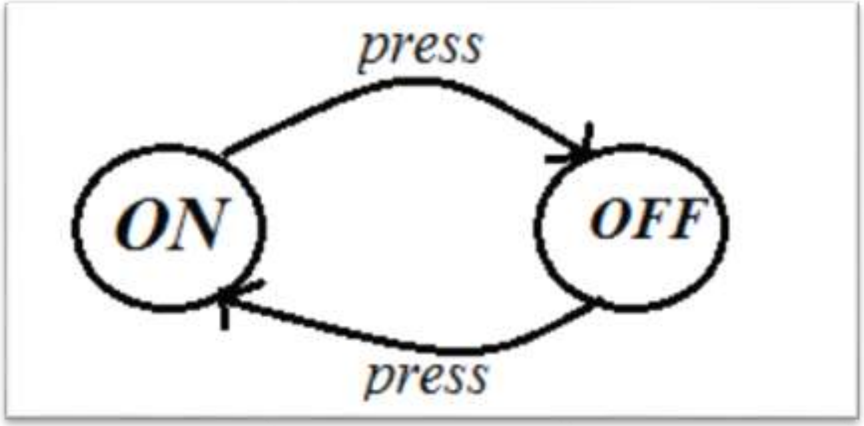
19CSB301 – AUTOMATA THEORY AND COMPILER DESIGN

- ***Blended Course***
- Unit I – Automata Theory
- Unit II to V - Compiler and phases of compiler
- ***What is Automata Theory?***
  - An mathematical model for Abstract machine
  - Prototype
  - Blueprint model
  - Real-Time Example:
    - On-Off Switch
    - Vacuum Cleaner
    - Automatic Car
    - ATM

Automata

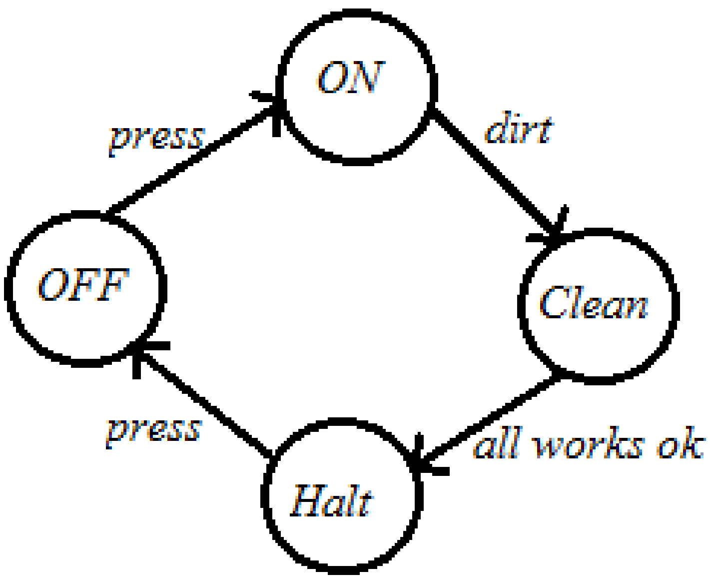
- Mathematical model – *Abstract Machine*
- *Example*
- ON-OFF Switch
  - States and Transition



