Analysis for correctness

Engineering designs need to be analyzed to be sure that they will do the job required in practice.

It is not good engineering to observe that a bridge hasn't fallen down yet and conclude that it therefore is safe.

One needs to analyze the design in order to ensure that it will work in *all* circumstances.

Testing will ensure that a system will work in a finite set of circumstances — provided the system is known to be deterministic.

When a system may be nondeterministic, testing does not even ensure that the system will work in the finite set of circumstances tested.[1]

A single analysis can ensure that a system will work in all circumstances.

Luckily, we software engineers don't write code willy-niilly: We have some reason for believing that the code we write will work.

Such a reason is —at least embryonically— an analysis.

In this part of the course, we will look at one way to record our reasons for believing that code will work, in a way that makes subsequent analysis a fairly mechanical process. This way is called **proof outline logic**.

[Reading: Proof Outline Logic—Part 0.]

[1] We won't look at nondeterministic systems in this course. But in your course on concurrent programming, you will find that nondeterministic systems are the norm when dealing with concurrency.

Assertions

As I write programs, I often think about connections between the data state and the program counter.

"If execution gets to here, then i must be less than N, but not negative."

That sort of thing.

This is called an assertion.

A **condition** is a boolean expression with free variables from the state. For example, if i and N are state variables

$$0 \leq i < N$$

is a condition. For some states it is true. For others it is false.

An **assertion** is a condition that we expect to be true whenever execution reaches a particular point in the program.

Dynamic assertion checking

Java provides an **assert** statement that allows assertions to be checked dynamically (i.e. at run-time)[2]

```
assert 0 <= i && i < N ; a[i] = c ;
```

C and C++ provide the similar `assert` macro.

```
assert( 0 <= i && i < N ) ; a[i] = c ;
```

Python provides an assert command

```
assert 0 <= i and i < N ; a[i] = c
```

[2] However, in Java, assert statements are not checked by default. You are better off to write and use your own assert method(s).

Static assertion checking

Better yet, tools such as VCC, Spec#, Code Contracts, Dafny (all from Microsoft), Spark Ada, and Alloy allow assertions to be checked statically (i.e. before execution)

This is better, because

- dynamic checking checks that assertions are true only for states actually reached, but
- static checking checks that assertions are true in all reachable states

For this course, I will put assertions in braces like this:

$\{0 \le i < N\}$
$a[i] := c$

Contracts and correctness

Contracts

A **contract** is a pair of conditions.

$$(\mathcal{P}, \mathcal{R})$$

- $\mathcal{P}$ is called the precondition and
- $\mathcal{R}$ is called the postcondition.

Partial correctness

Defn: A command $\mathcal{C}$ is **partially correct** with respect to a contract $(\mathcal{P}, \mathcal{R})$ iff, whenever $\mathcal{C}$ is executed, starting in a state where $\mathcal{P}$ is true,

- no error occurs,
- $\mathcal{R}$ holds if and when the execution terminates.

Examples

$x := x + 1$ is partially correct w.r.t.

$$[0 \le x < 99,\ 0 < x < 100]$$

$\mathbf{val}\ t := x; x := y; y := t$ is partially correct w.r.t.

$$[x = 23 \wedge y = 42,\ y = 42 \wedge x = 23]$$

Specification variables and program variables

In these examples, x and y are *program variables*.

Over time they represent various values.

Specifications variables always represent the same value.

We'll use capital letters for these *specifications variables*.

For example,

$x := x + 1$ is partially correct w.r.t.

$$[A \leq x < Z,\ A < x < Z + 1]$$

$\mathbf{val}\ t := x; x := y; y := t$ is partially correct w.r.t.

$$[x = X \wedge y = Y,\ y = X \wedge x = Y]$$

Partially correct means partially correct for all values of the specification variables.

Proof outlines

A proof outline is a (possibly compound) command

- preceded by an assertion,
- followed by an assertion, and
- with internal assertions, so that every subcommand is preceded by an assertion

For example

```
$\{x = X \wedge y = Y\}$
**if** $x > y$ **then**
    $\{x = X \wedge y = Y \wedge x > y\}$
    $m := x$
**else**
    $\{x = X \wedge y = Y \wedge x \le y\}$
    $m := y$
**end if**
$\{m = \max(X, Y)\}$
```

When executing a proof outline, assertions are treated as comments.

