Axiomatic Semantics

- Stansifer Ch 2.4, Ch. 9
- Winskel Ch.6
- Slonneger and Kurtz Ch. 11

Outline

- Introduction
  - What are axiomatic semantics?
  - First-order logic & assertions about states
- Results (triples)
  - Proof system for deriving valid results
- Long examples: Division & Fibonnaci
- Total correctness
- Summary

Operational vs. Axiomatic

- Operational semantics
  - Explicitly describes the effects of program constructs on program state
  - Shows not only **what** the program does, but also **how** it does it
  - Essentially describes an interpreter
- Axiomatic semantics
  - Describes properties of program state, using first-order logic
  - Concerned with constructing proofs for such properties

Axiomatic Semantics

- Concerned w/ properties of program state
  - Properties are described (specified) through first-order logic
- Axiomatic semantics is a set of rules for constructing proofs of such properties
  - Should be able to prove all true statements about the program, and not be able to prove any false statements

State

- State: a function **σ** from variables to values
- E.g., program with 3 variables x, y, z

  $\sigma(x) = 9$

  $\sigma(y) = 5$

  $\sigma(z) = 2$
- For simplicity, we will only consider integer variables
  - σ: Variables $\rightarrow \{0,-1,+1,-2,2,\ldots\}$

Sets of States

- Need to talk about sets of states
- E.g., "$x=1, y=2, z=1$ <u>or</u> $x=1, y=2, z=2$ <u>or</u> $x=1, y=2, z=3$"
- We use assertions in first-order logic

  $$x=1 \wedge y=2 \wedge 1 \leq z \leq 3$$
- An assertion **p** represents the set of states that satisfy the assertion
  - We will write **{p}** to denote this set of states

Use of First-Order Logic

- Variables from the program
  - In the program they are part of the syntax, here they are part of the assertion
    - programming language vs. meta-language of assertions
- Extra "helper" variables
- The usual suspects from first-order logic
  $= \quad < \quad \wedge \quad \vee \quad \neg \quad \exists \quad \forall$ true false
- Operations from the programming language: e.g. +, -, …

First-Order Logic

- Terms
  - If **x** is a variable, **x** is a term
  - If **n** is an integer constant, **n** is a term
  - If $\mathbf{t_1}$ and $\mathbf{t_2}$ are terms, so are $\mathbf{t_1+t_2}$, $\mathbf{t_1-t_2}$,…
- Formulas
  - **true** and **false**
  - $\mathbf{t_1<t_2}$ and $\mathbf{t_1=t_2}$ for terms $\mathbf{t_1}$ and $\mathbf{t_2}$
  - $\mathbf{f_1 \wedge f_2}$, $\mathbf{f_1 \vee f_2}$, $\neg \mathbf{f_1}$ for formulas $\mathbf{f_1}$,$\mathbf{f_2}$
  - $\exists \mathbf{x.f}$ and $\forall \mathbf{x.f}$ for a formula **f**

When Does a State Satisfy an Assertion?

- Value of a term in some state σ
  - σ(x) for variable x, n for constant n, the usual arithmetic for terms t_1+t_2, t_1-t_2,…
- σ satisfies the assertion $\mathbf{t_1=t_2}$ if and only if $\mathbf{t_1}$ and $\mathbf{t_2}$ have the same value in σ
  - Similarly for assertion $\mathbf{t_1<t_2}$
- σ satisfies $\mathbf{f_1 \wedge f_2}$ if and only if it satisfies $\mathbf{f_1}$ and $\mathbf{f_2}$
  - Similarly for $f_1 \vee f_2$ and $\neg f_1$

When Does a State Satisfy an Assertion?

- σ satisfies $\forall x.f$ if and only if for every integer n, σ satisfies f[n/x]
  - Which states satisfy $\forall x.(x+y=y+x)$?
    - Which ones satisfy f[5/x] (i.e., 5+y=y+5)?
- σ satisfies $\exists x.f$ if and only if for some integer n, σ satisfies f[n/x]
  - Which states satisfy $\exists i.k=i*j$?

