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 & Fibonacci
- 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 \( \sigma \) 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
  - \( \sigma: \text{Variables} \rightarrow \{0, -1, +1, -2, 2, \ldots\} \)

Sets of States

- Need to talk about sets of states
  - E.g., "\( x=1, y=2, z=1 \text{ or } x=1, y=2, z=2 \text{ or } x=1, y=2, z=3 \)"
- We use assertions in first-order logic
  \[
  x=1 \land y=2 \land 1 \leq z \leq 3
  \]
- 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
  \( = \land \lor \neg \exists \forall \true \false \)
- 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 \( t_1 \) and \( t_2 \) are terms, so are \( t_1 + t_2 \), \( t_1 - t_2 \), ...
- Formulas
  - true and false
  - \( t_1 < t_2 \) and \( t_1 = t_2 \) for terms \( t_1 \) and \( t_2 \)
  - \( f_1 \land f_2 \), \( f_1 \lor f_2 \), \( \neg f_1 \) for formulas \( f_1 \), \( f_2 \)
  - \( \exists x. f \) and \( \forall x. f \) for a formula \( f \)

When Does a State Satisfy an Assertion?

- Value of a term in some state \( \sigma \)
  - \( \sigma(x) \) for variable \( x \), \( n \) for constant \( n \), the usual arithmetic for terms \( t_1 + t_2 \), \( t_1 - t_2 \), ...
- \( \sigma \) satisfies the assertion \( t_1 = t_2 \) if and only if \( t_1 \) and \( t_2 \) have the same value in \( \sigma \)
- Similarly for assertion \( t_1 < t_2 \)
- \( \sigma \) satisfies \( f_1 \land f_2 \) if and only if it satisfies \( f_1 \) and \( f_2 \)
  - Similarly for \( f_1 \lor f_2 \) and \( \neg f_1 \)

When Does a State Satisfy an Assertion?

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

When Does a State Satisfy an Assertion?

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

Examples of Assertions

- Three program variables: \( x, y, z \)
  - \( \{x = 1 \land 1 \leq y \leq 5 \land 1 \leq z \leq 10\} \)
  - \( \{x = 1 \land y = 2\} \)
  - \( \{x = 1 \land 1 \leq y \leq 5\} \)
  - \( \{x = y + z\} \)
  - \( \{x = x\} \)
  - \( \{true\} \)
  - \( \{false\} \)
Examples of Assertions

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

Examples of Assertions

- Three program variables: x, y, z
- \{ x = 1 \land 1 \leq y \leq 5 \land 1 \leq z \leq 10 \}: set of size 50
- \{ x = 1 \land y = 2 \}: infinite set
- \{ x = 1 \land 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
- \{ true \}: the set of all states
- \{ x \neq x \}: the empty set
- \{ 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 ::= skip \mid id ::= \langle a e \rangle \mid \langle c \rangle_2 : \langle c \rangle_3 \mid if \langle b e \rangle then \langle c \rangle_2 else \langle c \rangle_3 \mid while \langle b e \rangle do \langle c \rangle_2
- \langle a e \rangle ::= id \mid int \mid \langle a e \rangle_2 + \langle a e \rangle_3 \mid \langle a e \rangle_2 - \langle a e \rangle_3 \mid \langle a e \rangle_2 \ast \langle a e \rangle_3
- \langle b e \rangle ::= true \mid false \mid \langle a e \rangle = \langle a e \rangle_2 \mid \langle a e \rangle_1 < \langle a e \rangle_2 \mid \lnot \langle b e \rangle_2 \mid \langle b e \rangle_2 \land \langle b e \rangle_3 \mid \langle b e \rangle_2 \lor \langle b e \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 & Fibonacci
- 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 \sigma that satisfies p, and if S terminates, then the resulting state \sigma' satisfies q
- Will refer to the triples as results
  - Think "results of proofs"
**Intuition**

- In \( p \rightarrow S \rightarrow q \), 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 \( p \rightarrow S \rightarrow q \) is valid if and only if for every state \( \sigma \):
  - if \( \sigma \) satisfies \( p \) (i.e., \( \sigma \) belongs to set \( p \))
  - and the execution of \( S \) starting in \( \sigma \) terminates in state \( \sigma' \)
  - then \( \sigma' \) satisfies \( q \) (i.e., \( \sigma' \) belongs to set \( q \))
- Is \( \{ \text{false} \} \rightarrow S \rightarrow \{ \text{false} \} \) valid?

