UNIVERSITY ”ALEXANDRU-IOAN CUZA” FROM IAȘI

FACULTY OF COMPUTER SCIENCE

BACHELOR THESIS

Verification of Boyer-Moore algorithm in F*

proposed by

Daniel-Antoniu Dumitru

Session: july, 2024

Scientific coordinator

Associate Professor Ciobâcă Ștefan

UNIVERSITY ”ALEXANDRU-IOAN CUZA” FROM IAȘI

FACULTY OF COMPUTER SCIENCE

Verification of Boyer-Moore algorithm in F*

Daniel-Antoniu Dumitru

Session: july, 2024

Scientific coordinator

Associate Professor Ciobâcă Ștefan

Approved,

Coordinator of the license,

Associate Professor Ciobâcă Ștefan.

Date: .... Signature: ....

Declaration regarding the originality of the content of the bachelor thesis

The undersigned **Dumitru Daniel-Antoniu** living in **Romania, Neamț county, Roman city, Smirodava street, block 6, first floor, apartment 20**, born on **19 december 2001**, identified through CNP **complete**, graduate of Faculty of Computer Science, **Faculty of Computer Science computer science** specialization, 2024 promotion, declare on my own liability, knowing the consequences of forgery in statements in the meaning of the article 326 from New Penal Code and the dispositions of National Education Laws number 1/2011 article 143 paragraphs 4 and 5 regarding plagiarism, that my bachelor thesis with the title **Verification of Boyer-Moore algorithm in F***, elaborated under the guidance of Mr **Associate Professor Ciobâcă Ștefan**, which I am going to support in front of the commission, is original, belongs to me and I assume the content in it's entirety.

I also declare that I agree to have my bachelor thesis checked through any legal way for originality confirmation, including consenting to introduce it's content into a database for this purpose.

I am aware of the fact that it is prohibited the sale of scientific works in order to facilitate the falsification of the author of a bachelor, diploma or dissertation thesis quality by the buyer and in this regard, I declare on my own liability that this thesis was not copied, but represents the fruit of the research that I undertook.

Date: .... Signature: ....

Declaration of consent

By this I declare that I agree for my bachelor thesis with the title **Verification of Boyer-Moore algorithm in F***, the source code of the programs and the rest of the contents (graphics, multimedia, test data, etc.) that are of this thesis to be used within the Faculty of Computer Science.

I also agree for Faculty of Computer Science from University ”Alexandru-Ioan Cuza” from Iași to use, modify, reproduce and distribute in non-commercial purposes computer programs, executable format and source, made by me in this bachelor thesis.

Graduate **Daniel-Antoniu Dumitru**

Date: .... Signature: ....

Cuprins

Motivation

One of my favourite courses from the faculty was functional programming.

Introducere

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Nunc mattis enim ut tellus elementum sagittis vitae et. Placerat in egestas erat imperdiet sed euismod. Urna id volutpat lacus laoreet non curabitur gravida. Blandit turpis cursus in hac habitasse platea. Eget nunc lobortis mattis aliquam faucibus. Est pellentesque elit ullamcorper dignissim cras tincidunt lobortis feugiat. Viverra maecenas accumsan lacus vel facilisis volutpat est. Non odio euismod lacinia at quis risus sed vulputate odio. Consequat ac felis donec et odio pellentesque diam volutpat commodo. Etiam sit amet nisl purus in. Tortor condimentum lacinia quis vel eros donec. Phasellus egestas tellus rutrum tellus pellentesque eu tincidunt. Aliquam id diam maecenas ultricies mi eget mauris pharetra. Enim eu turpis egestas pretium.

Capitolul 1

Boyer-Moore algorithm

Boyer-Moore is one of the most efficient algorithms used in practice. It is used in text editors for functions like find and replace, but also in instruments like grep.[1] This algorithm is based on two heuristics: the bad character and the good suffix. Each of them can be used independently and helps in shifting the pattern such that we can find a match(if there exists one) in an efficient amount of time. One particularity is that the instructions are executed faster as the size of the input data increases.

1.1 The bad character heuristic

To use this heuristic, it is necessary to preprocess the pattern string. In order to do that, an array of the same length as the alphabet is created. Let's call this array bc. Each index from bc corresponds to the character from the alphabet stored in the same index (for example, the second index from bc corresponds to the second character from the alphabet). In each index is stored the last position of the character in pattern. To treat the case where a value from the alphabet is not in the pattern, all of the indices from bc will be initially set to -1. After that, we go through each index from the pattern, from the first one to the last one, and we store the position of the character in bc. If a value appears in more than one position , the last one will overwrite the other ones.