When Does a State Satisfy an Assertion?

- { p } denotes the set of states that satisfy assertion p
- $\{p \vee q\} \leftrightarrow \{p\} \cup \{q\}$; $\{p \wedge q\} \leftrightarrow \{p\} \cap \{q\}$
- $\{\neg p\} \leftrightarrow U - \{p\}$ (U is the universal set)
- Suppose that $p \Rightarrow q$ is true w.r.t. standard mathematics; then $\{p\} \subseteq \{q\}$
  - $x=2 \wedge y=3 \Rightarrow x=2$, so $\{ x=2 \wedge y=3 \} \subseteq \{ x=2 \}$

Examples of Assertions

- Three program variables: x, y, z
- $\{ x = 1 \wedge 1 \leq y \leq 5 \wedge 1 \leq z \leq 10 \}$
- $\{ x = 1 \wedge y = 2 \}$
- $\{ x = 1 \wedge 1 \leq y \leq 5 \}$
- $\{ x = y + z \}$
- $\{ x = x \}$
- { true }
- $\{ x \neq x \}$
- { false }

Examples of Assertions

- Three program variables: x, y, z
- $\{ x = 1 \wedge 1 \leq y \leq 5 \wedge 1 \leq z \leq 10 \}$: set of size 50
- $\{ x = 1 \wedge y = 2 \}$: infinite set
- $\{ x = 1 \wedge 1 \leq y \leq 5 \}$: infinite set
- $\{ x = y + z \}$: all states s.t. $\sigma(x) = \sigma(y) + \sigma(z)$
- $\{ x = x \}$
- $\{ \text{true} \}$
- $\{ x \neq x \}$
- $\{ \text{false} \}$

Examples of Assertions

- Three program variables: x, y, z
- $\{ x = 1 \wedge 1 \leq y \leq 5 \wedge 1 \leq z \leq 10 \}$: set of size 50
- $\{ x = 1 \wedge y = 2 \}$: infinite set
- $\{ x = 1 \wedge 1 \leq y \leq 5 \}$: infinite set
- $\{ x = y + z \}$: all states s.t. $\sigma(x) = \sigma(y) + \sigma(z)$
- $\{ x = x \}$: the set of all states
- $\{ \text{true} \}$: the set of all states
- $\{ x \neq x \}$: the empty set
- $\{ \text{false} \}$: the empty set

Simplified Programming Language

- IMP: simple imperative language
- From the code generation example with attribute grammars
  - With I/O added
- Only integer variables
- No procedures or functions
- No explicit variable declarations

Simple Imperative Language (IMP)

$\langle c \rangle_1$ ::= **skip** | **id** := $\langle ae \rangle$ | $\langle c \rangle_2$; $\langle c \rangle_3$
| **if** $\langle be \rangle$ **then** $\langle c \rangle_2$ **else** $\langle c \rangle_3$
| **while** $\langle be \rangle$ **do** $\langle c \rangle_2$

$\langle ae \rangle_1$ ::= **id** | **int** | $\langle ae \rangle_2$ + $\langle ae \rangle_3$
| $\langle ae \rangle_2$ - $\langle ae \rangle_3$ | $\langle ae \rangle_2$ * $\langle ae \rangle_3$

$\langle be \rangle_1$ ::= **true** | **false**
| $\langle ae \rangle_1$ = $\langle ae \rangle_2$ | $\langle ae \rangle_1$ < $\langle ae \rangle_2$
| $\neg$ $\langle be \rangle_2$ | $\langle be \rangle_2$ $\wedge$ $\langle be \rangle_3$
| $\langle be \rangle_2$ $\vee$ $\langle be \rangle_3$

Outline

- Introduction
  - What are axiomatic semantics?
  - First-order logic & assertions about states
- Results (triples)
  - Proof system for deriving valid results
- Long examples: Division & Fibonnaci
- Total correctness
- Summary

