UNIVERSITY BORDEAUX 1
MASTER INFORMATIC - SOFTWARE ENGINEERING

A PROJECT REPORT
ON

FORMAL DESIGN

BY

NGUYEN Quang Anh

2014-2015

ABSTRACT

Formal Methods in System Design allows the designing, implementing, and validating the correctness of the system. In this project, I implemented and proved the correctness of the machines that check a given array if it's sorted in ascending order, sorted (ascending or descending), check two given arrays if they are identicals, if one array included the other, and sort a given array in ascending order

The purpose of this paper, is to deliver the process of proving the proof obligations, those that was proved interactively, as well as the reasoning for the unproved ones.

1 INTRODUCTION

In the last lesson of Formal Design, we were given a project to finish. Our job is to design several machines that :

- check if a given array is sorted in ascending order
- check if a given array is sorted (ascending or descending order)
- check if two given arrays are identical
- given two arrays, check if one array included the other one
- given an array with pairwise distinct values, sort this array in ascending order
- given an array, sort this array in ascending order

I used a software, named **Rodin**, to do this project. Some theories will be proved in this paper, if cannot be proved in the program.

2 Check if array is sorted in ascending order

- Specification: TestAscendingMachine
- Implementation: TestAscendingMachineImplementation

The only PO that need to be proved interactively, is **liveness/WD** int **TestAscendingMachineImplementation**. I proved this using tthe tactic **Disjunction to implication**

3 Check if array is sorted

- Specification: TestSortedMachine
- Implementation: TestSortedMachineImplementation

The POs that need to be proved interactively is :

- inv9/WD
- liveness/WD
- liveness/THM
- INITIALISATION/inv7/INV
- IS_SORTED/grd1/GRD
- LOOP_EQUAL/inv9/INV

The only tatic that I used, is **Disjuction to implication**

4 Check if two given arrays are identical

- Specification: CompareArraysMachine
- Implementation: CompareArraysMachineImplementation

The POs that need to be proved interactively:

- In CompareArraysMachine
  - NOT_IDENTICAL/grd1/WD
- In CompareArraysMachineImplementation
  - liveness/WD
  - IS_IDENTICAL/grd1/GRD
  - NOT_IDENTICAL/grd1/WD

The only tactic used to solve, is **Disjuction to implication**

5 Check if the values of one array is included in another one

- Specification: ValueIncludedMachine
- Implementation:
  - ValueIncludedMachineImplementation
  - ValueIncludedMachineImplementation2

ValueIncludedMachineImplementation is a machine that check the values of the two arrays whether they satisfy :

$$\forall i \cdot i \in dom(array1) \Rightarrow (\exists j \cdot j \in dom(array2) \Rightarrow array1(i) = array2(j))$$

The complexity of this problem is:

$$O(n_1, n_2) = n_1.n_2 \quad n_1 = card(dom(array1)) \wedge n_2 = card(dom(array2))$$

ValueIncludedMachineImplementation2 is a special case of **ValueIncludedMachineImplementation**, where the two given arrays are sorted in ascending order.

The POs that need to be proved interactively are:

- liveness/WD (in both of the implementations)
- liveness/THM (in both of the implementations)

The tactic that was used to prove, is **Disjunction to implication**

6 Sort an array pairwised distinct values

- Specification: SortArrayDistinctValueMachine
- Implementation: SortArrayDistinctValueAscendingMachineImplementation

In this machine, the POs that need to be proved interactively is many, and the proving process is more complicated than before.

6.1 LOOP_SWAP/inv6/INV

I added some hypotheses:

- $a'(last\_index + 1) = a(last\_index + 1)$
- $last\_index + 1 > indice + 1$
- $i = indice \lor i = indice + 1 \lor (i \neq indice \land i \neq indice + 1)$

6.2 LOOP_SWAP/inv7/INV

Hypothese added:
$i = indice \lor i = indice + 1 \lor i < indice$
After that, I proved by case. First case, $i < indice$, add hypothese : $a'(i) = a(i)$.
For two last cases, $i = indice \lor i = indice + 1$, it was proved automatic.

6.3 LOOP_SWAP/inv10/INV

Demonstrate:

$$\forall i \cdot i \in last\_index + 1..size \land i + 1 \in last\_index + 1..size \Rightarrow a'(i) \leq a'(i + 1)$$

I applied this hypothese for i and $i + 1$

$$\forall i \cdot i \in 1..size \land \neg i = indice \land \neg i = indice + 1 \Rightarrow a'(i) = a(i)$$

After that I applied this hypothese, and it was done.

$$\forall i \cdot i \in last\_index + 1..size \land i + 1 \in last\_index + 1..size \Rightarrow a(i) \leq a(i + 1$$

6.4 LOOP_SWAP/inv12/INV

Demonstrate:

$$\forall i \cdot i \in 1..size \Rightarrow (\exists j \cdot j \in 1..size \Rightarrow array(i) = a'(j))$$

Applying case distinction :

$$i = indice \lor i = indice + 1 \lor (i \neq indice \land i \neq indice + 1)$$

After that, each case was proved automatically.

6.5 LOOP_SWAP/act4/FIS

Prove that :

$$\begin{aligned} &\exists a' \cdot a' \in 1..size \rightarrow array[1..size] \\ &\land (\forall i \cdot i \in 1..size \land i \neq indice \land i \neq indice + 1 \Rightarrow a'(i) = a(i)) \\ &\land a'(indice) = a(indice + 1) \land a'(indice + 1) = a(indice) \end{aligned}$$

Here I chose that $a' = a < +\{indice \mapsto a(indice + 1), indice + 1 \mapsto a(indice)\}$ (I tried to define the function overriding symbol like in Rodin with $< +$)

After that, the PO was proved automatically.

