

Program Analysis and Verification

0368-4479

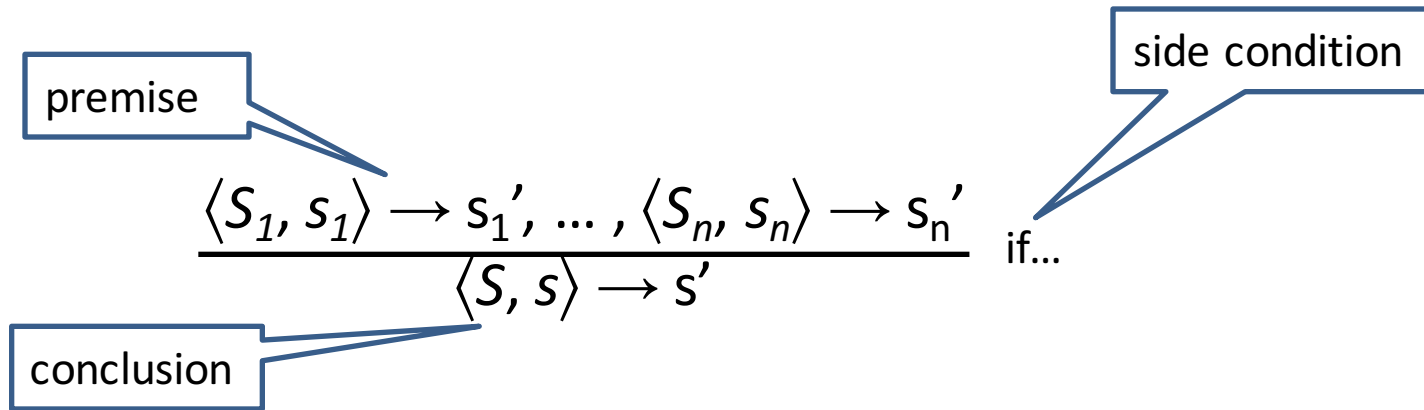
Noam Rinetzky

Lecture 3: Axiomatic Semantics

Slides credit: Tom Ball, Dawson Engler, Roman Manevich, Erik Poll,
Mooly Sagiv, Jean Souyris, Eran Tromer, Avishai Wool, Eran Yahav

Natural operating semantics

- \rightarrow defined by rules of the form



- The meaning of compound statements is defined using the meaning immediate constituent statements

Natural semantics for **While**

$$[\text{ass}_{\text{ns}}] \quad \langle x := a, s \rangle \rightarrow s[x \mapsto \mathcal{A}[[a]]s]$$

$$[\text{skip}_{\text{ns}}] \quad \langle \text{skip}, s \rangle \rightarrow s$$

axioms

$$[\text{comp}_{\text{ns}}] \quad \frac{\langle S_1, s \rangle \rightarrow s', \langle S_2, s' \rangle \rightarrow s''}{\langle S_1; S_2, s \rangle \rightarrow s''}$$

$$[\text{if}^{\text{tt}}_{\text{ns}}] \quad \frac{\langle S_1, s \rangle \rightarrow s'}{\langle \text{if } b \text{ then } S_1 \text{ else } S_2, s \rangle \rightarrow s'} \quad \text{if } \mathcal{B} [[b]] s = \text{tt}$$

$$[\text{if}^{\text{ff}}_{\text{ns}}] \quad \frac{\langle S_2, s \rangle \rightarrow s'}{\langle \text{if } b \text{ then } S_1 \text{ else } S_2, s \rangle \rightarrow s'} \quad \text{if } \mathcal{B} [[b]] s = \text{ff}$$

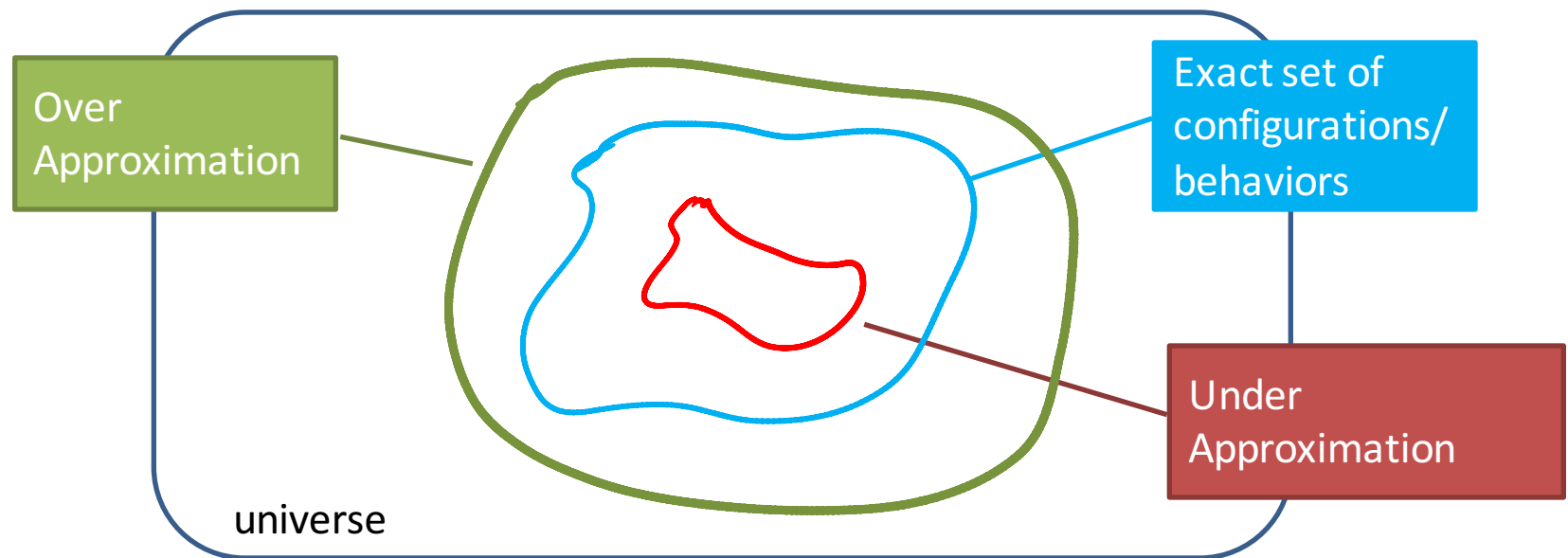
Natural semantics for **While**

$[\text{while}_{\text{ns}}^{\text{ff}}]$ $\langle \text{while } b \text{ do } S, s \rangle \rightarrow s$ if $\mathcal{B} \llbracket b \rrbracket s = \text{ff}$

Non-compositional

$[\text{while}_{\text{ns}}^{\text{tt}}]$
$$\frac{\langle S, s \rangle \rightarrow s', \langle \text{while } b \text{ do } S, s' \rangle \rightarrow s''}{\langle \text{while } b \text{ do } S, s \rangle \rightarrow s''}$$
 if $\mathcal{B} \llbracket b \rrbracket s = \text{tt}$

Verification by over-approximation



Axiomatic Semantics

Edsger W. Dijkstra



Robert W. Floyd



C.A.R. Hoare



Axiomatic Semantics

Edsger W. Dijkstra



Robert W. Floyd



C.A.R. Hoare



BTW, what do all these people have in common?

Axiomatic Semantics

Edsger W. Dijkstra



1972

For fundamental contributions to programming as a high, intellectual challenge; for eloquent insistence and practical demonstration that programs should be composed correctly, not just debugged into correctness; for illuminating perception of problems at the foundations of program design.

Robert W. Floyd



1978

For having a clear influence on methodologies for the creation of efficient and reliable software, and for helping to found the following important subfields of computer science: the theory of parsing, the semantics of programming languages, automatic program verification, automatic program synthesis, and analysis of algorithms.

C.A.R. Hoare



1980

For his fundamental contributions to the definition and design of programming languages.

Proving program correctness

- **Why** prove correctness?
- **What** is correctness?
- **How?**
 - Reasoning at the operational semantics level
 - Tedious
 - Error prone
 - Formal reasoning using “axiomatic” semantics
 - Syntactic technique (“game of tokens”)
 - Mechanically checkable
 - Sometimes automatically derivable

A simple imperative language: **While**

Abstract syntax:

$a ::= n \mid x \mid a_1 + a_2 \mid a_1 \star a_2 \mid a_1 - a_2$

$b ::= \mathbf{true} \mid \mathbf{false}$

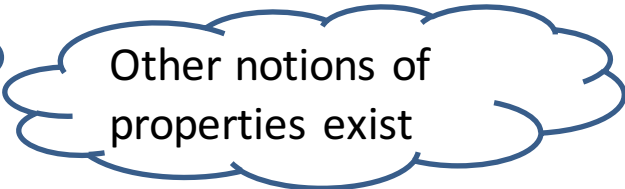
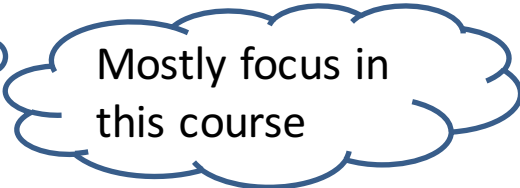
$\mid a_1 = a_2 \mid a_1 \leq a_2 \mid \neg b \mid b_1 \wedge b_2$

$S ::= x := a \mid \mathbf{skip} \mid S_1; S_2$

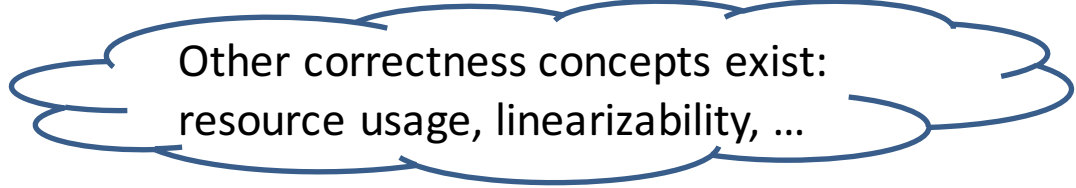
$\mid \mathbf{if } b \mathbf{ then } S_1 \mathbf{ else } S_2$

$\mid \mathbf{while } b \mathbf{ do } S$

Program correctness concepts

- **Property** = a certain relationship between initial state and final state 
- **Partial correctness** = properties that hold *if* program terminates 
- **Termination** = program always terminates
 - i.e., for every input state

partial correctness + termination = **total correctness**



Other correctness concepts exist:
resource usage, linearizability,...