A syntax for proof outlines

$\mathcal{V}$ — variables. $\mathcal{E}$— expressions. $\mathcal{C}$ —commands. $\mathcal{A}$—conditions. $\mathcal{O}$—outlines.

There are 2 kinds of simple commands

$$\begin{array}{lll} \mathcal{C} \rightarrow \mathcal{V} := \mathcal{E} & \text{Assignment} \\ \mathcal{C} \rightarrow \mathbf{skip} & \text{Skip (do nothing)} \end{array}$$

There are 4 ways to make compound commands

$$\begin{array}{ll} \mathcal{C} \rightarrow \mathbf{if}\ \mathcal{E}\ \mathbf{then}\ \{\mathcal{A}\}\ \mathcal{C}\ \mathbf{else}\ \{\mathcal{A}\}\ \mathcal{C}\ \mathbf{end\ if} & \text{Alternation} \\ \mathcal{C} \rightarrow \mathbf{if}\ \mathcal{E}\ \mathbf{then}\ \{\mathcal{A}\}\ \mathcal{C}\ \mathbf{end\ if} & \text{Alternation} \\ \mathcal{C} \rightarrow \mathbf{while}\ \mathcal{E}\ \mathbf{do}\ \{\mathcal{A}\}\ \mathcal{C}\ \mathbf{end\ while} & \text{Iteration} \\ \mathcal{C} \rightarrow \mathcal{C}\ \{\mathcal{A}\}\ \mathcal{C} & \text{Sequential composition} \end{array}$$

Finally there is one rule for proof outlines

$$\mathcal{O} \rightarrow \{\mathcal{A}\}\ \mathcal{C}\ \{\mathcal{A}\} \quad \text{Proof outline}$$

Note that, in a proof outline, every command is preceded by an assertion.

If $\{\mathcal{P}\}\ \mathcal{S}\ \{\mathcal{R}\}$ is a proof outline,

- $\mathcal{P}$ is called its **precondition**
- $\mathcal{Q}$ is called its **postcondition**
- Assertions within $\mathcal{S}$ are called its **internal assertions**.

Correctness of proof outlines

Defn: A proof outline $\{\mathcal{P}\}\ \mathcal{C}\ \{\mathcal{R}\}$ is **partially correct** iff, whenever command $\mathcal{C}$ is executed, beginning in a state where $\mathcal{P}$ holds,

- no error occurs,
- each internal assertion of $\mathcal{C}$ holds each time it is reached and
- $\mathcal{R}$ holds if and when the execution terminates.

(for all values of the specification variables).

Notes:

- There is no requirement that $\mathcal{C}$, executed beginning in a state where $\mathcal{P}$ holds, must terminate. We will treat termination as a separate concern.
- If a proof outline $\{\mathcal{P}\}\ \mathcal{C}\ \{\mathcal{R}\}$ is partially correct, then the command $\mathcal{C}$ is partially correct with respect to contract $[\mathcal{P}, \mathcal{R}]$.

Some partially correct proof outlines

(a)

```
$\{x = X \wedge y = Y \wedge x \leq y\}$
$m := y$
$\{m = \max(X, Y)\}$
```

is a partially correct proof outline.

(b)

```
$\{x = X \wedge y = Y\}$
**if** $x > y$ **then**
    $\{x = X \wedge y = Y \wedge x > y\}$
    $m := x$
**else**
    $\{x = X \wedge y = Y \wedge x \leq y\}$
    $m := y$
**end if**
$\{m = \max(X, Y)\}$
```

is a partially correct proof outline.