Hoare Triples

- By C. A. R. Hoare (Tony Hoare)
- **{p} S {q}**
  - S is a piece of code (program fragment)
  - p and q are assertions
  - p: pre-condition, q: post-condition
- If we start executing S from any state σ that satisfies p, and if S terminates, then the resulting state σ' satisfies q
- Will refer to the triples as **results**
  - Think "results of proofs"

Intuition

- In {p} S {q}, the relationship between p and q captures the essence of the semantics of S
- Abstract description of constraints that any implementation of the language must satisfy
  - Says nothing about how these relationships will be achieved (in contrast to operational semantics)

Valid Results

- A result {p} S {q} is valid if and only if for every state σ
  - if σ satisfies p (i.e., σ belongs to set {p})
  - and the execution of S starting in σ terminates in state σ'
  - then σ' satisfies q (i.e., σ' belongs to set {q})
- Is {false} S {q} valid?

Examples

- { x=1 } skip { x=1 }
- { x=1 ∧ y=1 } skip { x=1 }
- { x=1 } skip { x=1 ∧ y=1 }
- { x=1 } skip { x=1 ∨ y=1 }
- { x=1 ∨ y=1 } skip { x=1 }
- { x=1 } skip { true }
- { x=1 } skip { false }
- { false } skip { x=1 }

Examples

- { x=1 } skip { x=1 } Valid
- { x=1 ∧ y=1 } skip { x=1 } Valid
- { x=1 } skip { x=1 ∧ y=1 } Invalid
- { x=1 } skip { x=1 ∨ y=1 } Valid
- { x=1 ∨ y=1 } skip { x=1 } Invalid
- { x=1 } skip { true } Valid
- { x=1 } skip { false } Invalid
- { false } skip { x=1 } Valid

More Examples

- { x=1 ∧ y=2 } x := x+1 { x=2 ∧ y=2 }
- { x=1 ∧ y=2 } x := x+1 { x ≥ 2 }
- { x=1 ∧ y=2 } x := x+1 { x=y }

- { x=0 } while x<10 do x:=x+1 { x=10 }
- { x<0 } while x<10 do x:=x+1 { x=10 }
- { x≥0 } while x<10 do x:=x+1 { x=10 }
- { x≥0 } while x<10 do x:=x+1 { x≥10 }

More Examples

- { x=1 ∧ y=2 } x := x+1 { x=2 ∧ y=2 } Valid
- { x=1 ∧ y=2 } x := x+1 { x ≥ 2 } Valid
- { x=1 ∧ y=2 } x := x+1 { x=y } Valid

- { x=0 } while x<10 do x:=x+1 { x=10 } Valid
- { x<0 } while x<10 do x:=x+1 { x=10 } Valid
- { x≥0 } while x<10 do x:=x+1 { x=10 } Invalid
- { x≥0 } while x<10 do x:=x+1 { x≥10 } Valid

Termination

- A result says: … if S terminates …
- What if S does not terminate?
  - We are **only** concerned with initial states for which S terminates
- **{ x=3 } while x≠10 do x:=x+1 { x=10 }**
- **{ x≥0 } while x≠10 do x:=x+1 { x=10 }**
- **{ true } while x≠10 do x:=x+1 { x=10 }**
- All of these results are valid

Observations

- What exactly does "valid result" mean?
- We had an operational model of how the code would operate, and we "executed" the code in our heads using this model
  - The result is valid w.r.t. the model
  - The operational model can be formalized
  - In our discussion: an implied "obvious" model
- Goal from now on: derive valid results without using operational reasoning
  - Purely formally, using a proof system

Terminology

- Assertion: may be satisfied or not satisfied by a particular state
- Result: may be valid or invalid in a particular operational model
- Result: may be derivable or not derivable in a given proof system
- Some meaningless statements (don't use!)
  - "{p} S {q} is true", "{p} S {q} is valid for *some* states", "assertion p is not valid"