Factorial example

$S_{\text{fac}} \equiv \mathbf{y} := 1; \text{ while } \neg(\mathbf{x}=1) \text{ do } (\mathbf{y} := \mathbf{y} * \mathbf{x}; \mathbf{x} := \mathbf{x} - 1)$

- $\langle S_{\text{fac}}, s \rangle \rightarrow s'$ implies $s' \mathbf{y} = (s \mathbf{x})!$

Factorial example

$S_{\text{fac}} \equiv \mathbf{y} := 1; \text{ while } \neg(\mathbf{x}=1) \text{ do } (\mathbf{y} := \mathbf{y} * \mathbf{x}; \mathbf{x} := \mathbf{x} - 1)$

- $\langle S_{\text{fac}}, s \rangle \rightarrow s'$ implies $s' \mathbf{y} = (s \mathbf{x})!$
- Factorial partial correctness property =
 - *if* the statement terminates *then* the final value of \mathbf{y} will be the factorial of the initial value of \mathbf{x}
 - What if $s \mathbf{x} < 0$?

Natural semantics for **While**

$$[\text{ass}_{\text{ns}}] \quad \langle x := a, s \rangle \rightarrow s[x \mapsto \mathcal{A}[[a]]s]$$

$$[\text{skip}_{\text{ns}}] \quad \langle \text{skip}, s \rangle \rightarrow s$$

$$[\text{comp}_{\text{ns}}] \quad \frac{\langle S_1, s \rangle \rightarrow s', \langle S_2, s' \rangle \rightarrow s''}{\langle S_1; S_2, s \rangle \rightarrow s''}$$

$$[\text{if}^{\text{tt}}_{\text{ns}}] \quad \frac{\langle S_1, s \rangle \rightarrow s'}{\langle \text{if } b \text{ then } S_1 \text{ else } S_2, s \rangle \rightarrow s'} \quad \text{if } \mathcal{B} [[b]] s = \text{tt}$$

$$[\text{if}^{\text{ff}}_{\text{ns}}] \quad \frac{\langle S_2, s \rangle \rightarrow s'}{\langle \text{if } b \text{ then } S_1 \text{ else } S_2, s \rangle \rightarrow s'} \quad \text{if } \mathcal{B} [[b]] s = \text{ff}$$

$$[\text{while}^{\text{ff}}_{\text{ns}}] \quad \langle \text{while } b \text{ do } S, s \rangle \rightarrow s \quad \text{if } \mathcal{B} [[b]] s = \text{ff}$$

$$[\text{while}^{\text{tt}}_{\text{ns}}] \quad \frac{\langle S, s \rangle \rightarrow s', \langle \text{while } b \text{ do } S, s' \rangle \rightarrow s''}{\langle \text{while } b \text{ do } S, s \rangle \rightarrow s''} \quad \text{if } \mathcal{B} [[b]] s = \text{tt}$$

Staged proof

The proof proceeds in three stages:

Stage 1: We prove that the body of the `while` loop satisfies:

$$\begin{aligned} &\text{if } \langle y := y \star x; x := x - 1, s \rangle \rightarrow s'' \text{ and } s'' \ x > 0 \\ &\text{then } (s \ y) \star (s \ x)! = (s'' \ y) \star (s'' \ x)! \text{ and } s \ x > 0 \end{aligned} \tag{*}$$

Stage 2: We prove that the `while` loop satisfies:

$$\begin{aligned} &\text{if } \langle \text{while } \neg(x=1) \text{ do } (y := y \star x; x := x - 1), s \rangle \rightarrow s'' \\ &\text{then } (s \ y) \star (s \ x)! = s'' \ y \text{ and } s'' \ x = 1 \text{ and } s \ x > 0 \end{aligned} \tag{**}$$

Stage 3: We prove the partial correctness property for the complete program:

$$\begin{aligned} &\text{if } \langle y := 1; \text{while } \neg(x=1) \text{ do } (y := y \star x; x := x - 1), s \rangle \rightarrow s' \\ &\text{then } s' \ y = (s \ x)! \text{ and } s \ x > 0 \end{aligned} \tag{***}$$

In each of the three stages the derivation tree of the given transition is inspected in order to prove the property.

First stage

Stage 1: We prove that the body of the `while` loop satisfies:

if $\langle y := y \star x; x := x - 1, s \rangle \rightarrow s''$ and $s'' \ x > 0$

then $(s \ y) \star (s \ x)! = (s'' \ y) \star (s'' \ x)!$ and $s \ x > 0$

(*)

In the *first stage* we consider the transition

$$\langle y := y \star x; x := x - 1, s \rangle \rightarrow s''$$

According to $[\text{comp}_{\text{ns}}]$ there will be transitions

$$\langle y := y \star x, s \rangle \rightarrow s' \text{ and } \langle x := x - 1, s' \rangle \rightarrow s''$$

for some s' . From the axiom $[\text{ass}_{\text{ns}}]$ we then get that $s' = s[y \mapsto \mathcal{A}[\![y \star x]\!]s]$ and that $s'' = s'[x \mapsto \mathcal{A}[\![x - 1]\!]s']$. Combining these results we have

$$s'' = s[y \mapsto (s \ y) \star (s \ x)][x \mapsto (s \ x) - 1]$$

Assuming that $s'' \ x > 0$ we can then calculate

$$(s'' \ y) \star (s'' \ x)! = ((s \ y) \star (s \ x)) \star ((s \ x) - 1)! = (s \ y) \star (s \ x)!$$

and since $s \ x = (s'' \ x) + 1$ this shows that (*) does indeed hold.

Second stage

Stage 2: We prove that the `while` loop satisfies:

if $\langle \text{while } \neg(x=1) \text{ do } (y := y \star x; x := x-1), s \rangle \rightarrow s''$
then $(s \ y) \star (s \ x)! = s'' \ y$ and $s'' \ x = \mathbf{1}$ and $s \ x > \mathbf{0}$ (**)

In the *second stage* we proceed by induction on the shape of the derivation tree for

$\langle \text{while } \neg(x=1) \text{ do } (y := y \star x; x := x-1), s \rangle \rightarrow s'$

$$\langle \text{while } \neg(\mathbf{x}=1) \text{ do } (\mathbf{y} := \mathbf{y} * \mathbf{x}; \mathbf{x} := \mathbf{x}-1), s \rangle \rightarrow s'$$

One of two axioms and rules could have been used to construct this derivation. If $[\text{while}_{\text{ns}}^{\text{ff}}]$ has been used then $s' = s$ and $\mathcal{B}[\neg(\mathbf{x}=1)]s = \mathbf{ff}$. This means that $s' \mathbf{x} = \mathbf{1}$ and since $\mathbf{1}! = \mathbf{1}$ we get the required $(s \mathbf{y}) * (s \mathbf{x})! = s \mathbf{y}$ and $s \mathbf{x} > \mathbf{0}$. This proves (**).

Next assume that $[\text{while}_{\text{ns}}^{\text{tt}}]$ is used to construct the derivation. Then it must be the case that $\mathcal{B}[\neg(\mathbf{x}=1)]s = \mathbf{tt}$ and

$$\langle \mathbf{y} := \mathbf{y} * \mathbf{x}; \mathbf{x} := \mathbf{x}-1, s \rangle \rightarrow s''$$

and

$$\langle \text{while } \neg(\mathbf{x}=1) \text{ do } (\mathbf{y} := \mathbf{y} * \mathbf{x}; \mathbf{x} := \mathbf{x}-1), s'' \rangle \rightarrow s'$$

for some state s'' . The induction hypothesis applied to the latter derivation gives that

$$(s'' \mathbf{y}) * (s'' \mathbf{x})! = s' \mathbf{y} \text{ and } s' \mathbf{x} = \mathbf{1} \text{ and } s'' \mathbf{x} > \mathbf{0}$$

From (*) we get that

$$(s \mathbf{y}) * (s \mathbf{x})! = (s'' \mathbf{y}) * (s'' \mathbf{x})! \text{ and } s \mathbf{x} > \mathbf{0}$$

Putting these results together we get

$$(s \mathbf{y}) * (s \mathbf{x})! = s' \mathbf{y} \text{ and } s' \mathbf{x} = \mathbf{1} \text{ and } s \mathbf{x} > \mathbf{0}$$

This proves (**) and thereby the second stage of the proof is completed.

