

|  |  |
| --- | --- |
| **ASSIGNMENT** | |
| **Course Code** | CSC211A |
| **Course Name** | Formal Languages and Automata Theory |
| **Programme** | B.Tech |
| **Department** | CSE |
| **Faculty** | FET |

#### 

|  |  |
| --- | --- |
| **Name of the Student** | Prachi Poddar |
| **Reg. No** | 17ETCS002122 |
| **Semester/Year** | 4TH/2ND |
| **Course Leader/s** | P.Padma Priya Dharishini |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Declaration Sheet** | | | | | | | | |
| Student Name | Prachi Poddar | | | | | | | |
| Reg. No | 17ETCS002122 | | | | | | | |
| Programme | B.Tech | | | | | Semester/Year | 4th/2nd | |
| Course Code | CSC211A | | | | | | | |
| Course Title | Formal Languages and Automata Theory | | | | | | | |
| Course Date |  | | to | |  | | | |
| Course Leader | P.Padma Priya Dharishini | | | | | | | |
| **Declaration**  The assignment submitted herewith is a result of my own investigations and that I have conformed to the guidelines against plagiarism as laid out in the Student Handbook. All sections of the text and results, which have been obtained from other sources, are fully referenced. I understand that cheating and plagiarism constitute a breach of University regulations and will be dealt with accordingly. | | | | | | | | |
| Signature of the Student | |  | | | | | Date |  |
| Submission date stamp  (by Examination & Assessment Section) | |  | | | | | | |
| Signature of the Course Leader and date | | | | Signature of the Reviewer and date | | | | |
|  | | | |  | | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Faculty of Engineering & Technology | | | |
| Ramaiah University of Applied Sciences | | | |
| Department | Computer Science and Engineering | Programme | B. Tech. |
| Semester/Batch | 4th/2017 | | |
| Course Code | CSC211A | Course Title | Formal Languages and Automata Theory |
| Course Leader | P.Padma Priya Dharishini, Prakash P. | | |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Assignment no 2 | | | | | | | | |
| Name of Student | | | Prachi Poddar | Register No | | 17ETCS002122 | | |
| Sections |  | Marking Scheme | | | Max Marks | | First Examiner Marks | Second Examiner Marks |
| Part-A | A1.1 | Introduction | | | 01 | |  |  |
| A1.2 | Discussion on FLAT aids in designing compilers for programming languages | | | 3 | |  |  |
| A1.3 | Conclusion | | | 1 | |  |  |
|  | **Part-A Max Marks** | | | **5** | |  |  |
| Part B 1 | B1.1 | Introduction and problem definition | | | 01 | |  |  |
| B1.2 | Problem solving approach | | | 02 | |  |  |
| B1.3 | Design and validation | | | 05 | |  |  |
| B1.4 | Concluding remarks (Summary, limitations and improvements) | | | 02 | |  |  |
|  | **B.1 Max Marks** | | | 10 | |  |  |
| Part B 2 | B2.1 | Introduction and problem definition | | | 01 | |  |  |
| B2.2 | Problem solving approach | | | 02 | |  |  |
| B2.3 | Design and validation | | | 05 | |  |  |
| B2.4 | Concluding remarks (Summary, limitations and improvements) | | | 02 | |  |  |
|  | **B.2 Max Marks** | | | **10** | |  |  |
|  | **Total Assignment Marks** | | | | **25** | |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Subject Marks Tabulation** | | | | |
| **Component- CET B Assignment** | **First Examiner** | **Remarks** | **Second Examiner** | **Remarks** |
| A |  |  |  |  |
| B.1 |  |  |  |  |
| B.2 |  |  |  |  |
| B.3 |  |  |  |  |
| B.4 |  |  |  |  |
| **Marks (Max 50 )** |  |  |  |  |
| **Marks (out of 25 )** |  |  |  |  |
| Signature of First Examiner Signature of Second Examiner | | | | |

**Please note:**

1. Documental evidence for all the components/parts of the assessment such as the reports, photographs, laboratory exam / tool tests are required to be attached to the assignment report in a proper order.
2. The First Examiner is required to mark the comments in RED ink and the Second Examiner’s comments should be in GREEN ink.
3. The marks for all the questions of the assignment have to be written only in the **Component – CET B: Assignment** table.
4. If the variation between the marks awarded by the first examiner and the second examiner lies within +/- 3 marks, then the marks allotted by the first examiner is considered to be final. If the variation is more than +/- 3 marks then both the examiners should resolve the issue in consultation with the Chairman BoE.

