

# OHI-2506 Program Verification: Exercises

Antero Kangas

Tampere University of Technology

Department of Software Systems

December 5, 2011

## Accomplishment

- Exercises are obligatory and probably the grades will be determined by the exercises.
- That will be decided during the first lectures.
- The course is valid for post-graduate studies.

## Exercises are given on the following way

- In the beginning of an exercise the instructor calls the names and asks who have done which problems, and selects the presenter.
- The precedence is usually given to him who have least presentations, so far. But there can be exceptions.
- If, while presentation, there reveals error(s) in the solution, then
  1. the presenter may try to fix it himself, with the instructors help if necessary,
  2. the problem is given to the presenter as a personal task for the next time, or
  3. another student can be asked to present his solution.
- Be prepared to present your answers. The grade is (or may be) given by the exercises. It is better to solve fewer problems carefully than many “poorly”.
- Although the grade is not determined by the number of the problems you have solved, but it is not possible to show what you have learned if you solve only few problems.

## Exercise group 1

### Preparatory problems

The idea is to show in which kind of programming problems the program verification techniques are useful. We are going to redo some of them after we have learned new means.

1. The following implementation of quicksort algorithm is used as an example in a C++ textbook:

```

1      void qsort (int *ia, int low, int high) {
2          if (low < high ) {
3              int lo = low;
4              int hi = high + 1;
5              int elem = ia [low]; // pivot
6              for (;;) {
7                  while (ia [++lo] < elem);
8                  while (ia [--hi] > elem);
9                  if (lo < hi)
10                     swap (ia, lo, hi);
11                     else break;
12             }
13             swap (ia, low, hi);
14             qsort (ia, low, hi - 1);
15             qsort (ia, hi + 1, high);
16         } // end, if (low < high)
17     }
```

- (a) The implementation has an error that the author seems not to have noticed. What?
  - (b) Estimate verbally the possibility to find the error by testing. Estimate also the possibility that the error never appears when the program is used.
2. Before beginning of the following binary search program for array  $A[1 \dots n]$  it holds that  $A[1] \leq A[2] \leq \dots \leq A[n]$ . The program tries to find the greatest  $a$  such that  $A[i] < key$  whenever  $1 \leq i < a$ .

```

1      a := 1; y := n;
2      while a < y do
3          v := (a + y) div 2;
4          if A[v] < key then a := v + 1;
5          else y := v
6          endif
7      endwhile
```

- (a) Find the error.
- (b) Estimate the probability that the error appears in  $m$  tests or production runs, if  $A[1], \dots, A[n]$  all are non-equal and  $key$  is one of them.

- (c) Like (b), but now all  $A[1], \dots, A[n]$ , and  $key$  are non-equal.
3. Does the following program find the longest proper ascending sequence  $B[1 \dots k]$  that is created from  $A[1 \dots n]$  by removing elements? Reason or give a counterexample.

```

1      k := 0;
2      for i := 1 to n do
3          h := 1; C[1] := A[i];
4          for j := i + 1 to n do
5              if A[j] > C[h] then h := h + 1; C[h] := A[j]
6              endif
7          endfor
8          if h > k then
9              k := h;
10             for j := 1 to k do B[j] := C[j] endfor
11         endif
12     endfor

```

4. Is it possible to find out the *height of vertex*  $u$  (= the longest length of a path from it to a vertex without any child-vertices) by using the following algorithms for (a) a tree (b) a directed loopless graph (c) a directed graph? The starting edges of the vertex are in linked list *outedges*. In the beginning all vertices have  $m = -1$ . Give a reasoning or a counterexample.

```

1      function Height1(u) is
2      if u↑.m = -1 then
3          u↑.m := 0; a := u↑.outedges;
4          while a ≠ Nil do
5              x := Height1(a↑.head); a := a↑.next;
6              if x ≥ u↑.m then u↑.m := x + 1 endif
7          endwhile
8      endif
9      return u↑.m

```

```

1      function Height2(u) is
2      if u↑.m = -1 then
3          k := 0; a := u↑.outedges;
4          while a ≠ Nil do
5              x := Height2(a↑.head); a := a↑.next;
6              if x ≥ k then k := x + 1 endif
7          endwhile
8          u↑.m := k;
9      endif
10     return u↑.m

```

## Repetition of propositional logics

5. Building a binary operator  $\star$  requires creating such an expression  $l(P, Q)$  that  $P \star Q \Leftrightarrow l(P, Q)$  for all combinations of truth values of  $P$  and  $Q$ . This definition generalises also for other operators than just for binary operators.