6.6 BUBBLE_SORT/inv10/INV

Demonstrate:

$$\forall i \cdot i \in last\_index..size \land i + 1 \in last\_index..size \Rightarrow a(i) \leq a(i + 1)$$

I applied case distinction tactic, with $i = last\_index \lor i > last\_index$, and the PO was proved automatically after that.

6.7 RETURN/grd2/GRD

Demonstrate:

$\forall i \cdot i \in dom(a) \wedge i+1 \in dom(a) \Rightarrow a(i) \leq a(i+1)$
with hypothese:
$(in\_loop = FALSE \wedge swap = FALSE \wedge indice = last\_index \wedge last\_index > 1)$
$\vee\, last\_index = 1$

Apply 'proof by case'-tatic in the hypothese.

6.7.1 First case:

We have:

$(in\_loop = FALSE \wedge swap = FALSE \wedge indice = last\_index \wedge last\_index > 1)$

Then do case distinction for i : $i < last\_index \vee i = last\_index \vee i > last\_index$

In case $i < last\_index$, apply hypothese :
$\forall i \cdot i \in 1..last\_index \wedge i+1 \in 1..last\_index \Rightarrow a(i) \leq a(i+1)$

In case $i = last\_index$, apply hypothese :
$last\_index < size \Rightarrow (\forall i \cdot i \in 1..last\_index \Rightarrow a(i) \leq a(last\_index + 1))$

In case $i > last\_index$, apply hypothese :
$\forall i \cdot i \in last\_index + 1..size \wedge i \in last\_index + 1..size \Rightarrow a(i) \leq a(i+1)$

6.7.2 Second case:

We have: $last\_index = 1$

Apply hypothese :
$\forall i \cdot i \in last\_index + 1..size \wedge i+1 \in last\_index + 1..size \Rightarrow a(i) \leq a(i+1)$

Then we do case distinction for i : $i = 1 \vee i > 1$

6.8 RETURN/grd3/GRD

Demonstrate :

$$\forall i \cdot i \in dom(array) \Rightarrow (\exists j \cdot j \in dom(a) \Rightarrow array(i) = a(j))$$

This condition is enough to ensure that no member in **array** is missing in **a**, because the number of occurence of each member is 1. In the next part, sorting a random array, we have to ensure that the number of occurrences of each member in **array** and **a** are the same

6.9 liveness/THM

Apply these hypothese (most of them are logic transformation) :

$(in\_loop = TRUE \wedge indice < last\_index \wedge a(indice) > a(indice + 1) \wedge last\_index > 1) \vee$
$(in\_loop = TRUE \wedge indice < last\_index \wedge a(indice) \leq a(indice + 1) \wedge last\_index > 1)$
$\Leftrightarrow in\_loop = TRUE \wedge indice < last\_index \wedge last\_index > 1$

$(in\_loop = TRUE \wedge indice < last\_index \wedge a(indice) > a(indice + 1) \wedge last\_index > 1) \vee$
$(in\_loop = TRUE \wedge indice < last\_index \wedge a(indice) \leq a(indice + 1) \wedge last\_index > 1) \vee$
$(in\_loop = TRUE \wedge indice = last\_index \wedge last\_index > 1)$
$\Leftrightarrow in\_loop = TRUE \wedge last\_index > 1 \wedge indice \leq last\_index$

$(in\_loop = TRUE \wedge indice = last\_index \wedge last\_index > 1) \vee$
$(in\_loop = FALSE \wedge swap = FALSE \wedge indice = last\_index \wedge last\_index > 1) \vee$
$(in\_loop = FALSE \wedge indice = last\_index \wedge swap = TRUE \wedge last\_index > 1)$
$\Leftrightarrow indice = last\_index \wedge last\_index > 1$

$(in\_loop = FALSE \wedge swap = FALSE \wedge indice = last\_index \wedge last\_index > 1) \vee$
$(in\_loop = TRUE \wedge indice < last\_index \wedge a(indice) > a(indice + 1) \wedge last\_index > 1) \vee$
$(in\_loop = TRUE \wedge indice < last\_index \wedge a(indice) \leq a(indice + 1) \wedge last\_index > 1) \vee$
$(in\_loop = TRUE \wedge indice = last\_index \wedge last\_index > 1) \vee$
$(in\_loop = FALSE \wedge swap = FALSE \wedge last\_index = size \wedge last\_index > 1) \vee$
$(in\_loop = FALSE \wedge indice = last\_index \wedge swap = TRUE \wedge last\_index > 1)$
$\Leftrightarrow last\_index > 1 \wedge$
$(indice = last\_index \vee in\_loop = TRUE \vee$
$(in\_loop = FALSE \wedge swap = FALSE \wedge last\_index = size))$

7 Sort a radom array

- Specification: SortArrayAscendingMachine
- Implementation: SortArrayAscendingMachineImplementation

We have 2 more condition to satisfy, since the array is totally random:

- $dom(new\_array) = dom(array) \wedge ran(new\_array) = ran(array)$
- $\forall y \cdot y \in ran(array) \Rightarrow card(\{u | u \in dom(array) \wedge u \mapsto y \in array\}) = card(\{x | x \in dom(new\_array) \wedge x \mapsto y \in new_a rray\})$