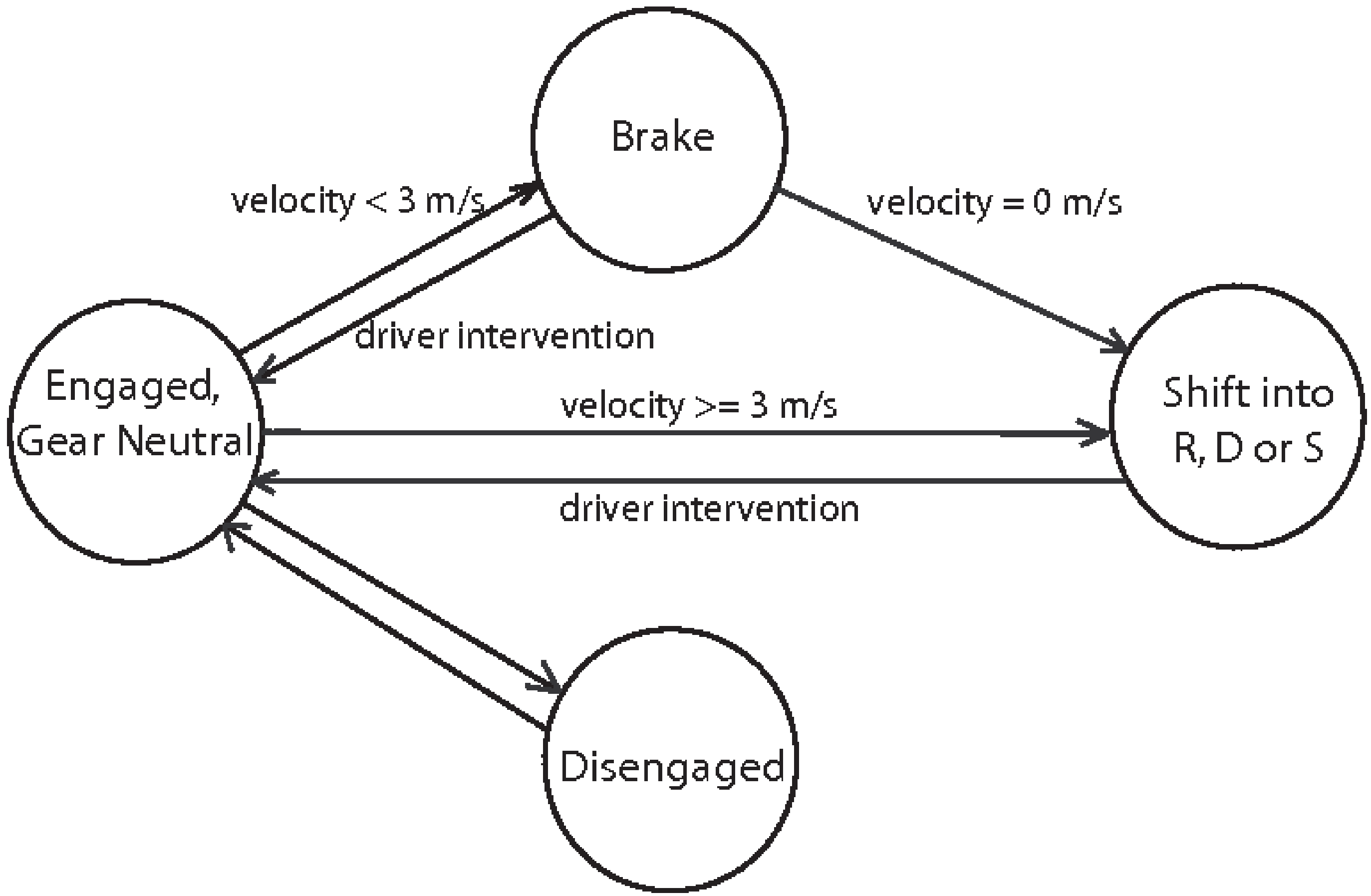
Automata for Switch Model

Vacuum Cleaner

States of a Automatic Vacuum Cleaner

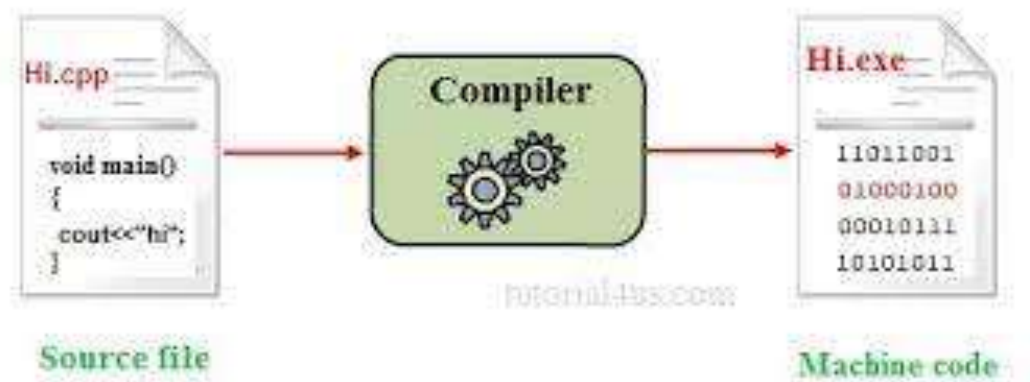


Autonomous Car



Unit 2 to Unit 4

- ***Why compiler design?***
  - To implement a new programming language
  - Design a compiler
  - How the Translation is done?



Central Concepts of Automata Theory

- Symbols
  - {a,b,c,….,0,1,2,….}
- Alphabet
  - Finite set of symbols
  - ∑ (a,b)
- String
  - Collection of Alphabets
  - Example: ∑ (a,b) = a, b, aa, ab, ba, bb, aaa, bbb
  - Length = 1 = {a,b} → |s| = 1 → $n^m=2^1=2$
  - Length = 2 = {aa,ab,ba,bb} → |s| = 2 → $n^m=2^2=4$
  - Length = 3 = {aaa,aab,aba,abb,baa,bab,bba,bbb} → |s| = 3 →$n^m=2^3=8$

Central Concepts of Automata Theory

- Language
  - Collection of Strings
  - Example : $\sum$ (c,d) → L1 = {cc,cd,dc,dd}
  - Finite language
    - L1 = String of length 2 ={cc,cd,dc,dd}
  - Infinite language
    - L2 = String that has at least 1 a
    - L2 = {a,aa,aaa,aaaa,……..
      {ba,aba,bba,abba,…..}
  - Find String {bbbba} in L2 → cannot do manually → Automata
  - Power $\sum^0$ = empty, {0}

Types of Automata

- *Finite State Machine*
  - ***Lexical Analysis*** of Compiler
  - Text Editors
- *Push Down Automata*
  - ***Syntax Analysis*** of Compiler
  - Stack Applications
- *Linear Bounded Automata*
  - ***Semantic Analysis*** of Compiler
  - Genetic Programming
- *Turing Machine*
  - Neural Network
  - Robotic Applications

<table>
<tr><td>UNIT I</td><td>FINITE AUTOMATA AND REGULAR LANGUAGES</td><td>9+6</td></tr>
