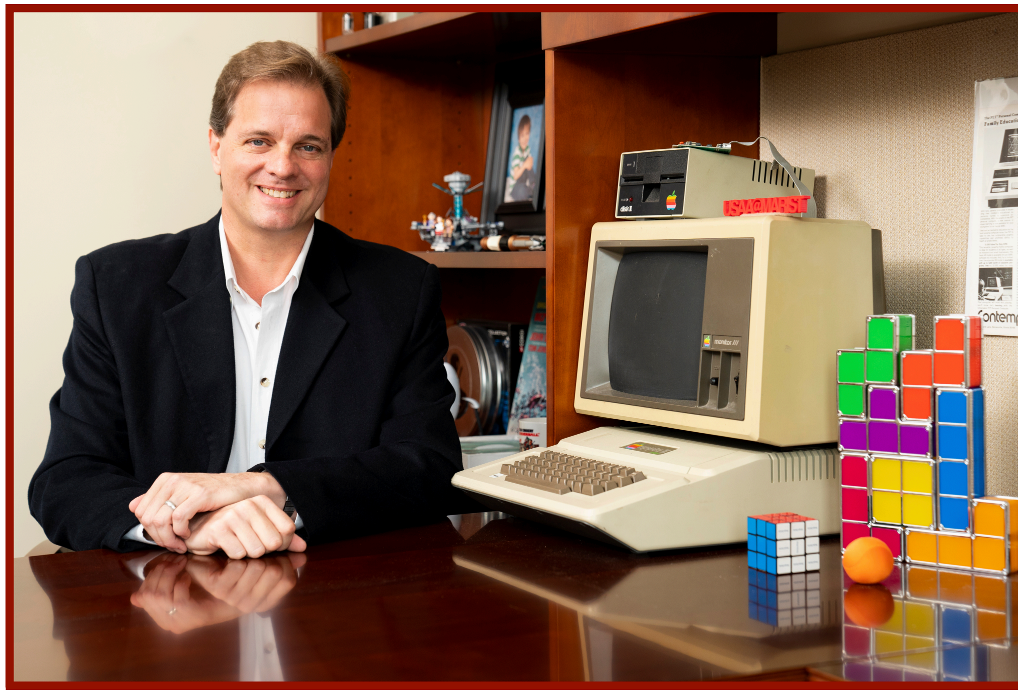
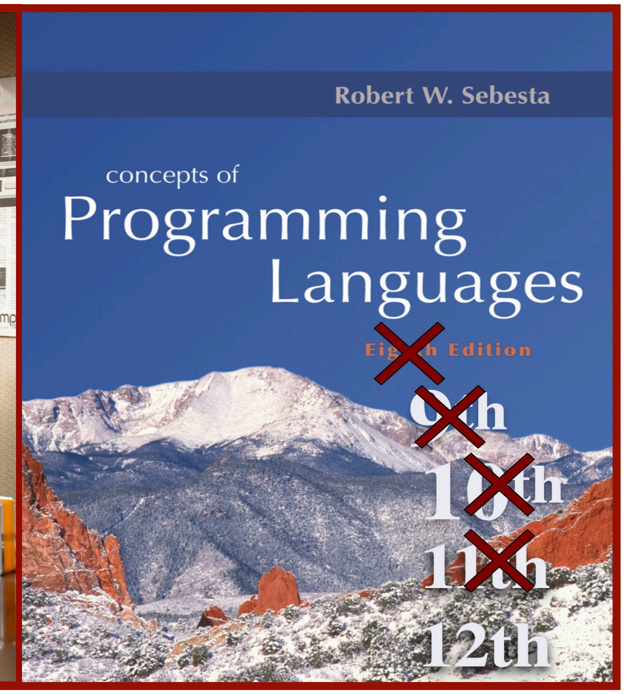
Axiomatic Semantics





Alan G. Labouseur, Ph.D.
Alan.Labouseur@Marist.edu

Axiomatic Semantics

Axiomatic semantics describe the meaning/effect of programs through logic and reasoning about their construction and constraints, allowing proofs.

Thanks to

- Robert Floyd (1967)
- C.A.R. Hoare (1969)
- Edsger Dijkstra (1978)

Axiomatic Semantics

Axiomatic semantics describe the meaning/effect of programs through logic and reasoning about their construction and constraints, allowing proofs.

Thanks to

• Robert Floyd (1967)	graph algorithms (SSSP), parsing, language design
• C.A.R. Hoare (1969)	language design, quicksort, communicating sequential processes
• Edsger Dijkstra (1978)	structured programming, semaphors, graph algorithms (shortest path)

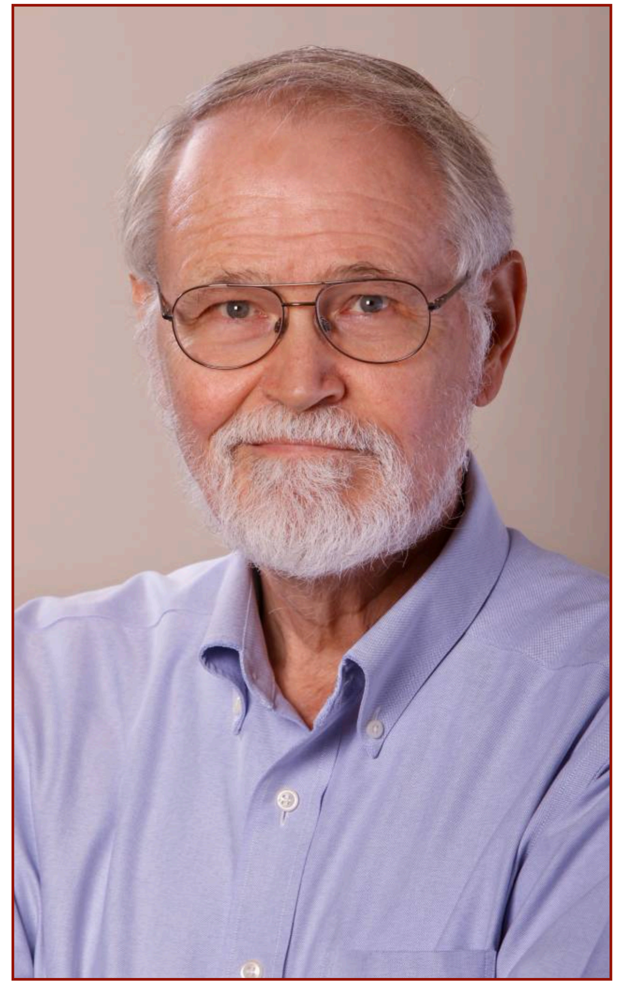
Axiomatic Semantics

Sir Charles Anthony Richard Hoare (C.A.R. Hoare)

“There are two ways of constructing a software design:
One way is to make it so simple that there are obviously no deficiencies, and the other way is to make it so complicated that there are no obvious deficiencies.”

An Aside

Brian Kernighan



"Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it."

Axiomatic Semantics

Sir Charles Anthony Richard Hoare (C.A.R. Hoare)

"Hoare semantics" (aka "Hoare logic") provide the basis for a **theory** of the **partial correctness** of programs.

With this, we can develop formal methods of program verification.

To do so, we'll need to cover the three fundamental requirements that a programming language must support:

- Sequence e.g., assignments, expressions, compound statements
- Alternation e.g., if - then, case, BNE
- Repetition e.g., while, repeat, for, recursion

What is a Theory?

A theory is a framework for proving **properties** about a domain.

For us, that domain is computer programs.

Such properties are called **theorems**, and theorems have proofs.

Components of a theory:

- Grammar — defines well-formed formulae (BNF is one example.)
- Axioms — formulae asserted to be true
- Inference Rules — ways to prove new theorems from previously-proved ones.

Inference Rules

An inference rule is written

$$\frac{f_1,\ f_2,\ \ldots f_n}{f_0}$$

It expresses that **if** $f_1, f_2, \ldots f_n$ are theorems — that is, they are proven well-formed formulae (WFF) — **then** we can infer that f_0 is another theorem.

That's nice, but how do we know?
How can we actually prove things?

Let's look at famous inference rule: Modus Ponens.

A Famous Inference Rule

Modus Ponens

$$\frac{p,\ p \Rightarrow q}{q}$$

Modus Ponens ("the mode that affirms") can be read:

if

we have p (meaning, p is true) **and**

p implies q

then

we can infer that q is true.

end if

The implication/conditional operator ($\Rightarrow$) is like a contract:
if p then q.

Inference Rule vs. Propositional Connective

Modus Ponens

$$\frac{p,\ p \Rightarrow q}{q}$$

Inference Rule

Modus Ponens ("the mode that affirms") can be read:

if

we have p (meaning, p is true) **and**

p implies q

then

we can infer that q is true.

end if

Propositional Connective

The implication/conditional operator ($\Rightarrow$) is like a contract: if p then q.

Let's review this in Propositional logic.

Propositional Logic

Truth Tables

p	q
0	0
0	1
1	0
1	1

Propositional logic has only false and true, no variables.
It also has logical operators like and, or, and not.
(propositional connectors)

Propositional Logic

Truth Tables

p	q	p∧q
0	0	0
0	1	0
1	0	0
1	1	1

Propositional logic has only false and true, no variables.
It also has logical operators like **and**, or, and not.

Propositional Logic

Truth Tables

p	q	p∧q	p∨q
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

Propositional logic has only false and true, no variables.
It also has logical operators like and, **or**, and not.

Propositional Logic

Truth Tables

p	q	p∧q	p∨q	¬p
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

Propositional logic has only false and true, no variables.
It also has logical operators like and, or, and **not**.

Do we need more? (Do we even need all of these?)

Propositional Logic

Truth Tables

p	q	p∧q	p∨q	¬p
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

Propositional logic has only false and true, no variables.
It also has logical operators like and, or, and not.

Do we need more? No. (Do we even need all of these? No.)

$$p \lor q = \neg(\neg p \land \neg q)$$

Propositional Logic

Truth Tables

p	q	p∧q	p∨q	¬p	p⇒q
0	0	0	0	1	
0	1	0	1	1	
1	0	0	1	0	
1	1	1	1	0	

Propositional logic has only false and true, no variables.
It also has logical operators like and, or, and not.

Implication is like a contract:
"if *p* then *q*" or "$p \Rightarrow q$".

Propositional Logic

Truth Tables

p	q	p∧q	p∨q	¬p	p⇒q
0	0	0	0	1	1
0	1	0	1	1	1
1	0	0	1	0	
1	1	1	1	0	

These are vacuously true because *p* is false and false can imply anything because it's an invalid premise.

Also, we take "if *p* then *q*" to be false **only** when *p* is true and *q* is false.

Propositional logic has only false and true, no variables.
It also has logical operators like and, or, and not.

Implication is like a contract:
"if *p* then *q*" or "$p \Rightarrow q$".

Propositional Logic

Truth Tables

p	q	p∧q	p∨q	¬p	p⇒q
0	0	0	0	1	1
0	1	0	1	1	1
1	0	0	1	0	0
1	1	1	1	0	

This is false because *p* is true and *q* is false, and "true implies false" is false.

Propositional logic has only false and true, no variables.
It also has logical operators like and, or, and not.

Implication is like a contract:
"if *p* then *q*" or "$p \Rightarrow q$".

Propositional Logic

Truth Tables

p	q	p∧q	p∨q	¬p	p⇒q
0	0	0	0	1	1
0	1	0	1	1	1
1	0	0	1	0	0
1	1	1	1	0	1

This is true because p is true and q is true, and "true implies true" is true.

Propositional logic has only false and true, no variables.
It also has logical operators like and, or, and not.

Implication is like a contract:
"if p then q" or "$p \Rightarrow q$".

Propositional Logic

Truth Tables

p	q	p∧q	p∨q	¬p	p⇒q	¬p∨q
0	0	0	0	1	1	
0	1	0	1	1	1	
1	0	0	1	0	0	
1	1	1	1	0	1	

Propositional logic has only false and true, no variables.
It also has logical operators like and, or, and not.

Implication is like a contract:
"if p then q" or "$p \Rightarrow q$".
Implication can also be written as $\neg p \vee q$.

Propositional Logic

Truth Tables

These two columns are the same.
Both are implication.

p	q	p∧q	p∨q	¬p	p⇒q	¬p∨q
0	0	0	0	1	1	1
0	1	0	1	1	1	1
1	0	0	1	0	0	0
1	1	1	1	0	1	1

Propositional logic has only false and true, no variables.
It also has logical operators like and, or, and not.

Implication is like a contract:
"if p then q" or "$p \Rightarrow q$".
Implication can also be written as $\neg p \vee q$.

Propositional Logic

Truth Tables

p	q	p∧q	p∨q	¬p	p⇒q	¬p∨q	p∧q⇒(p⇒q)
0	0	0	0	1	1	1	
0	1	0	1	1	1	1	
1	0	0	1	0	0	0	
1	1	1	1	0	1	1	

Propositional logic has only false and true, no variables.
It also has logical operators like and, or, and not.

Implication is like a contract:
"if p then q" or "$p \Rightarrow q$".
Implication can also be written as $\neg p \vee q$.

A tautology is logical statement that is always true regardless of the truth values of its components. Here's one: $p \wedge q \Rightarrow (p \Rightarrow q)$

Propositional Logic

Truth Tables

p	q	$p\wedge q$	$p\vee q$	$\neg p$	$p\Rightarrow q$	$\neg p\vee q$	$p\wedge q\Rightarrow(p\Rightarrow q)$
0	0	0	0	1	1	1	1
0	1	0	1	1	1	1	1
1	0	0	1	0	0	0	1
1	1	1	1	0	1	1	1

Tautology

Propositional logic has only false and true, no variables.
It also has logical operators like and, or, and not.

Implication is like a contract:
"if p then q" or "$p \Rightarrow q$".
Implication can also be written as $\neg p \vee q$.

A tautology is logical statement that is always true regardless of the truth values of its components. Here's one: $p \wedge q \Rightarrow (p \Rightarrow q)$

Propositional Logic

Truth Tables

p	q	p∧q	p∨q	¬p	p⇒q	¬p∨q	p∧q⇒(p⇒q)
0	0	0	0	1	1	1	1
0	1	0	1	1	1	1	1
1	0	0	1	0	0	0	1
1	1	1	1	0	1	1	1

What's the opposite of a tautology, where the statement is always false?

Propositional logic has only false and true, no variables.
It also has logical operators like and, or, and not.

Implication is like a contract:
"if p then q" or "$p \Rightarrow q$".
Implication can also be written as $\neg p \vee q$.

A tautology is logical statement that is always true regardless of the truth values of its components. Here's one: $p \wedge q \Rightarrow (p \Rightarrow q)$

Propositional Logic

Truth Tables

p	q	p∧q	p∨q	¬p	p⇒q	¬p∨q	p∧q⇒(p⇒q)
0	0	0	0	1	1	1	1
0	1	0	1	1	1	1	1
1	0	0	1	0	0	0	1
1	1	1	1	0	1	1	1

What's the opposite of a tautology, where the statement is always false?
A contradiction.

Propositional logic has only false and true, no variables.
It also has logical operators like and, or, and not.

Implication is like a contract:
"if p then q" or "$p \Rightarrow q$".
Implication can also be written as $\neg p \vee q$.

A tautology is logical statement that is always true regardless of the truth values of its components. Here's one: $p \wedge q \Rightarrow (p \Rightarrow q)$

Propositional Logic

Truth Tables

A contradiction

p	q	p∧q	p∨q	¬p	p⇒q	¬p∨q	p∧q⇒(p⇒q)	p∧¬p
0	0	0	0	1	1	1	1	0
0	1	0	1	1	1	1	1	0
1	0	0	1	0	0	0	1	0
1	1	1	1	0	1	1	1	0

Contradictions cannot exist.

Propositional logic has only false and true, no
It also has logical operators like and, or, and n

Implication is like a contract:
"if *p* then *q*" or "$p \Rightarrow q$".
Implication can also be written as $\neg p \vee q$.

A tautology is logical statement that is always true regardless of the truth values of its components. Here's one: $p \wedge q \Rightarrow (p \Rightarrow q)$

Back to that Famous Inference Rule

Propositional Logic for Modus Ponens

p	q	p∧q	p∨q	¬p	p⇒q	¬p∨q	p∧q⇒(p⇒q)
0	0	0	0	1	1	1	1
0	1	0	1	1	1	1	1
1	0	0	1	0	0	0	1
1	1	1	1	0	1	1	1

Modus Ponens

$$\frac{p,\ p \Rightarrow q}{q}$$

can be written "if p and $p \Rightarrow q$ then q", which can be written

$$(p \wedge (p \Rightarrow q)) \Rightarrow q$$

A Famous Inference Rule

Propositional Logic for Modus Ponens

p	q	$p \wedge q$	$p \vee q$	$\neg p$	$p \Rightarrow q$	$\neg p \vee q$	$p \wedge q \Rightarrow (p \Rightarrow q)$	$p \wedge (p \Rightarrow q)$
0	0	0	0	1	1	1	1	0
0	1	0	1	1	1	1	1	0
1	0	0	1	0	0	0	1	0
1	1	1	1	0	1	1	1	1

Modus Ponens

$$\frac{p,\ p \Rightarrow q}{q}$$

can be written "if p and $p \Rightarrow q$ then q", which can be written

$$\underline{(p \wedge (p \Rightarrow q))} \Rightarrow q$$

A Famous Inference Rule

Propositional Logic for Modus Ponens

p	q	$p \wedge q$	$p \vee q$	$\neg p$	$p \Rightarrow q$	$\neg p \vee q$	$p \wedge q \Rightarrow (p \Rightarrow q)$	$p \wedge (p \Rightarrow q)$	$(p \wedge (p \Rightarrow q)) \Rightarrow q$
0	0	0	0	1	1	1	1	0	1
0	1	0	1	1	1	1	1	0	1
1	0	0	1	0	0	0	1	0	1
1	1	1	1	0	1	1	1	1	1

Modus Ponens

$$\frac{p,\ p \Rightarrow q}{q}$$

can be written "if p and $p \Rightarrow q$ then q", which can be written

$$\underline{(p \wedge (p \Rightarrow q)) \Rightarrow q}$$

A Famous Inference Rule

Propositional Logic for Modus Ponens

p	q	p∧q	p∨q	¬p	p⇒q	¬p∨q	p∧q⇒(p⇒q)	p∧(p⇒q)	(p∧(p⇒q))⇒q
0	0	0	0	1	1	1	1	0	1
0	1	0	1	1	1	1	1	0	1
1	0	0	1	0	0	0	1	0	1
1	1	1	1	0	1	1	1	1	1

Modus Ponens

$$\frac{p,\ p \Rightarrow q}{q}$$

can be written "if p and $p \Rightarrow q$ then q", which can be written

$$(p \wedge (p \Rightarrow q)) \Rightarrow q$$

Tautology. Woot!

Inference Rules for Axiomatic Semantics

With Modus Ponens proved and used as the basis for inference rules, we need to move from Propositional logic to Predicate logic.

The complexity of proving programs correct cannot be handled with truth tables because we need to accommodate ideas like *any*, *all*, or *some*. Also, we need variables and functions. This leads us to . . .

First Order Logic

- variables
- domains
- named constants
- relations (>, <, etc.)
- functions (math operations)
- logical operators
- quantifiers (for-all “∀” and there-exists “∃”)

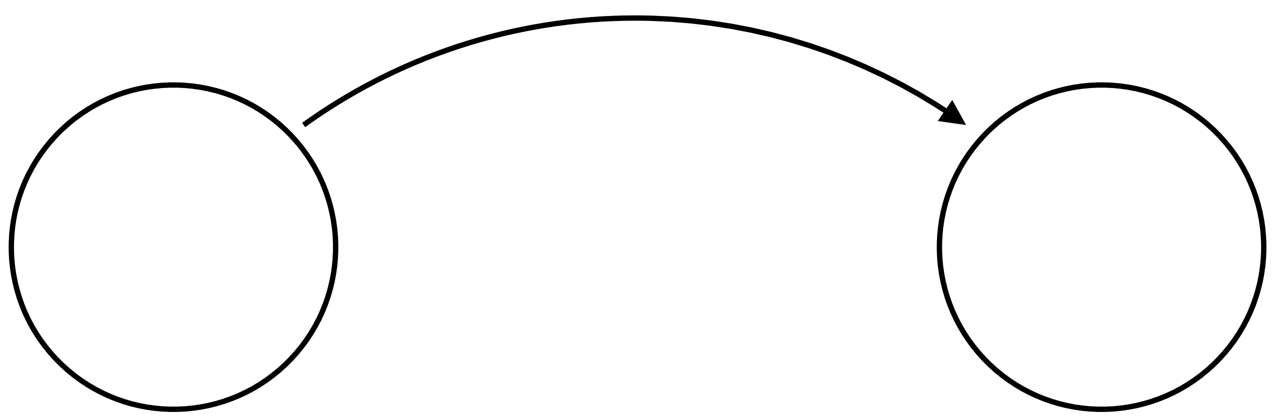
Now we can reason about program correctness.

A Program as State Transitions

We will define dynamic semantics on state transitions.

A state is a mapping of variables to their values.

Executing a program can be viewed as a sequence of state transitions.

