle \to n_0, \ \ \langle a_1, \sigma\rangle \to n_1}{\langle a_0 \times a_1, \sigma\rangle \to n_0 \cdot n_1}$$

- $n_0, n_1, a_0, a_1, \sigma$: metavariables

Recall: Denotational Semantics

Definition through Well-Founded Recursion

- $\mathcal{A}[\![n]\!](\sigma) := n$ for any state σ;
- $\mathcal{A}[\![X]\!](\sigma) := \sigma(X)$ for any state σ;
- $\mathcal{A}[\![a_0 + a_1]\!](\sigma) := \mathcal{A}[\![a_0]\!](\sigma) + \mathcal{A}[\![a_1]\!](\sigma)$ for any state σ;
- $\mathcal{A}[\![a_0 - a_1]\!](\sigma) := \mathcal{A}[\![a_0]\!](\sigma) - \mathcal{A}[\![a_1]\!](\sigma)$ for any state σ;
- $\mathcal{A}[\![a_0 \times a_1]\!](\sigma) := \mathcal{A}[\![a_0]\!](\sigma) \times \mathcal{A}[\![a_1]\!](\sigma)$ for any state σ;

Equivalence with Operational Semantics

Boolean Expressions

Prove by structural induction that

$$\forall b \in \mathbf{Bexp}.\forall \sigma \in \Sigma.\forall t \in \{\mathbf{true}, \mathbf{false}\}.\,(\mathcal{B}[\![b]\!](\sigma) = t \Leftrightarrow \langle b, \sigma \rangle \to t)$$

Equivalence with Operational Semantics

Commands

We need to prove that

$$\forall c \in \textbf{Com}. \forall \sigma, \sigma' \in \Sigma. ((\sigma, \sigma') \in \mathcal{C}[\![c]\!] \Leftrightarrow \langle c, \sigma \rangle \rightarrow \sigma')$$

Equivalence with Operational Semantics

Commands: One Direction

$$\forall c \in \textbf{Com}.\forall \sigma, \sigma' \in \Sigma.\,(\langle c, \sigma \rangle \to \sigma' \Rightarrow (\sigma, \sigma') \in \mathcal{C}[\![c]\!])$$

Proof

By special rule induction:

- $A := \{(c, \sigma, \sigma') \mid \langle c, \sigma \rangle \to \sigma'\}$
- $Q(c, \sigma, \sigma') := (\sigma, \sigma') \in \mathcal{C}[\![c]\!]$
- Then we have that $\forall a \in A.Q(a)$ iff

$$\forall (X/y) \in R.\,[(X \subseteq I_R \ \&\ y \in A \ \&\ (\forall x \in X \cap A.Q(x))) \Rightarrow Q(y)] \quad .$$

Equivalence with Operational Semantics

Atomic Commands

- $$\frac{}{\langle \textbf{skip}, \sigma\rangle \to \sigma} \qquad \frac{\langle a, \sigma\rangle \to m}{\langle X := a, \sigma\rangle \to \sigma\,[m/X]}$$
- $\mathcal{C}[\![\textbf{skip}]\!](\sigma) = \sigma$
- $\mathcal{C}[\![X := a]\!](\sigma) = \sigma\,[\mathcal{A}[\![a]\!](\sigma)/X]$

Equivalence with Operational Semantics

Sequential Composition

- $$\frac{\langle c_0, \sigma\rangle \to \sigma'',\ \langle c_1, \sigma''\rangle \to \sigma'}{\langle c_0; c_1, \sigma\rangle \to \sigma'}$$
- $\mathcal{C}[\![c_0; c_1]\!] = \mathcal{C}[\![c_1]\!] \circ \mathcal{C}[\![c_0]\!]$

Equivalence with Operational Semantics

If-Branch

- $$\frac{\langle b, \sigma\rangle \to \mathbf{true},\ \langle c_0, \sigma\rangle \to \sigma'}{\langle \mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1, \sigma\rangle \to \sigma'}$$
  $$\frac{\langle b, \sigma\rangle \to \mathbf{false},\ \langle c_1, \sigma\rangle \to \sigma'}{\langle \mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1, \sigma\rangle \to \sigma'}$$
- $\mathcal{C}[\![\mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1]\!]$ is the union of the following two sets:
  - $\{(\sigma, \sigma') \mid \mathcal{B}[\![b]\!](\sigma) = \mathbf{true}$ and $(\sigma, \sigma') \in \mathcal{C}[\![c_0]\!]\}$;
  - $\{(\sigma, \sigma') \mid \mathcal{B}[\![b]\!](\sigma) = \mathbf{false}$ and $(\sigma, \sigma') \in \mathcal{C}[\![c_1]\!]\}$;

Equivalence with Operational Semantics

While Loops

- $$\frac{\langle b, \sigma\rangle \to \textbf{false}}{\langle \textbf{while } b \textbf{ do } c, \sigma\rangle \to \sigma}$$
  $$\frac{\langle b, \sigma\rangle \to \textbf{true},\ \ \langle c, \sigma\rangle \to \sigma'',\ \ \langle \textbf{while } b \textbf{ do } c, \sigma''\rangle \to \sigma'}{\langle \textbf{while } b \textbf{ do } c, \sigma\rangle \to \sigma'}$$
- $w = \textbf{while } b \textbf{ do } c;$
- $$\begin{aligned}\mathcal{C}[\![w]\!] = \{(\sigma, \sigma) \mid \mathcal{B}[\![b]\!](\sigma) = \textbf{false}\} \cup \\ \{(\sigma, \sigma') \mid \mathcal{B}[\![b]\!](\sigma) = \textbf{true} \text{ and } (\sigma, \sigma') \in \mathcal{C}[\![w]\!] \circ \mathcal{C}[\![c]\!]\}\end{aligned}$$

Equivalence with Operational Semantics

Commands: The Other Direction

$$\forall c \in \mathbf{Com}. \forall \sigma, \sigma' \in \Sigma. \left((\sigma, \sigma') \in \mathcal{C}[\![c]\!] \Rightarrow \langle c, \sigma \rangle \to \sigma' \right)$$

Proof

By structural induction on c.

Equivalence with Operational Semantics

Atomic Commands

- $$\frac{}{\langle \textbf{skip}, \sigma\rangle \to \sigma} \qquad \frac{\langle a, \sigma\rangle \to m}{\langle X := a, \sigma\rangle \to \sigma\,[m/X]}$$
- $\mathcal{C}[\![\textbf{skip}]\!](\sigma) = \sigma$
- $\mathcal{C}[\![X := a]\!](\sigma) = \sigma\,[\mathcal{A}[\![a]\!](\sigma)/X]$

Equivalence with Operational Semantics

Sequential Composition

- $\mathcal{C}[\![c_0; c_1]\!] = \mathcal{C}[\![c_1]\!] \circ \mathcal{C}[\![c_0]\!]$
- $$\dfrac{\dfrac{\vdots}{\langle c_0, \sigma\rangle \to \sigma''} \qquad \dfrac{\vdots}{\langle c_1, \sigma''\rangle \to \sigma'}}{\langle c_0; c_1, \sigma\rangle \to \sigma'}$$

Equivalence with Operational Semantics

If-Branch

- $\mathcal{C}[\![\textbf{if } b \textbf{ then } c_0 \textbf{ else } c_1]\!]$ is the union of the following two sets:
  - $\{(\sigma, \sigma') \mid \mathcal{B}[\![b]\!](\sigma) = \textbf{true} \text{ and } (\sigma, \sigma') \in \mathcal{C}[\![c_0]\!]\}$;
  - $\{(\sigma, \sigma') \mid \mathcal{B}[\![b]\!](\sigma) = \textbf{false} \text{ and } (\sigma, \sigma') \in \mathcal{C}[\![c_1]\!]\}$;
-

$$\dfrac{\dfrac{\vdots}{\langle b, \sigma\rangle \to \textbf{true}} \qquad \dfrac{\vdots}{\langle c_0, \sigma\rangle \to \sigma'}}{\langle \textbf{if } b \textbf{ then } c_0 \textbf{ else } c_1, \sigma\rangle \to \sigma'}$$

$$\dfrac{\dfrac{\vdots}{\langle b, \sigma\rangle \to \textbf{false}} \qquad \dfrac{\vdots}{\langle c_1, \sigma\rangle \to \sigma'}}{\langle \textbf{if } b \textbf{ then } c_0 \textbf{ else } c_1, \sigma\rangle \to \sigma'}$$

Equivalence with Operational Semantics

While Loops

- $w =$ **while** b **do** c;
- Define $\Gamma : (\Sigma \rightharpoonup \Sigma) \to (\Sigma \rightharpoonup \Sigma)$ by

$$\begin{aligned}\Gamma(F) := \{(\sigma, \sigma) \mid \mathcal{B}[\![b]\!](\sigma) = \textbf{false}\} \cup \\ \{(\sigma, \sigma') \mid \mathcal{B}[\![b]\!](\sigma) = \textbf{true} \text{ and } (\sigma, \sigma') \in F \circ \mathcal{C}[\![c]\!]\}\end{aligned}$$

- $((\Sigma \rightharpoonup \Sigma), \subseteq)$: the complete partial order
- Γ: a continuous function for $((\Sigma \rightharpoonup \Sigma), \subseteq)$
- $\mathcal{C}[\![w]\!] := \mathit{fix}(\Gamma) = \bigsqcup_{n \in \omega} \Gamma^n(\bot) = \bigcup_{n \in \omega} \Gamma^n(\emptyset)$;
- the goal: $\forall n \in \mathbb{N}. \forall (\sigma, \sigma') \in \Gamma^n(\emptyset). \langle w, \sigma \rangle \to \sigma'$

Equivalence with Operational Semantics

While Loops

- $w = \textbf{while } b \textbf{ do } c$;
- $\mathcal{C}[\![w]\!] := \mathit{fix}(\Gamma) = \bigsqcup_{n\in\omega} \Gamma^n(\bot) = \bigcup_{n\in\omega} \Gamma^n(\emptyset)$;
- the goal: $\forall n \in \mathbb{N}. \forall (\sigma, \sigma') \in \Gamma^n(\emptyset). \langle w, \sigma\rangle \to \sigma'$
- the approach: an extra induction on n that

$$\forall \sigma, \sigma' \in \Sigma. \left((\sigma, \sigma') \in \mathcal{C}[\![c]\!] \Rightarrow \langle c, \sigma\rangle \to \sigma'\right)$$

implies

$$\forall (\sigma, \sigma') \in \Gamma^n(\emptyset). \langle w, \sigma\rangle \to \sigma'$$

Equivalence with Operational Semantics

Base Step: $n = 0$

- $w = \textbf{while } b \textbf{ do } c$;
- $\mathcal{C}[\![w]\!] := \mathit{fix}(\Gamma) = \bigsqcup_{n \in \omega} \Gamma^n(\bot) = \bigcup_{n \in \omega} \Gamma^n(\emptyset)$;
- $\Gamma^0(\emptyset) = \emptyset$
- $\forall (\sigma, \sigma') \in \Gamma^0(\emptyset). \langle w, \sigma \rangle \rightarrow \sigma'$

Equivalence with Operational Semantics

Inductive Step: $n \geq 1$

- $w = \textbf{while } b \textbf{ do } c$;
-

$$\begin{aligned}\Gamma^{n+1}(\emptyset) := \{(\sigma, \sigma) \mid \mathcal{B}[\![b]\!](\sigma) = \textbf{false}\} \cup \\ \{(\sigma, \sigma') \mid \mathcal{B}[\![b]\!](\sigma) = \textbf{true} \text{ and } (\sigma, \sigma') \in \Gamma^{n}(\emptyset) \circ \mathcal{C}[\![c]\!]\}\end{aligned}$$

- the goal: to prove $\forall (\sigma, \sigma') \in \Gamma^{n+1}(\emptyset). \langle w, \sigma \rangle \to \sigma'$ under the main induction hypothesis for $\mathcal{C}[\![c]\!]$.
- proof: from the rules

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

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

Equivalence with Operational Semantics

While Loops

- w = **while** b **do** c, $\mathcal{C}[\![w]\!] = \bigcup_{n \in \omega} \Gamma^n(\emptyset)$
- $\forall(\sigma, \sigma').\,((\sigma, \sigma') \in \mathcal{C}[\![c]\!] \Rightarrow \langle c, \sigma\rangle \to \sigma')$ implies

$$\forall n \in \mathbb{N}.\forall(\sigma, \sigma') \in \Gamma^n(\emptyset).\langle w, \sigma\rangle \to \sigma'$$

- $\forall(\sigma, \sigma').\,((\sigma, \sigma') \in \mathcal{C}[\![c]\!] \Rightarrow \langle c, \sigma\rangle \to \sigma')$ implies

$$\forall(\sigma, \sigma').\,((\sigma, \sigma') \in \mathcal{C}[\![w]\!] \Rightarrow \langle w, \sigma\rangle \to \sigma')$$

Equivalence with Operational Semantics

What have we proved ?

- $\forall c \in \mathbf{Com}. \forall \sigma, \sigma' \in \Sigma. (\langle c, \sigma \rangle \to \sigma' \Rightarrow (\sigma, \sigma') \in \mathcal{C}[\![c]\!])$
  (rule induction)
- $\forall c \in \mathbf{Com}. \forall \sigma, \sigma' \in \Sigma. ((\sigma, \sigma') \in \mathcal{C}[\![c]\!] \Rightarrow \langle c, \sigma \rangle \to \sigma')$
  (structural induction)
- $\forall c \in \mathbf{Com}. \forall \sigma, \sigma' \in \Sigma. (\langle c, \sigma \rangle \to \sigma' \Leftrightarrow (\sigma, \sigma') \in \mathcal{C}[\![c]\!])$

Equivalence with Operational Semantics

Impact

- the equivalence between the semantics
- the legitimacy of least fixed points

Knaster-Tarski's Fixed-Point Theorem

Textbook, Page 74 – 75

Knaster-Tarski's Fixed-Point Theorem

- an alternative fixed-point theorem
- do not require: complete partial order
- do not require: continuity prerequisite
- require: (greatest) lower bound

Knaster-Tarski's Fixed-Point Theorem

Recall: Upper Bounds

- $(D, \sqsubseteq)$: a partial order
- X: a subset of D (i.e., $X \subseteq D$)
- y: an element in D

Then y is an upper bound for X if it holds that $\forall x \in X . x \sqsubseteq y$.

Knaster-Tarski's Fixed-Point Theorem

Lower Bounds

- $(D, \sqsubseteq)$: a partial order
- X: a subset of D (i.e., $X \subseteq D$)
- y: an element in D

Then $y \in D$ is a lower bound for X if it holds that $\forall x \in X. y \sqsubseteq x$.

Greatest Lower Bounds

We say that y is a (unique) greatest lower bound for X if we have:

- y is a lower bound;
- for all lower bounds z for X, it holds that $z \sqsubseteq y$.
- notation: $\bigsqcap X$ for y

Knaster-Tarski's Fixed-Point Theorem

Complete Lattices

- $(D, \sqsubseteq)$: a partial order

$(D, \sqsubseteq)$ is a complete lattice if $\bigsqcap X$ exists for every $X \subseteq D$.

Some Special Elements

- the least element: $\bot := \bigsqcap D$ such that $\forall x \in D, \bot \sqsubseteq x$
- the greatest element: $\top := \bigsqcap \emptyset$ such that $\forall x \in D, x \sqsubseteq \top$

Knaster-Tarski's Fixed-Point Theorem

Complete Lattices

- $(D, \sqsubseteq)$: a partial order

$(D, \sqsubseteq)$ is a complete lattice if $\bigsqcap X$ exists for every $X \subseteq D$.

Exercise

Every $X \subseteq D$ has a least upper bound.

- $Y := \{y \in D \mid \forall x \in X. x \sqsubseteq y\}$
- $\bigsqcup X = \bigsqcap Y$

Knaster-Tarski's Fixed-Point Theorem

Terminology

- least upper bound: supremum
- greatest lower bound: infimum

Knaster-Tarski's Fixed-Point Theorem

Examples

- $(\mathbb{N}, \leq)$ is not a complete lattice.
- $([0,1], \leq)$ is a complete lattice.
- $(2^D, \subseteq)$ is a complete lattice for any set D.

Knaster-Tarski's Fixed-Point Theorem

Notation

- $(D, \sqsubseteq)$: a complete lattice
- $f : D \to D$: a monotonic function
- $Z := \{d \in D \mid f(d) = d\}$

Then:

- the least fixed point $lfp(f)$ is the least element of Z if it exists:

$$lfp(f) \in Z \ \& \ \forall d \in Z . lfp(f) \sqsubseteq d$$

- the greatest fixed point $gfp(f)$ is the greatest element of Z if it exists:

$$gfp(f) \in Z \ \& \ \forall d \in Z . d \sqsubseteq gfp(f)$$

Knaster-Tarski's Fixed-Point Theorem

Theorem Statement

- $(D, \sqsubseteq)$: a complete lattice
- $f : D \to D$: a monotonic function (not necessarily continuous)

Then:

- $\mathit{lfp}(f) = \bigsqcap\{d \in D \mid f(d) \sqsubseteq d\}$;
- $\mathit{gfp}(f) = \bigsqcup\{d \in D \mid d \sqsubseteq f(d)\}$.

Knaster-Tarski's Fixed-Point Theorem

Theorem Statement

- $lfp(f) = \bigsqcap\{d \in D \mid f(d) \sqsubseteq d\}$;

Proof

- $d' := \bigsqcap\{d \in D \mid f(d) \sqsubseteq d\}$.
- $f(d') \sqsubseteq f(d) \sqsubseteq d$ for all $d \in D$ such that $f(d) \sqsubseteq d$.
- $f(d') \sqsubseteq d'$ and $f(d') \in \{d \in D \mid f(d) \sqsubseteq d\}$
- $f(d') = d'$

Tarski's Fixed-Point Theorem

Theorem

- $gfp(f) = \bigsqcup\{d \in D \mid d \sqsubseteq f(d)\}$.

Proof

- $d'' := \bigsqcup\{d \in D \mid d \sqsubseteq f(d)\}$.
- $d \sqsubseteq f(d) \sqsubseteq f(d'')$ for all $d \in D$ such that $d \sqsubseteq f(d)$.
- $d'' \sqsubseteq f(d'')$ and $f(d'') \in \{d \in D \mid d \sqsubseteq f(d)\}$
- $f(d'') = d''$

Tarski's Fixed-Point Theorem

Question

Can we replace complete partial orders by complete lattices in our denotational semantics?

The Bottom Element $\perp$

Textbook, Page 72 – 73

The Bottom Element $\bot$

The CPO $\Sigma_\bot$

- $\bot$: an element for non-termination
- $\Sigma_\bot := \Sigma \cup \{\bot\}$
- $\sqsubseteq := \{(\bot, \sigma) \mid \sigma \in \Sigma\} \cup \{(d, d) \mid d \in \Sigma_\bot\}$

Exercise

Verify that $(\Sigma_\bot, \sqsubseteq)$ is a cpo with bottom.

The Bottom Element $\perp$

The CPO $\Sigma_\perp$

- $\perp$: an element for non-termination
- $\Sigma_\perp := \Sigma \cup \{\perp\}$
- $\sqsubseteq := \{(\perp, \sigma) \mid \sigma \in \Sigma\} \cup \{(d, d) \mid d \in \Sigma_\perp\}$

1-1 correspondence

- $F : \Sigma \rightharpoonup \Sigma$: a partial function
- $F' : \Sigma \to \Sigma_\perp$: $F'(\sigma) = \perp$ whenever $F(\sigma)$ is undefined.
- the partial order: $F' \sqsubseteq G'$ iff $F'(\sigma) \sqsubseteq G'(\sigma)$ for all $\sigma \in \Sigma$.
- a property: $F \subseteq G$ iff $F' \sqsubseteq G'$
- an exercise: $((\Sigma \to \Sigma_\perp), \sqsubseteq)$ is a cpo with bottom.

Summary

- equivalence with operational semantics
- Naster-Tarski's Fixed-Point Theorem
- the bottom element

Exercise 5

Problem 1

- D, E, F: cpo's (with their implicit ordering relations)
- $f : D \to E$ and $g : E \to F$: continuous functions

Prove that the function $g \circ f : D \to F$ is continuous.

Exercise 5

Problem 2

Let $(D, \sqsubseteq_D)$ and $(E, \sqsubseteq_E)$ be complete partial orders (cpo's) with bottom elements $\perp_D, \perp_E$ respectively. Consider the partial order $(D \times E, \sqsubseteq)$ defined through **lexicographic ordering**, i.e., for all $(d, e), (d', e') \in D \times E$ we have $(d, e) \sqsubseteq (d', e')$ iff it holds that either $d \sqsubseteq_D d'$ and $d \neq d'$, or $d = d'$ and $e \sqsubseteq_E e'$. Determine whether $(D \times E, \sqsubseteq)$ is always a cpo with bottom or not, and **prove/disprove** your answer. You **don't need** to prove that $(D \times E, \sqsubseteq)$ is a partial order.

Note: Please write out the **main points** of the proofs as **complete** as possible.

The axiomatic semantics of IMP

Topics

Axiomatic Semantics

- logical specifications for programs
- partial correctness assertions
- proof rules for partial correctness assertions

Axiomatic Semantics: An Intuition

Textbook, Page 77 – 78

Axiomatic Semantics: An Intuition

A Simple Example

Consider the command (program) c as follows:

$$\begin{array}{l} S := 0\,; \\ N := 1\,; \\ \textbf{while}\ \ \neg(N = 101)\ \ \textbf{do}\ \ (\,S := S + N\,;\ \ N := N + 1\,) \end{array}$$

Axiomatic Semantics: An Intuition

A Simple Example

Consider the command (program) c as follows:

$$
\begin{array}{l}
S := 0\,; \\
N := 1\,; \\
\textbf{while}\ \ \neg(N = 101)\ \ \textbf{do}\ \ (\,S := S + N\,;\ \ N := N + 1\,)
\end{array}
$$

Our Goal

For any $\sigma, \sigma' \in \Sigma$, $\langle c, \sigma \rangle \to \sigma'$ implies $\sigma'(S) = \sum_{k=1}^{100} k = 5050$.

Axiomatic Semantics: An Intuition

The First Part

$$\{\textbf{true}\}\ \ S := 0\,;\ \ N := 1\ \ \{S = 0 \wedge N = 1\}$$

Axiomatic Semantics: An Intuition

The Loop Body

$$\{S = \sum_{k=1}^{N-1} k \wedge \neg(N = 101)\}$$

$$S := S + N\,; \quad N := N + 1$$

$$\{S = \sum_{k=1}^{N-1} k\}$$

The Whole While-Loop

$$\{S = \sum_{k=1}^{N-1} k\}$$

$$\textbf{while} \ \ \neg(N = 101) \ \ \textbf{do} \ \ (S := S + N\,; \quad N := N + 1)$$

$$\{S = \sum_{k=1}^{N-1} k \wedge N = 101\}$$

Axiomatic Semantics: An Intuition

Putting Together

$$\{\mathbf{true}\}\ \ S := 0\,;\ \ N := 1\ \ \{S = 0 \land N = 1\}$$

$$\{S = \textstyle\sum_{k=1}^{N-1} k\}$$

$$\textbf{while}\ \ \neg(N = 101)\ \ \textbf{do}\ \ (S := S + N\,;\ \ N := N + 1)$$

$$\{S = \textstyle\sum_{k=1}^{N-1} k \land N = 101\}$$

- $(S = \sum_{k=1}^{N-1} k \land N = 101) \Rightarrow S = \sum_{k=1}^{100} k = 5050$

Axiomatic Semantics: An Intuition

The Logical Layout

- c: a command
- A, B: logical formulas

Then the assertion $\{A\}c\{B\}$ means that

for all states σ that satisfy A, if $\langle c, \sigma\rangle \to \sigma'$ then σ' satisfies B.