(c) For this example, assume that x and y can hold any integer; there is no overflow.

```
$\{X = x \geq 0\}$
$y := 1$
$\{\mathcal{I} : x \geq 0 \wedge y \times 2^x = 2^X\}$
**while** $x > 0$ **do**
    $\{x > 0 \wedge y \times 2^x = 2^X\}$
    $x := x - 1$
    $\{x \geq 0 \wedge 2 \times y \times 2^x = 2^X\}$
    $y := 2 \times y$
**end while**
$\{y = 2^X\}$
```

Note that the assertion labelled $\mathcal{I}$ is considered to *be reached* not only before the first iteration of the loop, *but also after every iteration of the loop*.

So we need to check that $\mathcal{I}$ will be true when the loop is first reached and after every iteration.

(d) For this example, assume that i can hold *any* integer. There is no overflow

```
{$\text{true}$}
**while** $i \neq 0$ **do**
    {$i \neq 0$}
    $i := i - 1$
**end while**
{$i = 0$}
```

This is a partially correct proof outline — even though the loop may not terminate.

Some proof-outlines that are not partially correct

(e) For this example, assume that x can hold any integer; there is no overflow

$\{x = X\}$
$x := x^2$
$\{x > X\}$

For some executions, both the pre- and postcondition are true. However there are some executions for which the precondition is true, yet the postcondition is false — for example, when $X = 0$ initially.

This is not a partially correct proof outline.

(f) For this example, assume that x can hold any integer; there is no overflow

$\{x = X\}$
$x := 2x$
$\{x = 3X\}$
$x := x + 1$
$\{x = 2X + 1\}$

The internal assertion might not be true when it is reached.

This is not a partially correct proof outline.

(g) For this example assume that i and s can hold any integer; there is no overflow.

```
$\{0 \le N \le \mathrm{length}(a)\}$
$i := 0$
$\{i = 0 \le N \le \mathrm{length}(a)\}$
$s := 0$
$\left\{0 \le i \le N \le \mathrm{length}(a) \wedge s = \sum_{k \in \{0,..i\}} a(i)\right\}$
**while** $i < N$ **do**
    $\left\{0 \le i < N \le \mathrm{length}(a) \wedge s = \sum_{k \in \{0,..i\}} a(i)\right\}$
    $s := s + a(i)$
**end while**
$\left\{i = N \wedge s = \sum_{k \in \{0,..i\}} a(i)\right\}$
```

This is not a partially correct proof outline. We'll return to this example later.

From here on, I'll use "partially correct" and "**correct**" interchangeably.

Checking correctness

Universally true

Defn: A boolean expression $\mathcal{P}$ **is universally true** iff it evaluates to true for *any* assignment of values (of the appropriate type) to its free variables. Suppose x is a variable of type $\mathbb{Z}$.

- The expression $x < 1 \Rightarrow x < 3$ is
- The expression $0 < x < 2 \Rightarrow x = 1$ is
- The expression $x < 5 \Rightarrow x < 3$ is

Verification conditions

Our goal will be to reduce checking the correctness of a proof outline to checking the universal trueness of boolean expressions.

If we can do that, we have reduced the problem of checking correctness of outlines to *conventional mathematics*.

The boolean expressions that come out of this analysis process are called **verification conditions**.

$$\begin{tikzcd} \boxed{\text{A proof outline}} \arrow[d, "\text{Analysis rules}"] \\ \boxed{\text{Verification conditions}} \arrow[d, "\text{Prove universally true}"] \\ \boxed{\text{yes/no}} \end{tikzcd}$$

Assignment commands

(Aside: We ignore errors for now. See next slide set.)

Let's look at a few assignment commands.

Example

$$\{y > 4\}\ x := 2\ \{y > x\}$$

For this outline to be correct, we need that for any initial state such that $y > 4$ holds, execution will end in a state where $y > x$ holds.

Since x will be 2 in the final state, we need that for any initial state such that $y > 4$ holds execution will end is a state where $y > 2$ holds.

Since y will be the same in the initial and final states, we need that for any state where $y > 4$ holds, $y > 2$ also holds.

This is true as $y > 4 \Rightarrow y > 2$ is .