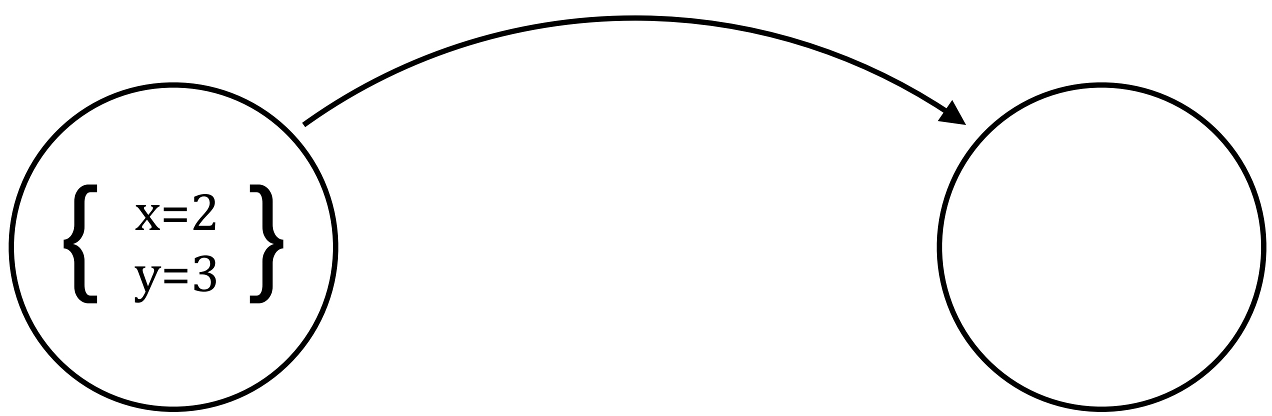


A Program as State Transitions

We will define dynamic semantics on state transitions.

A state is a mapping of variables to their values.

Executing a program can be viewed as a sequence of state transitions.



Precondition

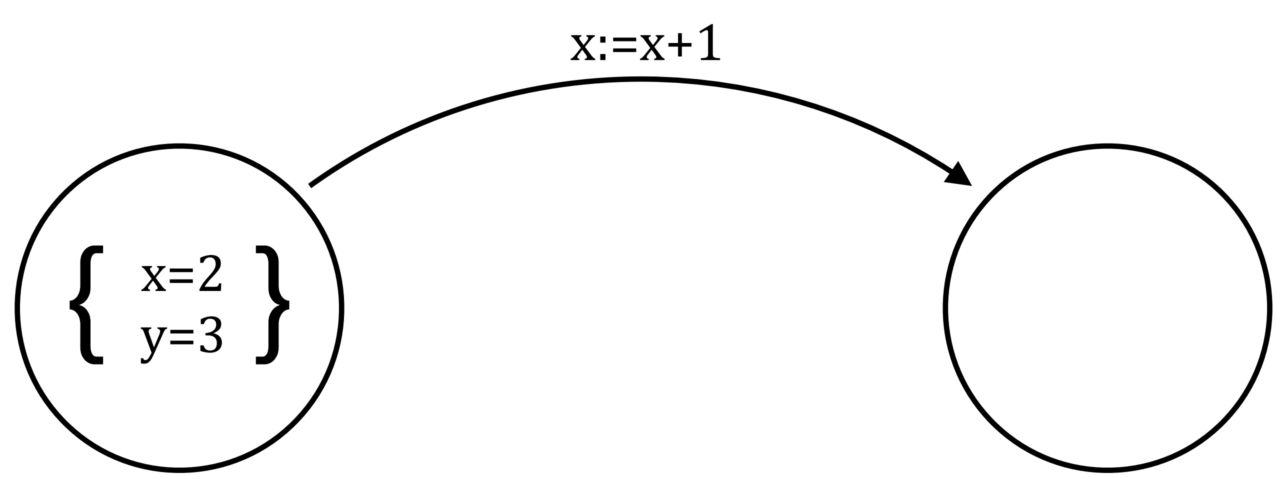
A Program as State Transitions

We will define dynamic semantics on state transitions.

A state is a mapping of variables to their values.

Executing a program can be viewed as a sequence of state transitions.

Statement



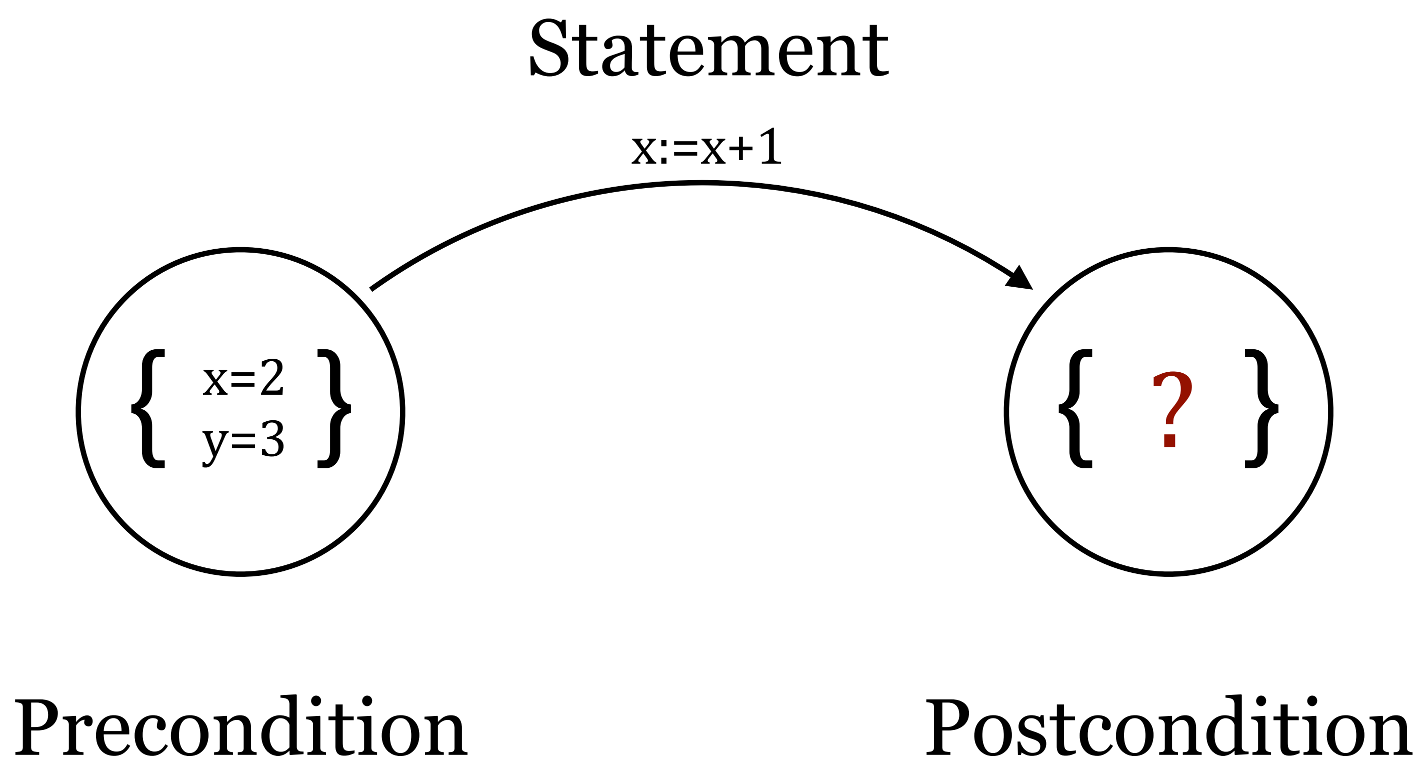
Precondition

A Program as State Transitions

We will define dynamic semantics on state transitions.

A state is a mapping of variables to their values.

Executing a program can be viewed as a sequence of state transitions.



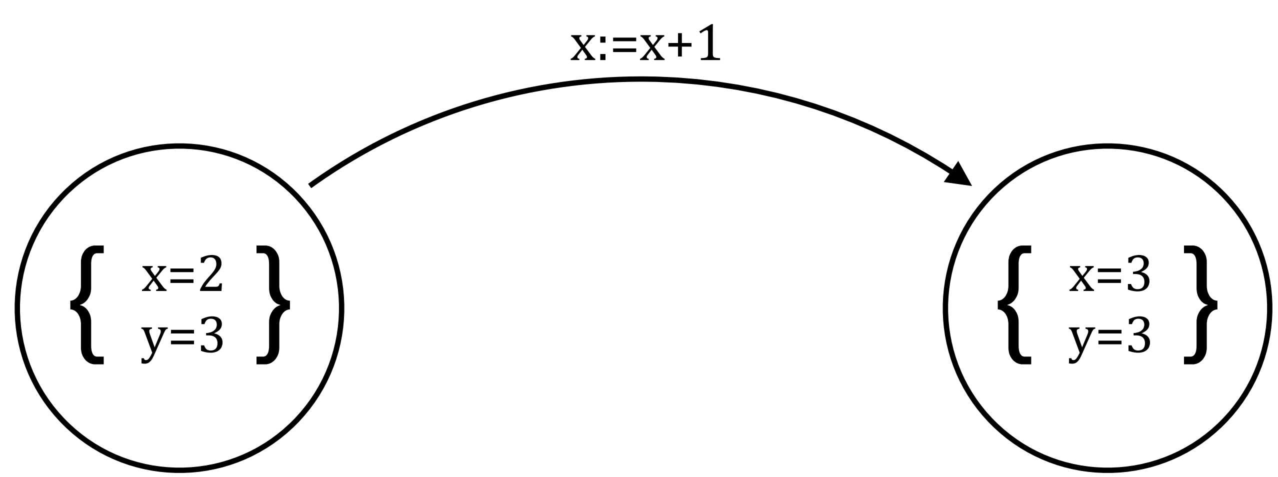
Reasoning About State Transitions

We will define dynamic semantics on state transitions.

A state is a mapping of variables to their values.

Executing a program can be viewed as a sequence of state transitions.

Statement



Precondition

Postcondition

Valid.

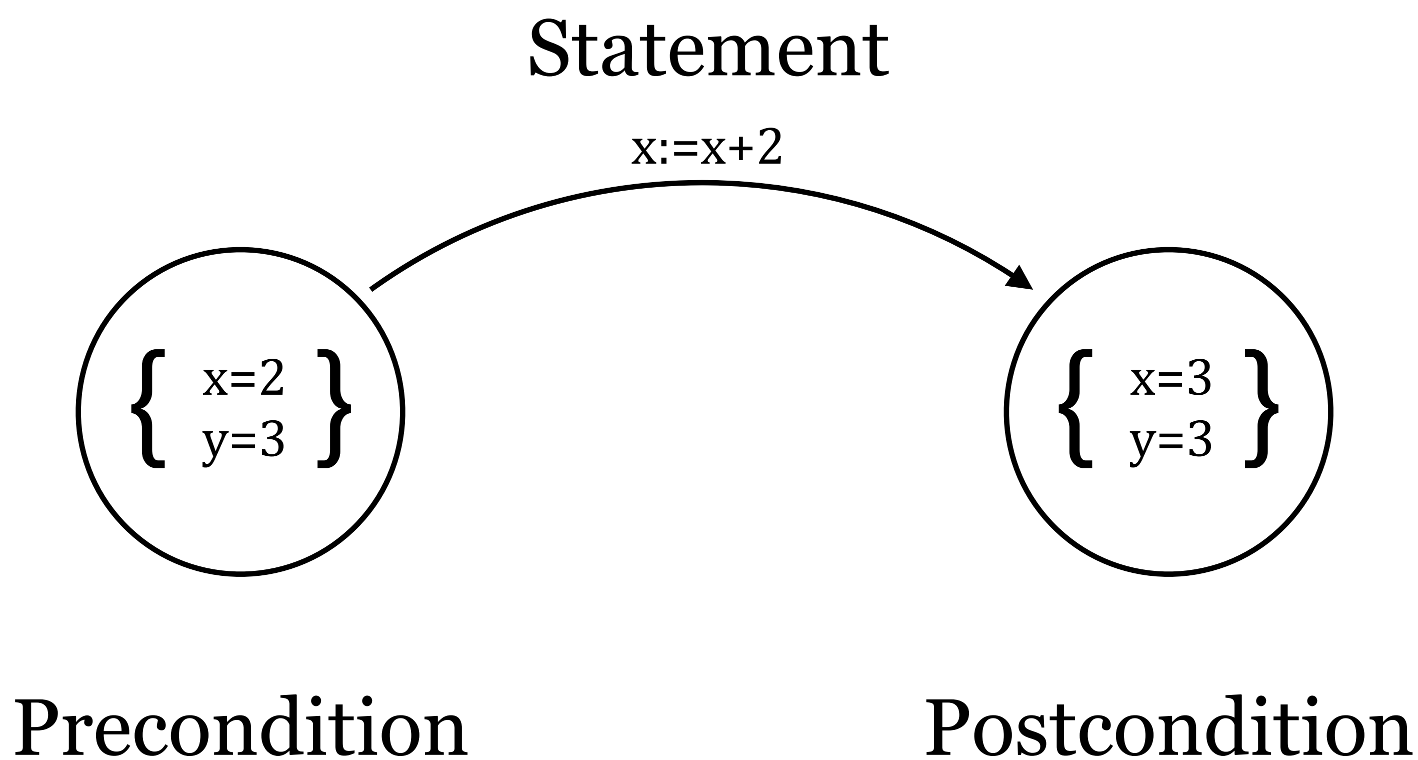
Beginning in a state satisfying the precondition, the instructions will result in a state satisfying the postcondition… if they terminate.

Reasoning About State Transitions

We will define dynamic semantics on state transitions.

A state is a mapping of variables to their values.

Executing a program can be viewed as a sequence of state transitions.



Not Valid.

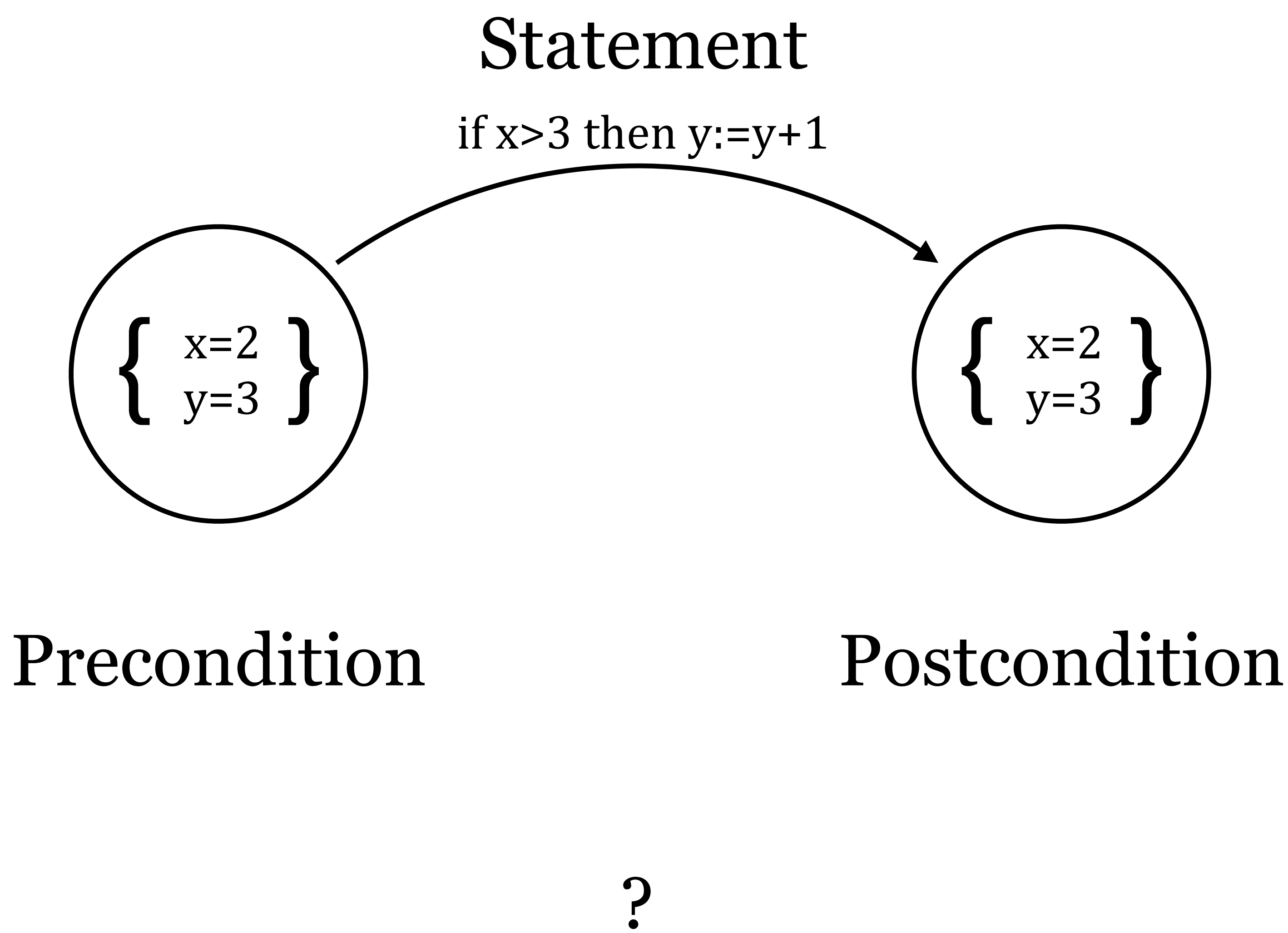
Beginning in a state satisfying the precondition, the instructions will **not** result in a state satisfying the postcondition.

Reasoning About State Transitions

We will define dynamic semantics on state transitions.

A state is a mapping of variables to their values.

Executing a program can be viewed as a sequence of state transitions.

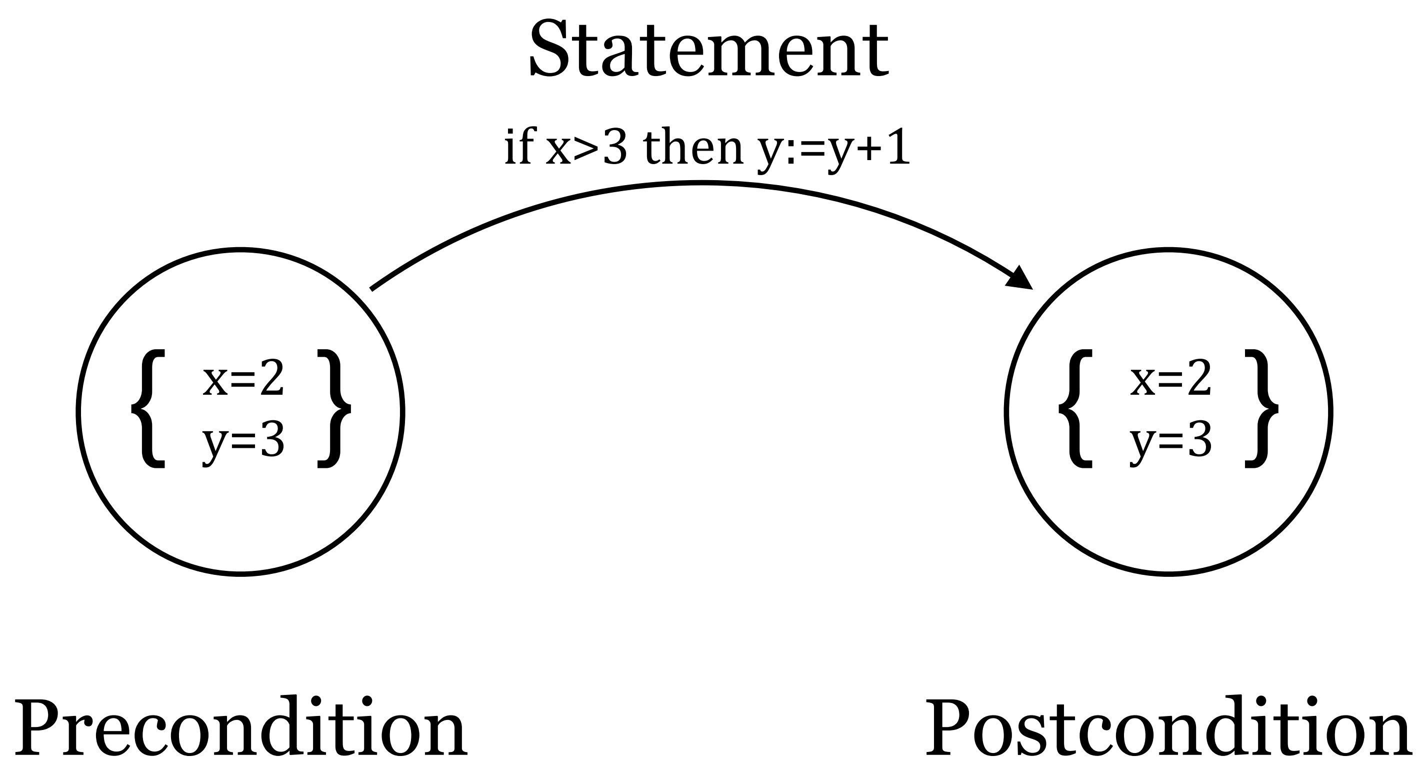


Reasoning About State Transitions

We will define dynamic semantics on state transitions.

A state is a mapping of variables to their values.

Executing a program can be viewed as a sequence of state transitions.