Soundness and Completeness

- Properties of a proof system (axiomatic semantics) A
  - w.r.t. an operational model M
- Soundness (consistency): every result we can prove (derive) in A is valid in M
- Completeness: every result that is valid in M can be derived (proven) in A

Proofs

- Proof = set of applications of instances of inference rules
  - Starting from one or more axioms
- Conclusions are subsequently used as premises
- The conclusion of the last production is proved (derived) by the proof
  - If a proof exists, the result is provable (derivable)

Proof System for IMP

- Goal: define a proof system for IMP
  - i.e., an axiomatic semantics
- Skip axiom: p is an arbitrary assertion

  { p } skip { p }
- Examples

{ x=1 } skip { x=1 } Provable

{ x=1 } skip { x=1 ∧ y=2 } Not provable

{ x=1 ∧ y=2 } skip { x=1 } Not provable

Inference Rule of Consequence

$$\frac{p' \Rightarrow p \qquad \{ p \}\ S\ \{ q \} \qquad q \Rightarrow q'}{\{ p' \}\ S\ \{ q' \}}$$

- Recall that $x \Rightarrow y$ means $\{x\} \subseteq \{y\}$

$$\frac{x{=}1 \wedge y{=}2 \ \Rightarrow\ x{=}1 \qquad \overline{\{ x{=}1 \}\ \text{skip}\ \{ x{=}1 \}}}{\{ x{=}1 \wedge y{=}2 \}\ \text{skip}\ \{ x{=}1 \}}$$

Simplified Versions

$$\frac{\{ p \}\ S\ \{ q \} \qquad q \Rightarrow q'}{\{ p \}\ S\ \{ q' \}}$$

$$\frac{p' \Rightarrow p \qquad \{ p \}\ S\ \{ q \}}{\{ p' \}\ S\ \{ q \}}$$

Exercise

- Show that the following rule will make the proof system inconsistent (unsound)
  - i.e. it will be possible to prove something that is not operationally valid

$$\frac{\{ p \}\ S\ \{ q \} \qquad q' \Rightarrow q}{\{ p \}\ S\ \{ q' \}}$$

Substitution

- Notation: **p[e/x]**
  - Other notations: p^x_e , **p[x:=e]**
- **p[e/x]** is the assertion **p** with all free occurrences of **x** replaced by **e**
  - To avoid conflicts, may have to rename some quantified variables
- Examples
  - $(x{=}y)[5/x] \Rightarrow 5{=}y$, $(x{=}y \wedge x{=}2)[5/x] \Rightarrow 5{=}y \wedge 5{=}2$
  - $(x{=}k \wedge \exists k.a_k{>}x)[y/k] \Rightarrow (x{=}y \wedge \exists k.a_k{>}x)$
  - $(x{=}k \wedge \exists k.a_k{>}x)[k/x] \Rightarrow (k{=}k \wedge \exists j.a_j{>}k)$

Free vs. Bound Variable Occurrences

- An occurrence of a variable x is bound if it is in the scope of $\exists$**x** or $\forall$**x**
  - An occurrence is free if it is not bound
- $\exists$**i.k=i*j**: k and j are free, i is bound
- $(x{+}1 < y{+}2) \wedge (\exists x.\ x{+}3{=}y{+}4)$
- Substitution: **f[e/x]** is the formula f with all free occurrences of **x** replaced by **e**
  - May have to rename variables (more later)

Assignment Axiom

$\{ p[e/x] \}\ x := e\ \{ p \}$ p is any assertion

- $\{ x{+}1 = y{+}z \}\ x := x{+}1\ \{ x = y{+}z \}$
- $\{ y{+}z > 0 \}\ x := y{+}z\ \{ x > 0 \}$
- $\{ y{+}z = y{+}z \}\ x := y{+}z\ \{ x = y{+}z \}$
- due to **true⇒y+z=y+z** and the consequence rule: $\{true\}\ x{:=}y{+}z\ \{ x = y{+}z \}$

