BILKENT UNIVERSITY ENGINEERING FACULTY DEPARTMENT

OF COMPUTER ENGINEERING

CS 399 SUMMER TRAINING REPORT

Berfin Küçük

December 10, 2019

Chalmers University of Technology

15.07.2019 - 15.09.2019

Contents

1 Introduction

I have done my internship at Chalmers University of Technology which is located in Gothenburg, Sweden. It focuses on research and education in technology, natural science, architecture, maritime and other management areas [1]. My training was at the Computer Science and Engineering (CSE) department. My reasons to choose this place were that I was looking for a place which focuses on research that will contribute to my future academic career, and also I wanted to see the living and working opportunities of Sweden so I may work there in the future. In my internship, I have worked on automata with decision trees to lex Unicode. To briefly explain the project, it aims to improve lexing Unicode with decision trees motivated as follows: With 8bit ASCII characters it was possible to represent each state of an automaton as a 256-entry array, mapping characters to successor states. However, with 32bit Unicode characters, this is no longer possible for space reasons, a single state would already occupy 4GB in memory. Thus, the processing of a Unicode character in an automaton state is done in two stages.

1. Map the character to its character class using decision trees. Only a small number of character classes are needed during lexing.
2. Based on the character class, select the successor state.

I have merged decision trees into automaton transitions, replacing the array that determines the successor state. Instead of assigning a character class, the decision tree would determine the next state.

The project is implemented in Haskell which is a functional programming language.

It is not a group project; I have implemented it individually along with the supervision of my supervisor.

In the rest of the report, detailed company information, and a detailed description of the work done is given, and then performance and outcomes of the internship are explained in depth. In the final part, I summarize the conclusions of my project. Additively, the github link that includes the code written during the internship is attached to the Appendices part to show the work that has been done in detail.

2 Company Information

2.1 About the company

Chalmers University of Technology conducts research and offers education in technology, science, shipping and architecture with a *sustainable future* as its global vision. Chalmers is well-known for providing an effective environment for innovation and has 13 departments. *Graphene Flagship*, a FET Flagship initiative by the European Commission, is coordinated by Chalmers. Situated in Gothenburg, Sweden, Chalmers has 10,300 full-time students and 3,100 employees [2].

2.2 About your department

The Computer Science and Engineering Department is shared between Chalmers University of Technology and the University of Gothenburg and conducts research and education in many fields: algorithms, computer architecture, computer security, distributed systems, electronics design, formal methods, logic, networking, software technology, software engineering, language technology and reliable computer systems [2].

2.3 About the hardware and software systems

Since Chalmers is a university that includes many departments, each department has its own manner of work. At the department that I did my internship each employee uses his/her own personal computer so there is no specific hardware system. The divison I worked in mostly uses functional programming languages such as Haskell and Agda.

2.4 About your supervisor

Supervisor's

***Name* :** Andreas Martin Abel

***JobTitle* :** Senior Lecturer

***Degrees* :**

2013 Privatdozent (Dr. habil.), Faculty of Mathematics, Computer Science, and Statistics, University of Munich

2006 Doctor rerum naturarum (Ph.D.), Department of Computer Science, University of Munich

1999 Informatik Diplom (M.Sc.), University of Munich

3 Work Done

3.1 Project Description and Details

Lexical analysis (short: lexing) is the problem of turning a character stream into a token stream. Usually, token classes are given as regular languages described by regular expressions. The lexer generator turns these regular expressions into finite automata which are then run on the input character stream to produce the output token stream.

With 8bit ASCII characters it used to be possible to represent each state of an automaton as a 256-entry array, mapping characters to successor states. However, with 32bit Unicode characters, this is no longer possible for space reasons, a single state would already occupy 4GB in memory. Thus, the processing of a Unicode character in an automaton state is done in two stages.

1. Map the character to its character class. Only a small number of character classes are needed during lexing.

2. Based on the character class, select the successor state. This mapping could be represented as an array the size of which would be the number of character classes.

Character classes are determined by checking whether a character lies within a certain interval or the union of intervals. At each node there is a decision pivot. We compare the character in question to the pivot element stored in the node, moving to the left subtree if the character is left of the pivot or to the right if it is right of the pivot element.

Finally, we may reach a leaf which assigns a class to the character.