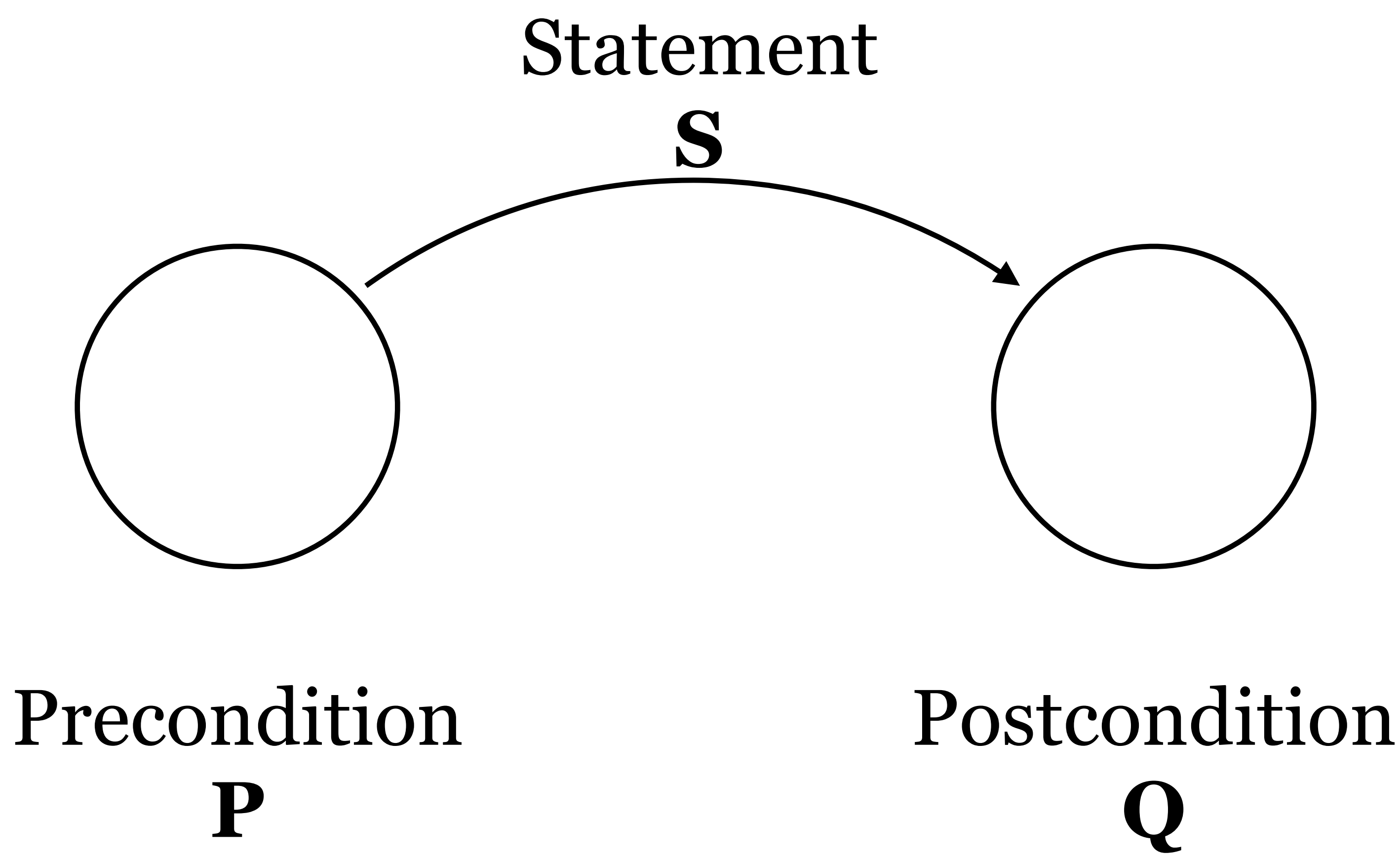
Valid.

Beginning in a state satisfying the precondition, the instructions will result in a state satisfying the postcondition… if they terminate.

Reasoning About State Transitions

In general, we write {P} S {Q} to represent these three objects.
This is a “Hoare Triple”. It means…

“S, starting in any state satisfying P, will satisfy Q on termination.”



Correctness

There been a lot of talk of *termination* in the past few slides.

{P} S {Q}

Total Correctness

- **S**, started in any state satisfying **P**, will terminate in a state satisfying **Q**.
- This requires that we prove termination, which can be difficult or **possibly impossible**.

Partial Correctness

- **S**, started in any state satisfying **P**, will — if it terminates — result in a state satisfying **Q**.
- Now we do not have to prove termination. But we can only call it **partially correct**.

A Fun Aside

What's that about proving termination being possibly impossible?

Surely, I must be joking.

Imagine the following program:

```
halts(p,i) = function {
  if program p halts on input i
    return TRUE
  else
    return FALSE
  endif
}
```

A Fun Aside

What's that about proving termination being possibly impossible?

Surely, I must be joking.

Imagine the following programs:

```
halts(p,i) = function {
  if program p halts on input i
    return TRUE
  else
    return FALSE
  endif
}
```

```
trouble(p) = function {
  if not halts(p,p)
    return TRUE
  else
    loop forever
  endif
}
```

This halts and returns true if the passed-in program does not halt when applied to itself, and it loops forever (i.e., does not halt) otherwise.

Trouble indeed.

A Fun Aside

What's that about proving termination being possibly impossible?

Surely, I must be joking.

Imagine the following programs:

```
halts(p,i) = function {
  if program p halts on input i
    return TRUE
  else
    return FALSE
  endif
}
```

```
trouble(p) = function {
  if not halts(p,p)
    return TRUE
  else
    loop forever
  endif
}
```

What happens when we call `trouble(trouble)`?

A Fun Aside

What's that about proving termination being possibly impossible?

Surely, I must be joking.

Imagine the following programs:

```
halts(p,i) = function {
  if program p halts on input i
    return TRUE
  else
    return FALSE
  endif
}
```

```
trouble(p) = function {
  if not halts(p,p)
    return TRUE
  else
    loop forever
  endif
}
```

What happens when we call `trouble(trouble)` ?

It evaluates `halts(trouble,trouble)`

A Fun Aside

What's that about proving termination being possibly impossible?

Surely, I must be joking?

Imagine the following programs:

```
halts(p,i) = function {
  if program p halts on input i
    return TRUE
  else
    return FALSE
  endif
}
```

```
trouble(p) = function {
  if not halts(p,p)
    return TRUE
  else
    loop forever
  endif
}
```

What happens when we call `trouble(trouble)` ?

It evaluates `halts(trouble,trouble)` There are two possibilities:

(1) it returns TRUE. Not TRUE is FALSE so loop forever, meaning do not halt.

A Fun Aside

What's that about proving termination being possibly impossible?

Surely, I must be joking?

Imagine the following programs:

```
halts(p,i) = function {
  if program p halts on input i
    return TRUE
  else
    return FALSE
  endif
}
```

```
trouble(p) = function {
  if not halts(p,p)
    return TRUE
  else
    loop forever
  endif
}
```

What happens when we call `trouble(trouble)` ?

It evaluates `halts(trouble,trouble)` There are two possibilities:

(1) it returns TRUE. Not TRUE is FALSE so loop forever, meaning do not halt.

(2) it returns FALSE. Not FALSE is TRUE so return TRUE, meaning it halts.

What?

A Fun Aside

What's that about proving termination being possibly impossible?

Surely, I must be joking…

Imagine the following programs:

```
halts(p,i) = function {
  if program p halts on input i
    return TRUE
  else
    return FALSE
  endif
}
```

```
trouble(p) = function {
  if not halts(p,p)
    return TRUE
  else
    loop forever
  endif
}
```

What happens when we call `trouble(trouble)` ?

It evaluates `halts(trouble,trouble)` There are two possibilities:

(1) it returns TRUE. Not TRUE is FALSE so loop forever, meaning do not halt.

(2) it returns FALSE. Not FALSE is TRUE so return TRUE, meaning it halts.

In other words, if `halts(trouble,trouble)` halts then it doesn't, and if `halts(trouble,trouble)` doesn't halt then it does. It halts and loops at the same time.

A Fun Aside

What's that about proving termination being possibly impossible?

I'm not joking. And don't call me Shirley.

Imagine the following programs:

```
halts(p,i) = function {
  if program p halts on input i
    return TRUE
  else
    return FALSE
  endif
}
```

```
trouble(p) = function {
  if not halts(p,p)
    return TRUE
  else
    loop forever
  endif
}
```

What happens when we call `trouble(trouble)` ?

It evaluates `halts(trouble,trouble)` There are two possibilities:

(1) it returns TRUE. Not TRUE is FALSE so loop forever, meaning do not halt.

(2) it returns FALSE. Not FALSE is TRUE so return TRUE, meaning it halts.

In other words, if `halts(trouble,trouble)` halts then it doesn't, and if `halts(trouble,trouble)` doesn't halt then it does. It halts and loops at the same time.

`halts()` is a contradiction. Contradictions cannot exist. Therefore, by our own reasoning, `halts()` cannot exist and promptly vanishes in a puff of logic.

A Fun Aside

What's that about proving termination being possibly impossible?

I'm not joking. And don't call me Shirley.

Imagine the following programs:

```
halts(p,i) = function {
  if program p halts on input i
    return TRUE
  else
    return FALSE
  endif
}
```

```
paradox(p) = function {
  if not halts(p,p)
    return TRUE
  else
    loop forever
  endif
}
```

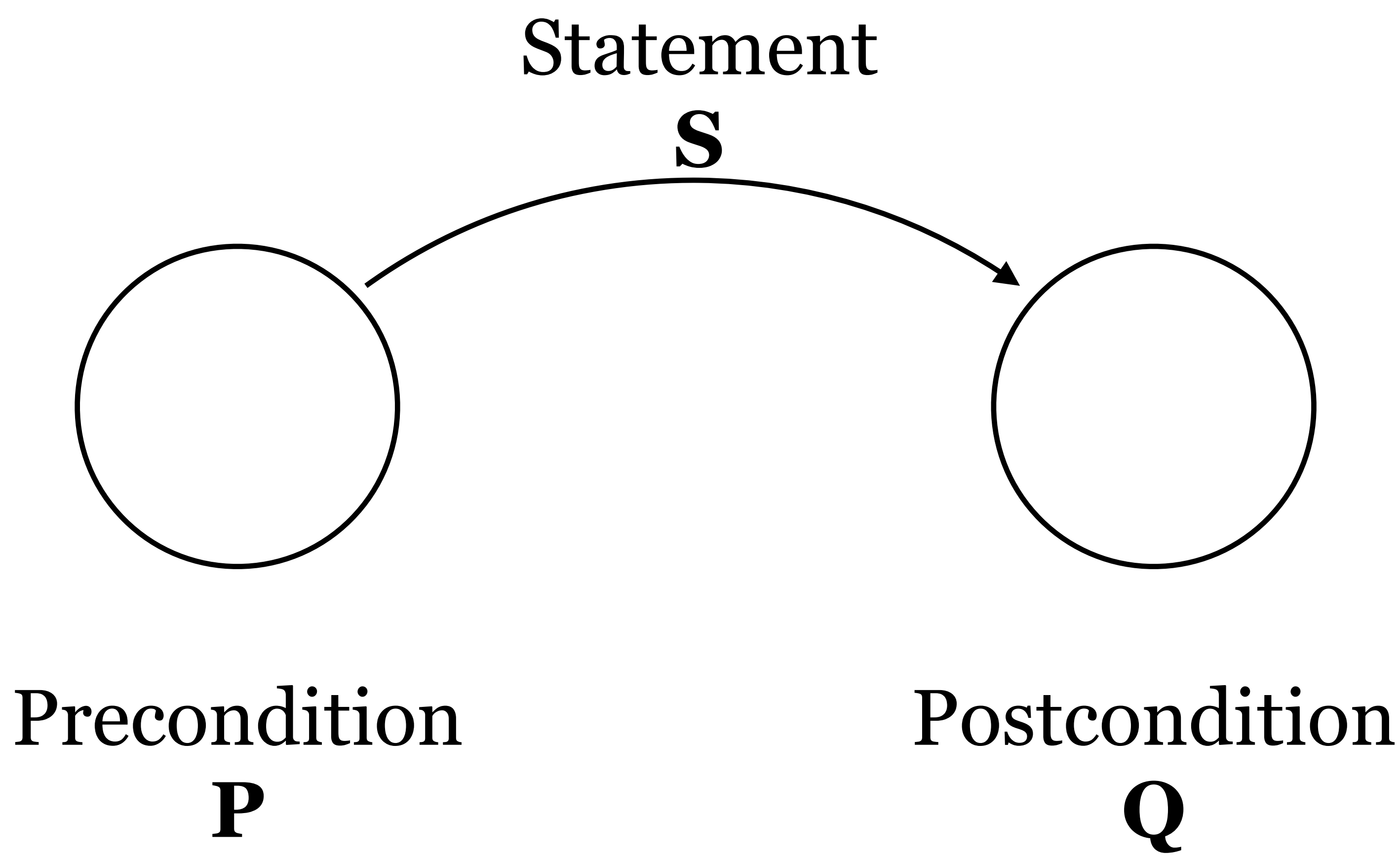
We might rename `trouble()` to `paradox()` and then call `paradox(paradox)` to be even more clear about what's going on.

There's an interesting blog post at `https://lacker.io/math/2022/02/24/godels-incompleteness-in-bash.html` that gives a similar example using bash scripts. It also connects the halting problem to Gödel's incompleteness theorems, which is very cool.

Back to State Transitions

In general, we write {P} S {Q} to represent these three objects.
This is a "Hoare Triple". It means…

"S, starting in any state satisfying P, will satisfy Q on termination."



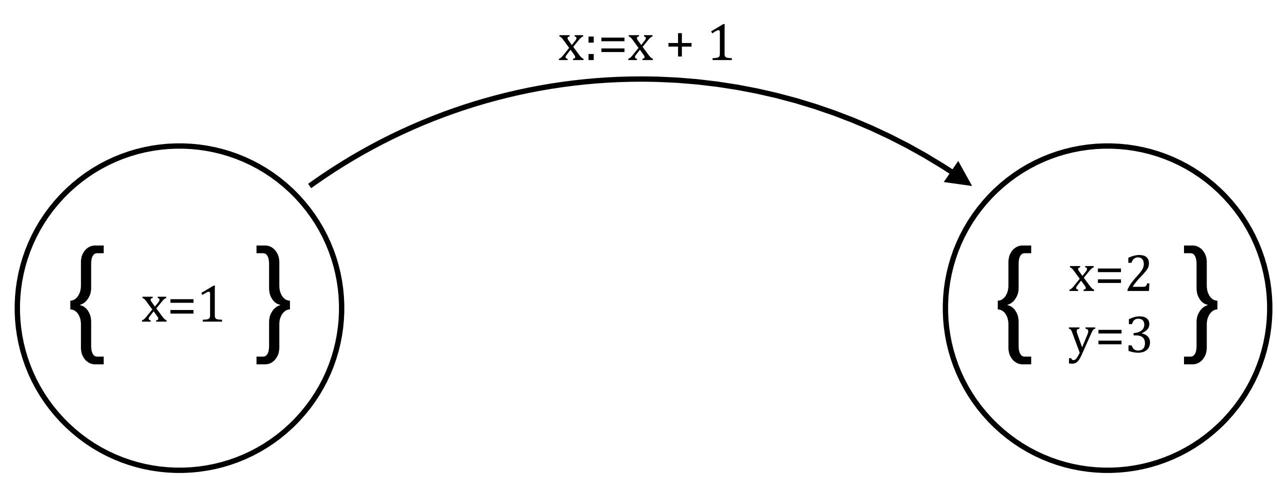
We want to prove statements and eventually programs correct.

Reasoning About State Transitions

We want to prove statements and eventually programs correct.

{x=1} x:=x+1 {x=2, y=3}

Statement



Precondition

Postcondition

Reasoning About State Transitions

We want to prove statements and eventually programs correct.

$\{x=1\}$ $x:=x+1$ $\{x=2, y=3\}$ Valid?

Reasoning About State Transitions

We want to prove statements and eventually programs correct.

{x=1} x:=x+1 {x=2, y=3}

Not valid.
The precondition says nothing about y so we cannot assert anything about y in the postcondition.

Reasoning About State Transitions

We want to prove statements and eventually programs correct.

{x=1} x:=x+1 {x=2, y=3}	Not valid.
{x=y+1} x:=x+1 {x=y}	Valid?

Reasoning About State Transitions

We want to prove statements and eventually programs correct.

{x=1} x:=x+1 {x=2, y=3}

Not valid.

{x=y+1} x:=x+1 {x=y}

Not Valid.

$x > y$ in the precondition, then x gets incremented, so x cannot be equal to y in the postcondition.

Reasoning About State Transitions

We want to prove statements and eventually programs correct.

{x=1} x:=x+1 {x=2, y=3}	Not valid.
{x=y+1} x:=x+1 {x=y}	Not Valid.
{x=y+1} y:=y+1 {x=y}	Valid?

Reasoning About State Transitions

We want to prove statements and eventually programs correct.

{x=1} x:=x+1 {x=2, y=3}	Not valid.
{x=y+1} x:=x+1 {x=y}	Not Valid.
{x=y+1} y:=y+1 {x=y}	Valid.
{x>y} if x>3 then x:=x+1 {x>y}	Valid?

Reasoning About State Transitions

We want to prove statements and eventually programs correct.

{x=1} x:=x+1 {x=2, y=3}	Not valid.
{x=y+1} x:=x+1 {x=y}	Not Valid.
{x=y+1} y:=y+1 {x=y}	Valid.
{x>y} if x>3 then x:=x+1 {x>y}	Valid.
{x=10} while x>0 do x:=x+1 {x=2}	Valid?

Reasoning About State Transitions

We want to prove statements and eventually programs correct.

{x=1} x:=x+1 {x=2, y=3} — Not valid.
{x=y+1} x:=x+1 {x=y} — Not Valid.
{x=y+1} y:=y+1 {x=y} — Valid.
{x>y} if x>3 then x:=x+1 {x>y} — Valid.
{x=10} while x>0 do x:=x+1 {x=2} — Valid? One the one hand, the statement cannot be partially correct because if it somehow terminates the post-condition will not be satisfied. But it will never terminate, so in that sense the statement is vacuously true. It's annoying, and borderline ridiculous.

Partial Correctness

- **S**, started in any state satisfying **P**, will — if it terminates — result in a state satisfying **Q**.

Reasoning About State Transitions

We want to prove statements and eventually programs correct.

{x=1} x:=x+1 {x=2, y=3}	Not valid.
{x=y+1} x:=x+1 {x=y}	Not Valid.
{x=y+1} y:=y+1 {x=y}	Valid.
{x>y} if x>3 then x:=x+1 {x>y}	Valid.
{x=10} while x>0 do x:=x+1 {x=2}	Vacuous and annoying.

{x=10} x:=x-3 {x=7}	Valid.
{x>10} x:=x-3 {x>7}	Valid.
{x>10} x:=x-3 {x>2}	Valid.
{x>10, y=2} x:=x-3 {x>7}	Valid.
{x=10} x:=x-3 {x=8}	Not valid.
{x=10} x:=x-3 {x>10}	Not valid.
{x=10} x:=x-3 {y>2}	Not valid.

We need to formalize our reasoning.

Reasoning About State Transitions

What is our reasoning?

$$\{x>10\}\ x:=x-3\ \{x>7\}$$

Why is this valid?

Reasoning About State Transitions

What is our reasoning?

$\{x>10\}\ x{:=}\boxed{x-3}\ \{x\boxed{>7}\}$

Why is this valid?

Make an equation of the instructions and postcondition…

$x-3 > 7$

Reasoning About State Transitions

What is our reasoning?

$$\{x>10\}\ x:=\boxed{x-3}\ \{x\boxed{>7}\}$$

Why is this valid?

Make an equation of
the instructions and
postcondition and solve it.

$$x-3 > 7$$
$$x > 10$$

Then compare to
the precondition.

If the solution matches the precondition then we know the
Hoare triple is valid.

Substitution

We'll use a tool called substitution to facilitate this.

$$C\,[\,A/B\,]$$

"Substitute A for B in expression C."

x[x/x] means "x for x in x" = x

x[y/x] means "y for x in x" = y

x[x/y] means "x for y in x" = x (because there is no *y* in *x*)

x[z/y] means "z for y in x" = x (because there is no *y* in *x*)

3 • x+1 [y/x] means "y for x in 3 • x+1" = 3 • y+1

Assignment Axiom

Remember the Hoare triple form:

$$\{P\}\ S\ \{Q\}$$

Given assignment statement S …

$$\{P\}\ x := e\ \{Q\}$$

… we use substitution to evaluate the validity of this triple.

Substitute Q[e/x] and compare the result to P.

$$\{P\}\ x := e\ \{Q\}[e/x]$$

Assignment Axiom

Remember the Hoare triple form:

$$\{P\}\ S\ \{Q\}$$

Given assignment statement S …

$$\{P\}\ x := e\ \{Q\}$$

… we use substitution to evaluate the validity of this triple.

Substitute Q[e/x] and compare the result to P.
"Substitute **e for x in Q** and compare the result to P."

$$\{P\}\ x := e\ \{Q\}[e/x]$$

Assignment Axiom

Remember the Hoare triple form:

$$\{P\}\ S\ \{Q\}$$

Given assignment statement S …

$$\{P\}\ x := e\ \{Q\}$$

… we use substitution to evaluate the validity of this triple.

Substitute Q[e/x] and compare the result to P.
"Substitute e for x in Q and **compare the result to P**."

$$\{P\}\ x := e\ \underline{\{Q\}[e/x]}$$

Assignment Axiom

Remember the Hoare triple form:

$$\{P\}\ S\ \{Q\}$$

Given assignment statement S ...

$$\{P\}\ x := e\ \{Q\}$$

... we use substitution to evaluate the validity of this triple.
Substitute Q[e/x] and compare the result to P.

Example: {P} S {Q}
{y>0} x:=y {x>0}

Assignment Axiom

Remember the Hoare triple form:

$$\{P\}\ S\ \{Q\}$$

Given assignment statement S …

$$\{P\}\ x := e\ \{Q\}$$

… we use substitution to evaluate the validity of this triple.
Substitute Q[e/x] and compare the result to P.

Example: {P} S {Q}

{y>0} x:=y {x>0}

{y>0} x:=y {x>0} [y/x] *// substitute y for x in {x>0}*

Assignment Axiom

Remember the Hoare triple form:

$$\{P\}\ S\ \{Q\}$$

Given assignment statement S …

$$\{P\}\ x := e\ \{Q\}$$

… we use substitution to evaluate the validity of this triple.
Substitute Q[e/x] and compare the result to P.

