

Axiomatic Semantics

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

1

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
 - Purely mathematical formalism: an example of a proof system
- Should be able to prove all true statements about the program, and not be able to prove any false statements

2

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** $\rightarrow \{0, -1, +1, -2, 2, \dots\}$

3

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
 $\{ x=1 \wedge y=2 \wedge 1 \leq z \leq 3 \}$
- An assertion represents the set of states that satisfy the assertion

4

Use of First-Order Predicate 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
 $= < \wedge \vee \neg \Rightarrow \exists \forall$ true false
- Operations from the programming language: e.g. +, -, ...

5

First-Order Predicate 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, \dots
- Formulas
 - true and false
 - $t_1 < t_2$ and $t_1 = t_2$ for terms t_1 and t_2
 - $f_1 \wedge f_2, f_1 \vee f_2, \neg f_1, f_1 \Rightarrow f_2$ for formulas f_1, f_2
 - $\exists x.f$ and $\forall x.f$ for a formula f

6

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)

7

States and Assertion

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

8

States and Assertions

- 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 $\exists i.k=i*j$?

9

States and Assertions

- $\{ p \}$ denotes the set P 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)
- $\{ p \Rightarrow q \}$: same as $\{ \neg p \vee q \}$
 - What is $\{ x=2 \wedge y=3 \Rightarrow x=2 \}$?
- Suppose that $p \Rightarrow q$ is true; then $P \subseteq Q$
 - $x=2 \wedge y=3 \Rightarrow x=2$, so $\{ x=2 \wedge y=3 \} \subseteq \{ x=2 \}$

10

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

11

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

12

Simple Imperative Language (IMP)

```
<c>1 ::= skip | <id> := <ae> | <c>2 ; <c>3
      | if <be> then <c>2 else <c>3
      | while <be> do <c>2
<ae>1 ::= <id> | <int> | <ae>2 + <ae>3
      | <ae>2 - <ae>3 | <ae>2 * <ae>3
<be>1 ::= true | false
      | <ae>1 = <ae>2 | <ae>1 < <ae>2
      | ¬ <be>2 | <be>2 ∧ <be>3
      | <be>2 ∨ <be>3
```

13

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"

14

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
- If **{p} S {q}** and **{p} T {q}**, S and T are semantically equivalent (w.r.t. p)

15

Valid Results

- A result $\{p\} S \{q\}$ is **valid** if and only if for every state σ
 - if σ satisfies p
 - and the execution of S starting in σ terminates in state σ'
 - then σ' satisfies q
- Is $\{\text{false}\} S \{q\}$ valid?

16

Examples

- $\{x=1\} \text{skip } \{x=1\}$ Valid
- $\{x=1 \wedge y=1\} \text{skip } \{x=1\}$ Valid
- $\{x=1\} \text{skip } \{x=1 \wedge y=1\}$ Invalid
- $\{x=1\} \text{skip } \{x=1 \vee y=1\}$ Valid
- $\{x=1 \vee y=1\} \text{skip } \{x=1\}$ Invalid
- $\{x=1\} \text{skip } \{\text{true}\}$ Valid
- $\{x=1\} \text{skip } \{\text{false}\}$ Invalid
- $\{\text{false}\} \text{skip } \{x=1\}$ Valid

17

More Examples

- $\{x=1 \wedge y=2\} x := x+1 \{x=2 \wedge y=2\}$ Valid
- $\{x=1 \wedge y=2\} x := x+1 \{x \geq 2\}$ Valid
- $\{x=1 \wedge y=2\} x := x+1 \{x=y\}$ Valid

- $\{x=0\} \text{while } x < 10 \text{ do } x := x+1 \{x=10\}$ Valid
- $\{x < 0\} \text{while } x < 10 \text{ do } x := x+1 \{x=10\}$ Valid
- $\{x \geq 0\} \text{while } x < 10 \text{ do } x := x+1 \{x=10\}$ Invalid
- $\{x \geq 0\} \text{while } x < 10 \text{ do } x := x+1 \{x \geq 10\}$ Valid

18

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 $x := x+1$ { $x=10$ }
- $\{ \text{true} \}$ while $x \neq 10$ do $x := x+1$ { $x=10$ }
- All of these results are **valid**

19

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: derive valid results **without** using operational reasoning
 - Purely formally, using a proof system

20

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
 - " $\{p\} S \{q\}$ is true", " $\{p\} S \{q\}$ is valid for *some* states", "assertion p is not valid"

