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 with 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 or x=1, y=2, z=2 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
  \[ < \land \lor \neg \exists \forall \text{ true false} \]

- 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 $\exists i. k=i*j$?
When Does a State Satisfy an Assertion?

- \{ p \} denotes the set of states that satisfy assertion \( p \)
- \( \{p \lor q\} \iff \{p\} \cup \{q\} ; \{p \land q\} \iff \{p\} \cap \{q\} \)
- \( \{\neg p\} \iff U - \{p\} \) (\( U \) 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 \}
- \{ \text{true} \}
- \{ x \neq x \}
- \{ \text{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 \}
- \{ \text{true} \}
- \{ x \neq x \}
- \{ \text{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
- \( \{ \text{true} \} \): 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)

\[ \begin{align*}
<c>_1 & ::= \text{skip} \mid \text{id} := <ae> \mid <c>_2 ; <c>_3 \\
& \quad \mid \text{if} <be> \text{ then } <c>_2 \text{ else } <c>_3 \\
& \quad \mid \text{while} <be> \text{ do } <c>_2 \\
\end{align*} \]

\[ \begin{align*}
<ae>_1 & ::= \text{id} \mid \text{int} \mid <ae>_2 + <ae>_3 \\
& \quad \mid <ae>_2 - <ae>_3 \mid <ae>_2 \ast <ae>_3 \\
\end{align*} \]

\[ \begin{align*}
<be>_1 & ::= \text{true} \mid \text{false} \\
& \quad \mid <ae>_1 = <ae>_2 \mid <ae>_1 < <ae>_2 \\
& \quad \mid \neg <be>_2 \mid <be>_2 \land <be>_3 \\
& \quad \mid <be>_2 \lor <be>_3 \\
\end{align*} \]
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 \(\sigma\) 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\} 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 \(\sigma\)
  - 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 \{false\} S \{q\} valid?
Examples

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

- \{ x=1 \} \text{ skip } \{ x=1 \} \quad \text{Valid}
- \{ x=1 \land y=1 \} \text{ skip } \{ x=1 \} \quad \text{Valid}
- \{ x=1 \} \text{ skip } \{ x=1 \land y=1 \} \quad \text{Invalid}
- \{ x=1 \} \text{ skip } \{ x=1 \lor y=1 \} \quad \text{Valid}
- \{ x=1 \lor y=1 \} \text{ skip } \{ x=1 \} \quad \text{Invalid}
- \{ x=1 \} \text{ skip } \{ \text{true} \} \quad \text{Valid}
- \{ x=1 \} \text{ skip } \{ \text{false} \} \quad \text{Invalid}
- \{ \text{false} \} \text{ skip } \{ x=1 \} \quad \text{Valid}
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 \} \text{ while } x<10 \text{ do } x:=x+1 \ \{ x=10 \}
- \{ x<0 \} \text{ while } x<10 \text{ do } x:=x+1 \ \{ x=10 \}
- \{ x \geq 0 \} \text{ while } x<10 \text{ do } x:=x+1 \ \{ x=10 \}
- \{ x \geq 0 \} \text{ while } x<10 \text{ do } x:=x+1 \ \{ x \geq 10 \}
More Examples

- \{ x=1 \land y=2 \} x := x+1 \{ x=2 \land y=2 \} Valid
- \{ x=1 \land y=2 \} x := x+1 \{ x \geq 2 \} Valid
- \{ x=1 \land 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\geq0 \} while x<10 do x:=x+1 \{ x=10 \} Invalid
- \{ x\geq0 \} while x<10 do x:=x+1 \{ x\geq10 \} 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 \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

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

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

\[ \begin{align*}
  x=1 \land y=2 & \Rightarrow x=1 \\
  \{ x=1 \} \text{ skip } \{ x=1 \} \\
  \{ x=1 \land y=2 \} \text{ skip } \{ x=1 \}
\end{align*} \]
Simplified Versions