**Examples**

- \( \{ x=1 \} \) skip \( \{ x=1 \} \)
- \( \{ x=1 \land y=1 \} \) skip \( \{ x=1 \} \)
- \( \{ x=1 \} \) skip \( \{ x=1 \land y=1 \} \)
- \( \{ x=1 \} \) skip \( \{ x=1 \lor y=1 \} \)
- \( \{ x=1 \lor y=1 \} \) skip \( \{ x=1 \} \)
- \( \{ x=1 \} \) skip \( \{ \text{true} \} \)
- \( \{ x=1 \} \) skip \( \{ \text{false} \} \)
- \( \{ \text{false} \} \) skip \( \{ x=1 \} \)

**More Examples**

- \( \{ x=1 \land y=2 \} \) \( x := x+1 \) \( \{ x=2 \land y=2 \} \)
- \( \{ x=1 \land y=2 \} \) \( x := x+1 \) \( \{ x \geq 2 \} \)
- \( \{ x=1 \land y=2 \} \) \( x := x+1 \) \( \{ x=y \} \)
- \( \{ x=0 \} \) 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 \( x := x+1 \) \( \{ x \geq 10 \} \)

**Valid**

- \( \{ false \} \) skip \( \{ \text{false} \} \)
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 \neq 10 do x := x + 1 \{ x=10 \}
- \{ x \geq 0 \} while x \neq 10 do x := x + 1 \{ x=10 \}
- \{ true \} while x \neq 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
  \[
  \{ p \} \text{skip} \{ p \}
  \]
- Examples
  \[
  \{ x=1 \} \text{skip} \{ x=1 \} \quad \text{Provable}
  \{ x=1 \} \text{skip} \{ x=1 \land y=2 \} \quad \text{Not provable}
  \{ x=1 \land y=2 \} \text{skip} \{ x=1 \} \quad \text{Not provable}
  \]
Inference Rule of Consequence

\[
p' \Rightarrow p \quad \{p\} S \{q\} \quad q \Rightarrow q'
\]

- Recall that \(x \Rightarrow y\) means \(\{x\} \subseteq \{y\}\)

\[
x = 1 \land y = 2 \Rightarrow x = 1 \quad \{x = 1\} \text{skip} \{x = 1\}
\]

\[
\{x = 1 \land y = 2\} \text{skip} \{x = 1\}
\]

Simplified Versions

\[
\{p\} S \{q\} \quad q \Rightarrow q'
\]

\[
\{p'\} S \{q'\}
\]

- \(\{x\} \subseteq \{y\}\)

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

\[
\{p\} S \{q\} \quad q \Rightarrow q'
\]

Substitution

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

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
- \((x+1 \land y+2) \land (\exists x. x+3=y+4)\)
- 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 \(true \Rightarrow y+z = y+z\) 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

\[
\begin{align*}
\{ p \} S1 \{ q \} & \quad \{ q \} S2 \{ r \} \\
\{ p \} S1;S2 \{ r \}
\end{align*}
\]

- Example

\[
\begin{align*}
(x+1=y+z) \text{ skip } (x+1=y+z) & \quad (x+1=y+z) x:=x+1 (x=y+z) \\
(x+1=y+z) \text{ skip; } x:=x+1 (x=y+z) &
\end{align*}
\]

Input/Output

- Idea: treat input and output streams as variables
- Use the assignment axiom
- write modifies the output stream
  - “write e” is \( \text{OUT} := \text{OUT}^ \cdot e \)
- read modifies the input variable and the input stream
  - “read x” is \( x := \text{head(IN)}; \ \text{IN} := \text{tail(IN)} \)

Write Axiom

\[
\{ p[\text{OUT}^\cdot e / \text{OUT}] \} \text{ write } e \{ p \}
\]

- Example

\[
\begin{align*}
\text{OUT}=<\rangle \Rightarrow \text{OUT}^\cdot 4=<4> \{ \text{OUT}^\cdot 4=<4> \} \text{ write } 4 \{ \text{OUT}=<4> \}
\end{align*}
\]