Axiomatic Semantics: An Overview

Textbook, Page 78 – 80

Axiomatic Semantics

Partial Correctness Assertions

- A, B: logical formulas
- $\sigma \models A$: σ satisfies A
- c: a command

A partial correctness assertion is of the form $\{A\}c\{B\}$, meaning

$$\forall \sigma, \sigma' \in \Sigma. ((\sigma \models A \wedge \langle c, \sigma\rangle \to \sigma') \Rightarrow \sigma' \models B) \quad .$$

Terminology

- A: precondition
- B: postcondition

Axiomatic Semantics

Partial Correctness Assertions

- A, B: logical formulas
- $\sigma \models A$: σ satisfies A
- c: a command

A partial correctness assertion is of the form $\{A\}c\{B\}$, meaning

$$\forall \sigma, \sigma' \in \Sigma. ((\sigma \models A \land \langle c, \sigma \rangle \to \sigma') \Rightarrow \sigma' \models B) \quad .$$

The Core of the Axiomatic Semantics

- logical properties for input-output relationships
- no guarantee of termination

Axiomatic Semantics

An Example

- c := **while true do skip**
- $\{\textbf{true}\}c\{\textbf{false}\}$

Question

Does $\{\textbf{true}\}c\{\textbf{false}\}$ hold?

Axiomatic Semantics

Total Correctness Assertions

- A, B: logical formulas
- c: a command

Then $[A]c[B]$ means that

- $\forall \sigma, \sigma' \in \Sigma. ((\sigma \models A \wedge \langle c, \sigma \rangle \to \sigma' \Rightarrow \sigma') \models B)$,
- $\forall \sigma \in \Sigma. [\sigma \models A \Rightarrow \exists \sigma' \in \Sigma. (\langle c, \sigma \rangle \to \sigma')]$.

Observation

total correctness $=$ termination $+$ partial correctness

Axiomatic Semantics

The Bottom Element

- $\bot$: the fresh element for non-termination.
- $\mathcal{C}[\![c]\!](\sigma) := \bot$ if c does not terminate on the initial state σ.
- $\mathcal{C}[\![c]\!](\bot) := \bot$.
- $\bot \models A$ for all logical formulas A.

Axiomatic Semantics

Definition with the Bottom Element

A partial correctness assertion $\{A\}c\{B\}$ means equivalently that

$$\forall \sigma \in \Sigma . (\sigma \models A \Rightarrow \mathcal{C}[\![c]\!](\sigma) \models B) \quad .$$

Axiomatic Semantics

The Central Question

How to build the axiomatic semantics (i.e. $\{A\}c\{B\}$) ?

Axiomatic Semantics

The Road Map

- a formal language for logical formulas
- a collection of rules for partial correctness assertions

An Assertion Language **Assn**

Textbook, Page 80 – 86

An Assertion Language **Assn**

Informal Description

- first-order logical formulas
- satisfaction defined over states

An Assertion Language **Assn**

Example: Primality

- $\mathrm{Prime} := X \geq 0 \wedge \neg(\exists i. \exists j. (i \geq 2 \wedge j \geq 2 \wedge X = i \times j))$
- X: a location (program variable)
- i, j: integer variables
- $\sigma \models \mathrm{Prime}$ iff $\sigma(X)$ is a prime number.

Observation

- locations, arithmetic expressions, propositional logical operators
- integer variables
- universal/existential quantification

An Assertion Language **Assn**

Extended Arithmetic Expressions **Aexpv**

$$a ::= n \mid X \mid i \mid a_0 + a_1 \mid a_0 - a_1 \mid a_0 \times a_1$$

- n: an integer
- X: a location
- i: an integer variable (from **Intvar**)

An Assertion Language **Assn**

Examples

- $X + Y - 3$
- $(i \times j) + k$
- $X + (i \times Y) + 5 - (4 \times j)$

Integer Variables

Why do we include integer variables?

- more expressibility for organizing logical properties
- more ability for representing unknown initial values

An Assertion Language **Assn**

Extended Boolean Assertions **Assn**

$$\begin{array}{rcl} A & ::= & \textbf{true} \mid \textbf{false} \mid a_0 = a_1 \mid a_0 \leq a_1 \mid \\ & & A_0 \wedge A_1 \mid A_0 \vee A_1 \mid \neg A \mid A_0 \Rightarrow A_1 \mid \\ & & \forall i.A \mid \exists i.A \end{array}$$

- a_0, a_1: extended arithmetic expressions from **Aexpv**
- i: an integer variable from **Intvar**
- $\wedge, \vee, \neg$: logical connectives from propositional logic
- $\forall, \exists$: quantifiers from first-order logic

An Assertion Language **Assn**

Extended Boolean Assertions **Assn**

$$\begin{array}{rcl} A & ::= & \mathbf{true} \mid \mathbf{false} \mid a_0 = a_1 \mid a_0 \leq a_1 \mid \\ & & A_0 \wedge A_1 \mid A_0 \vee A_1 \mid \neg A \mid A_0 \Rightarrow A_1 \mid \\ & & \forall i.A \mid \exists i.A \end{array}$$

Satisfaction Relation $\models$: An Intuition

A state σ satisfies an assertion $A \in$ **Assn** (written as $\sigma \models A$) if A is true when all locations X in A is replaced by $\sigma(X)$.

An Assertion Language **Assn**

Satisfaction Relation $\models$: Main Issues

- quantifiers
- integer variables

Important Points

- free/bound variables
- substitution
- interpretations

An Assertion Language **Assn**

Free and Bound Variables

- free variables: integer variables not associated with quantifiers
- bound variables: integer variables associated with quantifiers
- notation: $FV(-)$

Free and Bound Variables: Examples

- $\exists i.(k = i \times l)$;
- $(i + 100 \leq 77) \wedge \forall i.(j + 1 = i + 3)$

Free and Bound Variables

Definition through Well-Founded Recursion

All integer variables in extended arithmetic expressions are free:

- $FV(n) = FV(X) := \emptyset$;
- $FV(i) := \{i\}$;
- $FV(a_0 + a_1) := FV(a_0) \cup FV(a_1)$;
- $FV(a_0 - a_1) := FV(a_0) \cup FV(a_1)$;
- $FV(a_0 \times a_1) := FV(a_0) \cup FV(a_1)$.

Free and Bound Variables

Definition through Well-Founded Recursion

Quantified integer variables are removed from free variables:

- $FV(\mathbf{true}) = FV(\mathbf{false}) := \emptyset$;
- $FV(a_0 = a_1) = FV(a_0 \leq a_1) := FV(a_0) \cup FV(a_1)$;
- $FV(A_0 \bowtie A_1) = FV(A_0) \cup FV(A_1)$ for $\bowtie \in \{\wedge, \vee, \Rightarrow\}$;
- $FV(\neg A) = FV(A)$;
- $FV(\forall i.A) = FV(\exists i.A) = FV(A) \setminus \{i\}$.

Free and Bound Variables

Definitions

- A: an assertion from **Assn**
- i: an integer variable that appears in A

Then:

- i is free in A if $i \in FV(A)$.
- i is bound in A if $i \notin FV(A)$.
- A is closed if $FV(A) = \emptyset$.

Free and Bound Variables

Examples

- $FV(i = 1) = \{i\}$;
- $FV(\forall i.\,(i \times i \geq 0)) = \emptyset$;
- $FV(i = 1 \vee \forall i.\,(i \times i \geq 0)) = \{i\}$;

Substitution

Informal Description

- $a \in$ **Aexp**: an arithmetic expression without integer variables
- i: an integer variable
- A: an assertion such that $i \in FV(A)$

Then $A[a/i]$ is the assertion obtained by substituting all free occurrences of i in A by a.

Substitution

Definition: Extended Arithmetic Expressions

- $n\,[a/i] := n$;
- $X\,[a/i] := X$;
- $j\,[a/i] := j$ if $j \neq i$;
- $j\,[a/i] := a$ if $j = i$;
- $(a_0 \bowtie a_1)\,[a/i] := a_0\,[a/i] \bowtie a_1\,[a/i]$ for $\bowtie \in \{+, -, \times\}$;

Substitution

Definition: Extended Boolean Assertions **Assn**

- $\mathbf{true}\,[a/i] := \mathbf{true}$;
- $\mathbf{false}\,[a/i] := \mathbf{false}$;
- $(a_0 = a_1)\,[a/i] := a_0\,[a/i] = a_1\,[a/i]$;
- $(a_0 \leq a_1)\,[a/i] := a_0\,[a/i] \leq a_1\,[a/i]$;
- $(A_0 \bowtie A_1)\,[a/i] := A_0\,[a/i] \bowtie A_1\,[a/i]$ for $\bowtie \in \{\wedge, \vee, \Rightarrow\}$;
- $(\neg A)\,[a/i] := \neg(A\,[a/i])$;

Substitution

Definition: Extended Boolean Assertions Assn

Universal Quantification:

- $(\forall j.A)\,[a/i] := \forall j.(A\,[a/i])$ if $j \neq i$;
- $(\forall j.A)\,[a/i] := \forall j.A$ if $j = i$;

Existential Quantification:

- $(\exists j.A)\,[a/i] := \exists j.(A\,[a/i])$ if $j \neq i$;
- $(\exists j.A)\,[a/i] := \exists j.A$ if $j = i$;

Substitution

Examples

- $(\exists j. i = j + 1)\,[X/i] = \exists j. X = j + 1$;
- $(\exists j. i = j + 1)\,[X/j] = \exists j. i = j + 1$;
- $(\exists j. i = j + 1)\,[X + j/i] = ?$;

Interpretation

Definition

- An interpretation is a function $I : \textbf{Intvar} \to \mathbb{Z}$ which assigns an integer to each integer variable.
- An interpretation instantiates every free integer variable.

Substitution

$$(I\,[n/i])(j) := \begin{cases} n & \text{if } j = i \\ I(j) & \text{otherwise} \end{cases}$$

Semantics of Assertions **Assn**

Definition over Extended Arithmetic Expressions

- I: an interpretation
- σ: a state

Then we have:

- $\mathcal{A}v[\![n]\!](I,\sigma) := n$;
- $\mathcal{A}v[\![X]\!](I,\sigma) := \sigma(X)$;
- $\mathcal{A}v[\![i]\!](I,\sigma) := I(i)$;
- $\mathcal{A}v[\![a_0 \bowtie a_1]\!](I,\sigma) := \mathcal{A}v[\![a_0]\!](I,\sigma) \bowtie \mathcal{A}v[\![a_1]\!](I,\sigma)$ for $\bowtie \in \{+,-,\times\}$;

Exercise

For all (unextended) arithmetic expressions $a \in$ **Aexp**, it holds that

$$\forall \sigma, I.\, (\mathcal{A}[\![a]\!](\sigma) = \mathcal{A}v[\![a]\!](I,\sigma)) \quad .$$

Semantics of Assertions **Assn**

The Satisfaction Relation $\models$

- σ: a state
- I: an interpretation
- A: an assertion from **Assn**

Defining $\sigma \models^I A$ ("σ satisfies A in I"):

- it always holds that $\sigma \models^I$ **true**;
- it always does not hold that $\sigma \models^I$ **false**;

Semantics of Assertions **Assn**

The Satisfaction Relation $\models$

- σ: a state
- I: an interpretation
- A: an assertion from **Assn**

Defining $\sigma \models^I A$ ("σ satisfies A in I"):

- $\sigma \models^I (a_0 = a_1)$ iff $\mathcal{A}v[\![a_0]\!](I, \sigma) = \mathcal{A}v[\![a_1]\!](I, \sigma)$;
- $\sigma \models^I (a_0 \leq a_1)$ iff $\mathcal{A}v[\![a_0]\!](I, \sigma) \leq \mathcal{A}v[\![a_1]\!](I, \sigma)$;

Semantics of Assertions from **Assn**

The Satisfaction Relation $\models$

- σ: a state
- I: an interpretation
- A: an assertion from **Assn**

Defining $\sigma \models^I A$ ("σ satisfies A in I"):

- $\sigma \models^I (A \wedge B)$ iff $\sigma \models^I A$ and $\sigma \models^I B$;
- $\sigma \models^I (A \vee B)$ iff $\sigma \models^I A$ or $\sigma \models^I B$;
- $\sigma \models^I \neg A$ iff (not $\sigma \models^I A$);
- $\sigma \models^I (A \Rightarrow B)$ iff (not $\sigma \models^I A$) or $\sigma \models^I B$;

Semantics of Assertions from **Assn**

The Satisfaction Relation $\models$

- σ: a state
- I: an interpretation
- A: an assertion from **Assn**

Defining $\sigma \models^I A$ ("σ satisfies A in I"):

- $\sigma \models^I \forall i.A$ iff for all integers n, $\sigma \models^{I[n/i]} A$;
- $\sigma \models^I \exists i.A$ iff there exists an integer n such that $\sigma \models^{I[n/i]} A$;

Semantics of Assertions from **Assn**

The Satisfaction Relation $\models$

- σ: a state
- I: an interpretation
- A: an assertion from **Assn**

Defining $\sigma \models^I A$ ("σ satisfies A in I"):

- $\bot \models^I A$ for all assertions $A \in$ **Assn**.

Notation

"not $\sigma \models^I A$" by "$\sigma \not\models^I A$"

Semantics of Assertions from **Assn**

Exercise

For all (unextended) boolean expressions $b \in \mathbf{Bexp}$, states $\sigma \in \Sigma$ and interpretations I, it holds that

- $\mathcal{B}[\![b]\!](\sigma) = \mathbf{true}$ iff $\sigma \models^I b$, and
- $\mathcal{B}[\![b]\!](\sigma) = \mathbf{false}$ iff $\sigma \not\models^I b$.

Semantics of Assertions from **Assn**

Exercise

For any extended arithmetic expression $a \in$ **Aexpv**, interpretation I and state σ, it holds that

$$\mathcal{A}v[\![a]\!](I\,[n/i], \sigma) = \mathcal{A}v[\![a\,[n/i]]\!](I, \sigma)$$

for all integers n and integer variables i.

Exercise

- $\sigma \models^I \forall i.A$ iff $\sigma \models^I A\,[n/i]$ for all integers n.
- $\sigma \models^I \exists i.A$ iff $\sigma \models^I A\,[n/i]$ for some integer n.
- solution: prove by induction on the structure of A that $\sigma \models^{I[n/i]} A$ iff $\sigma \models^I A\,[n/i]$

Semantics of Assertions from **Assn**

Extension of Assertions

- A: an assertion in **Assn**
- I: an interpretation
- $A^I := \{\sigma \in \Sigma_\perp \mid \sigma \models^I A\}$.

Validity for **Assn**

- A is valid: $\models A$ iff for all interpretations I and all states σ, $\sigma \models^I A$.

Partial Correctness Assertions

Textbook, Page 87 – 89

Partial Correctness Assertions

Definition

A partial correctness assertion is of the form

$$\{A\}c\{B\}$$

where $A, B \in \mathbf{Assn}$ and $c \in \mathbf{Com}$.

Satisfaction Relation $\models$

- I: an interpretation
- σ: an element in $\Sigma_\perp$

We define that $\sigma \models^I \{A\}c\{B\}$ iff $(\sigma \models^I A \Rightarrow \mathcal{C}[\![c]\!](\sigma) \models^I B)$.

Partial Correctness Assertions

Satisfaction Relation $\models$

- I: an interpretation
- σ: an element in $\Sigma_\perp$

We define that $\sigma \models^I \{A\}c\{B\}$ iff $(\sigma \models^I A \Rightarrow \mathcal{C}[\![c]\!](\sigma) \models^I B)$.

Validity

- Define that $\models^I \{A\}c\{B\}$ iff $\forall \sigma \in \Sigma_\perp . \sigma \models^I \{A\}c\{B\}$.
- Define that $\models \{A\}c\{B\}$ iff $\forall I . \models^I \{A\}c\{B\}$.
- The partial correctness assertion $\{A\}c\{B\}$ is valid if $\models \{A\}c\{B\}$.

Partial Correctness Assertions

Validity

We have $\models \{A\}c\{B\}$ *holds iff for all interpretations* I *and all states* σ*, if* σ *satisfies* A *in* I *and the execution of* c *terminates in* σ' *from* σ*, then* σ' *satisfies* B *in* I*.*

Recall: Validity for **Assn**

- A is valid: $\models A$ iff for all interpretations I and all states σ, $\sigma \models^I A$.

Partial Correctness Assertions

Validity: Examples

- $\{i \leq X\}X := X + 1\{i \leq X\}$ is valid.
- $\{i \leq X\}X := X - 1\{i \leq X\}$ is not valid.

Validity and Extension Sets

- $\models A \Rightarrow B$ iff $A^I \subseteq B^I$ for all interpretations I.
- $\models \{A\}c\{B\}$ iff $\mathcal{C}[\![c]\!](A^I) \subseteq B^I$ for all interpretations I.

Proof Rules for Partial Correctness Assertions

Textbook, Page 89 – 93

Proof Rules

Motivation

- Manual validation of the validity $\models \{A\}c\{B\}$ is tedious.
- Rules for deriving the validity $\models \{A\}c\{B\}$ makes the task easier.

Proof Rules

Hoare Rules

- rules for each type of commands
- derivation trees built from rule instances
- correctness for each rule

Hoare Rules

Skip

$$\frac{}{\{A\}\mathbf{skip}\{A\}}$$

Hoare Rules

Assignment

$$\frac{}{\{B[a/X]\}X := a\{B\}}$$

Hoare Rules

Sequencing

$$\frac{\{A\}c_0\{C\}\ ,\ \{C\}c_1\{B\}}{\{A\}c_0;c_1\{B\}}$$

Hoare Rules

Conditional Branch

$$\frac{\{A \wedge b\}c_0\{B\} \ , \ \{A \wedge \neg b\}c_1\{B\}}{\{A\}\textbf{if } b \textbf{ then } c_0 \textbf{ else } c_1\{B\}}$$

Hoare Rules

While Loop

$$\frac{\{A \land b\}c\{A\}}{\{A\}\textbf{while } b \textbf{ do } c\{A \land \neg b\}}$$

- A: the loop invariant

Hoare Rules

Consequence

$$\frac{\models A \Rightarrow A' \;,\; \{A'\}c\{B'\} \;,\; \models B' \Rightarrow B}{\{A\}c\{B\}}$$

Hoare Rules

The Proof System

- proofs as derivation trees
- theorems as conclusions
- notation for theorems: $\vdash \{A\}c\{B\}$

Summary

- extended arithmetic and boolean assertions
- partial correctness assertions
- a proof system from Hoare rules

Topics

Axiomatic Semantics

- soundness of Hoare rules
- examples for using Hoare rules
- a start with completeness of Hoare rules

Soundness of Hoare Rules

Textbook, Page 91 – 93

Soundness and Completeness of Hoare Rules

Soundness

$\vdash \{A\}c\{B\}$ implies $\models \{A\}c\{B\}$.

Completeness

$\models \{A\}c\{B\}$ implies $\vdash \{A\}c\{B\}$.

Soundness of Hoare Rules

Soundness

$\vdash \{A\}c\{B\}$ implies $\models \{A\}c\{B\}$.

Proof: Rule Induction

Prove that every rule is sound, i.e., the conclusion always holds if all the premises hold.

Soundness of Hoare Rules

Properties of Substitution

- a, a_0: extended arithmetic expressions in **Aexpv**
- X: a location (program variable)

Then for all interpretations I and states σ,

$$\mathcal{A}v[\![a_0\,[a/X]]\!](I,\sigma) = \mathcal{A}v[\![a_0]\!](I,\sigma\,[\mathcal{A}v[\![a]\!](I,\sigma)/X])$$

Proof

By structural induction on a_0.

Soundness of Hoare Rules

Properties of Substitution

- B: an extended boolean assertion from **Assn**
- X: a location (identifier)
- a: an arithmetic expression from **Aexp**

Then for all interpretations I and states σ, we have

$$\sigma \models^I B[a/X] \text{ iff } \sigma[\mathcal{A}[\![a]\!](\sigma)/X] \models^I B$$

Proof

By structural induction on B.

Soundness of Hoare Rules

Soundness

$\vdash \{A\}c\{B\}$ implies $\models \{A\}c\{B\}$.

Proof: Rule Induction

Prove that for every rule instance, if all the extended boolean assertions and partial correctness assertions in its premises are valid, then so is its conclusion.

Soundness of Hoare Rules

Skip

$$\frac{}{\{A\}\mathbf{skip}\{A\}}$$

- $\langle \mathbf{skip}, \sigma \rangle \to \sigma$
- $\sigma \models^I A \Rightarrow \sigma \models^I A$
- $\models \{A\}\mathbf{skip}\{A\}$

Soundness of Hoare Rules

Assignment

$$\frac{}{\{B\,[a/X]\}X := a\{B\}}$$

- $\langle a, \sigma\rangle \rightarrow n$
- $\langle X := a, \sigma\rangle \rightarrow \sigma\,[n/X]$
- $\sigma \models^{I} B\,[a/X]$ iff $\sigma\,[n/X] \models^{I} B$
- $\models \{B\,[a/X]\}X := a\{B\}$

Hoare Rules

Sequencing

$$\frac{\{A\}c_0\{C\}\ ,\ \{C\}c_1\{B\}}{\{A\}c_0;c_1\{B\}}$$

- $\langle c_0; c_1, \sigma\rangle \to \sigma'$
- $\langle c_0, \sigma\rangle \to \sigma''$, $\langle c_1, \sigma''\rangle \to \sigma'$
- from $\models \{A\}c_0\{C\}$: $\sigma \models^I A \Rightarrow \sigma'' \models^I C$
- from $\models \{C\}c_1\{B\}$: $\sigma'' \models^I C \Rightarrow \sigma' \models^I B$
- $\sigma \models^I A \Rightarrow \sigma' \models^I B$
- $\models \{A\}c_0; c_1\{B\}$

Hoare Rules

Conditional Branch

$$\frac{\{A \wedge b\} c_0 \{B\}\ ,\ \{A \wedge \neg b\} c_1 \{B\}}{\{A\} \textbf{if } b \textbf{ then } c_0 \textbf{ else } c_1 \{B\}}$$

- $\sigma \models^I (A \wedge b) \Rightarrow \mathcal{C}[\![c_0]\!](\sigma) \models^I B$
- $\sigma \models^I (A \wedge \neg b) \Rightarrow \mathcal{C}[\![c_1]\!](\sigma) \models^I B$
- $\sigma \models^I A \Rightarrow \mathcal{C}[\![\textbf{if } b \textbf{ then } c_0 \textbf{ else } c_1]\!](\sigma) \models^I B$
- $\models \{A\} \textbf{if } b \textbf{ then } c_0 \textbf{ else } c_1 \{B\}$

Hoare Rules

While Loop

$$\frac{\{A \wedge b\}c\{A\}}{\{A\}\textbf{while } b \textbf{ do } c\{A \wedge \neg b\}}$$

- $w = \textbf{while } b \textbf{ do } c$;
- $\langle w, \sigma\rangle \to \sigma'$, $\sigma \models^I A$
- Case 1:

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

- $\sigma \models^I \neg b$
- $\sigma \models^I A \wedge \neg b$;

Hoare Rules

While Loop

$$\frac{\{A \wedge b\}c\{A\}}{\{A\}\textbf{while } b \textbf{ do } c\{A \wedge \neg b\}}$$

- $w = \textbf{while } b \textbf{ do } c;$
- $\langle w, \sigma \rangle \to \sigma'$, $\sigma \models^I A$
- Case 2 (a nested induction on derivation trees):

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

- $\sigma'' \models^I A$ from the main induction hypothesis
- $\sigma' \models^I A \wedge \neg b$ from the nested induction hypothesis

Hoare Rules

Consequence

$$\frac{\models A \Rightarrow A' \ , \ \{A'\}c\{B'\} \ , \ \models B' \Rightarrow B}{\{A\}c\{B\}}$$

- $\langle c, \sigma \rangle \to \sigma'$, $\sigma \models^{I} A$
- $\sigma \models^{I} A'$ from $\models A \Rightarrow A'$
- $\sigma' \models^{I} B'$ from $\models \{A'\}c\{B'\}$
- $\sigma' \models^{I} B$ from $\models B' \Rightarrow B$

Soundness of Hoare Rules

Soundness

$\vdash \{A\}c\{B\}$ implies $\models \{A\}c\{B\}$.

