Curvature Estimation Algorithm's Implementation

NGUYEN Quang Anh, REBMANN Guillaume

April 29, 2015

Abstract

ABSTRACT

Formal Methods in System Design allows the designing, implementing, and validating the correctness of the system. In this project, I designed a sorted array, and the way to evaluate if it's sorted in ascending order.

1 INTRODUCTION

In the last lesson of Formal Design, we were given a project to finish individually. Our job is to design a sorted array in ascending order, and a method to confirm if one array is sorted.

I used a software, named **Rodin**, to do this project. There were three steps needed in this project

- Definition of an array
- Specification of a machine that will evaluate an array to see if it's sorted in ascending order
- Implementation of the machine

2 ARRAY'S CONTEXT

In this project, we defined an array as a morphism

$$f : X \to \mathbb{Z}, X = \{i | i \in \mathbb{N} \land 1 \leq i \leq n\} \quad \textit{(n is the size of our array)}$$

3 MACHINE'S SPECIFICATION

Our machine's job is to evaluate an array if it satisfies the condition, means that it is sorted in ascending order. So we have to give the definition of the condition first

If an array satisfies the condition, we must have :

$$\forall i, i \in 1..n-1 \implies array(i) \leq array(i+1) \quad \textit{(n is the size of array)}$$

Means that if an array doesn't satisfy the condition, we will have :

$$\exists i, i \in 1..n-1 \implies array(i) > array(i+1) \quad \textit{(n is the size of array)}$$

4 MACHINE'S IMPLEMENTATION

For examining an array, we need to run through all of the members of the array. So I introduced a variable *indice* which hold the current position of the last array's element that was visited.

$$indice \in 1..size$$

The machine will stop when it find an element that doesn't satisfy the condition in section 3. That means all the member before the *indice* must satisfy the condition.

$$\forall i \cdot i \in 1..size \land i < indice \implies array(i) \leq array(i+1)$$

At first *indice* start with value of 1. Then, as long as it satisfies the condition below, we increase *indice* by 1

$$indice < size \land array(indice) \leq array(indice+1)$$

Now, our array is sorted in ascending order if $indice \geq size$, and it isn't if $indice < size \land array(indice) > array(indice+1)$

To test that our machine will not stop half way, we need to justify that the expression below is true

$$(indice \geq size) \vee (indice < size \wedge array(indice) > array(indice + 1))$$

$$\vee(indice < size \wedge array(indice) \leq array(indice + 1))$$

To test that our machine will not fall into an infinite loop, we need to point out a value that would decrease each time we go through a loop, and is greater than zero. I choose this value : $size - indice$. This value is indeed decreased each time we go through the loop, as *indice* would increase and *size* is fixed, and because of the definition of *indice*, $size - indice \geq 0$

5 CONCLUSION

After finishing this project, I have obtained many knowledge, first of all, is the usage of *Rodin* and the way to define a system in a logical way. I have to think about the problems that I have never thought about before when programming. And most importantly, is that I could find out that all my knowledges about boolean algebra are still useful, and it could contribute to build a system in a clear and logical way.