

# 1 Assignment 2.5

## 1.1 Exercises 1

Consider the program  $C$  and pre- and postconditions  $F$  and  $H$  as follows:

- Program  $C$ :  $x = 2$ ;
- Precondition  $F$ :  $x > 0$
- Postcondition  $H$ :  $x > 1$

A counterexample for Rule A:

Let  $G: x > 2$ , using Rule A, if we apply it to the triple  $\vdash \{\{ F \} \} C \{\{ H \} \}$ , we get:

$$\frac{\vdash \{\{ x > 0 \} \} x = 2 \{\{ x > 1 \} \}}{\vdash \{\{ x > 0 \wedge x > 2 \} \} x = 2 \{\{ x > 1 \wedge x > 2 \} \}} \text{ruleA}$$

We have:  $\models \{\{ x > 0 \} \} x = 2 \{\{ x > 1 \} \}$ , which is true. However, simplifying the conditions, we have  $\vdash \{\{ x > 2 \} \} x = 2 \{\{ x > 2 \} \}$ , the postcondition  $x > 2$  is not satisfied by the program, since after execution, we have  $x = 2$  which doesn't satisfy the condition.

A Counterexample for Rule B:

Let  $a = x+1$ , using Rule B, if we apply it to the triple  $\vdash \{\{ F \} \} C \{\{ H \} \}$ , we get:

$$\frac{\vdash \{\{ x > 0 \} \} x = 2 \{\{ x > 1 \} \}}{\vdash \{\{ x > 1 \} \} x = 2 \{\{ x > 2 \} \}} \text{ruleA}$$

We have:  $\models \{\{ x > 0 \} \} x = 2 \{\{ x > 1 \} \}$ , which is true. However, simplifying the conditions, we have  $\vdash \{\{ x > 1 \} \} x = 2 \{\{ x > 2 \} \}$ , the postcondition  $x > 2$  is not satisfied by the program, since after execution, we have  $x = 2$  which doesn't satisfy the condition.

In both cases, we have shown that applying the proposed rules can lead to incorrect conclusions, which demonstrates their unsoundness.

## 1.2 Exercises 2

To show that we can verify all triples  $\{\{ F \} \} C \{\{ H \} \}$  if we know how to verify triples of the form  $\{\{ \text{true} \} \} C' \{\{ \text{true} \} \}$ , we'll construct a command  $C'$ .

Let's consider the verification conditions:

1. For  $\{\{ \text{true} \} \} C' \{\{ \text{true} \} \}$ , we have  $\text{true} \rightarrow \text{wp}[C'](\text{true})$ , which simplifies to  $\text{true} \rightarrow \text{true}$ , which is trivially true.
2. For  $\{\{ F \} \} C \{\{ H \} \}$ , we have  $F \rightarrow \text{wp}[C](H)$ .

Now, we want to construct  $C'$  such that:

$$\vdash \{\{ \text{true} \} \} C' \{\{ \text{true} \} \} \iff \vdash \{\{ F \} \} C \{\{ H \} \}$$

The idea is to make  $C'$  a sequential composition of two commands:

$$C' = \text{assume } F; C; \text{assert } H$$

Now, let's justify why this construction is correct:

1.  $\Rightarrow$  (Forward direction)

Assume  $\vdash \{\{ \text{true} \} \} C' \{\{ \text{true} \} \}$ . This means  $\text{true} \rightarrow \text{wp}[C'](\text{true})$ .

By definition of weakest precondition,  $\text{wp}[C'](\text{true})$  is the weakest formula that holds before the execution of  $C'$  and guarantees that  $\text{true}$  holds after the execution of  $C'$ .

Since  $\text{true}$  always holds, this means that  $\text{wp}[C'](\text{true})$  must also always hold.

Therefore,  $F \rightarrow \text{wp}[C](H)$ , as required.

2.  $\Leftarrow$  (Backward direction)

Assume  $\vdash \{\{ F \} \} C \{\{ H \} \}$ . This means  $F \rightarrow \text{wp}[C](H)$ .

We want to show that  $\vdash \{\{ \text{true} \} \} C' \{\{ \text{true} \} \}$ , i.e.,  $\text{true} \rightarrow \text{wp}[C'](\text{true})$ .

By definition of weakest precondition,  $\text{wp}[C'](\text{true})$  is the weakest formula that holds before the execution of  $C'$  and guarantees that  $\text{true}$  holds after the execution of  $C'$ .

The command  $C'$  is constructed such that it first assumes  $F$ , then executes  $C$ , and finally asserts  $H$ .

This means that the weakest formula that holds before the execution of  $C'$  is  $F$ . And since  $\text{true}$  always holds, the condition  $\text{true}$  holds after the execution of  $C'$ .

Therefore,  $\text{true} \rightarrow \text{wp}[C'](\text{true})$ , as required.

In conclusion, by constructing  $C'$  as `assume  $F$ ;  $C$ ; assert  $H$` , we have shown that we can verify all triples  $\{\{ F \} \} C \{\{ H \} \}$  if we know how to verify triples of the form  $\{\{ \text{true} \} \} C' \{\{ \text{true} \} \}$ .

### 1.3 Exercises 3

```
method Example1() {
  var x: Int
  var y: Int

  assume x == y
  x := y + 1
  assert x > y
}
```

In this example, the program assumes that  $x$  and  $y$  are equal, assigns  $x$  the value  $y + 1$ , and asserts that  $x$  is greater than  $y$ . The program is functional, partial and total correct.

```
method Example2() {
  var x: Int
  assert x > 0
}
```

In this example, the program asserts that  $x$  is greater than 0. The program is functional correct, but not partial and total correct.

```
method Example3() {
  var x: Int

  while (true) {
```

```

    x := 1
  }
  assert x == 0
}

```

In this example, the program starts with a while loop, finally asserts that  $x$  equals to 0. The program is functional and partial correct, but not total correct.

Example 1 and 3 pass Viper verification, while the second one failed, which means Viper verifies programs with respect to partial correct.

## 1.4 Exercises 4

Now, we can define the safe transformer as follows if our command contains assert  $F$ :

$\text{safe}[C](F) = \text{sp}[C](\text{safe}[C'](F))$  where  $C'$  is the remaining portion of the command  
if  $C = C1;C2;C3...$

The key idea is to recursively apply the strongest postcondition transformer to calculate the logical formula  $\text{safe}[C](F)$  that represents the safety condition for running command  $C$  with precondition  $F$ .