1.2 The good suffix heuristic

As in the bad character heuristic, the pattern is preprocessed and the values are stored in an array. Let's call this array gs.

Capitolul 2

F* language

2.1 Titlul secțiunii 1

Eros donec ac odio tempor. Facilisi morbi tempus iaculis urna id volutpat. Faucibus in ornare quam viverra orci sagittis eu. Amet tellus cras adipiscing enim eu turpis egestas. Integer feugiat scelerisque varius morbi. Platea dictumst vestibulum rhoncus est pellentesque elit ullamcorper dignissim. Bibendum arcu vitae elementum curabitur. Eu nisl nunc mi ipsum faucibus. Id aliquet lectus proin nibh nisl condimentum id venenatis a. Cras adipiscing enim eu turpis egestas pretium. Quisque non tellus orci ac auctor augue mauris augue. Malesuada pellentesque elit eget gravida cum. Ut lectus arcu bibendum at. Massa id neque aliquam vestibulum morbi blandit. Posuere ac ut consequat semper viverra nam. Viverra adipiscing at in tellus integer feugiat scelerisque varius morbi. Morbi enim nunc faucibus a pellentesque sit amet porttitor eget. Eu feugiat pretium nibh ipsum consequat nisl vel. Nisl purus in mollis nunc sed.

2.2 Titlul secțiunii 2

Elementum sagittis vitae et leo duis ut diam quam nulla. Purus sit amet volutpat consequat mauris nunc. Tincidunt augue interdum velit euismod in pellentesque massa. Nunc sed augue lacus viverra vitae congue. Porttitor leo a diam sollicitudin. Faucibus pulvinar elementum integer enim. Adipiscing bibendum est ultricies integer quis auctor elit. Blandit aliquam etiam erat velit scelerisque in. A iaculis at erat pellentesque adipiscing commodo elit at. Erat nam at lectus urna duis. Consequat ac felis donec et. Fermentum posuere urna nec tincidunt praesent semper feugiat nibh sed. Proin gravida hendrerit lectus a. Pretium viverra suspendisse potenti nullam ac tortor vitae purus. Arcu cursus euismod quis viverra nibh cras pulvinar mattis. Gravida arcu ac tortor dignissim convallis aenean. Quam nulla porttitor massa id neque aliquam vestibulum morbi. Sed viverra ipsum nunc aliquet. Quis enim lobortis scelerisque fermentum dui faucibus in.

Capitolul 3

Proof for Boyer-Moore algorithm

One of the problems

3.1 Defining the input data

A key element in Boyer-Moore algorithm is the alphabet array. All of the characters from text and pattern needs to be elements from this vector. In order to define it, I created a type in F* which contains the first 2 letters from the english alphabet.

```
type englishLetters = | A | B
```

Then I defined the alphabet array as a list which contains both of the englishLetters. I used a refinement type to specify that the list cannot be empty and all of the elements of type englishLetters are in the array.

```
val alphabet : (l:list englishLetters(forall (x:englishLetters).
mem x l = true) / l <> [])
let alphabet = [A;B]
```

The next two global variables are text and pattern. Like alphabet, each of these variables are lists of englishLetters. Text is a non-empty list and pattern is a list of length less or equal than the length of text. From the definition, the second list can be null.

```
val text : (l:list englishLettersl <> []) let text = [A;A;A;A;B;A;
val pattern : (l:list englishLetterslength l <= length text) let
pattern = [A;B;B]
```

Length and mem are functions defined in FStar.List.Tot.Base library. The first function (length l) returns the length of the list l and the second function (mem x l) verifies if x is in the list l (x has the same type as the elements from l).