The idea is to merge decision trees into automaton transitions, replacing the array that determines the successor state. Instead of assigning a character class, in this project, the decision tree determines the next state.

In the following two sections, standard lexing and automata along with lexing and automata that I have implemented for Unicode lexing are explained.

3.1.1 Introduction to Standard Lexing and Automata

The task of lexing is to split an input string into tokens. The lexer reads the source code character by character, and sends tokens to the parser [3]. Normally, any character in the input is an ASCII character 0..255 and the automaton can be used for recognition of tokens. Therefore, each state of an automaton can be represented as a 256 entry array for 8 bit ASCII characters. Moreover, the standard first step of compiling regular expressions is generating a non-deterministic finite automaton which thereafter can be converted to a deterministic automaton for efficiency [3].
A Nondeterministic Finite-state Automaton (NFA) is a tuple (Q,S,q_0,F,d) where

Q, a finite set of states

S, a finite set of input symbols

q_0 : Q, the start state

$F \subseteq Q$, the set of final states

$d : Q \times S' \rightarrow P(Q)$, transition function ($S' = S \uplus \varepsilon$).

As stated above, each state in a NFA can possibly be represented as 256-entry array, mapping characters to successor states. To give an example, transition function (d) of the automaton below (Figure 1) can be represented as 2-entry array since it only accepts 0s and 1s:

d =((q0,0),q1),((q0,1),1), ((q1,0),q2),((q1,1),q2), ((q2,0),q2),((q2,1),q2), ((0,0),0),((0,1),1), ((1,0),2),((1,1),0), ((2,0),1),((2,1),2)

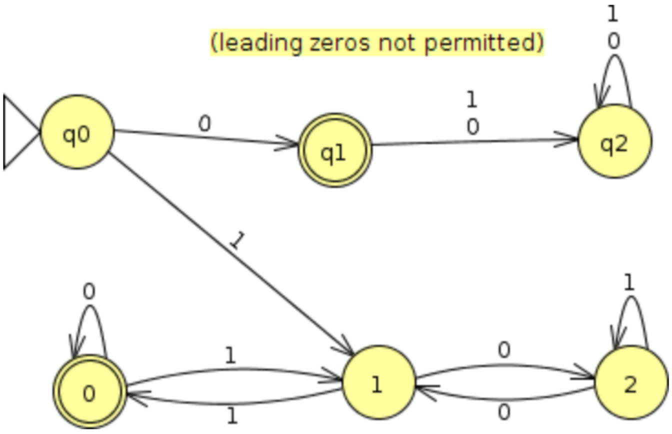

Figure 1: Automaton that leading zeros not permitted(i.e $RE = (0 + 1(0(01^{*}0)^{*}1)^{*})$ [4]

However, with 32bit Unicode characters mapping characters to successor states with a 256-entry array is no longer possible for space reasons because a single state would already occupy 4GB in memory. Thus, automata with decision trees are implemented in this project to lex Unicode which is explained in the next sections.

3.1.2 Automata with Decision Trees

3.1.2.1 Identifying Character Classes

To lex Unicode, a character should be mapped to its character class first. Only a small number of classes are needed during lexing, e.g. to distinguish upper case letters, lower case letters, digits, special symbols, etc. The Unicode Standard Version 12 general category contains 7 major and 30 minor categories [5]. Thus, in that case, 30 categories in addition to individual characters like (,),;, -, *, etc would be needed. Moreover, the determination of character classes can be achieved by checking whether a character lies within a certain interval or union of intervals. In this project, this check is organized in decision trees where at each node the character in question is compared to a pivot element, i.e., a bound. The decision can have three outcomes

a) The character is equal to the pivot. Then, a character class can be assigned right away.

b) The character is less than the pivot. Then, we continue the comparison, advancing into the left subtree.

c) The character is greater than the pivot. Then we go into the right subtree.

Finally, we may reach a leaf which assigns a class to the character. See the example and illustrations below for the detailed explanation.

For example, considering the first 56 characters of The Unicode® Standard Version 12.0 character classes — assuming it consists just first 56 characters of the

Unicode— the boundaries would be as follows (Figure 2). Abbreviations are taken from The Unicode® Standard Version 12.0: Other, control (Cc), Separator, space (Zs), Punctuation, other (Po), Number, decimal digit (Nd) [5].