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

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

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

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

\[
\begin{align*}
\{ p \} & \text{ S } \{ q \} \quad q' \Rightarrow q \\
\hline
\{ p \} & \text{ S } \{ q' \}
\end{align*}
\]
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
  - $(x=y)[5/x] \Rightarrow 5=y$, $(x=y \land x=2)[5/x] \Rightarrow 5=y \land 5=2$
  - $(x=k \land \exists k.a_k>x)[y/k] \Rightarrow (x=y \land \exists k.a_k>x)$
  - $(x=k \land \exists k.a_k>x)[k/x] \Rightarrow (k=k \land \exists j.a_j>k)$
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

- $\exists i. k = i \cdot j$: $k$ and $j$ are free, $i$ is bound

- $(x+1 < y+2) \land (\exists x. x+3 = y+4)$

- **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

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

- \{ 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

\[
\begin{array}{ccc}
\{ p \} & S1 & \{ q \} \\
& \downarrow & \downarrow \\
\{ q \} & S2 & \{ r \} \\
\hline
\{ p \} & S1;S2 & \{ r \}
\end{array}
\]

- Example

\[
\begin{array}{c}
\{ x+1=y+z \} & \text{skip} & \{ x+1=y+z \} \\
\{ x=\text{skip} \} & \text{:=} & \{ x=y+z \} \\
\hline
\{ x+1=y+z \} & \text{skip;} & \{ x=\text{:=}x+1 \} \\
\{ x=y+z \}
\end{array}
\]
Input/Output

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

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

- Example

\begin{align*}
\text{OUT}=&<> \Rightarrow \text{OUT}^4=&<4> \quad \{\text{OUT}^4=&<4>\} \text{ write } 4 \{\text{OUT}=&<4>\} \\
\end{align*}

\begin{align*}
\{ \text{OUT}=&<> \} \text{ write } 4 \{\text{OUT}=&<4>\}
\end{align*}
Read Axiom

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

\[
\{ \text{tail(IN)}=\langle 4 \rangle \land \text{head(IN)}=3 \} \text{ read } x \{ \text{IN}=\langle 4 \rangle \land x=3 \}
\]

\[
\text{IN}=\langle 3,4 \rangle \implies \text{tail(IN)}=\langle 4 \rangle \land \text{head(IN)}=3
\]

\[
\{ \text{IN}=\langle 3,4 \rangle \} \text{ read } x \{ \text{IN}=\langle 4 \rangle \land x=3 \}
\]
Alternative Notation

- write axiom
  \[ \{ p^{OUT}_{OUT^e} \} \text{ write } e \{ p \} \]

- read axiom
  \[ \{ (p^IN_{\text{tail(IN)}})^x_{\text{head(IN)}} \} \text{ read } x \{ p \} \]
Example

Prove

\{ \text{IN} = \{3, 4\} \land \text{OUT} = \{\} \}
read x;
read y;
write x+y;
\{ \text{OUT} = \{7\} \}
Example

1. Using the write axiom and the postcondition:
   \[ \{ \text{OUT}^{x+y} = <7> \} \ \text{write} \ x+y \ \{ \text{OUT} = <7> \} \]

2. Using (1) and the rule of consequence:
   \[ \{ x+y = 7 \land \text{OUT} = <> \} \ \text{write} \ x+y \ \{ \text{OUT} = <7> \} \]

3. Using read axiom:
   \[ \{ x + \text{head(IN)} = 7 \land \text{OUT} = <> \} \ \text{read} \ y \ \{ x + y = 7 \land \text{OUT} = <> \} \]

4. Using (2), (3), and sequential composition:
   \[ \{ x + \text{head(IN)} = 7 \land \text{OUT} = <> \} \ \text{read} \ y; \ \text{write} \ x+y \ \{ \ \text{OUT} = <7> \} \]
Example

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

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

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

\[
\{ p \land b \} \quad S1 \quad \{ q \} \quad \{ p \land \neg b \} \quad S2 \quad \{ q \}
\]

\[
\{ p \} \quad \text{if } b \text{ then } S1 \text{ else } S2 \quad \{ q \}
\]

