2016060104002_2017060103023_2

Haodong Liao & Fan Gao

July 19, 2019

1 Preface

This was a LaTeXreport written by Haodong Liao and Fan Gao, students of UESTC who participated in the summer school of UCPH. More codes of this summer school were at this repository.

2 Introduction

Little Claus and Big Claus (Danish: Lille Claus og store Claus) is a tale about a poor farmer, who outsmarts a rich farmer[1]. After analyzing and generating target text according to the story's statistics last week, it was time to move on.

In this assignment, we designed a spell checker which was based on the text by H.C. Andersen. In detail, we used pattern-matching, currying, higher-order function and other powerful tools to complete our functions to meet the requirements, which gave us a sense of accomplishment.

3 Analysis and Design

3.1 Understand the data structure Trie

```
type Trie<'T when 'T : comparison> = {
    IsTerminal: bool;
    Children: Map<'T, Trie<'T>>;}
```

As explained by Prof.Düdder, we would build a ”forest” at the end, which meant when we got a ”root”, there was a whole ”word forest” behind it. If the subtree made up a word, *IsTerminal* was true.

3.2 Lookup Function

```
lookup (prefix: string) (trie: Trie<char>) : Trie<char> Option
```

It was a preparation for completing the *autoComplete* function. In this function worked well, if the *prefix* was ”ab”, then it should return things like ”abstract”, ”absolute” and so on. Which meant there were two branches to be handled when we match the pattern of *prefix*. If it was empty, then *None* should be the output option, else the output should be the option of a trie. To meet the requirement, we could use the *find* function provided in the material. The key part of the pseudocode was shown as following:

```
let lookup (prefix: string) (trie: Trie<char>) : Trie<char> Option =
    match prefix with
    | empty -> None
    | not empty -> use "find" function to return the option of a trie
```

3.3 AutoComplete Function

```
autoComplete (prefex: string) (trie: Trie<char>) : string seq
```

The goal of this sub-assignment was to return an option of the (sub)trie found by the *lookup* function, if the word began with the *prefix* exists, after which *autoComplete* function should return a sequence of strings of words which *prefix* in the trie.

[1] http://andersen.sdu.dk/vaerk/hersholt/LittleClausAndBigClaus_e.html

Same as *lookup*, there were two branches needed to be handled, if the *prefix* argument was empty, then we returned an empty sequence, else we *lookup* the target option of a trie and returned it. The key part of the pseudocode is shown as following:

```
let autoComplete (prefix: string) (trie: Trie<char>) : string seq =
    match prefix with
    | empty -> returns an empty sequence
    | str ->
        match (result of "lookup" function) with
            | Not find -> returns an empty sequence
            | Some x -> returns the sequence
```

3.4 SpellCheck Function

```
spellCheck (word: string) (trie: Trie<char>) : bool
```

We needed to check if a word was correct by checking containment in the given trie. Thanks to the *contains* function given in the material, we could meet the requirement simply by taking the *word* argument as the parameter of *contains* function. The key part of the pseudocode was shown as following:

```
let spellCheck (word: string) (trie: Trie<char>) : bool =
    contains word trie
```

3.5 Generate Random Word

```
randWord (trie: Trie<char>): char list
```

This sub-assignment required to pick a random word from the given trie, and this was the preparation part of *genText* function. The key to this sub-task was to get the "word library" first and pick a word randomly from it. The key part of the pseudocode was shown as following:

```
let randWord (trie: Trie<char>) : char list =
    Get the "word library"
    Generate a random number whose scope covers the entire vocabulary
    Pick a word according to the random number (the position of the word)
```

3.6 Generate a text

```
genText (len:int) (trie: Trie<char>) : string
```

In this sub-assignment, we needed to generate a text based on a length *len* and *trie*, the result was a string consisting of *len* words drawn out of the trie.