Example

Now consider an arbitrary precondition $\mathcal{P}$

$$\{\mathcal{P}\}\ x := 2\ \{y > x\}$$

We need that, prior to the assignment, if $\mathcal{P}$ is true, $y > 2$ will also be true. I.e. that

Example

$$\{\mathcal{P}\}\ x := x+1\ \{y > x\}$$

We need that prior to the assignment, if $\mathcal{P}$ is true then $\boxed{}$ will also be true. I.e., that

$$\boxed{}$$

The general case

Consider any outline

$$\{\mathcal{P}\}\ \mathcal{V} := \mathcal{E}\ \{\mathcal{R}\}$$

For $\mathcal{R}$ to be true after an assignment $\mathcal{V} := \mathcal{E}$, we need that $\mathcal{R}$, with $\mathcal{V}$ replaced by $\mathcal{E}$, be true before the assignment.

Recall that $\mathcal{R}[\mathcal{V} : \mathcal{E}]$ means a formula just like $\mathcal{R}$ except with all (free) occurrences of $\mathcal{V}$ replaced by $\mathcal{E}$.

We'll call $\mathcal{R}[\mathcal{V} : \mathcal{E}]$ "the substituted postcondition".

You can think of $\mathcal{R}[\mathcal{V} : \mathcal{E}]$ as the projection of $\mathcal{R}$ into the initial state.

The precondition of an assignment can be any formula that always implies $\mathcal{R}[\mathcal{V} : \mathcal{E}]$.

The assignment rule:

If $\mathcal{P} \Rightarrow \mathcal{R}[\mathcal{V} : \mathcal{E}]$ is universally true
then $\{\mathcal{P}\}\ \mathcal{V} := \mathcal{E}\ \{\mathcal{R}\}$ is correct.

Why? Well if $\mathcal{P} \Rightarrow \mathcal{R}[\mathcal{V} : \mathcal{E}]$ is universally true and $\mathcal{P}$ is true before executing the assignment,

- then $\mathcal{R}[\mathcal{V} : \mathcal{E}]$ will be true before executing the assignment, and so
- $\mathcal{R}$ will be true after executing the assignment.

Example

For this example, x and y are real variables.

What needs to be checked to ensure that

$\{x < 0\}\ x := x + y\ \{x < y\}$ is correct?

Example

For this example, a, b and m are integer variables.

What is the weakest condition $\mathcal{P}$ so that

$\{\mathcal{P}\}\ m := \left\lfloor \dfrac{a+b}{2} \right\rfloor\ \{a < m < b\}$ is correct?

(The brackets $\lfloor x \rfloor$ mean the floor of x. I.e. the largest integer not larger than x.)

Can we simplify it?

Here are some useful facts about floor that might help.
For all integers a and b

$$(a < \lfloor x \rfloor) = (a + 1 \leq x)$$
$$(\lfloor x \rfloor < b) = (x < b)$$

Some fine print

One expression is an **alias** of another expression if they both represent the same memory location. E.g. references in C++ are often aliases, as are expressions such as $*q$.

When aliasing is a possibility, we need to consider also what variables might be aliased by the expression on the left-hand side of the assignment.

For example, in C, for

$\{\mathcal{P}\}\ *q = 1;\ \{*q == 1 \wedge x == 0\}$ to be correct

requires that $\mathcal{P}$ not only imply $x == 0$, but also $q\ != \ \&x$.

Similarly in Java

$\{\mathcal{P}\}\ a.x = 1;\ \{a.x == 1 \wedge b.x == 0\}$ to be correct

requires that $\mathcal{P}$ not only imply $b.x == 0$, but also $a\ != \ b$.

Aliasing complicates the assignment rule. We won't consider it further.

Other commands

For each kind of command, we can state an inference rule

The skip command

Since skip doesn't do anything the postcondition must be true initially. Thus it should be (universally) implied by the precondition.

The skip rule

If $\mathcal{P} \Rightarrow \mathcal{R}$ is universally true
then $\{\mathcal{P}\}\ \mathbf{skip}\ \{\mathcal{R}\}$ is correct.

Omitting skip

For brevity, we sometimes leave out skip commands. The rule becomes

If $\mathcal{P} \Rightarrow \mathcal{R}$ is universally true
then $\{\mathcal{P}\}\ \{\mathcal{R}\}$ is correct.