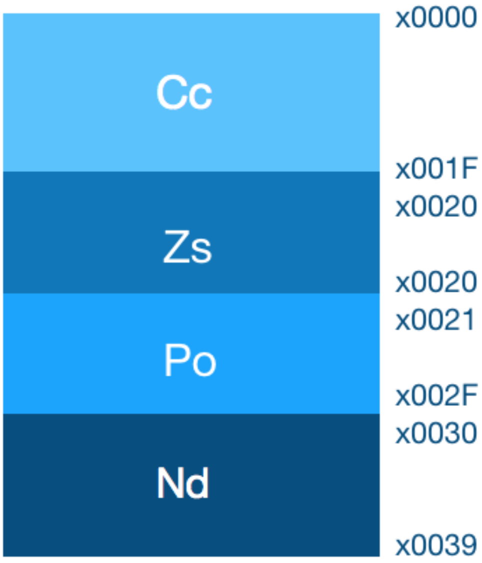

Figure 2: Boundaries of the first 56 characters of Unicode.

A decision tree of the example (boundaries) above is given as follows (Figure 3).

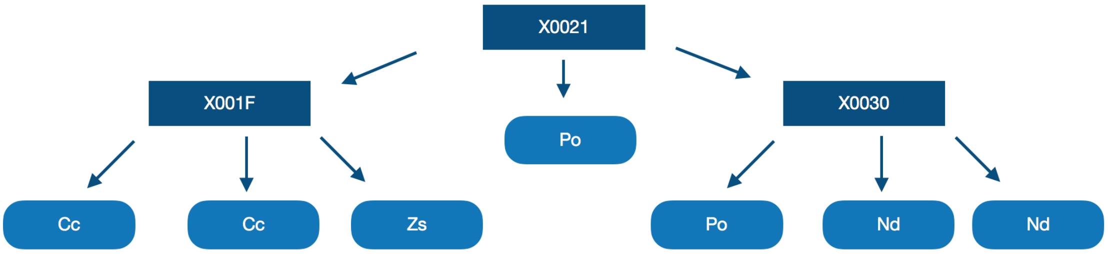

Figure 3: A representation of the boundaries as a decision tree

An alternative decision tree for a representation of the boundaries is given in Figure 4.

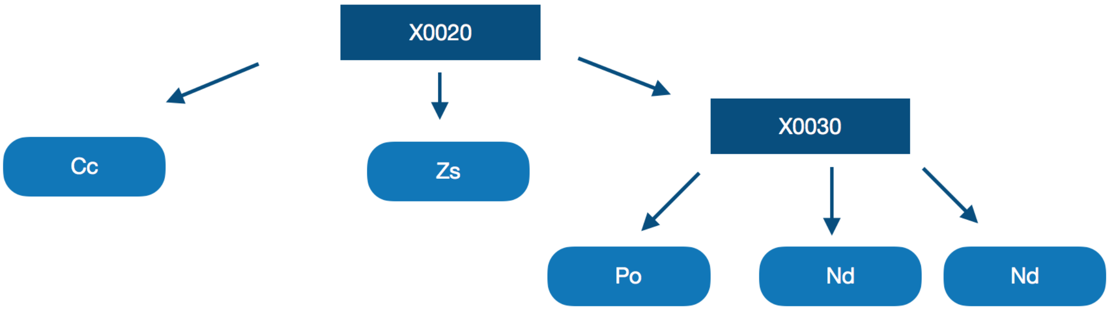

Figure 4: Another representation of the boundaries as a decision tree

The Haskell code for determining character classes along with the data structure of the decision tree would simply be as follows.

```
data DecTree uc
= Leaf uc
| Cmp{ pivot:: Char
, lt :: DecTree uc
, eql :: uc  --character class or nodeid in our cases
, gt :: DecTree uc
} deriving (Show,Eq,Functor)
decide ::  Char -> DecTree uc -> uc
decide x (Leaf uc) = uc
decide x (Cmp p lt eql gt) =
  case (compare x p) of
       EQ -> eql  --match the character to its class
       GT -> decide x gt  --go to gt branch of the tree
       LT -> decide x lt  --go to lt brach of the tree
```

3.1.2.2 Mapping to Successor States

The idea is now to merge decision trees into automaton transitions, replacing the array that determines the successor state. Instead of assigning a character class, the decision tree would determine the next state. The size of the lexer could be smaller using decision trees, in comparison to using arrays. The new automaton model would look as follows: The states are labeled with pivot elements, and have 3 successor states, one for equal to the pivot, one for less than the pivot, and one for greater. Finally, a state can have an action to output the scanned token. To give an example, in Figure 5, there is a simple automaton which illustrates transitions. Numbers inside circles indicate states and arrows indicate character classes for consumed characters.