21

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

22

Post System

- Post system: purely formal, unrelated to programming languages
 - Based on the work of the logician Emil Post
 - Alphabet of **symbols**
 - Set of **variables**
 - **Term**: string of symbols and variables
 - **Word**: string of symbols
- A Post system can be used to express **derivations (proofs)** of terms

23

Productions

- Also called "inference rules"
 - $$\frac{t_1 \ t_2 \ \dots \ t_n}{t}$$

t_i and t : terms
 t_i : premises
 t : **conclusion** - if all premises are true, so is the conclusion
- **Axiom**: rule with no premises
- A production is a concise representation of a set of **production instances**
 - Production instance: each variable is replaced with a string of symbols (a word)

24

Proofs

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

25

Example: Unary Numbers

- Alphabet {N, |}
- Rules
 - x is a variable
- Proof

\bar{N}	$\frac{Nx}{Nx }$
-----------	------------------

$\frac{\bar{N}}{\frac{\bar{N}|}{\bar{N}||}}$

26

Unary Number Addition

- Alphabet {N, |, +, =}
- Rules
- Proof

\bar{N}	$\frac{Nx}{Nx }$
$\frac{Ny}{+y=y}$	$\frac{x+y=z}{x +y=z }$

$\frac{\bar{N}}{\frac{\bar{N}|}{\frac{\bar{N}||}{\frac{+||=||}{\frac{|+|=|||}{||+|=||||}}}}}$

27

Better Rules

- Alphabet
 - {N, |, +, =}
- Rules

\overline{N}	$\frac{Nx}{Nx }$
$\frac{Nx \ Ny}{x+y=xy}$	

$$\frac{\overline{N} \quad \overline{N}}{\overline{N|} \quad \overline{N|}}$$

$$\frac{\overline{N|} \quad \overline{N|}}{||+||=||||}$$
- Proof

28

Proof System for IMP

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

$\frac{\{ p \} \text{ skip } \{ p \}}{\text{term in the Post system}}$
- Examples
 - $\{ x=1 \} \text{ skip } \{ x=1 \}$ **Provable**
 - $\{ x=1 \} \text{ skip } \{ x=1 \wedge y=2 \}$ **Not provable** (good)
 - $\{ x=1 \wedge y=2 \} \text{ skip } \{ x=1 \}$ **Not provable** (bad)

29

Inference Rule of Consequence

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

- $x \Rightarrow y$ means $X \subseteq Y$
- production: p, p', q, q', S are variables
- axiom instance

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

production instance: each variable is replaced with a word

30

Simplified Versions

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

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

31

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

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

32

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
 - $(x=y)[5/x] \Rightarrow 5=y$, $(x=y \wedge x=2)[5/x] \Rightarrow 5=y \wedge 5=2$
 - $(x=k \wedge \exists k.a_k \triangleright x)[y/k] \Rightarrow (x=y \wedge \exists k.a_k \triangleright x)$
 - $(x=k \wedge \exists k.a_k \triangleright x)[k/x] \Rightarrow (k=k \wedge \exists j.a_j \triangleright k)$

33

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 $\text{true} \Rightarrow y+z = y+z$ and the consequence rule: $\{ \text{true} \} x := y+z \{ x = y+z \}$

34

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"

35

Inference Rule of Composition

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

- Example

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

36

Input/Output

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

37

Write Axiom

$\{ p[OUT \hat{=} e / OUT] \} \text{ write } e \{ p \}$

- Example

$OUT = \langle \rangle \Rightarrow OUT \hat{=} 4 = \langle 4 \rangle \quad \{ OUT \hat{=} 4 = \langle 4 \rangle \} \text{ write } 4 \{ OUT = \langle 4 \rangle \}$
 $\{ OUT = \langle \rangle \} \text{ write } 4 \{ OUT = \langle 4 \rangle \}$

38

Read Axiom

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

$\{ tail(IN) = \langle 4 \rangle \wedge head(IN) = 3 \} \text{ read } x \{ IN = \langle 4 \rangle \wedge x = 3 \}$

$IN = \langle 3, 4 \rangle \Rightarrow tail(IN) = \langle 4 \rangle \wedge head(IN) = 3$

$\{ IN = \langle 3, 4 \rangle \} \text{ read } x \{ IN = \langle 4 \rangle \wedge x = 3 \}$

39

Alternative Notation

- write axiom

$\{ p_{OUT^e}^{OUT} \} \text{ write } e \{ p \}$

- read axiom

$\{ (p_{tail(IN)}^{IN})^x_{head(IN)} \} \text{ read } x \{ p \}$

40

Example

- Prove

$\{ IN = \langle 3,4 \rangle \wedge OUT = \langle \rangle \}$

read x;

read y;

write x+y;

