Logic in Computer Science (ECS666U/7018P/U)

Nikos Tzevelekos

Lecture 10
Part A: Hoare Logic, Syntax and Semantics

Hoare Logic

Also known as Floyd-Hoare Logic and Program Logic. It is:

- a method of reasoning mathematically about imperative programs
  - can be used directly to verify programs, as originally proposed
  - tedious and error prone
  - impractical for large programs
- the basis of semi- and fully-automated verification systems
  - construct verification conditions from specification in Hoare Logic
  - run programs "symbolically" from Hoare logic specifications
- under active development
  - for example Separation Logic for reasoning about pointers

Outline

- Program specification using Hoare Triples
- Inference rules of Hoare Logic
- Soundness and completeness
- Verification conditions

- Further reading (optional):
  - Mike Gordon: Hoare Logic
  - Jean Pichon: Hoare Logic and Model Checking

Example: Hoare Triples

Program c0

```
n := 0;
y := 0;
while (n < x) {
  y := y + (2 * n + 1);
  n := n + 1;
}
```

Inputs and corresponding outputs

x	0	1	2	3	4	...	100	...
y	0	1	4					

Example: Hoare Triples

Program c0	Inputs and corresponding outputs

```
n := 0;
y := 0;
while (n < x) {
  y := y + (2 * n + 1);
  n := n + 1;
}
```

x	0	1	2	3	4	...	100	...
y	0	1	4					

- Hypothesis 1: For any integer input x, the program c0 has output y that satisfies the following condition:

$$y = x^2$$

Example: Hoare Triples

<table>
<tr><th>Program <code>c0</code></th><th>Inputs and corresponding outputs</th></tr>
<tr><td>

```
n := 0;
y := 0;
while (n < x) {
   y := y + (2 * n + 1);
   n := n + 1;
}
```

</td><td>

x	0	1	2	3	4	...	100	...
y	0	1	4					

</td></tr>
</table>

- Hypothesis 1: For any integer input `x`, the program `c0` has output `y` that satisfies the following condition:

$$y = x^2$$

- What happens in the case of `x = -1`?

Example: Hoare Triples

Program `c0`	Inputs and corresponding outputs

```
n := 0;
y := 0;
while (n < x) {
  y := y + (2 * n + 1);
  n := n + 1;
}
```

x	0	1	2	3	4	-1
y	0	1	4	9	16	?

- Hypothesis 2: For any integer input `x` which satisfies the condition

$$x \geq 0$$

the program `c0` output `y` satisfies the following condition:

$$y = x^2$$

Example: Hoare Triples

<table>
<tr><th>Program co</th><th>Inputs and corresponding outputs</th></tr>
<tr><td>

```
n := 0;
y := 0;
while (n < x) {
  y := y + (2 * n + 1);
  n := n + 1;
}
```

</td><td>

x	0	1	2	3	4	-1
y	0	1	4	9	16	?

</td></tr>
</table>

- Formally, we write Hoare triple in the form of

$$\{Pre\}\ \texttt{co}\ \{Post\}$$

- The condition *Pre* on the input is called precondition
- The condition *Post* on the output is called postcondition
- In this example:

$$\{x \geq 0\}\ \texttt{co}\ \{y = x^2\}$$

Example: Hoare Triples

<table>
<tr><th>Program <code>c0</code></th><th>Inputs and corresponding outputs</th></tr>
<tr><td>

```
n := 0;
y := 0;
while (n < x) {
  y := y + (2 * n + 1);
  n := n + 1;
}
```

</td><td>

x	0	1	2	3	4	-1
y	0	1	4	9	16	?

</td></tr>
</table>

- $\{x \geq 0\}$ `c0` $\{y = x^2\}$ is valid

Example: Hoare Triples

<table>
<tr><th>Program co</th><th>Inputs and corresponding outputs</th></tr>
<tr><td>

```
n := 0;
y := 0;
while (n < x) {
  y := y + (2 * n + 1);
  n := n + 1;
}
```

</td><td>

x	0	1	2	3	4	-1
y	0	1	4	9	16	?

</td></tr>
</table>

- $\{x \geq 0\}$ co $\{y = x^2\}$ is valid
- $\{x = 5\}$ co $\{y = 25\}$

Example: Hoare Triples

<table>
<tr><th>Program c0</th><th>Inputs and corresponding outputs</th></tr>
<tr><td><pre>
n := 0;
y := 0;
while (n < x) {
  y := y + (2 * n + 1);
  n := n + 1;
}
</pre></td><td>

x	0	1	2	3	4	-1
y	0	1	4	9	16	?