Example:

{P} S {Q}
{y>0} x:=y {x>0}
{y>0} x:=y {x>0} [y/x] *// substitute y for x in {x>0}*
{y>0} x:=y {y>0}

Assignment Axiom

Remember the Hoare triple form:

$$\{P\}\ S\ \{Q\}$$

Given assignment statement S ...

$$\{P\}\ x := e\ \{Q\}$$

... we use substitution to evaluate the validity of this triple.
Substitute Q[e/x] and compare the result to P.

Example: {P} S {Q}

{y>0} x:=y {x>0}

{y>0} x:=y {x>0} [y/x] *// substitute y for x in {x>0}*

{y>0} x:=y {y>0} *// compare {new Q} to {P}*

Valid.

Assignment Axiom

Remember the Hoare triple form:

$$\{P\}\ S\ \{Q\}$$

Given assignment statement S …

$$\{P\}\ x := e\ \{Q\}$$

… we use substitution to evaluate the validity of this triple.
Substitute Q[e/x] and compare the result to P.

Example: $\{P\}$ S $\{Q\}$

$\{y>z-2\}\ x{:=}\boxed{x+1}\ \{y>z-2\}$

$\{y>z-2\}\ x{:=}x+1\ \underline{\{y>z-2\}\ [x+1/x]}$

"Substitute x+1 for x in y>z-2."

Assignment Axiom

Remember the Hoare triple form:

$$\{P\}\ S\ \{Q\}$$

Given assignment statement S …

$$\{P\}\ x := e\ \{Q\}$$

… we use substitution to evaluate the validity of this triple.
Substitute Q[e/x] and compare the result to P.

Example: {P} S {Q}

{y>z-2} x:=x+1 {y>z-2}

{y>z-2} x:=x+1 {y>z-2} [x+1/x]

{y>z-2} x:=x+1 {y>z-2} *// No change. There is no x in y>z-2.*

Valid. Not interesting. But valid.

Assignment Axiom

Remember the Hoare triple form:

$$\{P\}\ S\ \{Q\}$$

Given assignment statement S ...

$$\{P\}\ x := e\ \{Q\}$$

... we use substitution to evaluate the validity of this triple.
Substitute Q[e/x] and compare the result to P.

Example: {P} S {Q}

{2+2=5} x:=x+1 {2+2=5}

?

Assignment Axiom

Remember the Hoare triple form:

$$\{P\}\ S\ \{Q\}$$

Given assignment statement S …

$$\{P\}\ x := e\ \{Q\}$$

… we use substitution to evaluate the validity of this triple.
Substitute Q[e/x] and compare the result to P.

Example: $\{P\}$ S $\{Q\}$

$\{2+2=5\}$ $x:=x+1$ $\{2+2=5\}$

Vacuously valid because false implies anything.

Assignment Axiom

Remember the Hoare triple form:

$$\{P\}\ S\ \{Q\}$$

Given assignment statement S …

$$\{P\}\ x := e\ \{Q\}$$

… we use substitution to evaluate the validity of this triple.
Substitute Q[e/x] and compare the result to P.

Example: $\{P\} \quad S \quad \{Q\}$

$\{x+1>0\}\ x:=x+1\ \{x>0\}$

$\{x+1>0\}\ x:=x+1\ \{x>0\}\ [x+1/x]$

Assignment Axiom

Remember the Hoare triple form:

$$\{P\}\ S\ \{Q\}$$

Given assignment statement S …

$$\{P\}\ x := e\ \{Q\}$$

… we use substitution to evaluate the validity of this triple.
Substitute Q[e/x] and compare the result to P.

Example: $\{P\}$ S $\{Q\}$

$\{x+1>0\}\ x:=x+1\ \{x>0\}$

$\{x+1>0\}\ x:=x+1\ \underline{\{x>0\}\ [x+1/x]}$

"Substitute x+1 for x in x>0."

Assignment Axiom

Remember the Hoare triple form:

$$\{P\}\ S\ \{Q\}$$

Given assignment statement S ...

$$\{P\}\ x := e\ \{Q\}$$

... we use substitution to evaluate the validity of this triple.
Substitute Q[e/x] and compare the result to P.

Example:

$\{P\} \quad S \quad \{Q\}$

$\{x+1>0\}\ x:=x+1\ \{x>0\}$

$\{x+1>0\}\ x:=x+1\ \{x>0\}\ [x+1/x]$

$\{x+1>0\}\ x:=x+1\ \{x+1>0\}$

Valid.

Assignment Axiom

Remember the Hoare triple form:

$$\{P\}\ S\ \{Q\}$$

Given assignment statement S …

$$\{P\}\ x := e\ \{Q\}$$

… we use substitution to evaluate the validity of this triple.
Substitute Q[e/x] and compare the result to P.

Example:

{P}	S	{Q}
$\{x=y+1\}$	$x:=x+1$	$\{x=y\}$
$\{x=y+1\}$	$x:=x+1$	$\{x=y\}\ [x+1/x]$
$\{x=y+1\}$	$x:=x+1$	$\{x+1=y\}$

Not Valid.

Assignment Axiom

Remember the Hoare triple form:

$$\{P\}\ S\ \{Q\}$$

Given assignment statement S …

$$\{P\}\ x := e\ \{Q\}$$

… we use substitution to evaluate the validity of this triple.
Substitute Q[e/x] and compare the result to P.

Example:

{P}	S	{Q}	
{x=10}	x:=x-3	{x>10}	
{x=10}	x:=x-3	{x>10}	[x-3/x]
{x=10}	x:=x-3	{x-3>10}	

Not Valid.

Computing the Precondition

Sometimes we want to compute the precondition.

$$\{\ ?\ \}\ sum := 2 \cdot x + 1\ \{sum \geq 1\}$$

Here are a few valid values for x: 10, 7, 1, 0, 2112, 42
Here are a few invalid values for x: -2, -8675309, -5150

What is the **minimum** valid value for x?
We'll call this the Weakest Precondition.

Computing the Weakest Precondition

Sometimes we want to compute the preconditions.

$$\{\ ?\ \}\ sum := 2 \cdot x + 1\ \{sum \geq 1\}$$

Here are a few valid values for x: 10, 7, 1, 0, 2112, 42
Here are a few invalid values for x: -2, -8675309, -5150

What is the **minimum** valid value for x?
We'll call this the Weakest Precondition.

Set up an equation using the assignment axiom and then solve it.

$$\{sum \geq 1\}\ [2 \cdot x + 1\ /\ sum]$$

$$\begin{aligned} 2 \cdot x + 1 &\geq 1 \\ 2 \cdot x &\geq 0 \\ x &\geq 0/2 \\ x &\geq 0 \end{aligned}$$

Computing the Weakest Precondition

Sometimes we want to compute the preconditions.

$$\{\ \mathbf{x \geq 0}\ \}\ sum := 2 \cdot x + 1\ \{sum \geq 1\}$$

Here are a few valid values for x: 10, 7, 1, 0, 2112, 42
Here are a few invalid values for x: -2, -8675309, -5150

What is the minimum valid value for x?
We'll call this the Weakest Precondition.

Set up an equation using the assignment axiom and then solve it.

$$\{sum \geq 1\}\ [2 \cdot x + 1\ /\ sum]$$

$$\begin{aligned} 2 \cdot x + 1 &\geq 1 \\ 2 \cdot x &\geq 0 \\ x &\geq 0/2 \\ x &\geq 0 \longleftarrow \text{Weakest Precondition} \end{aligned}$$

Computing the Weakest **Liberal** Precondition

Sometimes we want to compute the preconditions.

$$\{\ \mathbf{x \geq 0}\ \}\ \text{sum} := 2 \cdot x + 1\ \{\text{sum} \geq 1\}$$

Here are a few valid values for x: 10, 7, 1, 0, 2112, 42

Here are a few invalid values for x: -2, -8675309, -5150

What is the minimum valid value for x?

We'll call this the Weakest Precondition.

Set up an equation using the assignment axiom and then solve it.

$$\{\text{sum} \geq 1\}\ [2 \cdot x + 1 \,/\, \text{sum}]$$

$$2 \cdot x + 1 \geq 1$$

$$2 \cdot x \geq 0$$

$$x \geq 0/2$$

$$x \geq 0$$

Actually, it's the Weakest **Liberal** Precondition (WLP) because we're not making any assertions about termination.

Computing the Weakest Liberal Precondition

$$\{\ ?\ \}\ \ x := x - 3\ \ \{x \geq 0\}$$

Set up an equation using the assignment axiom ...

$$\mathbf{\{P\}\ x := e\ \{Q\}}$$
$$\mathbf{Q[e/x]}$$

$$\{x \geq 0\}\ [x - 3\ /\ x]$$

... and solve it.

$$x - 3 \geq 0$$
$$x \geq 3$$

Weakest Liberal Precondition (WLP)

Computing the Weakest Liberal Precondition

WLP

{ ? } a := b/2 - 1 {a < 10}

Set up an equation using the assignment axiom ...

{P} x := e {Q}

Q[e/x]

{a < 10} [b/2 - 1 / a]

... and solve it.

b/2 - 1 < 10

b/2 < 11

b < 22 ← WLP

{ b < 22 } a := b/2 - 1 {a < 10}

Valid

Computing the Weakest Liberal Precondition

WLP

$\{\ ?\ \}\ x := 2 \cdot y - 3\ \{x > 25\}$

Set up an equation using the assignment axiom …

$$\mathbf{\{P\}\ x := e\ \{Q\}}$$
$$\mathbf{Q[e/x]}$$

$\{x > 25\}\ [2 \cdot y - 3\ /\ x]$

… and solve it.

$$2 \cdot y - 3 > 25$$
$$2 \cdot y > 28$$
$$y > 14$$

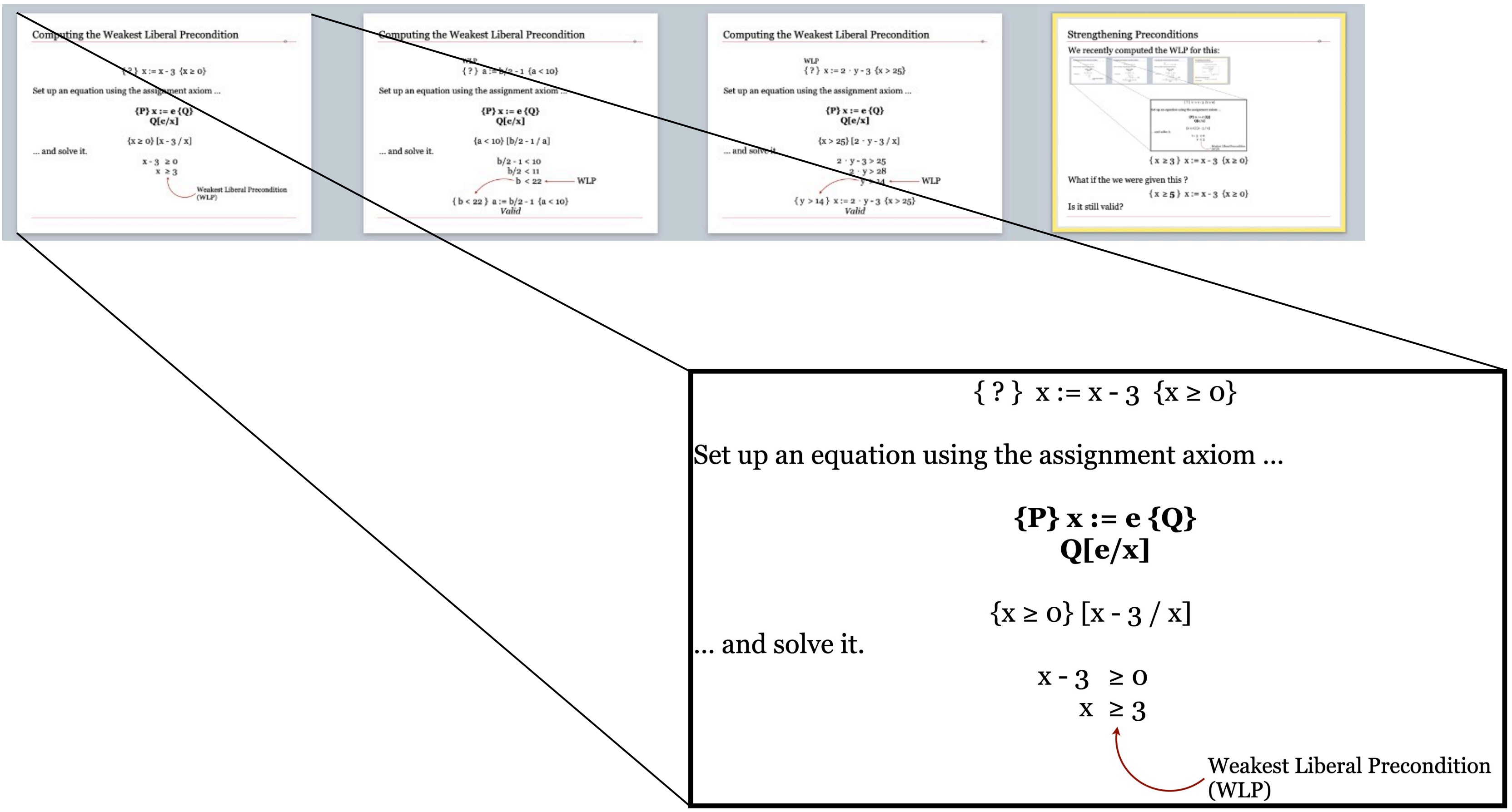
WLP

$\{\ y > 14\ \}\ x := 2 \cdot y - 3\ \{x > 25\}$

Valid

Stronger Preconditions

We recently computed the WLP for this:



$$\{ x \geq 3 \} \;\; x := x - 3 \;\; \{x \geq 0\}$$

What if the we were given this ?

$$\{ x \geq \mathbf{5} \} \;\; x := x - 3 \;\; \{x \geq 0\}$$

Is it still valid?

Stronger Preconditions

Is this valid?

$$\{ x \geq 5 \} \; x := x - 3 \; \{ x \geq 0 \}$$

Compute the WLP using the **Assignment Axiom**…

$\{P\}$ x := e **$\{Q\}[e/x]$**

Bad?

$\{x \geq 0\}$ $[x - 3 / x]$

$x - 3 \geq 0$

$x \geq 3$

On the one hand, $x \geq 5 \neq x \geq 3$ so it seems bad.

Stronger Preconditions

Is this valid?

$$\{ x \geq 5 \}\ x := x - 3\ \{ x \geq 0 \}$$

Compute the WLP using the Assignment Axiom…

$$\{P\}\ x := e\ \{Q\}[e/x]$$

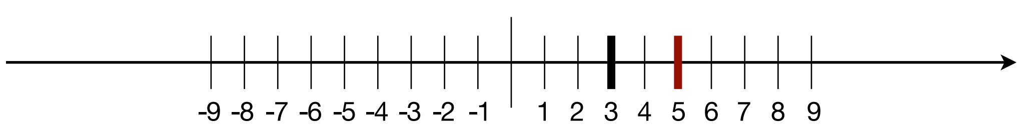
? $$\{x \geq 0\}\ [x - 3 / x]$$

$$x - 3 \geq 0$$

$$x \geq 3$$

On the one hand, $x \geq 5 \neq x \geq 3$ so it seems bad.

On the other hand, if $x \geq 5$ then it's also true that $x \geq 3$.



In other words, $x \geq 5$ is a stronger **pre**condition than $x \geq 3$.

Stronger Preconditions

Is this valid?

$$\{ x \geq 5 \}\ \ x := x - 3\ \ \{ x \geq 0 \}$$

Compute the WLP using the Assignment Axiom…

$$\{P\}\ x := e\ \{Q\}[e/x]$$

Good!

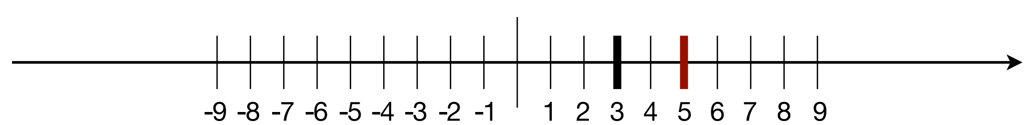
$$\{x \geq 0\}\ [x - 3 / x]$$

$$x - 3 \geq 0$$

$$x \geq 3$$

On the one hand, $x \geq 5 \neq x \geq 3$ so it seems bad.

On the other hand, if $x \geq 5$ then it's also true that $x \geq 3$.



In other words, $x \geq 5$ is a stronger **pre**condition than $x \geq 3$.
It's valid. We are allowed to **strengthen** preconditions, to have given preconditions that are stronger than the WLP.

Rule of Consequence

The (inference) Rule of Consequence allows us to have . . .

Stronger Preconditions

$$\frac{P \Rightarrow P',\ \{P\}\ S\ \{Q\}}{\{P'\}\ S\ \{Q\}}$$

$\{x \geq 5\}\ x := x - 3\ \{x \geq 0\}$

WLP is $x \geq 3$

The given precondition (P) $x \geq 5$ is a stronger **pre**condition
than the computed WLP (P′) $x \geq 3$
because $x \geq 5 \Rightarrow x \geq 3$

Weaker Postconditions

What about this?

$$\{ x \geq 3 \}\ x := x - 3\ \{ x \geq \mathbf{-1} \}$$

We can use the Assignment Axiom **backwards** *

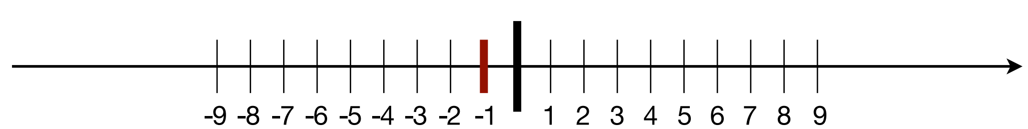
$$\mathbf{[x/e]\{P\}}\ \ x := e\ \{Q\}$$

to compute the postcondition:

$$[x \,/\, x - 3]\{x \geq 3\}$$
$$[x \,/\, x - 3]\{x-3 \geq 0\}$$
$$x \geq 0$$

?

Again, $x \geq -1 \neq x \geq 0$… but it's true that if $x \geq 0$ then $x \geq -1$.



In other words, $x \geq -1$ is a weaker **post**condition than $x \geq 0$.

* This is an advanced technique. Do not attempt this while under the influence of mind-altering substances.

Weaker Postconditions

What about this?

$$\{ x \geq 3 \}\ x := x - 3\ \{ x \geq \mathbf{-1} \}$$

We can use the Assignment Axiom **backwards**

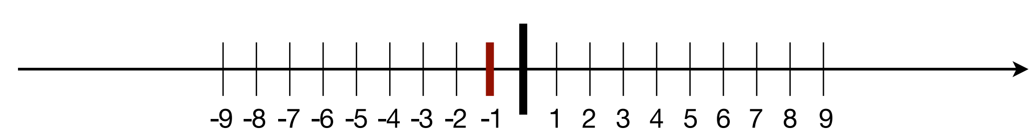
$$\mathbf{[x/e]\{P\}}\ \ x := e\ \{Q\}$$

to compute the postcondition:

$$[x / x - 3]\{x \geq 3\}$$
$$[x / x - 3]\{x\text{-}3 \geq 0\}$$
$$x \geq 0$$

!

Again, $x \geq -1 \neq x \geq 0$… but it’s true that if $x \geq 0$ then $x \geq -1$.



In other words, $x \geq -1$ is a weaker **post**condition than $x \geq 0$.
It’s valid. We are allowed to **weaken** postconditions, to have given postconditions that are weaker than the computed one.

Rule of Consequence

The (inference) Rule of Consequence allows us to have . . .

Stronger Preconditions

$$\frac{P \Rightarrow P',\ \{P\}\ S\ \{Q\}}{\{P'\}\ S\ \{Q\}}$$

and

Weaker Postconditions

$$\frac{\{P\}\ S\ \{Q\},\ Q' \Rightarrow Q}{\{P\}\ S\ \{Q'\}}$$

$\{ x \geq 3 \}\ \ x := x - 3\ \ \{x \geq -1\}$

WLP is $x \geq 0$

The given postcondition (Q) $x \geq -1$ is a weaker **post**condition
than the computed one (Q′) $x \geq 0$
because $x \geq 0 \Rightarrow x \geq -1$

Rule of Consequence

The (inference) Rule of Consequence allows us to have . . .

Stronger Preconditions

$$\frac{P \Rightarrow P',\ \{P\}\ S\ \{Q\}}{\{P'\}\ S\ \{Q\}}$$

and

Weaker Postconditions

$$\frac{\{P\}\ S\ \{Q\},\ Q' \Rightarrow Q}{\{P\}\ S\ \{Q'\}}$$

at

the same time

$$\frac{P \Rightarrow P',\ \{P\}\ S\ \{Q\},\ Q' \Rightarrow Q}{\{P'\}\ S\ \{Q'\}}$$

$$\{\, x \geq 3 \,\}\ x := x - 3\ \{x \geq 0\}$$

$$\Uparrow \quad x \geq 5 \Rightarrow x \geq 0 \quad \Downarrow$$

$$x \geq 0 \Rightarrow x \geq -1$$

$$\{\, x \geq 5 \,\}\ x := x - 3\ \{x \geq -1\}$$

Reasoning about Small Programs

Keeping in mind our ability to strengthen preconditions and weaken postconditions, we can use weakest liberal preconditions to prove programs correct.

How?

Reasoning about Small Programs

Keeping in mind our ability to strengthen preconditions and weaken postconditions, we can use weakest liberal preconditions to prove programs correct.

How? By chaining them together.

Consider a program as a **sequence** of statements S_1, S_2, S_3, ... S_n **composed** together.

For each statement S_i in the program, we take its postcondition (Q) and compute its weakest liberal precondition (WLP). If the computed WLP matches the precondition for S_i for then S_i is proved partially correct.

S_i : if WPL(Q) = P then S_i is partially correct.

(Of course, we'll take into account that we can strengthen preconditions and weaken postconditions thanks to the Rule of Consequence.)