In principle  $\vee$  and  $\neg$  suffice for the basic operators of propositional logic, since other operators `False`, `True`,  $\wedge$ ,  $\rightarrow$ , and  $\leftrightarrow$  can be build using them if we have at least one propositional symbol. E.g.  $P \wedge Q \Leftrightarrow \neg(\neg P \vee \neg Q)$ , and `True`  $\Leftrightarrow P \vee \neg P$ .

- (a) We have at least one propositional symbol, and operators  $\wedge$  and  $\neg$ . Show how operators `False`, `True`,  $\vee$ ,  $\rightarrow$ , and  $\leftrightarrow$  can be created using them.
- (b) Sometimes – with propositional symbols – just one operator suffices to create the other operators. Let us denote by  $\nabla$  the operator that is defined as:

$\nabla$	False	True
False	True	False
True	False	False

Show how by using  $\nabla$  the common operators `False`, `True`,  $\wedge$ ,  $\neg$ ,  $\vee$ ,  $\rightarrow$ , and  $\leftrightarrow$  can be created.

(Hint: think first how  $\nabla$  is created using the common operators and first create  $\neg$ .)

6. Let us denote by  $\triangle$  the operator that is defined as:

$\triangle$	False	True
False	True	True
True	True	False

How can you build  $\nabla$  using  $\triangle$ ? And how  $\triangle$  using  $\nabla$ ?

**Some notifications considering this course:**

- Use of truth tables is usually not strictly denied (unless when it is explicitly denied), but since in practical calculations they are quite useless or at least inefficient, it is strongly recommended to avoid them. (Of course you can use them for verifying your answers.)
- For truth values we use symbols *False* and *True*, which can be abbreviated as *F* and *T*, correspondingly. It is not allowed to use values 0 and 1 to express logical values since (1) even they are used in digital techniques, they denote *bits* not *logical values*, and (2) we do have need to use both integers and logical expressions in same formulas and therefore it is possible that values *False* and 0, and/or *True* and 1 exist in same formulas.
- Also, remember to write a sufficient amount of intermediate phases so that your solution is easy to follow.

**Exercise group 2**

7. Reduce the followings. Do not use truth tables.

- (a)  $P \rightarrow P \rightarrow P \rightarrow P$
- (b)  $P \rightarrow (P \rightarrow P) \rightarrow P$
- (c)  $P \rightarrow ((P \rightarrow P) \rightarrow P)$
- (d)  $P \rightarrow (P \rightarrow (P \rightarrow P))$
- (e)  $(P \rightarrow P) \rightarrow (P \rightarrow P)$

8. Do the following claims hold or not? Reason your answers. Do not use truth tables.

- (a)  $P \rightarrow (P \wedge Q) \Rightarrow P \rightarrow (P \vee Q)$ .
- (b)  $P \rightarrow (Q \wedge R) \Rightarrow P \rightarrow (Q \vee R)$ .
- (c)  $(P \wedge Q) \rightarrow R \Rightarrow (P \vee Q) \rightarrow R$ .

Prove the following absorption laws. That is, without using themselves (and neither their correspondings for two propositional symbols). Do not use truth tables.

- (d)  $P \wedge (P \vee Q \vee R) \Leftrightarrow P$ .
- (e)  $P \vee (P \wedge Q \wedge R) \Leftrightarrow P$ .

9. Prove the followings. Do not use truth tables.

- (a) Constructive Dilemma  $(P \rightarrow Q) \wedge (R \rightarrow S) \wedge (P \vee R) \Rightarrow Q \vee S$ .
- (b) Destructive Dilemma  $(P \rightarrow Q) \wedge (R \rightarrow S) \wedge (\neg Q \vee \neg S) \Rightarrow \neg P \vee \neg R$ .
- (c) Bidirectional Dilemma  $(P \rightarrow Q) \wedge (R \rightarrow S) \wedge (P \vee \neg S) \Rightarrow Q \vee \neg R$ .

**Introduction to problems 10–12:** In a country there are two kind of people, *citizens* who always speak truth and *politicians* who always lie. We assume that everyone in that country is either citizen or politician, unless something else is told in the problem. Solve the following problems by (1) formalising the problem, (2) solving it using means of propositional logic, and (3) interpret your solution.

An example problem and its solution:

The problem: someone asks from a person A (who is either citizen or politician): “Are you citizen?”. Where he answers: “If I am citizen, I’ll eat my hat”. Prove that A must eat his hat.

Solution: we introduce the following abbreviations:

- $X$  X is citizen (where X is A, B, C, ...)
- $L$  The truth value of L, the sentence spoken by X.
- $H$  A eats his hat.

Now, if X is citizen then the sentence spoken by him, L, is true, in other words,  $X \rightarrow L$ . Correspondingly, if X is politician, it holds that  $\neg X$ , and we know that L is not true, in other words,  $\neg X \rightarrow \neg L$ . These are also the only alternatives. In other words, the premisses of the problem are  $X \rightarrow L$  and  $\neg X \rightarrow \neg L$  which can be written in a shorter form  $X \leftrightarrow L$ . This means that X is citizen if and only if the sentence spoken by him is true.