Hoare Rules: Examples

Textbook, Page 93 – 96

Hoare Rules: Examples

```
$S := 0$ ;
$N := 1$ ;
while  $\neg(N = 101)$  do
    $S := S + N$ ;
    $N := N + 1$
```

Hoare Rules: Examples

```
$\{\mathbf{true}\}$ implies $\{0 = 0\}$
$S := 0\,;$
$\{S = 0\}$ implies $\{S = 0 \wedge 1 = 1\}$
$N := 1\,;$
$\{S = 0 \wedge N = 1\}$ implies $\{2 \times S = N \cdot (N - 1)\}$
$\mathbf{while}\ \ \neg(N = 101)\ \ \mathbf{do}$
    $\{2 \times S = N \times (N - 1) \wedge \neg(N = 101)\}$ implies
    $\{2 \times (S + N) = N \times (N + 1) \wedge \neg(N = 101)\}$
    $S := S + N\,;$
    $\{2 \times S = (N + 1) \times N \wedge \neg(N = 101)\}$ implies
    $\{2 \times S = (N + 1) \times N\}$
    $N := N + 1$
    $\{2 \times S = N \times (N - 1)\}$
$\{2 \times S = N \times (N - 1) \wedge \neg(\neg N = 101)\}$ implies
$\{S = 5050\}$
```

Hoare Rules: Examples

```
$P := 0$ ;
$C := 1$ ;
while  $C \leq N$  do
    $P := P + M$ ;
    $C := C + 1$
```

Hoare Rules: Examples

$\{1 \leq N\}$
$P := 0\,;\quad \{1 \leq N \land P = 0\}$
$C := 1\,;\quad \{1 \leq N \land P = 0 \land C = 1\}$

$\{P = M \times (C - 1) \land C \leq N + 1\}$
$\textbf{while}\ \ C \leq N\ \ \textbf{do}$
$\quad \{P = M \times (C - 1) \land C \leq N + 1 \land C \leq N\}$
$\quad P := P + M\,;\quad \{P = M \times C \land C \leq N + 1 \land C \leq N\}$
$\quad C := C + 1\ \ \{P = M \times (C - 1) \land C \leq N + 1\}$

$\{P = M \times (C - 1) \land C \leq N + 1 \land \neg(C \leq N)\}$
$\{P = M \times N\}$

Hoare Rules: Examples

```
while ¬(Y = 0) do
    Y := Y − 1 ;
    X := 2 × X
```

Hoare Rules: Examples

$\{i \geq 0 \wedge Y = i \wedge X = 1\}$
$\{X \times 2^{Y} = 2^{i} \wedge Y \geq 0\}$

$\textbf{while} \ \neg(Y = 0) \ \textbf{do}$
$\quad \{X \times 2^{Y} = 2^{i} \wedge Y \geq 0 \wedge \neg(Y = 0)\}$
$\quad Y := Y - 1; \ \{Y \geq 0 \wedge 2 \times X \times 2^{Y} = 2^{i}\}$
$\quad X := 2 \times X \ \{X \cdot 2^{Y} = 2^{i} \wedge Y \geq 0\}$

$\{X \cdot 2^{Y} = 2^{i} \wedge Y \geq 0 \wedge Y = 0\}$
$\{X = 2^{i}\}$

Hoare Rules: Examples

```
while ¬(X ≤ 0) do
    Y := X × Y ;
    X := X − 1
```

Hoare Rules: Examples

$\{X = n \land n \geq 0 \land Y = 1\}$
$\{Y \times X! = n! \land X \geq 0\}$

$\textbf{while}\;\; X > 0 \;\;\textbf{do}$
$\quad \{Y \times X! = n! \land X \geq 0 \land X > 0\}$
$\quad Y := X \times Y\,;$
$\quad \{Y \times X! = n! \cdot X \land X \geq 0 \land X > 0\}$
$\quad X := X - 1$
$\quad \{Y \times X! = n! \land X \geq 0\}$

$\{Y \times X! = n! \land X \geq 0 \land \neg(X > 0)\}$
$\{Y = n!\}$

Hoare Rules: Examples

```
while $\neg(Y = 0)$ do
    (while even($Y$) do $X := X \times X$;  $Y := Y/2$);
    $Z := Z \times X$;
    $Y := Y - 1$
```

Hoare Rules: Examples

```
$\{X = m \land Y = n \land Z = 1 \land n \geq 0\}$
$\{Y \geq 0 \land m^n = Z \times X^Y\}$
while $\neg(Y = 0)$ do
    $\{Y \geq 0 \land m^n = Z \times X^Y \land \neg(Y = 0)\}$
    ( while even($Y$) do
        $\{Y \geq 0 \land m^n = Z \times X^Y \land \neg(Y = 0) \land \mathbf{even}(Y)\}$
        $X := X \times X$;  $\{Y \geq 0 \land m^n = Z \times X^{Y/2} \land \neg(Y = 0) \land \mathbf{even}(Y)\}$
        $Y := Y/2$  $\{Y \geq 0 \land m^n = Z \times X^Y \land \neg(Y = 0)\}$ );
    $\{Y \geq 0 \land m^n = Z \times X^Y \land \neg(Y = 0) \land \neg\mathbf{even}(Y)\}$
    $Z := Z \times X$;  $\{Y \geq 0 \land m^n = Z \times X^{Y-1} \land \neg(Y = 0) \land \neg\mathbf{even}(Y)\}$
    $Y := Y - 1$  $\{Y \geq 0 \land m^n = Z \times X^Y\}$
$\{Y \geq 0 \land m^n = Z \times X^Y \land Y = 0\}$
$\{m^n = Z\}$
```

Hoare Rules: Examples

```
while $\neg(M = N)$ do
   if $M \leq N$ then
      $N := N - M$
   else
      $M := M - N$
```

Hoare Rules: Examples

```
$\{M = m \land N = n \land 1 \leq m \land 1 \leq n\}$
$\{\mathrm{gcd}(M, N) = \mathrm{gcd}(m, n)\}$

while $\neg(M = N)$ do
    $\{\mathrm{gcd}(M, N) = \mathrm{gcd}(m, n) \land \neg(M = N)\}$
    if $M \leq N$ then
        $\{\mathrm{gcd}(M, N) = \mathrm{gcd}(m, n) \land \neg(M = N) \land M \leq N\}$
        $N := N - M$  $\{\mathrm{gcd}(M, N) = \mathrm{gcd}(m, n)\}$
    else
        $\{\mathrm{gcd}(M, N) = \mathrm{gcd}(m, n) \land \neg(M = N) \land \neg(M \leq N)\}$
        $M := M - N$  $\{\mathrm{gcd}(M, N) = \mathrm{gcd}(m, n)\}$
    $\{\mathrm{gcd}(M, N) = \mathrm{gcd}(m, n)\}$

$\{\mathrm{gcd}(M, N) = \mathrm{gcd}(m, n) \land M = N\}$
$\{N = \mathrm{gcd}(m, n)\}$
```

Exercises

Problem

Consider the command c to be

$$Z := X; X := Y; Y := Z\ .$$

with locations X, Y, Z. Prove through Hoare rules that the partial correctness assertion

$$\{X = i \wedge Y = j\}c\{X = j \wedge Y = i\}$$

is valid, where i, j are integer variables.

Exercises

Problem

Let c be the command **while** $X \leq 100$ **do** $X := X + 2$ with location X. Prove through the Hoare rules that

$$\models \{X \leq 100 \wedge (\exists i. X = 2 \times i + 1)\} c \{X = 101\}$$

where i is an integer variable.

Completeness of the Hoare rules

Completeness of Hoare Rules

Completeness of Hoare Rules

Effective Proof Systems

A proof system is effective if there exists an algorithm such that

- upon an input rule instance, then the algorithm outputs “yes”,
- otherwise the algorithm outputs “no” or does not terminate.

Completeness of Hoare Rules

Consequence

$$\frac{\models A \Rightarrow A' \ , \ \{A'\}c\{B'\} \ , \ \models B' \Rightarrow B}{\{A\}c\{B\}}$$

Problem

- How to check $\models A \Rightarrow A'$ and $\models B' \Rightarrow B$?

Completeness of Hoare Rules

Gödel's Incompleteness Theorem

There is no effective proof system for **Assn** such that the theorems coincide with valid assertions in **Assn**.

Corollary

There is no effective proof system for partial correctness assertions such that its theorems are precisely the valid partial correctness assertions.

Proof

$\models B$ iff $\models \{\textbf{true}\}\textbf{skip}\{B\}$.

Completeness of Hoare Rules

Corollary

There is no effective proof system for partial correctness assertions such that its theorems are precisely the valid partial correctness assertions.

Proof (by contradiction)

- $\models \{\mathbf{true}\}c\{\mathbf{false}\}$ iff c diverges (does not terminate) on all states.
- The set $\{c \mid \forall\sigma \in \Sigma.\mathcal{C}[\![c]\!](\sigma) = \bot\}$ is not checkable (or recursively enumerable) (Textbook, Appendix A).

Corollary

The proof system of Hoare rules is not effective.

Completeness of Hoare Rules

Relative Completeness

The Hoare rules are relatively complete if $\models \{A\}c\{B\}$ implies $\vdash \{A\}c\{B\}$ for all parital correctness assertions $\{A\}c\{B\}$.

Theorem

The proof system of Hoare rules is relatively complete.

Completeness of Hoare Rules

Weakest Preconditions

- motivation: $\vdash \{A\} c_0; c_1 \{B\}$
- approach: an extended boolean assertion C such that $\vdash \{A\} c_0 \{C\}$ and $\vdash \{C\} c_1 \{B\}$

Question

Does such C really exist?

Completeness of Hoare Rules

Weakest Preconditions

- c: a command
- B: an extended boolean assertion
- I: an interpretation

Then we define the weakest precondition $wp^I[\![c, B]\!]$ by

$$wp^I[\![c, B]\!] := \{\sigma \in \Sigma_\perp \mid \mathcal{C}[\![c]\!](\sigma) \models^I B\}$$

Weakest Precondition

Weakest Preconditions

- $wp^I[\![c, B]\!] := \{\sigma \in \Sigma_\perp \mid \mathcal{C}[\![c]\!](\sigma) \models^I B\}$
- $\models^I \{A\}c\{B\}$ iff $A^I \subseteq wp^I[\![c, B]\!]$

Completeness of Hoare Rules

Weakest Precondition: Our Goal

For every c, B, there exists $A \in$ **Assn** such that $A^I = wp^I[\![c, B]\!]$ for every interpretation I.

Corollary

- $\sigma \models^I \{A'\}c\{B\}$ iff $\sigma \models^I A' \Rightarrow A$
- $\models \{A'\}c\{B\}$ iff $\models A' \Rightarrow A$

Summary

- soundness of Hoare rules
- examples for Hoare rules
- relative completeness
- weakest preconditions

Topics

- relative completeness of Hoare rules
- a proof for Gödel's Incompleteness Theorem

Relative Completeness of Hoare Rules

Textbook, Page 100 – 110

Completeness of Hoare Rules

Weakest Preconditions

- c: a command
- B: an extended boolean assertion
- I: an interpretation

Then we define the weakest precondition $wp^I[\![c, B]\!]$ by

$$wp^I[\![c, B]\!] := \{\sigma \in \Sigma_\perp \mid \mathcal{C}[\![c]\!](\sigma) \models^I B\}$$

Completeness of Hoare Rules

Definition: Expressiveness

The set **Assn** of extended boolean assertions is expressive if for every command c and extended boolean assertion B, there exists $A \in \textbf{Assn}$ such that $A^I = wp^I[\![c, B]\!]$ for all interpretations I.

Completeness of Hoare Rules

Theorem

Assn is expressive.

Proof

- by structural induction on commands c:

$$\forall B \in \textbf{Assn}. \exists w[\![c, B]\!] \in \textbf{Assn}. \forall I. (w[\![c, B]\!]^I = wp^I[\![c, B]\!])$$

Completeness of Hoare Rules

Proof: Skip

- $c = \textbf{skip}$;
- $w[\![c, B]\!] := B$;
- $$\begin{aligned} \sigma \in wp^I[\![\textbf{skip}, B]\!] \quad & \text{iff} \quad \mathcal{C}[\![\textbf{skip}]\!](\sigma) \models^I B \\ & \text{iff} \quad \sigma \models^I B \\ & \text{iff} \quad \sigma \in w[\![c, B]\!]^I \end{aligned}$$

Completeness of Hoare Rules

Proof: Assignment

- $c = \; X{:=}a;$
- $w[\![c, B]\!] := B\,[a/X];$
-

$$
\begin{aligned}
\sigma \in wp^{I}[\![X{:=}a, B]\!] \quad & \text{iff} \quad \mathcal{C}[\![X{:=}a]\!](\sigma) \models^{I} B \\
& \text{iff} \quad \sigma\,[\mathcal{A}[\![a]\!](\sigma)/X] \models^{I} B \\
& \text{iff} \quad \sigma \models^{I} B\,[a/X] \\
& \text{iff} \quad \sigma \in w[\![c, B]\!]^{I}
\end{aligned}
$$

Completeness of Hoare Rules

Proof: Sequential Composition

- $c = \ c_0; c_1\ ;$
- $w[\![c, B]\!] := w[\![c_0, w[\![c_1, B]\!]]\!];$
- $$\begin{aligned} \sigma \in wp^I[\![c_0; c_1, B]\!] \quad & \text{iff} \quad \mathcal{C}[\![c_0; c_1]\!](\sigma) \models^I B \\ & \text{iff} \quad \mathcal{C}[\![c_1]\!](\mathcal{C}[\![c_0]\!](\sigma)) \models^I B \\ & \text{iff} \quad \mathcal{C}[\![c_0]\!](\sigma) \in wp^I[\![c_1, B]\!] \\ & \text{iff} \quad \mathcal{C}[\![c_0]\!](\sigma) \models^I w[\![c_1, B]\!] \\ & \text{iff} \quad \sigma \in wp^I[\![c_0, w[\![c_1, B]\!]]\!] \\ & \text{iff} \quad \sigma \in w[\![c_0, w[\![c_1, B]\!]]\!]^I \\ & \text{iff} \quad \sigma \in w[\![c, B]\!]^I \end{aligned}$$

Completeness of Hoare Rules

Proof: Conditional Branch

- $c =$ **if** b **then** c_0 **else** c_1 ;
- $w\llbracket c, B\rrbracket = (b \wedge w\llbracket c_0, B\rrbracket) \vee (\neg b \wedge w\llbracket c_1, B\rrbracket)$;
-

$$
\begin{aligned}
\sigma \in wp^I\llbracket c, B\rrbracket \quad & \text{iff} \quad (\mathcal{B}\llbracket b\rrbracket(\sigma) = \textbf{true} \;\&\; \mathcal{C}\llbracket c_0\rrbracket(\sigma) \models^I B) \\
& \qquad \text{or } (\mathcal{B}\llbracket b\rrbracket(\sigma) = \textbf{false} \;\&\; \mathcal{C}\llbracket c_1\rrbracket(\sigma) \models^I B) \\
& \text{iff} \quad (\sigma \models^I b \;\&\; \sigma \in wp^I\llbracket c_0, B\rrbracket) \\
& \qquad \text{or } (\sigma \models^I \neg b \;\&\; \sigma \in wp^I\llbracket c_1, B\rrbracket)) \\
& \text{iff} \quad (\sigma \models^I b \;\&\; \sigma \models^I w\llbracket c_0, B\rrbracket) \\
& \qquad \text{or } (\sigma \models^I \neg b \;\&\; \sigma \models^I w\llbracket c_1, B\rrbracket)) \\
& \text{iff} \quad \sigma \models^I (b \wedge w\llbracket c_0, B\rrbracket) \vee (\neg b \wedge w\llbracket c_1, B\rrbracket) \\
& \text{iff} \quad \sigma \models^I w\llbracket c, B\rrbracket \\
& \text{iff} \quad \sigma \in w\llbracket c, B\rrbracket^I
\end{aligned}
$$

Completeness of Hoare Rules

Proof: While Loop

- $c =$ **while** b **do** c' ;
- $\mathcal{C}[\![c]\!](\sigma) \models^I B$ iff it holds that

$$
\begin{array}{l}
\forall k \geq 0. \forall \sigma_0, \ldots, \sigma_k \in \Sigma. \big(\\
\quad \big[\sigma = \sigma_0 \ \& \\
\quad\quad \forall 0 \leq i < k. (\sigma_i \models^I b \ \& \\
\quad\quad\quad \mathcal{C}[\![c']\!](\sigma_i) = \sigma_{i+1}) \\
\quad \big] \\
\quad \Rightarrow \sigma_k \models^I b \vee (\neg b \wedge B) \\
\big)
\end{array}
$$

Completeness of Hoare Rules

Proof: While Loop

- difficulty: translation into an assertion in **Assn**
- solution: Chinese Remainder Theorem

Completeness of Hoare Rules

Chinese Remainder Theorem

- $m_1, \ldots, m_n$: positive relatively-prime natural numbers (i.e., $\gcd(m_i, m_j) = 1$ whenever $i \neq j$)

Then for any integers $a_1, \ldots, a_n$ there exists a natural number x such that $x \equiv a_i \pmod{m_i}$ for all $i = 1, \ldots, n$.

Proof

For $i = 1, \ldots, n$, define $M_i := \prod_{j \neq i} m_j$. Then m_i and M_i are relatively prime. Thus we can find through the Euclidean Algorithm an integer b_i such that $b_i \cdot M_i \equiv 1 \pmod{m_i}$. Define

$$x := \left(\sum_{i=1}^{n} a_i \cdot b_i \cdot M_i \right) + \left(K \cdot \prod_{i=1}^{n} m_i \right) \quad .$$

Then x satisfies the conditions in the statement of the theorem.

Completeness of Hoare Rules

The Gödel's Predicate

- $a \bmod b$: the remainder of a divided by b

The Gödel's predicate β over natural numbers is defined by:

$$\beta(a, b, i, x) := \ x = (a \bmod (1 + (1 + i) \cdot b)) \ .$$

Exercise

- Give an assertion in **Assn** that expresses $x = (a \bmod b)$.
- Prove that β can be expressed in **Assn**.

Completeness of Hoare Rules

Lemma

For any sequence $n_0, \ldots, n_k$ of natural numbers there are natural numbers $n, m > 0$ such that

$$\forall 0 \leq j \leq k. \forall x. (\beta(n, m, j, x) \Leftrightarrow x = n_j)$$

Proof

Define $m := (\max\{k, n_0, \ldots, n_k\})!$ and $p_i := 1 + (1 + i) \cdot m$ for $i = 0, \ldots, k$.

- $p_0, \ldots, p_k$ are relative primes.
- $n_i < p_i$ for $i = 0, \ldots, k$.

By Chinese Remainder Theorem, there exists a natural number n such that $n \equiv n_i \pmod{p_i}$ for $i = 0, \ldots, k$. From $0 \leq n_i < p_i$, $(n \bmod p_i) = n_i$.

Completeness of Hoare Rules

The Predicate F

$$
\begin{aligned}
F(x, y) := x \geq 0 \ \& \ \exists z \geq 0.[(x = 2 \cdot z \Rightarrow y = z) \\
\& \ (x = 2 \cdot z + 1 \Rightarrow y = -z - 1)]
\end{aligned}
$$

Properties:

- $(F(x, y)$ and x is even$) \Rightarrow y = \frac{x}{2}$;
- $(F(x, y)$ and x is odd$) \Rightarrow y = -\frac{x-1}{2} - 1$;
- a bijection between natural numbers and integers

Completeness of Hoare Rules

The Predicate $\beta^{\pm}$

$$\beta^{\pm}(n, m, j, y) := \exists x. \, (\beta(n, m, j, x) \wedge F(x, y))$$

Lemma

For any sequence $n_0, \ldots, n_k$ of integers, there are natural numbers $n, m > 0$ such that for all $0 \leq j \leq k$ and all integers y we have

$$\beta^{\pm}(n, m, j, y) \Leftrightarrow y = n_j \quad .$$

Gödel's Predicate

Lemma

For any sequence $n_0, \ldots, n_k$ of integers, there are natural numbers $n, m > 0$ such that for all $0 \leq j \leq k$ and all integers y we have

$$\beta^{\pm}(n, m, j, y) \Leftrightarrow y = n_j \quad .$$

Proof

Construct the sequence $n'_0, \ldots, n'_k$ such that $F(n'_j, n_j)$ holds for all $0 \leq j \leq k$. From the previous lemma for β, there exist natural numbers $n, m > 0$ such that

$$\forall 0 \leq j \leq k . \forall x . \left(\beta(n, m, j, x) \Leftrightarrow x = n'_j \right)$$

Then the result follows from that $(F(x, y) \ \& \ x = n'_j) \Rightarrow y = n_j$.

Completeness of Hoare Rules

Proof: While Loop

- $c =$ **while** b **do** c' ;
- $\mathcal{C}[\![c]\!](\sigma) \models^{I} B$ iff it holds that

$$
\begin{array}{l}
\forall k \geq 0. \forall \sigma_0, \ldots, \sigma_k \in \Sigma. \big(\\
\quad \big[\sigma = \sigma_0 \ \& \\
\qquad \forall 0 \leq i < k. (\sigma_i \models^{I} b \ \& \\
\qquad\quad \mathcal{C}[\![c']\!](\sigma_i) = \sigma_{i+1}) \\
\quad \big] \\
\quad \Rightarrow \sigma_k \models^{I} b \vee (\neg b \wedge B) \\
\big)
\end{array}
$$

Completeness of Hoare Rules

Proof: While Loop

- ℓ locations (program variables): $\bar{X} := (X_1, \ldots, X_\ell)$
- encoding: each σ_i as an integer vector $\bar{s}_i = (s_{i,1}, \ldots, s_{i,\ell})$

$\mathcal{C}[\![c]\!](\sigma) \models^I B$ iff it holds that

$$
\begin{array}{l}
\forall k \geq 0. \forall \bar{s}_0, \ldots, \bar{s}_k. (\\
\quad \big[\sigma \models^I X = \bar{s}_0 \ \& \\
\qquad \forall 0 \leq i < k. (\models^I b \left[\bar{s}_i / \bar{X}\right] \ \& \\
\qquad\quad \models^I (w[\![c', \bar{X} = \bar{s}_{i+1}]\!] \wedge \neg w[\![c', \mathbf{false}]\!]) \left[\bar{s}_i / \bar{X}\right]) \\
\quad \big] \\
\quad \Rightarrow \models^I (b \vee B) \left[\bar{s}_k / \bar{X}\right] \\
)
\end{array}
$$

Completeness of Hoare Rules

Proof: While Loop

$\mathcal{C}[\![c]\!](\sigma) \models^I B$ iff $\sigma \models^I w[\![c, B]\!]$ where

$$
\begin{array}{l}
w[\![c, B]\!] := \\
\\
\forall k \geq 0. \forall n_1, m_1, \ldots, n_\ell, m_\ell > 0. \Big(\\
\quad \Big[(\bigwedge_{j=1}^{\ell} \beta^{\pm}(n_j, m_j, 0, X_j)) \ \wedge \\
\qquad \forall 0 \leq i < k. (\forall \bar{y}. (\bigwedge_{j=1}^{\ell} \beta^{\pm}(n_j, m_j, i, y_j) \Rightarrow b \left[\bar{y}/\bar{X}\right]) \ \wedge \\
\qquad\quad (\forall \bar{y}, \bar{z}. [\bigwedge_{j=1}^{\ell} (\beta^{\pm}(n_j, m_j, i, y_j) \wedge \beta^{\pm}(n_j, m_j, i+1, z_j)) \ \Rightarrow \\
\qquad\qquad (w[\![c', \bar{X} = \bar{z}]\!] \wedge \neg w[\![c', \mathbf{false}]\!]) \left[\bar{y}/\bar{X}\right]])) \\
\quad \Big] \\
\quad \Rightarrow (\forall \bar{y}. (\bigwedge_{j=1}^{\ell} \beta^{\pm}(n_j, m_j, k, y_j)) \Rightarrow (b \vee B) \left[\bar{y}/\bar{X}\right]) \\
\Big)
\end{array}
$$

Completeness of Hoare Rules

Theorem (Expressiveness)

For every command c and extended boolean assertion B, there exists $A \in$ **Assn** such that $A^I = wp^I[\![c, B]\!]$ for all interpretations I.

Completeness of Hoare Rules

Lemma

For any command c and assertion $B \in \textbf{Assn}$, if $w[\![c, B]\!]$ is any assertion satisfying that $w[\![c, B]\!]^I = wp^I[\![c, B]\!]$ for all I, then $\vdash \{w[\![c, B]\!]\}c\{B\}$.

Proof

By structural induction on c.