Third stage

Stage 3: We prove the partial correctness property for the complete program:

$$\begin{array}{l} \text{if } \langle y := 1; \text{ while } \neg(x=1) \text{ do } (y := y \star x; x := x-1), s \rangle \rightarrow s' \\ \text{then } s' y = (s x)! \text{ and } s x > 0 \end{array} \quad (***)$$

Finally, consider the *third stage* of the proof and the derivation

$$\langle y := 1; \text{ while } \neg(x=1) \text{ do } (y := y \star x; x := x-1), s \rangle \rightarrow s'$$

According to [comp_{ns}] there will be a state s'' such that

$$\langle y := 1, s \rangle \rightarrow s''$$

and

$$\langle \text{while } \neg(x=1) \text{ do } (y := y \star x; x := x-1), s'' \rangle \rightarrow s'$$

From axiom [ass_{ns}] we see that $s'' = s[y \mapsto 1]$ and from (***) we get that $s'' x > 0$ and therefore $s x > 0$. Hence $(s x)! = (s'' y) \star (s'' x)!$ holds and using (***) we get

$$(s x)! = (s'' y) \star (s'' x)! = s' y$$

as required. This proves the partial correctness of the factorial statement.

How easy was that?

- Proof is very laborious
 - Need to connect all transitions and argues about relationships between their states
 - Reason: too closely connected to semantics of programming language
- Is the proof correct?
- How did we know to find this proof?
 - Is there a methodology?

Axiomatic verification approach

- What do we need in order to prove that the program does what it supposed to do?
- Specify the required behavior
- Compare the behavior with the one obtained by the operational semantics
- Develop a proof system for showing that the program satisfies a requirement
- Mechanically use the proof system to show correctness
- The meaning of a program is a set of verification rules

Axiomatic Verification: Spec

$S_{\text{fac}} \equiv \mathbf{y} := 1; \text{ while } \neg(\mathbf{x}=1) \text{ do } (\mathbf{y} := \mathbf{y} * \mathbf{x}; \mathbf{x} := \mathbf{x} - 1)$

- $\langle S_{\text{fac}}, s \rangle \rightarrow s'$ implies $s' \mathbf{y} = (s \mathbf{x})!$
- $\{x = N\} S_{\text{fac}} \{y = N!\}$
 - $\{\text{Pre-condition}(s)\} \text{Command}(S_{\text{fac}}) \{\text{post-state}(s')\}$
 - Not $\{\text{true}\} S_{\text{fac}} \{y = x!\}$

Partial vs. Total Correctness

$S_{\text{fac}} \equiv \mathbf{y} := 1; \text{ while } \neg(\mathbf{x}=1) \text{ do } (\mathbf{y} := \mathbf{y} * \mathbf{x}; \mathbf{x} := \mathbf{x} - 1)$

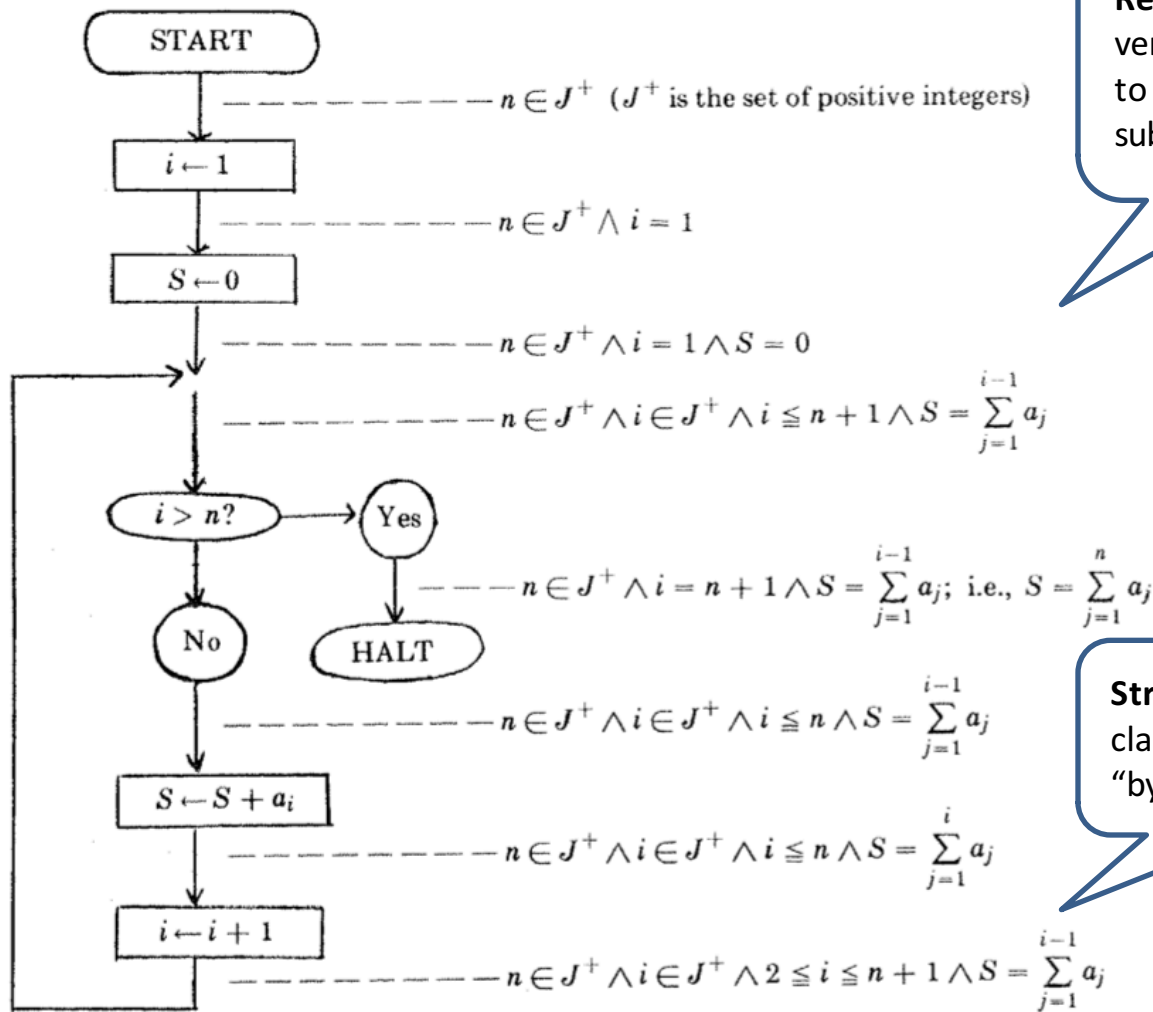
- $\langle S_{\text{fac}}, s \rangle \rightarrow s'$ implies $s' \mathbf{y} = (s \mathbf{x})!$
- $\{x = N\} S_{\text{fac}} \{y = N!\}$
 - {Pre-condition (s)} Command (S_{fac}) {post-state(s')}
 - **Not** $\{\text{true}\} S_{\text{fac}} \{y = x!\}$
- $[x = N] S_{\text{fac}} [y = N!]$

Hoare Triples

Verification: Assertion-Based [Floyd, '67]

- **Assertion:** invariant at specific program point
 - E.g., `assert(e)`
- use assertions as foundation for **static correctness proofs**
- specify assertions at *every* program point
- correctness reduced to **reasoning about individual statements**

Annotated Flow Programs



Reduction: Program verification is reduced to claims about the subject of discourse

Straight line code: claims are determined "by construction"

FIGURE 1. Flowchart of program to compute $S = \sum_{j=1}^n a_j$ ($n \geq 0$)