Read Axiom

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

\[
\begin{align*}
\text{IN}=<3,4> \Rightarrow \text{tail(IN)}=<4> \land \text{head(IN)}=3 \\
\text{IN}=<3,4> \Rightarrow \text{read } x \{ \text{IN}=<4> \land x=3 \}
\end{align*}
\]

Alternative Notation

- write axiom
  \[
  \{ p^\cdot \text{OUT} \} \text{ write } e \{ p \}
  \]
- read axiom
  \[
  \{ (p^\cdot \text{IN})^x \} \text{ read } x \{ p \}
  \]
Example

1. Using the write axiom and the postcondition:
   \{ \text{OUT}^\text{w}(x+y) = \langle 7 \rangle \} \, \text{write } x+y \, \{ \text{OUT}=\langle 7 \rangle \}

2. Using (1) and the rule of consequence:
   \{ x+y=7 \land \text{OUT}=\langle \rangle \} \, \text{write } x+y \, \{ \text{OUT}=\langle 7 \rangle \}

3. Using read axiom:
   \{ x+\text{head(IN)}=7 \land \text{OUT}=\langle \rangle \} \, \text{read } y \, \{ x+y=7 \land \text{OUT}=\langle \rangle \}

4. Using (2), (3), and sequential composition:
   \{ x+\text{head(IN)}=7 \land \text{OUT}=\langle \rangle \} \, \text{read } y \, \text{write } x+y \, \{ \text{OUT}=\langle 7 \rangle \}

Example

5. Using the read axiom:
   \{ \text{head(IN)} + \text{head(tail(IN))} = 7 \land \text{OUT} = \langle \rangle \}
   \text{read } x
   \{ x + \text{head(IN)} = 7 \land \text{OUT} = \langle \rangle \}

6. Using (5) and the rule of consequence
   \{ \text{IN} = \langle 3,4 \rangle \land \text{OUT} = \langle \rangle \}
   \text{read } x
   \{ x + \text{head(IN)} = 7 \land \text{OUT} = \langle \rangle \}

7. Using (4), (6), and sequential composition
   \{ \text{IN} = \langle 3,4 \rangle \land \text{OUT} = \langle \rangle \}
   \text{read } x; \text{read } y; \text{write } x+y;
   \{ \text{OUT} = \langle 7 \rangle \}

Example

5. Using the read axiom:
   \{ \text{head(IN)} + \text{head(tail(IN))} = 7 \land \text{OUT} = \langle \rangle \}
   \text{read } x
   \{ x + \text{head(IN)} = 7 \land \text{OUT} = \langle \rangle \}

6. Using (5) and the rule of consequence
   \{ \text{IN} = \langle 3,4 \rangle \land \text{OUT} = \langle \rangle \}
   \text{read } x
   \{ x + \text{head(IN)} = 7 \land \text{OUT} = \langle \rangle \}

7. Using (4), (6), and sequential composition
   \{ \text{IN} = \langle 3,4 \rangle \land \text{OUT} = \langle \rangle \}
   \text{read } x; \text{read } y; \text{write } x+y;
   \{ \text{OUT} = \langle 7 \rangle \}

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

\[
\begin{array}{c}
\{ p \land b \} S_1 \{ q \} \\
\{ p \land \neg b \} S_2 \{ q \}
\end{array}
\]

\{ p \} \text{ if } b \text{ then } S_1 \text{ else } S_2 \{ q \}

Example:

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

If-Then-Else Example

\[
\begin{array}{c}
y=1 \land y=1 \Rightarrow 1=1 \\
\{1=1\} \, x:=1 \{ x=1 \}
\end{array}
\]

\[
\begin{array}{c}
y=1 \land \neg(y=1) \Rightarrow 2=1 \\
\{2=1\} \, x:=2 \{ x=1 \}
\end{array}
\]

\[
\begin{array}{c}
\{ y=1 \} \text{ if } y=1 \text{ then } x:=1 \text{ else } x:=2 \{ x=1 \}
\end{array}
\]
**Simplified If-Then-Else Rule**

- Why not simply

  \[
  \{ p \} S_1 \{ q \} \quad \{ p \} S_2 \{ q \}
  \]

  \[
  \{ p \} \text{ if } b \text{ then } S_1 \text{ else } S_2 \{ q \}
  \]