Intuition

- The initial state must satisfy the same assertion except for **e** playing the role of **x**
- Operational intuition: you cannot use it in an axiomatic derivation
  - Only allowed to use the axioms and rules
- E.g. **{ x > 0 } x := 1 { x = 1 }**
  - Not: "After assigning 1 to x, we end up in a state in which x=1"
  - But: "This can be proved using the assignment axiom and the rule of consequence"

Inference Rule of Composition

$$\frac{\{ p \} \; S1 \; \{ q \} \qquad \{ q \} \; S2 \; \{ r \}}{\{ p \} \; S1;S2 \; \{ r \}}$$

- Example

$$\frac{\{x+1=y+z\} \text{ skip } \{x+1=y+z\} \qquad \{x+1=y+z\} \; x:=x+1 \; \{x=y+z\}}{\{x+1=y+z\} \text{ skip; } x:=x+1 \; \{x=y+z\}}$$

Input/Output

- Idea: treat input and output streams as variables
- Use the assignment axiom
- **write** modifies the output stream
  - **"write e"** is **OUT := OUT ^ e**
- **read** modifies the input variable and the input stream
  - **"read x"** is **x := head(IN); IN := tail(IN)**

Write Axiom

{ p[OUT^e / OUT] } write e { p }

- Example

$$\frac{OUT=\langle\rangle \Rightarrow OUT\hat{}4=\langle 4\rangle \qquad \{OUT\hat{}4=\langle 4\rangle\} \text{ write } 4 \; \{OUT=\langle 4\rangle\}}{\{ OUT=\langle\rangle\} \text{ write } 4 \; \{OUT=\langle 4\rangle\}}$$

Read Axiom

{ (p[tail(IN)/IN]) [head(IN)/x] } read x {p}

$$\frac{\{tail(IN)=\langle 4\rangle \wedge head(IN)=3\} \text{ read } x \; \{IN=\langle 4\rangle \wedge x=3\} \qquad IN=\langle 3,4\rangle \Rightarrow tail(IN)=\langle 4\rangle \wedge head(IN)=3}{\{ IN=\langle 3,4\rangle \} \text{ read } x \; \{ IN=\langle 4\rangle \wedge x=3 \}}$$

Alternative Notation

- write axiom

$$\{ p^{OUT}_{OUT\hat{}e} \} \text{ write } e \; \{ p \}$$

- read axiom

$$\{ (p^{IN}_{tail(IN)})^{x}_{head(IN)} \} \text{ read } x \; \{ p \}$$

Example

- Prove

{ IN = <3,4> ∧ OUT = <> }
read x;
read y;
write x+y;
{ OUT = <7> }

Example

1. Using the write axiom and the postcondition:
   { OUT^(x+y) = <7> } write x+y { OUT=<7> }
2. Using (1) and the rule of consequence:
   { x+y=7 ∧ OUT=<> } write x+y { OUT=<7> }
3. Using read axiom:
   {x+head(IN)=7∧OUT=<>} read y {x+y=7∧OUT=<>}
4. Using (2), (3), and sequential composition:
   {x+head(IN)=7∧OUT=<>}
   read y; write x+y { OUT=<7> }

Example

5. Using the read axiom:
   {head(IN) + head(tail(IN)) = 7 ∧ OUT = <>}
   read x
   { x + head(IN) = 7 ∧ OUT = <> }
6. Using (5) and the rule of consequence
   { IN = <3,4> ∧ OUT = <> }
   read x
   { x + head(IN) = 7 ∧ OUT = <> }
7. Using (4), (6), and sequential composition
   { IN = <3,4> ∧ OUT = <> }
   read x; read y; write x+y;
   { OUT = <7> }

Proof Strategy

- For any sequence of assignments and input/output operations:
  - Start with the last statement
  - Apply the assignment/read/write axioms working backwards
  - Apply the rule of consequence to make the preconditions "nicer"