3.2 Defining the bad character list

Initially, all of the values from bc list are equal with -1. In F*, this property can be achieved with a recursive function.

let rec create$_bc(i : nat)$

$: Tot(listint)(decreasesi)$

$= matchiwith$

$|0- > []$

$|_- > (-1) :: (create_bc(i-1))$

Tot is an effect label which indicates that the function will always return a result. The label is optional. The instruction (decreases i) is used to ensure that the recursive call will end, since i decreases until it arrives to 0, the least natural number.[2]

To ensure that the function gives the correct result, I proved that the length of the generated list is at least zero and is equal with the value of the parameter i. I also proved that each value is equal with -1. For each of these properties, I made a separate Lemma. For example, for the last property, the proof looks like this:

let rec index$_{nb}c_is_minusone(i : nat)$

$: Lemma(ensures$

$(letl = (create_bci)in$

$forall(n : nat).n < lengthl ==> indexln = -1))$

$= matchiwith$

$|0- > ()|_- > index_{nb}c_is_minusone(i-1)$

A Lemma is a function which always returns the ():unit value. The type of it carries usefull information about certain properties which are provable. If the proof was verified and no errors were given, then the proof is valid.[2]

Index (index l n) is another function from the FStar.List.Tot.Base library. It returns the n-th item from the list l. As a constraint, the value of n needs to be less than the length of l.

3.3 Defining a function which stores a list of indices

Some usefull functions for the lists are defined in F⋆ libraries. However, there was no function which returns the index where a certain element is stored in a list, so I made one which returns a list with all of the indices where a value can be found, named item$_indices.Themainus$

let rec item$_indices(a : eqtype)(item : a)(l : lista)(i : nat)$

$: listnat$

$= matchlwith$

$|[]- > []$

$|hd :: tl- > ifhd = item$

$theni :: item_indicesitemtl(i + 1)$

$elseitem_indicesitemtl(i + 1)$

In order to return the correct result, the i parameter needs to start with the value 0. In this way, item$_indiceswillhaveforeachitemfromth$

let rec item$_list_has_correct_length(a : eqtype)(l : lista)(i : nat)$

$: Lemma(ensuresforall(item : a).$

$length(item_indicesitemli) = countiteml)$

$= matchlwith$

$|[]- > ()$

$|hd :: tl- > item_list_has_correct_lengthtl(i + 1)$

This is a proof by induction. The base case of the lemma is when l is an empty list. The result of (item$_indicesitemli)isalsoanemptylist, an$

$ithepropertyistrueandweprovethecriteriafor the valuei.The following equalities hold :$

if hd = item then (

length (item$_indicesitemli) =$

$1 + length(tail(item_indicesitemli)) =$

$1 + length(tail(i :: item_indicesitemtl(i + 1))) =$

$1 + length(item_indicseitemtl(i + 1));$

$countiteml =$

$1 + countitemtl$

$)$

$else($

$length(item_indicesitemli) =$

$length(item_indicesitemtl(i + 1));$

$countiteml = countitemtl$

)

The equality remains valid due to the recursive call in the lemma. After a number of steps, the recursion arrives to the base case, where the property is proved. The second criteria of the output data is that each value from the resulted list needs to be in the interval [i, i + length l). For this, I made another lemma.

let rec item$_indices_is_in_interval(a : eqtype)(item : a)$

$(l : lista)(i : nat)(x : nat)$

$: Lemma(ensuresmemx(item_indicesitemli) ==>$

$i <= xx < i + lengthl)$

$= matchlwith$

$|[]- > ()$

$|hd :: tl- > item_indices_is_in_intervalitemtl(i + 1)x$

As in the previous lemma, I used proof by induction. Here, the property is valid for an arbitrary natural number x. We want to demonstrate the implication for all possible values of x. To achieve this, I used forall$_introfunctionfromtheFStar.Classicallibrary.$

Concluzii

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Nunc mattis enim ut tellus elementum sagittis vitae et. Placerat in egestas erat imperdiet sed euismod. Urna id volutpat lacus laoreet non curabit gravida. Blandit turpis cursus in hac habitasse platea. Eget nunc lobortis mattis aliquam faucibus. Est pellentesque elit ullamcorper dignissim cras tincidunt lobortis feugiat. Viverra maecenas accumsan lacus vel facilisis volutpat est. Non odio euismod lacinia at quis risus sed vulputate odio. Consequat ac felis donec et odio pellentesqu diam volutpat commodo. Etiam sit amet nisl purus in. Tortor condiment lacinia quis vel eros donec. Phasellus egestas tellus rutrum tellus pellentesque eu tincidunt. Aliquam id diam maecenas ultricies mi eget mauris pharetra. Enim eu turpis egestas pretium.

Bibliografie

- Author1, *Book1*, 2018
- Author2, *Boook2*, 2017