Completeness of Hoare Rules

Proof: Skip

- $c = \textbf{skip}$;

$$\frac{}{\{A\}\textbf{skip}\{A\}}$$

- $w[\![c, B]\!]^I = wp^I[\![c, B]\!]$ for all I;
- $\sigma \models^I w[\![c, B]\!]$ iff $\sigma \models^I B$;
- $\models w[\![c, B]\!] \Leftrightarrow B$;
- $\vdash \{w[\![c, B]\!]\}c\{B\}$;

Completeness of Hoare Rules

Proof: Assignment

- $c = X{:=}a$;

$$\frac{}{\{B\,[a/X]\}X := a\{B\}}$$

- $w[\![c, B]\!]^I = wp^I[\![c, B]\!]$ for all I;
-

$$\begin{aligned} \sigma \in w[\![c, B]\!]^I \quad & \text{iff} \quad \sigma \in wp^I[\![X{:=}a, B]\!] \\ & \text{iff} \quad \mathcal{C}[\![X{:=}a]\!](\sigma) \models^I B \\ & \text{iff} \quad \sigma\,[\mathcal{A}[\![a]\!](\sigma)/X] \models^I B \\ & \text{iff} \quad \sigma \models^I B\,[a/X] \end{aligned}$$

- $\models w[\![c, B]\!] \Leftrightarrow B\,[a/X]$ and hence $\vdash \{w[\![c, B]\!]\}c\{B\}$

Completeness of Hoare Rules

Proof: Sequential Composition

- $c = \ c_0; c_1 \ ;$

$$\frac{\{A\}c_0\{C\}\ ,\ \{C\}c_1\{B\}}{\{A\}c_0; c_1\{B\}}$$

- $w[\![c, B]\!]^I = wp^I[\![c, B]\!]$ for all I;
-

$$\begin{array}{rll} \sigma \in w[\![c, B]\!]^I & \text{iff} & \sigma \in wp^I[\![c_0; c_1, B]\!] \\ & \text{iff} & \mathcal{C}[\![c_0; c_1]\!](\sigma) \models^I B \\ & \text{iff} & \mathcal{C}[\![c_1]\!](\mathcal{C}[\![c_0]\!](\sigma)) \models^I B \\ & \text{iff} & \mathcal{C}[\![c_0]\!](\sigma) \models^I w[\![c_1, B]\!] \\ & \text{iff} & \sigma \in w[\![c_0, w[\![c_1, B]\!]]\!]^I \end{array}$$

Completeness of Hoare Rules

Proof: Sequential Composition

- $c = c_0; c_1$;

$$\frac{\{A\}c_0\{C\} \; , \; \{C\}c_1\{B\}}{\{A\}c_0; c_1\{B\}}$$

- $\models w[\![c, B]\!] \Leftrightarrow w[\![c_0, w[\![c_1, B]\!]]\!]$
- $\vdash \{w[\![c_1, B]\!]\}c_1\{B\}$
- $\vdash \{w[\![c_0, w[\![c_1, B]\!]]\!]\}c_0\{w[\![c_1, B]\!]\}$
- $\vdash \{w[\![c_0, w[\![c_1, B]\!]]\!]\}c\{B\}$
- $\vdash \{w[\![c, B]\!]\}c\{B\}$

Completeness of Hoare Rules

Proof: Conditional Branch

- $c =$ **if** b **then** c_0 **else** c_1 ;

$$\frac{\{A \wedge b\}c_0\{B\}\ ,\ \{A \wedge \neg b\}c_1\{B\}}{\{A\}\textbf{if } b \textbf{ then } c_0 \textbf{ else } c_1\{B\}}$$

- $w[\![c, B]\!]^I = wp^I[\![c, B]\!]$ for all I;
-

$$\begin{array}{lll}
\sigma \models^I w[\![c, B]\!] & \text{iff} & \sigma \in wp^I[\![c, B]\!] \\
 & \text{iff} & (\mathcal{B}[\![b]\!](\sigma) = \textbf{true} \ \&\ \mathcal{C}[\![c_0]\!](\sigma) \models^I B) \\
 & & \text{or } (\mathcal{B}[\![b]\!](\sigma) = \textbf{false} \ \&\ \mathcal{C}[\![c_1]\!](\sigma) \models^I B) \\
 & \text{iff} & (\sigma \models^I b \ \&\ \sigma \models^I w[\![c_0, B]\!]) \\
 & & \text{or } (\sigma \models^I \neg b \ \&\ \sigma \models^I w[\![c_1, B]\!]) \\
 & \text{iff} & \sigma \models^I (b \wedge w[\![c_0, B]\!]) \vee (\neg b \wedge w[\![c_1, B]\!])
\end{array}$$

Completeness of Hoare Rules

Proof: Conditional Branch

- $c =$ **if** b **then** c_0 **else** c_1 ;

$$\frac{\{A \wedge b\}c_0\{B\}\ ,\ \{A \wedge \neg b\}c_1\{B\}}{\{A\}\textbf{if } b \textbf{ then } c_0 \textbf{ else } c_1\{B\}}$$

- $\models w[\![c, B]\!] \Leftrightarrow [(b \wedge w[\![c_0, B]\!]) \vee (\neg b \wedge w[\![c_1, B]\!])]$
- $\vdash \{w[\![c_0, B]\!]\}c_0\{B\}$ and $\vdash \{w[\![c_1, B]\!]\}c_1\{B\}$
- $\models (w[\![c, B]\!] \wedge b) \Rightarrow w[\![c_0, B]\!]$
- $\models (w[\![c, B]\!] \wedge \neg b) \Rightarrow w[\![c_1, B]\!]$
- $\vdash \{w[\![c, B]\!] \wedge b\}c_0\{B\}$ and $\vdash \{w[\![c, B]\!] \wedge \neg b\}c_1\{B\}$
- $\vdash \{w[\![c, B]\!]\}c\{B\}$

Completeness of Hoare Rules

Proof: While Loop

- $c =$ **while** b **do** c';
- $A := w[\![c, B]\!]$;

We show that

- $\models \{A \wedge b\} c' \{A\}$;
- $\models (A \wedge \neg b) \Rightarrow B$.

Then we have

- $\vdash \{A \wedge b\} c' \{A\}$ from the induction hypothesis;
- $\vdash \{A\} c \{A \wedge \neg b\}$ from the while-loop rule;
- $\vdash \{A\} c \{B\}$ from the consequence rule;

Completeness of Hoare Rules

Proof: While Loop

- $c = \textbf{while } b \textbf{ do } c'$;
- $A := w[\![c, B]\!]$;

We show that $\models \{A \wedge b\}c'\{A\}$. The reasoning is as follows.

- $\sigma \models^I A \wedge b$
- $\sigma \models^I w[\![c, B]\!]$ and $\sigma \models^I b$
- $\mathcal{C}[\![c]\!](\sigma) \models^I B$ and $\sigma \models^I b$
- $\mathcal{C}[\![c]\!] = \mathcal{C}[\![\textbf{if } b \textbf{ then } c'; c \textbf{ else skip}]\!]$
- $\mathcal{C}[\![c'; c]\!](\sigma) \models^I B$
- $\mathcal{C}[\![c]\!](\mathcal{C}[\![c']\!](\sigma)) \models^I B$
- $w[\![c, B]\!]^I = wp^I[\![c, B]\!]$;
- $\mathcal{C}[\![c']\!](\sigma) \models^I w[\![c, B]\!](= A)$
- $\models \{A \wedge b\}c'\{A\}$

Completeness of Hoare Rules

Proof: While Loop

- $c = \textbf{while } b \textbf{ do } c'$;
- $A := w[\![c, B]\!]$;

We show that $\models (A \wedge \neg b) \Rightarrow B$. The reasoning is as follows.

- $\sigma \models^I A \wedge \neg b$
- $\mathcal{C}[\![c]\!](\sigma) \models^I B$ and $\sigma \models^I \neg b$
- $\mathcal{C}[\![c]\!] = \mathcal{C}[\![\textbf{if } b \textbf{ then } c'; c \textbf{ else skip}]\!]$
- $\mathcal{C}[\![c]\!](\sigma) = \sigma$ and $\sigma \models^I B$
- $\sigma \models^I (A \wedge \neg b) \Rightarrow B$
- $\models (A \wedge \neg b) \Rightarrow B$

Completeness of Hoare Rules

Theorem (Relative Completeness)

For any partial correctness assertion $\{A\}c\{B\}$, $\models \{A\}c\{B\}$ implies $\vdash \{A\}c\{B\}$.

Proof

- Suppose that $\models \{A\}c\{B\}$.
- We have $\vdash \{w[\![c, B]\!]\}c\{B\}$ where $w[\![c, B]\!]^I = wp^I[\![c, B]\!]$ for all interpretations I.
- By the consequence rule and $\models A \Rightarrow w[\![c, B]\!]$, we obtain $\vdash \{A\}c\{B\}$.

Proving Gödel's Incompleteness Theorem

Textbook, Page 110 – 112

Gödel's Incompleteness Theorem

Theorem

The set $\{A \in \textbf{Assn} \mid \models A\}$ is not recursively enumerable.

Proof (by Contradiction)

- Suppose that $\{A \in \textbf{Assn} \mid \models A\}$ is recursively enumerable.
- For each command c, construct the assertion

$$A := w[\![c, \textbf{false}]\!] \left[\vec{0}/\bar{X}\right] \quad .$$

- c does not terminate on the input $\vec{0}$ iff $\models A$.
- However, the set of all those c's is known to be not recursively enumerable (Textbook, Appendix A).

Gödel's Incompleteness Theorem

Gödel's Incompleteness Theorem

There is no effective proof system for **Assn** such that its theorems coincide with the valid assertions in **Assn**.

Proof (by Contradiction)

- Suppose that there is an effective proof system for **Assn**.
- By enumerating the set of all proofs (derivation trees), the set

$$\{A \in \mathbf{Assn} \mid \models A\}$$

would become recursively enumerable. Contradiction.

Summary

- relative completeness of Hoare rules
- a proof for Gödel's Incompleteness Theorem
- finishing all of the operational, denotational and axiomatic semantics for imperative programs

Exercise

Problem

Let c be the command **while** $X \leq 100$ **do** $X := (2 \times X) + 1$ with location X. Calculate the weakest precondition $wp^I[\![c, B]\!]$ where the postcondition $B = X \geq 150$ and the interpretation I is dummy (i.e., of no use) here.

Exercise

Problem

Let c be the following command:

```
$\textbf{while}\ N \leq M\ \textbf{do}$
  $[L := 1\,;$
    $\textbf{while}\ 2 \times L \times N \leq M\ \textbf{do}\ L := 2 \times L\,;$
    $K := K + L\,;$
    $M := M - (L \times N)]$
```

Prove through the Hoare rules that $\models \{A\}c\{B\}$ where

- the precondition A is $M = m \wedge M \geq 0 \wedge N \geq 1 \wedge K = 0$, and
- the postcondition B is $m = (K \times N) + M \wedge 0 \leq M \wedge M < N$, and
- m is an integer variable.

Chapter 8

Introduction to domain theory

Domain Theory

- advanced constructions on complete partial orders (cpo's)
- a meta-language for complete partial orders

Topics

- advanced constructions on complete partial orders (cpo's)

Complete Partial Orders

Complete Partial Orders

Recall: Partial Orders

A partial order is an ordered pair $(P, \sqsubseteq)$ such that P is a set and $\sqsubseteq$ is a binary relation $\sqsubseteq \ \subseteq P \times P$ satisfying the following conditions:

- (reflexibility) $\forall p \in P.p \sqsubseteq p$;
- (transitivity) $\forall p, q, r \in P. [(p \sqsubseteq q \ \& \ q \sqsubseteq r) \Rightarrow p \sqsubseteq r]$;
- (antisymmetry) $\forall p, q \in P. [(p \sqsubseteq q \ \& \ q \sqsubseteq p) \Rightarrow p = q]$.

Complete Partial Orders

Recall: Upper Bounds

- $(P, \sqsubseteq)$: a partial order
- X: a subset of P (i.e., that satisfies $X \subseteq P$)

$p \in P$ is an upper bound of X if $\forall q \in X. q \sqsubseteq p$.

Recall: Least Upper Bounds

$p \in P$ is a least upper bound (in short, lub) of X if

- p is an upper bound of X, and
- for all upper bounds q of X, $p \sqsubseteq q$

Complete Partial Orders

Recall: Least Upper Bounds

$p \in P$ is a least upper bound (in short, lub) of X if

- p is an upper bound of X, and
- for all upper bounds q of X, $p \sqsubseteq q$

Recall: Notation

- The least upper bound of X (if exists) is denoted by $\bigsqcup X$.
- If $X = \{d_1, \ldots, d_n\}$, then $d_1 \sqcup \cdots \sqcup d_n := \bigsqcup X$.

Complete Partial Orders

Recall: ω-Chains

- $(P, \sqsubseteq)$: a partial order

An ω-chain in P is an infinite sequence $d_0, d_1, \ldots, d_n, \ldots$ in P such that $d_0 \sqsubseteq d_1 \sqsubseteq \cdots \sqsubseteq d_n \sqsubseteq \ldots$.

Recall: Complete Partial Orders (CPOs)

$(P, \sqsubseteq)$ is a complete partial order (cpo) if for any ω-chain

$$d_0 \sqsubseteq d_1 \sqsubseteq \cdots \sqsubseteq d_n \sqsubseteq \ldots$$

in P, the least upper bound

$$\bigsqcup_{n\in\omega} d_n := \bigsqcup\{d_n \mid n \in \omega\} = \bigsqcup\{d_0, d_1, \ldots, d_n, \ldots\}$$

exists in P.

Complete Partial Orders

Recall: Least Elements

- $(P, \sqsubseteq)$: a partial order

$p \in P$ is a least element if $\forall q \in P . p \sqsubseteq q$.

Recall: CPOs with Bottom

- $(P, \sqsubseteq)$: a cpo

$(P, \sqsubseteq)$ is a cpo with bottom if P has a (unique) least element $\perp_P$.

Complete Partial Orders

Recall: Set Inclusion

- A: a set
- $D := 2^A$
- $\sqsubseteq := \{(X, Y) \in D \times D \mid X \subseteq Y\}$

Complete Partial Orders

Recall: Partial Functions

- B, C: sets
- $D := \{F \mid F : B \rightharpoonup C\}$
- $\sqsubseteq := \{(F, G) \in D \times D \mid F \subseteq G\}$

Complete Partial Orders

Monotonic Functions

- $(D, \sqsubseteq_D)$ and $(E, \sqsubseteq_E)$: partial orders

A function $f : D \to E$ is monotonic if

$$\forall d, d' \in D. [d \sqsubseteq_D d' \Rightarrow f(d) \sqsubseteq_E f(d')]$$

Continuous Functions

Definition

- $(D, \sqsubseteq_D)$ and $(E, \sqsubseteq_E)$: cpo's

A function $f : D \to E$ is continuous if the followings hold:

- f is monotonic;
- for all ω-chains $d_0 \sqsubseteq d_1 \sqsubseteq \cdots \sqsubseteq d_n \sqsubseteq \ldots$ in D, we have that

$$\bigsqcup_{n\in\omega} f(d_n) = f\left(\bigsqcup_{n\in\omega} d_n\right)$$

Fixed Points

Definition

- $(D, \sqsubseteq_D)$: a partial order
- $f : D \to D$: a function

An element $d \in D$ is:

- a fixed point of f if $f(d) = d$;
- a prefixed point of f if $f(d) \sqsubseteq d$;

Complete Partial Orders

The Fixed-Point Theorem

- $(D, \sqsubseteq_D)$: a cpo with bottom $\bot_D$
- $f : D \to D$: a continuous function
- $\bot_D \sqsubseteq_D f(\bot_D) \sqsubseteq_D \cdots \sqsubseteq_D f^n(\bot_D) \sqsubseteq_D \ldots$
- $\mathit{fix}(f) := \bigsqcup_{n \in \omega} f^n(\bot_D)$

Then

- $\mathit{fix}(f)$ is a fixed point of f: $f(\mathit{fix}(f)) = \mathit{fix}(f)$
- $\mathit{fix}(f)$ is the least prefixed point of f: $f(d) \sqsubseteq d \Rightarrow \mathit{fix}(f) \sqsubseteq d$
- $\mathit{fix}(f)$ is the least fixed point of f: $f(d) = d \Rightarrow \mathit{fix}(f) \sqsubseteq d$

Complete Partial Orders

Isomorphisms

- $(D, \sqsubseteq_D)$ and $(E, \sqsubseteq_E)$: cpo's

A continuous function $f : D \to E$ is an isomorphism if

- f is a 1−1 correspondence;
- $\forall d, d' \in D.\, [d \sqsubseteq_D d' \Leftrightarrow f(d) \sqsubseteq_E f(d')]$

Complete Partial Orders

Exercise

- D, E, F: cpo's (with their implicit ordering relations)
- $f : D \to E$ and $g : E \to F$: continuous functions

Then:

- the identity function $\mathrm{Id}_D : D \to D$ (such that $\mathrm{Id}_D(d) = d$ for all $d \in D$) is continuous;
- the function $g \circ f : D \to F$ is continuous.

Constructions on CPO's

Discrete CPO's

Discrete CPO's

Discrete CPO's

A discrete cpo is a partial order $(D, \sqsubseteq)$ such that $\sqsubseteq$ is the identity relation on D, i.e., $\sqsubseteq = \{(d, d) \mid d \in D\}$.

Exercise

Prove that the discrete cpo $(D, \sqsubseteq)$ above is indeed a cpo.

Discrete CPO's

Exercise

- D: a discrete cpo
- E: a cpo

Prove that any function $f : D \to E$ is continuous.

Product CPO's

Product CPO's

Cartesian Product $D_1 \times \cdots \times D_k$

- $D_1, \ldots, D_k$: cpo's
- $D_1 \times \cdots \times D_k$: the Cartesian product of $D_1, \ldots, D_k$,

$$(d_1, \ldots, d_k) \in D_1 \times \cdots \times D_k \text{ iff } d_i \in D_i \text{ for } i = 1, \ldots, k$$

Then the product cpo $(D, \sqsubseteq)$ is given by:

- $D := D_1 \times \cdots \times D_k$;
- $(d_1, \ldots, d_k) \sqsubseteq (d'_1, \ldots, d'_k)$ iff $d_i \sqsubseteq d'_i$ for $i = 1, \ldots, k$

Product CPO's

Exercise

For any ω-chain

$$(d_{1,0},\ldots,d_{k,0}) \sqsubseteq (d_{1,1},\ldots,d_{k,1}) \sqsubseteq \ \ldots \ \sqsubseteq (d_{1,n},\ldots,d_{k,n}) \sqsubseteq \ldots$$

in $D_1 \times \cdots \times D_k$, we have

$$\bigsqcup_{n\in\omega}(d_{1,n},\ldots,d_{k,n}) = (\bigsqcup_{n\in\omega} d_{1,n},\ldots,\bigsqcup_{n\in\omega} d_{k,n}).$$

Proposition

$D_1 \times \cdots \times D_k$ is a cpo.

Product CPO's

The Projection Function

- $D_1, \ldots, D_k$: cpo's
- $D_1 \times \cdots \times D_k$: the Cartesian product of $D_1, \ldots, D_k$

Define the projection functions

$$\pi_i : D_1 \times \cdots \times D_k \to D_i \ (i = 1, \ldots, k)$$

by

$$\pi_i(d_1, \ldots, d_k) := d_i.$$

Exercise

Prove that each π_i is continuous.

Product CPO's

Tupling Function

- $E, D_1, \ldots, D_k$: cpo's
- $f_i : E \to D_i$ $(i = 1, \ldots, k)$: continuous functions

Define the tupling function

$$\langle f_1, \ldots, f_k \rangle : E \to D_1 \times \cdots \times D_k$$

by

$$\langle f_1, \ldots, f_k \rangle(e) := (f_1(e), \ldots, f_k(e))$$

Product CPO's

The Continuity of $\langle f_1, \ldots, f_k \rangle$

For any ω-chain

$$e_1 \sqsubseteq e_2 \sqsubseteq \cdots \sqsubseteq e_n \sqsubseteq \ldots$$

we have

$$\begin{array}{rcl}
\langle f_1, \ldots, f_k \rangle(\bigsqcup_{n\in\omega} e_n) & = & (f_1(\bigsqcup_{n\in\omega} e_n), \ldots, f_k(\bigsqcup_{n\in\omega} e_n)) \\
& = & (\bigsqcup_{n\in\omega} f_1(e_n), \ldots, \bigsqcup_{n\in\omega} f_k(e_n)) \\
& = & \bigsqcup_{n\in\omega}(f_1(e_n), \ldots, f_k(e_n)) \\
& = & \bigsqcup_{n\in\omega}\langle f_1, \ldots, f_k \rangle(e_n)
\end{array}$$

Product CPO's

Exercise

- $D_1, \ldots, D_k, E_1, \ldots, E_k$: cpo's
- $f_i : D_i \to E_i$ $(i = 1, \ldots, k)$: continuous functions

Define the function

$$f_1 \times \cdots \times f_k : D_1 \times \cdots \times D_k \to E_1 \times \cdots \times E_k$$

by

$$f_1 \times \cdots \times f_k(d_1, \ldots, d_k) := (f_1(d_1), \ldots, f_k(d_k)).$$

Prove that $f_1 \times \cdots \times f_k$ is continuous.

Product CPO's

Lemma

- $E, D_1, \ldots, D_k$: cpo's
- $h : E \to D_1 \times \cdots \times D_k$: a function

Then h is continuous iff for $i = 1, \ldots, k$, the functions $\pi_i \circ h : E \to D_i$ are continuous.

Proof

"$\Rightarrow$": From compositionality of continuous functions.
"$\Leftarrow$": For all $e \in E$,

$$\begin{aligned} h(e) &= (\pi_1(h(e)), \ldots, \pi_k(h(e))) \\ &= (\pi_1 \circ h(e), \ldots, \pi_k \circ h(e)) \\ &= \langle \pi_1 \circ h, \ldots, \pi_k \circ h \rangle (e) \end{aligned}$$

Product CPO's

Lemma

- $E, D_1, \ldots, D_k$: cpo's
- $f : D_1 \times \cdots \times D_k \to E$: a function

Then f is continuous iff for all $1 \leq i \leq k$ and all elements

$$d_1, \ldots, d_{i-1}, d_{i+1}, \ldots, d_k$$

we have that the function $h_i : D_i \to E$ defined as

$$d \mapsto f(d_1, \ldots, d_{i-1}, d, d_{i+1}, \ldots, d_k)$$

is continuous.

Product CPO's

Proposition

- E: a cpo
- $e_{n,m}$ $(n \in \omega, m \in \omega)$: elements of the cpo E
- $e_{n,m} \sqsubseteq e_{n',m'}$ whenever $n \leq n'$ and $m \leq m'$

Then we have that

- The set $\{e_{n,m} \mid n, m \in \omega\}$ has a (unique) least upper bound.
- $$\bigsqcup_{n,m\in\omega} e_{n,m} = \bigsqcup_{n\in\omega}(\bigsqcup_{m\in\omega} e_{n,m}) = \bigsqcup_{m\in\omega}(\bigsqcup_{n\in\omega} e_{n,m}) = \bigsqcup_{n\in\omega} e_{n,n}$$

Product CPO's

Proposition

- $e_{n,m}$ $(n \in \omega, m \in \omega)$: elements of the cpo E
- $e_{n,m} \sqsubseteq e_{n',m'}$ whenever $n \leq n'$ and $m \leq m'$

Then we have that

- The set $\{e_{n,m} \mid n, m \in \omega\}$ has a (unique) least upper bound.
- $\bigsqcup_{n,m} e_{n,m} = \bigsqcup_n (\bigsqcup_m e_{n,m}) = \bigsqcup_m (\bigsqcup_n e_{n,m}) = \bigsqcup_n e_{n,n}$

Proof

- $\bigsqcup_{n,m \in \omega} e_{n,m} = \bigsqcup_{n,n \in \omega} e_{n,n}$;
- $\bigsqcup_{n,m \in \omega} e_{n,m} = \bigsqcup_{n \in \omega} (\bigsqcup_{m \in \omega} e_{n,m})$;
- $\bigsqcup_{n,m \in \omega} e_{n,m} = \bigsqcup_{m \in \omega} (\bigsqcup_{n \in \omega} e_{n,m})$;

Product CPO's

Lemma

- $E, D_1, \ldots, D_k$: cpo's
- $f : D_1 \times \cdots \times D_k \to E$: a function

Then f is continuous iff for all $1 \leq i \leq k$ and all elements

$$d_1, \ldots, d_{i-1}, d_{i+1}, \ldots, d_k$$

we have that the function $h_i : D_i \to E$ defined as

$$d \mapsto f(d_1, \ldots, d_{i-1}, d, d_{i+1}, \ldots, d_k)$$

is continuous.