If-Then-Else Rule

$$\frac{\{ p \wedge b \}\ S1\ \{ q \} \qquad \{ p \wedge \neg b \}\ S2\ \{ q \}}{\{ p \}\ \text{if } b \text{ then } S1 \text{ else } S2\ \{ q \}}$$

Example:

{ y = 1 }
if y = 1 then x := 1 else x := 2
{ x = 1 }

If-Then-Else Example

$$\frac{y=1 \wedge y=1 \Rightarrow 1=1 \qquad \{1=1\}\ x:=1\ \{x=1\}}{\{ y=1 \wedge y=1 \}\ x:=1\ \{ x=1 \}}$$

$$\frac{y=1 \wedge \neg(y=1) \Rightarrow 2=1 \qquad \{2=1\}\ x:=2\ \{x=1\}}{\{ y=1 \wedge \neg(y=1) \}\ x:=2\ \{ x=1 \}}$$

{y=1} if y=1 then x:=1 else x:=2 {x=1}

Simplified If-Then-Else Rule

- Why not simply

$$\frac{\{ p \}\ S1\ \{ q \} \qquad \{ p \}\ S2\ \{ q \}}{\{ p \}\ \text{if } b \text{ then } S1 \text{ else } S2\ \{ q \}}$$

- Works for

  {true} if y=1 then x:=1 else x:=2 {x=1∨x=2}

- Easy to prove that
  - { true } x:=1 { x = 1 ∨ x = 2 }
  - { true } x:=2 { x = 1 ∨ x = 2 }
  - with assignment axiom and consequence

Simplified If-Then-Else Rule

- Does not work for

  { y=1 } if y=1 then x:=1 else x:=2 { x=1 }

- Attempt for a proof: we need
  - { y=1 } x:=1 { x=1 }, { y=1 } x:=2 { x=1 }
  - The second result cannot be proven using axioms and rules
- With the simplified rule, the proof system becomes incomplete
  - i.e. it becomes impossible to prove something that is, in fact, operationally valid

While Loop Rule

- Problem: proving

  { P } while B do S end { Q }

  for arbitrary **P** and **Q** is undecidable
  - Need to encode the knowledge that went into constructing the loop
- For each loop, we need an invariant **I** – an assertion that must be satisfied by
  - the state at beginning of the loop
  - the state at the end of each iteration
  - the state immediately after the loop exits
- Finding a loop invariant is the hard part

While Loop Rule

$$\frac{\{ I \wedge b \}\ S\ \{ I \}}{\{ I \}\ \text{while } b \text{ do } S \text{ end}\ \{ I \wedge \neg b \}}$$

- In practice often combined with the rule of consequence

$$\frac{p \Rightarrow I \quad \{ I \wedge b \}\ S\ \{ I \} \quad (I \wedge \neg b) \Rightarrow q}{\{ p \}\ \text{while } b \text{ do } S \text{ end}\ \{ q \}}$$

Outline

- Introduction
  - What are axiomatic semantics?
  - First-order logic & assertions about states
- Results (triples)
  - Proof system for deriving valid results
- Long examples: Division & Fibonnaci
- Total correctness
- Summary

Example: Division

- Prove

```
{ (x≥0) ∧ (y>0) }
q := 0;
r := x;
while (r - y) ≥ 0 do
    q := q + 1;
    r := r - y
end
{ (x=q*y+r) ∧ (0≤r<y) }
```

q: quotient
r: remainder

Note: what if y>0 was not in the precondition? Is the result valid? Is it derivable?