</td></tr>
</table>

- $\{x \geq 0\}$ c0 $\{y = x^2\}$ is valid
- $\{x = 5\}$ c0 $\{y = 25\}$ is valid (Why?)

Example: Hoare Triples

<table>
<tr><th>Program c0</th><th>Inputs and corresponding outputs</th></tr>
<tr><td>

```
n := 0;
y := 0;
while (n < x) {
  y := y + (2 * n + 1);
  n := n + 1;
}
```

</td><td>

x	0	1	2	3	4	-1
y	0	1	4	9	16	?

</td></tr>
</table>

- $\{x \geq 0\}$ c0 $\{y = x^2\}$ is valid
- $\{x = 5\}$ c0 $\{y = 25\}$ is valid (Why?)
  - execution of this code with concrete input 5 outputs 25

Example: Hoare Triples

Program `c0`	Inputs and corresponding outputs

`n := 0;`
`y := 0;`
`while (n < x) {`
`  y := y + (2 * n + 1);`
`  n := n + 1;`
`}`
	<table><tr><td>x</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>-1</td></tr><tr><td>y</td><td>0</td><td>1</td><td>4</td><td>9</td><td>16</td><td>?</td></tr></table>

- $\{x \geq 0\}$ `c0` $\{y = x^2\}$ is valid
- $\{x = 5\}$ `c0` $\{y = 25\}$ is valid (Why?)
  - execution of this code with concrete input 5 outputs 25
- $\{x \geq 5\}$ `c0` $\{y = 25\}$

Example: Hoare Triples

Program c0	Inputs and corresponding outputs

`n := 0;`
`y := 0;`
`while (n < x) {`
`  y := y + (2 * n + 1);`
`  n := n + 1;`
`}`
	<table><tr><td>x</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>-1</td></tr><tr><td>y</td><td>0</td><td>1</td><td>4</td><td>9</td><td>16</td><td>?</td></tr></table>

- $\{x \geq 0\}$ c0 $\{y = x^2\}$ is valid
- $\{x = 5\}$ c0 $\{y = 25\}$ is valid (Why?)
  - execution of this code with concrete input 5 outputs 25
- $\{x \geq 5\}$ c0 $\{y = 25\}$ is not valid (Why?)

Example: Hoare Triples

<table>
<tr><th>Program <code>c0</code></th><th>Inputs and corresponding outputs</th></tr>
<tr><td><pre>n := 0;
y := 0;
while (n < x) {
  y := y + (2 * n + 1);
  n := n + 1;
}</pre></td><td>

x	0	1	2	3	4	-1
y	0	1	4	9	16	?

</td></tr>
</table>

- $\{x \geq 0\}$ `c0` $\{y = x^2\}$ is valid
- $\{x = 5\}$ `c0` $\{y = 25\}$ is valid (Why?)
  - execution of this code with concrete input 5 outputs 25
- $\{x \geq 5\}$ `c0` $\{y = 25\}$ is not valid (Why?)
  - a counterexample $[x \mapsto 6]$

Example: Hoare Triples

<table>
<tr><th>Program c0</th><th>Inputs and corresponding outputs</th></tr>
<tr><td>

```
n := 0;
y := 0;
while (n < x) {
  y := y + (2 * n + 1);
  n := n + 1;
}
```

</td><td>

x	0	1	2	3	4	-1
y	0	1	4	9	16	?

</td></tr>
</table>

- $\{x \geq 0\}$ c0 $\{y = x^2\}$ is valid
- $\{x = 5\}$ c0 $\{y = 25\}$ is valid (Why?)
  - execution of this code with concrete input 5 outputs 25
- $\{x \geq 5\}$ c0 $\{y = 25\}$ is not valid (Why?)
  - a counterexample $[x \mapsto 6]$
- $\{x \geq 5\}$ c0 $\{y \geq 25\}$

Example: Hoare Triples

Program `c0`	Inputs and corresponding outputs

<pre>n := 0;
y := 0;
while (n < x) {
  y := y + (2 * n + 1);
  n := n + 1;
}</pre>	

x	0	1	2	3	4	-1
y	0	1	4	9	16	?

- $\{x \geq 0\}$ `c0` $\{y = x^2\}$ is valid
- $\{x = 5\}$ `c0` $\{y = 25\}$ is valid (Why?)
  - execution of this code with concrete input 5 outputs 25
- $\{x \geq 5\}$ `c0` $\{y = 25\}$ is not valid (Why?)
  - a counterexample $[x \mapsto 6]$
- $\{x \geq 5\}$ `c0` $\{y \geq 25\}$ is valid. (Why?)