Product CPO's

Proof

"$\Rightarrow$": Straightforward.

"$\Leftarrow$": The monotonicity is straightforward. For the rest of the proof, we take $k = 2$. Consider any ω-chain

$$(x_0, y_0) \sqsubseteq \cdots \sqsubseteq (x_n, y_n) \sqsubseteq \ldots$$

in $D_1 \times D_2$. Then

$$\begin{array}{rcl} f(\bigsqcup_n (x_n, y_n)) & = & f(\bigsqcup_n x_n, \bigsqcup_m y_m) \\ & = & \bigsqcup_n f(x_n, \bigsqcup_m y_m) \\ & = & \bigsqcup_n \bigsqcup_m f(x_n, y_m) \\ & = & \bigsqcup_n f(x_n, y_n) \end{array}$$

Function Space

Function Space

Definitions

- D, E: cpo's
- $[D \to E] := \{f \mid f : D \to E \text{ is continuous.}\}$.
- For $f, g \in [D \to E]$, $f \sqsubseteq g$ iff $\forall d \in D. f(d) \sqsubseteq g(d)$.

Function Space

Theorem

$([D \to E], \sqsubseteq)$ is a complete partial order.

Proof

Consider any ω-chain

$$f_0 \sqsubseteq f_1 \sqsubseteq \cdots \sqsubseteq f_n \sqsubseteq \ldots$$

in $([D \to E], \sqsubseteq)$. Then the least upper bound $\bigsqcup_{n \in \omega} f_n$ is given by:

$$(\bigsqcup_{n \in \omega} f_n)(d) := \bigsqcup_{n \in \omega}(f_n(d)) \text{ for all } d \in D.$$

We still need to prove that $\bigsqcup_{n \in \omega} f_n \in [D \to E]$.

Function Space

Theorem

$([D \rightarrow E], \sqsubseteq)$ is a complete partial order.

Proof (Continued)

We still need to prove that $\bigsqcup_{n\in\omega} f_n \in [D\rightarrow E]$. For any ω-chain

$$d_0 \sqsubseteq d_1 \sqsubseteq \cdots \sqsubseteq d_m \sqsubseteq \ldots$$

in D, we have that

$$\begin{aligned}
(\textstyle\bigsqcup_{n\in\omega} f_n)(\bigsqcup_{m\in\omega} d_m) &= \textstyle\bigsqcup_{n\in\omega} f_n(\bigsqcup_{m\in\omega} d_m) \\
&= \textstyle\bigsqcup_{n\in\omega} \bigsqcup_{m\in\omega}(f_n(d_m)) \\
&= \textstyle\bigsqcup_{m\in\omega} \bigsqcup_{n\in\omega}(f_n(d_m)) \\
&= \textstyle\bigsqcup_{m\in\omega}(\bigsqcup_{n\in\omega} f_n)(d_m).
\end{aligned}$$

Function Space

Bottom Element

- D, E: cpo's
- $[D{\rightarrow}E] := \{f \mid f : D \rightarrow E \text{ is continuous.}\}$.
- For $f, g \in [D{\rightarrow}E]$, $f \sqsubseteq g$ iff $\forall d \in D. f(d) \sqsubseteq g(d)$.

If E has a bottom element $\perp_E$, then $[D{\rightarrow}E]$ also has a bottom element given by:

$$\perp_{[D\rightarrow E]}(d) := \perp_E \text{ for all } d \in D.$$

Function Space

Powers

If D is a discrete cpo, then $[D \rightarrow E]$ is a power, denoted by E^D.

Function Space

Application

- D, E: cpo's

Define $apply : ([D \to E] \times D) \to E$ by:

$$apply(f, d) := f(d) \text{ for all } f \in [D \to E], d \in D.$$

Theorem

The function *apply* is continuous.

Function Space

Theorem

The function *apply* is continuous.

Proof

apply is continuous in its first argument:

- monotonicity;
- consider any ω-chain $f_0 \sqsubseteq \cdots \sqsubseteq f_n \sqsubseteq \ldots$ in $[D \rightarrow E]$;
- $apply(\bigsqcup_n f_n, d) = (\bigsqcup_n f_n)(d) = \bigsqcup_n (f_n(d)) = \bigsqcup_n apply(f_n, d)$.

apply is continuous in its second argument:

- monotonicity;
- consider any $d_0 \sqsubseteq \cdots \sqsubseteq d_n \sqsubseteq \ldots$ in D;
- $apply(f_n, \bigsqcup_n d_n) = f(\bigsqcup_n d_n) = \bigsqcup_n (f(d_n)) = \bigsqcup_n apply(f, d_n)$.

Function Space

λ-Notation

- X, Y: sets
- $f : X \to Y$: a function
- e: an expression representing f (e.g., $e = x + 1$ and $f(x) = x + 1$)

Then we denote also by $\lambda x \in X.e$ the function f.

Examples

- $\lambda x \in \omega.(x + 1)$: the function $f(x) = x + 1$
- $\lambda x \in \mathbb{R}. \sin x$: the function $f(x) = \sin x$

Function Space

Currying

- D, E, F: cpo's
- $g : F \times D \to E$: a continuous function

Define the function $curry(g) : F \to [D{\to}E]$ by:

$$curry(g) := \lambda v \in F.(\lambda d \in D.g(v, d))$$

Theorem

- For all $v \in F$, $curry(g)(v) \in [D{\to}E]$.
- $curry(g)$ is continuous.

Function Space

Theorem

- For all $v \in F$, $curry(g)(v) \in [D \to E]$.

Proof

- $curry(g)(v) = \lambda d \in D.g(v, d)$;
- g is continuous in its second argument.

Function Space

Theorem

- $\mathit{curry}(g)$ is continuous.

Proof

- monotonicity;
- Consider any ω-chain $v_0 \sqsubseteq v_1 \sqsubseteq \cdots \sqsubseteq v_n \sqsubseteq \ldots$ in F. Then for all $d \in D$,

$$\begin{array}{rcl}(\mathit{curry}(g)(\bigsqcup_n v_n))(d) & = & g(\bigsqcup_n v_n, d) \\ & = & \bigsqcup_n g(v_n, d) \\ & = & \bigsqcup_n((\mathit{curry}(g)(v_n))(d)) \\ & = & (\bigsqcup_n(\mathit{curry}(g)(v_n)))(d)\end{array}$$

- $\mathit{curry}(g)(\bigsqcup_n v_n) = \bigsqcup_n(\mathit{curry}(g)(v_n))$

Lifting

Lifting

Definition

- D: a cpo
- $\bot$: a fresh bottom element
- $\lfloor - \rfloor$: a copy function on D such that
  - for all $d, d' \in D$, $d = d' \Leftrightarrow \lfloor d \rfloor = \lfloor d' \rfloor$;
  - $\lfloor d \rfloor \neq \bot$ for all $d \in D$;

Then we define the lifted cpo $D_\bot$ by:

- $D_\bot := \{\lfloor d \rfloor \mid d \in D\} \cup \{\bot\}$;
- for all $d'_0, d'_1 \in D_\bot$, $d'_0 \sqsubseteq d'_1$ iff
  - either $d'_0 = \bot$,
  - or $d'_0 = \lfloor d_0 \rfloor, d'_1 = \lfloor d_1 \rfloor$ and $d_0 \sqsubseteq_D d_1$.

Lifting

Definition

We define the lifted cpo $D_\bot$ by:

- $D_\bot := \{\lfloor d \rfloor \mid d \in D\} \cup \{\bot\}$;
- for all $d'_0, d'_1 \in D_\bot$, $d'_0 \sqsubseteq d'_1$ iff
  - either $d'_0 = \bot$,
  - or $d'_0 = \lfloor d_0 \rfloor, d'_1 = \lfloor d_1 \rfloor$ and $d_0 \sqsubseteq_D d_1$.

Exercise

- $D_\bot$ is a cpo with bottom.
- $\lfloor - \rfloor : D \to D_\bot$ is continuous.

Lifting

The Operator $(-)^*$

- D: a cpo
- $\bot$: a fresh bottom element
- E: a cpo with the bottom element $\bot_E$
- $f : D \to E$: a continuous function

Define $f^* : D_\bot \to E$ by:

$$f^*(d') := \begin{cases} f(d) & \text{if } d' = \lfloor d \rfloor \text{ for some } d \in D \\ \bot_E & \text{otherwise (i.e. } d' = \bot) \end{cases}$$

Then (exercise) f^* is continuous.

Lifting

Continuity of $(-)^*$

- monotonicity: straightforward by definition;
- $f_0 \sqsubseteq f_1 \sqsubseteq \cdots \sqsubseteq f_n \sqsubseteq \ldots$: an ω-chain in $[D \to E]$

Consider any $d' \in D_\perp$:

- if $d' = \perp$, then $(\bigsqcup_{n\in\omega} f_n)^*(d') = (\bigsqcup_{n\in\omega}(f_n)^*)(d') = \perp_E$;
- if $d' = \lfloor d \rfloor$, then

$$\begin{array}{rcl} (\bigsqcup_{n\in\omega} f_n)^*(d') & = & (\bigsqcup_{n\in\omega} f_n)(d) \\ & = & \bigsqcup_{n\in\omega}(f_n(d)) \\ & = & \bigsqcup_{n\in\omega}((f_n)^*(d')) \\ & = & (\bigsqcup_{n\in\omega}(f_n)^*)(d') \end{array}$$

Thus $(\bigsqcup_{n\in\omega} f_n)^* = \bigsqcup_{n\in\omega}(f_n)^*$.

Lifting

"let" Notation

- D: a cpo
- $\bot$: a fresh bottom element
- E: a cpo with the bottom element $\bot_E$
- $f : D \to E$: a continuous function
- $\lambda x \in D.e$: a lambda notation for f

Define

$$\mathit{let}\ x \Leftarrow d'.e := (\lambda x \in D.e)^*(d') \text{ for } d' \in D_\bot.$$

Lifting

Abbreviation

- $let\ x_1 \Leftarrow c_1.(let\ x_2 \Leftarrow c_2.(\cdots\ (let\ x_k \Leftarrow c_k.\ e)\ \cdots))$
- $let\ x_1 \Leftarrow c_1, \cdots, x_k \Leftarrow c_k.\ e$

Lifting

Truth Values

- $\mathbf{T} = \{\mathbf{true}, \mathbf{false}\}$;
- $\vee : \mathbf{T} \times \mathbf{T} \to \mathbf{T}$: the or-function (from the truth table);

Define $\vee_\perp : \mathbf{T}_\perp \times \mathbf{T}_\perp \to \mathbf{T}_\perp$ by:

$$x_1 \vee_\perp x_2 := \mathit{let}\ t_1 \Leftarrow x_1, t_2 \Leftarrow x_2. \lfloor t_1 \vee t_2 \rfloor \quad .$$

Lifting

Arithmetic Operations

$$x_1 +_\perp x_2 := \mathit{let}\ n_1 \Leftarrow x_1, n_2 \Leftarrow x_2 . \lfloor n_1 + n_2 \rfloor \qquad .$$

Sums (Disjoint Unions)

Sums (Disjoint Unions)

Definition

- $D_1, \ldots, D_k$: cpo's
- $in_1, \ldots, in_k$: 1–1 injection functions that make disjoint copies

Define $D_1 + \cdots + D_k$ to be the cpo as follows:

- the underlying set is the disjoint union

$$\{in_1(d_1) \mid d_1 \in D_1\} \cup \cdots \cup \{in_k(d_k) \mid d_k \in D_k\};$$

- $d_1 \sqsubseteq d_2$ iff

$$\exists 1 \leq i \leq k . \exists d_1', d_2' . (d_1 = in_i(d_1') \; \& \; d_2 = in_i(d_2') \; \& \; d_1' \sqsubseteq_{D_i} d_2')$$

Exercise

- $D_1 + \cdots + D_k$ is a cpo.
- each $in_i : D_i \to D_1 + \cdots + D_k$ is continuous.

Sums (Disjoint Unions)

Combination of Continuous Functions

- $E, D_1, \ldots, D_k$: cpo's
- $f_i : D_i \to E$ $(i = 1, \ldots, k)$: continuous functions

Define $[f_1, \ldots, f_k] : D_1 + \cdots + D_k \to E$ by

$$[f_1, \ldots, f_k](in_i(d_i)) := f_i(d_i) \text{ for all } i \text{ and } d_i \in D_i.$$

Exercise

- $[f_1, \ldots, f_k]$ is a continuous function.
- The map $(f_1, \ldots, f_k) \mapsto [f_1, \ldots, f_k]$ is continuous.

Sums (Disjoint Unions)

Conditional Branches

- $\mathbf{T} = \{\mathbf{true}, \mathbf{false}\} = \{\mathbf{true}\} + \{\mathbf{false}\}$
- E: a cpo
- e_1, e_2: elements in E
- $\lambda x_1.e_1 : \{\mathbf{true}\} \to E$
- $\lambda x_2.e_2 : \{\mathbf{false}\} \to E$
- $cond(t, e_1, e_2) := [\lambda x_1.e_1, \lambda x_2.e_2](t)$
- $$cond(t, e_1, e_2) = \begin{cases} e_1 & \text{if } t = \mathbf{true} \\ e_2 & \text{if } t = \mathbf{false} \end{cases}$$

Sums (Disjoint Unions)

Conditional Branches

- $\mathbf{T} = \{\mathbf{true}, \mathbf{false}\} = \{\mathbf{true}\} + \{\mathbf{false}\}$
- E: a cpo with bottom $\perp_E$
- $$(b \to e_1 \mid e_2) := let\ t \Leftarrow b.cond(t, e_1, e_2) = \begin{cases} e_1 & \text{if } b = \lfloor \mathbf{true} \rfloor \\ e_2 & \text{if } b = \lfloor \mathbf{false} \rfloor \\ \perp_E & \text{if } b = \perp \end{cases}$$

Sums (Disjoint Unions)

Case Construction

- $E, D_1, \ldots, D_k$: cpo's
- d: an element in $D_1 + \cdots + D_k$
- $\lambda x_i.e_i : D_i \to E$ $(i = 1, \ldots, k)$: continuous functions
- $$\begin{aligned} \textbf{case } d \textbf{ of } & in_1(x_1).e_1 \mid \\ & \vdots \\ & in_k(x_k).e_k \end{aligned}$$
- $[\lambda x_1.e_1, \ldots, \lambda x_k.e_k](d)$

Summary

Advanced Constructions for CPO's

- discrete cpo's
- product cpo's
- function space
- lifting
- sums (disjoint unions)

Topics

- a meta-language for cpo's and continuous functions

A Metalanguage

A Metalanguage

Motivation

- an programming-language-like syntax for continuous functions
- guaranteed continuity from the syntax

A Metalanguage

λ-Notation

- D, E: cpo's
- x: a variable representing an element in D
- e: an expression that represents an element in E (e.g. $x + 1$)

We use the notation

$$\lambda x \in D.e \text{ (or simply } \lambda x.e\text{)}$$

for the function $h : D \to E$ such that $h(d) := e\,[d/x]$ for all $d \in D$.

A Metalanguage

λ Notation

- D_1, D_2, E: cpo's
- e: an expression with variables x, y

We write

- $\lambda z \in D_1 \times D_2.(e[\pi_1(z)/x, \pi_2(z)/y])$
- $\lambda(x, y) \in D_1 \times D_2.e$
- $\lambda x \in D_1, y \in D_2.e$
- $\lambda x, y.e$

A Metalanguage

Continuity of Expressions

- D, E: cpo's
- e: an expression representing an element in E
- x: a variable ranging over elements from D

The expression e is continuous in the variable x if the function

$$\lambda x \in D.e : D \to E$$

is continuous no matter which values the other free variables take.

A Metalanguage

Continuity of Expressions

- D, E: cpo's
- e: an expression representing an element in E

Then e is continuous in its variables if e is continuous in all its variables.

A Metalanguage

The Roadmap

- expressions for continuous functions
- recursive construction

A Metalanguage

Variables

Each single variable x is continuous in its variables.

Proof

- $\lambda x.x$ (the identity function)
- $\lambda y.x$ ($y \neq x$) (a constant function)

A Metalanguage

Constants

Constant expressions are continuous in their variables since they represent constant functions.

Examples

- a bottom element $\perp_D$ of a cpo D
- truth values **true**, **false**
- projections functions π_i's
- function application *apply*
- the operator $(-)^*$
- ...

A Metalanguage

Tupling

- $E_1, \ldots, E_k$: cpo's
- e_i $(1 \leq i \leq k)$: expressions for elements of E_i
- $(e_1, \ldots, e_k)$: the tuple expression for elements of $E_1 \times \cdots \times E_k$

Then the expression $(e_1, \ldots, e_k)$ is continuous in its variables iff every e_i is continuous in its variables.

Proof

For all variables x, we have

$$\begin{array}{ll} & (e_1, \ldots, e_k) \text{ is continuous in } x \\ \Leftrightarrow & \lambda x.(e_1, \ldots, e_k) \text{ is continuous} \\ \Leftrightarrow & \pi_i \circ (\lambda x.(e_1, \ldots, e_k)) \text{ is continuous for all } i \\ \Leftrightarrow & \lambda x.e_i \text{ is continuous for all } i \\ \Leftrightarrow & e_i \text{ is continuous in } x \text{ for all } i \end{array}$$

A Metalanguage

Application

- K: a constant continuous function (e.g., π_i, *apply*)
- e: an expression

Then the expression $K(e)$ is continuous in its variables if the expression e is continuous in its variables.

Proof

For all variables x, we have:

$$\begin{array}{ll} & K(e) \text{ is continuous in } x \\ \Leftrightarrow & \lambda x.K(e) \text{ is continuous} \\ \Leftrightarrow & K \circ (\lambda x.e) \text{ is continuous} \\ \Leftarrow & \lambda x.e \text{ is continuous} \\ \Leftrightarrow & e \text{ is continuous in } x \end{array}$$

A Metalanguage

Application

- e_1, e_2: two expressions

Then the expression $e_1(e_2)$ is continuous in its variables if both e_1, e_2 are continuous in their variables.

Proof

We have that $e_1(e_2) = apply(e_1, e_2)$, a composition of tupling and *apply*.

A Metalanguage

λ-Abstraction

- D, E: cpo's
- e: an expression that represents an element in E
- y: a variable

Then $\lambda y.e$ is continuous in its variables if the expression e is continuous in its variables.

Proof

For in all variables x, we have:

- if $x = y$ then $\lambda x.\lambda y.e$ is a constant function;

A Metalanguage

λ-Abstraction

- D, E: cpo's
- e: an expression that represents an element in E
- y: a variable

Then the expression $\lambda y.e$ is continuous in its variables if the expression e is continuous in its variables.

Proof

- if $x \neq y$ then

$$\begin{array}{ll} & \lambda x.\lambda y.e \text{ is continuous} \\ \Leftrightarrow & \mathit{curry}(\lambda x, y.e) \text{ is continuous} \\ \Leftarrow & \lambda x, y.e \text{ is continuous} \\ \Leftrightarrow & e \text{ is continuous in } x, y \end{array}$$

A Metalanguage

λ-Abstraction

- e_1, e_2: expressions
- $e_1 \circ e_2 := \lambda x.(e_1(e_2))$

Then $e_1 \circ e_2$ is continuous in its variables if both e_1, e_2 are continuous in their variables.

A Metalanguage

let-Construction

- D: a cpo
- E: a cpo with bottom
- e_1: a expression representing an element in $D_\perp$
- e_2: a expression representing an element in E

Then the expression

$$let\ x \Leftarrow e_1.e_2$$

is continuous in its variables if both e_1, e_2 are continuous.

Proof

We have $let\ x \Leftarrow e_1.e_2 = (\lambda x.e_2)^*(e_1)$.

A Metalanguage

case-Construction

- $E, D_1, \ldots, D_k$: cpo's
- e: an expression representing an element in $D_1 + \cdots + D_k$
- $e_1, \ldots, e_k$: expressions representing elements in E

Then the case expression

$$\begin{aligned} \textbf{case } e \textbf{ of } & in_1(x_1).e_1 \mid \\ & \vdots \\ & in_k(x_k).e_k \end{aligned}$$

is continuous if all $e, e_1, \ldots, e_k$ are continuous.

Proof

The case expression is defined to be $[\lambda x_1.e_1, \ldots, \lambda x_k.e_k](e)$.

A Metalanguage

Fixed-Point Operator

- D: a cpo with a bottom element $\bot$
- $\mathit{fix} : [D \to D] \to D$: the least-fixed-point operator $f \mapsto \mathit{fix}(f)$

Then fix is a continuous function (i.e. $\mathit{fix} \in [[D \to D] \to D]$).

Proof

We have

$$\mathit{fix} = \bigsqcup_{n \in \omega} (\lambda f. f^n(\bot))$$

where

$$\lambda f. \bot \sqsubseteq \lambda f. f(\bot) \sqsubseteq \cdots \sqsubseteq \lambda f. f^n(\bot) \sqsubseteq \dots \quad .$$

is an ω-chain of continuous functions in $[[D \to D] \to D]$ (why?).

Inductive Construction of Continuous Expressions

Fixed-Point Operator

- D: a cpo with a bottom element $\bot$
- $\mathit{fix} : [D \to D] \to D$: the least-fixed-point operator $f \mapsto \mathit{fix}(f)$
- e: an expression representing an element in D

We define $\mu x.e := \mathit{fix}(\lambda x.e)$.

Proposition

The fixed-point expression $\mu x.e$ is continuous in its variables if e is continuous in its variables.

Summary

A Metalanguage for Continuous Functions

- variables
- constants
- tupling
- application
- λ-abstraction
- *let*-construction
- case-construction
- fixed-point operator

Exercise

Problem

- D: a cpo with a bottom element $\bot$
- $\mathit{fix} : [D \to D] \to D$: the least-fixed-point operator $f \mapsto \mathit{fix}(f)$

Prove that *fix* is a continuous function (i.e. $\mathit{fix} \in [[D \to D] \to D]$).

Languages with higher types

Topics

Typed Languages

- a functional programming language
- eager operational semantics
- lazy operational semantics

A Basic Functional Language

Textbook, Page 183 – 186

A Basic Functional Language

Types

The types τ are generated from the grammar:

$$\tau ::= \mathbf{int} \mid \tau_1 * \tau_2 \mid \tau_1 \rightarrow \tau_2$$

- **int**: the basic type for integers
- $\tau_1 * \tau_2$: product type for ordered pairs (e.g., type $\mathbf{int} * \mathbf{int}$ for integer pairs $(0, 1), (-1, 2), \ldots$)
- $\tau_1 \rightarrow \tau_2$: function type from τ_1 to τ_2 (e.g., type $\mathbf{int} \rightarrow \mathbf{int}$ for functions from integers to integers)

A Basic Functional Language

Variables

- **Var** $= \{x, y, \ldots\}$: a set of variables
- **type**(x): the uniquely-fixed type for the variable x

We write $x : \tau$ to stress that **type**$(x) = \tau$.