**Assignment – 2**

**Term - 2**

**Instructions to students:**

1. The assignment consists of 3 questions: Part A – 1 Question, Part B- 2 Questions.
2. Maximum marks is 25.
3. The assignment has to be neatly word processed as per the prescribed format.
4. The maximum number of pages should be restricted to 1**0**.
5. Restrict your report for Part-A to 3 pages only.
6. Restrict your report for Part-B to a maximum of 7 pages.
7. The printed assignment must be submitted to the subject leader.
8. **Submission Date: 18 MARCH 2019**
9. **Submission after the due date is not permitted.**
10. **IMPORTANT**: It is essential that all the sources used in preparation of the assignment must be suitably referenced in the text.
11. Marks will be awarded only to the sections and subsections clearly indicated as per the problem statement/exercise/question

**Preamble:**

This Course is intended to develop an understanding of the concepts of automata theory and formal languages and their relationship to computation models. Students are taught regular, context-free, context-sensitive and universal languages, their generating grammars and properties along with the related automata and machine models. Formal relationships among machines, languages and grammars are covered. Students are trained to design automata and machine models for a given formal language requirements.

**PART A 5 Marks**

**Preamble**

Most of the programming languages use compiler to convert source program to machine understandable code. Each programming languages has specific compiler, some are machine dependent and machine independent. Researchers believe that automata theory play a vital part in designing a compiler phases.

In this context, develop an essay on **“Formal Language and Automata theory (FLAT) aids in designing the Compilers for Programming Languages”**

Your essay should comprise the following:

**A1.1** Introduction

**A1.2** Discussion on FLAT aids in designing compilers for programming languages

**A1.3** Conclusion

# Question No. A

Solution to Question No. A:

## Introduction:

A compiler is a type of translator for computer which converts high level language into object language. It check for syntactical errors in the programs. In simple way it can be said that compiler is a program in any programming language which converts source code to machine language. It is used so because computer can only read in machine language so after converting the source code to machine language, computer reads and performs operation. Compiler has different types of phases. While compiling, process contains sequence of various types of phases. Different phases of compilers are:-

1. Lexical analysis: It is the first phase of complier which scans the source code and convert characters into lexemes and represented as tokens.
2. Syntax analysis or parsing: It is the next phase in which a tree is generated by the tokens as input.
3. Semantic analysis: In this phase tree is checked for its validation i.e it is checked whether the tree follows the rules or not.
4. Intermediate code generation: It generates an intermidiate code which makes the translation code easy
5. Code Optimization: Optimization of the code is done in this phase.
6. Code generation: It converts the obtained intermidiate code and optimized code into machine code.

## Discussion on FLAT aids in designing compilers for programming languages:

Compliers has different types of phases in which deterministic automata theory is used for complilation.

Lexical analysis is the first phase of the compiler phases in which the source code is scanned. Source code are nothing but different types of characters. It reads the characters and then group them into lexemes. Lexeme is a sequence of characters in the source program. From lexemes, a sequence of tokes are produced as output. With the help of dfa, tokens are generated and converted to regular expressions which are sent to the next phase as input. After the conversion of source code to tokens are done then it is send to the next phase that is parser or syntax analysis.

In parser, tokens generated by the lexical analysis is taken as input and then a parse tree is generated. This phase checks for syntactical error. Regular expression formed in the first phase is parser is the input in the this phase. Parser checks if the expression made by the tokens is syntactically correct or not. Validation of regular expression is performed. Parse tree is constructed from the tokens.

In the semantic analysis, parse tree is checked whether the tree is following the rules of languages or not. Using derivation tree, it’s checked. It is based on the concept of context free grammar.

The above three phases are uses automata theory. There are other phases also but only these three phases uses FLAT.

## Conclusion:

From this it can be said that formal language and automata theory has many application and compiler is one of them. In compiler there are many phases but the first three phases are the main phases in which automata theory is used to convert into tokens and tree and checked. If these three phases are valid then only other phases will be processed. After all the phases are processed then only the program will be complied. At last I can say that FLAT has many more important application.

**PART B 20 Marks**

**B1 10 Marks**

Consider a simple Seat Belt Controller (SBC). The requirements for SBC are the following:

* Initially SBC is in idle state
* when a person is seated, not fasten the seat belt within ‘x’ time units and engine is ON, SBC is responsible for automatically switch off the engine
* On fastening of seat belt, SBC allow the person to switch ON the engine
* when a person is seated, not fasten the seat belt within ‘x’ time units and engine is OFF, SBC is responsible for raising an alarm
* On fastening of seat belt, SBC has to switch off the alarm
* When a person is not in seat then SBC has to be in idle state

Design a Push down automata for SBC based on the given requirements.

Document the following:

**B1.1** **Introduction**

A Seat-Belt Controller system is to be made with the following constrains,

* Initially SBC is in idle state.
* When a person is seated, not fasten the seat belt within ‘x’ time units and engine is ON, SBC is responsible for automatically switch off the engine.
* On fastening of seat belt, SBC allow the person to switch ON the engine.
* When a person is seated, not fasten the seat belt within ‘x’ time units and engine is OFF, SBC is responsible for raising an alarm.
* On fastening of seat belt, SBC has to switch off the alarm
* When a person is not in seat then SBC has to be in idle state

The problem requires us to make either a PDA/NDPDA for the seat belt controller process. Since it’s easier to design a ND-PDA (Non-Deterministic Push Down Automata), we would be doing so in this problem.