Example: Hoare Triples

Program `c0`	Inputs and corresponding outputs

`n := 0;`
`y := 0;`
`while (n < x) {`
`  y := y + (2 * n + 1);`
`  n := n + 1;`
`}`
	<table><tr><td>x</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>-1</td></tr><tr><td>y</td><td>0</td><td>1</td><td>4</td><td>9</td><td>16</td><td>?</td></tr></table>

- $\{x \geq 0\}$ `c0` $\{y = x^2\}$ is valid
- $\{x = 5\}$ `c0` $\{y = 25\}$ is valid (Why?)
  - execution of this code with concrete input 5 outputs 25
- $\{x \geq 5\}$ `c0` $\{y = 25\}$ is not valid (Why?)
  - a counterexample $[x \mapsto 6]$
- $\{x \geq 5\}$ `c0` $\{y \geq 25\}$ is valid. (Why?)
  - the output y of `C0` is monotone in the input x

Example: Hoare Triples

Program C0	Inputs and corresponding outputs

<pre>n := 0;
y := 0;
while (n < x) {
  y := y + (2 * n + 1);
  n := n + 1;
}</pre>
	x: 0, 1, 2, 3, 4, -1 / y: 0, 1, 4, 9, 16, ?

x	0	1	2	3	4	-1
y	0	1	4	9	16	?

- $\{x \geq 0\}$ C0 $\{y = x^2\}$ is valid
- $\{x = 5\}$ C0 $\{y = 25\}$ is valid (Why?)
  - execution of this code with concrete input 5 outputs 25
- $\{x \geq 5\}$ C0 $\{y = 25\}$ is not valid (Why?)
  - a counterexample $[x \mapsto 6]$
- $\{x \geq 5\}$ C0 $\{y \geq 25\}$ is valid. (Why?)
  - the output y of C0 is monotone in the input x
- How to prove and disprove validity of Hoare triples?

Example: Hoare triples

<table>
<tr><th>Program <code>C1</code></th><th>Inputs and corresponding outputs</th></tr>
<tr><td><pre>while (x != 1) {
  if (x % 2 = 0) {
    x := x/2;
  }
  else {
    x := 3*x + 1;
  }
};
y := x;</pre></td><td>

x	1	2	3	4	...	1001	...
y	1	1	1	1	...	1	...

</td></tr>
</table>

- Hypothesis: For any positive integer input `x` the program `C1` terminates and the output `y` is 1.

Example: Hoare triples

<table>
<tr><th>Program <code>C1</code></th><th>Inputs and corresponding outputs</th></tr>
<tr><td>

```
while (x != 1) {
  if (x % 2 = 0) {
    x := x/2;
  }
  else {
    x := 3*x + 1;
  }
};
y := x;
```

</td><td>

x	1	2	3	4	...	1001	...
y	1	1	1	1	...	1	...

</td></tr>
</table>

- Hypothesis: For any positive integer input `x` the program `C1` terminates and the output `y` is 1.
- How to check such statements?
- What is the issue - to check that `C1` terminates or that it outputs 1?

Syntax and Semantics of Hoare Triples

Syntax of Hoare Triples

- Triple $\{P\}$ `c` $\{Q\}$
- P and Q are predicate logic formulas
- `c` is a code fragment in a simple programming language
- Example: $\{x \geq 0\}$ `y:=x+x` $\{y = 2 * x\}$

Programming Language - Syntax

- Assignment: `x:=E`
- Sequential composition: `C1;C2`
- Conditional: `if B C1 else C2`
- Loop: `while B C`

- where `E` is an integer expression and `B` is a boolean expression

Partial Correctness Specification

- $\{P\}$ `c` $\{Q\}$ is valid if
  - if `C` is executed from any state satisfying P
    and the execution of `C` terminates
  - then the state in which the execution of `C` terminates satisfies Q

Partial Correctness Specification

- $\{P\}$ `c` $\{Q\}$ is valid if
  - if `C` is executed from any state satisfying P and the execution of `C` terminates
  - then the state in which the execution of `C` terminates satisfies Q

- These specifications are "partial" because for $\{P\}$ `c` $\{Q\}$ to be true we do not care what happens when `c` runs forever.