Now, the core idea is the implementation of a single state with decision trees. Since the boundaries of this example are also needed to draw the decision tree for the automaton in Figure 5, see the boundaries simply selected for this example, in Figure 6.

Thus, an illustration of decision trees for the initial state (State 0 that is indicated as '0') example above would be as follows (Figure 7). Numbers indicate states in the transition table and the boundaries such as b3,b5,b7 indicate characters to

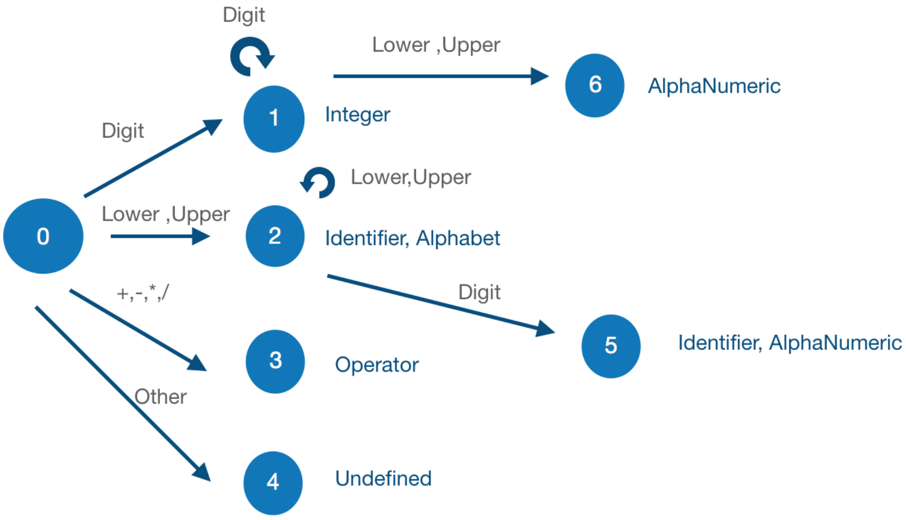

Figure 5: A simple automaton as an example.

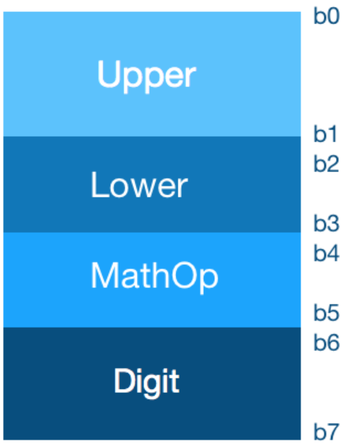

Figure 6: The boundaries of the example in Figure 5.

be compared. In this decision tree, a character is firstly compared with b3 if the character is smaller than b3 then it is a lower or upper character according to the boundaries given in Figure 6, so it will match with State 2 which contains Identifier and Alphabet as tokens. However, if the character is bigger than b3 then it will continue searching in 'greater than' branch of b3 and so on.

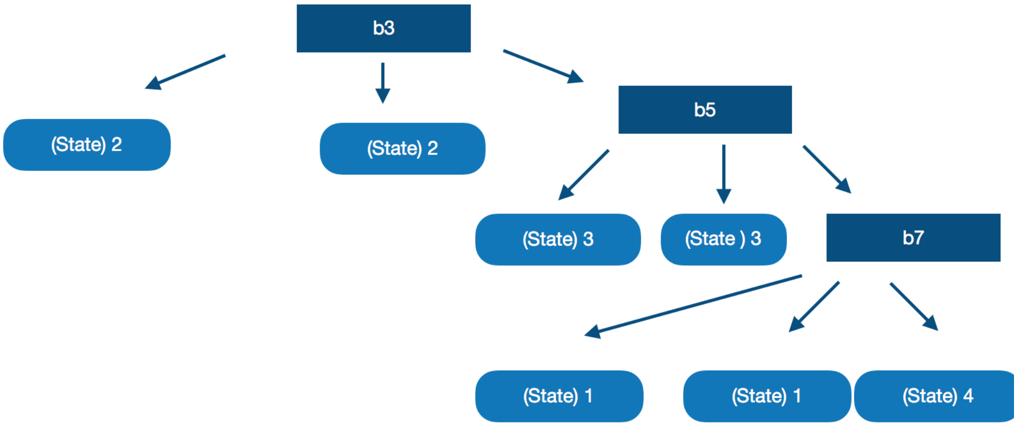

Figure 7: An illustration of one of the decision trees for the example in Figure 5 and Figure 6.