Annotated Flow Programs

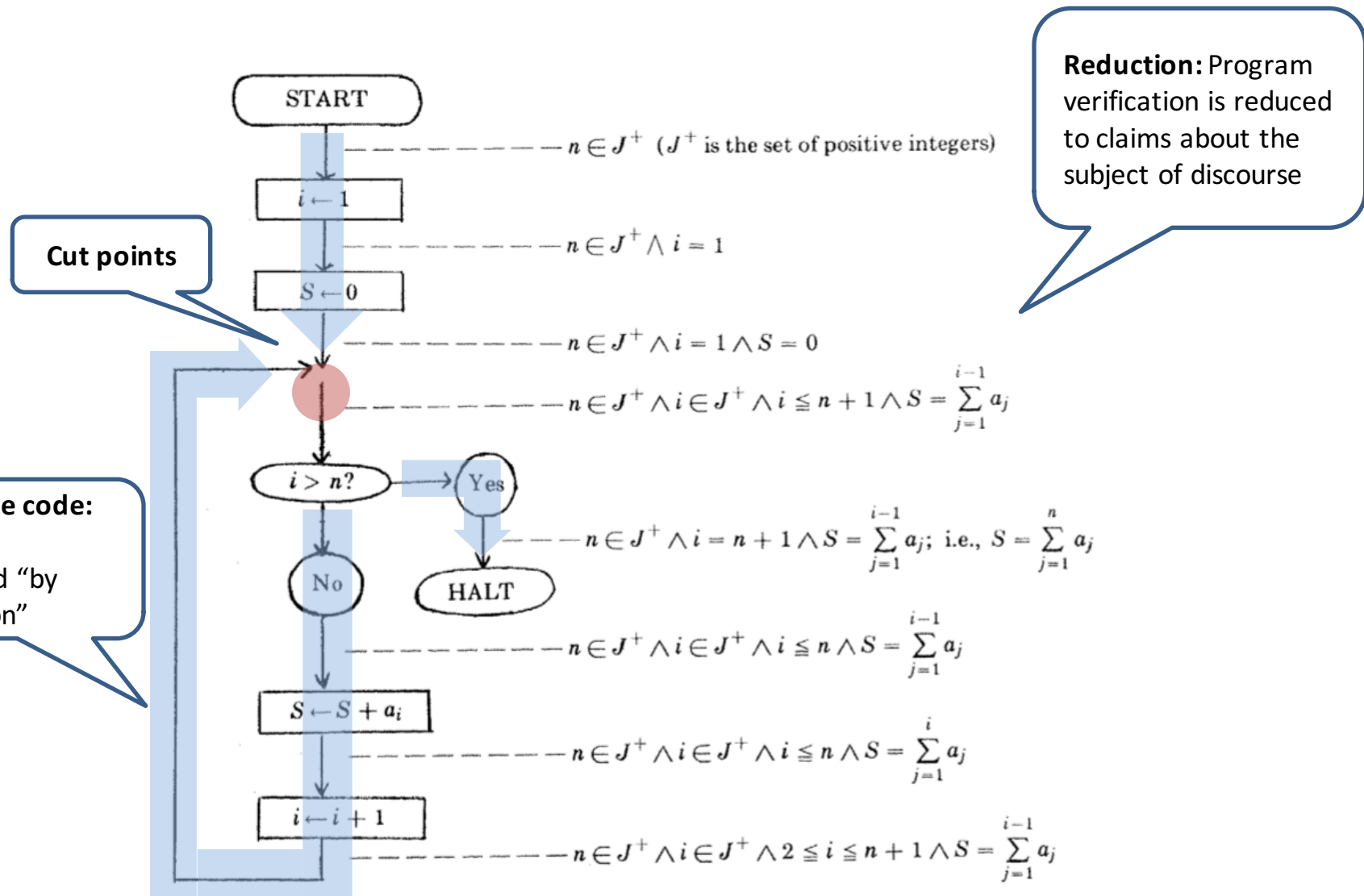


FIGURE 1. Flowchart of program to compute $S = \sum_{j=1}^n a_j$ ($n \geq 0$)

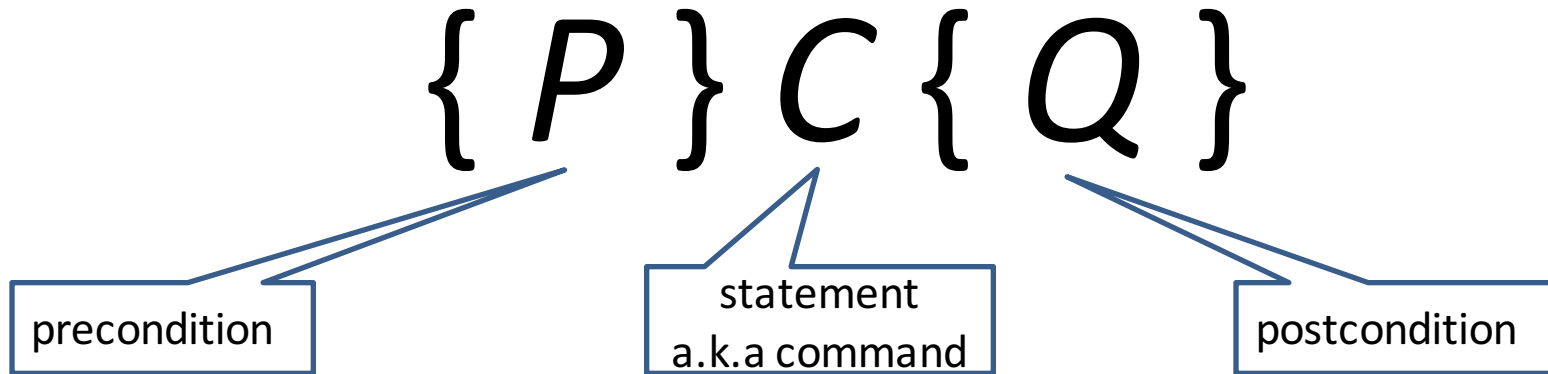
Assertion-Based Verification [Floyd, '67]

- **Assertion:** invariant at specific program point
 - E.g., `assert(e)`
- Proof reduced to logical claims
 - Considering the effect of statements
 - But, not reusable
- Challenge: Finding invariants at cut points in **loops**

Floyd-Hoare Logic 1969

- Use Floyd's ideas to define **axiomatic semantics**
 - Structured programming
 - No gotos
 - Modular (reusable claims)
 - Hoare triples
 - $\{P\} C \{Q\}$
 - $[P] C [Q]$ (often $\langle P \rangle C \langle Q \rangle$)
 - Define the programming language semantics as a **proof system**

Assertions, a.k.a Hoare triples



- P and Q are state predicates
 - Example: $x > 0$
- **If** P holds in the initial state, and **if** execution of C terminates on that state, **then** Q will hold in the state in which C halts
- C is not required to always terminate
 - `{true} while true do skip {false}`

Total correctness assertions

$$[P] C [Q]$$

- *If* P holds in the initial state, execution of C ***must terminate*** on that state, ***and*** Q will hold in the state in which C halts

Factorial example

$\{ ? \}$
`y := 1; while $\neg(x=1)$ do (y := y*x; x := x-1)`
 $\{ ? \}$

First attempt

We need a way to
“remember” value of
x before execution

$\{ \mathbf{x} > 0 \}$

$y := 1; \text{ while } \neg(x=1) \text{ do } (y := y * x; x := x - 1)$

$\{ \mathbf{y} = \mathbf{x}! \}$

Holds only for value of x at
state after execution finishes

Fixed assertion

A **logical** variable, must not appear in statement - immutable

$\{ \mathbf{x} = n \}$

$y := 1; \text{ while } \neg(x=1) \text{ do } (y := y * x; x := x - 1)$

$\{ \mathbf{y} = n! \wedge n > 0 \}$

The proof outline

```
{ x=n }  
y := 1;  
{ x>0  $\Rightarrow$  y*x!=n!  $\wedge$  n $\geq$ x }  
while  $\neg$ (x=1) do  
    { x-1>0  $\Rightarrow$  (y*x) *(x-1) !=n!  $\wedge$  n $\geq$ (x-1) }  
    y := y*x;  
    { x-1>0  $\Rightarrow$  y*(x-1) !=n!  $\wedge$  n $\geq$ (x-1) }  
    x := x-1  
{ y*x!=n!  $\wedge$  n>0  $\wedge$  x=1 }
```

Factorial example

$S_{\text{fac}} \equiv \mathbf{y} := 1; \text{ while } \neg(\mathbf{x}=1) \text{ do } (\mathbf{y} := \mathbf{y} * \mathbf{x}; \mathbf{x} := \mathbf{x} - 1)$

- Factorial partial correctness property = *if* the statement terminates *then* the final value of \mathbf{y} will be the factorial of the initial value of \mathbf{x}
 - What if $s \mathbf{x} < 0$?
- Formally, using natural semantics:
 $\langle S_{\text{fac}}, s \rangle \rightarrow s'$ implies $s' \mathbf{y} = (s \mathbf{x})!$

Staged proof

The proof proceeds in three stages:

Stage 1: We prove that the body of the `while` loop satisfies:

$$\begin{aligned} &\text{if } \langle y := y \star x; x := x - 1, s \rangle \rightarrow s'' \text{ and } s'' \ x > 0 \\ &\text{then } (s \ y) \star (s \ x)! = (s'' \ y) \star (s'' \ x)! \text{ and } s \ x > 0 \end{aligned} \tag{*}$$

Stage 2: We prove that the `while` loop satisfies:

$$\begin{aligned} &\text{if } \langle \text{while } \neg(x=1) \text{ do } (y := y \star x; x := x - 1), s \rangle \rightarrow s'' \\ &\text{then } (s \ y) \star (s \ x)! = s'' \ y \text{ and } s'' \ x = 1 \text{ and } s \ x > 0 \end{aligned} \tag{**}$$

Stage 3: We prove the partial correctness property for the complete program:

$$\begin{aligned} &\text{if } \langle y := 1; \text{while } \neg(x=1) \text{ do } (y := y \star x; x := x - 1), s \rangle \rightarrow s' \\ &\text{then } s' \ y = (s \ x)! \text{ and } s \ x > 0 \end{aligned} \tag{***}$$

In each of the three stages the derivation tree of the given transition is inspected in order to prove the property.