**B1.2 Problem solving approach**

Replace fastened with sbf, engine\_on with eon, Z with z\_0, seated with s

PDA Tuple Elements

Transition Functions

Context Free Grammar

**B1.3 Design and validation**

Validation

string: **s sbf eon**

Hence we have successfully verified that the input string s sbf eon is accepted by the CGF using the production rules defined earlier.

**B1.4 Concluding remarks (Summary, limitations and improvements)**

Concluding Remarks

In conclusion we have created a ND-PDA for the given Seat-Belt Controller system, which was tested for some inputs in JFLAP, and successfully simulated and ended up in the final state SBC\_OK.

Limitations

The designed PDA has a few limitations such as,

* Since this is a trivial PDA there is no concept of time here, a PDA can only work with one thing at a time, the problem requires a ‘x’ units of time delay which was implemented by pushed ‘x’ into the stack, this can be done formally by using a Timed Push-Down Automata.
* The PDA designed is a one time run thing, i.e. it has to be reset after every usage, this ca be fixed by using more symbols to store the states in the stack, although this would make it more complicated and which is out of scope of this assignment.

Improvements

* The PDA here has very few features, since ND-PDA’s are flexible, it can be further extended to work for more inputs, also a Turing Machine can be made for better control, since in PDA’s we can only work with stack’s top, in Turing machines we can access any arbitrary location data, which makes it easier to handle memory.

**(Parosh Aziz Abdulla, Department of Information Technology, Uppsala University, Sweden)**

# Question 3

Solution to Question 2 Part B

**B.2.1 Introduction**

In formal language theory, a Context Free Language is a language generated by some Context Free Grammar.

The set of all CFL is identical to the set of languages accepted by Push-Down Automata.

Context Free Grammar is defined by 4 tuples as where

V = Set of Variables or Non-Terminal Symbols

∑ = Set of Terminal Symbols

S = Start Symbol

P = Production Rule

Context Free Grammar has Production Rules of the form

Here we are required to make the Context Free Grammar of the Seat Belt Controller designed previously. The Steps of doing so are described in the latter part of this assignment.

**B.2.2 Problem solving approach**

To form the CFG from the PDA, the productions in P are induced by moves of PDA as follows,

1. S productions are given by for every
2. Each erasing move induces production
3. Each non-erasing move induces many productions of form

where each state can be state in

Another common logic that we have to use is that in CFG,

* If the symbol pushed at the beginning is the symbol popped at the end, the stack is empty only at the beginning and the end of P’s computation on x.
* Else the initially pushed symbol must get popped at some point before the end of x, and thus the stack becomes empty at this point.
* For any string x that take P from p and q, starting and ending with an empty stack, P’s first move on x must be a push; the last move on x must be a pop.

**B.2.3 Design and Validation**

A PDA can be converted into a CFG to generate the Language, i.e. the grammar defines the set of strings that the automata accepts. The PDA made in 2.3.1 has to be converted to a Context Free Grammar, this is done manually since JFLAP exceeds out the time limit, as there are too many states to simplify, a few of the states are taken into consideration with the unwanted states removed for a simpler CFG.

**Design**

The Design was created using the rules we have defined in 3.2. The unnecessary rules that we generated are removed and the simplified version of the production rules are as below,

P:

Here “-” represent that any of the states from q0 to q8 can be placed here.

f = fastened

s = seated

e = engine\_on

When the Stack becomes “empty” the string is accepted by the PDA.

**Validation**

Since we have already tried to validate our PDA using the input string

seated fastened engine\_on

which is, if the user is seated, and has fastened seat-belt and turns on engine then the SBC is OK.

We should be able to generate the same using the Production Rules defined above,

Hence, we have successfully generated seated fastened engine\_on as an accepted string by the Context Free Grammer using the Production rules defined.

Another accepted string is,

seated engine\_on x

which is, if the user is seated and turns on engine, as the user has not fastened seatbelt, within x unit of time the engine is turned off and the SBC returns to OK state.

Hence here also our string is accepted, the sequence of rules has been defined above.

**B.2.3 Concluding Remarks**

The Context Free Grammar for the PDA made in 3.2 was generated, that can validate if a string can be accepted by the language or not, a few sets of combinations are taken for validating the CFG, and assuming that it would work for most of the cases, now we analyze the limitations and improvements associated with it.

Limitations

Since our aim was to simplify the automata, we’ve missed some of the production rules in the automata, it works for most of the input strings that could be expected to be worked by the seat belt controller, although not all the transitions are considered here.

Improvements

The CFG was far too complex to take in every possible input string, hence the automata have to be simplified further with fewer states and transition, this can be achieved by finding the Normal form of it, normal forms make it easier to handle because of the simpler structure, so we can find Chomsky Normal Form of the automata and use the CYK-algorithm.

The key advantage of Chomsky Normal Form is that every derivation of a string of n letters has exactly 2n-1 steps, thus one can determine if a string is in the language by exhaustive search of all derivations.

\*\*\*\*\*\*\*