Reasoning about Small Programs by Chaining WLPs

Let's consider a two-statement program:

{P1} S1 {Q1}

{P2} S2 {Q2}

P1 is the starting state.
Q2 is the finishing state.

What is the intermediate state?

Reasoning about Small Programs by Chaining WLPs

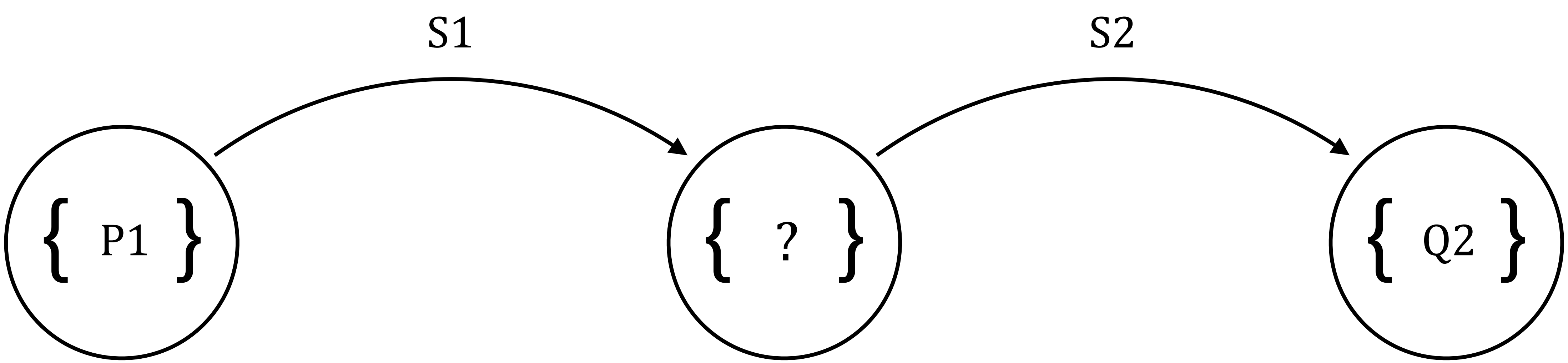
Let's consider a two-statement program:

$$\{P1\}\ S1\ \{Q1\}$$

$$\{P2\}\ S2\ \{Q2\}$$

P1 is the starting state.
Q2 is the finishing state.

What is the intermediate state?



Reasoning about Small Programs by Chaining WLPs

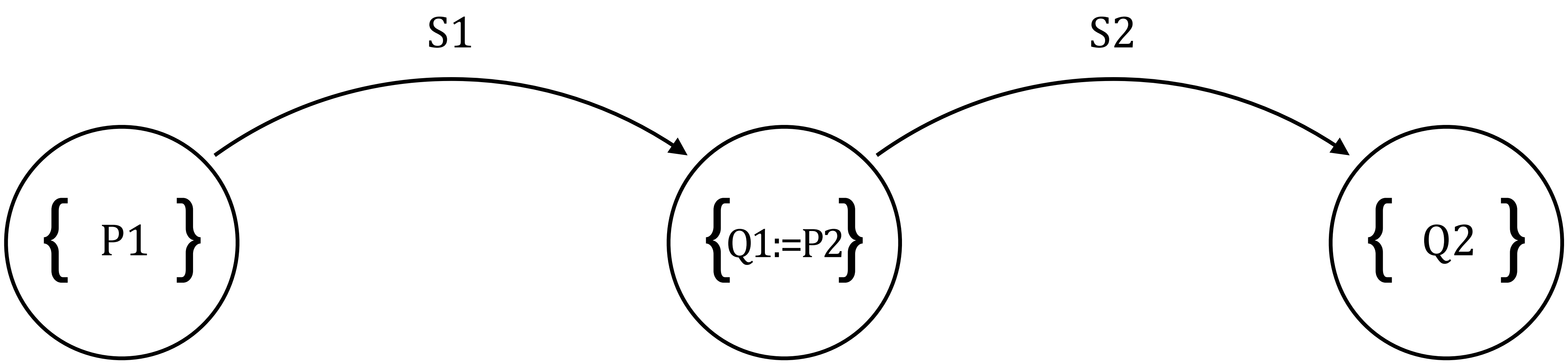
Let's consider a two-statement program:

{P1} S1 {Q1}

{P2} S2 {Q2}

P1 is the starting state.
Q2 is the finishing state.

The intermediate state is P2 = Q1.



Sequential Composition

Formalizing our two-statement program…

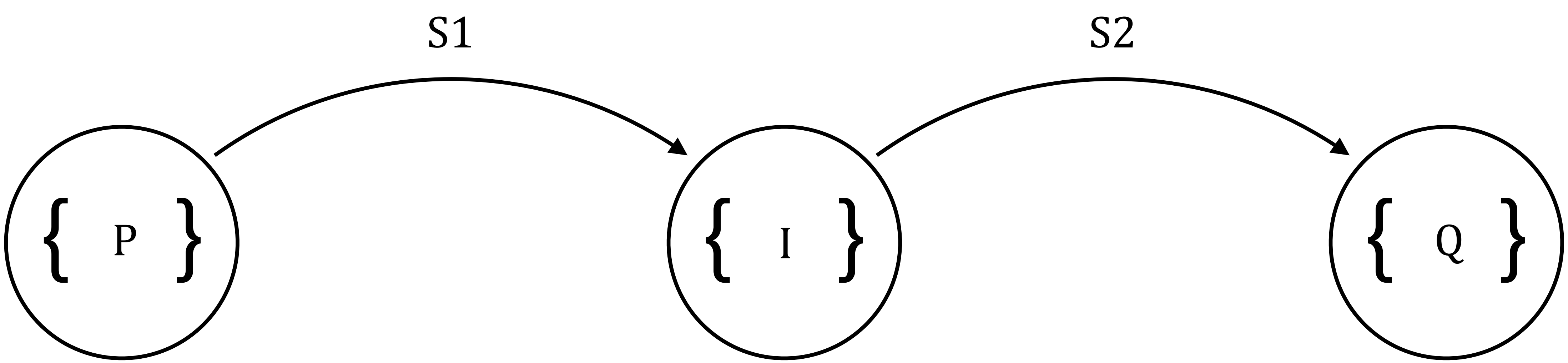
{P} S1 {I}

{I} S2 {Q}

… leads to the Sequential Composition rule

- [x] Sequence
- [ ] Alternation
- [ ] Repetition

$$\frac{\{P\}\ S1\ \{I\},\ \{I\}\ S2\ \{Q\}}{\{P\}\ S1;\ S2\ \{Q\}}$$



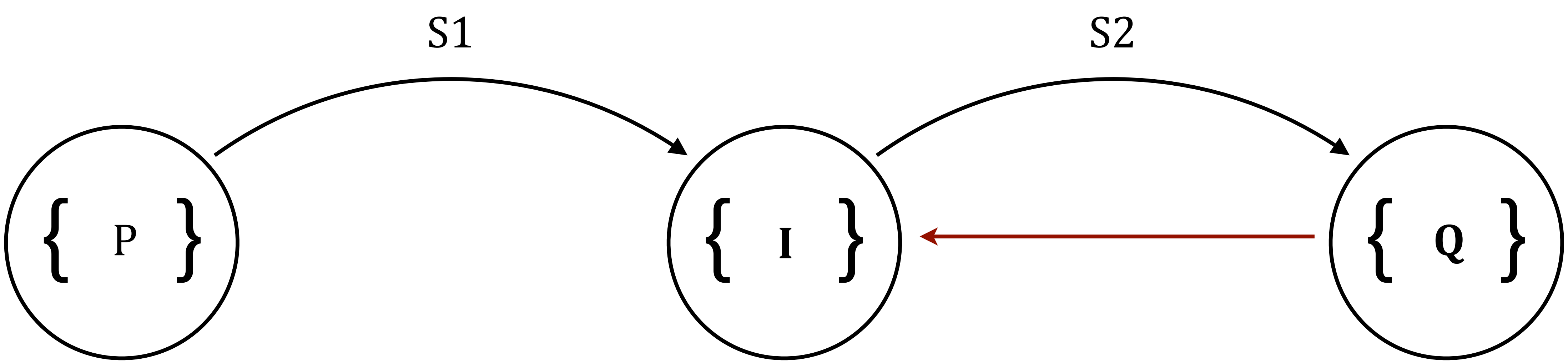
Sequential Composition

We can chain WLPs from the **bottom-up** to reason about the correctness of small programs.

Calculate the WLP of S2's postcondition to get S2's precondition.

- [x] Sequence
- [ ] Alternation
- [ ] Repetition

$$\frac{\{P\}\ S1\ \{I\},\ \{\mathbf{I}\}\ S2\ \{\mathbf{Q}\}}{\{P\}\ S1;\ S2\ \{Q\}}$$



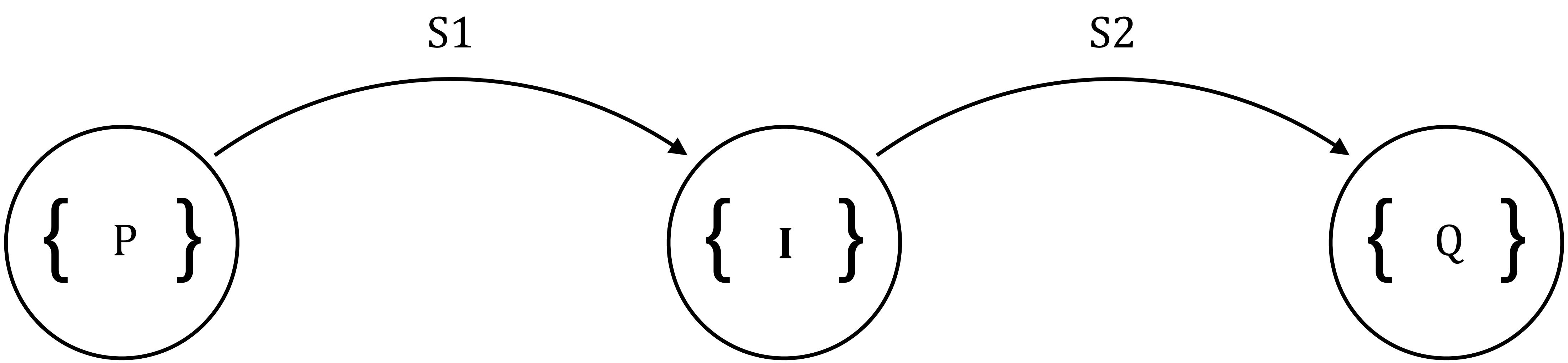
Sequential Composition

We can chain WLPs from the **bottom-up** to reason about the correctness of small programs.

Calculate the WLP of S2's postcondition to get S2's precondition.
Assign that (the intermediate state) to S1's postcondition.

- ☑ Sequence
- ☐ Alternation
- ☐ Repetition

$$\frac{\{P\}\ S1\ \{\mathbf{I}\},\ \{\mathbf{I}\}\ S2\ \{Q\}}{\{P\}\ S1;\ S2\ \{Q\}}$$



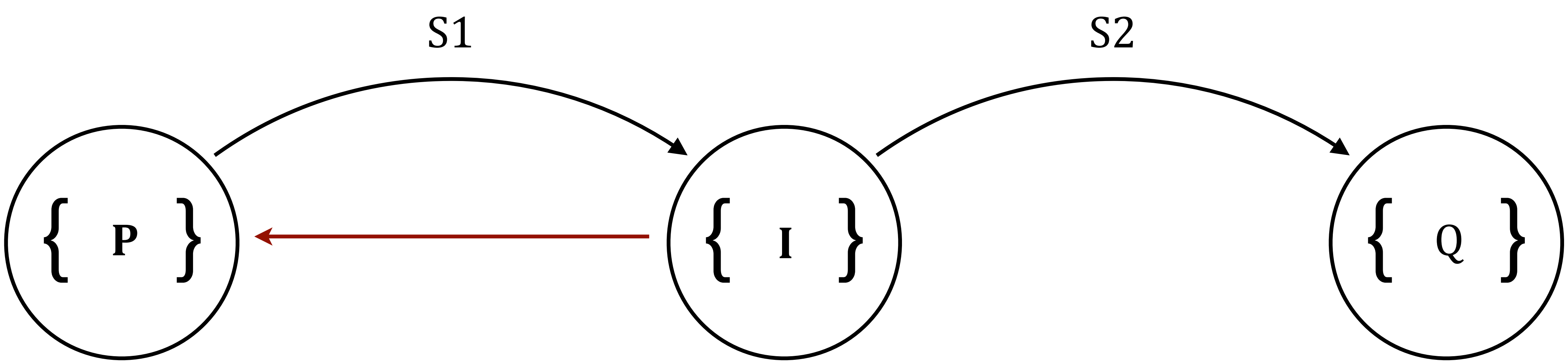
Sequential Composition

We can chain WLPs from the **bottom-up** to reason about the correctness of small programs.

Calculate the WLP of S2's postcondition to get S2's precondition.
Assign that (the intermediate state) to S1's postcondition.
Calculate the WLP of S1's postcondition to get S1's precondition.

- [x] Sequence
- [ ] Alternation
- [ ] Repetition

$$\frac{\{P\}\ S1\ \{I\},\ \{I\}\ S2\ \{Q\}}{\{P\}\ S1;\ S2\ \{Q\}}$$



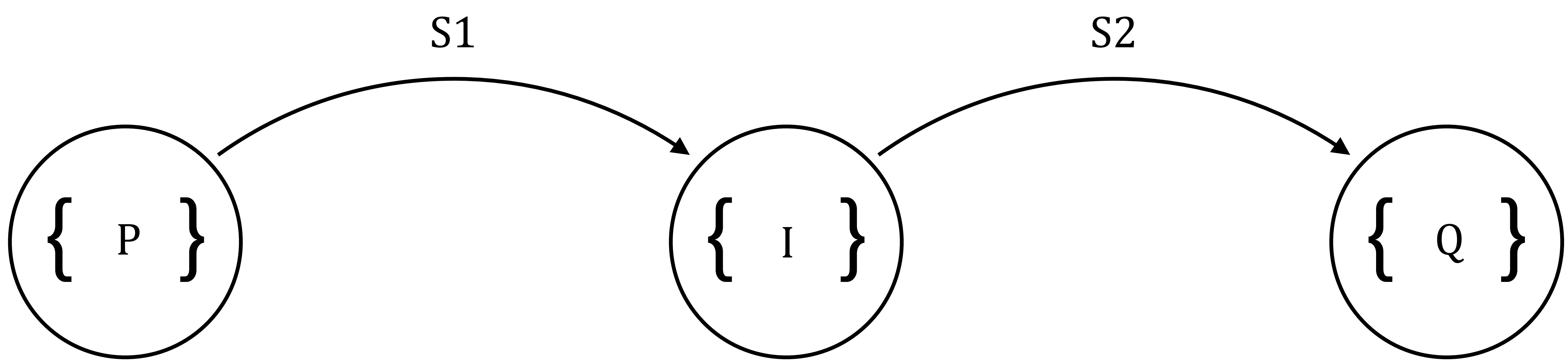
Sequential Composition

We can chain WLPs from the **bottom-up** to reason about the correctness of small programs.

Calculate the WLP of S2's postcondition to get S2's precondition.
Assign that (the intermediate state) to S1's postcondition.
Calculate the WLP of S1's postcondition to get S1's precondition.
Compare to what's given to judge correctness of the program.

- [x] Sequence
- [ ] Alternation
- [ ] Repetition

$$\frac{\{P\}\ S1\ \{I\},\ \{I\}\ S2\ \{Q\}}{\{P\}\ S1;\ S2\ \{Q\}}$$



Sequential Composition

Consider this program:

$$\{ x > y \}\ x := x - 1;\ y := y - 1\ \{ x > y \}$$

helpfully rewritten as:

$$\begin{array}{ccc} \{P\} & S1 & \{I\} \\ \{ x > y \} & x := x - 1 & \{ WLP \} \end{array}$$

$$\begin{array}{ccc} \{ WLP \} & y := y - 1 & \{ x > y \} \\ \{I\} & S2 & \{Q\} \end{array}$$

Sequential Composition

Consider this program:

$$\{ x > y \}\ x := x - 1;\ y := y - 1\ \{ x > y \}$$

helpfully rewritten as:

$$\begin{array}{ccc} \{P\} & S1 & \{I\} \\ \{ x > y \} & x := x - 1 & \{ WLP \} \\ \{ WLP \} & \boxed{y := y - 1\ \{ x > y \}} & \\ \{I\} & S2 & \{Q\} \end{array}$$

Start at the end and work back to the beginning.

$$WLP = \{ x > y \}[y - 1 / y]$$

$$= \{ x > y - 1 \}$$

Sequential Composition

Consider this program:

$$\{ x > y \}\ x := x - 1;\ y := y - 1\ \{ x > y \}$$

helpfully rewritten as:

$$\begin{array}{ccc} \{P\} & S1 & \{I\} \\ \{ x > y \} & x := x - 1 & \{ WLP \} \\ \{ x > y - 1 \} & y := y - 1 & \{ x > y \} \\ \{I\} & S2 & \{Q\} \end{array}$$

$$\begin{aligned} WLP &= \{ x > y \}[y - 1 / y] \\ &= \{ x > y - 1 \} \end{aligned}$$

Sequential Composition

Consider this program:

$$\{ x > y \}\ x := x - 1;\ y := y - 1\ \{ x > y \}$$

helpfully rewritten as:

$$\begin{array}{ccc} \{P\} & S1 & \{I\} \\ \{ x > y \} & x := x - 1 & \{ x > y - 1 \} \\ \{ x > y - 1 \} & y := y - 1 & \{ x > y \} \\ \{I\} & S2 & \{Q\} \end{array}$$

Propagate the intermediate state up to the prior statement in sequence.

Sequential Composition

Consider this program:

$$\{ x > y \}\ x := x - 1;\ y := y - 1\ \{ x > y \}$$

helpfully rewritten as:

$$\begin{array}{ccc} \{P\} & S1 & \{I\} \\ \{ x > y \} & \boxed{x := x - 1\ \{ x > y - 1 \}} & \end{array}$$

$$\begin{array}{ccc} \{ x > y - 1 \} & y := y - 1 & \{ x > y \} \\ \{I\} & S2 & \{Q\} \end{array}$$

$$\{P\} = \{ x > y - 1\}[x - 1 / x]$$

$$= \{ x - 1 > y - 1 \}$$

Sequential Composition

Consider this program:

$$\{ x > y \}\ x := x - 1;\ y := y - 1\ \{ x > y \}$$

helpfully rewritten as:

$$\begin{array}{ccc} \{P\} & S1 & \{I\} \\ \{ x - 1 > y - 1 \} & x := x - 1 & \{ x > y - 1 \} \\ \{ x > y - 1 \} & y := y - 1 & \{ x > y \} \\ \{I\} & S2 & \{Q\} \end{array}$$

$$\{P\} = \{ x > y - 1\}[x - 1 / x]$$
$$= \{ x - 1 > y - 1 \}$$

Sequential Composition

Consider this program:

$$\boxed{\{\, x > y \,\}}\ x := x - 1;\ y := y - 1\ \ \{\, x > y \,\}$$

helpfully rewritten as:

$$\underset{}{\overset{\{P\}}{\boxed{\{\, x - 1 > y - 1 \,\}}}}\ \overset{S1}{x := x - 1}\ \overset{\{I\}}{\{\, x > y - 1 \,\}}$$

$$\underset{\{I\}}{\{\, x > y - 1 \,\}}\ \underset{S2}{y := y - 1}\ \underset{\{Q\}}{\{\, x > y \,\}}$$

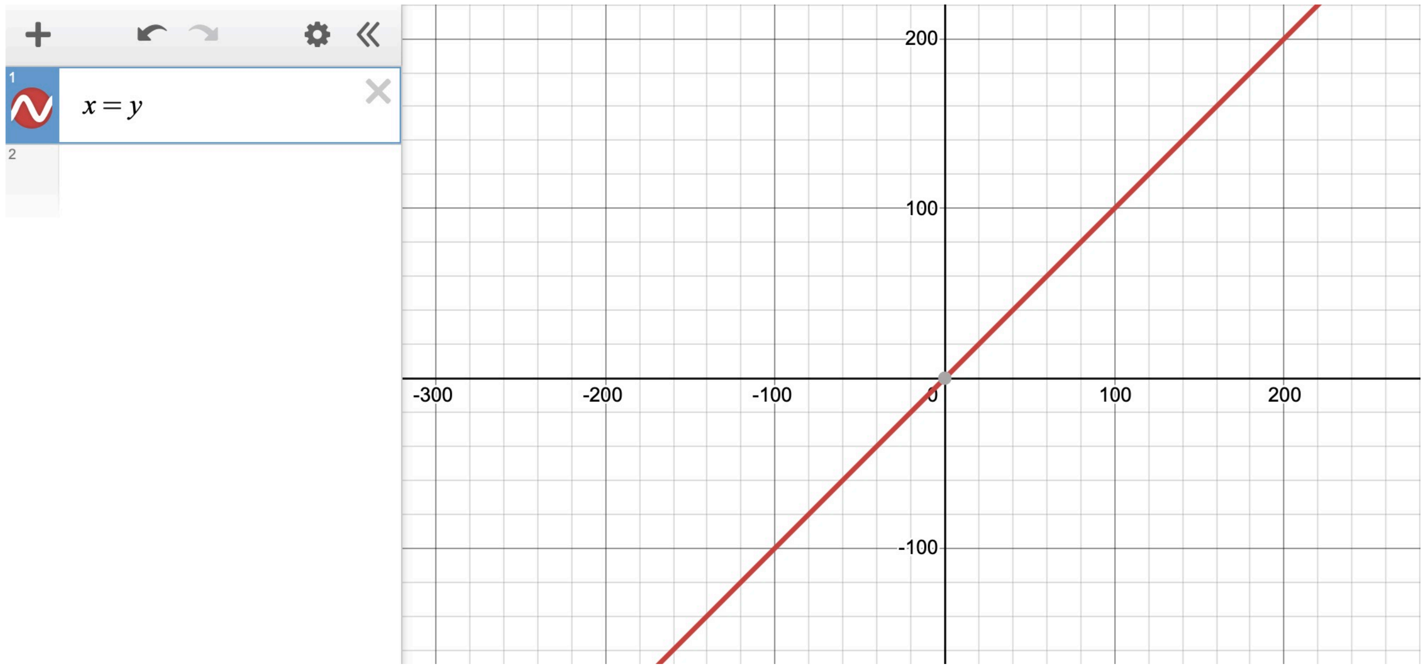
$$\{P\} = \{\, x > y - 1\}[x - 1 \,/\, x]$$