Stages

$$s \cdot y \cdot (s \cdot x)! = s''' \cdot y \cdot (s''' \cdot x)! \wedge s \cdot x > 0$$

S y := y*x; x := x-1 **S'''**

$$s \cdot y \cdot (s \cdot x)! = s'' \cdot y \cdot (s'' \cdot x)! \wedge s'' \cdot x = 1 \wedge s \cdot x > 0$$

S while ¬(x=1) do (y := y*x; x := x-1) **S''**

$$s' \cdot y = (s \cdot x)! \wedge s \cdot x > 0$$

S y := 1; while ¬(x=1) do (y := y*x; x := x-1) **S'**

Inductive proof over iterations

$$s \cdot y \cdot (s \cdot x)! = s''' \cdot y \cdot (s''' \cdot x)! \wedge s \cdot x > 0$$

S `(y := y*x; x := x-1)` S'''

S''' `while $\neg(x=1)$ do (y := y*x; x := x-1)` S''

$$s''' \cdot y \cdot (s''' \cdot x)! = s'' \cdot y \cdot (s'' \cdot x)! \wedge s'' \cdot x = 1 \wedge s''' \cdot x > 0$$

S `while $\neg(x=1)$ do (y := y*x; x := x-1)` S''

$$s \cdot y \cdot (s \cdot x)! = s'' \cdot y \cdot (s'' \cdot x)! \wedge s'' \cdot x = 1 \wedge s \cdot x > 0$$

Assertions, a.k.a Hoare triples

$$\{ P \} C \{ Q \}$$

- P and Q are state predicates
 - Example: $x > 0$
- ***If*** P holds in the initial state, and ***if*** execution of C terminates on that state, ***then*** Q will hold in the state in which C halts
- C is not required to always terminate

```
{true} while true do skip {false}
```

Total correctness assertions

$$[P] C [Q]$$

- *If* P holds in the initial state, execution of C ***must terminate*** on that state, ***and*** Q will hold in the state in which C halts

Factorial assertion

A **logical** variable, must not appear in statement - immutable

$\{ \mathbf{x} = n \}$

$y := 1; \text{ while } \neg(x=1) \text{ do } (y := y * x; x := x - 1)$

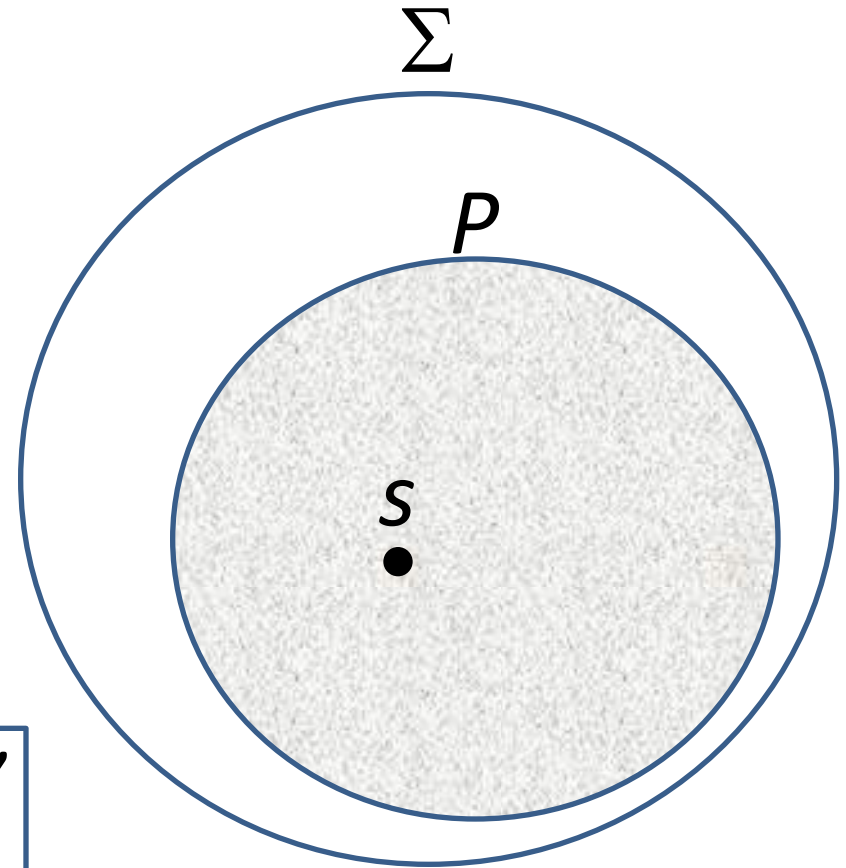
$\{ \mathbf{y} = n! \wedge n > 0 \}$

Factorial partial correctness proof

```
{ x=n }  
y := 1;  
{ x>0  $\Rightarrow$  y*x!=n!  $\wedge$  n $\geq$ x }  
while  $\neg$ (x=1) do  
    { x-1>0  $\Rightarrow$  (y*x) *(x-1) !=n!  $\wedge$  n $\geq$ (x-1) }  
    y := y*x;  
    { x-1>0  $\Rightarrow$  y*(x-1) !=n!  $\wedge$  n $\geq$ (x-1) }  
    x := x-1  
{ y*x!=n!  $\wedge$  n>0  $\wedge$  x=1 }
```

Formalizing partial correctness

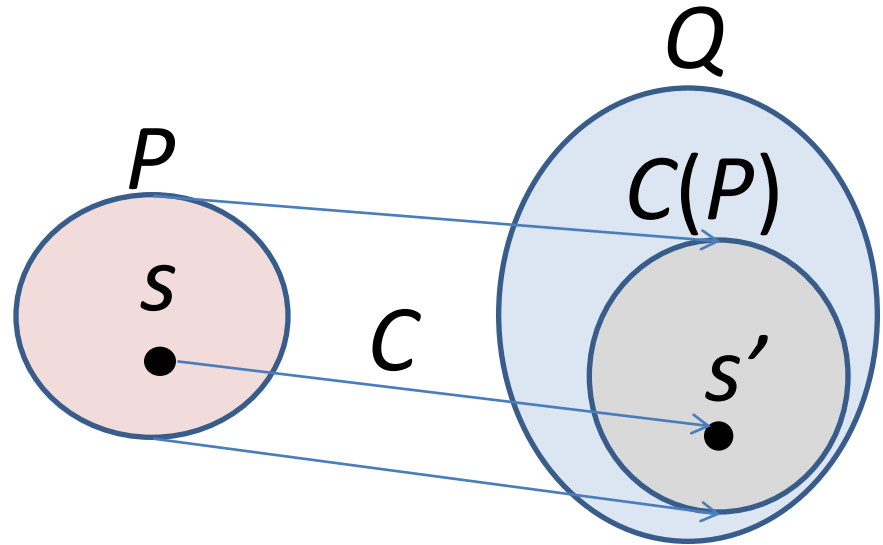
- $s \models P$
 - P holds in state s
- Σ – program states
 \perp – undefined



$$S_{\text{ns}} \llbracket C \rrbracket s = \begin{cases} s' & \text{if } \langle C, s \rangle \rightarrow s' \\ \perp & \text{else} \end{cases}$$

Formalizing partial correctness

- $s \models P$
 - P holds in state s
- Σ – program states
- \perp – undefined



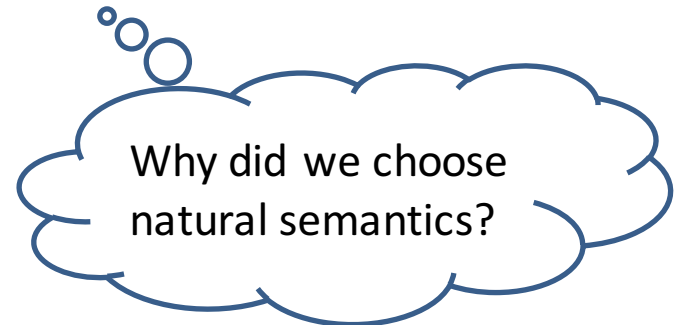
- $\{P\} C \{Q\}$

– $\forall s, s' \in \Sigma . (s \models P \wedge \langle C, s \rangle \rightarrow s') \Rightarrow s' \models Q$
 alternatively

– $\forall s \in \Sigma . (s \models P \wedge S_{ns} \llbracket C \rrbracket s \neq \perp) \Rightarrow S_{ns} \llbracket C \rrbracket s \models Q$

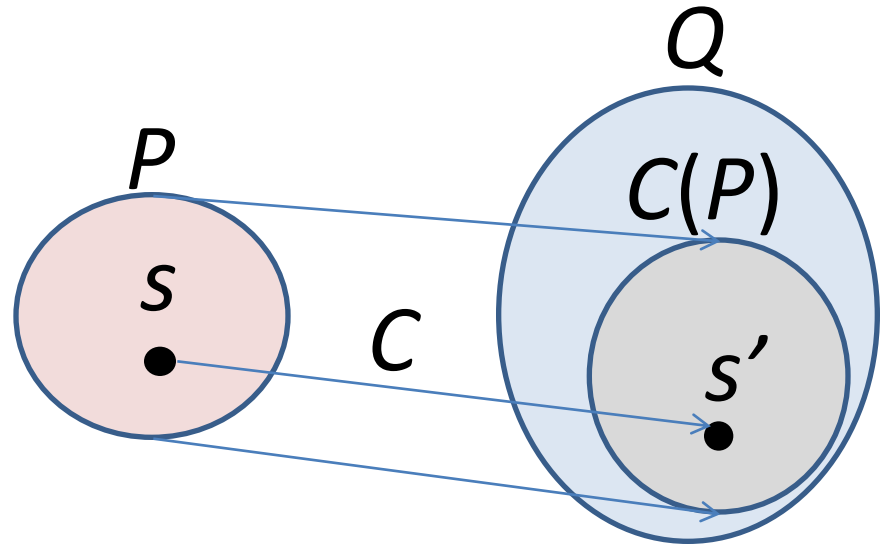
– Convention: $\perp \models P$ for all P

$\forall s \in \Sigma . s \models P \Rightarrow S_{ns} \llbracket C \rrbracket s \models Q$



Formalizing partial correctness

- $s \models P$
 - P holds in state s
- Σ – program states
- \perp – undefined



- $\{P\} C \{Q\}$

– $\forall s, s' \in \Sigma . (s \models P \wedge \langle C, s \rangle \Rightarrow^* s') \Rightarrow s' \models Q$
 alternatively

– $\forall s \in \Sigma . (s \models P \wedge S_{\text{sos}}[[C]] s \neq \perp) \Rightarrow S_{\text{sos}}[[C]] \models Q$

– Convention: $\perp \models P$ for all P

$\forall s \in \Sigma . s \models P \Rightarrow S_{\text{sos}}[[C]] s \models Q$

How do we express predicates?