  For example, this is a valid triple:

$$\{x = 1\}\ \texttt{while True do x:=x}\ \{x = 42\}$$

Total Correctness Specification

- $[P]$ `C` $[Q]$ is valid if
  - if `C` is executed from any state satisfying P
  - then the execution of `C` terminates
    and the state in which the execution terminates satisfies Q
- Total correctness is the ultimate goal

Partial vs Total Correctness

- Total correctness is stronger than partial correctness:
  - $\{x = 1\}$ `while True do x := x` $\{x = 42\}$ is valid
  - $[x = 1]$ `while True do x := x` $[x = 42]$ is not valid

Partial vs Total Correctness

- Total correctness is stronger than partial correctness:
  - $\{x = 1\}$ `while True do x := x` $\{x = 42\}$ is valid
  - $[x = 1]$ `while True do x := x` $[x = 42]$ is not valid
- Informally: total correctness $=$ termination $+$ partial correctness

Partial vs Total Correctness

- Total correctness is stronger than partial correctness:
  - $\{x = 1\}$ `while True do x := x` $\{x = 42\}$ is valid
  - $[x = 1]$ `while True do x := x` $[x = 42]$ is not valid
- Informally: total correctness = termination + partial correctness
- Usually easier to show separately partial correctness and termination

Partial vs Total Correctness

- Total correctness is stronger than partial correctness:
  - $\{x = 1\}$ `while True do x := x` $\{x = 42\}$ is valid
  - $[x = 1]$ `while True do x := x` $[x = 42]$ is not valid
- Informally: total correctness $=$ termination $+$ partial correctness
- Usually easier to show separately partial correctness and termination
- In real software, termination is often straightforward to show
- In general, termination is undecidable

Example: Termination is Hard

<table>
<tr><th>Program <code>c1</code></th><th>Inputs and corresponding outputs</th></tr>
<tr><td>

```
while (x != 1) {
  if (x % 2 = 0) {
    x := x/2;
  }
  else {
    x := 3*x + 1;
  }
};
y := x;
```

</td><td>

x	1	2	3	4	...	1001	...
y	1	1	1	1	...	1	...

</td></tr>
</table>

- Formally, is $\{x > 0\}$ `c1` $\{y = 1\}$ valid?

Example: Termination is Hard

<table>
<tr><th>Program C1</th><th>Inputs and corresponding outputs</th></tr>
<tr><td>

```
while (x != 1) {
  if (x % 2 = 0) {
    x := x/2;
  }
  else {
    x := 3*x + 1;
  }
};
y := x;
```

</td><td>

x	1	2	3	4	...	1001	...
y	1	1	1	1	...	1	...

</td></tr>
</table>

- Formally, is $\{x > 0\}$ `C1` $\{y = 1\}$ valid?

  Yes!

- However, no one knows whether `C1` terminates for all values of `x`
- Collatz conjecture: it terminates with `y=1`
- Formally, is $[x > 0]$ `C1` $[y = 1]$ valid?

Connection to Predicate Logic and its Semantics

Connection to Predicate Logic

$$\{x \geq 0\} \ \texttt{y:=x+x} \ \{y = 2 * x\}$$

- The precondition and postcondition are formulas of predicate logic
- To every program variable we assign a corresponding logical variable
- So, pre- and post-conditions talk about the values of the program variables

Connection to Predicate Logic

$$\{x \geq 0\}\ \texttt{y:=x+x}\ \{y = 2 * x\}$$

- The precondition and postcondition are formulas of predicate logic
- To every program variable we assign a corresponding logical variable
- So, pre- and post-conditions talk about the values of the program variables

The model M we consider will be an integer arithmetic model.
We will focus on program states.

Connection to Predicate Logic

$$\{x \geq 0\}\ \texttt{y:=x+x}\ \{y = 2 * x\}$$

- The precondition and postcondition are formulas of predicate logic
- To every program variable we assign a corresponding logical variable
- So, pre- and post-conditions talk about the values of the program variables

The model M we consider will be an integer arithmetic model.
We will focus on program states.

- Program state is a mapping from variables to values, i.e. the program state tells us what value is stored in each variable
- This is exactly the same as evaluation maps:
  - e.g. $s = [\mathrm{x} \mapsto 5, \mathrm{y} \mapsto 10]$ is a program state
    and $E = [x \mapsto 5, y \mapsto 10]$ is a corresponding evaluation of logical variables
- What does it mean for a program state to satisfy a precondition?

Another Example

$$\{x = 0 \land y = 1\}\ \texttt{x:=x+y}\ \{\exists z. x > z\}$$

- Initial program state $s = [\mathrm{x} \mapsto 0, \mathrm{y} \mapsto 1]$
- Final program state $s' = [\mathrm{x} \mapsto 1, \mathrm{y} \mapsto 1]$

- Does state s satisfy the precondition $x = 0 \land y = 1$?