Person A’s answer or sentence can be written in form  $A \rightarrow H$ , meaning if I am citizen (A), then I shall eat my hat (H). The problem can now be presented in a form where we start from the premiss  $A \leftrightarrow (A \rightarrow H)$  and we must derive H using means of logic.

$$\begin{array}{ll}
 A \leftrightarrow (A \rightarrow H) & \Leftrightarrow \text{(remove implication)} \\
 A \leftrightarrow (\neg A \vee H) & \Leftrightarrow \text{(remove equivalence)} \\
 A \wedge (\neg A \vee H) \vee \neg A \wedge \neg(\neg A \vee H) & \Leftrightarrow \text{(distributive law and DeMorgan)} \\
 A \wedge \neg A \vee A \wedge H \vee \neg A \wedge \neg\neg A \wedge \neg H & \Leftrightarrow (2 \times \text{miscellaneous law}) \\
 \text{False} \vee A \wedge H \vee \text{False} \wedge \neg H & \Leftrightarrow \text{(commutativity) and “}\varphi \wedge \text{F} \Leftrightarrow \text{F”)} \\
 \text{False} \vee A \wedge H \vee \text{False} & \Leftrightarrow \text{(commutativity and } 2 \times \text{“}\varphi \vee \text{F} \Leftrightarrow \varphi\text{”)} \\
 A \wedge H & \Rightarrow \text{(miscellaneous law)} \\
 H &
 \end{array}$$

Especially elimination of operator  $\leftrightarrow$  can easily lead to long expressions. Often a problem can be solved easier by eliminating some variable, that is, by setting its value to True and False, in turn. The previous problem can be solved by elimination e.g. in the following way:

$$\text{If } A \Leftrightarrow \text{True then } A \leftrightarrow (A \rightarrow H) \Leftrightarrow \text{True} \leftrightarrow (\text{True} \rightarrow H) \Leftrightarrow \text{True} \leftrightarrow H \Leftrightarrow H.$$

$$\text{But, if } A \Leftrightarrow \text{False then } A \leftrightarrow (A \rightarrow H) \Leftrightarrow \text{False} \leftrightarrow (\text{False} \rightarrow H) \Leftrightarrow \text{False} \leftrightarrow \text{True} \Leftrightarrow \text{False.}$$

By setting True to A the value of H became True, but setting False to A causes a contradiction. Therefore the only possible solution is that both A and H are true. That implies H, in other words, “A eats his hat” is true.

10. We are now in the country of citizens and politicians. Formalize the following problems and solve them using means of propositional logic.
- I met two people, A and B, of that country. B said that if A is politician then also he is. What type A and B are?
  - X and Y are prosecuted for involvement of a robbery. A and B saw the case. A says that if X is guilty then also Y is. B claims that **either** X is not guilty **or** Y is guilty (notice the exclusive or!). Are A and B same type? (Are they both either citizens or politicians.)
  - Once again, there had been a robbery. A, B, and C were brought to police station for interrogation. The police found out that
    - no one else than A, B, or C has been involved in the case,
    - A will never do a heist alone, and
    - that C is not guilty
 Is B guilty or not guilty?

11. We move to the next country where half of its people are citizens and half are politicians. Also in this country the citizens tell always true and the politicians always lie. But in this country half of the people are crazy having delusions and therefore they believe that the true sentences are untrue and the false sentences are true. Instead of that half of the citizens are sane and they know which claims are true and which are untrue.

Therefore the people of this country are of four different types:

- sane citizens* who speak always true because they know what is true and what is not,
- crazy citizens* who always lie but because they believe that lies are true,
- sane politicians* who always lie even they know what is true and what is untrue, and
- crazy politicians* who speak always true but because they believe that they are lying.

Let us investigate a person X of that other country.

- Let us denote that *he (=X) is sane* by  $S(X)$ , and *he (=X) is citizen* by  $C(X)$  (if we have only one person we can use only symbols  $S$  and  $C$ ). Formalize the four types of people of that country.
  - Let us denote (*the truth value of*) *the sentence that X says* by  $L(X)$ . Formalize the problem: “Person X says sentence L. What type he is?” (C.f the introduction for citizen/politician-problems.)
  - Formalize the following problem and solve it using means of propositional logic: I once met a guy from that country and he said that he is sane or citizen. Which type he is? What about if he had said that he is **either** sane **or** citizen?
12. We are now in the other country. Formalize the following problems and solve them using means of propositional logic.
- A native claimed that he is not sane citizen. What type he is?
  - Another native told me that he is crazy citizen. What type he is?