- Extensional approach
 - Abstract mathematical functions
 - $P : \mathbf{State} \rightarrow \mathbf{T}$
- Intensional approach
 - Via language of formulae

An assertion language

- **Bexp** is not expressive enough to express predicates needed for many proofs
 - Extend **Bexp**
- Allow quantifications
 - $\forall z. \dots$
 - $\exists z. \dots$
 - $\exists z. z = k \times n$
- Import well known mathematical concepts
 - $n! \equiv n \times (n-1) \times \dots \times 2 \times 1$

An assertion language

Either a program variables or a logical variable

$a ::= n \mid x \mid a_1 + a_2 \mid a_1 \star a_2 \mid a_1 - a_2$

$A ::= \mathbf{true} \mid \mathbf{false}$

$\mid a_1 = a_2 \mid a_1 \leq a_2 \mid \neg A \mid A_1 \wedge A_2 \mid A_1 \vee A_2$

$\mid A_1 \Rightarrow A_2 \mid \forall z. A \mid \exists z. A$

First Order Logic Reminder

Free/bound variables

- A variable is said to be **bound** in a formula when it occurs in the scope of a quantifier. Otherwise it is said to be **free**
 - $\exists i. k=i \times m$
 - $(i+100 \leq 77) \wedge \forall i. j+1=i+3$
- $FV(A) \equiv$ the free variables of A
- Defined inductively on the abstract syntax tree of A

Free variables

$$\text{FV}(n) \equiv \{\}$$

$$\text{FV}(x) \equiv \{x\}$$

$$\text{FV}(a_1 + a_2) \equiv \text{FV}(a_1 \star a_2) \equiv \text{FV}(a_1 - a_2) \equiv \text{FV}(a_1) \cup \text{FV}(a_2)$$

$$\text{FV}(\mathbf{true}) \equiv \text{FV}(\mathbf{false}) \equiv \{\}$$

$$\text{FV}(a_1 = a_2) \equiv \text{FV}(a_1 \leq a_2) \equiv \text{FV}(a_1) \cup \text{FV}(a_2)$$

$$\text{FV}(\neg A) \equiv \text{FV}(A)$$

$$\text{FV}(A_1 \wedge A_2) \equiv \text{FV}(A_1 \vee A_2) \equiv \text{FV}(A_1 \Rightarrow A_2)$$

$$\text{FV}(\forall z. A) \equiv \text{FV}(\exists z. A) \equiv \text{FV}(A) \setminus \{z\}$$

Substitution

What if t is not pure?

- An expression t is **pure** (a **term**) if it does not contain quantifiers
- $A[t/z]$ denotes the assertion A' which is the same as A , except that all instances of the free variable z are replaced by t
- $A \equiv \exists i. k=i \times m$
 $A[5/k] =$
 $A[5/i] =$

Calculating substitutions

$$n[t/z] = n$$

$$x[t/z] = x$$

$$x[t/x] = t$$

$$(a_1 + a_2)[t/z] = a_1[t/z] + a_2[t/z]$$

$$(a_1 \star a_2)[t/z] = a_1[t/z] \star a_2[t/z]$$

$$(a_1 - a_2)[t/z] = a_1[t/z] - a_2[t/z]$$

Calculating substitutions

$$\mathbf{true}[t/x] = \mathbf{true}$$

$$\mathbf{false}[t/x] = \mathbf{false}$$

$$(a_1 = a_2)[t/z] = a_1[t/z] = a_2[t/z]$$

$$(a_1 \leq a_2)[t/z] = a_1[t/z] \leq a_2[t/z]$$

$$(\neg A)[t/z] = \neg(A[t/z])$$

$$(A_1 \wedge A_2)[t/z] = A_1[t/z] \wedge A_2[t/z]$$

$$(A_1 \vee A_2)[t/z] = A_1[t/z] \vee A_2[t/z]$$

$$(A_1 \Rightarrow A_2)[t/z] = A_1[t/z] \Rightarrow A_2[t/z]$$

$$(\forall z. A)[t/z] = \forall z. A$$

$$(\forall z. A)[t/y] = \forall z. A[t/y]$$

$$(\exists z. A)[t/z] = \exists z. A$$

$$(\exists z. A)[t/y] = \exists z. A[t/y]$$

Proof Rules

Axiomatic semantics for **While**

$$[\text{ass}_p] \quad \{ P[a/x] \} x := a \{ P \}$$

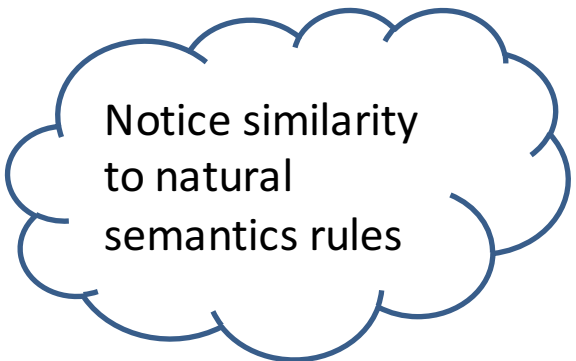
$$[\text{skip}_p] \quad \{ P \} \text{skip} \{ P \}$$

$$[\text{comp}_p] \quad \frac{\{ P \} S_1 \{ Q \}, \{ Q \} S_2 \{ R \}}{\{ P \} S_1; S_2 \{ R \}}$$

$$[\text{if}_p] \quad \frac{\{ b \wedge P \} S_1 \{ Q \}, \{ \neg b \wedge P \} S_2 \{ Q \}}{\{ P \} \text{if } b \text{ then } S_1 \text{ else } S_2 \{ Q \}}$$

$$[\text{while}_p] \quad \frac{\{ b \wedge P \} S \{ P \}}{\{ P \} \text{while } b \text{ do } S \{ \neg b \wedge P \}}$$

$$[\text{cons}_p] \quad \frac{\{ P' \} S \{ Q' \}}{\{ P \} S \{ Q \}} \quad \text{if } P \Rightarrow P' \text{ and } Q' \Rightarrow Q$$



Notice similarity
to natural
semantics rules

Assignment rule

$$[ass_p] \quad \{ P[a/x] \} x := a \{ P \}$$

- A “backwards” rule
- $x := a$ always finishes
- Why is this true?
 - Recall operational semantics:

$$[ass_{ns}] \quad \langle x := a, s \rangle \rightarrow s[x \mapsto \mathcal{A}[[a]]s]$$

- Example: $\{ y * z < 9 \} \quad x := y * z \quad \{ x < 9 \}$
What about $\{ y * z < 9 \wedge w = 5 \} \quad x := y * z \quad \{ w = 5 \}$?

$$s[x \mapsto \mathcal{A}[[a]]s] \models P$$

skip rule

$[skip_p] \{P\} skip \{P\}$

$[skip_{ns}] \langle skip, s \rangle \rightarrow s$

Composition rule

$$[\text{comp}_p] \frac{\{P\} S_1 \{Q\}, \quad \{Q\} S_2 \{R\}}{\{P\} S_1; S_2 \{R\}}$$

$$[\text{comp}_{ns}] \frac{\langle S_1, s \rangle \rightarrow s', \quad \langle S_2, s' \rangle \rightarrow s''}{\langle S_1; S_2, s \rangle \rightarrow s''}$$

- Holds when S_1 terminates in every state where P holds and then Q holds and S_2 terminates in every state where Q holds and then R holds

Condition rule

$$[\text{if}_p] \frac{\{b \wedge P\} S_1 \{Q\}, \{\neg b \wedge P\} S_2 \{Q\}}{\{P\} \text{if } b \text{ then } S_1 \text{ else } S_2 \{Q\}}$$

$$[\text{if}_{ns}^{\text{tt}}] \frac{\langle S_1, s \rangle \rightarrow s'}{\langle \text{if } b \text{ then } S_1 \text{ else } S_2, s \rangle \rightarrow s'} \quad \text{if } \mathcal{B} \llbracket b \rrbracket s = \text{tt}$$

$$[\text{if}_{ns}^{\text{ff}}] \frac{\langle S_2, s \rangle \rightarrow s'}{\langle \text{if } b \text{ then } S_1 \text{ else } S_2, s \rangle \rightarrow s'} \quad \text{if } \mathcal{B} \llbracket b \rrbracket s = \text{ff}$$

Loop rule

$$[\text{while}_p] \frac{\{b \wedge P\} S \{P\}}{\{P\} \text{while } b \text{ do } S \{\neg b \wedge P\}}$$

$$[\text{while}_{ns}^{\text{ff}}] \quad \langle \text{while } b \text{ do } S, s \rangle \rightarrow s \quad \text{if } \mathcal{B} \llbracket b \rrbracket s = \mathbf{ff}$$

$$[\text{while}_{ns}^{\text{tt}}] \quad \frac{\langle S, s \rangle \rightarrow s', \langle \text{while } b \text{ do } S, s' \rangle \rightarrow s''}{\langle \text{while } b \text{ do } S, s \rangle \rightarrow s''} \quad \text{if } \mathcal{B} \llbracket b \rrbracket s = \mathbf{tt}$$

- Here P is called an **invariant** for the loop
 - Holds before and after each loop iteration
 - Finding loop invariants – most challenging part of proofs
- When loop finishes, b is false

Rule of consequence

$$[\text{cons}_p] \frac{\{P'\} S \{Q'\}}{\{P\} S \{Q\}} \quad \text{if } P \Rightarrow P' \text{ and } Q' \Rightarrow Q$$