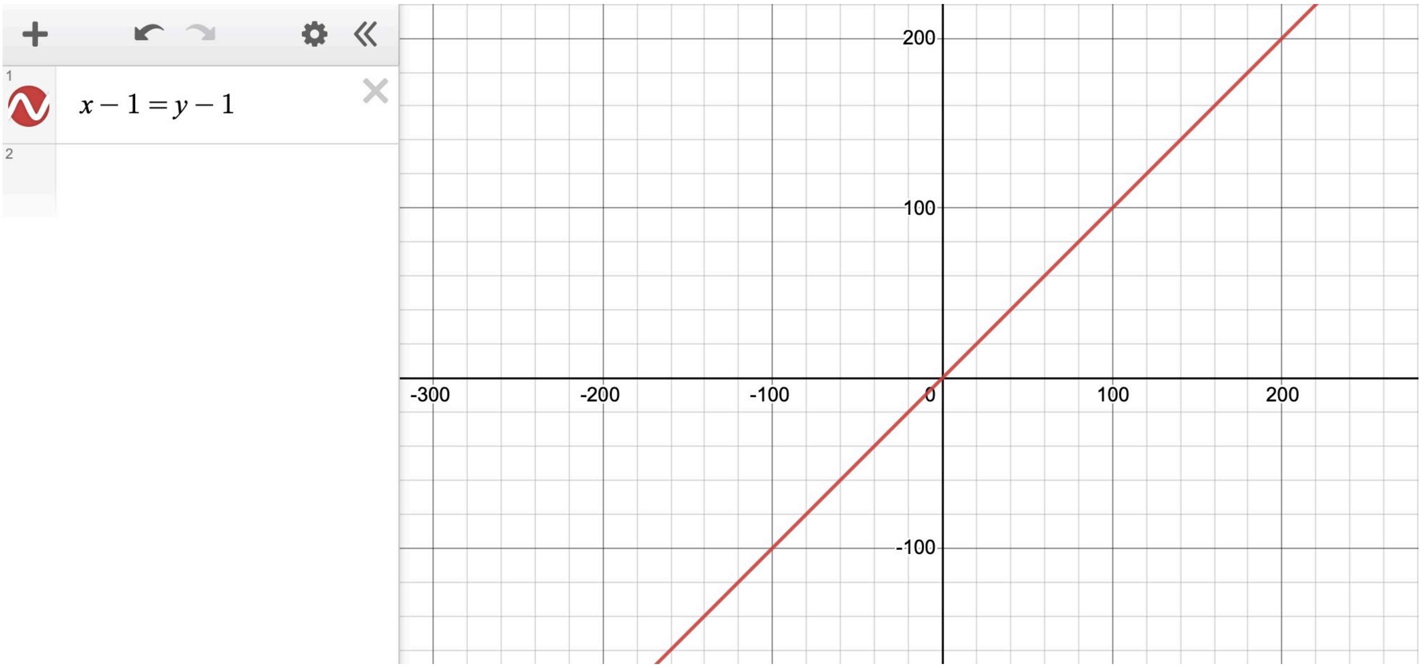
$$= \{\, x - 1 > y - 1 \,\}$$

Compare computed WLP to original {P}: $x > y \stackrel{?}{=} x\text{-}1 > y\text{-}1$

Sequential Composition

Wait, can we show (x > y) = (x-1 > y-1) ? Yes.





Also:

x-1 > y-1

x-1+1 > y-1+1

x > y

Sequential Composition

Consider this program:

$$\boxed{\{\ x > y\ \}}\ x := x - 1;\ y := y - 1\ \ \{\ x > y\ \}$$

helpfully rewritten as:

$$\begin{array}{ccc} \{P\} & S1 & \{I\} \\ \boxed{\{\ x - 1 > y - 1\ \}} & x := x - 1 & \{\ x > y - 1\ \} \\ \{\ x > y - 1\ \} & y := y - 1 & \{\ x > y\ \} \\ \{I\} & S2 & \{Q\} \end{array}$$

$$\{P\} = \{\ x > y - 1\}[x - 1 \,/\, x]$$
$$= \{\ x - 1 > y - 1\ \}$$

Compare computed WLP to original {P}: $x > y = x\text{-}1 > y\text{-}1$ so we have proved this sequence of statements (program) partially correct.

Sequential Composition

What is the WLP {P} that will make this program correct?

$$\{P\}\ \ y = 3 \cdot x + 1;\ x := y + 3\ \ \{x < 10\}$$

Rewrite as…

$$\{P\}\ \ y = 3 \cdot x + 1\ \{I\}$$

$$\{I\}\ x := y + 3\ \ \{\ x < 10\ \}$$

… and solve from the bottom-up.

Sequential Composition

What is the WLP {P} that will make this program correct?

$$\{P\}\ \ y = 3 \cdot x + 1;\ x := y + 3\ \ \{x < 10\}$$

Rewrite as…

$$\{P\}\ \ y = 3 \cdot x + 1\ \{I\}$$

$$\{I\}\ x := y + 3\ \ \{\ x < 10\ \}$$

… and solve from the bottom-up.

$$I = \ \{\ x < 10\ \}[y + 3\ /\ x]$$

$$y + 3 < 10$$

$$y < 7$$

Sequential Composition

What is the WLP {P} that will make this program correct?

$$\{P\}\ \ y = 3 \cdot x + 1;\ x := y + 3\ \ \{x < 10\}$$

Rewrite as…

$$\{P\}\ \ y = 3 \cdot x + 1\ \{\mathbf{y < 7}\}$$

$$\{\mathbf{y < 7}\}\ x := y + 3\ \ \{\ x < 10\ \}$$

… and solve from the bottom-up.

Sequential Composition

What is the WLP {P} that will make this program correct?

$$\{P\}\ \ y = 3 \cdot x + 1;\ x := y + 3\ \ \{x < 10\}$$

Rewrite as…

$$\{P\}\ \ y = 3 \cdot x + 1\ \{y < 7\}$$

$$\{y < 7\}\ x := y + 3\ \ \{\ x < 10\ \}$$

… and solve from the bottom-up.

$$P = \ \{\ y < 7\ \}[3 \cdot x + 1\ /\ y]$$

$$3 \cdot x + 1 < 7$$

$$3 \cdot x < 6$$

$$x < 2$$

Sequential Composition

What is the WLP {P} that will make this program correct?

$$\{P\}\ \ y = 3 \cdot x + 1;\ x := y + 3\ \ \{x < 10\}$$

Rewrite as…

$$\{\mathbf{x < 2}\}\ \ y = 3 \cdot x + 1\ \{y < 7\}$$

$$\{y < 7\}\ x := y + 3\ \ \{\ x < 10\ \}$$

$x < 2$ is the WLP {P} that makes this program partially correct.

$$\{x < 2\}\ \ y = 3 \cdot x + 1;\ x := y + 3\ \ \{x < 10\}$$

Sequential Composition

Is this program (partially) correct?

$$\{B > 6\}\ J = 2 \cdot B + 007\ ;\ B := J - 17\ \{ B > 5 \}$$

Rewrite:

$$\{B > 6\}\ J = 2 \cdot B + 007\ \{I\}$$

$$\{I\}\ B := J - 17\ \{ B > 5 \}$$

Sequential Composition

Is this program (partially) correct?

$$\{B > 6\}\ J = 2 \cdot B + 007\ ;\ B := J - 17\ \{ B > 5 \}$$

Rewrite:

$$\{B > 6\}\ J = 2 \cdot B + 007\ \{I\}$$

$$\{I\}\ B := J - 17\ \{ B > 5 \}$$

$$\{B > 5\}[J - 17 / B\}$$

$$J - 17 > 5$$

$$J > 22$$

Sequential Composition

Is this program (partially) correct?

$$\{B > 6\}\ \ J = 2 \cdot B + 007\ ;\ \ B\ := J - 17\ \ \{ B > 5 \}$$

Rewrite:

$$\{B > 6\}\ \ J = 2 \cdot B + 007\ \ \{\mathbf{J > 22}\}$$

$$\{\mathbf{J > 22}\}\ B\ := J - 17\ \ \{ B > 5 \}$$

Sequential Composition

Is this program (partially) correct?

$$\{B > 6\}\ J = 2 \cdot B + 007\ ;\ B := J - 17\ \{ B > 5 \}$$

Rewrite:

$$\{B > 6\}\ J = 2 \cdot B + 007\ \{J > 22\}$$

$$\{J > 22\}\ B := J - 17\ \{ B > 5 \}$$

$$\{J > 22\}[2 \cdot B + 007 / J\}$$

$$2 \cdot B + 007 > 22$$

$$2 \cdot B > 15$$

$$B > 7.5$$

Sequential Composition

Is this program (partially) correct?

$$\boxed{\{B > 6\}}\ J = 2 \cdot B + 007\ ;\ \ B\ := J - 17\ \ \{ B > 5 \}$$

Rewrite:

$$\boxed{\{B > 7.5\}}\ J = 2 \cdot B + 007\ \ \{J > 22\}$$

$$\{J > 22\}\ B\ := J - 17\ \ \{ B > 5 \}$$

We calculated the WLP for the entire program to be B > 7.5
The given initial precondition is B > 6.

So is this program (partially) correct or not?

Sequential Composition

Is this program (partially) correct?

$$\boxed{\{B > 6\}}\ J = 2 \cdot B + 007\ ;\ \ B\ := J - 17\ \ \{\ B > 5\ \}$$

Rewrite:

$$\boxed{\{B > 7.5\}}\ J = 2 \cdot B + 007\ \ \{J > 22\}$$

$$\{J > 22\}\ B\ := J - 17\ \ \{\ B > 5\ \}$$

We calculated the WLP for the entire program to be B > 7.5
The given initial precondition is B > 6.

So is this program (partially) correct or not?
Is the given precondition stronger than the WLP?

Sequential Composition

Is this program (partially) correct?

$$\boxed{\{B > 6\}}\ J = 2 \cdot B + 007\ ;\ \ B\ := J - 17\ \ \{ B > 5 \}$$

Rewrite:

$$\boxed{\{B > 7.5\}}\ J = 2 \cdot B + 007\ \ \{J > 22\}$$

$$\{J > 22\}\ B\ := J - 17\ \ \{ B > 5 \}$$

We calculated the WLP for the entire program to be $B > 7.5$
The given initial precondition is $B > 6$.

So is this program (partially) correct or not? **NO**!
Is the given precondition stronger than the WLP? **NO**!
$B > 6 \not\Rightarrow B > 7.5$
and we are not allowed to weaken preconditions.

Don't believe me? Try it with B=7.

Sequential Composition

This works for larger programs as well:

$\{x < 1\}$
$y = 2x + 2;$
$x = 4y - 2;$
$x = x + 1;$
$\{x < 23\}$

Try it!

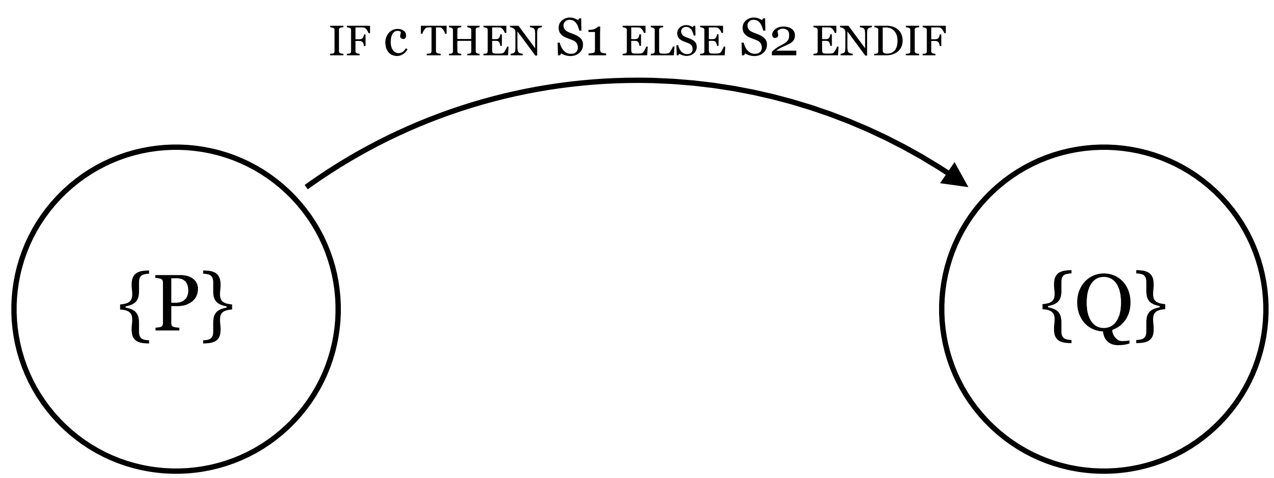
Conditional Rule

We've got sequence.
Now we need alternation.

- [x] Sequence
- [ ] Alternation
- [ ] Repetition

For that, there's the Conditional Rule.

Think of it like this:



Some states in P satisfy c, causing the IF block (S1) to execute, while other states in P do not satisfy c, causing ELSE block (S2) to execute.

Here's another way to visualize it.

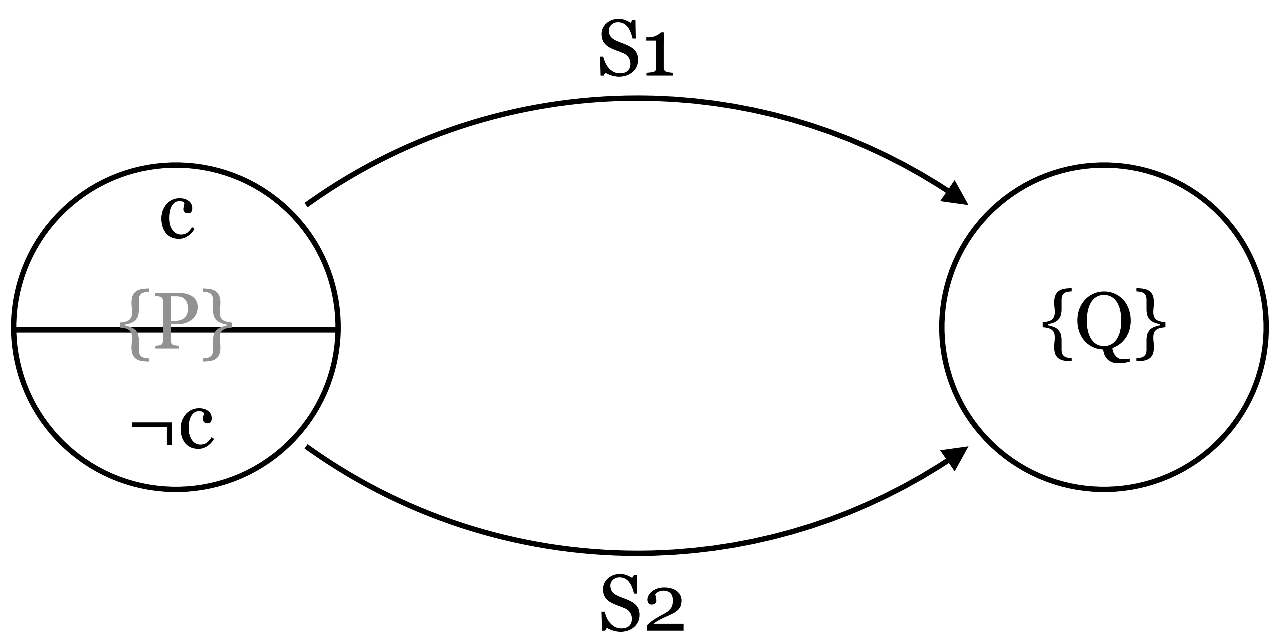
Conditional Rule

We've got sequence.
Now we need alternation.

For that, there's the Conditional Rule.

- ☑ Sequence
- ☐ Alternation
- ☐ Repetition

Think of it like IF c THEN S1 ELSE S2 ENDIF



Conditional Rule

We’ve got sequence.
Now we need alternation.

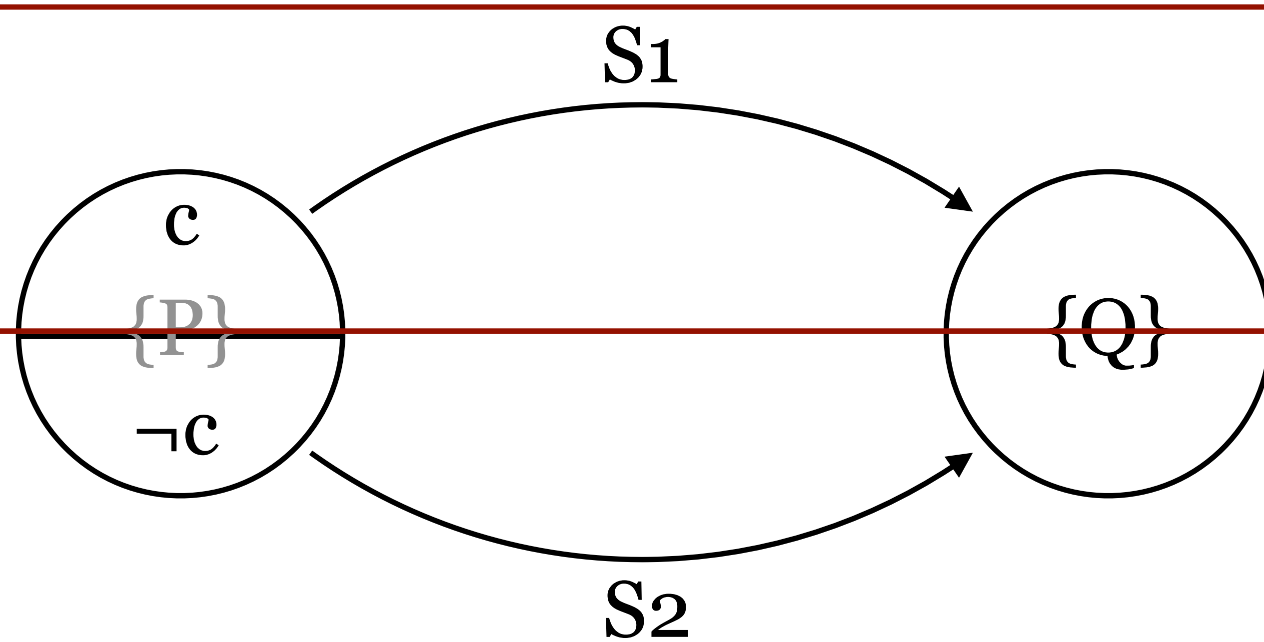
For that, there’s the Conditional Rule.

- ☑ Sequence
- ☐ Alternation
- ☐ Repetition

Think of it like $\boxed{\text{IF c THEN S1}}$ ELSE S2 ENDIF

$\{P \wedge c\}\ S1\ \{Q\}$

$\{P \wedge \neg c\}\ S2\ \{Q\}$



Conditional Rule

We've got sequence.
Now we need alternation.

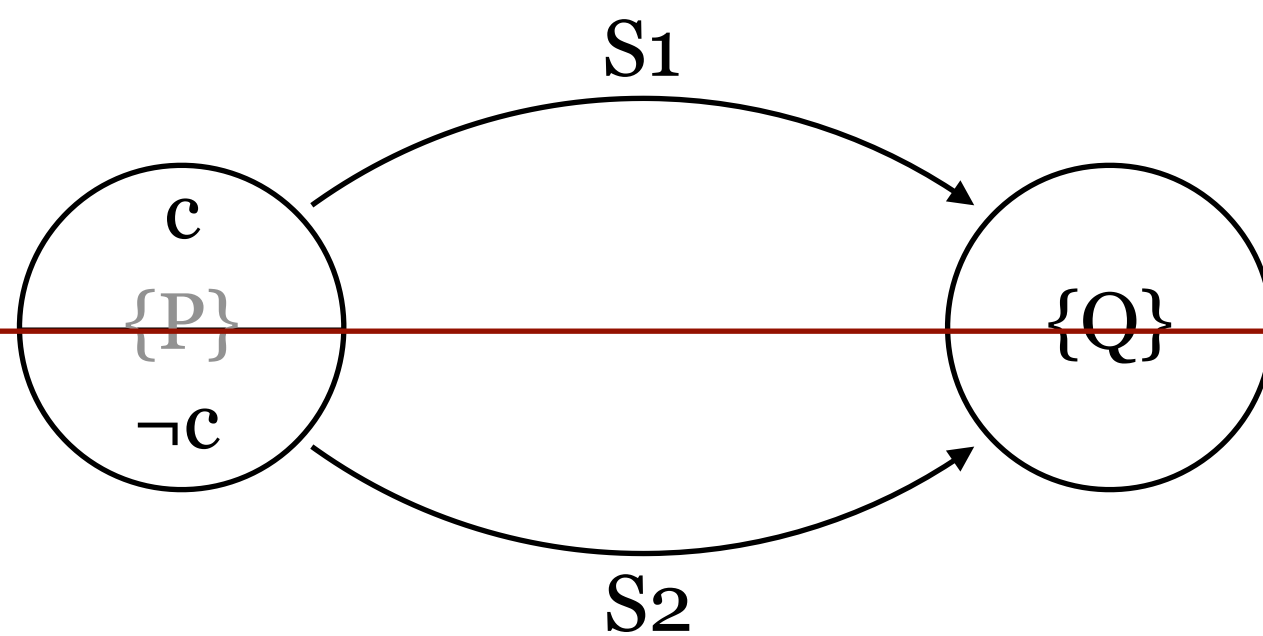
- [x] Sequence
- [ ] Alternation
- [ ] Repetition

For that, there's the Conditional Rule.