Another Example

$$\{x = 0 \land y = 1\}\ \texttt{x:=x+y}\ \{\exists z. x > z\}$$

- Initial program state $s = [\mathrm{x} \mapsto 0, \mathrm{y} \mapsto 1]$
- Final program state $s' = [\mathrm{x} \mapsto 1, \mathrm{y} \mapsto 1]$

- Does state s satisfy the precondition $x = 0 \land y = 1$?
  - we let $E = [x \mapsto 0, y \mapsto 1]$, and check that $M, E \models x = 0 \land y = 1$
  - less formally, check whether replacing x for 0 and y for 1 makes the precondition true
  - to simplify notation, we simply write $s \models x = 0 \land y = 1$ (i.e. avoid mentioning M, E)

Another Example

$$\{x = 0 \land y = 1\} \ \texttt{x:=x+y} \ \{\exists z. x > z\}$$

- Initial program state $s = [\mathrm{x} \mapsto 0, \mathrm{y} \mapsto 1]$
- Final program state $s' = [\mathrm{x} \mapsto 1, \mathrm{y} \mapsto 1]$

- Does state s satisfy the precondition $x = 0 \land y = 1$?
  - we let $E = [x \mapsto 0, y \mapsto 1]$, and check that $M, E \models x = 0 \land y = 1$
  - less formally, check whether replacing x for 0 and y for 1 makes the precondition true
  - to simplify notation, we simply write $s \models x = 0 \land y = 1$ (i.e. avoid mentioning M, E)
- Does state s' satisfy the postcondition $\exists z. x > z$?

Another Example

$$\{x = 0 \land y = 1\}\ \texttt{x:=x+y}\ \{\exists z. x > z\}$$

- Initial program state $s = [\mathrm{x} \mapsto 0, \mathrm{y} \mapsto 1]$
- Final program state $s' = [\mathrm{x} \mapsto 1, \mathrm{y} \mapsto 1]$

- Does state s satisfy the precondition $x = 0 \land y = 1$?
  - we let $E = [x \mapsto 0, y \mapsto 1]$, and check that $M, E \models x = 0 \land y = 1$
  - less formally, check whether replacing x for 0 and y for 1 makes the precondition true
  - to simplify notation, we simply write $s \models x = 0 \land y = 1$ (i.e. avoid mentioning M, E)
- Does state s' satisfy the postcondition $\exists z. x > z$?
  - we check whether $s' \models \exists z. x > z$.

    This holds indeed, e.g. by choosing z to be 0.

General idea

$$\{P\}\ \mathtt{c}\ \{Q\}$$

The triple is valid if:

- for any program state s with corresponding evaluation E
- if $M, E \models P$ holds and execution of $\mathtt{c}$ from state s terminates
- then the execution terminates in a state s', with corresponding evaluation E', such that $M, E' \models Q$ holds

Or, **more simply**, the triple is valid if:

- for any program state s
- if $s \models P$ holds and execution of $\mathtt{c}$ from state s terminates
- then it terminates in a state s' such that $s' \models Q$ holds

Simple Examples

1. $\{\mathsf{true}\}\ \mathtt{c}\ \{Q\}$

Simple Examples

1. $\{\text{true}\}\ \texttt{C}\ \{Q\}$
   - this says that if $\texttt{C}$ terminates then Q holds

Simple Examples

1. $\{\text{true}\}\ \texttt{c}\ \{Q\}$
   - this says that if $\texttt{C}$ terminates then Q holds
2. $\{P\}\ \texttt{c}\ \{\text{true}\}$

Simple Examples

1. $\{\text{true}\}$ `c` $\{Q\}$
   - this says that if `C` terminates then Q holds
2. $\{P\}$ `c` $\{\text{true}\}$
   - valid for any precondition P and any code `C`

Simple Examples

1. $\{\text{true}\}$ `c` $\{Q\}$
   - this says that if `c` terminates then Q holds
2. $\{P\}$ `c` $\{\text{true}\}$
   - valid for any precondition P and any code `c`
3. $[P]$ `c` $[\text{true}]$

Simple Examples

1. $\{\text{true}\}\ \texttt{c}\ \{Q\}$
   - this says that if `c` terminates then Q holds
2. $\{P\}\ \texttt{c}\ \{\text{true}\}$
   - valid for any precondition P and any code `c`
3. $[P]\ \texttt{c}\ [\text{true}]$
   - this says that `c` terminates whenever the initial state satisfies P
   - it does not say anything about the final state