A Basic Functional Language

Terms

The terms t are generated from the grammar:

$$
\begin{array}{rcl}
t & ::= & x \ (\text{variables}) \\
& | & n \ (\text{integer constants}) \\
& | & t_1 \bowtie t_2 \ (\bowtie \in \{+, -, \times\}) \ (\text{arithmetic operations}) \\
& | & \textbf{if } t_0 \textbf{ then } t_1 \textbf{ else } t_2 \\
& | & (t_1, t_2) \ (\text{ordered pairs}) \\
& | & \textbf{fst}(t) \ (\text{first entry of ordered pairs}) \\
& | & \textbf{snd}(t) \ (\text{second entry of ordered pairs}) \\
& | & \lambda x.t \ (\lambda\text{-abstraction}) \\
& | & (t_1 \ t_2) \ (\text{function application } t_1(t_2)) \\
& | & \textbf{let } x \Leftarrow t_1 \textbf{ in } t_2 \ (\text{let-notation } t_2[t_1/x]) \\
& | & \textbf{rec}y.(\lambda x.t) \ (\text{recursion})
\end{array}
$$

A Basic Functional Language

Example

- x : **int**
- y : **int** → **int**

"Legal" Terms:

- $\lambda x.(x+1)$
- $(\lambda x.(x+1), 2)$
- $(\lambda x.(x+1)\ 2)$
- $\textbf{rec}\, y.(\lambda x.\textbf{if}\ x\ \textbf{then}\ 1\ \textbf{else}\ x \times (y\ (x-1)))$

A Basic Functional Language

Example

- $x : \mathbf{int}$

"Illegal" Terms

- $(\lambda x.x) + 1$
- $(\lambda x.x) + (\lambda x.(x + 1))$

Typing Rules

Variables

$$\frac{}{x : \tau} \quad (\mathbf{type}(x) = \tau)$$

Typing Rules

Arithmetic Operations

$$\frac{}{n : \mathbf{int}} \; (n \in \mathbb{Z}) \qquad \frac{t_1 : \mathbf{int}, \;\; t_2 : \mathbf{int}}{t_1 \;\mathrm{op}\; t_2 : \mathbf{int}} \quad (\mathrm{op} \in \{+, -, \times\})$$

Typing Rules

Conditional Branch

$$\frac{t_0 : \textbf{int},\ \ t_1 : \tau,\ \ t_2 : \tau}{\textbf{if}\ t_0\ \textbf{then}\ t_1\ \textbf{else}\ t_2 : \tau}$$

Typing Rules

Products

$$\frac{t_1 : \tau_1, \;\; t_2 : \tau_2}{(t_1, t_2) : \tau_1 * \tau_2} \qquad \frac{t : \tau_1 * \tau_2}{\textbf{fst}(t) : \tau_1} \qquad \frac{t : \tau_1 * \tau_2}{\textbf{snd}(t) : \tau_2}$$

Typing Rules

Functions

$$\frac{x : \tau',\ \ t : \tau}{\lambda x.t : \tau' \to \tau} \qquad \frac{t_1 : \tau' \to \tau,\ \ t_2 : \tau'}{(t_1\ \ t_2) : \tau}$$

Typing Rules

“Let” Notation

$$\frac{x : \tau_1, \quad t_1 : \tau_1, \quad t_2 : \tau_2}{\textbf{let } x \Leftarrow t_1 \textbf{ in } t_2 : \tau_2}$$

Typing Rules

Recursion

$$\frac{y : \tau,\ \ \lambda x.t : \tau}{\mathbf{rec}\, y.(\lambda x.t) : \tau}$$

Typing Rules

Typable Terms

- A term t is typable if $t : \tau$ for some type τ.
- A term t is uniquely typable if $t : \tau$ for some unique type τ.
  (i.e., $t : \tau_1$ and $t : \tau_2$ implies $\tau_1 = \tau_2$)

Exercise

Every typable term is uniquely typable.

Free Variables $FV(-)$

Definition through Well-Founded Recursion

- $FV(n) := \emptyset$;
- $FV(x) := \{x\}$;
- $FV(t_1 \text{ op } t_2) := FV(t_1) \cup FV(t_2)$;
- $FV(\textbf{if } t_0 \textbf{ then } t_1 \textbf{ else } t_2) := FV(t_0) \cup FV(t_1) \cup FV(t_2)$;
- $FV((t_1,\ t_2)) = FV((t_1\ t_2)) := FV(t_1) \cup FV(t_2)$;
- $FV(\textbf{fst}(t)) = FV(\textbf{snd}(t)) := FV(t)$;
- $FV(\lambda x.t) := FV(t) \setminus \{x\}$;
- $FV(\textbf{let } x \Leftarrow t_1 \textbf{ in } t_2) := FV(t_1) \cup (FV(t_2) \setminus \{x\})$;
- $FV(\textbf{rec}y.(\lambda x.t)) := FV(\lambda x.t) \setminus \{y\}$.

Free Variables $FV(-)$

Closed Terms

A term t is closed if $FV(t) = \emptyset$.

Free Variables $FV(-)$

Substitution

- t: a term
- s: a closed term
- x: a free variable in t

Then we have

- $t\,[s/x]$: the term obtained from substituting all free occurrences of x by s in t
- $t[s_1/x_1, \ldots, s_k/x_k]$: the term obtained from substituting all free occurrences of x_i by closed terms s_i $(1 \leq i \leq k)$ in t

Free Variables $FV(-)$

Example

- $x : \textbf{int}$
- $t = \textbf{let } x \Leftarrow x \textbf{ in } (x + 1)$
- $t[4/x] = \textbf{let } x \Leftarrow 4 \textbf{ in } (x + 1)$

Eager Operational Semantics

Textbook, Page 186 – 188

Eager Operational Semantics

Ordered Pairs

- $t = (3 + 1, (\lambda x.(x + 1)\ 4))$

How can we evaluate **fst**(t) eagerly:

- first we evaluate both $3 + 1$ and $(\lambda x.(x + 1)\ 4)$;
- then the final result is the evaluation from $3 + 1$.

Eager Operational Semantics

Function Application

- $t_1 = \lambda x.1$;
- $t_2 = (\textbf{rec}\, y.(\lambda x.(y\ x))\ 4)$;
- $t = (t_1\ t_2)$

How can we evaluate t eagerly:

- first we evaluate both t_1 and t_2;
- then the final result is the function application.

Eager Operational Semantics

Canonical Forms (Values)

- τ: a type

The set $C_{\tau}^{\mathfrak{e}}$ of canonical forms of type τ is a subset of terms recursively defined as follows:

- $C_{\textbf{int}}^{\mathfrak{e}} := \mathbb{Z}$;
- $C_{\tau_1 * \tau_2}^{\mathfrak{e}} := C_{\tau_1}^{\mathfrak{e}} \times C_{\tau_2}^{\mathfrak{e}}$;
- $C_{\tau_1 \to \tau_2}^{\mathfrak{e}} := \{\lambda x.t \mid \lambda x.t : \tau_1 \to \tau_2 \text{ is closed.}\}$

Exercise

Prove that for any type τ and term $t \in C_{\tau}^{\mathfrak{e}}$, t is closed.

Eager Operational Semantics

The Evaluation Relation

- t: a typable closed term with type τ
- c: a canonical term in $C_{\tau}^{\mathfrak{e}}$

Then

- $t \rightarrow^{\mathfrak{e}} c$: t evaluates to c in eager operational semantics

Evaluation Rules

Canonical Forms

$$\frac{}{c \to^{\mathfrak{e}} c}\ (c \in C^{\mathfrak{e}}_{\tau})$$

Evaluation Rules

Arithmetic Operations

$$\frac{t_1 \rightarrow^{\mathfrak{e}} n_1, \;\; t_2 \rightarrow^{\mathfrak{e}} n_2}{(t_1 \;\text{op}\; t_2) \rightarrow^{\mathfrak{e}} n_1 \;\text{op}\; n_2} \;\; \text{op} \in \{+, -, \times\}$$

Evaluation Rules

Conditional Branch

$$\frac{t_0 \rightarrow^{\mathfrak{e}} 0,\ \ t_1 \rightarrow^{\mathfrak{e}} c_1}{\textbf{if } t_0 \textbf{ then } t_1 \textbf{ else } t_2 \rightarrow^{\mathfrak{e}} c_1}$$

$$\frac{t_0 \rightarrow^{\mathfrak{e}} n,\ \ t_2 \rightarrow^{\mathfrak{e}} c_2}{\textbf{if } t_0 \textbf{ then } t_1 \textbf{ else } t_2 \rightarrow^{\mathfrak{e}} c_2}\ (n \neq 0)$$

Evaluation Rules

Product

$$\frac{t_1 \to^{\mathfrak{e}} c_1, \;\; t_2 \to^{\mathfrak{e}} c_2}{(t_1, t_2) \to^{\mathfrak{e}} (c_1, c_2)}$$

$$\frac{t \to^{\mathfrak{e}} (c_1, c_2)}{\mathbf{fst}(t) \to^{\mathfrak{e}} c_1} \qquad \frac{t \to^{\mathfrak{e}} (c_1, c_2)}{\mathbf{snd}(t) \to^{\mathfrak{e}} c_2}$$

Evaluation Rules

Function Application

$$\frac{t_1 \rightarrow^{\mathfrak{e}} \lambda x.t_1',\ \ t_2 \rightarrow^{\mathfrak{e}} c_2,\ \ t_1'\,[c_2/x] \ \rightarrow^{\mathfrak{e}} c}{(t_1\ \ t_2) \rightarrow^{\mathfrak{e}} c}$$

Evaluation Rules

"Let" Expression

$$\frac{t_1 \rightarrow^{\mathfrak{e}} c_1, \;\; t_2\left[c_1/x\right] \rightarrow^{\mathfrak{e}} c_2}{\textbf{let } x \Leftarrow t_1 \textbf{ in } t_2 \rightarrow^{\mathfrak{e}} c_2}$$

Evaluation Rules

Recursion

$$\frac{}{\textbf{rec}\,y.(\lambda x.t) \to^{\mathfrak{e}} \lambda x.(t\,[\textbf{rec}\,y.(\lambda x.t)/y])}$$

Eager Operational Semantics

Proposition

- t: a closed term with type τ
- c, c': canonical forms

Then we have that:

- if $t \rightarrow^{\mathfrak{e}} c$ and $t \rightarrow^{\mathfrak{e}} c'$ then $c = c'$;
- if $t \rightarrow^{\mathfrak{e}} c$ then $c : \tau$.

Proof

By a simple rule induction.

Eager Operational Semantics

Homework

- $x : \textbf{int}$
- $y : \textbf{int} \to \textbf{int}$
- $fact := \textbf{rec}\, y.(\lambda x.\, (\textbf{if } x \textbf{ then } 1 \textbf{ else } x \times (y\ (x - 1))))$
- Find the type of *fact* through the typing rules.
- Evaluate $(fact\ 2)$ under the eager operational semantics.

Lazy Operational Semantics

Textbook, Page 200 – 202

Lazy Operational Semantics

- eager semantics: evaluate every sub-term
- lazy semantics: evaluate only necessary sub-terms

Lazy Operational Semantics

Terms

$$
\begin{array}{lcl}
t & ::= & x \ (\text{variables}) \\
 & | & n \ (\text{integer constants}) \\
 & | & t_1 \bowtie t_2 \ (\bowtie \in \{=, -, \times\}) \ (\text{arithmetic operations}) \\
 & | & \textbf{if } t_0 \textbf{ then } t_1 \textbf{ else } t_2 \\
 & | & (t_1, t_2) \ (\text{ordered pairs}) \\
 & | & \textbf{fst}(t) \ (\text{first entry of ordered pairs}) \\
 & | & \textbf{snd}(t) \ (\text{second entry of ordered pairs}) \\
 & | & \lambda x.t \ (\lambda\text{-abstraction}) \\
 & | & (t_1 \ t_2) \ (\text{function application } t_1(t_2)) \\
 & | & \textbf{let } x \Leftarrow t_1 \textbf{ in } t_2 \ (\text{let-notation } t_2[t_1/x])) \\
 & | & \textbf{rec} y.t \ (\text{recursion})
\end{array}
$$

Typing Rules

Recursion

$$\frac{y : \tau, \;\; t : \tau}{\mathbf{rec}\, y.t : \tau}$$

Typing Rules

Typable Terms

- A term t is *typable* if $t : \tau$ for some type τ.
- A term t is *uniquely typable* if $t : \tau$ for some unique type τ. (i.e., $t : \tau_1$ and $t : \tau_2$ implies $\tau_1 = \tau_2$)

A Simple Exercise

Every typable term is uniquely typable.

Terms

Free Variables

- $FV(\mathbf{rec} y.t) := FV(t) \setminus \{y\}$.
- A term t is *closed* if $FV(t) = \emptyset$.

Lazy Operational Semantics

Canonical Forms

- τ: a type

The set $C^{\mathfrak{l}}_{\tau}$ is recursively defined as follows:

- $C^{\mathfrak{l}}_{\textbf{int}} := \mathbb{Z}$;
- $C^{\mathfrak{l}}_{\tau_1 * \tau_2} := \{(t_1, t_2) \mid t_1 : \tau_1, t_2 : \tau_2$ and t_1, t_2 are closed.$\}$;
- $C^{\mathfrak{l}}_{\tau_1 \to \tau_2} := \{\lambda x.t \mid \lambda x.t : \tau_1 \to \tau_2$ is closed.$\}$

Lazy Operational Semantics

The Evaluation Relation

- t: a closed term with type τ
- c: a canonical term in $C^{\mathfrak{c}}_{\tau}$

Then

- $t \to^{\mathfrak{l}} c$: t evaluates to c in lazy operational semantics

Lazy Operational Semantics

Evaluation Rules

- canonical terms:

$$\frac{}{c \to^{\mathfrak{l}} c} \quad (c \in C^{\mathfrak{l}}_{\tau})$$

Lazy Operational Semantics

Evaluation Rules

- arithmetic operations:

$$\frac{t_1 \rightarrow^{\mathfrak{l}} n_1, \;\; t_2 \rightarrow^{\mathfrak{l}} n_2}{t_1 \ \mathrm{op}\ t_2 \ \rightarrow^{\mathfrak{l}} \ n_1 \ \mathrm{op}\ n_2} \;\; \mathrm{op} \in \{+, -, \times\}$$

Lazy Operational Semantics

Evaluation Rules

- conditional branch:

$$\frac{t_0 \to^{\mathfrak{l}} 0\ ,\ t_1 \to^{\mathfrak{l}} c_1}{\textbf{if}\ t_0\ \textbf{then}\ t_1\ \textbf{else}\ t_2 \to^{\mathfrak{l}} c_1}$$

$$\frac{t_0 \to^{\mathfrak{l}} n\ ,\ t_2 \to^{\mathfrak{l}} c_2}{\textbf{if}\ t_0\ \textbf{then}\ t_1\ \textbf{else}\ t_2 \to^{\mathfrak{l}} c_2}\ (n \neq 0)$$

Lazy Operational Semantics

Evaluation Rules

- Product:

$$\frac{t \rightarrow^{l} (t_1, t_2),\;\; t_1 \rightarrow^{l} c_1}{\mathbf{fst}(t) \rightarrow^{l} c_1} \qquad \frac{t \rightarrow^{l} (t_1, t_2),\;\; t_2 \rightarrow^{l} c_2}{\mathbf{snd}(t) \rightarrow^{l} c_2}$$

Lazy Operational Semantics

Evaluation Rules

- Function Application:

$$\frac{t_1 \ \rightarrow^{\mathfrak{l}} \ \lambda x.t_1' \ , t_1' \, [t_2/x] \rightarrow^{\mathfrak{l}} c}{(t_1 \ \ t_2) \rightarrow^{\mathfrak{l}} c}$$

Lazy Operational Semantics

Evaluation Rules

- "Let" Notation:

$$\frac{t_2\,[t_1/x] \rightarrow^{\mathfrak{l}} c}{\textbf{let } x \Leftarrow t_1 \textbf{ in } t_2 \rightarrow^{\mathfrak{l}} c}$$

Lazy Operational Semantics

Evaluation Rules

- Recursion:

$$\frac{t\,[\mathbf{rec}\,y.t/y] \to^{\mathfrak{l}} c}{\mathbf{rec}\,y.t \to^{\mathfrak{l}} c}$$

Lazy Operational Semantics

Proposition

- t: a closed term with type τ
- c, c': canonical terms

Then we have that:

- if $t \to^{\mathfrak{l}} c$ and $t \to^{\mathfrak{l}} c'$ then $c = c'$;
- if $t \to^{\mathfrak{l}} c$ then $c : \tau$.

Proof.

By a simple rule induction. ☐

Summary

Functional Programming Languages

- types and terms
- eager operational semantics
- lazy operational semantics

Exercise

Problem

- $x : \textbf{int}$
- $y : \textbf{int} \to \textbf{int}$
- $\mathit{fact} := \textbf{rec} y.(\lambda x.\,(\textbf{if } x \textbf{ then } 1 \textbf{ else } x \times (y\ (x - 1))))$
- Find the type of *fact* through the typing rules.
- Evaluate (*fact* 2) under the eager operational semantics.

Topics

- eager denotational semantics
- lazy denotational semantics

Eager Denotational Semantics

Textbook, Chapter 11.3

Eager Denotational Semantics

An Overview

- terms as functions from environments to values
- cpo's and continuous functions as mathematical backbone

Eager Denotational Semantics

Values

- τ: a type

The cpo $V^{\mathfrak{e}}_{\tau}$ of values associated with the type τ is recursively defined as follows:

- $V^{\mathfrak{e}}_{\textbf{int}} := \mathbb{Z}$ (discrete cpo);
- $V^{\mathfrak{e}}_{\tau_1 * \tau_2} := V^{\mathfrak{e}}_{\tau_1} \times V^{\mathfrak{e}}_{\tau_2}$ (product cpo);
- $V^{\mathfrak{e}}_{\tau_1 \to \tau_2} := [V^{\mathfrak{e}}_{\tau_1} \to (V^{\mathfrak{e}}_{\tau_2})_{\perp}]$ (function space);

Question

Why do we incorporate $\perp$ in the last definition?

Eager Denotational Semantics

Environments

- **Var**: the set of variables

An *environment* ρ is a function

$$\rho : \mathbf{Var} \to \bigcup\{V^{\mathfrak{e}}_{\tau} \mid \tau \text{ a type}\}$$

such that

$$\forall x \in \mathbf{Var}.(x : \tau \Rightarrow \rho(x) \in V^{\mathfrak{e}}_{\tau}) \ .$$

We denote by $\mathbf{Env}^{\mathfrak{e}}$ the set of environments under eager semantics.

Eager Denotational Semantics

Intuition

- t: a typable term with type τ
- $[\![t]\!]^{\mathfrak{e}} : \mathbf{Env}^{\mathfrak{e}} \to (V^{\mathfrak{e}}_{\tau})_{\perp}$: the denotational semantics of t

Eager Denotational Semantics

Well-Founded Recursion

- $[\![x]\!]^{\mathfrak{e}} := \lambda\rho.\lfloor\rho(x)\rfloor$
- $[\![n]\!]^{\mathfrak{e}} := \lambda\rho.\lfloor n\rfloor$

Eager Denotational Semantics

Well-Founded Recursion

- $[\![t_1 \textbf{ op } t_2]\!]^{\mathfrak{e}} := \lambda\rho.([\![t_1]\!]^{\mathfrak{e}}(\rho) \textbf{ op}_{\perp} [\![t_2]\!]^{\mathfrak{e}}(\rho))$

Eager Denotational Semantics

Well-Founded Recursion

- $[\![\textbf{if } t_0 \textbf{ then } t_1 \textbf{ else } t_2]\!]^{\mathfrak{e}} := \lambda\rho.\mathit{cond}([\![t_0]\!]^{\mathfrak{e}}(\rho), [\![t_1]\!]^{\mathfrak{e}}(\rho), [\![t_2]\!]^{\mathfrak{e}}(\rho))$
- Conditional (Chapter 9.3):

$$\mathit{cond}(z_0, z_1, z_2) := \begin{cases} z_1 & \text{if } z_0 = \lfloor 0 \rfloor \\ z_2 & \text{if } z_0 = \lfloor n \rfloor \text{ and } n \neq 0 \\ \bot & \text{if } z_0 = \bot \end{cases}$$

-

$$\begin{aligned}&[\![\textbf{if } t_0 \textbf{ then } t_1 \textbf{ else } t_2]\!]^{\mathfrak{e}} := \\ &\qquad \lambda\rho.(\mathit{let}\ n \Leftarrow [\![t_0]\!]^{\mathfrak{e}}(\rho).[[\![t_1]\!]^{\mathfrak{e}}(\rho), [\![t_2]\!]^{\mathfrak{e}}(\rho)](n))\end{aligned}$$

Eager Denotational Semantics

Well-Founded Recursion

- $[\![(t_1, t_2)]\!]^{\mathfrak{e}} := \lambda\rho.let\ v_1 \Leftarrow [\![t_1]\!]^{\mathfrak{e}}(\rho), v_2 \Leftarrow [\![t_2]\!]^{\mathfrak{e}}(\rho).\lfloor(v_1, v_2)\rfloor;$
- $[\![\mathbf{fst}(t)]\!]^{\mathfrak{e}} := \lambda\rho.let\ v \Leftarrow [\![t]\!]^{\mathfrak{e}}(\rho).\lfloor\pi_1(v)\rfloor$
- $[\![\mathbf{snd}(t)]\!]^{\mathfrak{e}} := \lambda\rho.let\ v \Leftarrow [\![t]\!]^{\mathfrak{e}}(\rho).\lfloor\pi_2(v)\rfloor$

Eager Denotational Semantics

Function Update (Chapter 9.3)

- $in_1 : \{x\} \to \textbf{Var}\text{: } x \mapsto x$
- $in_2 : \textbf{Var} \setminus \{x\} \to \textbf{Var}\text{: } y \mapsto y, y \neq x$

$$\begin{aligned}\rho[v/x] = \lambda y.\textbf{case } y \textbf{ of } & in_1(y_1).v \mid \\ & in_2(y_2).\rho(y_2)\end{aligned}$$

Eager Denotational Semantics

Well-Founded Recursion

- $[\![\lambda x.t]\!]^{\mathfrak{e}} := \lambda \rho.\lfloor \lambda v \in V^{\mathfrak{e}}_{\tau_1}.[\![t]\!]^{\mathfrak{e}}(\rho[v/x])\rfloor$ for $\lambda x.t : \tau_1 \to \tau_2$
- $[\![(t_1\ t_2)]\!]^{\mathfrak{e}} := \lambda \rho.let\ F \Leftarrow [\![t_1]\!]^{\mathfrak{e}}(\rho), v \Leftarrow [\![t_2]\!]^{\mathfrak{e}}(\rho).F(v)$

Eager Denotational Semantics

Well-Founded Recursion

- $[\![\textbf{let}\ x \Leftarrow t_1\ \textbf{in}\ t_2]\!]^{\mathfrak{e}} := \lambda\rho.\mathit{let}\ v \Leftarrow [\![t_1]\!]^{\mathfrak{e}}(\rho).[\![t_2]\!]^{\mathfrak{e}}(\rho\,[v/x])$
- $[\![\textbf{rec} y.(\lambda x.t)]\!]^{\mathfrak{e}} := \lambda\rho.\lfloor \mu F.(\lambda v.[\![t]\!]^{\mathfrak{e}}(\rho[v/x, F/y]))\rfloor$

Eager Denotational Semantics

Lemma

- t: a typable term
- ρ, ρ': two environments such that for all $x \in FV(t)$, we have $\rho(x) = \rho'(x)$

Then we have $[\![t]\!]^{\mathfrak{e}}(\rho) = [\![t]\!]^{\mathfrak{e}}(\rho')$.

Proof

A simple structural induction on t.

Eager Denotational Semantics

Substitution Lemma

- s: a typable closed term with type τ
- x: a variable with type τ
- t: a typable term with type τ'

If $[\![s]\!]^{\mathfrak{e}}(\rho) = \lfloor v \rfloor$, then we have that:

- $t\,[s/x] : \tau'$;
- $[\![t\,[s/x]]\!]^{\mathfrak{e}}(\rho) = [\![t]\!]^{\mathfrak{e}}(\rho\,[v/x])$.

Proof

By structural induction on t.

Eager Denotational Semantics

Lemma

- If $t : \tau$, then for all ρ we have $[\![t]\!]^{\mathfrak{e}}(\rho) \in (V^{\mathfrak{e}}_{\tau})_{\perp}$.
- If $c \in C^{\mathfrak{e}}_{\tau}$, then for all ρ we have $[\![c]\!]^{\mathfrak{e}}(\rho) \neq \perp$ (the bottom element of $(V^{\mathfrak{e}}_{\tau})_{\perp}$).

Proof

By structural induction.

Agreement of Eager Semantics

Textbook, Chapter 11.4

Agreement of Eager Semantics

A First Statement

- t: a closed typable term
- c: a canonical term

Then we may expect that $t \rightarrow^{\mathfrak{e}} c$ iff $[\![t]\!]^{\mathfrak{e}}(\rho) = [\![c]\!]^{\mathfrak{e}}(\rho)$.

