Technische Universität München
Fakultät für Informatik
Dr. Peter Lammich
Simon Wimmer

WS 2018/19
11.12.2018

Semantics of Programming Languages

Exercise Sheet 9

Exercise 9.1 DFS Search

In this exercise we will design and verify a simple DFS search through a graph. Given some start node s, our goal will be to check whether we can reach a given target x using the edges in the graph. For simplicity, nodes in our graph can be any integer numbers, and the edges of the graph will be represented by a single array a, specifying the *successors* of a node as successive integers in the array. More precisely, we define the successors of node i in a as:

definition *succs* **where**
  "succs a i ≡ a ‘ {i+1..<a i}" **for** *a* :: *"int ⇒ int"*

Now the edge set (as a set of pairs) of the graph can simply be defined as:

definition *Edges* **where**
  "Edges a ≡ {(i, j). j ∈ succs a i}"

The set of states that are reachable from a given state s is $(Edges\ a)^*$ ‘‘ $\{s\}$. (Click on the individual parts to understand the definition). Now try to program a DFS search that decides $x \in (Edges\ a)^*$ ‘‘ $\{s\}$. Your program should use one array as a stack to keep track of the nodes it still needs to visit and and another array to represent the set of nodes that the search has already explored. For now we do not concern ourselves with termination, so you can use the keyword (*partial*) for your definition. Use the following outline for your code:

```
b = 0;
i = 1; — i will point to the next free space in the stack (i.e. it is the size of the stack)
stack[0] = s; — Initially, we put s on the stack.
while ... {
  i = i - 1;
   next = stack[i]; — Take the top most element from the stack.
— If it is the target, we are done. Set b = 1.
— Else, mark next as visited, and put its successors on the stack if they have not been
visited yet.
}
```

Prove that your program fulfills the following specification:

program_spec (*partial*) *dfs*
  assumes "$0 \le x \wedge 0 \le s$"
  ensures "$b = 1 \longrightarrow x \in (Edges\ a)^*$ `` $\{s\}$"

Hint: You will likely want to factor out an inner loop for finding the successors of a node as a subprogram.

Assuming that the input graph is finite, we can also prove that the algorithm terminates. We will thus use an *Isabelle context* to fix a certain finite graph and a start state:

context
  fixes *start* :: *int* **and** *edges*
  assumes *finite_graph*: "*finite* $((Edges\ edges)^*$ `` $\{start\})$"
begin

Prove the following specification for your program:

program_spec *dfs1*
  assumes "$0 \le x \wedge 0 \le s \wedge start = s \wedge edges = a \wedge visited$ ` $\{x.\ True\} = \{0\}$"
    ensures "$b = 1 \longrightarrow x \in (Edges\ a)^*$ `` $\{s\}$"

Homework 9.1 Redundant Assignments

Submission until Tuesday, December 18, 2018, 10:00am.

It is quite obvious that the sequence $x[] = a$; $a[] = x$ can be simplified to the statement $x[] = a$. Prove this equivalence:

theorem *array_assign*: "$(x[] ::= a;;\ a[] ::= x) \sim (x[] ::= a)$"

We can try to apply the same idea for single variables instead of arrays and conjecture that the sequence $x[i] := a[j]$; $a[j] := x[i]$ is equivalent to the statement $x[i] := a[j]$. However the following cannot be proved:

theorem
  "$(x[i] ::= Vidx\ a\ j;;\ a[j] ::= Vidx\ x\ i) \sim (x[i] ::= Vidx\ a\ j)$"

Explain the problem and find a weaker formulation of the theorem (i.e. add assumptions) that is valid. The theorem should still be as general as possible. Prove the theorem. Note that i and j can be *any* expressions of type *aexp*.

Homework 9.2 Array Partitioning

Submission until Tuesday, December 18, 2018, 10:00am.

Write down and verify a program that partitions an array around the value *0*. That is, your program should fulfill the following specification:

program_spec *partition*

assumes *"$l \le h$"*
ensures *"$mset\_ran\ a\ \{l_0..<i\} = filter\_mset\ (\lambda x.\ x \ge 0)\ (mset\_ran\ a_0\ \{l_0..<h_0\})$
$\wedge\ mset\_ran\ a\ \{i..<h_0\} = filter\_mset\ (\lambda x.\ x < 0)\ (mset\_ran\ a_0\ \{l_0..<h_0\})$"*

Hint: Your program can **strongly** resemble the program for filtering in an array that you have seen before.

Homework 9.3 String Search

Submission until Tuesday, December 18, 2018, 10:00am.

Note: This is a bonus exercise worth up to five additional bonus points. Partial credit can be given.

Write down and verify a program that checks whether an array a (with bounds l and h) contains the substring given by the array b (with bounds 0 and len), setting $match = 1$ if the substring is found. That is, your program should fulfill the following specification:

program_spec *substring_final*
assumes *"$l \le h \wedge 0 \le len$"*
ensures *"$match = 1 \longleftrightarrow (\exists\, as\ bs.\ lran\ a\ l_0\ h_0 = as\ @\ lran\ b\ 0\ len\ @\ bs)$"*
for *$l\ h\ len\ match\ a[]\ b[]$*

You should use modularization. Your string search doesn't need to be efficient. Thus, you can use an outer loop to scan through the possible starting indices for the substring in a, and an inner loop to match the contents of b against the contents of a, starting at a given position in a. To prove the final specification, the following theorems can be helpful (find them in the template):

lemma *lran_split*:
"$lran\ a\ l\ h = lran\ a\ l\ p\ @\ lran\ a\ p\ h$" **if** *"$l \le p$" "$p \le h$"*

lemma *lran_eq_append_iff*:
"$lran\ a\ l\ h = as\ @\ bs \longleftrightarrow (\exists\ i.\ l \le i \wedge i \le h \wedge as = lran\ a\ l\ i \wedge bs = lran\ a\ i\ h)$" **if** *"$l \le h$"*