Simple Examples

1. $\{\text{true}\}\ \texttt{c}\ \{Q\}$
   - this says that if $\texttt{c}$ terminates then Q holds
2. $\{P\}\ \texttt{c}\ \{\text{true}\}$
   - valid for any precondition P and any code $\texttt{c}$
3. $[P]\ \texttt{c}\ [\text{true}]$
   - this says that $\texttt{c}$ terminates whenever the initial state satisfies P
   - it does not say anything about the final state
4. $[\text{true}]\ \texttt{c}\ [Q]$

Simple Examples

1. $\{\text{true}\}$ `c` $\{Q\}$
   - this says that if `C` terminates then Q holds
2. $\{P\}$ `c` $\{\text{true}\}$
   - valid for any precondition P and any code `C`
3. $[P]$ `c` $[\text{true}]$
   - this says that `C` terminates whenever the initial state satisfies P
   - it does not say anything about the final state
4. $[\text{true}]$ `c` $[Q]$
   - this says that `C` always terminates and ends in a final state that satisfies Q

Simple Examples

1. $\{\text{true}\}$ `c` $\{Q\}$
   - this says that if `C` terminates then Q holds
2. $\{P\}$ `c` $\{\text{true}\}$
   - valid for any precondition P and any code `C`
3. $[P]$ `c` $[\text{true}]$
   - this says that `C` terminates whenever the initial state satisfies P
   - it does not say anything about the final state
4. $[\text{true}]$ `c` $[Q]$
   - this says that `C` always terminates and ends in a final state that satisfies Q
5. $[\text{true}]$ `c` $[\text{true}]$

Simple Examples

1. $\{\text{true}\}\ \texttt{c}\ \{Q\}$
   - this says that if `C` terminates then Q holds
2. $\{P\}\ \texttt{c}\ \{\text{true}\}$
   - valid for any precondition P and any code `C`
3. $[P]\ \texttt{c}\ [\text{true}]$
   - this says that `C` terminates whenever the initial state satisfies P
   - it does not say anything about the final state
4. $[\text{true}]\ \texttt{c}\ [Q]$
   - this says that `C` always terminates and ends in a final state that satisfies Q
5. $[\text{true}]\ \texttt{c}\ [\text{true}]$
   - this says that the code `C` terminates for all inputs

Example: Loop

Program c2	Inputs and corresponding outputs

`r := x;`
`q := 0;`
`while (y <= r) {`
`  r := r - y;`
`  q := q + 1;`
`}`
	see table below

x	1	2	3	4	0	5	-5	5
y	1	1	2	2	1	0	2	-2
q	1	2	1	2	0	?	?	?
r	0	0	1	0	0	?	?	?

Example: Loop

Program `c2`

```
r := x;
q := 0;
while (y <= r) {
  r := r - y;
  q := q + 1;
}
```

Inputs and corresponding outputs

x	1	2	3	4	0	5	-5	5
y	1	1	2	2	1	0	2	-2
q	1	2	1	2	0	?	?	?
r	0	0	1	0	0	?	?	?

- Specification: $\{\text{true}\}$ `c2` $\{r < y \wedge x = r + y * q\}$
- The triple is valid if whenever the execution of `c2` terminates, then `q` is the quotient and `r` is the remainder resulting from dividing `x` by `y`

Example: Loop

Program C2	Inputs and corresponding outputs
`r := x;` `q := 0;` `while (y <= r) {` `r := r - y;` `q := q + 1;` `}`	see table below

```
r := x;
q := 0;
while (y <= r) {
  r := r - y;
  q := q + 1;
}
```

x	1	2	3	4	0	5	-5	5
y	1	1	2	2	1	0	2	-2
q	1	2	1	2	0	?	?	?
r	0	0	1	0	0	?	?	?

- Specification: $\{\text{true}\}$ `C2` $\{r < y \wedge x = r + y * q\}$
- The triple is valid if whenever the execution of `C2` terminates, then `q` is the quotient and `r` is the remainder resulting from dividing `x` by `y`
- Is it valid if `y` is 0?

Example: Loop

<table>
<tr><th>Program <code>C2</code></th><th>Inputs and corresponding outputs</th></tr>
<tr><td>

```
r := x;
q := 0;
while (y <= r) {
  r := r - y;
  q := q + 1;
}
```

</td><td>

x	1	2	3	4	0	5	-5	5
y	1	1	2	2	1	0	2	-2
q	1	2	1	2	0	?	?	?
r	0	0	1	0	0	?	?	?

</td></tr>
</table>

- Specification: $\{\text{true}\}$ `C2` $\{r < y \wedge x = r + y * q\}$
- The triple is valid if whenever the execution of `C2` terminates, then `q` is the quotient and `r` is the remainder resulting from dividing `x` by `y`
- Is it valid if `y` is 0?
- Is it valid if `x` is initially negative?