Example:

\[
\{ y = 1 \}
\]

if \( y = 1 \) then \( x := 1 \) else \( x := 2 \)

\[
\{ x = 1 \}
\]
If-Then-Else Example

\[ y=1 \land y=1 \implies 1=1 \quad \{ 1=1 \} \quad x:=1 \quad \{ x=1 \} \]
\[ \{ y=1 \land y=1 \} \quad x:=1 \quad \{ x=1 \} \]

\[ y=1 \land \neg(y=1) \implies 2=1 \quad \{ 2=1 \} \quad x:=2 \quad \{ x=1 \} \]
\[ \{ y=1 \land \neg(y=1) \} \quad x:=2 \quad \{ x=1 \} \]

\{y=1\} \text{ if } y=1 \text{ then } x:=1 \text{ else } x:=2 \quad \{ x=1 \}
Simplified If-Then-Else Rule

- Why not simply

\[
\begin{align*}
\{ p \} S1 \{ q \} & \quad \{ p \} S2 \{ q \} \\
\{ p \} & \text{if } b \text{ then } S1 \text{ else } S2 \{ q \}
\end{align*}
\]

- Works for

\{true\} if \(y=1\) then \(x:=1\) else \(x:=2\) \(\{x=1 \lor x=2\}\)

- Easy to prove that

\[
\begin{align*}
\{ \text{true} \} & \ x:=1 \ \{ x = 1 \ \lor \ x = 2 \} \\
\{ \text{true} \} & \ x:=2 \ \{ x = 1 \ \lor \ x = 2 \} \\
& \text{with assignment axiom and consequence}
\end{align*}
\]
Simplified If-Then-Else Rule

- Does not work for

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

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

\[
\begin{align*}
\{ I \land b \} & \Rightarrow S \{ I \} \\
\{ I \} \text{ while } b \text{ do } S \text{ end } & \Rightarrow \{ I \land \neg b \}
\end{align*}
\]

- In practice often combined with the rule of consequence

\[
\begin{align*}
p \Rightarrow I & \quad \{ I \land b \} \Rightarrow S \{ I \} \\
(I \land \neg b) & \Rightarrow q \\
\{ p \} \text{ while } b \text{ do } S \text{ end } & \Rightarrow \{ q \}
\end{align*}
\]
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

- **Prove**

\[
\{ \ (x \geq 0) \land (y > 0) \ \} \\
q := 0; \\
r := x; \\
\textbf{while} \ (r - y) \geq 0 \ \textbf{do} \\
\qquad q := q + 1; \\
\qquad r := r - y \\
\textbf{end} \\
\{ \ (x = q \cdot y + r) \land (0 \leq r < y) \ \} \\
\]

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
  \[(x=q*y+r) \land (0 \leq r)\]
Example: Division

\[
\{ (x \geq 0) \land (y > 0) \}\n\]
\[
q := 0;
\]
\[
r := x;
\]
\[
\{ (x=q*y+r) \land (0 \leq r) \}\n\]
\[
\text{while } (r - y) \geq 0 \text{ do }
\]
\[
\quad q := q + 1;
\]
\[
\quad r := r - y
\]
\[
\text{end}
\]
\[
\{ (x=q*y+r) \land (0 \leq r) \land (r-y<0))\}
\]
\[
\{ (x=q*y+r) \land (0 \leq r < y) \}\}
Example: Division

- Code before the loop
  \[
  \begin{align*}
  \{ \ (x \geq 0) \ &\land \ (y > 0) \ \} \\
  q &:= 0; \\
r &:= x; \\
\{ \ (x=q\times y+r) \ &\land \ (0 \leq r) \ \} \ - \ the \ invariant
  \end{align*}
  \]

- Proof: assignment, composition, and consequence lead to

\[(x \geq 0) \ \land \ (y > 0) \ \Rightarrow \ (x=0\times y+x) \ \land \ (0 \leq x) \]

obviously true
Example: Division

Need: \( \{ I \land b \} \rightarrow S \{ I \} \)

\( \{ (x=q*y+r) \land (0\leq r) \land (r-y \geq 0) \} \)

\( q := q + 1; \)

\( r := r - y \)