## Exercise group 3

### Repetition of predicate logic

13. Let  $A[1 \dots n]$  ja  $B[1 \dots m]$  be arrays. Write state predicates that define – with suitable parameters – the follows. Try to make and use suitable auxiliary notions.
- Both arrays  $A$  and  $B$  have equally many elements and the sums of the elements of their corresponding indexes perform a palindrom (= a sequence that is same if it is read in either directions).
  - The smallest element of  $A$  is also the smallest element of  $B$ .
  - Array  $A$  has indexes  $k$  and  $l$  so that all the elements before  $k$  are smaller than the element in position  $k$ , and correspondingly all the elements after  $l$  are greater than the element in position  $l$ .
14. Write state predicates that define – with suitable parameters – the follows. Try to make and use suitable auxiliary notions.
- Set  $C$  has no smallest nor greatest element.
  - The part  $A[k \dots l]$  is the longest part of array  $A[1 \dots n]$  that is in descending order.
  - The elements of  $A$  form an *arithmetic sequence of second kind*, that is, the sequence that is formed by its differencies of consecutive elements is an arithmetic sequence (of first kind). E.g. sequence 4,7,11,16 is arithmetic sequence of second kind since the differences  $7 - 4 = 3$ ,  $11 - 7 = 4$ ,  $16 - 11 = 5$  form the arithmetic sequence 3,4,5.
15. The non-empty part  $A[i \dots j]$  is *smooth* if and only if its consecutive elements differ from each other by at most one (1). The length of the smooth part  $A[i \dots j]$  is  $j - i + 1$ . Define the following notions that speak about array  $A[1 \dots n]$ . Use – whenever possible – notions you have already defined. Remember that every predicate must be *well-defined*, that is, there is no division by zero, indexing past the range of an array, etc. For predicates, give also a short name and parameter list.
- Part  $A[i \dots j]$  is smooth.
  - $p$  is the length of the longest smooth part.
  - The smooth part  $A[i \dots j]$  is left-maximal, that is, its extension to the left is not anymore smooth.
  - Specification for a program that calculates the length of the longest smooth part.

## Exercise group 4

16. The task is to show that the following program assigns to  $z$  the greater value of variables  $x$  and  $y$ .

```

1      if  $x > y$  then  $z := x$ 
2      else  $z := y$ 
3      endif

```

- (a) Write a state predicate that says the followings:
- $z$  is either of the initial values of  $x$  and  $y$
  - $z$  is at least the initial value of  $x$
  - $z$  is at least the initial value of  $y$
- (b) Execute the program symbolically
- (c) Prove that the predicate you wrote in the end of the item (a) holds in the end of the program. (Warning! Remember that “ $\Rightarrow$ ” does not have the congruence property.)
17. Let array  $A[1 \dots n]$  be given. Write specifications for the following programs.
- (a) The program reverses the ordering of elements of  $A$ .
- (b) The program counts how many different elements is in array  $A$ .
- (c) The program reports is the average of all elements of  $A$  same as the average of its first and last element.
18. Let array  $A[1 \dots n]$  be given. Write specifications for the following programs.
- (a) The program adds the sum of all elements of  $A$  to its every element.
- (b) The program finds the greatest value that exists exactly two times in the array, otherwise it reports that there is no such element (how).
- (c) The program finds two different elements whose sum is  $x$  or reports that that there are no such elements (how).

## Weakest preconditions

19. Calculate and reduce the following weakest preconditions.

- (a)  $wp(x := 0, x = y)$
- (b)  $wp(x := 3 \cdot x, x \neq 0)$
- (c)  $wp(x := x - y, x > 0)$
- (d)  $wp(x := y - x, \frac{1}{x} > 0)$
- (e)  $wp(x := x + a; n := n + 1, x = n \cdot a)$
- (f)  $wp(x := x \cdot a; n := n + 1, x = a^n \wedge n > 0)$  (remember:  $0^0 = 1$ ).  
Does your solution work when  $a = 0$ ?

## Exercise group 5

20. Calculate the following weakest preconditions.

- (a)  $wp(\mathbf{if } i > j \mathbf{ then } j := j + 1 \mathbf{ else } i := i + 1 \mathbf{ endif}, i \geq j)$   
 (b)  $wp(\mathbf{if } A[i] < x \mathbf{ then } i := i + 1 \mathbf{ endif},$   
 $\forall j; 1 \leq j < i : A[j] < x) \quad (A[1 \dots n] \text{ is defined})$