- Allows strengthening the precondition and weakening the postcondition
- The only rule that is not sensitive to the form of the statement

Rule of consequence

$$[\text{cons}_p] \frac{\{P'\} S \{Q'\}}{\{P\} S \{Q\}} \quad \text{if } P \Rightarrow P' \text{ and } Q' \Rightarrow Q$$

- Why do we need it?
- Allows the following

$$\frac{\{y * z < 9\} \quad x := y * z \quad \{x < 9\}}{\{y * z < 9 \wedge w = 5\} \quad x := y * z \quad \{x < 10\}}$$

Axiomatic semantics for **While**

Axiom for every primitive statement

$$[\text{ass}_p] \quad \{ P[a/x] \} x := a \{ P \}$$

$$[\text{skip}_p] \quad \{ P \} \text{skip} \{ P \}$$

$$[\text{comp}_p] \quad \frac{\{ P \} S_1 \{ Q \}, \quad \{ Q \} S_2 \{ R \}}{\{ P \} S_1; S_2 \{ R \}}$$

$$[\text{if}_p] \quad \frac{\{ b \wedge P \} S_1 \{ Q \}, \quad \{ \neg b \wedge P \} S_2 \{ Q \}}{\{ P \} \text{if } b \text{ then } S_1 \text{ else } S_2 \{ Q \}}$$

Inference rule for every composed statement

$$[\text{while}_p] \quad \frac{\{ b \wedge P \} S \{ P \}}{\{ P \} \text{while } b \text{ do } S \{ \neg b \wedge P \}}$$

$$[\text{cons}_p] \quad \frac{\{ P' \} S \{ Q' \}}{\{ P \} S \{ Q \}} \quad \text{if } P \Rightarrow P' \text{ and } Q' \Rightarrow Q$$

Inference trees

- Similar to derivation trees of natural semantics
- Leaves are ...
- Internal nodes correspond to ...
- Inference tree is called
 - **Simple** if tree is only an axiom
 - **Composite** otherwise
- Similar to derivation trees of natural semantics
 - Reasoning about immediate constituent

Factorial proof

Goal: $\{x=n\} y:=1; \text{ while } (x \neq 1) \text{ do } (y:=y*x; x:=x-1) \{y=n! \wedge n>0\}$

$W = \text{ while } (x \neq 1) \text{ do } (y:=y*x; x:=x-1)$

$\text{INV} = x > 0 \Rightarrow (y \cdot x! = n! \wedge n \geq x)$

[comp] $\frac{\{ \text{INV}[x-1/x][y*x/y] \} y:=y*x \{ \text{INV}[x-1/x] \} \quad \{ \text{INV}[x-1/x] \} x:=x-1 \{ \text{INV} \}}{\{ \text{INV}[x-1/x][y*x/y] \} y:=y*x; x:=x-1 \{ \text{INV} \}}$

[cons] $\frac{\{ \mathbf{x \neq 1} \wedge \text{INV} \} y:=y*x; x:=x-1 \{ \text{INV} \}}{\{ \text{INV} \} W \{ \mathbf{x=1} \wedge \text{INV} \}}$

[cons] $\frac{\{ \text{INV}[1/y] \} y:=1 \{ \text{INV} \}}{\{ x=n \} y:=1 \{ \text{INV} \}}$

[cons] $\frac{\{ \text{INV} \} W \{ \mathbf{x=1} \wedge \text{INV} \}}{\{ \text{INV} \} W \{ \mathbf{y=n!} \wedge n>0 \}}$

[comp] $\frac{\{ x=n \} y:=1 \{ \text{INV} \} \quad \{ \text{INV} \} W \{ \mathbf{y=n!} \wedge n>0 \}}{\{ x=n \} \text{ while } (x \neq 1) \text{ do } (y:=y*x; x:=x-1) \{ \mathbf{y=n!} \wedge n>0 \}}$

Factorial proof

Goal: $\{x=n\} y:=1; \text{ while } (x \neq 1) \text{ do } (y:=y*x; x:=x-1) \{y=n! \wedge n>0\}$

$W = \text{ while } (x \neq 1) \text{ do } (y:=y*x; x:=x-1)$

$\text{INV} = x > 0 \Rightarrow (y \cdot x! = n! \wedge n \geq x)$

[comp] $\frac{\{ \text{INV}[x-1/x][y*x/y] \} y:=y*x \{ \text{INV}[x-1/x] \} \quad \{ \text{INV}[x-1/x] \} x:=x-1 \{ \text{INV} \}}{\{ \text{INV}[x-1/x][y*x/y] \} y:=y*x; x:=x-1 \{ \text{INV} \}}$

[cons]

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

[while] $\frac{\{ x \neq 1 \wedge \text{INV} \} y:=y*x; x:=x-1 \{ \text{INV} \}}{\{ \text{INV} \} W \{ x=1 \wedge \text{INV} \}}$

[cons] $\frac{\{ \text{INV}[1/y] \} y:=1 \{ \text{INV} \}}{\{ x=n \} y:=1 \{ \text{INV} \}}$

[cons] $\frac{\{ \text{INV} \} W \{ x=1 \wedge \text{INV} \}}{\{ \text{INV} \} W \{ y=n! \wedge n>0 \}}$

[comp]

$\{ x=n \} \text{ while } (x \neq 1) \text{ do } (y:=y*x; x:=x-1) \{ y=n! \wedge n>0 \}$

Factorial proof

Goal: $\{x=n\} \mathbf{y}:=1; \mathbf{while} (x \neq 1) \mathbf{do} (\mathbf{y}:=\mathbf{y} * \mathbf{x}; \mathbf{x}:=\mathbf{x}-1) \{ \mathbf{y}=n! \wedge n > 0 \}$

$W = \mathbf{while} (x \neq 1) \mathbf{do} (\mathbf{y}:=\mathbf{y} * \mathbf{x}; \mathbf{x}:=\mathbf{x}-1)$

$INV = x > 0 \Rightarrow (y \cdot x! = n! \wedge n \geq x)$

[comp] $\frac{\{ INV[x-1/x][y*x/y] \} \mathbf{y}:=\mathbf{y} * \mathbf{x} \{ INV[x-1/x] \} \quad \{ INV[x-1/x] \} \mathbf{x}:=\mathbf{x}-1 \{ INV \}}{\{ INV[x-1/x][y*x/y] \} \mathbf{y}:=\mathbf{y} * \mathbf{x}; \mathbf{x}:=\mathbf{x}-1 \{ INV \}}$

[cons]

$\frac{\{ P' \} S \{ Q' \}}{\{ P \} S \{ Q \}}$ if $P \Rightarrow P'$ and $Q' \Rightarrow Q$

[cons] $\frac{\{ INV[1/y] \} \mathbf{y}:=1 \{ INV \}}{\{ x=n \} \mathbf{y}:=1 \{ INV \}}$

[while] $\frac{\{ \mathbf{x} \neq 1 \wedge INV \} \mathbf{y}:=\mathbf{y} * \mathbf{x}; \mathbf{x}:=\mathbf{x}-1 \{ INV \}}{\{ INV \} W \{ \mathbf{x}=1 \wedge INV \}}$

[cons] $\frac{\{ INV \} W \{ \mathbf{x}=1 \wedge INV \}}{\{ INV \} W \{ \mathbf{y}=n! \wedge n > 0 \}}$

[comp]

$\{ x=n \} \mathbf{while} (x \neq 1) \mathbf{do} (\mathbf{y}:=\mathbf{y} * \mathbf{x}; \mathbf{x}:=\mathbf{x}-1) \{ \mathbf{y}=n! \wedge n > 0 \}$

Provability

- We say that an assertion $\{ P \} C \{ Q \}$ is **provable** if there exists an inference tree
 - Written as $\vdash_p \{ P \} C \{ Q \}$

Annotated programs

- A streamlined version of inference trees
 - Inline inference trees into programs
 - A kind of “[proof carrying code](#)”
 - Going from annotated program to proof tree is a linear time translation

Annotating composition

- We can inline inference trees into programs
- Using proof equivalence of $S_1; (S_2; S_3)$ and $(S_1; S_2); S_3$ instead writing deep trees, e.g.,

$$\frac{\frac{\{P\} S_1 \{P'\} \quad \{P'\} S_2 \{P''\}}{\{P\} (S_1; S_2) \{P''\}} \quad \frac{\{P''\} S_3 \{P'''\} \quad \{P'''\} S_4 \{P''\}}{\{P''\} (S_3; S_4) \{Q\}}}{\{P\} (S_1; S_2); (S_3; S_4) \{Q\}}$$

- We can annotate a composition $S_1; S_2; \dots; S_n$ by

$$\{P_1\} S_1 \{P_2\} S_2 \dots \{P_{n-1}\} S_{n-1} \{P_n\}$$

Annotating conditions

[if_p]

$$\frac{\{b \wedge P\} S_1 \{Q\}, \{\neg b \wedge P\} S_2 \{Q\}}{\{P\} \text{if } b \text{ then } S_1 \text{ else } S_2 \{Q\}}$$

{P}

if *b* then

 {*b* ∧ *P*}

*S*₁

else

*S*₂

{Q}

Annotating conditions

[if_p]

$$\frac{\{b \wedge P\} S_1 \{Q\}, \{\neg b \wedge P\} S_2 \{Q\}}{\{P\} \text{if } b \text{ then } S_1 \text{ else } S_2 \{Q\}}$$

```
{P}
if b then
    {b ∧ P}
    S1
    {Q1}
else
    S2
    {Q2}
{Q}
```