Sequential composition

The sequential composition rule.

For sequential composition of two statements, we have

If	$\{\mathcal{P}\}\ \mathcal{S}\ \{\mathcal{Q}\}$ is correct
and	$\{\mathcal{Q}\}\ \mathcal{T}\ \{\mathcal{R}\}$ is correct
then	$\{\mathcal{P}\}\ \mathcal{S}\ \{\mathcal{Q}\}\ \mathcal{T}\ \{\mathcal{R}\}$ is correct.

Example

$$\{x = X \wedge y = Y\}\ t := x\ \{t = X \wedge y = Y\}$$

is correct,

$$\{t = X \wedge y = Y\}\ x := y\ \{t = X \wedge x = Y\}$$

is correct, and

$$\{t = X \wedge x = Y\}\ y := t\ \{y = X \wedge x = Y\}$$

is correct; therefore

$$\begin{array}{l} \{x = X \wedge y = Y\} \\ t := x \\ \{t = X \wedge y = Y\} \\ x := y \\ \{t = X \wedge x = Y\} \\ y := t \\ \{y = X \wedge x = Y\} \end{array}$$

is correct.

Alternation

The alternation rule (2-branches)

If $\mathcal{P} \wedge \mathcal{E} \Rightarrow \mathcal{Q}_0$ is universally true,
$\mathcal{P} \wedge \neg\mathcal{E} \Rightarrow \mathcal{Q}_1$ is universally true,
$\{\mathcal{Q}_0\}\ \mathcal{S}\ \{\mathcal{R}\}$ is correct,
and $\{\mathcal{Q}_1\}\ \mathcal{T}\ \{\mathcal{R}\}$ is correct
then $\{\mathcal{P}\}$ **if** $\mathcal{E}$ **then** $\{\mathcal{Q}_0\}\ \mathcal{S}$ **else** $\{\mathcal{Q}_1\}\ \mathcal{T}$ **end if** $\{\mathcal{R}\}$ is correct.

Example: Is this outline correct?

```
{ $x = X \wedge y = Y$ }
if $x > y$ then
    { $x = X \wedge y = Y \wedge x > y$ }
    $m := x$
else
    { $x = X \wedge y = Y \wedge x \leq y$ }
    $m := y$
end if
{ $m = \max(X, Y)$ }
```

We need to show the following

-
-
-
-

The alternation rule (1-branch)

If $\mathcal{P} \wedge \mathcal{E} \Rightarrow \mathcal{Q}$ is universally true,
$\mathcal{P} \wedge \neg\mathcal{E} \Rightarrow \mathcal{R}$ is universally true,
and $\{\mathcal{Q}\}\ \mathcal{S}\ \{\mathcal{R}\}$ is correct,
then $\{\mathcal{P}\}\ \mathbf{if}\ \mathcal{E}\ \mathbf{then}\ \{\mathcal{Q}\}\ \mathcal{S}\ \mathbf{end\ if}\ \ \{\mathcal{R}\}$ is correct.

Iteration

We can think of an iteration

$$\{\mathcal{P}\}\ \mathbf{while}\ \mathcal{E}\ \mathbf{do}\ \{\mathcal{Q}\}\ \mathcal{S}\ \mathbf{end\ while}\ \{\mathcal{R}\}$$

as being equivalent to its infinite unrolling — remember that the backward jump after $\mathcal{S}$ is considered to land at the start of $\mathcal{P}$.

```
$\{\mathcal{P}\}$
 if $\mathcal{E}$
 then $\{\mathcal{Q}\}\ \mathcal{S}\ \{\mathcal{P}\}$  if $\mathcal{E}$
                    then $\{\mathcal{Q}\}\ \mathcal{S}\ \{\mathcal{P}\}$  if $\mathcal{E}$
                                       then $\{\mathcal{Q}\}\ \mathcal{S}\ \{\mathcal{P}\}$
                                                          $\ddots$
                                       end if
                    end if
 end if
$\{\mathcal{R}\}$
```