Example: Loop

Program `c2`	Inputs and corresponding outputs

```
r := x;
q := 0;
while (y <= r) {
  r := r - y;
  q := q + 1;
}
```

x	1	2	3	4	0	5	-5	5
y	1	1	2	2	1	0	2	-2
q	1	2	1	2	0	?	?	?
r	0	0	1	0	0	?	?	?

- Specification: $\{\text{true}\}$ `c2` $\{r < y \wedge x = r + y * q\}$
- The triple is valid if whenever the execution of `c2` terminates, then `q` is the quotient and `r` is the remainder resulting from dividing `x` by `y`
- Is it valid if `y` is 0?
- Is it valid if `x` is initially negative?
- Is it valid if `y` is initially negative?

Example: Loop

Program C2	Inputs and corresponding outputs

```
r := x;
q := 0;
while (y <= r) {
  r := r - y;
  q := q + 1;
}
```

x	1	2	3	4	0	5	-5	5
y	1	1	2	2	1	0	2	-2
q	1	2	1	2	0	?	?	?
r	0	0	1	0	0	?	?	?

- Specification: $\{\text{true}\}$ `C2` $\{r < y \wedge x = r + y * q\}$
- The triple is valid if whenever the execution of `C2` terminates, then `q` is the quotient and `r` is the remainder resulting from dividing `x` by `y`
- Is it valid if `y` is 0?
- Is it valid if `x` is initially negative?
- Is it valid if `y` is initially negative?
- How to specify total correctness of this program?

Exercise

- Write a specification which is valid if and only if the execution of `C` always terminates when the execution is started in a state satisfying P

- Write a specification which is valid if and only if the code `C` has the effect of multiplying the values of `x` and `y` and storing the result in `z` (and terminate)

Auxiliary Variables

Tricky Example

- The program `c` should multiply the values of `x` and `y` and store the result in `x` (and terminate)

$$[\mathsf{true}]\ \mathtt{c}\ [x = x * y]$$

- Are the following programs suitable?
  - `x := x*y;`
  - `y := 1; x := 1`

Tricky Example

- The program `c` should multiply the values of `x` and `y` and store the result in `x` (and terminate)

$$[\text{true}]\ \texttt{c}\ [x = x * y]$$

- Are the following programs suitable?
  - `x := x*y;`
  - `y := 1; x := 1`

- There is a problem: the postcondition $x = x * y$ requires that `x` is `x` times `y` in the final state
- but what we really wanted is to relate the final and the initial values of `x`

Auxiliary Variables

- $[x_0 = x \wedge y_0 = y]$ `x:=x*y` $[x = x_0 * y_0]$
- The variables x_0 and y_0 are used in this example in order to remember the initial values of program variables `x` and `y`

- Auxiliary variables are free variables introduced in preconditions and do not occur in the code
- As such, auxiliary variables do not change value between the pre- and post-condition
- Auxiliary variables are sometimes called ghost variables

Another Example

- $\{x_0 = x \land y_0 = y\}$ `r:=x; x:=y; y:=r` $\{x_0 = y \land y_0 = x\}$
- If the execution of `r:=x; x:=y; y:=r` terminates (which it does),

  then the values of `x` and `y` are exchanged
- Formally, the above triple is valid if:
  - if we pick any values for the auxiliary variables x_0, y_0
  - and we execute the code from any state satisfying $x_0 = x \land y_0 = y$ and the execution terminates
  - then the execution terminates in a state satisfying $x_0 = y \land y_0 = x$

Predicate Logic view

In terms of predicate logic, $\{P\}\ \mathtt{c}\ \{Q\}$ is valid if:

- for every program state s, and corresponding evaluation E, and any evaluation E_0 of all auxiliary variables in P,
- if $M, E \cup E_0 \models P$ holds and the execution of $\mathtt{c}$ from s terminates
- then the execution terminates at a program state s', with corresponding evaluation E', such that $M, E' \cup E_0 \models Q$ holds.