Usually Q is the result of using the consequence rule, so a more explicit annotation is

Annotating loops

$$[\text{while}_p] \frac{\{b \wedge P\} S \{P\}}{\{P\} \text{while } b \text{ do } S \{\neg b \wedge P\}}$$

$\{P\}$
while b do
 $\{b \wedge P\}$
 S
 $\{\neg b \wedge P\}$

Annotating loops

$$[\text{while}_p] \frac{\{b \wedge P\} S \{P\}}{\{P\} \text{while } b \text{ do } S \{\neg b \wedge P\}}$$

$\{P\}$

while b do

$\{b \wedge P\}$

S

$\{P'\}$

~~$\{\neg b \wedge P\}$~~ $\{Q\}$

P' implies P

$\neg b \wedge P$ implies Q

Annotated factorial program

```
{ x=n }  
y := 1;  
{ x>0  $\Rightarrow$  y*x!=n!  $\wedge$  n $\geq$ x }  
while  $\neg$ (x=1) do  
    { x-1>0  $\Rightarrow$  (y*x) *(x-1) !=n!  $\wedge$  n $\geq$ (x-1) }  
    y := y*x;  
    { x-1>0  $\Rightarrow$  y*(x-1) !=n!  $\wedge$  n $\geq$ (x-1) }  
    x := x-1  
{ y*x!=n!  $\wedge$  n>0 }
```

- Contrast with proof via natural semantics
- Where did the inductive argument over loop iterations go?

Properties of the semantics

Equivalence

- What is the analog of program equivalence in axiomatic verification?

Soundness

- Can we prove incorrect properties?

Completeness

- Is there something we can't prove?

Provability

- We say that an assertion $\{ P \} C \{ Q \}$ is **provable** if there exists an inference tree
 - Written as $\vdash_p \{ P \} C \{ Q \}$
 - Are inference trees unique?
 $\{ \text{true} \} x:=1; x:=x+5 \{ x \geq 0 \}$
- Proofs of properties of axiomatic semantics use *induction on the shape of the inference tree*
 - Example: prove $\vdash_p \{ P \} C \{ \mathbf{true} \}$ for any P and C

Provable equivalence

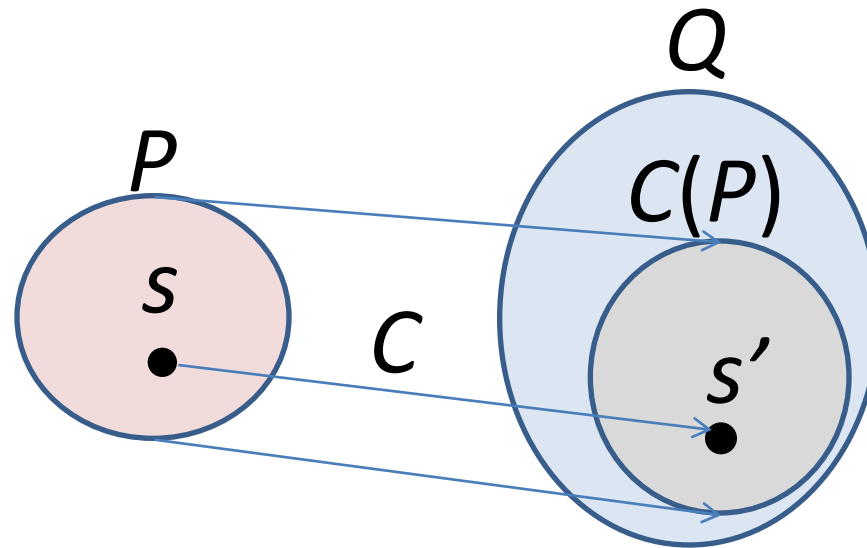
- We say that C_1 and C_2 are **provably equivalent** if for all P and Q

$\vdash_p \{ P \} C_1 \{ Q \}$ if and only if $\vdash_p \{ P \} C_2 \{ Q \}$

- Examples:
 - S ; **skip** and S
 - S_1 ; $(S_2$; $S_3)$ and $(S_1$; $S_2)$; S_3

Valid assertions

- We say that $\{ P \} C \{ Q \}$ is **valid** if for all states s , if $s \models P$ and $\langle C, s \rangle \rightarrow s'$ then $s' \models Q$
- Denoted by $\models_p \{ P \} C \{ Q \}$



Logical implication and equivalence

- We write $A \Rightarrow B$ if for all states s
if $s \models A$ then $s \models B$
 - $\{s \mid s \models A\} \subseteq \{s \mid s \models B\}$
 - For every predicate A : $false \Rightarrow A \Rightarrow true$
- We write $A \Leftrightarrow B$ if $A \Rightarrow B$ and $B \Rightarrow A$
 - $false \Leftrightarrow 5=7$
- In writing Hoare-style proofs, we will often replace a predicate A with A' such that $A \Leftrightarrow A'$ and A' is “simpler”

Soundness and completeness

- The inference system is **sound**:
 - $\vdash_p \{ P \} C \{ Q \}$ implies $\models_p \{ P \} C \{ Q \}$

- The inference system is **complete**:
 - $\models_p \{ P \} C \{ Q \}$ implies $\vdash_p \{ P \} C \{ Q \}$

Hoare logic is sound and (relatively) complete

- **Soundness:**

$$\vdash_p \{P\} C \{Q\} \text{ implies } \models_p \{P\} C \{Q\}$$

- **(Relative) completeness:**

$$\models_p \{P\} C \{Q\} \text{ implies } \vdash_p \{P\} C \{Q\}$$

– Provided we can prove any implication $R \implies R'$

Hoare logic is sound and (relatively) complete

- **Soundness:**

$$\vdash_p \{P\} C \{Q\} \text{ implies } \models_p \{P\} C \{Q\}$$

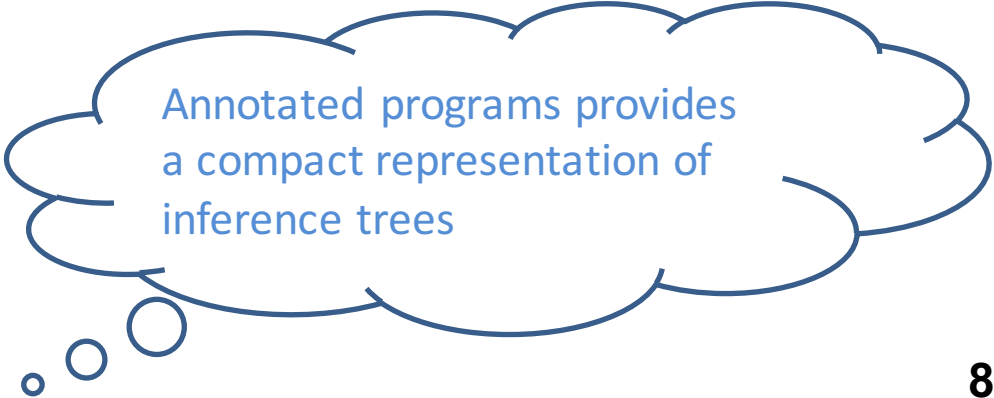
- **(Relative) completeness:**

$$\models_p \{P\} C \{Q\} \text{ implies } \vdash_p \{P\} C \{Q\}$$

- Provided we can prove any implication $R \implies R'$
 - FYI, nobody tells us how to find a proof...

Is there an Algorithm?

```
{ x=n }  
y := 1;  
{ x>0  $\Rightarrow$  y*x!=n!  $\wedge$  n $\geq$ x }  
while  $\neg$ (x=1) do  
  { x-1>0  $\Rightarrow$  (y*x) *(x-1) !=n!  $\wedge$  n $\geq$ (x-1) }  
  y := y*x;  
  { x-1>0  $\Rightarrow$  y*(x-1) !=n!  $\wedge$  n $\geq$ (x-1) }  
  x := x-1  
{ y*x!=n!  $\wedge$  n>0 }
```



Annotated programs provides
a compact representation of
inference trees

?

Predicate Transformers

Weakest liberal precondition

- A backward-going predicate transformer
- The **weakest liberal precondition** for Q is

$$s \models \text{wlp}(C, Q)$$

if and only if for all states s'

if $\langle C, s \rangle \rightarrow s'$ then $s' \models Q$

Propositions:

1. $\models_p \{ \text{wlp}(C, Q) \} C \{ Q \}$

2. If $\models_p \{ P \} C \{ Q \}$ then $P \Rightarrow \text{wlp}(C, Q)$

Strongest postcondition

- A forward-going predicate transformer
- The **strongest postcondition** for P is

$$s' \models \text{sp}(P, C)$$

if and only if there exists s such that

if $\langle C, s \rangle \rightarrow s'$ and $s \models P$

1. $\models_p \{ P \} C \{ \text{sp}(P, C) \}$
2. If $\models_p \{ P \} C \{ Q \}$ then $\text{sp}(P, C) \Rightarrow Q$

Predicate transformer semantics

- wlp and sp can be seen functions that transform predicates to other predicates
 - $\text{wlp}[[C]] : \text{Predicate} \rightarrow \text{Predicate}$
 $\{ P \} C \{ Q \}$ if and only if $\text{wlp}[[C]] Q = P$
 - $\text{sp}[[C]] : \text{Predicate} \rightarrow \text{Predicate}$
 $\{ P \} C \{ Q \}$ if and only if $\text{sp}[[C]] P = Q$