Think of it like IF c THEN S1 $\boxed{\text{ELSE S2 ENDIF}}$

$\{P \wedge c\}$ S1 $\{Q\}$

$\{P \wedge \neg c\}$ S2 $\{Q\}$



Conditional Rule

We've got sequence.
Now we need alternation.

For that, there's the Conditional Rule.

- ☑ Sequence
- ☑ Alternation
- ☐ Repetition

$$\frac{\{P \wedge c\}\ S1\ \{Q\},\ \{P \wedge \neg c\}\ S2\ \{Q\}}{\{P\}\ \text{IF } c \text{ THEN } S1 \text{ ELSE } S2 \text{ ENDIF}\ \{Q\}}$$

Conditional Rule

$$\frac{\{P \wedge c\}\ S1\ \{Q\},\ \{P \wedge \neg c\}\ S2\ \{Q\}}{\{P\}\ \text{IF}\ c\ \text{THEN}\ S1\ \text{ELSE}\ S2\ \text{ENDIF}\ \{Q\}}$$

{true} IF $y \leq 0$ THEN $x:=1$ ELSE $x := y$ ENDIF $\{x > 0\}$

{P} {c} {S1} {S2} {Q}

Conditional Rule

$$\frac{\{P \wedge c\}\ S1\ \{Q\},\ \{P \wedge \neg c\}\ S2\ \{Q\}}{\{P\}\ \text{IF } c \text{ THEN } S1 \text{ ELSE } S2 \text{ ENDIF } \{Q\}}$$

$\{true\}$ IF $y \leq 0$ THEN $x{:=}1$ ELSE $x := y$ ENDIF $\{x > 0\}$

$\{P\}$ $\{c\}$ $\{S1\}$ $\{S2\}$ $\{Q\}$

$\{P \wedge c\}\ S1\ \{Q\}$

$\{true \wedge y \leq 0\}\ x{:=}1\ \{x>0\}$

$\{x>0\}[1/x]$

$\{1>0\}$

$= \{true\}$

but we're looking for

$\{true \wedge y \leq 0\}$

Can we prove

$true \wedge y \leq 0 \Rightarrow true$?

Conditional Rule

$$\frac{\{P \wedge c\}\ S1\ \{Q\},\ \{P \wedge \neg c\}\ S2\ \{Q\}}{\{P\}\ \text{IF } c \text{ THEN } S1 \text{ ELSE } S2 \text{ ENDIF}\ \{Q\}}$$

$\{true\}$ IF $y \leq 0$ THEN $x:=1$ ELSE $x := y$ ENDIF $\{x > 0\}$

$\{P\}$ $\{c\}$ $\{S1\}$ $\{S2\}$ $\{Q\}$

$\{P \wedge c\}\ S1\ \{Q\}$

$\{true \wedge y \leq 0\}\ x:=1\ \{x>0\}$

$\{x>0\}[1/x]$

$\{1>0\}$

$= \{true\}$

but we're looking for

$\{true \wedge y \leq 0\}$

$\vdash true \wedge y \leq 0 \Rightarrow true$

True	(y≤0)	True ∧ (y≤0)	(True ∧ (y≤0))⇒ True
1	0	0	1
1	1	1	1

Conditional Rule

$$\frac{\{P \wedge c\}\ S1\ \{Q\},\ \{P \wedge \neg c\}\ S2\ \{Q\}}{\{P\}\ \text{IF}\ c\ \text{THEN}\ S1\ \text{ELSE}\ S2\ \text{ENDIF}\ \{Q\}}$$

$\{true\}$ IF $y \leq 0$ THEN $x:=1$ ELSE $x := y$ ENDIF $\{x > 0\}$

$\{P\}$ $\{c\}$ $\{S1\}$ $\{S2\}$ $\{Q\}$

$\{P \wedge c\}\ S1\ \{Q\}$

$\{true \wedge y \leq 0\}\ x:=1\ \{x>0\}$

$\{x>0\}[1/x]$

$\{1>0\}$

$= \{true\}$

$\vdash true \wedge y \leq 0 \Rightarrow true$

$\{true \wedge y \leq 0\}$

Conditional Rule

$$\frac{\{P \wedge c\}\ S1\ \{Q\},\ \{P \wedge \neg c\}\ S2\ \{Q\}}{\{P\}\ \text{IF } c \text{ THEN } S1 \text{ ELSE } S2 \text{ ENDIF}\ \{Q\}}$$

$\{true\}$ IF $y \leq 0$ THEN $x:=1$ ELSE $x := y$ ENDIF $\{x > 0\}$

$\{P\}$ $\{c\}$ $\{S1\}$ $\{S2\}$ $\{Q\}$

$\{P \wedge c\}\ S1\ \{Q\}$

$\{true \wedge y \leq 0\}\ x:=1\ \{x>0\}$

$\{x>0\}[1/x]$

$\{1>0\}$

$= \{true\}$

$\vdash true \wedge y \leq 0 \Rightarrow true$

$\{true \wedge y \leq 0\}$

$\{P \wedge \neg c\}\ S2\ \{Q\}$

$\{true \wedge \neg y \leq 0\}\ x:=y\ \{x>0\}$

$\{x>0\}[y/x]$

$\{y>0\}$

$= \{\neg y \leq 0\}$

$\vdash true \wedge \neg y \leq 0 \Rightarrow \neg y \leq 0$

$\{true \wedge \neg y \leq 0\}$

Conditional Rule

$$\frac{\{P \wedge c\}\ S1\ \{Q\},\ \{P \wedge \neg c\}\ S2\ \{Q\}}{\{P\}\ \text{IF}\ c\ \text{THEN}\ S1\ \text{ELSE}\ S2\ \text{ENDIF}\ \{Q\}}$$

$\{true\}$ IF $y \leq 0$ THEN $x{:=}1$ ELSE $x := y$ ENDIF $\{x > 0\}$

$\{P\}$ $\{c\}$ $\{S1\}$ $\{S2\}$ $\{Q\}$

$\{P \wedge c\}\ S1\ \{Q\}$

$\{true \wedge y \leq 0\}\ x{:=}1\ \{x>0\}$

$\{x>0\}[1/x]$

$\{1>0\}$

$= \{true\}$

$\vdash true \wedge y \leq 0 \Rightarrow true$

$\{true \wedge y \leq 0\}$

$\{P \wedge \neg c\}\ S2\ \{Q\}$

$\{true \wedge \neg y \leq 0\}\ x{:=}y\ \{x>0\}$

$\{x>0\}[y/x]$

$\{y>0\}$

$= \{\neg y \leq 0\}$

$\vdash true \wedge \neg y \leq 0 \Rightarrow \neg y \leq 0$

$\{true \wedge \neg y \leq 0\}$

Both paths are valid; the statement is valid.

Skip Rule

Sometimes you just need a NOP.

$$\{P\}\ \textsc{skip}\ \{P\}$$

While Loop Rule

With sequence and alternation done,
all that's left is repetition.

We need something like

{P} WHILE c DO S ENDWHILE {Q}

- ☑ Sequence
- ☑ Alternation
- ☐ Repetition

Note: If you're wondering about DO, FOR, and REPEAT loops, remember that we can rewrite any of those loops into a WHILE loop.

While Loop Rule

With sequence and alternation done,
all that's left is repetition.

We need something like

{P} WHILE c DO S ENDWHILE {Q}

☑ Sequence

☑ Alternation

☐ Repetition

The problem is that correctness proofs for arbitrary P and Q are undecidable in this context because there are an unknown number of traces through the code. (Do we enter the loop? How many times do we loop? Does the loop terminate?) We need more information about the construction of the loop. We need its **invariant**.

A loop invariant is an assertion that's . . .

- true immediately before the loop begins,
- true during the loop, and
- true immediately after the loop exits.

In other words, it does not vary, because it's (wait for it) **invariant**.

Loop Invariants

A loop invariant is an assertion that's . . .

- true immediately before the loop begins,
- true during the loop, and
- true immediately after the loop exits.

Example: Selection sort
What is the loop invariant ?

```
i := 0
while i < n
   i := i + 1
   find smallest item
   move to array[i]
endwhile
print i 'items sorted'
```

Loop Invariants

A loop invariant is an assertion that's . . .

- true immediately before the loop begins,
- true during the loop, and
- true immediately after the loop exits.

Example: Selection sort
What is the loop invariant ?

```
i := 0
while i < n
    i := i + 1
    find smallest item
    move to array[i]
endwhile
print i 'items sorted'
```

The invariant is i = the number of items in sorted order.

- Before the loop, 0 items are sorted.
- As we iterate repeatedly through the loop, the first i items are in sorted order.
- At the end of the loop, all n=i items are in sorted order.

While Loop Rule

While this won't do:

{P} WHILE c DO S ENDWHILE {Q}

this will:

$$\frac{\{I \wedge c\}\ S\ \{I\}}{\{I\}\ \text{WHILE c DO S ENDWHILE}\ \{I \wedge \neg c\}}$$

While Loop Rule

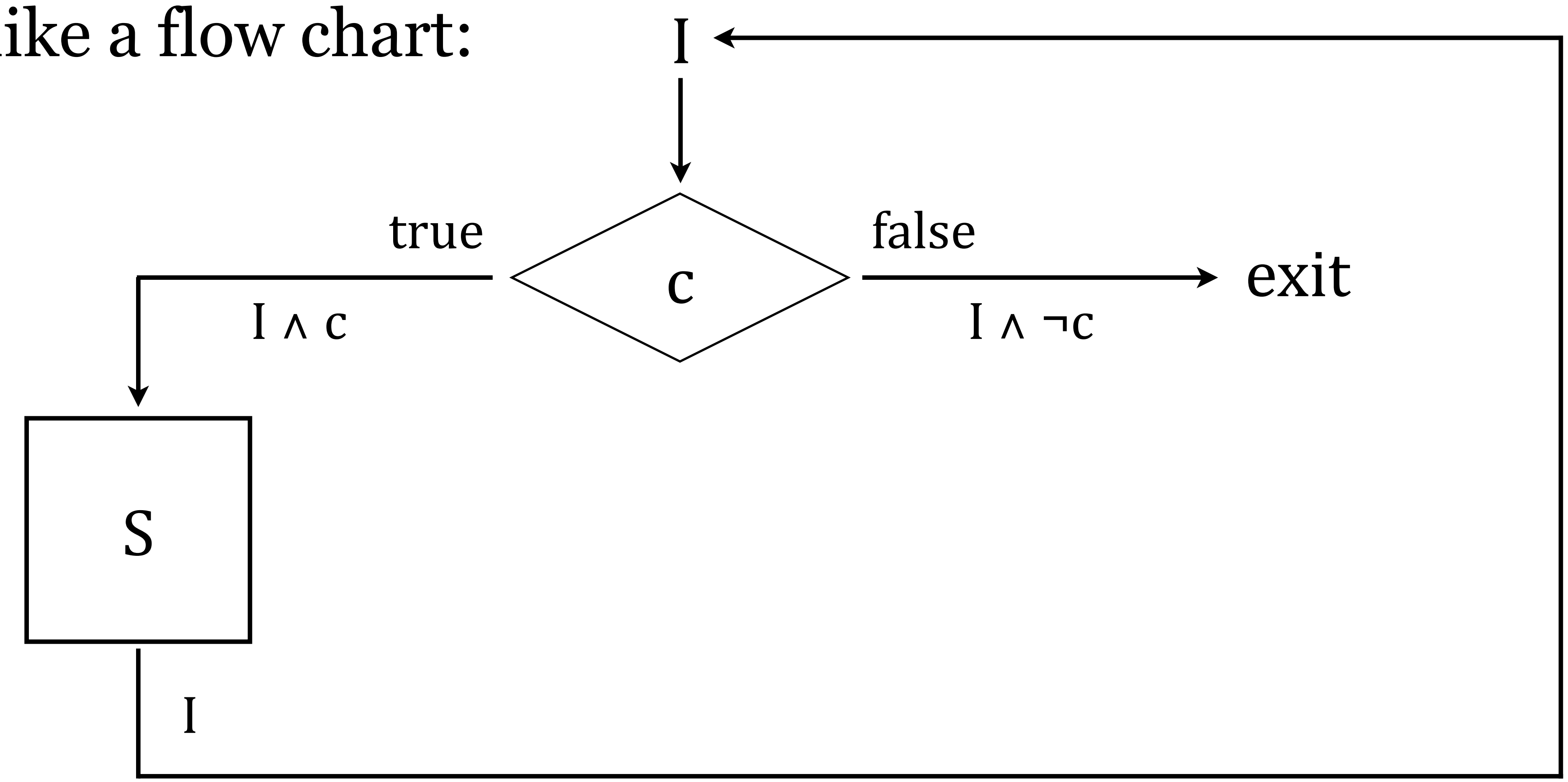
While this won't do:

{P} WHILE c DO S ENDWHILE {Q}

this will:

$$\frac{\{I \wedge c\}\ S\ \{I\}}{\{I\}\ \text{WHILE } c \text{ DO } S \text{ ENDWHILE } \{I \wedge \neg c\}}$$

Think of it like a flow chart:



While Loop Rule

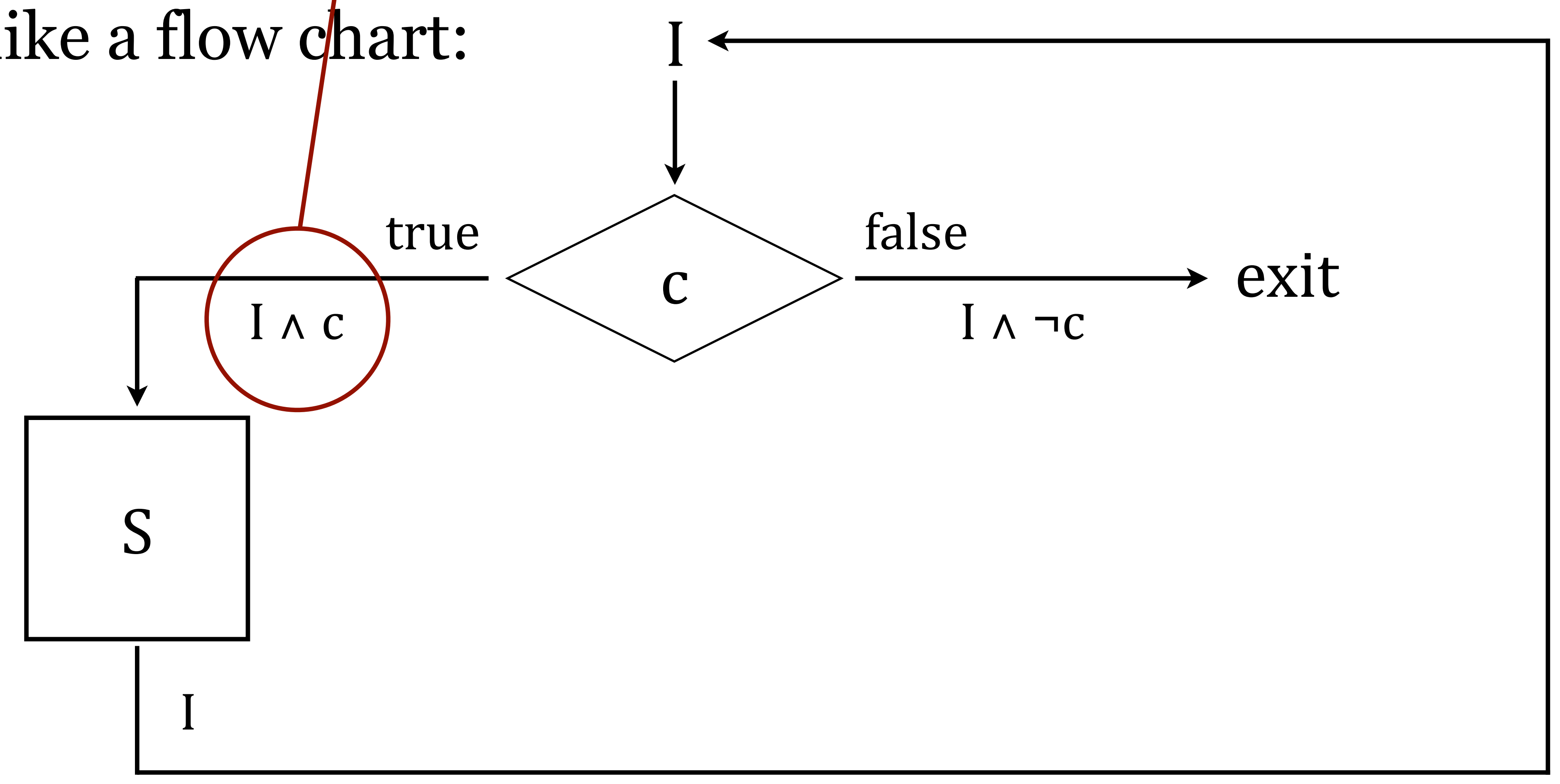
While this won't do:

{P} WHILE c DO S ENDWHILE {Q}

this will:

$$\frac{\{I \wedge c\}\ S\ \{I\}}{\{I\}\ \text{WHILE c DO S ENDWHILE}\ \{I \wedge \neg c\}}$$

Think of it like a flow chart:



While Loop Rule

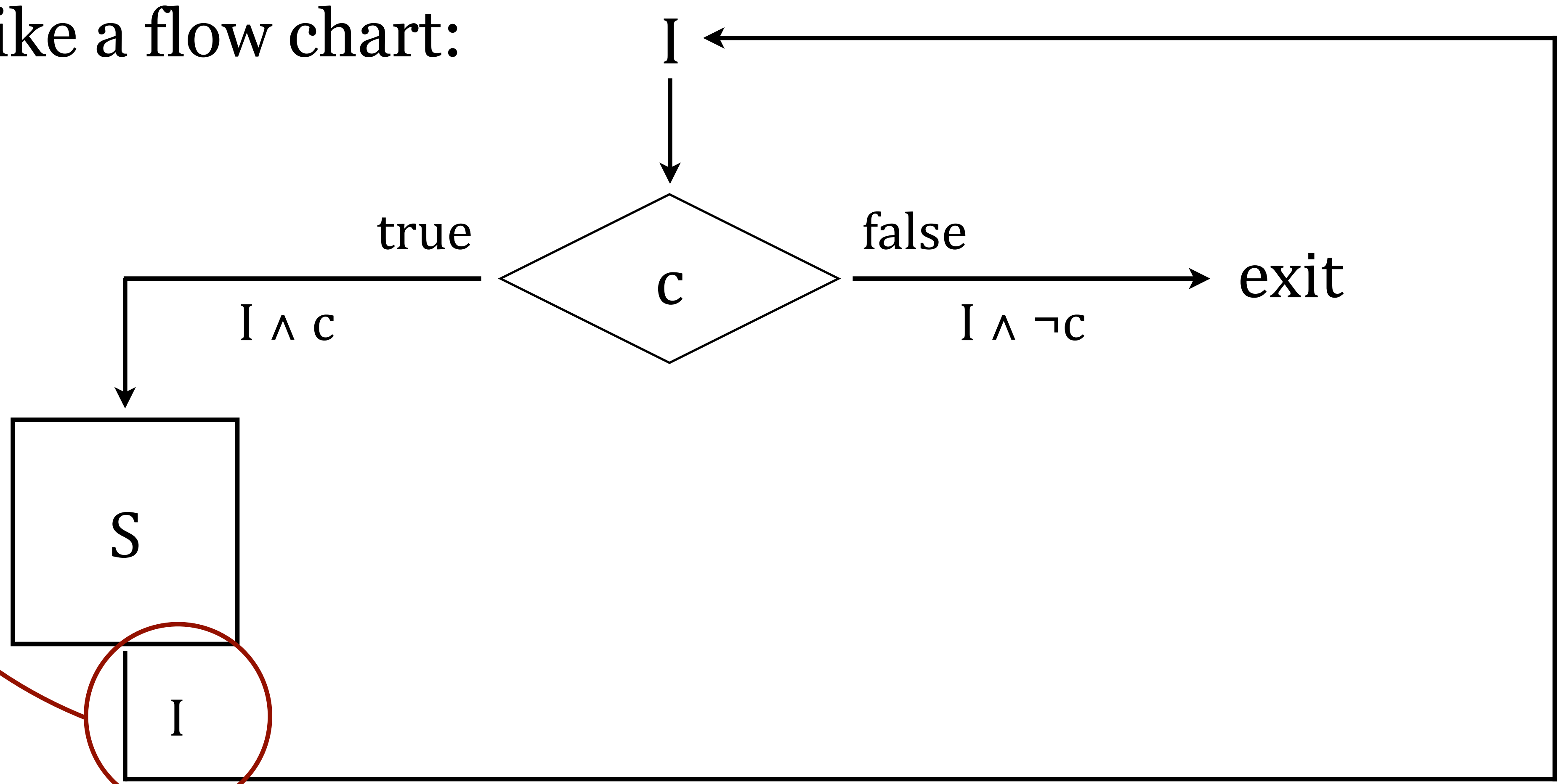
While this won't do:

{P} WHILE c DO S ENDWHILE {Q}

this will:

$$\frac{\{I \land c\}\ S\ \{I\}}{\{I\}\ \text{WHILE } c \text{ DO } S \text{ ENDWHILE}\ \{I \land \neg c\}}$$