The iteration rule

If $\mathcal{P} \wedge \mathcal{E} \Rightarrow \mathcal{Q}$ is universally true,
$\mathcal{P} \wedge \neg\mathcal{E} \Rightarrow \mathcal{R}$ is universally true
and $\{\mathcal{Q}\}\ \mathcal{S}\ \{\mathcal{P}\}$ is correct,
then $\{\mathcal{P}\}\ \mathbf{while}\ \mathcal{E}\ \mathbf{do}\ \{\mathcal{Q}\}\ \mathcal{S}\ \mathbf{end\ while}\ \{\mathcal{R}\}$ is correct.

The precondition of an iteration command is called **a loop invariant**.

Understanding the loop invariant is often the key to understanding an algorithm.

Therefore documenting loop invariants is highly advisable.

Example

Above I claimed that, with i able to hold any integer (no overflow),

```
{true}
while $i \neq 0$ do
    {$i \neq 0$}
    $i := i - 1$
end while
{$i = 0$}
```

is correct.

From the iteration rule, we see that, to show this proof outline is correct, it is sufficient to show the following:

1.
2.
3.

The formulae in 1 and 2 can easily be simplified to true and thus are universally true.

From the assignment rule, the proof outline in 3 is correct if

Recall that: *correct does not imply termination.*

Example

For this example, $a\{0,..i\}$ means the set of the first i items of array a.

```
{true} $i := 0$ {$i = 0$} $f :=$ false
{$\mathcal{I} : 0 \le i \le a.\text{length} \wedge f = (t \in a\{0,..i\})$}
**while** $\mathcal{G} : i \ne a.\text{length} \wedge \neg f$ **do**
    {$\mathcal{A} : 0 \le i < a.\text{length} \wedge t \notin a\{0,..i\}$}
    $f := t = a(i)$
    {$\mathcal{B} : 0 \le i + 1 \le a.\text{length} \wedge f = (t \in a\{0,..i+1\})$}
    $i := i + 1$
**end while**
{$\mathcal{R} : f = (t \in a\{0,..a.\text{length}\})$}
```

To show this is correct, it is sufficient to show:

1. From $i := 0$: That
2. From $f :=$ false: That
3. From the loop that the following are universally true
   a.
   b.
   c. and the following proof outline is correct:

   i.e., that and are both univ. true.

The word analyze literally means to 'break up'.

We have broken up a single complex (and computational) question ("Is this algorithm correct?") into 6 simple (and noncomputational) questions:

- Is $\text{true} \Rightarrow ((i = 0)\,[i : 0])$ i.e. $\text{true} \Rightarrow (0 = 0)$ univ. tr.?
- Is $i = 0 \Rightarrow \mathcal{I}[f : \text{false}]$, i.e.
  $$i = 0 \Rightarrow 0 \le i \le a.\text{length} \wedge \text{false} = (t \in a\,\{0, ..i\}) \quad ,$$
  universally true?
- Is $\mathcal{I} \wedge \mathcal{G} \Rightarrow \mathcal{A}$, i.e.,
  $$\begin{array}{rl} & 0 \le i \le a.\text{length} \wedge f = (t \in a\,\{0, ..i\}) \\ & \wedge\ i \ne a.\text{length} \wedge \neg f \\ \Rightarrow & 0 \le i < a.\text{length} \wedge t \notin a\,\{0, ..i\} \quad , \end{array}$$
  universally true?
- Is $\mathcal{I} \wedge \neg\mathcal{G} \Rightarrow \mathcal{R}$, i.e.,
  $$\begin{array}{rl} & 0 \le i \le a.\text{length} \wedge f = (t \in a\,\{0, ..i\}) \\ & \wedge\ \neg\,(i \ne a.\text{length} \wedge \neg f) \\ \Rightarrow & f = (t \in a\,\{0, ..a.\text{length}\}) \quad , \end{array}$$
  universally true?
- Is $\mathcal{A} \Rightarrow \mathcal{B}[f : t = a(i)]$, i.e.,
  $$\begin{array}{rl} & 0 \le i < a.\text{length} \wedge t \notin a\,\{0, ..i\} \\ \Rightarrow & 0 \le i + 1 \le a.\text{length} \wedge (t = a(i)) = (t \in a\,\{0, ..i + 1\}) \quad , \end{array}$$
  universally true?
- Is $\mathcal{B} \Rightarrow \mathcal{I}[i : i + 1]$, i.e.,
  $$\begin{array}{rl} & 0 \le i + 1 \le a.\text{length} \wedge f = (t \in a\,\{0, ..i + 1\}) \\ \Rightarrow & 0 \le i + 1 \le a.\text{length} \wedge f = (t \in a\,\{0, ..i + 1\}) \quad , \end{array}$$
  universally true?