Summary: Syntax and Semantics of Hoare Triples

- Syntax of Hoare Triples: $\{P\}$ `C` $\{Q\}$
  - P and Q are formulas in predicate logic
  - `C` is a code fragment in a simple programming language
  - Auxiliary variables are variables in P and Q that do not appear in `C`
- Semantics of Hoare Triples
  - Partial correctness: $\{P\}$ `C` $\{Q\}$ is valid if and only if

    For all initial states that satisfy the precondition P, if the execution of `C` terminates then the final state satisfies Q
  - Total correctness: $[P]$ `C` $[Q]$ is valid if and only if

    For all initial states that satisfy the precondition P, the execution of `C` terminates and the final state satisfies Q
- For partial correctness, the execution might not terminate even for input states that satisfy the precondition
- For total correctness, termination is guaranteed for all input states that satisfy the precondition

Extensions

Extensions of Hoare Logic

- Hoare Logic has been extended to support other features of programming languages

- Blocks and local variables
- Procedures
- Arrays
- Different loop constructs: `for (i = 1; i < n; i++) do C`
- Concurrency and parallel composition: `C1 || C2`
- ...
- Low-level imperative programs that manipulate pointers and shared mutable data structures

Pointers - Example

- $\{Q\}$ `*y := 4; *z := 5` $\{*y \neq *z\}$
  - contents of `y` and `z` are different
  - what precondition Q makes this triple valid?

Pointers - Example

- $\{Q\}$ `*y := 4; *z := 5` $\{*y \neq *z\}$
  - contents of `y` and `z` are different
  - what precondition Q makes this triple valid?
- $\{y \neq z\}$ `*y := 4; *z := 5` $\{*y \neq *z\}$

Pointers - Example

- $\{Q\}$ `*y := 4; *z := 5` $\{*y \neq *z\}$
  - contents of `y` and `z` are different
  - what precondition Q makes this triple valid?
- $\{y \neq z\}$ `*y := 4; *z := 5` $\{*y \neq *z\}$
- $\{y \neq z \land Q\}$ `*y := 4; *z := 5` $\{*y \neq *z \land *x = 3\}$
  - contents of `x` stay the same
  - what is the precondition Q to make this triple valid?

Pointers - Example

- $\{Q\}$ `*y := 4; *z := 5` $\{*y \neq *z\}$
  - contents of `y` and `z` are different
  - what precondition Q makes this triple valid?
- $\{y \neq z\}$ `*y := 4; *z := 5` $\{*y \neq *z\}$
- $\{y \neq z \wedge Q\}$ `*y := 4; *z := 5` $\{*y \neq *z \wedge *x = 3\}$
  - contents of `x` stay the same
  - what is the precondition Q to make this triple valid?
- $\{y \neq z \wedge *x = 3\}$ `*y := 4; *z := 5` $\{*y \neq *z \wedge *x = 3\}$
  - is this triple valid?

Pointers - Example

- $\{Q\}$ `*y := 4; *z := 5` $\{*y \neq *z\}$
  - contents of `y` and `z` are different
  - what precondition Q makes this triple valid?
- $\{y \neq z\}$ `*y := 4; *z := 5` $\{*y \neq *z\}$
- $\{y \neq z \wedge Q\}$ `*y := 4; *z := 5` $\{*y \neq *z \wedge *x = 3\}$
  - contents of `x` stay the same
  - what is the precondition Q to make this triple valid?
- $\{y \neq z \wedge *x = 3\}$ `*y := 4; *z := 5` $\{*y \neq *z \wedge *x = 3\}$
  - is this triple valid?
- Precondition on aliasing:

  $\{y \neq z \wedge *x = 3 \wedge x \neq y \wedge x \neq z\}$

  `*y := 4; *z := 5`

  $\{*y \neq *z \wedge *x = 3\}$

Framing Problem

- Example of the framing problem:

$$\frac{\{y \neq z\}\ \mathtt{c}\ \{*y \neq *z\}}{\{*x = 3 \wedge y \neq z\}\ \mathtt{c}\ \{*y \neq *z \wedge *x = 3\}}$$

  What are the conditions on aliasing between `x, y, z` ?

- Framing problem:

$$\frac{\{P\}\ \mathtt{c}\ \{Q\}}{\{R \wedge P\}\ \mathtt{c}\ \{Q \wedge R\}}$$

  What are the conditions on `c` and R in presence of aliasing and heap?

Framing Problem

- Example of the framing problem:

$$\frac{\{y \neq z\}\ \texttt{c}\ \{*y \neq *z\}}{\{*x = 3 \wedge y \neq z\}\ \texttt{c}\ \{*y \neq *z \wedge *x = 3\}}$$

What are the conditions on aliasing between `x, y, z` ?

- Framing problem:

$$\frac{\{P\}\ \texttt{c}\ \{Q\}}{\{R \wedge P\}\ \texttt{c}\ \{Q \wedge R\}}$$

What are the conditions on `c` and R in presence of aliasing and heap?

- Solution: Separation logic introduces a new connective $*$

$$\frac{\{P\}\ \texttt{c}\ \{Q\}}{\{R * P\}\ \texttt{c}\ \{Q * R\}}$$