- Works for

  (true) if \( y = 1 \) then \( x := 1 \) else \( x := 2 \) \( \langle x = 1 \lor x = 2 \rangle \)

- Easy to prove that

  - \( \{ \text{true} \} x := 1 \ (x = 1 \lor x = 2) \)
  - \( \{ \text{true} \} x := 2 \ (x = 1 \lor x = 2) \)
  - with assignment axiom and consequence

**While Loop Rule**

- Problem: proving

  \[
  \{ P \} \text{ while } B \text{ do } S \text{ end } \{ Q \}
  \]

  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

**Example: Division**

- Prove

  \[
  \{(x \geq 0) \land (y > 0)\}
  \]

  \[
  q := 0; \quad q: \text{ quotient}
  \]

  \[
  r := x; \quad r: \text{ remainder}
  \]

  \[
  \text{while } (r - y) \geq 0 \text{ do}
  \]

  \[
  q := q + 1; \quad \text{ end}
  \]

  \[
  r := r - y
  \]

  \[
  \{(x = q \ast y + r) \land (0 \leq r < y)\}
  \]

**While Loop Rule**

- In practice often combined with the rule of consequence

  \[
  p \Rightarrow I \quad \{ I \land b \} S \{ I \} \quad (I \land \neg b) \Rightarrow q
  \]

  \[
  \{ p \} \text{ while } b \text{ do } S \text{ end } \{ q \}
  \]

**Outline**

- Introduction
  - What are axiomatic semantics?
  - First-order logic & assertions about states

- Results (triples)
  - Proof system for deriving valid results

- Long examples: Division & Fibonacci

- Total correctness

- Summary
Example: Division

- Loop invariant
  - Should state relationship between variables used in loop
    \( (x=q\cdot y+r) \)

- Needs a boundary condition to make the proof work
  \( (x=q\cdot y+r) \land (0 \leq r) \)

Example: Division

\( \{ (x>0) \land (y>0) \} \)

\( q := 0; \)
\( r := x; \)
\( \{ (x=q\cdot y+r) \land (0 \leq r) \} \)

while \( (r - y) \geq 0 \) do
  \( q := q + 1; \)
  \( r := r - y \)
end

\( \{ (x=q\cdot y+r) \land (0 \leq r) \land (r > 0) \} \)

\( \{ (x=q\cdot y+r) \land (0 \leq r < y) \} \)

Example: Division

- Code before the loop
  \( \{ (x \geq 0) \land (y > 0) \} \)
  \( q := 0; \)
  \( r := x; \)
  \( \{ (x=q\cdot y+r) \land (0 \leq r) \} \) - the invariant

- Proof: assignment, composition, and consequence lead to
  \( (x \geq 0) \land (y > 0) \Rightarrow (x = 0 \cdot y + x) \land (0 \leq x) \)
  obviously true

Example: Division

Need: \( \{ I \land b \} S \{ I \} \)
\( \{ (x=q\cdot y+r) \land (0 \leq r) \land (r - y \geq 0) \} \)
\( q := q + 1; \)
\( r := r - y \)
\( \{ (x=q\cdot y+r) \land (0 \leq r) \} \)

- Eventually we have the implication
  \( (x = q\cdot y + r) \land (0 \leq r) \land (r - y \geq 0) \Rightarrow \)
  \( (x = (q+1)\cdot y + r - y) \land (r - y \geq 0) \)

Simple arithmetic proves this

Example: Division

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

\( (x=q\cdot y+r) \land (0 \leq r) \land (r - y < 0) \Rightarrow \)

\( (x=q\cdot y+r) \land (0 \leq r < y) \)

Trivially true

Example: Fibonacci Numbers

\( \{ n > 0 \} \)

Math definition:
\( \text{fib}(1) = 1 \)
\( \text{fib}(2) = 1 \)

\( \ldots \)

while \( i > 1 \) do
  \( h := h + f; \)
  \( f := h - f; \)
  \( i := i - 1 \)
end

\( \{ f = \text{fib}(n) \} \)
Example: Fibonacci Numbers

- Invariant: \( f = \text{fib}(n-i+1) \land h = \text{fib}(n-i+2) \land i>0 \)
- Steps
  
  \( n>0 \Rightarrow 1 = \text{fib}(n-n+1) \land 1 = \text{fib}(n-n+2) \land n>0 \)
  