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

- Eventually we have the implication

\( (x=q*y+r) \land (0\leq r) \land (r-y \geq 0) \implies (x=(q+1)*y+r-y) \land (r-y \geq 0) \)

Simple arithmetic proves this
Example: Division

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

\[(x=q\times y+r) \land (0 \leq r) \land (r-y<0) \Rightarrow (x=q\times y+r) \land (0 \leq r< y)\]

Trivially true
Example: Fibonacci Numbers

\{ n > 0 \}
\begin{align*}
i &:= n; \\
f &:= 1; \\
h &:= 1; \\
\text{while } i > 1 \text{ do} \\
& \quad h := h + f; \\
& \quad f := h - f; \\
& \quad i := i - 1 \\
\end{align*}
\{ f = \text{fib}(n) \}

Math definition:
\begin{align*}
\text{fib}(1) &= 1 \\
\text{fib}(2) &= 1 \\
\text{fib}(i+1) &= \text{fib}(i) + \text{fib}(i-1) \\
\end{align*}
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: Fibonacci Numbers

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

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

\[ \{ \text{f=fib(n-(i-1)+1)} \land \text{h=fib(n-(i-1)+2)} \land (i-1)>0 \} \]
\text{i:=i-1 - after this, we get the loop invariant}

end of loop: \[ \{ \text{f=fib(n-i+1)} \land \text{h=fib(n-i+2)} \land i>0 \land i\leq1 \} \Rightarrow f=fib(n) \]
Example: I/O

\{ \text{IN}=\{1,2,\ldots,100\} \land \text{OUT}=\langle\rangle \} \\
read x; \\
while x \neq 100 do \\
\quad write x; \\
\quad read x; \\
\end \\
\{ \text{OUT} = \{1,2,\ldots,99\} \}
Proof

Loop invariant: \( OUT^{x^IN} = <1,2,\ldots,100> \)
{ \( IN=\langle 1,2,\ldots,100 \rangle \land OUT=\langle \rangle \) } read \( x \);
{ \( x=1 \land IN=\langle 2,\ldots,100 \rangle \land OUT=\langle \rangle \) }
{ \( I \land x\neq 100 \) } write \( x \); read \( x \); { \( I \) }

\( I \land x\neq 100 \Rightarrow OUT^{x^\text{head(IN)}^\text{tail(IN)}}=\langle 1,\ldots,100 \rangle \)
{ \( p_{\text{OUT}^x} \) } write \( x \) { \( p \) }:

{ \( OUT^{\text{head(IN)}^\text{tail(IN)}} = \langle 1,2,\ldots,100 \rangle \) }

{ \( (p_{\text{IN}^\text{tail(IN)}}^x)_{\text{head(IN)}} \) } read \( x \) { \( p \) }:
{ \( OUT^{x^IN} = \langle 1,2,\ldots,100 \rangle \) }
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

- \( \langle p \mid S \mid q \rangle \)
  - \( S \) is a piece of code (program fragment)
  - \( p \): pre-condition, \( q \): post-condition

- If we start executing \( S \) from any state \( \sigma \) that satisfies \( p \), then \( S \) terminates and the resulting state \( \sigma' \) 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**
  \[ p \Rightarrow (IN\neq\leftrightarrow \land (q[tail(IN)/IN])[head(IN)/x]) \]
  \[
  \langle p \mid \text{read } x \mid q \rangle
  \]
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”

\[
(I \land b) \implies f > 0 \quad \langle I \land b \land f = k \mid S \mid I \land f < k \rangle
\]

\[
\langle I \mid \text{while } b \text{ do } S \text{ end} \mid I \land \neg b \rangle
\]
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

\[ \langle s = 0 \land x = 0 \mid \text{while } x \neq 10 \text{ do } x := x + 1; s := s + x \text{ end} \mid s = \sum_{k=0}^{10} k \rangle \]

Invariant: \( 0 \leq x \leq 10 \land s = \sum_{k=0}^{x} k \)

Progress function: \( 10 - x \)
Other Total Correctness Rules

- Essentially identical: e.g.

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

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