$\{ OUT = \langle 7 \rangle \}$

41

Example

1. Using the write axiom and the postcondition:

$\{ OUT^{(x+y)} = \langle 7 \rangle \} \text{ write } x+y \{ OUT = \langle 7 \rangle \}$

2. Using (1) and the rule of consequence:

$\{ x+y=7 \wedge OUT = \langle \rangle \} \text{ write } x+y \{ OUT = \langle 7 \rangle \}$

3. Using read axiom:

$\{ x+head(IN)=7 \wedge OUT = \langle \rangle \} \text{ read } y \{ x+y=7 \wedge OUT = \langle \rangle \}$

4. Using (2), (3), and sequential composition:

$\{ x+head(IN)=7 \wedge OUT = \langle \rangle \}$

read y; write x+y { OUT = $\langle 7 \rangle$ }

42

Example

5. Using the read axiom:
 $\{ \text{head}(\text{IN}) + \text{head}(\text{tail}(\text{IN})) = 7 \wedge \text{OUT} = \langle \rangle \}$
`read x`
 $\{ x + \text{head}(\text{IN}) = 7 \wedge \text{OUT} = \langle \rangle \}$
6. Using (5) and the rule of consequence
 $\{ \text{IN} = \langle 3, 4 \rangle \wedge \text{OUT} = \langle \rangle \}$
`read x`
 $\{ x + \text{head}(\text{IN}) = 7 \wedge \text{OUT} = \langle \rangle \}$
7. Using (4), (6), and sequential composition
 $\{ \text{IN} = \langle 3, 4 \rangle \wedge \text{OUT} = \langle \rangle \}$
`read x; read y; write x+y;`
 $\{ \text{OUT} = \langle 7 \rangle \}$

43

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"

44

If-Then-Else Rule

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

Example:

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

45

If-Then-Else Example

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

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

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

46

Simplified If-Then-Else Rule

- Why not simply

$$\frac{\{p\} S1 \{q\} \quad \{p\} S2 \{q\}}{\{p\} \text{ if } b \text{ then } S1 \text{ else } S2 \{q\}}$$
- Works for

$$\{\text{true}\} \text{ if } y=1 \text{ then } x:=1 \text{ else } x:=2 \{x=1 \vee x=2\}$$
- Easy to prove that
 - $\{\text{true}\} x:=1 \{x = 1 \vee x = 2\}$
 - $\{\text{true}\} x:=2 \{x = 1 \vee x = 2\}$
 - with assignment axiom and consequence

47

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 is impossible to prove something that is operationally valid

48

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
 - I must be true at beginning of the loop
 - I must be true after each iteration
 - I must be true at end of the loop
- Finding a loop invariant is the hard part

49

Simplified While Loop Rule

- Initial idea
 $\{ I \} S \{ I \}$
 $\{ I \} \text{ while } b \text{ do } S \text{ end } \{ I \wedge \neg b \}$
- We can prove
 $\{ \text{true} \} \text{ while } x \neq 10 \text{ do } x := x + 1 \text{ end } \{ x = 10 \}$
using the invariant $\{ \text{true} \}$

50

Simplified While Loop Rule

- The semantics is incomplete: can't prove
 $\{ x=0 \wedge y=0 \} \text{ while } x \neq 0 \text{ do } y := y + 1 \text{ end } \{ x=0 \wedge y=0 \}$
Suppose there exists an invariant p'
 1. $x=0 \wedge y=0 \Rightarrow p'$
 2. $\{ p' \} y := y + 1 \{ p' \}$
 3. $p' \wedge x=0 \Rightarrow x=0 \wedge y=0$
 4. State $\langle x=0, y=0 \rangle$ belongs to P' (by 1)
 5. State $\langle x=0, y=1 \rangle$ belongs to P' (by 2) and therefore to the subset of P' for which $x=0$
 6. By 3 and 5: $x=0 \wedge y=1 \Rightarrow x=0 \wedge y=0$

51

While Loop Rule

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

- In practice usually combined with the rule of consequence

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

52

Example: Division

- Prove

$$\{ (x \geq 0) \wedge (y > 0) \}$$

$q := 0;$

q: quotient

$r := x;$

r: reminder

while $(r - y) \geq 0$ **do**

$q := q + 1;$

$r := r - y$

end

$$\{ (x = q * y + r) \wedge (0 \leq r < y) \}$$

53

Example: Division

- Loop invariant
 - Should state relationship between variables used in loop
- $$(x = q * y + r)$$
- Needs some boundary conditions to make the proof work