21. Specify the weakest preconditions for the following programs.

- (a)  $y := x;$   
 $x := y$   
 $\langle P(x, y) \rangle$   
 (b)  $i := i + 1;$   
 $factorial := factorial \cdot i$   
 $\langle factorial = n! \rangle$   
 (c)  $factorial := factorial \cdot (i + 1);$   
 $i := i + 1;$   
 $\langle factorial = n! \rangle$   
 (d)  $x := 2 \cdot x + 1;$   
 $y := y - 1;$   
 $\langle y = 3 \cdot x \rangle$

22. Let  $Q \Leftrightarrow x \cdot b^i = a^n$ . Calculate

- (a)  $wp(b := b \cdot b, Q)$ .  
 (b)  $wp(i := i \text{ div } 2; b := b \cdot b, Q)$   
 (c)  $wp(S, Q)$ , when  $S$  is

$\mathbf{if } i \text{ mod } 2 = 1 \mathbf{ then } x := b \cdot x \mathbf{ endif}$   
 $i := i \text{ div } 2; b := b \cdot b .$

23. Let  $A$  be defined as array  $A[1 \dots n]$ . Prove the correctness of the following program, or, if that is not possible, analyse why your proof-trial fails and derive a counterexample.

$P \Leftrightarrow \forall j; 1 \leq j < i : A[j] \leq max \wedge$   
 $\exists j; 1 \leq j < i : A[j] = max .$

$\langle P \wedge 1 \leq i \leq n \rangle$   
 $\mathbf{if } A[i] > max \mathbf{ then } max := A[i]$   
 $\mathbf{endif};$   
 $i := i + 1$   
 $\langle P \rangle$

## Exercise group 6

24. Calculate and reduce the weakest preconditions assuming that  $A$  is  $A[1 \dots n]$ .
- (a)  $wp( A[i] := 5, \exists j ; i \leq j \leq n : A[i] \leq A[j] )$
  - (b)  $wp( A[i] := 5, \exists j ; i \leq j \leq n : A[i] < A[j] )$
  - (c)  $wp( A[i] := 5, \forall j ; i \leq j \leq n : A[i] \leq A[j] )$
25. Calculate and reduce the weakest preconditions assuming that  $A$  is  $A[1 \dots n]$ .
- (a)  $wp( A[i] := 5, \forall j ; 1 \leq j \leq n : A[j] = A_0[j] )$
  - (b)  $wp( A[i] := 5, A[A[i]] = 0 )$
  - (c)  $wp( A[i] := i, A[A[i]] = i )$
26. Calculate and reduce the weakest preconditions assuming that  $A$  is  $A[1 \dots n]$ .
- (a)  $wp( A[i] := j, A[A[j]] = A[i] )$
  - (b)  $wp( A[i] := A[i - 1] + A[i], A[i] = \sum_{j=1}^{i-1} A[j] )$
  - (c)  $wp( A[i] := A[i - 1] + A[i], A[i] = \sum_{j=1}^i A_0[j] )$
  - (d)  $wp( A[A[i]] := i, A[i] = 0 )$
  - (e)  $wp( A[A[i]] := i, \forall i ; 1 \leq i \leq n : A[i] = i )$
  - (f)  $wp( A[A[i]] := i, A[i] = 1 )$
  - (g)  $wp( A[A[i]] := j, A[j] = j )$
27. Let  $A$  be  $A[1 \dots n]$ . Calculate and reduce the weakest precondition for the following program
- ```

if  $A[i] > A[j]$  then
   $A[i] := A[i] - A[j]$ 
else
   $A[j] := A[j] - A[i]$ 
endif
 $\langle A[i] < A[j] \rangle$ 

```

## Exercise group 7

28. Let  $A$  be  $A[i \dots n]$ . Calculate the weakest precondition for that the following program swaps the values of  $A[i]$  and  $A[j]$ .

$$t := A[i]; A[i] := A[j]; A[j] := t$$

29. What is

- (a)  $wp(\mathbf{while\ True\ do\ skip\ endwhile}, Q)$ ?
- (b)  $wp(\mathbf{while\ False\ do\ skip\ endwhile}, Q)$ ?
- (c)  $wp(\mathbf{while\ } B \mathbf{\ do\ skip\ endwhile}, Q)$ ?