Agreement of Eager Semantics

A Problem

- t: a closed typable term
- c: a canonical term

$[\![t]\!]^{\mathfrak{e}}(\rho) = [\![c]\!]^{\mathfrak{e}}(\rho) \Rightarrow t \rightarrow^{\mathfrak{e}} c$ may not hold.

Agreement of Eager Semantics

The Correct Theorem

- t: a closed typable term
- c: a canonical term

Then we have:

- $t \rightarrow^{\mathfrak{e}} c$ implies $[\![t]\!]^{\mathfrak{e}}(\rho) = [\![c]\!]^{\mathfrak{e}}(\rho)$;
- The two eager semantics agree on the convergence of t.

Agreement of Eager Semantics

Operational Convergence

- t: a typable closed term

We say that t is *operationally convergent*, denoted by $t{\downarrow}^{\mathfrak{e}}$, if it holds that $\exists c.t \to^{\mathfrak{e}} c$.

Agreement of Eager Semantics

Denotational Convergence

- t: a typable closed term with type τ

We say that t is *denotationally convergent*, denoted by $t\Downarrow^{\mathfrak{e}}$, if it holds that $\exists v \in V_\tau^{\mathfrak{e}}.[\![t]\!]^{\mathfrak{e}}(\rho) = \lfloor v \rfloor$.

Agreement of Eager Semantics

The Correct Theorem

- t: a closed typable term
- c: a canonical term

Then we have:

- $t \to^{\mathfrak{e}} c$ implies $[\![t]\!]^{\mathfrak{e}}(\rho) = [\![c]\!]^{\mathfrak{e}}(\rho)$;
- $t{\downarrow}^{\mathfrak{e}}$ iff $t{\Downarrow}^{\mathfrak{e}}$.

Agreement of Eager Semantics

A Corollary

- t: a closed typable term with type **int**

Then we have that $t \rightarrow^{\mathfrak{e}} n$ iff $[\![t]\!]^{\mathfrak{e}}(\rho) = \lfloor n \rfloor$.

Agreement of Eager Semantics

The Correct Theorem

- t: a closed typable term
- c: a canonical term

Then we have:

- $t \to^{\mathfrak{e}} c$ implies $[\![t]\!]^{\mathfrak{e}}(\rho) = [\![c]\!]^{\mathfrak{e}}(\rho)$;
- $t{\downarrow}^{\mathfrak{e}}$ iff $t{\Downarrow}^{\mathfrak{e}}$.

Main Task

How can we prove this theorem?

Agreement of Eager Semantics

Lemma

- t: a closed typable term
- c: a canonical term

If $t \to^{\mathfrak{e}} c$ then $\llbracket t \rrbracket^{\mathfrak{e}}(\rho) = \llbracket c \rrbracket^{\mathfrak{e}}(\rho)$ (for any environment ρ).

Proof.

By rule induction on evaluation of terms. $\square$

Agreement of Eager Semantics

The Rule for **fst**$(-)$

$$\frac{t \to^{\mathfrak{e}} (c_1,\ c_2)}{\mathbf{fst}(t) \to^{\mathfrak{e}} c_1}$$

- $[\![t]\!]^{\mathfrak{e}}(\rho) = [\![(c_1,\ c_2)]\!]^{\mathfrak{e}}(\rho)$;
- $[\![(c_1,\ c_2)]\!]^{\mathfrak{e}}(\rho) = \mathit{let}\ v_1 \Leftarrow [\![c_1]\!]^{\mathfrak{e}}(\rho), v_2 \Leftarrow [\![c_2]\!]^{\mathfrak{e}}(\rho).\lfloor (v_1, v_2) \rfloor$;
- $(c_1,\ c_2){\Downarrow}^{\mathfrak{e}}$;
- $\lfloor v_1 \rfloor = [\![c_1]\!]^{\mathfrak{e}}(\rho)$ and $\lfloor v_2 \rfloor = [\![c_2]\!]^{\mathfrak{e}}(\rho)$;
- $[\![\mathbf{fst}(t)]\!]^{\mathfrak{e}}(\rho) = \mathit{let}\ v \Leftarrow [\![t]\!]^{\mathfrak{e}}(\rho).\lfloor \pi_1(v) \rfloor = \lfloor v_1 \rfloor = [\![c_1]\!]^{\mathfrak{e}}(\rho)$.

Agreement of Eager Semantics

The Rule for Function Application

$$\frac{t_1 \ \rightarrow^{\mathfrak{e}} \ \lambda x.t_1', \ \ t_2 \ \rightarrow^{\mathfrak{e}} \ c_2, \ \ t_1'\,[c_2/x] \ \rightarrow^{\mathfrak{e}} \ c}{(t_1 \ \ t_2) \ \rightarrow^{\mathfrak{e}} \ c}$$

- $[\![t_1]\!]^{\mathfrak{e}}(\rho) = [\![\lambda x.t_1']\!]^{\mathfrak{e}}(\rho)$, $[\![t_2]\!]^{\mathfrak{e}}(\rho) = [\![c_2]\!]^{\mathfrak{e}}(\rho)$ and $[\![t_1'\,[c_2/x]]\!]^{\mathfrak{e}}(\rho) = [\![c]\!]^{\mathfrak{e}}(\rho)$;

$$\begin{array}{rcl}
[\![(t_1 \ \ t_2)]\!]^{\mathfrak{e}}(\rho) & = & \mathit{let}\ F \Leftarrow [\![t_1]\!]^{\mathfrak{e}}(\rho), v \Leftarrow [\![t_2]\!]^{\mathfrak{e}}(\rho).F(v) \\
& = & \mathit{let}\ F \Leftarrow [\![\lambda x.t_1']\!]^{\mathfrak{e}}(\rho), v \Leftarrow [\![t_2]\!]^{\mathfrak{e}}(\rho).F(v) \\
& = & \mathit{let}\ F \Leftarrow \lfloor \lambda v.[\![t_1']\!]^{\mathfrak{e}}(\rho\,[v/x]) \rfloor, v \Leftarrow [\![c_2]\!]^{\mathfrak{e}}(\rho).F(v) \\
& = & [\![t_1']\!]^{\mathfrak{e}}(\rho\,[v/x]) \text{ where } \lfloor v \rfloor = [\![c_2]\!]^{\mathfrak{e}}(\rho) \\
& = & [\![t_1'\,[c_2/x]]\!]^{\mathfrak{e}}(\rho) \\
& = & [\![c]\!]^{\mathfrak{e}}(\rho)
\end{array}$$

Agreement of Eager Semantics

The Rule for Recursion

$$\frac{}{\mathbf{rec}\,y.(\lambda x.t) \;\to^{\mathfrak{e}}\; \lambda x.(t\,[\mathbf{rec}\,y.(\lambda x.t)/y])}$$

- $[\![\mathbf{rec}\,y.(\lambda x.t)]\!]^{\mathfrak{e}}(\rho) = \lfloor F^* \rfloor$;
- $F^* = \mu F.(\lambda v.[\![t]\!]^{\mathfrak{e}}(\rho[v/x, F/y]))$;

$$\begin{array}{rcl}
[\![\lambda x.t\,[\mathbf{rec}\,y.(\lambda x.t)/y]]\!]^{\mathfrak{e}}(\rho) & = & [\![\lambda x.t]\!]^{\mathfrak{e}}(\rho\,[F^*/y]) \\
 & = & \lfloor \lambda v.[\![t]\!]^{\mathfrak{e}}(\rho[v/x, F^*/y]) \rfloor \\
 & = & \lfloor F^* \rfloor \\
 & = & [\![\mathbf{rec}\,y.(\lambda x.t)]\!]^{\mathfrak{e}}(\rho)
\end{array}$$

Agreement of Eager Semantics

Lemma

- t: a closed typable term
- c: a canonical term

If $t \rightarrow^{\mathfrak{e}} c$ then $[\![t]\!]^{\mathfrak{e}}(\rho) = [\![c]\!]^{\mathfrak{e}}(\rho)$ (for any environment ρ).

Corollary

- t: a closed typable term

Then we have that $t{\downarrow}^{\mathfrak{e}} \Rightarrow t{\Downarrow}^{\mathfrak{e}}$.

Agreement of Eager Semantics

The Difficult Part

- t: a closed typable term

Then we have that $t{\Downarrow^{\mathfrak{e}}} \Rightarrow t{\downarrow^{\mathfrak{e}}}$.

The First Attempt

Structural induction on t.

Agreement of Eager Semantics

The Case $t = (t_1 \ t_2)$

- induction hypothesis: $(t_1{\Downarrow}^{\mathfrak{e}} \Rightarrow t_1{\downarrow}^{\mathfrak{e}})$ & $(t_2{\Downarrow}^{\mathfrak{e}} \Rightarrow t_2{\downarrow}^{\mathfrak{e}})$

Then

- $t{\Downarrow}^{\mathfrak{e}}$
- $[\![t]\!]^{\mathfrak{e}}(\rho) = \mathit{let}\ F \Leftarrow [\![t_1]\!]^{\mathfrak{e}}(\rho), v \Leftarrow [\![t_2]\!]^{\mathfrak{e}}(\rho).F(v)$
- $t_1{\Downarrow}^{\mathfrak{e}}$ & $t_2{\Downarrow}^{\mathfrak{e}}$
- $t_1 \rightarrow^{\mathfrak{e}} \lambda x.t_1'$ & $t_2 \rightarrow^{\mathfrak{e}} c_2$
- $[\![t]\!]^{\mathfrak{e}}(\rho) = F(v)$ where $F = \lambda u.[\![t_1']\!]^{\mathfrak{e}}(\rho\,[u/x])$ and $\lfloor v \rfloor = [\![c_2]\!]^{\mathfrak{e}}(\rho)$
- $[\![t]\!]^{\mathfrak{e}}(\rho) = [\![t_1']\!]^{\mathfrak{e}}(\rho\,[v/x]) = [\![t_1'\,[c_2/x]]\!]^{\mathfrak{e}}(\rho)$
- $t_1'\,[c_2/x]{\downarrow}^{\mathfrak{e}}$ and $t_1'\,[c_2/x] \rightarrow^{\mathfrak{e}} c$
- $t \rightarrow^{\mathfrak{e}} c$

Agreement of Eager Semantics

Question

What's wrong with the proof?

Proof

- ...
- $t_1 \rightarrow^{\mathfrak{e}} \lambda x.t_1' \;\&\; t_2 \rightarrow^{\mathfrak{e}} c_2$
- ...
- $[\![t]\!]^{\mathfrak{e}}(\rho) = [\![t_1']\!]^{\mathfrak{e}}(\rho\,[v/x]) = [\![t_1'\,[c_2/x]]\!]^{\mathfrak{e}}(\rho)$
- $t_1'\,[c_2/x]{\downarrow^{\mathfrak{e}}}$ and $t_1'\,[c_2/x] \rightarrow^{\mathfrak{e}} c$??
- $t \rightarrow^{\mathfrak{e}} c$

Agreement of Eager Semantics

Question

What's wrong with the proof?

The Problem

- It is not guaranteed that $t_1'[c_2/x]\Downarrow^{\mathfrak{e}} \Rightarrow t_1'[c_2/x]\downarrow^{\mathfrak{e}}$.

Agreement of Eager Semantics

Solution

- a stronger induction hypothesis
- a *logical relation* between values and types

Agreement of Eager Semantics

Logical Relations

- τ: a type

Then we will define:

- $\lesssim_{\tau}^{\circ} \subseteq V_{\tau}^{\mathfrak{e}} \times C_{\tau}^{\mathfrak{e}}$
- $\lesssim_{\tau} \subseteq (V_{\tau}^{\mathfrak{e}})_{\perp} \times \mathit{ClosedTerms}$

Agreement of Eager Semantics

The Relation $\lesssim_\tau$

- τ: a type

We define the relation $\lesssim_\tau \subseteq (V^{\mathfrak{e}}_\tau)_\perp \times \mathit{ClosedTerms}$ by:

$$d \lesssim_\tau t \text{ iff } \forall v \in V^{\mathfrak{e}}_\tau . [d = \lfloor v \rfloor \Rightarrow (\exists c.(t \to^{\mathfrak{e}} c \;\&\; v \lesssim^\circ_\tau c))]$$

Agreement of Eager Semantics

The Relations $\lesssim^\circ_\tau \subseteq V^\mathfrak{e}_\tau \times C^\mathfrak{e}_\tau$

- ground types:

$$n \lesssim^\circ_{\mathbf{int}} n \text{ for all integers } n$$

- product types:

$$(v_1, v_2) \lesssim^\circ_{\tau_1 * \tau_2} (c_1, c_2) \text{ iff } v_1 \lesssim^\circ_{\tau_1} c_1 \text{ and } v_2 \lesssim^\circ_{\tau_2} c_2$$

- function types:

$$F \lesssim^\circ_{\tau_1 \to \tau_2} \lambda x.t \text{ iff } \forall v \in V^\mathfrak{e}_{\tau_1}, c \in C^\mathfrak{e}_{\tau_1}. v \lesssim^\circ_{\tau_1} c \Rightarrow F(v) \lesssim_{\tau_2} t\,[c/x]$$

Agreement of Eager Semantics

Lemma

- t: a closed term with type τ

We have that

- $\bot_{(V_\tau^{\mathfrak{e}})_\bot} \lesssim_\tau t$;
- for any $d, d' \in (V_\tau^{\mathfrak{e}})_\bot$, it holds that

$$(d \sqsubseteq d' \ \&\ d' \lesssim_\tau t \Rightarrow d \lesssim_\tau t)\,;$$

- for any $d_0 \sqsubseteq d_1 \sqsubseteq \cdots \sqsubseteq d_n \sqsubseteq \ldots$ in $(V_\tau^{\mathfrak{e}})_\bot$, it holds that

$$(\forall n. d_n \lesssim_\tau t) \Rightarrow \bigsqcup_{n\in\omega} d_n \lesssim_\tau t.$$

Agreement of Eager Semantics

Lemma

- $\perp_{(V_\tau^{\mathfrak{e}})_\perp} \lesssim_\tau t$;

Proof

By definition:

$$d \lesssim_\tau t \text{ iff } \forall v \in V_\tau^{\mathfrak{e}}. [d = \lfloor v \rfloor \Rightarrow (\exists c.(t \to^{\mathfrak{e}} c \;\&\; v \lesssim_\tau^{\circ} c))]$$

Agreement of Eager Semantics

Lemma

- for any $d_0 \sqsubseteq d_1 \sqsubseteq \cdots \sqsubseteq d_n \sqsubseteq \ldots$ in $(V_\tau^{\mathfrak{e}})_\perp$, it holds that

$$(\forall n. d_n \lesssim_\tau t) \Rightarrow \bigsqcup_{n\in\omega} d_n \lesssim_\tau t.$$

Proof (Structural Induction on Types)

base type: $\tau = \textbf{int}$. Straightforward.

Agreement of Eager Semantics

Lemma

- for any $d_0 \sqsubseteq d_1 \sqsubseteq \cdots \sqsubseteq d_n \sqsubseteq \ldots$ in $(V^{\mathfrak{e}}_{\tau})_{\perp}$, it holds that

$$(\forall n.d_n \lesssim_{\tau} t) \Rightarrow \bigsqcup_{n\in\omega} d_n \lesssim_{\tau} t.$$

Proof (Structural Induction on Types)

function types: $\tau = \tau_1 \to \tau_2$.

- Suppose $d_0 \sqsubseteq d_1 \sqsubseteq \cdots \sqsubseteq d_n \sqsubseteq \ldots$ in $(V^{\mathfrak{e}}_{\tau_1\to\tau_2})_{\perp}$.
- Suppose that $\forall n.d_n \lesssim_{\tau_1\to\tau_2} t$.
- easy case: $\forall n.d_n = \perp$

Agreement of Eager Semantics

Proof (Structural Induction on Types)

function types: $\tau = \tau_1 \to \tau_2$.

- Suppose that $d_0 \sqsubseteq d_1 \sqsubseteq \dots$ in $(V^{\mathfrak{e}}_{\tau_1 \to \tau_2})_\bot$ and $\forall n. d_n \lesssim_{\tau_1 \to \tau_2} t$.
- nontrivial case: $\exists n. d_n \neq \bot$ and $t \to^{\mathfrak{e}} \lambda x.t'$.
- from definition: $\forall n \geq N. (d_n = \lfloor F_n \rfloor \;\&\; F_n \lesssim^{\circ}_{\tau_1 \to \tau_2} \lambda x.t')$
- $\forall n \geq N. \forall (v, c) \in V^{\mathfrak{e}}_{\tau_1} \times C^{\mathfrak{e}}_{\tau_1}. v \lesssim^{\circ}_{\tau_1} c \Rightarrow F_n(v) \lesssim_{\tau_2} t' \, [c/x]$
- induction hypothesis: $(\bigsqcup_n F_n)(v) = \bigsqcup_n (F_n(v)) \lesssim_{\tau_2} t' \, [c/x]$
- $\forall (v, c). v \lesssim^{\circ}_{\tau_1} c \Rightarrow (\bigsqcup_n F_n)(v) \lesssim_{\tau_2} t' \, [c/x]$
- $\bigsqcup_{n \in \omega} F_n \lesssim^{\circ}_{\tau_1 \to \tau_2} \lambda x.t'$
- $\bigsqcup_{n \in \omega} d_n = \lfloor \bigsqcup_{n \in \omega} F_n \rfloor \lesssim_{\tau_1 \to \tau_2} t$

Agreement of Eager Semantics

Lemma

- t: a typable close term

We have that $t{\Downarrow}^{\mathfrak{e}} \Rightarrow t{\downarrow}^{\mathfrak{e}}$.

Proof (by Structural Induction)

We prove by structural induction on terms that:

- $t : \tau$: a term
- $x_1 : \tau_1, \ldots, x_k : \tau_k$: free variables in t
- $s_1 : \tau_1, \ldots, s_k : \tau_k$: closed terms
- $v_i \in V^{\mathfrak{e}}_{\tau_i}$ $(1 \leq i \leq k)$: elements such that $\lfloor v_i \rfloor \lesssim_{\tau_i} s_i$

Then $[\![t]\!]^{\mathfrak{e}}(\rho[v_1/x_1, \ldots, v_k/x_k]) \lesssim_\tau t[s_1/x_1, \ldots, s_k/x_k]$.

Agreement of Eager Semantics

Proof (by Structural Induction)

We prove by structural induction on terms that:

- $t : \tau$: a term
- $x_1 : \tau_1, \ldots, x_k : \tau_k$: free variables in t
- $s_1 : \tau_1, \ldots, s_k : \tau_k$: closed terms
- $v_i \in V^{\mathfrak{e}}_{\tau_i}$ $(1 \leq i \leq k)$: elements such that $\lfloor v_i \rfloor \lesssim_{\tau_i} s_i$

Then $[\![t]\!]^{\mathfrak{e}}(\rho[v_1/x_1, \ldots, v_k/x_k]) \lesssim_{\tau} t[s_1/x_1, \ldots, s_k/x_k]$.

Agreement of Eager Semantics

Proof (by Structural Induction)

- $t : \tau$: a term
- $x_1 : \tau_1, \ldots, x_k : \tau_k$: free variables in t
- $s_1 : \tau_1, \ldots, s_k : \tau_k$: closed terms
- $v_i \in V^{\mathfrak{e}}_{\tau_i}$ $(1 \leq i \leq k)$: elements such that $\lfloor v_i \rfloor \lesssim_{\tau_i} s_i$

Then $[\![t]\!]^{\mathfrak{e}}(\rho[v_1/x_1, \ldots, v_k/x_k]) \lesssim_\tau t[s_1/x_1, \ldots, s_k/x_k]$.

Base Step: $t = x$ and $x : \tau$

- Suppose $\lfloor v \rfloor \lesssim_\tau s$.
- $[\![x]\!]^{\mathfrak{e}}(\rho\,[v/x]) = \lfloor v \rfloor \lesssim_\tau s = x\,[s/x]$.

Agreement of Eager Semantics

Proof (by Structural Induction)

- $t : \tau$: a term
- $x_1 : \tau_1, \ldots, x_k : \tau_k$: free variables in t
- $s_1 : \tau_1, \ldots, s_k : \tau_k$: closed terms
- $v_i \in V^{\mathfrak{e}}_{\tau_i}$ $(1 \leq i \leq k)$: elements such that $\lfloor v_i \rfloor \lesssim_{\tau_i} s_i$

Then $[\![t]\!]^{\mathfrak{e}}(\rho[v_1/x_1, \ldots, v_k/x_k]) \lesssim_{\tau} t[s_1/x_1, \ldots, s_k/x_k]$.

Base Step: $t = n$

- $n \lesssim^{\circ}_{\mathbf{int}} n$.

Agreement of Eager Semantics

Proof (by Structural Induction)

- $[\![t]\!]^{\mathfrak{e}}(\rho[v_1/x_1,\ldots,v_k/x_k]) \lesssim_\tau t[s_1/x_1,\ldots,s_k/x_k]$.

Inductive Step: $t = t_1$ **op** t_2

- Suppose that $[\![t_1 \textbf{ op } t_2]\!]^{\mathfrak{e}}(\rho[v_1/x_1,\ldots,v_k/x_k]) = \lfloor n \rfloor$.
- Then
  - $[\![t_1]\!]^{\mathfrak{e}}(\rho[v_1/x_1,\ldots,v_k/x_k]) = \lfloor n_1 \rfloor$.
  - $[\![t_2]\!]^{\mathfrak{e}}(\rho[v_1/x_1,\ldots,v_k/x_k]) = \lfloor n_2 \rfloor$.
  - $n = n_1 \textbf{ op } n_2$.
- By induction hypothesis,
  - $\lfloor n_1 \rfloor \lesssim_{\textbf{int}} t_1[v_1/x_1,\ldots,v_k/x_k]$
  - $\lfloor n_2 \rfloor \lesssim_{\textbf{int}} t_2[v_1/x_1,\ldots,v_k/x_k]$

Agreement of Eager Semantics

Proof (by Structural Induction)

- $[\![t]\!]^{\mathfrak{e}}(\rho[v_1/x_1,\ldots,v_k/x_k]) \lesssim_\tau t[s_1/x_1,\ldots,s_k/x_k]$.

Inductive Step: $t = t_1 \ \mathbf{op}\ t_2$

- By induction hypothesis,
  - $\lfloor n_1 \rfloor \lesssim_{\mathbf{int}} t_1[v_1/x_1,\ldots,v_k/x_k]$
  - $\lfloor n_2 \rfloor \lesssim_{\mathbf{int}} t_2[v_1/x_1,\ldots,v_k/x_k]$
- From the definition of $\lesssim_{\mathbf{int}}$,
  - $t_1[v_1/x_1,\ldots,v_k/x_k] \rightarrow^{\mathfrak{e}} n_1$
  - $t_2[v_1/x_1,\ldots,v_k/x_k] \rightarrow^{\mathfrak{e}} n_2$
- Hence, $t[v_1/x_1,\ldots,v_k/x_k] \rightarrow^{\mathfrak{e}} n$.
- Finally, $[\![t]\!]^{\mathfrak{e}}(\rho[v_1/x_1,\ldots,v_k/x_k]) \lesssim_\tau t[s_1/x_1,\ldots,s_k/x_k]$.

Agreement of Eager Semantics

Proof (by Structural Induction)

- $[\![t]\!]^{\mathfrak{e}}(\rho[v_1/x_1,\ldots,v_k/x_k]) \lesssim_\tau t[s_1/x_1,\ldots,s_k/x_k]$.

Inductive Step: $t = \textbf{if } t_0 \textbf{ then } t_1 \textbf{ else } t_2$

- Suppose that $[\![t]\!]^{\mathfrak{e}}(\rho[v_1/x_1,\ldots,v_k/x_k]) = \lfloor u \rfloor$.
- Then either
  - $[\![t_0]\!]^{\mathfrak{e}}(\rho[v_1/x_1,\ldots,v_k/x_k]) = \lfloor 0 \rfloor$.
  - $[\![t_1]\!]^{\mathfrak{e}}(\rho[v_1/x_1,\ldots,v_k/x_k]) = \lfloor u_1 \rfloor$.

  or
  - $[\![t_0]\!]^{\mathfrak{e}}(\rho[v_1/x_1,\ldots,v_k/x_k]) = \lfloor n \rfloor$ $(n > 0)$.
  - $[\![t_2]\!]^{\mathfrak{e}}(\rho[v_1/x_1,\ldots,v_k/x_k]) = \lfloor u_2 \rfloor$.
- ...

