Lab Sheet 6

Alireza Dehghani – alireza.dehghani@dcu.ie (mailto:alireza.dehghani@dcu.ie)

Table of Contents

CA320 Homepage
Lab sheet 6: Implementing Pushdown Automata
1. Implementing a Pushdown Automaton
2. Designing Your Own Pushdown Automaton
3. Designing Another Pushdown Automaton

CA320 Homepage (https://ca320.computing.dcu.ie/)

Click on **`CA320 Homepage`** above or in Table of Contents to go back to the course homepage.

Important notes

Here is the ***Einstein*** upload (https://ca320.computing.dcu.ie/einstein/) link.

Deadline for this labsheet is: **`Monday 30th of Nov 9 p.m`**.

Lab sheet 6: Implementing Pushdown Automata

Aim

The aim of this lab is to implement the functionality which will allow the simulation of pushdown automata using Haskell, and to design and run some example pushdown automata using this implementation.

1. Implementing a Pushdown Automaton

Task

You are to implement the functionality which will allow the simulation of pushdown automata. Pushdown automata should have the following type:

```
type PDA = (Int,[Int],[Transition])
```

where the `Int` gives the start state, the `[Int]` gives the accepting states, and the `[Transitions]` gives the transitions of the pushdown automaton.

The transitions of the pushdown automaton should have the following type:

```
type Transition = ((Int,String,String),(Int,String))
```

where in the first tuple `(Int,String,String)`, the Int gives the current state, the first `String` gives the next character of the input (or the empty string if the transition does not consume the next character of the input), and the second `String` gives the character at the top of the stack (or the empty string if the transition does not pop the top character off the stack). In the second tuple of a transition `(Int,String)`, the `Int` gives the new state and the `String` gives the new character to be pushed on to the top of the stack (or the empty string if no character is pushed).

Configurations of a pushdown automaton should have the following type:

```
type Configuration = (Int,String,String)
```

where the `Int` is the current state (this should be the initial state for the starting configuration), the first `String` is the remaining input (this should be the initial input for the starting configuration), and the second `String` is the contents of the stack (this should be the empty string for the starting configuration).

The result produced by a pushdown automaton should have the following type:

```
data Result = Accept | Reject deriving Show
```

This indicates whether or not the initial input string is accepted or rejected by the pushdown automaton.

You are to implement a function `run` which simulates the execution of a pushdown automaton and has the following type:

```
run :: PDA -> String -> Result
```

where the **`PDA`** gives the definition of the pushdown automaton, the **`String`** gives the input string, and the **`Result`** gives the result of running the pushdown automaton on the input string.

You should bear in mind that pushdown automata are non-deterministic in general, so they may have a number of different possible configurations at each step of execution. Your simulation should therefore keep track of all the possible configurations of the pushdown automaton at each step. The possible configurations at the next step will therefore be the result of a single transition from each of the possible configurations at the previous step.

Your simulation should return the result **`Accept`** if any of the current possible configurations of the pushdown automaton correspond to an accepting state, an empty input and an empty stack. Otherwise, your simulation should return the result **`Reject`** if there are no transitions from any of the current possible configurations.

To test your pushdown automaton simulation, try out the pushdown automaton which accepts the language $\{W\epsilon\{a,b\}^* | \, W = W^R\}$. This pushdown automaton is defined as follows:

```
pal = (1,[2],[((1,"a",""),(1,"a")),
              ((1,"b",""),(1,"b")),
              ((1,"a",""),(2,"")),
              ((1,"b",""),(2,"")),
              ((1,"",""),(2,"")),
              ((2,"a","a"),(2,"")),
              ((2,"b","b"),(2,""))])
```

You should obtain the following results when running this pushdown automaton:

```
> run pal "abbabba"
Accept
> run pal "abaaba"
Accept
> run pal ""
Accept
> run pal "ab"
Reject
> run pal "abaabba"
Reject
```

To check your answer is correct (and to register your progress), Save your program to a file named `pdaq1.hs` (this **should include** all of the types defined above as well as your definition of the `run` function), and upload it to **CA320 Einstein page (https://ca320.computing.dcu.ie/einstein)**.

> **This is worth %60 of the marks for this lab**.

2. Designing Your Own Pushdown Automaton

Task

Design a pushdown automaton ijk to accept the language $\{a^i b^j c^k | a, j, k \geq 0 \; and \; i = j \; or \; i = k\}$. This pushdown automaton should accept the following strings:

```
""
"b"
"abbc"
"aabbc"
"aabbcc"
```

And reject the following strings:

```
""
"a"
"aabc"
"abbcc"
```

To submit your answer (and to check that it is correct), you should save your pushdown automaton to a file named `pdaq2.hs` and make sure your pushdown automaton is called ijk (this should have the same format as the definition of the palindrome pushdown automaton shown above, and should **not** include any type definitions or your own implementation of the `run` function), and upload it to **CA320 Einstein page (https://ca320.computing.dcu.ie/einstein)**.

> **This is worth %20 of the marks for this lab**.

3. Designing Another Pushdown Automaton

Task

Design a pushdown automaton abc to accept the language $\{W \epsilon \{a,b,c\}^* | \ the\ number\ of\ c^{'}s\ is\ equal\ to\ the\ number\ of\ a's\ plus\ the\ number\ of\ b's\}$.

This pushdown automaton should accept the following strings:

```
""
"cabc"
"cacbca"
"aabbcccc"
"acbcbcac"
```

And reject the following strings:

```
"a"
"abc"
"cacbcac"
```

To submit your answer (and to check that it is correct), you should save your pushdown automaton to a file named `pdaq3.hs` and make sure your pushdown automaton is called abc (this should have the same format as the definition of the palindrome pushdown automaton shown above, and should **not** include any type definitions or your own implementation of the `run` function), and upload it to **CA320 Einstein page (https://ca320.computing.dcu.ie/einstein)**.

This is worth %20 of the marks for this lab.

Quote of the Day

Courage is found in unlikely places.

— J. R. R. Tolkien

Last updated 2020-12-01 16:42:00 UTC