30. The following program is given

```

if    $x \leq y \wedge x \leq z \rightarrow t := x$ 
 $\square$    $y \leq x \wedge y \leq z \rightarrow t := y$ 
 $\square$    $z \leq x \wedge z \leq y \rightarrow t := z$ 
fi

```

- (a) What is does?
  - (b) Prove that it does it.
  - (c) the program is nondeterministic. How?
31. Prove that the two formulas given for  $wp(\mathbf{if}, Q)$  are equivalent, in other words, that

$$\begin{aligned} \uparrow B \wedge (\neg B \vee wp(S_1, Q)) \wedge (B \vee wp(S_2, Q)) &\Leftrightarrow \\ \uparrow B \wedge (B \wedge wp(S_1, Q) \vee \neg B \wedge wp(S_2, Q)) & \end{aligned}$$

32.  $A[1 \dots n]$  is an array. The following program with its specification is given. Prove that it is correct or show why it is incorrect. If it is incorrect then fix it and prove that the fixed program is correct.

```

 $\langle n \geq 0 \rangle$ 
 $k := 0; s := 0;$ 
while  $k \neq n$  do
    $s := s + A[k]; k := k + 1$ 
endwhile
 $\langle s = \sum_{i=1}^n A[i] \rangle$ 

```

## Exercise group 8

33. Prove that connecting statements consequitively is associative, that is, the meaning of the program

$$S1; S2; S3$$

is same as the meaning of programs  $S1; (S2; S3)$  and  $(S1; S2); S3$

## Proving correctness of a program

34. Prove that the following program is partial correct and that it terminates. The most important part of the invariant is  $x \cdot b^i = a^n$ .

```

{ n ≥ 0 }
i := n; b := a; x := 1
while i > 0 do
  if i mod 2 = 1 then x := b · x endif
  i := i div 2; b := b · b
endwhile
{ x = an }

```

35. Let  $A[1 \dots n]$  be an array, and

$$\text{wrong-order}(i, j) : \Leftrightarrow i < j \wedge A[i] > A[j], \text{ when } 1 \leq i \leq n \text{ and } 1 \leq j \leq n.$$

- (a) Prove that  $|W|$  is a bound function for the following program, if

$$W = \{ (i, j) \in \{1, \dots, n\}^2 \mid \text{wrong-order}(i, j) \}.$$

```

while  $\exists i, j; 1 \leq i \leq n \wedge 1 \leq j \leq n : \text{wrong-order}(i, j)$ 
do
  choose  $i$  and  $j$  so that  $\text{wrong-order}(i, j)$ 
  swap(  $A[i], A[j]$  )
endwhile

```

- (b) How the result of item (a) can be used for proving correctness of various sorting algorithms?
36. Using the properties of invariant and bound function for ordinary loop statement (presented in lectures) as a model, list the properties that must be required for invariant and bound function for nondeterministic loop statement.
37. Formulate the rules for proving the correctness of an ordinary **for**-loop:

$$\text{for } i := \text{low} \text{ to } \text{high} \text{ do } S \text{ endfor}$$

Notice that *low* and *high* may be expressions. How do you take that in to account?

## Exercise group 9

### Proving correctness of a program

38. The form “ $B_1$  **andthen**  $B_2$ ” causes that first  $B_1$  is evaluated and if it is True then  $B_2$  is evaluated. Correspondingly “ $B_1$  **orelse**  $B_2$ ” causes that first  $B_1$  is evaluated and if it is False then  $B_2$  is evaluated. Rewrite the following statements using ordinary **if-then-[else-]endif**-statements, that is, without **andthen** and/or **orelse** forms, so that the meaning of the original statements remains, and construct the *wp*-semantics for them.
- if**  $B_1$  **andthen**  $B_2$  **then**  $S$  **endif**
  - if**  $B_1$  **orelse**  $B_2$  **then**  $S$  **endif**
  - if**  $B_1$  **andthen**  $B_2$  **then**  $S_1$  **else**  $S_2$  **endif**
  - if**  $B_1$  **orelse**  $B_2$  **then**  $S_1$  **else**  $S_2$  **endif**
39. Rewrite the following statements using ordinary statements, e.g. assignment, **if-then-[else-]endif**, and **while-do-endwhile**, but no **andthen**, **orelse**, **elseif**, **for-to/downto-do-endfor**, **repeat-until** and/or **loop-loopend**.
- DO**  $i = low, high$   $S$  **END DO**, where  $S$  is executed at least once (slightly simplified from the ancient form of FORTRAN DO-loop)
  - repeat**  $S$  **until**  $B$
  - loop**  $S_1$ ; **if**  $B$  **then break endif**;  $S_2$  **endloop**
  - loop**  $S_1$ ; **if**  $B_1$  **then break endif**;  $S_2$ ; **if**  $B_2$  **then break endif**;  $S_3$  **endloop**
40. The forms “ $B_1$  **andthen**  $B_2$ ” and “ $B_1$  **orelse**  $B_2$ ” are as in problem 38. Rewrite the following statements using ordinary statements, e.g. assignment, **if-then-[else-]endif**, and **while-do-endwhile**, but no **andthen**, **orelse**, **elseif**, **for-to/downto-do-endfor**, **repeat-until** and/or **loop-loopend**.
- while**  $B_1$  **andthen**  $B_2$  **do**  $S$  **endwhile**
  - while**  $B_1$  **orelse**  $B_2$  **do**  $S$  **endwhile**
  - repeat**  $S$  **until**  $B_1$  **andthen**  $B_2$
  - repeat**  $S$  **until**  $B_1$  **orelse**  $B_2$