  \( i:=n; f:=1; h:=1 \)
  
  \{ f = \text{fib}(n-i+1) \land h = \text{fib}(n-i+2) \land i>0 \} \text{ [invariant]}

  start of loop

  \{ f = \text{fib}(n-i+1) \land h = \text{fib}(n-i+2) \land i>0 \land i>1 \} \Rightarrow \{ h = \text{fib}(n-i+2) \land h+f = \text{fib}(n-i+3) \land (i-1)>0 \} \Rightarrow \{ h+f-f = \text{fib}(n-(i-1)+1) \land h+f = \text{fib}(n-(i-1)+2) \land (i-1)>0 \}

Example: I/O

\{ \text{IN}=<1,2,\ldots,100> \land \text{OUT}=<> \} \\\nread x; \\\nwhile x\neq100 do \\\n  write x; \\\n  read x; \\\nend \\\n\{ \text{OUT} = <1,2,\ldots,99> \} \\

Example: Fibonacci Numbers

\{ h+f-f = \text{fib}(n-(i-1)+1) \land h+f = \text{fib}(n-(i-1)+2) \land (i-1)>0 \} \\\nh:=h+f; \\\n\{ h-f = \text{fib}(n-(i-1)+1) \land h = \text{fib}(n-(i-1)+2) \land (i-1)>0 \} \Rightarrow \{ h = \text{fib}(n-(i-1)+2) \land h+f = \text{fib}(n-(i-1)+3) \land (i-1)>0 \} \Rightarrow \{ h+f-f = \text{fib}(n-(i-1)+1) \land h+f = \text{fib}(n-(i-1)+2) \land (i-1)>0 \}

Example: I/O

\{ \text{IN}=<1,2,\ldots,100> \land \text{OUT}=<> \} \\\nread x; \\\nwhile x\neq100 do \\\n  write x; \\\n  read x; \\\nend \\\n\{ \text{OUT} = <1,2,\ldots,99> \}

Proof

Loop invariant: \( \text{OUT}^x \text{IN} = <1,2,\ldots,100> \)
\{ \text{IN}=<1,2,\ldots,100> \land \text{OUT}=<> \} \text{ read x;}
\{ x=1 \land \text{IN}=<2,\ldots,100> \land \text{OUT}=<> \}
\{ \text{I} \land x\neq100 \} \Rightarrow \text{OUT}^x \text{head(IN)} \text{tail(IN)} = <1,\ldots,100>
\{ \text{OUT}^x \text{p} \} \text{ write } x \{ \text{p} \}:
\{ \text{OUT}^x \text{head(IN)} \text{tail(N)} = <1,2,\ldots,100> \}
\{ \text{IN}^x \text{head(IN)} \text{head(N)} \} \text{ read } x \{ \text{p} \}:
\{ \text{OUT}^x \text{IN} = <1,2,\ldots,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 & Fibonacci
- 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

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

Total Correctness Rule

- New assignment axiom
  \[
p \Rightarrow (D(e) \land q[e/x])
  \]
  \[
  \langle p \mid x := e \mid q \rangle
  \]
  where \( D(e) \) means "e is well-defined"
- New read axiom
  \[
p \Rightarrow (IN \not\equiv \land (q[tail(IN)/IN][head(IN)/x]) \]
  \[
  \langle p \mid read x \mid q \rangle
  \]

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"
  \[
  \langle I \land b \rangle \Rightarrow f > 0
  \]
  \[
  \langle I \land b \land f = k \mid S \mid I \land f < k \rangle
  \]
  \[
  \langle I \mid while b do S end \mid I \land \neg b \rangle
  \]

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

- \[
  \langle s = 0 \land x = 0 \mid while x \neq 10 \ do \ x := x + 1; \ s := s + x \ end \mid s = \sum_{k=0}^{10} k \rangle
  \]
  Invariant: \( 0 \leq x \leq 10 \land s = \sum_{k=0}^{x-1} k \)
  Progress function: \( 10-x \)
Other Total Correctness Rules

- Essentially identical: e.g.

\[
\langle p \mid S_1 \mid q \rangle \langle q \mid S_2 \mid r \rangle \\
\langle p \mid S_1; S_2 \mid r \rangle
\]

Outline

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