A way to think about iteration

(Credit to Jeff Edmonds and his book on Algorithms.)

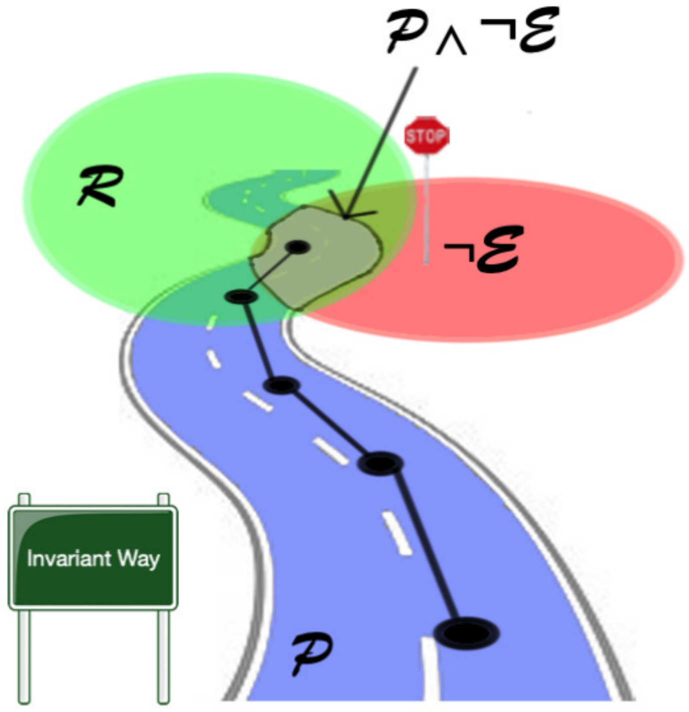
$$\{\mathcal{P}\}\ \textbf{while}\ \mathcal{E}\ \textbf{do}\ \{\mathcal{Q}\}\ \mathcal{S}\ \textbf{end while}\ \{\mathcal{R}\}$$

Suppose we want to reach some goal $\mathcal{R}$.

We know that $\mathcal{R}$ intersects a road $\mathcal{P}$.

To get to $\mathcal{R}$, we do the following.

- Get on to the road $\mathcal{P}$.
- Stop when a condition $\neg\mathcal{E}$ holds, where $\neg\mathcal{E} \wedge \mathcal{P} \Rightarrow \mathcal{R}$ is universally true.
- Otherwise, if $\mathcal{E}$ holds, move forward without leaving the road.



(clipart courtesy of clipart.email)

Provably Correct

Defn: A proof outline $\{\mathcal{P}\}\ \mathcal{S}\ \{\mathcal{R}\}$ is **provably partially correct** iff it can be shown to be correct using the inference rules to produce a set of verification conditions and the fact that the resulting verification conditions are universally true.

Example: x is a mathematical integer (so no overflow).

$\{x > 0\}$
$x := x + 1$
$\{x > 0\}$
$x := x + 1$
$\{x > 2\}$

This is partially correct, but it is not *provably* correct because our inference rules for composition and assignment yield the following verification conditions

$$x > 0 \Rightarrow x + 1 > 0$$
$$x > 0 \Rightarrow x + 1 > 2$$

and the second one is not universally true.