Think of it like a flow chart:



While Loop Rule

While this won't do:

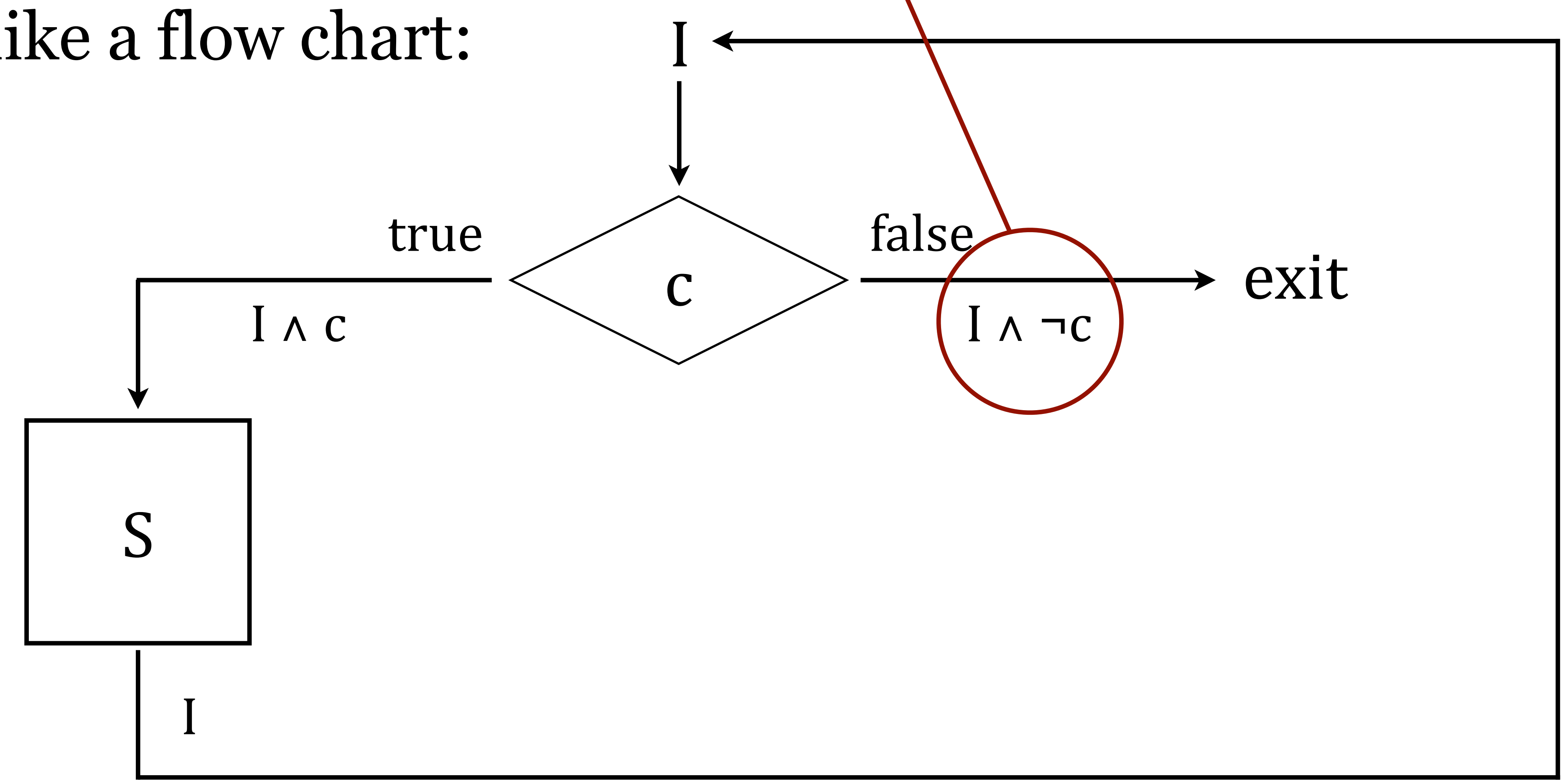
{P} WHILE c DO S ENDWHILE {Q}

this will:

$$\frac{\{I \wedge c\}\ S\ \{I\}}{\{I\}\ \text{WHILE } c \text{ DO } S \text{ ENDWHILE } \{I \wedge \neg c\}}$$

- ☑ Sequence
- ☑ Alternation
- ☑ Repetition

Think of it like a flow chart:



While Loop Rule

$\{x \leq 10\}$ WHILE $x < 10$ DO $x{:=}x{+}1$ ENDWHILE $\{x{=}10\}$

{P} {c} {S} {Q}

What's a good invariant?

While Loop Rule

$\{x \leq 10\}$ WHILE $x < 10$ DO $x{:=}x{+}1$ ENDWHILE $\{x{=}10\}$

$\{P\}$ $\{c\}$ $\{S\}$ $\{Q\}$

What's a good invariant?
Let's try $\{I\} = x \leq 10$

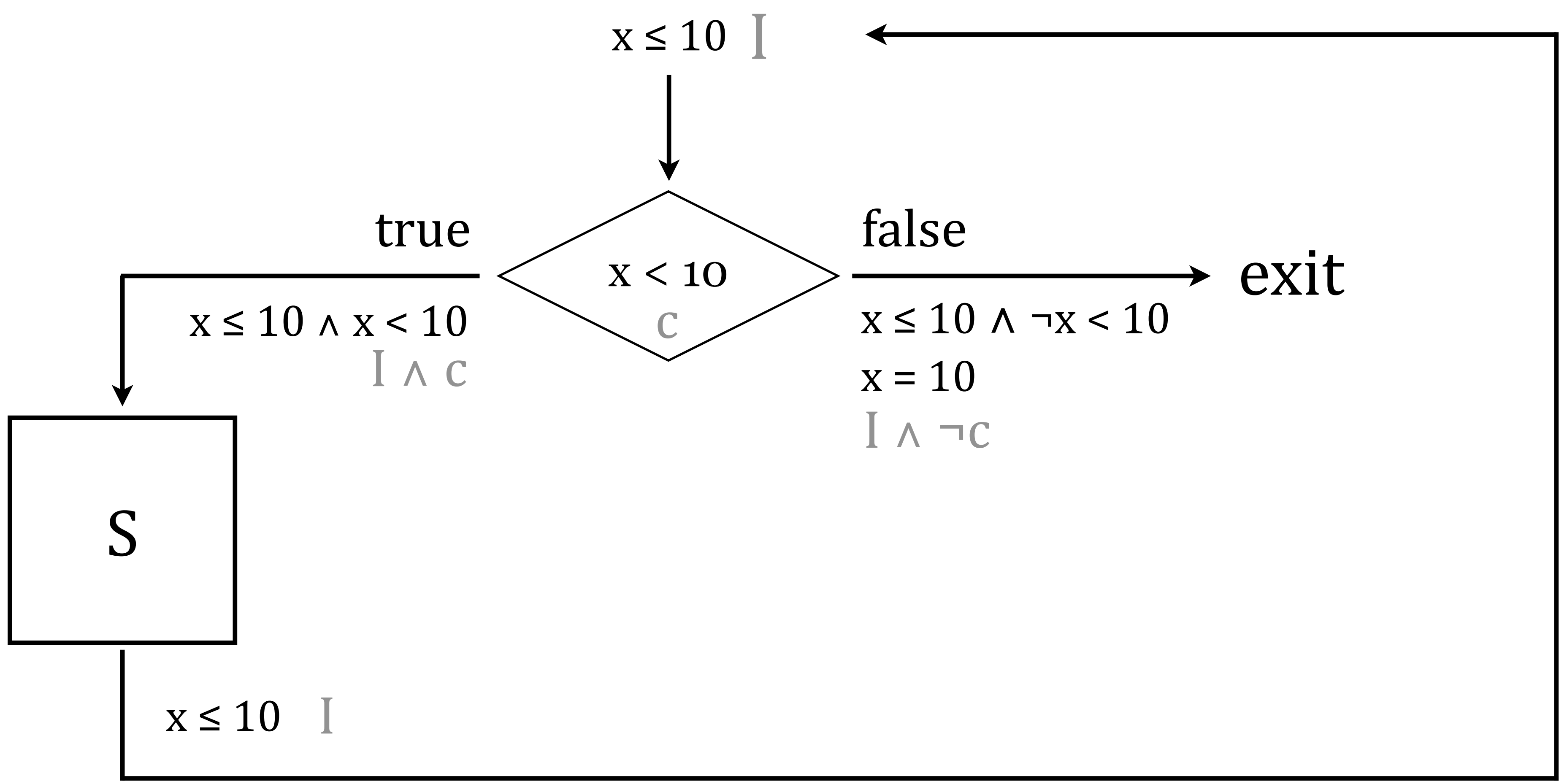
$$\frac{\{I \wedge c\}\ S\ \{I\}}{\{I\}\ \text{WHILE}\ c\ \text{DO}\ S\ \text{ENDWHILE}\ \{I \wedge \neg c\}}$$

$$\frac{\{x \leq 10 \wedge x < 10\}\ x{:=}x{+}1\ \{x \leq 10\}}{\{x \leq 10\}\ \text{WHILE}\ x < 10\ \text{DO}\ x{:=}x{+}1\ \text{ENDWHILE}\ \{x \leq 10 \wedge \neg x < 10\}}$$

While Loop Rule

$$\{x \le 10\}\ \text{WHILE } x < 10 \text{ DO } x{:=}x{+}1 \text{ ENDWHILE } \{x{=}10\}$$

$$\frac{\{x \le 10 \wedge x < 10\}\ x{:=}x{+}1\ \{x \le 10\}}{\{x \le 10\}\ \text{WHILE } x < 10 \text{ DO } x{:=}x{+}1 \text{ ENDWHILE } \{x \le 10 \wedge \neg x < 10\}}$$



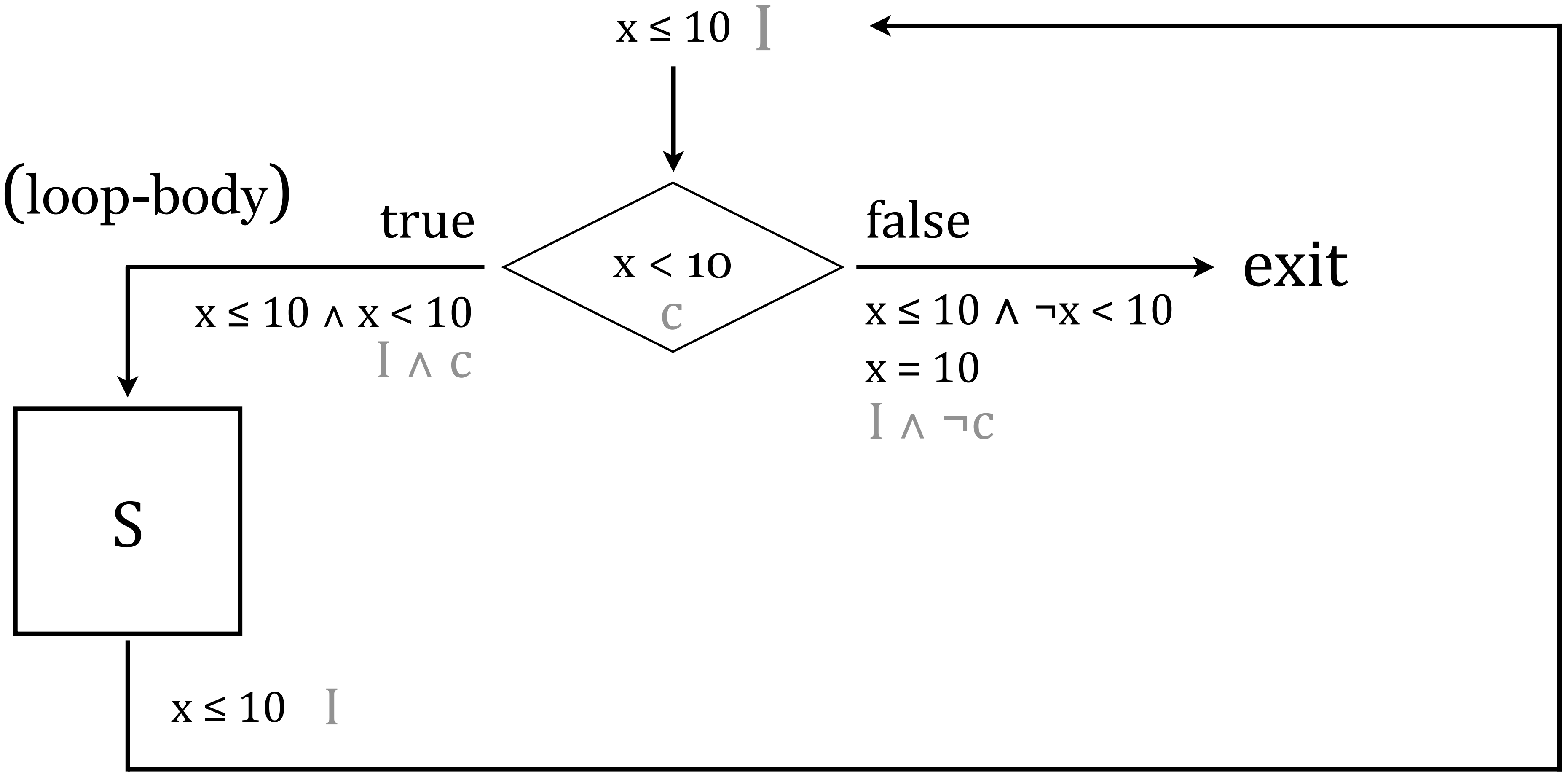
While Loop Rule

$\{x \le 10\}$ WHILE $x < 10$ DO $x:=x+1$ ENDWHILE $\{x=10\}$

$$\frac{\{x \le 10 \wedge x < 10\}\ x:=x+1\ \{x \le 10\}}{\{x \le 10\}\ \text{WHILE } x < 10 \text{ DO } x:=x+1 \text{ ENDWHILE } \{x \le 10 \wedge \neg x < 10\}}$$

We need to prove

1. $P \Rightarrow I$
2. $I \wedge \neg c \Rightarrow Q$
3. $I \wedge c \Rightarrow \text{WLP(loop-body)}$



While Loop Rule

$$\{\mathbf{x \le 10}\}\ \text{WHILE } x < 10 \text{ DO } x{:=}x+1 \text{ ENDWHILE } \{x=10\}$$

$$\frac{\{x \le 10 \wedge x < 10\}\ x{:=}x+1\ \{x \le 10\}}{\{x \le 10\}\ \text{WHILE } x < 10 \text{ DO } x{:=}x+1 \text{ ENDWHILE } \{x \le 10 \wedge \neg x < 10\}}$$

We need to prove

1. $\mathbf{P \Rightarrow I}$
2. $I \wedge \neg c \Rightarrow Q$
3. $I \wedge c \Rightarrow$ WLP(loop-body)

$P = x \le 10$

$I = x \le 10$

$x \le 10 \Rightarrow x \le 10$

While Loop Rule

$$\{x \leq 10\}\ \text{WHILE}\ \mathbf{x < 10}\ \text{DO}\ x{:=}x{+}1\ \text{ENDWHILE}\ \{\mathbf{x{=}10}\}$$

$$\frac{\{x \leq 10 \wedge x < 10\}\ x{:=}x{+}1\ \{x \leq 10\}}{\{x \leq 10\}\ \text{WHILE}\ x < 10\ \text{DO}\ x{:=}x{+}1\ \text{ENDWHILE}\ \{x \leq 10 \wedge \neg x < 10\}}$$

We need to prove

1. $P \Rightarrow I$
2. $\mathbf{I \wedge \neg c \Rightarrow Q}$
3. $I \wedge c \Rightarrow$ WLP(loop-body)

$$
\begin{aligned}
I &= x \leq 10 \\
c &= x < 10 \\
\neg c &= x \geq 10 \\
I \wedge \neg c &= x \leq 10 \wedge x \geq 10 \\
&= x = 10 \\
Q &= x = 10
\end{aligned}
$$

$$x = 10 \Rightarrow x = 10$$

While Loop Rule

$$\{x \le 10\}\ \text{WHILE } x < 10 \text{ DO } \mathbf{x{:=}x+1} \text{ ENDWHILE } \{\mathbf{x=10}\}$$

$$\frac{\{x \le 10 \land x < 10\}\ x{:=}x+1\ \{x \le 10\}}{\{x \le 10\}\ \text{WHILE } x < 10 \text{ DO } x{:=}x+1 \text{ ENDWHILE } \{x \le 10 \land \neg x < 10\}}$$

We need to prove

1. $P \Rightarrow I$
2. $I \land \neg c \Rightarrow Q$
3. $\mathbf{I \land c \Rightarrow}$ **WLP(**loop-body**)**

loop body:

$\{I \land c\}\ S\ \{I\}$

$\{x \le 10 \land x < 10\}\ x{=}x+1\ \{x \le 10\}$

$\{x \le 10\}[x+1/x]$

$\{x+1 \le 10\}$

$\{x \le 9\}$ WLP

$I \land c \Rightarrow$ WLP(loop-body)

$\{x \le 10 \land x < 10\} \Rightarrow x \le 9$ for integers

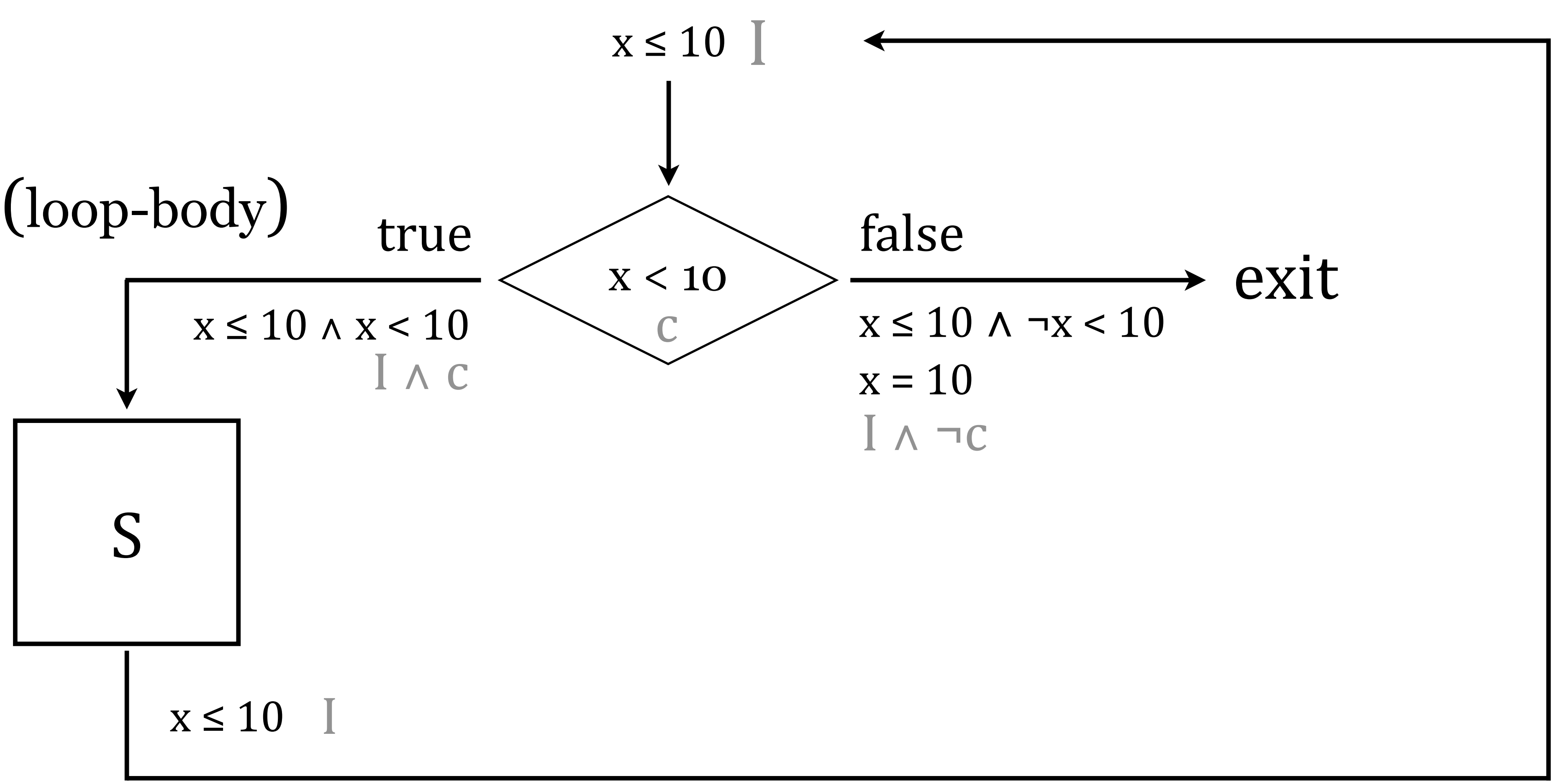
While Loop Rule

$$\{x \le 10\}\ \text{WHILE } x < 10 \text{ DO } \mathbf{x{:=}x{+}1} \text{ ENDWHILE } \{\mathbf{x=10}\}$$

$$\frac{\{x \le 10 \wedge x < 10\}\ x{:=}x{+}1\ \{x \le 10\}}{\{x \le 10\}\ \text{WHILE } x < 10 \text{ DO } x{:=}x{+}1 \text{ ENDWHILE } \{x \le 10 \wedge \neg x < 10\}}$$

We need to prove

1. $P \Rightarrow I$
2. $I \wedge \neg c \Rightarrow Q$
3. $I \wedge c \Rightarrow \text{WLP(loop-body)}$



Axiomatic Semantics

That does it!

- ☑ Sequence
- ☑ Alternation
- ☑ Repetition

Go forth and prove programs correct.