Agreement of Eager Semantics

Proof (by Structural Induction)

- $[\![t]\!]^{\mathfrak{e}}(\rho[v_1/x_1,\ldots,v_k/x_k]) \lesssim_\tau t[s_1/x_1,\ldots,s_k/x_k]$.

Inductive Step: $t = \textbf{if } t_0 \textbf{ then } t_1 \textbf{ else } t_2$

- Suppose that $[\![t]\!]^{\mathfrak{e}}(\rho[v_1/x_1,\ldots,v_k/x_k]) = \lfloor u \rfloor$.
- Then either
  - $[\![t_0]\!]^{\mathfrak{e}}(\rho[v_1/x_1,\ldots,v_k/x_k]) = \lfloor 0 \rfloor$.
  - $[\![t_1]\!]^{\mathfrak{e}}(\rho[v_1/x_1,\ldots,v_k/x_k]) = \lfloor u_1 \rfloor$.

  or
  - $[\![t_0]\!]^{\mathfrak{e}}(\rho[v_1/x_1,\ldots,v_k/x_k]) = \lfloor n \rfloor$ $(n > 0)$.
  - $[\![t_2]\!]^{\mathfrak{e}}(\rho[v_1/x_1,\ldots,v_k/x_k]) = \lfloor u_2 \rfloor$.
- ...

Agreement of Eager Semantics

Proof (by Structural Induction)

- $[\![t]\!]^{\mathfrak{e}}(\rho[v_1/x_1,\ldots,v_k/x_k]) \lesssim_{\tau} t[s_1/x_1,\ldots,s_k/x_k]$.

Inductive Step: $t = (t_1, t_2)$

- Suppose $[\![t]\!]^{\mathfrak{e}}(\rho[v_1/x_1,\ldots,v_k/x_k]) = \lfloor u \rfloor$ and $t_1 : \tau_1, t_2 : \tau_2$.
- Then
  - $[\![t_1]\!]^{\mathfrak{e}}(\rho[v_1/x_1,\ldots,v_k/x_k]) = \lfloor u_1 \rfloor$.
  - $[\![t_2]\!]^{\mathfrak{e}}(\rho[v_1/x_1,\ldots,v_k/x_k]) = \lfloor u_2 \rfloor$.
  - $u = (u_1, u_2)$.
- By induction hypothesis,
  - $\lfloor u_1 \rfloor \lesssim_{\tau_1} t_1[s_1/x_1,\ldots,s_k/x_k]$;
  - $\lfloor u_2 \rfloor \lesssim_{\tau_2} t_2[s_1/x_1,\ldots,s_k/x_k]$.

Agreement of Eager Semantics

Proof (by Structural Induction)

- $[\![t]\!]^{\mathfrak{e}}(\rho[v_1/x_1,\ldots,v_k/x_k]) \lesssim_\tau t[s_1/x_1,\ldots,s_k/x_k]$.

Inductive Step: $t = (t_1, t_2)$

- By induction hypothesis,
  - $\lfloor u_1 \rfloor \lesssim_{\tau_1} t_1[s_1/x_1,\ldots,s_k/x_k]$;
  - $\lfloor u_2 \rfloor \lesssim_{\tau_2} t_2[s_1/x_1,\ldots,s_k/x_k]$.
- By definition,
  - $t_1[s_1/x_1,\ldots,s_k/x_k] \to^{\mathfrak{e}} c_1$ and $u_1 \lesssim^{\circ}_{\tau_1} c_1$;
  - $t_2[s_1/x_1,\ldots,s_k/x_k] \to^{\mathfrak{e}} c_2$ and $u_2 \lesssim^{\circ}_{\tau_2} c_2$.
- Then $(u_1, u_2) \lesssim^{\circ}_{\tau} (c_1, c_2)$ and $t[s_1/x_1,\ldots,s_k/x_k] \to^{\mathfrak{e}} (c_1, c_2)$.
- Finally, $[\![t]\!]^{\mathfrak{e}}(\rho[v_1/x_1,\ldots,v_k/x_k]) \lesssim_\tau t[s_1/x_1,\ldots,s_k/x_k]$

Agreement of Eager Semantics

Proof (by Structural Induction)

- $[\![t]\!]^{\mathfrak{e}}(\rho[v_1/x_1,\ldots,v_k/x_k]) \lesssim_\tau t[s_1/x_1,\ldots,s_k/x_k]$.

Inductive Step: $t = \mathbf{fst}(t')$

- Suppose that $[\![t]\!]^{\mathfrak{e}}(\rho[v_1/x_1,\ldots,v_k/x_k]) = \lfloor u \rfloor$ and $t' : \tau_1 * \tau_2$;
- Then $[\![t']\!]^{\mathfrak{e}}(\rho[v_1/x_1,\ldots,v_k/x_k]) = \lfloor (u_1,u_2) \rfloor$ and $u = u_1$;
- By induction hypothesis, $\lfloor (u_1,u_2) \rfloor \lesssim_{\tau_1 * \tau_2} t'[s_1/x_1,\ldots,s_k/x_k]$;
- Hence, $t'[s_1/x_1,\ldots,s_k/x_k] \rightarrow^{\mathfrak{e}} (c_1,c_2)$ and $(u_1,u_2) \lesssim^{\circ}_{\tau_1 * \tau_2} (c_1,c_2)$;
- Then, $t[s_1/x_1,\ldots,s_k/x_k] \rightarrow^{\mathfrak{e}} c_1$ and $u_1 \lesssim^{\circ}_{\tau_1} c_1$;
- Finally, $[\![t]\!]^{\mathfrak{e}}(\rho[v_1/x_1,\ldots,v_k/x_k]) \lesssim_\tau t[s_1/x_1,\ldots,s_k/x_k]$.

Agreement of Eager Semantics

Proof (by Structural Induction)

- $[\![t]\!]^{\mathfrak{e}}(\rho[v_1/x_1, \ldots, v_k/x_k]) \lesssim_\tau t[s_1/x_1, \ldots, s_k/x_k]$.

Inductive Step: $t = \mathbf{snd}(t')$

By similar proof.

Agreement of Eager Semantics

Proof (by Structural Induction)

- $[\![t]\!]^{\mathfrak{e}}(\rho[v_1/x_1,\ldots,v_k/x_k]) \lesssim_{\tau} t[s_1/x_1,\ldots,s_k/x_k]$.

Inductive Step: $t = \lambda x.t'$ with $x:\tau$, $t':\tau'$

- $[\![\lambda x.t']\!]^{\mathfrak{e}}(\rho[v_1/x_1,\ldots,v_k/x_k]) = \lfloor F \rfloor$.
- Then $F = \lambda v \in V^{\mathfrak{e}}_{\tau}.[\![t]\!]^{\mathfrak{e}}(\rho[v_1/x_1,\ldots,v_k/x_k,v/x])$
- For any v, c such that $v \lesssim^{\circ}_{\tau} c$, $\lfloor v \rfloor \lesssim_{\tau} c$ and $F(v) = [\![t']\!]^{\mathfrak{e}}(\rho[v_1/x_1,\ldots,v_k/x_k,v/x])$
- By induction hypothesis,

$$[\![t']\!]^{\mathfrak{e}}(\rho[v_1/x_1,\ldots,v_k/x_k,v/x]) \lesssim_{\tau'} t'[s_1/x_1,\ldots,s_k/x_k,c/x]$$

- $F \lesssim^{\circ}_{\tau\to\tau'} \lambda x.t'[s_1/x_1,\ldots,s_k/x_k]$
- $\lfloor F \rfloor \lesssim^{\circ}_{\tau\to\tau'} t[s_1/x_1,\ldots,s_k/x_k]$

Agreement of Eager Semantics

Proof (by Structural Induction)

- $[\![t]\!]^{\mathfrak{e}}(\rho[v_1/x_1,\ldots,v_k/x_k]) \lesssim_\tau t[s_1/x_1,\ldots,s_k/x_k]$.

Inductive Step: $t=(t_1\ t_2)$ with $t_1:\tau'\to\tau$, $t_2:\tau'$

- Suppose that $[\![(t_1\ t_2)]\!]^{\mathfrak{e}}(\rho[v_1/x_1,\ldots,v_k/x_k]) = \lfloor u\rfloor$.
- Then we have
  - $[\![t_1]\!]^{\mathfrak{e}}(\rho[v_1/x_1,\ldots,v_k/x_k]) = \lfloor F\rfloor$.
  - $[\![t_2]\!]^{\mathfrak{e}}(\rho[v_1/x_1,\ldots,v_k/x_k]) = \lfloor v\rfloor$.
  - $F(v)=\lfloor u\rfloor$.
- By induction hypothesis,
  - $\lfloor F\rfloor \lesssim_{\tau'\to\tau} t_1[s_1/x_1,\ldots,s_k/x_k]$.
  - $\lfloor v\rfloor \lesssim_{\tau'} t_2[s_1/x_1,\ldots,s_k/x_k]$.

Agreement of Eager Semantics

Proof (by Structural Induction)

- $[\![t]\!]^{\mathfrak{e}}(\rho[v_1/x_1, \ldots, v_k/x_k]) \lesssim_\tau t[s_1/x_1, \ldots, s_k/x_k]$.

Inductive Step: $t = (t_1\ t_2)$ with $t_1 : \tau' \to \tau''$, $t_2 : \tau'$

- By induction hypothesis,
  - $\lfloor F \rfloor \lesssim_{\tau' \to \tau} t_1[s_1/x_1, \ldots, s_k/x_k]$.
  - $\lfloor v \rfloor \lesssim_{\tau'} t_2[s_1/x_1, \ldots, s_k/x_k]$.
- By definition,
  - $t_1[s_1/x_1, \ldots, s_k/x_k] \to^{\mathfrak{e}} \lambda x.t_1'$ and $F \lesssim^{\circ}_{\tau' \to \tau} \lambda x.t_1'$;
  - $t_2[s_1/x_1, \ldots, s_k/x_k] \to^{\mathfrak{e}} c_2$ and $v \lesssim^{\circ}_{\tau'} c_2$.
- From $F \lesssim^{\circ}_{\tau' \to \tau} \lambda x.t_1'$, we have $\lfloor u \rfloor = F(v) \lesssim_\tau t_1'[c_2/x]$.
- Then, there is $c \in C^{\mathfrak{e}}_\tau$ such that $t_1'[c_2/x] \to^{\mathfrak{e}} c$ and $u \lesssim^{\circ}_\tau c$;
- $(t_1\ t_2)[s_1/x_1, \ldots, s_k/x_k] \to^{\mathfrak{e}} c$;
- Finally, $[\![t]\!]^{\mathfrak{e}}(\rho[v_1/x_1, \ldots, v_k/x_k]) \lesssim_\tau t[s_1/x_1, \ldots, s_k/x_k]$

Agreement of Eager Semantics

Proof (by Structural Induction)

- $[\![t]\!]^{\mathfrak{e}}(\rho[v_1/x_1,\ldots,v_k/x_k]) \lesssim_{\tau} t[s_1/x_1,\ldots,s_k/x_k]$.

Inductive Step: $t = \mathit{let}\ x \Leftarrow t_1.t_2$ with $x:\tau_1,\ t_1:\tau_1,\ t_2:\tau$

- Suppose that $[\![t]\!]^{\mathfrak{e}}(\rho[v_1/x_1,\ldots,v_k/x_k]) = \lfloor u \rfloor$
- Then there is $u_1 \in V^{\mathfrak{e}}_{\tau_1}$ such that
  - $[\![t_1]\!]^{\mathfrak{e}}(\rho[v_1/x_1,\ldots,v_k/x_k]) = \lfloor u_1 \rfloor$;
  - $[\![t_2]\!]^{\mathfrak{e}}(\rho[v_1/x_1,\ldots,v_k/x_k][u_1/x]) = \lfloor u \rfloor$;
- From induction hypothesis, there are canonical forms c_1, c such that
  - $u_1 \lesssim^{\circ}_{\tau_1} c_1$ and $t_1[s_1/x_1,\ldots,s_k/x_k] \rightarrow^{\mathfrak{e}} c_1$
  - $u \lesssim^{\circ}_{\tau_1} c$ and $t_2[s_1/x_1,\ldots,s_k/x_k][c_1/x] \rightarrow^{\mathfrak{e}} c$
- Thus, $t[s_1/x_1,\ldots,s_k/x_k] \rightarrow^{\mathfrak{e}} c$.

Agreement of Eager Semantics

Proof (by Structural Induction)

- $[\![t]\!]^{\mathfrak{e}}(\rho[v_1/x_1,\ldots,v_k/x_k]) \lesssim_\tau t[s_1/x_1,\ldots,s_k/x_k]$.

Inductive Step: $t = \mathbf{rec}\,y.(\lambda x.t')$ with $x : \tau''$, $t' : \tau'$

- Suppose that $[\![t]\!]^{\mathfrak{e}}(\rho[v_1/x_1,\ldots,v_k/x_k]) = \lfloor G \rfloor$ for $G \in V^{\mathfrak{e}}_{\tau''\to\tau'}$
- $G = \mu F.(\lambda v.[\![t']\!]^{\mathfrak{e}}(\rho[v_1/x_1,\ldots,v_k/x_k,v/x,F/y]))$
- $G = \bigsqcup_{n\in\omega} G_n = \bigcup_{n\in\omega} G_n$ where
  - $G_0 := \bot_{V^{\mathfrak{e}}_{\tau''\to\tau'}}$
  - $G_{n+1} := \lambda v.[\![t']\!]^{\mathfrak{e}}(\rho[v_1/x_1,\ldots,v_k/x_k,v/x,G_n/y])$
- By induction: for all n, $G_n \lesssim^{\circ}_{\tau''\to\tau'} (\lambda x.t')[\mathbf{s}/\mathbf{x}, t[\mathbf{s}/\mathbf{x}]/y]$
- Then $G \lesssim^{\circ}_{\tau''\to\tau'} (\lambda x.t')[\mathbf{s}/\mathbf{x}, t[\mathbf{s}/\mathbf{x}]/y]$.
- Note that $t[s_1/x_1,\ldots,s_k/x_k] \to^{\mathfrak{e}} (\lambda x.t')[\mathbf{s}/\mathbf{x}, t[\mathbf{s}/\mathbf{x}]/y]$.
- Thus, $\lfloor G \rfloor \lesssim^{\circ}_{\tau''\to\tau'} t[s_1/x_1,\ldots,s_k/x_k]$.

Agreement of Eager Semantics

Proof (by Structural Induction)

- $[\![t]\!]^{\mathfrak{e}}(\rho[v_1/x_1,\ldots,v_k/x_k]) \lesssim_{\tau} t[s_1/x_1,\ldots,s_k/x_k]$.

Inductive Step: $t = \mathbf{rec}\, y.(\lambda x.t')$ with $x : \tau''$, $t' : \tau'$

- $G_0 := \bot_{V^{\mathfrak{e}}_{\tau''\to\tau'}}$ and $G_{n+1} := \lambda v.[\![t']\!]^{\mathfrak{e}}(\rho[\vec{\mathbf{v}}/\vec{\mathbf{x}}, v/x, G_n/y])$
- By induction: for all n, $G_n \lesssim^{\circ}_{\tau''\to\tau'} (\lambda x.t')[\mathbf{s}/\mathbf{x}, t[\mathbf{s}/\mathbf{x}]/y]$
- Base Step: $\bot_{V^{\mathfrak{e}}_{\tau''\to\tau'}} \lesssim^{\circ}_{\tau''\to\tau'} (\lambda x.t')[\mathbf{s}/\mathbf{x}, t[\mathbf{s}/\mathbf{x}]/y]$
- For all v, c such that $v \lesssim^{\circ}_{\tau''} c$,

$$\bot_{V^{\mathfrak{e}}_{\tau''\to\tau'}}(v) \lesssim_{\tau'} t'[\mathbf{s}/\mathbf{x}, c/x, t[\mathbf{s}/\mathbf{x}]/y].$$

Agreement of Eager Semantics

Inductive Step: $t = \mathbf{rec}y.(\lambda x.t')$ with $x : \tau''$, $t' : \tau'$

- $G_0 := \bot_{V^{\mathfrak{e}}_{\tau'' \to \tau'}}$ and $G_{n+1} := \lambda v.[\![t']\!]^{\mathfrak{e}}(\rho[\vec{\mathbf{v}}/\vec{\mathbf{x}}, v/x, G_n/y])$
- By induction: for all n, $G_n \lesssim^{\circ}_{\tau'' \to \tau'} (\lambda x.t')[\mathbf{s}/\mathbf{x}, t[\mathbf{s}/\mathbf{x}]/y]$
- Inductive Step: suppose $G_n \lesssim^{\circ}_{\tau'' \to \tau'} (\lambda x.t')[\mathbf{s}/\mathbf{x}, t[\mathbf{s}/\mathbf{x}]/y]$.
- Then $\lfloor G_n \rfloor \lesssim_{\tau'' \to \tau'} t[\mathbf{s}/\mathbf{x}]$.
- Consider any $v \lesssim^{\circ}_{\tau''} c$ so that $\lfloor v \rfloor \lesssim_{\tau''} c$.
- Note that $G_{n+1}(v) = [\![t']\!]^{\mathfrak{e}}(\rho[\vec{\mathbf{v}}/\vec{\mathbf{x}}, v/x, G_n/y])$.
- By the main induction hypothesis,

$$G_{n+1}(v) \lesssim_{\tau'} t'[\mathbf{s}/\mathbf{x}, c/x, t[\mathbf{s}/\mathbf{x}]/y].$$

- $G_{n+1} \lesssim^{\circ}_{\tau'' \to \tau'} (\lambda x.t')[\mathbf{s}/\mathbf{x}, t[\mathbf{s}/\mathbf{x}]/y]$.

Agreement of Eager Semantics

Corollary

- t : **int**: a closed term

Then $t \rightarrow^{\mathfrak{e}} n$ iff $[\![t]\!]^{\mathfrak{e}}(\rho) = \lfloor n \rfloor$.

Lazy Denotational Semantics

Textbook, Chapter 11.7

Lazy Denotational Semantics

Values

- τ: a type

The discrete cpo $V^{\mathfrak{l}}_{\tau}$ of values associated with type τ is recursively defined as follows:

- $V^{\mathfrak{l}}_{\textbf{int}} := \mathbb{Z}$;
- $V^{\mathfrak{l}}_{\tau_1 * \tau_2} := (V^{\mathfrak{l}}_{\tau_1})_{\perp} \times (V^{\mathfrak{l}}_{\tau_2})_{\perp}$;
- $V^{\mathfrak{l}}_{\tau_1 \to \tau_2} := [(V^{\mathfrak{l}}_{\tau_1})_{\perp} \to (V^{\mathfrak{l}}_{\tau_2})_{\perp}]$

Question

Why do we have extra $\perp$'s?

Lazy Denotational Semantics

Environments

- **Var**: the set of variables

An *environment* ρ is a function

$$\rho : \mathbf{Var} \to \bigcup\{(V^{\mathfrak{l}}_{\tau})_{\perp} \mid \tau \text{ a type}\}$$

such that

$$\forall x \in \mathbf{Var}.(x : \tau \Rightarrow \rho(x) \in (V^{\mathfrak{l}}_{\tau})_{\perp}) \ .$$

We denote by $\mathbf{Env}^{\mathfrak{l}}$ the set of environments under lazy semantics.

Lazy Denotational Semantics

Intuition

- t: a typable term with type τ
- $[\![t]\!]^{\mathfrak{l}} : \mathbf{Env}^{\mathfrak{l}} \to (V_\tau^{\mathfrak{l}})_\perp$: the denotational semantics of t

Lazy Denotational Semantics

Inductive Definition

- $\llbracket x \rrbracket^{\mathfrak{l}} := \lambda\rho.\rho(x);$
- $\llbracket n \rrbracket^{\mathfrak{l}} := \lambda\rho.\lfloor n \rfloor;$

Lazy Denotational Semantics

Inductive Definition

- $[\![t_1 \ \mathbf{op} \ t_2]\!]^{\mathfrak{l}} := \lambda\rho.([\![t_1]\!]^{\mathfrak{l}}(\rho) \ \mathbf{op}_{\perp} \ [\![t_2]\!]^{\mathfrak{l}}(\rho))$;
- $$[\![\mathbf{if} \ t_0 \ \mathbf{then} \ t_1 \ \mathbf{else} \ t_2]\!]^{\mathfrak{l}} := \lambda\rho.\mathit{cond}([\![t_0]\!]^{\mathfrak{l}}(\rho), [\![t_1]\!]^{\mathfrak{l}}(\rho), [\![t_2]\!]^{\mathfrak{l}}(\rho));$$

Lazy Denotational Semantics

Inductive Definition

- $[\![(t_1, t_2)]\!]^{\mathfrak{l}} := \lambda\rho.\lfloor([\![t_1]\!]^{\mathfrak{l}}(\rho), [\![t_2]\!]^{\mathfrak{l}}(\rho))\rfloor;$
- $[\![\mathbf{fst}(t)]\!]^{\mathfrak{l}} := \lambda\rho.\mathit{let}\ v \Leftarrow [\![t]\!]^{\mathfrak{l}}(\rho).\pi_1(v)$
- $[\![\mathbf{snd}(t)]\!]^{\mathfrak{l}} := \lambda\rho.\mathit{let}\ v \Leftarrow [\![t]\!]^{\mathfrak{l}}(\rho).\pi_2(v)$

Lazy Denotational Semantics

Inductive Definition

- $[\![\lambda x.t]\!]^{\mathfrak{l}} := \lambda \rho.\lfloor \lambda v \in (V^{\mathfrak{l}}_{\tau_1})_{\perp}.[\![t]\!]^{\mathfrak{l}}(\rho[v/x])\rfloor$ for $\lambda x.t : \tau_1 \to \tau_2$
- $[\![(t_1\ t_2)]\!]^{\mathfrak{l}} := \lambda \rho.let\ F \Leftarrow [\![t_1]\!]^{\mathfrak{l}}(\rho).F([\![t_2]\!]^{\mathfrak{l}}(\rho))$

Lazy Denotational Semantics

Inductive Definition

- $[\![\textbf{let } x \Leftarrow t_1 \textbf{ in } t_2]\!]^{\mathfrak{l}} := \lambda\rho.[\![t_2]\!]^{\mathfrak{l}}\left(\rho\left[[\![t_1]\!]^{\mathfrak{l}}(\rho)/x\right]\right)$
- $[\![\textbf{rec} y.t]\!]^{\mathfrak{l}} := \lambda\rho.(\mu F.[\![t]\!]^{\mathfrak{l}}(\rho[F/y]))$

Agreement of Lazy Semantics

Operational Convergence

- t: a typable closed term

We say that t is *operationally convergent*, denoted by $t{\downarrow}^{\mathfrak{l}}$, if it holds that $\exists c.t \rightarrow^{\mathfrak{l}} c$.

Agreement of Lazy Semantics

Denotational Convergence

- t: a typable closed term with type τ

We say that t is *denotationally convergent*, denoted by $t\Downarrow^{\mathfrak{l}}$, if it holds that $\exists v \in V_{\tau}^{\mathfrak{l}}.[\![t]\!]^{\mathfrak{l}}(\rho) = \lfloor v \rfloor$.

Agreement of Lazy Semantics

The Theorem

- t: a closed typable term
- c: a canonical term

Then we have:

- $t \rightarrow^{\mathfrak{l}} c$ implies $[\![t]\!]^{\mathfrak{l}}(\rho) = [\![c]\!]^{\mathfrak{l}}(\rho)$;
- $t{\downarrow}^{\mathfrak{l}}$ iff $t{\Downarrow}^{\mathfrak{l}}$.

Agreement of Lazy Semantics

A Corollary

- t: a closed typable term with type **int**

Then we have that $t \rightarrow^{l} n$ iff $[\![t]\!]^{l}(\rho) = \lfloor n \rfloor$.

Summary

- eager denotational semantics
- lazy denotational semantics
- agreement of the semantics

Special Thanks

Many thanks to Prof. Hongfei Fu for providing the source file of the presentation slides.

The original version can be downloaded here.