The Haskell code for the decision tree above would be as follows.

```
type NodeId = Int
dectree:: DecTree NodeId
decTree = Cmp "b3" (Leaf 2) 2 (Cmp "b5" (Leaf 3)  3 (Cmp "b7"
        (Leaf 1)1(Leaf 4)))
```

Also, see the 'run' function from the Github link given in the Appendices to see how is the longest match is taken on transitions in an automaton.

3.1.3 Rebalancing Decision Trees with Huffman Coding

The order of comparisons can be arranged such that the most likely transitions require the least amount of comparisons. This would require a statistical analysis which comparisons are carried out how often in each state, and then the comparisons could be rearranged to perform the most frequent comparisons first. This can be done similar to Huffman Trees, which are arranged in a way that the most frequent characters obtain the shortest codes. In this project, an interpreter is developed to run DTAs (Decision Tree Automata) on some input and the statistical information about how often which branches are taken is collected and Huffman algorithm is developed to rearrange DTAs to optimize them according to the collected statistical information.

To rebalance a decision tree to make traversal more efficient, the cost of the decision tree must be taken into account. The cost of a decision tree can be calculated as

$$\sum_i f_i * h_i$$

assuming i's represent leaves of a decision tree while f and h represent frequency and height of a leaf respectively.

Haskell code for calculating the cost of a decision tree is as follows and type of the Statistics is

```haskell
type Statistics = IntMap.IntMap(IntMap.IntMap Freq)
```

and Statistics about how often which branches are taken are collected while our interpreter run DTAs (Decision Tree Automata) on some input.

```
cost :: Statistics -> NodeId -> DecTree NodeId -> Int
cost st i dt = loop dt 0
  where
    f':: NodeId -> IntMap.IntMap Freq -> Freq
    f' i mapf = do
      case (IntMap.lookup i mapf) of
        (Just freq) -> freq
        _ -> 0
    f :: NodeId -> Freq
    f j = do
      case (IntMap.lookup i st) of
        (Just  sndMap) -> f' j sndMap
        _ -> 0
    loop:: DecTree NodeId -> Int -> Int
    loop (Leaf j) k = (f j ) * k
    loop (Cmp p lt eql gt) k = ((f eql) * (k+1)) + (loop lt (k + 1))
                                 + loop gt (k + 1)
    -- it is assumed that eql is at the same level with the lt gt
```

Now, to rebalance a decision tree, firstly, I wrote a function which takes a decision tree and returns the alternate list which will be used to create a new decision tree.

```
data AltList a b = Singleton a
                 | Cons a b (AltList a b) deriving (Show,Functor)
```

```
--(Int,NodeId)
toAltList :: DecTree NodeId -> AltList (DecTree NodeId) (Int,
 ↪  NodeId)
toAltList (Leaf i) = Singleton (Leaf i )
toAltList (Cmp p lt eql gt)  = (join' l1 ((Char.ord p),eql) l2)
     where
      l1 = toAltList lt
      l2 = toAltList gt

join':: AltList a b -> b -> AltList a b -> AltList a b
join' (Singleton a) b l = Cons a b l
join' (Cons a1 b1 l1) b l = Cons a1 b1 (join' l1 b l)
```

For example, the alternate list of the decision tree below is as follows.

```
decTree':: DecTree NodeId
    decTree' = Cmp 'x' (Cmp '-' (Leaf 1) 1 (Leaf 2)) 2 (Leaf 5)

-- the corresponding alternate list of the decTree'
Cons (Leaf 1) (45,1) (Cons (Leaf 2) (120,2) (Singleton (Leaf 5)))
```

