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

PDA Tuple Elements

Transition Functions

Context Free Grammar

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.

# Question 2

Solution to Question 1 Part B

## 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.

## Problem Solving Approach

A ND-PDA differs from an NFA in the case of memory, PDA has an infinite memory stack to keep track of previous transitions while NFA does not. We use this feature of the PDA to keep track of the different states of the Seat-Belt Controller.

The different states chosen for the Automata are,

IDLE, SEATED, FASTENED, ENGINE\_OFF, ENGINE\_ON, WAIT\_STATE, ALARM\_ON, ALARM\_OFF, SBC\_OK

To keep track if the stack is empty we use Z as the symbol in the stack, which indicates that the stack is empty.

The approach in the form of an Algorithm:

Step 1: IDLE

Step 2: Read seated and push seated to stack

Step 3: Read fastened and push fastened to stack

Step 4: If fastened and seated in stack and eon then turn on Engine

Step 5: If only Seated in stack and eoff wait for x units of time, if not fastened then raise alarm.

Step 6: When fastened turn off alarm

Step 7: If only Seated in stack and eon wait for x units of time and go to eoff.

Step 8: If the stack is empty at any of the steps above then go to final state DONE.

To keep track if the user is seated, fastened, and for x units of time wait, we use 3 different stack symbols.

When the user is seated, seated is pushed to the stack, now whenever we want to check if the user was seated or not we can check for this symbol in the stack, if the user unseats, then we can pop seated from the stack.

A similar logic applies for fastened. For example, when the Engine is turned ON, we can check in the stack if fastened and seated exists, then the user is allowed to keep the engine on and the final state SBC\_OK is reached in this case.

Since this is a normal ND-PDA we cannot keep track of time, in that case we would require a Timed Non-Deterministic Push Down Automata, which is not in the scope of this assignment, instead we push ‘x’ into the stack to inform that we are waiting for x units of time and go to the WAIT, here we can check for further inputs, if no inputs are given the element ‘x’ is popped and we transition to the timed\_out state, whichever that may be, in this question it is ALARM\_ON.

## Design and Validation

The Design for the above approach was done in JFLAP 8.0 beta

### Design

The Design is very simple for the reason that we have used PDA’s instead of NFA’s, the number of states required are way less as we can use the Stack to keep track of different previous user inputs.

A Push Down Automata is a 7-tuple structure

The Final State is,

The States are,

The Start state is,

The transition functions are,

The final state here is or SBC\_OK

### Validation

For the Validation we have taken 2 cases, one where the string is accepted and one where it is rejected.

1. w = s sbf eon

Reading the first input symbol s we transition from q0 using function 1, to state q1.

From q1 on reading sbf we transition to q2, using function 2

On reading eon we transition from q2 to state q3 using function 4

From state q3 we transition to q8 using function 12

q8 is the final state and hence the string is accepted.

## 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.
* The timing issue can be solved using a TPDA, which is described as,

# Question 3

Solution to Question 2 Part B

## 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.

## 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.

## 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:

sbf = seat belt fastened

s = seated

eon = 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

s sbf eon

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,

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.

## 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 can be further simplified by taking the Chomsky Normal Form of it,

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.

In formal language theory, a context-free grammar G is said to be in Chomsky normal form (first described by Noam Chomsky) if all of its production rules are of the form.

A → BC, or

A → a, or

S → ε,

Every grammar in Chomsky normal form is context-free, and conversely, every context-free grammar can be transformed into an equivalent one which is in Chomsky normal form and has a size no larger than the square of the original grammar's size.