41. Let us denote the number of how many times  $x$  exists in array  $A[1 \dots n]$  by symbol  $\#(A, x)$ .

- (a) The element  $x$  is the *majority element* of  $A$ , if more than half of the elements of  $A$  have the value  $x$ . Write a predicate  $maj(x, A[1 \dots n])$  that says that either  $x$  is the majority element of  $A$  or  $A$  has no majority element.
- (b) Prove the total correctness of the following program. Hint: if  $e$  is the majority element of the whole  $A$  then in the beginning of every cycle either  $x = e \wedge k \geq 2 \cdot \#(A[1 \dots i - 1], e) - i + 1$ , or  $x \neq e \wedge -k \geq 2 \cdot \#(A[1 \dots i - 1], e) - i + 1$ .

```

⟨  $n \geq 0$  ⟩
 $k := 0$ ;
for  $i := 1$  to  $n$  do
  if  $k = 0$  then  $x := A[i]$ ;  $k := 1$ 
  elsif  $A[i] = x$  then  $k := k + 1$ 
  else  $k := k - 1$ 
  endif
endfor
⟨  $maj(x, A)$  ⟩

```

2. hint: (1) It easy to see that always  $k \geq 0$ . (2) Divide the cases into two parts, there either exists a majority element in  $A$  or not.

- (c) Design a program that produces the majority element of  $A$  or the answer “ $A$  has no majority element” by two browsings of  $A$ , and prove its correctness.

42. Prove that the 2. **for**-loop of COUNTING SORT satisfies its specification.

43. Find a suitable invariant and prove the 3. **for**-loop of COUNTING SORT satisfies its specification.

44. Why  $0, \dots, n$  suffices for the of array  $C$  in COUNTING SORT. Use in your reasoning formulas that are proved in the proof of COUNTING SORT.

## Exercise group 10

45. the proof of Fermat-program we actually proved that some program  $S$  satisfies the following specification

$$\langle x \geq 1 \wedge y \geq 1 \wedge z \geq 1 \wedge n \geq 3 \rangle$$

$$S$$

$$\langle x = y = z = 1 \wedge n = 3 \rangle$$

if and only if  $\forall x, y, z, n \in \mathbb{Z}^+ : n \geq 3 \rightarrow x^n + y^n \neq z^n$ .

Write  $S$ .

Notice that for  $S$  it does not suffice e.g.

$$x := 1; y := 1; z := 1; n := 3$$

Since it would satisfy the specification even there would exist  $x, y, z$  ja  $n \in \mathbb{Z}^+$  so that  $n \geq 3$  and  $x^n + y^n = z^n$ .

46. A program  $S$  as *inverted* is the program  $\neg S$  (sometime noted also  $S^{-1}$ ) for which  $\langle Q \rangle \neg S \langle P \rangle$  if and only if  $\langle P \rangle S \langle Q \rangle$ . Find rules for inverting different types of statements and sequences of statements. *Hint*: if necessary, use nondeterministic statements.

## Proving correctness as a means of review

47. What fails (if somethin fails) in the proof of binary search when the program is modifies as in the followings? If the proof breaks but the program not then fix the proof.

```

    < True >
1  a := 1; y := n + 1;
2  while a < y do
3      v := (a + y) div 2;
4      if A[v] < key then a := v + 1
5      else y := v
6      endif
7  endwhile
    < (a = n + 1 ∨ A[a] ≥ key) ∧ (a = 1 ∨ A[a - 1] < key) >

```

- (a) Line 1 is modified into form  $a := 1; y := n$ ;
- (b) Line 4 is modified into form **if**  $A[v] \leq key$  **then**  $a := v + 1$
- (c) Line 4 is modified into form **if**  $A[v] < key$  **then**  $a := v$
- (d) Line 5 is modified into form **elsif**  $A[v] > key$  **then**  $y := v - 1$  **else**  $a := v; y := v$

48. Some “hero” could not understand that his Basic-program, like the following, was incorrect. The program should find the maximum among some numbers. What error? Calculate the weakest precondition for that his program works correct, when the input is  $n(> 0), a_1, a_2, \dots, a_n$ .

```
10  read n
20  for i = 1 to n
30  read x
40  if x <= max then goto 60
50  max = x
60  next i
70  print max
```