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

54

Example: Division

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

55

Example: Division

- Code before the loop
 $\{ (x \geq 0) \wedge (y > 0) \}$
 $q := 0;$
 $r := x;$
 $\{ (x = q * y + r) \wedge (0 \leq r) \wedge (y > 0) \}$ - the invariant
- Proof: assignment, composition, and consequence lead to
 $(x \geq 0) \wedge (y > 0) \Rightarrow (x = 0 * y + x) \wedge (0 \leq x) \wedge (y > 0)$
 - obviously true

56

Example: Division

- Need: $\{ I \wedge b \} S \{ I \}$
- ```
{ (x=q*y+r) ∧ (0≤r) ∧ (y>0) ∧ (r-y≥0) }
q := q + 1;
r := r - y
{ (x=q*y+r) ∧ (0≤r) ∧ (y>0) }
```
- Eventually we have the implication  
 $(x = q * y + r) \wedge (0 \leq r) \wedge (y > 0) \wedge (r - y \geq 0) \Rightarrow$   
 $(x = (q + 1) * y + r - y) \wedge (y > 0) \wedge (r - y \geq 0)$   
Simple arithmetic proves this

57

---

---

---

---

---

---

---

---

### Example: Division

- At exit: need the implication  $(I \wedge \neg b) \Rightarrow q$   
 $(x=q*y+r) \wedge (0 \leq r) \wedge (y > 0) \wedge (r-y < 0) \Rightarrow$   
 $(x=q*y+r) \wedge (0 \leq r < y)$   
Trivially true

58

---

---

---

---

---

---

---

---

### 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)  
...

59

---

---

---

---

---

---

---

---

### Example: Fibonacci Numbers

- Invariant:  $\{f=fib(n-i+1) \wedge h=fib(n-i+2) \wedge i > 0\}$
- Steps  
 $n > 0 \Rightarrow 1=fib(n-n+1) \wedge 1=fib(n-n+2) \wedge n > 0$   
 $i:=n; f:=1; h:=1$   
 $\{ f=fib(n-i+1) \wedge h=fib(n-i+2) \wedge i > 0 \}$  [invariant]  
start of loop  
 $\{ f=fib(n-i+1) \wedge h=fib(n-i+2) \wedge i > 0 \wedge i > 1 \} \Rightarrow$   
 $\{ h=fib(n-i+2) \wedge h+f=fib(n-i+3) \wedge (i-1) > 0 \} \Rightarrow$   
 $\{ h+f-f=fib(n-(i-1)+1) \wedge h+f=fib(n-(i-1)+2) \wedge$   
 $(i-1) > 0 \}$

60

---

---

---

---

---

---

---

---

### Example: Fibonacci Numbers

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

61

---

---

---

---

---

---

---

---

### Example: Euclid's Algorithm

```
{ m ≠ 0 ∨ n ≠ 0 }
z := m;
r := n;
while r ≠ 0 do
 h := z rem r;
 z := r;
 r := h
end
{ z = gcd(m, n) }
```

62

---

---

---

---

---

---

---

---

### Example: Euclid's Algorithm

```
• Invariant: gcd(z,r) = gcd(m,n) ∧ (z≠0 ∨ r≠0)
m≠0 ∨ n≠0 ⇒
gcd(m,n) = gcd(m,n) ∧ (m≠0 ∨ n≠0)
z := m; r := n;
gcd(z,r) = gcd(m,n) ∧ (z≠0 ∨ r≠0) ∧ r≠0 ⇒
gcd(r, z rem r) = gcd(m,n) ∧ (r≠0 ∨ z rem r ≠ 0)
h := z rem r;
gcd(r,h) = gcd(m,n) ∧ (r≠0 ∨ h≠0)
z:= r; r:=h - gives us the loop invariant
gcd(z,r) = gcd(m,n) ∧ (z≠0 ∨ r≠0) ∧ r=0 ⇒
z = gcd(m,n)
```