The tricky part of this sub-task was how to handle the linked part between words and the last word. The solution was that there should be a "space" between two words and we should deal with the last word separately. So there were three branches to be handled, i.e., *len* was bigger than one, *len* was equal to one. Other situations should be taken as an error. The key part of the pseudocode was shown as following:

```
let rec genText (len: int) (trie: Trie<char>) : string =
    match len with
    | x when x > 1 -> Generate and concatenate the words
    | 1 -> Generate the last word
    | _ -> Error situations
```

4 Program description

4.1 Lookup Function

Our implementation of **lookup** function was as follows:

```
let lookup (prefix: string) (trie: Trie<char>) : Trie<char> Option =
    match prefix with
    | "" -> None
    | str -> find (stringToCharlist str) trie
```

As described in 3.2, the key to this function was to take advantage of the *find* function. To do that, we needed to create our own *stringToCharlist* function so to make the *prefix* argument meet the requiremnt. And the implementation of **stringToCharlist** function was as follows:

```
let rec stringToCharlist (input: string) : char list =
    match input with
    | "" -> []
    | str -> str.[0]::(stringToCharlist str.[1..])
```

4.2 AutoComplete Function

Implementation of **autoComplete** function was as follows:

```
let autoComplete (prefix: string) (trie: Trie<char>) : string seq =
    match prefix with
    | "" -> Seq.empty
    | str ->
        try
            match (lookup str trie) with
            | Option.None -> Seq.empty
            | Some x -> Seq.map (fun x-> x.ToString()) (trieToSeq trie)
        with
            | ex -> eprintf "An exception occurred with message
                    Seq.empty
```

As described in 3.3, to complete the string correctly, we needed to match the pattern of *prefix*, for *prefix* with content, we use *try..with* method to make sure everything go well. In the *try* part, we match the result of *lookup* function with two branches, and we used *Seq.map* to change the *Seq<char list>* to *Seq<string>*.

4.3 SpellCheck Function

Implementation of **spellCheck** function was as follows:

```
let spellCheck (word: string) (trie: Trie<char>) : bool =
    contains (stringToCharlist word) trie
```

As described in 3.4, it was not hard for us to use the given *contains* function to complete this sub-task. And all we needed to do was to convert the *word* argument to *char list* so everything goes well.

4.4 Generate Target Text

Implementations of **randWord** and **genText** function were as follows:

```
let randWord (trie: Trie<char>) : char list =
    let wordList = Seq.toList (trieToSeq trie)  //char list list
    wordList.[Random().Next(1,wordList.Length)] // char list

let rec genText (len: int) (trie: Trie<char>) : string =
    match len with
    | x when x > 1 -> (charlistToString (randWord trie))
                        + " " + (genText (x-1) trie)
    | 1 -> (charlistToString (randWord trie))
    | _ -> "Error"
```

As described in 3.5 and 3.6, to generate a target text, the first step was to pick a word randomly from the given trie. To achieve this, we needed to get the "word library" (achieved at the first line of *randWord* function) as we discussed so that we could pick the word randomly from the library (achieved at the second line of *randWord* function).

The tricky part of the *genText* function was that we confused about the concatenation method of *string*, we used *::* to joint the string and things did not work well. Then we used + to concatenate the string and it worked this time.

Another thing we needed to do was convert the *char list* to *string* so that we returned the text correctly. The Implementation of **charlistToString** was as follows:

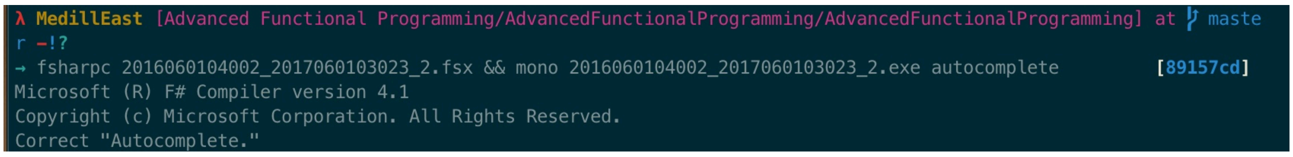
```
let rec charlistToString (input: char list): string =
    match input with
    | [] -> ""
    | fst::rest -> string(fst) + charlistToString(rest)
```

5 Evaluation

The testing environment was macOS Mojave 10.14.5 system with iTerm and Microsoft (R) F# Compiler version 4.1.