Now, it is possible to recreate a decision tree with Huffman-like tree generation as in the code below, see comments to understand the functionality each function

```
    type AltListid = AltList (DecTree NodeId) (Int, NodeId)
    -- Produce a list of worklists where one pair of the original
```

```
  -- worklist forms a new tree.
  cands ::  Statistics -> NodeId -> AltListid
          -> [(Int, AltListid)]
  cands _ _ (Singleton x) = [] --another statament between
  cands st i (Cons dta' (p',nodei') (Cons dtb' (p2', nodei2') aLs))
  = (cost st i dt', w2) :
  Prelude.map  (\ (c,ts) -> (c, Cons dta' (p',nodei') ts )  )
     (cands st i  (Cons dtb' (p2', nodei2') aLs) )
    where
      dt'= Cmp (Char.chr p') dta' nodei' dtb'
      w2 = Cons dt' (p2',nodei2') aLs
  cands st i (Cons dta (p,nodei) (Singleton dtb))
  = [(cost st i dt, w1)]
    where
    dt = Cmp (Char.chr p) dta nodei dtb
    w1 = Singleton dt

-- Create candidate and select the one with minimum cost
-- of the newly created tree.
  step :: Statistics -> NodeId -> AltListid -> Maybe (AltListid)
  step st i aLs@(Cons _ _ _)
  = Just $ snd $ minimumBy (compare `on` fst) $ cands st i aLs
  step _ _ _ = Nothing
```

```haskell
-- Repeat a function until it returns Nothing.
trampoline :: (a -> b -> c -> Maybe c) -> a-> b -> c -> c
trampoline f a b c = case f a b c of
  Nothing -> c
  Just a' -> trampoline f a b a'
```

3.2 Tools and Technologies Learned

At the beginning of this project, I was unfamiliar with the Haskell functional programming language. Since I was more familiar with imperative languages than functional languages, this project was a real challenge to me including both learning a functional programming language and implementing a project in that language. Mostly, I have learned Haskell from the tutorial that my supervisor suggested to me: 'Learn You a Haskell for Great Good!'[6]. During this project, I have also learned details of many libraries from Hoogle Search tool which is a Haskell API search engine that allows us to search many standard Haskell libraries by either function name, or by approximate type signature. I also learned how to add dependencies to my project using cabal which is a Common Architecture for Building Applications and Libraries and a framework for packaging, building, and installing any tool developed in the Haskell language [7].

4 Performance and Outcomes

4.1 Applying Knowledge and Skills Learned at Bilkent

While I was interning, I have realized that I have gained many abilities at Bilkent which can be used in an academic environment like doing a research internship at Chalmers. First of all, the most important skill that I gained at Bilkent is learning independently and deal with even an unprecedented information. The school does not only teach us the engineering curriculum because in many lectures we read, we make researches about the topic and write something about that topic or we are sometimes assigned to do a work which we are not exactly taught how to do it. Therefore, I get used to do a work which I have not encountered before like implementing a project in a language that I did not know before. When I started the internship, I did not have much idea and knowledge about functional programming languages before but even in that time, I felt confident thanks to Bilkent which has shown me that I can handle such things. Moreover, CS315 Programming Languages course that I took at Bilkent helped me to get the idea of the project since the project was about regular expressions, lexing and parsing that I have learned in that course. In addition to that, CS202 Fundamental Structures of Computer Science II and CS473 Algorithms I course that I took at Bilkent helped me while implementing decision trees and Huffman-like tree as well.

4.2 Solving Engineering Problems

During the internship, I tried to solve the problems and issues through internet searching, discussing with the supervisor and also trying to solve problems by thinking on them in depth. Sometimes, I struggled with some problems but I tried to push harder as much as I could do and at the end, I have dealt with the problems related to the project.

4.3 Team Work

At the internship, the project was not a group project which I have implemented individually along with the supervision of my supervisor. Therefore, I haven't experienced the power of team work in that project but it was the first time that I have implemented such a big project on my own and shouldering the all responsibility. Thus, personally I think, working alone in this project was a good experience for me.

4.4 Multi-Disciplinary Work

During internship, I have combined and involved several academic disciplines in the project. I have written a detailed technical report for the description and the details of the project. Moreover, supervision by a instructor who specialized in the topic of my project helped me to see perspective of another discipline of work and apply it to my project. On the other side, I think that if it was a group project, I would experience multi-disciplinary work more due to working with people from different backgrounds. However, during internship, I have attended meetings that I had a chance to learn disciplinary of others by listening work they had done and how.