63

---

---

---

---

---

---

---

---

### Example: I/O

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

64

---

---

---

---

---

---

---

---

### Proof

Loop invariant:  $OUT \hat{x} IN = \langle 1, 2, \dots, 100 \rangle$   
{ IN=<1,2,...,100> ^ OUT=<> } read x;  
{ x=1 ^ IN=<2,...,100> ^ OUT=<> }  
**{ I ^ x≠100 } write x; read x; { I }**  
 $I \wedge x \neq 100 \Rightarrow OUT \hat{x} head(IN) \hat{tail}(N) = \langle 1, \dots, 100 \rangle$   
{  $p_{OUT \hat{x}}$  } write x { p }:  
{  $OUT \hat{x} head(IN) \hat{tail}(N) = \langle 1, 2, \dots, 100 \rangle$  }  
{  $(p_{tail(IN) \hat{head}(IN)}^x)$  } read x { p }:  
{  $OUT \hat{x} IN = \langle 1, 2, \dots, 100 \rangle$  }     **^ x = 100**

65

---

---

---

---

---

---

---

---

### Exercise: Finish the Proof

```
{ IN=<1,2,...,100> }
s:=0; read x;
while x≠100 do
 s:=s+x;
 read x;
end
{ s = $\sum_{k=0}^{99} k$ }
```

Invariant:  $\#IN = 100 - x \wedge \forall k \leq \#IN. IN[k] = x + k$   
 $\wedge 1 \leq x \leq 100 \wedge s = \sum_{k=0}^{x-1} k$

66

---

---

---

---

---

---

---

---

### 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

67

---

---

---

---

---

---

---

---

### Example: If-Then-Else Rule

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

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

Later assignment: is this consistent? complete?

To be consistent: anything derivable with the new rule must be also derivable with the original rule (we know that the old rule is consistent)

68

---

---

---

---

---

---

---

---

### Total Correctness

- So far we only had **partial correctness**
- Want to handle
  - Reading from empty input
  - Division by zero and other run-time errors
  - Non-termination
- Want
  - $\{x = 3 \wedge y = 0\} z := x/y \{ \text{false} \}$
  - $\{ \text{IN} = \langle \rangle \} \text{read } x \{ \text{false} \}$
  - Idea: add sanity check to precondition

69

---

---

---

---

---

---

---

---

### 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:  $[p] S [q]$

70

---

---

---

---

---

---

---

---

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

71

---

---

---

---

---

---

---

---

### Total Correctness Rule for While

- Idea: find **termination function**  $f$ 
    - Decreases with every iteration
    - Always positive at start of loop body
    - Also called "**progress function**"
- $$\frac{(I \wedge b) \Rightarrow f > 0 \quad \langle I \wedge b \wedge f = k \mid S \mid I \wedge f < k \rangle}{\langle I \mid \text{while } b \text{ do } S \text{ end} \mid I \wedge \neg b \rangle}$$

72

---

---

---

---

---

---

---

---

### Examples of Termination Functions

- Division example
  - Remainder  $r$  decreases in every step and does not get negative
- Euclid's algorithm
  - Input may be negative: use  $\text{abs}(r)$
- Fibonacci numbers
  - There already is an explicit counter  $i$

73

---

---

---

---

---

---

---

---

### Example: Euclid's Algorithm

```
< m ≠ 0 ∨ n ≠ 0 |
z := m;
r := n;
while r ≠ 0 do
 h := z rem r;
 z := r;
 r := h
end
| z = gcd(m, n) >
```

74

---

---

---

---

---

---

---

---

### Example: Euclid's Algorithm

- Invariant:  $\text{gcd}(z,r)=\text{gcd}(m,n) \wedge (z \neq 0 \vee r \neq 0)$
  - Termination function:  $\text{abs}(r)$
  - Termination proof for while loop
- $I \wedge r \neq 0 \Rightarrow \text{abs}(r) > 0$
- $\text{gcd}(z,r) = \text{gcd}(m,n) \wedge (z \neq 0 \vee r \neq 0) \wedge r \neq 0 \wedge \text{abs}(r)=k$
- leads to  $\text{gcd}(r, z \text{ rem } r) = \text{gcd}(m,n) \wedge$   
 $(r \neq 0 \vee z \text{ rem } r \neq 0) \wedge \text{abs}(z \text{ rem } r) < k$

75

---

---

---

---

---

---

---

---

### Another Progress Function

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

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

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

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

Progress function:  $f(x) = 10 - x$

76

---

---

---

---

---

---

---

---

### Other Total Correctness Rules

- Essentially identical: e.g.

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

77

---

---

---

---

---

---

---

---

### Summary: Axiomatic Semantics

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

78

---

---

---

---

---

---

---

---

### 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

79

---

---

---

---

---

---

---

---

### 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

80

---

---

---

---

---

---

---

---