5.1 Test of AutoComplete

Except for the given positive test of *"st"; "ca"; "cu"*, negative test of *"gts"; "kns"; "iqw"*, we extended the *"wh"; "ab"; "cl"* into the positive test, and the *"ijk"; "tzw"* into the negative test, and the result of this function was shown as follows:

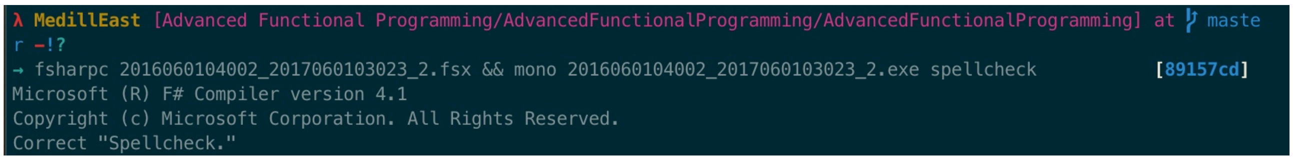


```
λ MedillEast [Advanced Functional Programming/AdvancedFunctionalProgramming/AdvancedFunctionalProgramming] at ᚠ maste
r -!?
→ fsharpc 2016060104002_2017060103023_2.fsx && mono 2016060104002_2017060103023_2.exe autocomplete          [89157cd]
Microsoft (R) F# Compiler version 4.1
Copyright (c) Microsoft Corporation. All Rights Reserved.
Correct "Autocomplete."
```

Figure 1: Testing of autoComplete function

5.2 Test of SpellCheck

Except for the given positive test of *"standing"; "can"; "about"*, negative test of *"gts"; "cfm"; "iqw"*, we extended the *"one"; "four"* into the positive test, and the *"txq"; "ihu"; "svv"* into the negative test, and the result of this function was shown as follows:

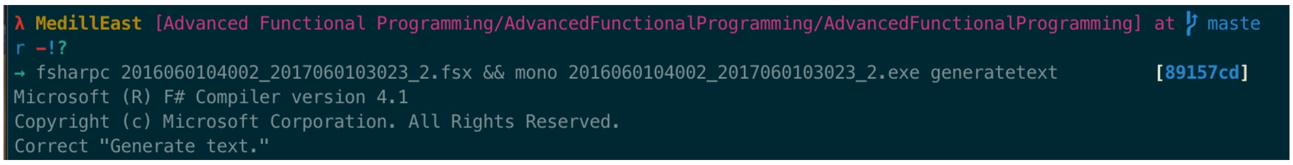


```
λ MedillEast [Advanced Functional Programming/AdvancedFunctionalProgramming/AdvancedFunctionalProgramming] at ᚠ maste
r -!?
→ fsharpc 2016060104002_2017060103023_2.fsx && mono 2016060104002_2017060103023_2.exe spellcheck            [89157cd]
Microsoft (R) F# Compiler version 4.1
Copyright (c) Microsoft Corporation. All Rights Reserved.
Correct "Spellcheck."
```

Figure 2: Testing of spellCheck function

5.3 Test of GenerateText

We test the function as required, and the result of this function was shown as follows:



```
λ MedillEast [Advanced Functional Programming/AdvancedFunctionalProgramming/AdvancedFunctionalProgramming] at ᚠ maste
r -!?
→ fsharpc 2016060104002_2017060103023_2.fsx && mono 2016060104002_2017060103023_2.exe generatetext          [89157cd]
Microsoft (R) F# Compiler version 4.1
Copyright (c) Microsoft Corporation. All Rights Reserved.
Correct "Generate text."
```

Figure 3: Testing of genText function

6 Discussion & Conclusion

As shown above, all of our function worked well as expected.

In this assignment, we designed a spell checker which was based on the text by H.C. Andersen. Thanks to the given powerful function, all we had to do was to figure out the conversion of different types, which reminds us that it is very important to use tools properly to work efficiently. How time flies, yesterday I had to worry about the program, and at this moment we are about to part. It is our pleasure to have Prof.Düdder as our teacher and Sarah as our teaching assistant, and we believe that we had a good time. Many thanks!