## Exercise group 11

49. For fixing the error found in the lectures the text element recognition program was modified as the following. But still it is not yet correct. Why not?

```

k := 1; (* k = cursor *)
while k ≤ linelength do
  if line[k] = ' ' then k := k + 1
  else
    match := false; i := 0;
    while i < nofelements ∧ ¬match do
      i := i + 1;
      if k + element[i].length - 1 ≤ linelength then
        match := true;
        for j := 1 to element[i].length do
          match := match ∧ element[i].c[j] = line[k + j - 1]
        endfor
      endif
    endwhile
    if match then
      write(i); k := k + element[i].length
    else
      write('error in location ', k);
    endif
  endif
endwhile

```

50. Design a program that computes in linear time  $n^4$ , where  $n$  is a natural number so that the program does not use multiplication nor raising to power operations. In other words, **in the same way as was done for the cube of a natural number in the lecture material.**

51. In exercise problem 15 we defined that the non-empty part  $A[i \dots j]$  is *smooth* if and only if its consecutive elements differ from each other by at most one (1). The length of the smooth part  $A[i \dots j]$  is  $j - i + 1$ .

The following program computes the length of the longest smooth part of  $A$ .

```

a := 1; p := 1;
for i := 1 to n - 1 do
  if | A[i] - A[i + 1] | ≤ 1 then
    if i - a + 2 > p then
      p = p + 1
    endif
  else
    a := i + 1
  endif
endfor

```

- Write predicates  $S$ , which tells that part  $A[i \dots j]$  is smooth,  $P$ , which tells that  $p$  is the length of the longest smooth part, and  $L$ , which tells that a smooth part  $A[i \dots j]$  is *left-maximal*, i.e. its extension to the left would not be smooth. (They are already written in exercise problem 15 but you may re-write them in a form that suits better for your solution.)
- Write pre- and postconditions for the program.
- Write loop invariant for the for-loop.
- Prove the total correctness of the program.

## Exercise group 12

52. arrays  $A[1 \dots n]$  and  $B[1 \dots m]$  are sorted in ascending order. The next program **Merge** merges them and produces array  $C[1 \dots n + m]$ , which is also in ascending order.

```

i := 1; j := 1; k := 1;
while i ≤ n and j ≤ m do
  if A[i] < B[j] then
    C[k] := A[i]; i := i + 1
  else
    C[k] := B[j]; j := j + 1
  endif;
  k := k + 1
endwhile
while i ≤ n do
  C[k] := A[i]; i := i + 1; k := k + 1;
endwhile;
while j ≤ m do
  C[k] := B[j]; j := j + 1; k := k + 1;
endwhile;

```

- (a) Define first auxiliary predicates *ordered*, which tells is the array given as a parameter in order, and *composed*, which has three arrays as parameters and tells does the third array contain exactly the elements of the two first arrays.
- (b) Write pre- and postconditions for the program.
- (c) Define loop invariants and bound functions for the **while**-statements
- (d) Prove the total correctness of the program
53. Is the following specification correct?

```

⟨ n ≥ 0 ⟩
i := 0; r := 0; s := 0; t := 4;
while i < n do
  i := i + 1; r := r + s; s := s + t; t := t + 6
endwhile
⟨ r = n3 - n2 ⟩

```

**Designing the program and its proof together**

54. (a) We forget to prove that the following holds in the beginning of the 1. **for**-loop of the inner sequence program

$$\forall h ; 0 \leq h \leq k : 0 \leq Last[h] < i$$

Do it now.

- (b) Does the following always hold in the beginning of the 1. **for**-loop

$$\forall j ; 1 \leq j < i : 0 \leq Prev[j] < i$$

55. (a) Can the initialisation  $Last[0] := 0$  left out without doing any other modifications to the program? Does the situation change if we assume that the language includes execution time checkings that ensure that all values to be assigned are of range  $0, \dots, n + 1$ ? (E.g. You can have these kind of checkings for a Pascal-program.)
- (b) After what kind of modifications the element  $Last[0]$  can totally be left out?

## Exercise Group 13

56. Prove that the following program finds an element in two-dimensional array  $A[1 \dots m, 1 \dots n]$ .

```

i := 1; j := 1;
do i ≤ m ∧ j ≤ n andthen A[i, j] ≠ key → j := j + 1
[] i ≤ m ∧ j > n → i := i + 1; j := 1
od

```

57. Arrays  $A[1 \dots n_A]$ ,  $B[1 \dots n_B]$ , and  $C[1 \dots n_C]$  are in ascending order, and there is at least one element that exists in all three. Design a program and its proof that finds the smallest element that exists in all three arrays.

## Proving Algorithms

58. (a) Where the algorithm that goes through nodes used the knowledge that
- i.  $|V| < \infty$
  - ii.  $\forall v \in V : |\text{neighbours}(v)| < \infty$
- (b) Would the algorithm that goes through nodes break, if line 10 is moved between lines 3 and 4? Do the reasoning for both the abstract and the concrete code. Which one was easier?