Another example: Here j is an integer variable

```
$\{len(A) = N\}$
$j := 0$
$\{\mathfrak{true}\}$
$s := 0$
$\{\mathfrak{true}\}$
while $j \neq N$ do
    $\{\mathfrak{true}\}$
    $s := s + A(j)$
    $j := j + 1$
end while
$\left\{s = \sum_{k \in \{0,..N\}} A(k)\right\}$
```

Exercise: Consider this strange proof outline.

```
$\{\mathfrak{true}\}$
$x := 5$
$\{0 \le x < 15\}$
while $x \ne 0$ do
    $\{0 < x < 15\}$
    if $x < 10$ then $\{0 < x < 10\}$ $x := x - 1$
    elsif $10 \le x < 20$ then $\{10 \le x < 15\}$ $x := x + 1$
    elsif $20 \le x$ then $\{\mathfrak{false}\}$ $x := 0$
    end if
end while
$\{0 \le x < 15\}$
```

Explain why it is correct.

Show it is not provably correct.

Show that you can change the internal assertions to make it provably correct.

Hint: either of these invariants can be made to work

$$0 \le x < 10$$
$$0 \le x$$

One is stronger than the original. One is weaker.

Omitting assertions

When internal assertions can be easily filled in, we may omit them.

Composition

Given and incomplete outline $\{\mathcal{P}\}\ \mathcal{S}\ \mathcal{T}\ \{\mathcal{R}\}$, you can always fill in the missing assertion with the weakest assertion $\mathcal{Q}$ such that

$$\{\mathcal{Q}\}\ \mathcal{T}\ \{\mathcal{R}\} \text{ is correct}$$

If $\mathcal{T}$ is an assignment, this will be the substituted postcondition.

Example: Consider

$$\{\mathcal{P}\}\ t := x\ ;\ x := y\ ;\ y := t\ \{\mathcal{R}\}$$

The weakest precondition $\mathcal{Q}_1$ so that

$$\{\mathcal{Q}_1\}\ y := t\ \{\mathcal{R}\} \text{ is correct}$$

is

$$\mathcal{R}[y : t]$$

The weakest precondition $\mathcal{Q}_0$ so that

$$\{\mathcal{Q}_0\}\ x := y\ \{\mathcal{R}[y : t]\}$$

is correct is

$$(\mathcal{R}[y : t])\,[x : y]$$

So

$$\{\mathcal{P}\}\ t := x\ ;\ x := y\ ;\ y := t\ \{y = X \wedge x = Y\}$$

is provably correct if

$$\mathcal{P} \Rightarrow ((\mathcal{R}[y : t])\,[x : y])\,[t : x] \text{ is universally true.}$$

Note the *reversed* sequence of substitutions into the postcondition.

In general

$\{\mathcal{P}\}\ \mathcal{V} :\ = \mathcal{E}\ \mathcal{W} := \mathcal{F}\ \{\mathcal{R}\}$ is correct if

$\mathcal{P} \Rightarrow (\mathcal{R}[\mathcal{W} : \mathcal{F}])\,[\mathcal{V} : \mathcal{E}]$ is universally true

Iteration

$\{\mathcal{P}\}$ while $\mathcal{E}$ do $\mathcal{S}$ end while $\{\mathcal{R}\}$

We can fill in the missing assertion like this

$\{\mathcal{P}\}$ while $\mathcal{E}$ do $\{\mathcal{P} \wedge \mathcal{E}\}$ $\mathcal{S}$ end while $\{\mathcal{R}\}$

Alternation

$\{\mathcal{P}\}$ if $\mathcal{E}$ then $\mathcal{S}$ else $\mathcal{T}$ end if $\{\mathcal{R}\}$

We can fill in the missing assertions with

$\{\mathcal{P}\}$ if $\mathcal{E}$ then $\{\mathcal{P} \wedge \mathcal{E}\}$ $\mathcal{S}$ else $\{\mathcal{P} \wedge \neg\mathcal{E}\}$ $\mathcal{T}$ end if $\{\mathcal{R}\}$

Summary

If putting in an internal assertion helps make it clear why an outline is correct, then do so.

If putting in the intermediate assertion obscures clarity, leave it out.

Don’t omit loop invariants.