Example: Division

- Loop invariant
  - Should state relationship between variables used in loop

    $(x=q*y+r)$

- Needs a boundary condition to make the proof work

  $(x=q*y+r) \wedge (0 \le r)$

Example: Division

$\{ (x \ge 0) \wedge (y>0) \}$
q := 0;
r := x;
$\{ (x=q*y+r) \wedge (0 \le r) \}$
while (r - y) ≥ 0 do
    q := q + 1;
    r := r - y
end
$\{ (x=q*y+r) \wedge (0 \le r) \wedge (r-y<0) \}$
$\{ (x=q*y+r) \wedge (0 \le r < y) \}$

Example: Division

- Code before the loop

  $\{ (x \ge 0) \wedge (y>0) \}$
  q := 0;
  r := x;
  $\{ (x=q*y+r) \wedge (0 \le r) \}$ – the invariant

- Proof: assignment, composition, and consequence lead to

$(x \ge 0) \wedge (y>0) \Rightarrow (x=0*y+x) \wedge (0 \le x)$
obviously true

Example: Division

Need: $\{ I \wedge b \}\ S\ \{ I \}$

$\{ (x=q*y+r) \wedge (0 \le r) \wedge (r-y \ge 0) \}$
q := q + 1;
r := r - y
$\{ (x=q*y+r) \wedge (0 \le r) \}$

- Eventually we have the implication

$(x=q*y+r) \wedge (0 \le r) \wedge (r-y \ge 0) \Rightarrow$

$(x=(q+1)*y+r-y) \wedge (r-y \ge 0)$

Simple arithmetic proves this

Example: Division

- At exit: need the implication $(I \wedge \neg b) \Rightarrow q$

$(x=q*y+r) \wedge (0 \le r) \wedge (r-y<0) \Rightarrow$

$(x=q*y+r) \wedge (0 \le r < y)$

Trivially true

Example: Fibonacci Numbers

$\{ n > 0 \}$
i := n;
f := 1;
h := 1;
while i > 1 do
    h := h + f;
    f := h - f;
    i := i - 1
end
$\{ f = fib(n) \}$

Math definition:
fib(1) = 1
fib(2) = 1
…
fib(i+1) = fib(i) + fib(i-1)
…

Example: Fibonacci Numbers

- Invariant: {f=fib(n-i+1) ∧ h=fib(n-i+2) ∧ i>0}
- Steps

n>0 ⇒ 1=fib(n-n+1) ∧ 1=fib(n-n+2) ∧ n>0

i:=n; f:=1; h:=1

{ f=fib(n-i+1) ∧ h=fib(n-i+2) ∧ i>0 } [invariant]

start of loop

{ f=fib(n-i+1) ∧ h=fib(n-i+2) ∧ i>0 ∧ i>1 } ⇒

{ h=fib(n-i+2) ∧ h+f=fib(n-i+3) ∧ (i-1)>0 } ⇒

{ h+f-f=fib(n-(i-1)+1) ∧ h+f=fib(n-(i-1)+2) ∧ (i-1)>0 }

Example: Fibonacci Numbers

{ h+f-f=fib(n-(i-1)+1) ∧ h+f=fib(n-(i-1)+2) ∧ (i-1>0) }

h:=h+f;

{ h-f=fib(n-(i-1)+1) ∧ h=fib(n-(i-1)+2) ∧ (i-1>0) }

f:=h-f;

{ f=fib(n-(i-1)+1) ∧ h=fib(n-(i-1)+2) ∧(i-1)>0}

i:=i-1 – after this, we get the loop invariant

end of loop: { f=fib(n-i+1) ∧ h=fib(n-i+2) ∧ i>0 ∧ i≤1 } ⇒ f=fib(n)

Example: I/O

```
{ IN=<1,2,..,100> ∧ OUT=<> }
read x;
while x≠100 do
  write x;
  read x;
end
{ OUT = <1,2,…,99> }
```


Proof

Loop invariant: OUT^x^IN = <1,2,…,100>

{ IN=<1,2,..,100> ∧ OUT=<> } read x;

{ x=1 ∧ IN=<2,..,100> ∧ OUT=<> }

{ I ∧ x≠100 } write x; read x; { I }

I∧x≠100⇒OUT^x^head(IN)^tail(IN)=<1,…,100>

$\{ p^{OUT}_{OUT\hat{}x} \}$ write x { p }:

{ OUT^head(IN)^tail(N) = <1,2,…,100>}

$\{ (p^{IN}_{tail(IN)})^{x}_{head(IN)} \}$ read x { p }:

{ OUT^x^IN = <1,2,…,100>}

Completeness and Consistency

- This set of rules is complete for IMP
  - Anything that is operationally valid can be proven
- Proving consistency/completeness is hard
- One approach: start with a known system A and make changes to obtain system A'
  - If A is complete and all results derivable in A are also derivable in A': A' is complete
  - If A is consistent and all results derivable in A' are also derivable in A: A' is consistent

Outline

- Introduction
  - What are axiomatic semantics?
  - First-order logic & assertions about states
- Results (triples)
  - Proof system for deriving valid results
- Long examples: Division & Fibonnaci
- Total correctness
- Summary

Total Correctness

- So far we only had partial correctness
- Want to handle
  - Reading from empty input
  - Division by zero and other run-time errors
  - Idea: add sanity check to precondition
- Also, want to handle non-termination
  - Do this through a termination function

Hoare Triples – Total Correctness

- $\langle p \mid S \mid q \rangle$
  - S is a piece of code (program fragment)
  - p: pre-condition, q: post-condition
- If we start executing S from any state σ that satisfies p, then S terminates and the resulting state σ' satisfies q
- Alternative notation: [p] S [q]

Total Correctness Rule

- New assignment axiom

$$\frac{p \Rightarrow (D(e) \wedge q[e/x])}{\langle p \mid x := e \mid q \rangle}$$

where **D(e)** means "**e** is well-defined"

- New read axiom

$$\frac{p \Rightarrow (IN \neq \langle\rangle \wedge (q[tail(IN)/IN])[head(IN)/x]}{\langle p \mid read\ x \mid q \rangle}$$

Total Correctness Rule for While

- Idea: find termination function **f** (some expression based on program variables)
  - Decreases with every iteration
  - Always positive at start of loop body
  - Also called "progress function"

$$\frac{(I \wedge b) \Rightarrow f>0 \quad \langle I \wedge b \wedge f=k \mid S \mid I \wedge f<k \rangle}{\langle I \mid while\ b\ do\ S\ end \mid I \wedge \neg b \rangle}$$

Examples of Termination Functions

- Division example
  - Remainder **r** decreases in every step and does not get negative
- Fibonacci numbers
  - There already is an explicit counter **i**

Another Progress Function

$\langle s = 0 \wedge x = 0 \mid$

while $x \neq 10$ do $x:=x+1$; $s:=s+x$ end

$\mid s = \Sigma_{k=0}^{10} k \rangle$

Invariant: $0 \leq x \leq 10 \wedge s = \Sigma_{k=0}^{x} k$

Progress function: 10-x

Other Total Correctness Rules

- Essentially identical: e.g.

$$\frac{\langle p \mid S1 \mid q \rangle \quad \langle q \mid S2 \mid r \rangle}{\langle p \mid S1;S2 \mid r \rangle}$$

Outline

- Introduction
  - What are axiomatic semantics?
  - First-order logic & assertions about states
- Results (triples)
  - Proof system for deriving valid results
- Long examples: Division & Fibonnaci
- Total correctness
- Summary

Summary: Axiomatic Semantics

- First-order logic formulas express set of possible states
- Hoare triples express partial (total) correctness conditions
- Proof rules used to define axiomatic semantics
- Must be sound (consistent) and complete relative to the operational model

Program Verification

- Given an already defined axiomatic semantics, we can try to prove partial or total correctness
  - S is a program fragment
  - p is something we can guarantee
  - q is something we want S to achieve
  - Try to prove {p} S {q} and/or $\langle p \mid S \mid q \rangle$
- If we find a proof, S is correct
- A counter-example uncovers a bug

Program Verification

- Specification using pre/post-conditions
- Need to find loop invariants
  - Express behavior of loop
- Backward substitution across multiple assignments
- Need to find termination function for proving total correctness