4.5 Professional and Ethical Issues

During the internship, I haven't encountered any professional and ethical issues.

4.6 Impact of Engineering Solutions

Computers become more prominent and essential in the contemporary world. Many technological devices have a computer within them. Computers are used to complete simple to very complex tasks. Since computers are very significant, this also emerges the significance of computer engineering. Computer engineers are responsible for enhancing current computer technology. However, sometimes they may encounter complex engineering issues anywhere so that this is also their job to solve these problems, and internship was a chance for me to experience these circumstances in real life.

4.7 Locating Sources and Self-Learning

During the internship, I have learned many new tools and used them while doing my work such as learning a new language and using its libraries and tools. Learning of these new tools were experiences of self-learning to me from tutorials and my supervisor, etc. Usually, I investigated the topic myself and learned their details to use these new tools in the most efficient way. I was capable of self-learning before thanks to Bilkent as I stated above. Thus, I improved myself more on self-learning during the internship because I have learned many new tools and used them in the project during the internship.

4.8 Knowledge about Contemporary Issues

Speed and memory use have always been issues since the invention of computer. Eventhough there are great developments regarding speed and memory consumption compared to the first computer invented, these are still issues in the contemporary world which are tried to be improved by many scientists and engineers. In my project, I also tried to understand and solve one of the contemporary issues that are memory problem about lexing Unicode and trying to handle it as fast as possible.

4.9 Using New Tools and Technologies

This part is mostly explained in section 3.2 Tools and Technologies Learned in detail since it is more related to work done. I have learned a lot of the use of tools and new technologies to complete my task and using them throughout the internship helped me to adapt to new environments and solving problems that I haven't encountered before.

5 Conclusions

In conlusion, I have done my internship at Chalmers University of Technology. I have worked at the Computer Science and Engineering (CSE) department and worked on the project that designing and implementing automata with decision trees to lex unicode. My training was quite enjoyable thanks to the Chalmers and my supervisor. Students have flexible working hours which makes work more effective and more pleasant, and also the university hosts really good human relations inside it. Moreover, doing my internship as a research internship gave me confidence for my academic future which shows me that I should focus on improving myself in a specific topic even though it is not related to this project. Tasks assigned to me also gave me that confidence due to the achievement of completed tasks. To sum up what I have learned at training apart from the project, I have learned Haskell which is a functional programming language. Before this internship, I have not implemented a project in a functional programming language. Therefore, it was a challenging experience for me and helped me to think more recursively. In conclusion, my training was fully beneficial for me regarding what I have learned and the people I met.

6 REFERENCES

[1] "Chalmers University of Technology - Maritime Supply Chain Management Education". www.edumaritime.net.

[2] Chalmers. (2019). About Chalmers. [online] Available at: https://www.chalmers.se/en/about-chalmers/Pages/default.aspx [Accessed 10 Sep. 2019].

[3] A. Ranta, "Lecture 4: Lexical Analysis, Regular Expressions, and Finite Automata", Cse.chalmers.se, 2019. [Online]. Available: http://www.cse.chalmers.se/edu/year/2009/c
04.html. [Accessed: 27- Aug- 2019]

[4] Zeil, S. (2019). Finite State Automata. [online] Cs.odu.edu. Available at: https://www.cs.odu.edu/ zeil/cs390/latest/Public/fsa/index.html [Accessed 27 Aug. 2019].

[5] "The Unicode® Standard Version 12.0", Unicode.org, 2019. [Online]. Available: https://www.unicode.org/versions/Unicode12.0.0/ch04.pdfG134153. [Accessed: 28- Aug- 2019]

[6] Lipovaca, M. (2019). Chapters - Learn You a Haskell for Great Good!. [online] Learnyouahaskell.com. Available at: http://learnyouahaskell.com/chapters [Accessed 3 Aug. 2019].

[7] "Applications and libraries - HaskellWiki", Wiki.haskell.org, 2019. [Online]. Available: https://wiki.haskell.org/Applications_and_libraries [Accessed: 10- Sep- 2019]

7 Appendices

The github link that includes the project : https://github.com/berfinkucukk/automata-with-decTrees/blob/master/Dta.hs