<tr><td colspan="3">Introduction - Central concepts of Automata Theory - Types of Grammars- Regular Expressions, Identity rules for Regular Expressions - Finite State Automata - Deterministic Finite State Automata(DFA), Non Deterministic Finite State Automata(NDFA) - Equivalence of DFA and NDFA -Pushdown Automata - Languages of a Pushdown Automata -- Turing Machines- Languages of Turing Machine.
Lab Practice: Construction of NFA from Regular Expression. Construction of minimized DFA from a given regular expression</td></tr>
<tr><td>UNIT II</td><td>COMPILERS AND LEXICAL ANALYSIS</td><td>9+6</td></tr>
<tr><td colspan="3">Introduction to Compiling – Compilers – Analysis of the source program – The phases – Cousins – The grouping of phases – Compiler construction tools. The role of the lexical analyzer – Input buffering – Specification and Recognition of tokens – Finite automata – Regular expression to finite automata – A language for specifying lexical analyzer – tool for generating lexical analyzer.
Lab Practice:Implementation of Lexical Analyzer, Implementation of LEX specification.</td></tr>
<tr><td>UNIT III</td><td>SYNTAX ANALYSIS AND SEMANTIC ANALYSIS</td><td>9+6</td></tr>
<tr><td colspan="3">Syntax Analysis – The role of the parser – Context-free grammarsL – Writing a grammar – Top down parsing – Bottom-up Parsing – LR parsers – SLR Parsers – Canonical LR Parsers – LALR Parsers – Constructing an LR parsing table – Tool to generate parser – Semantic Analysis: Type Checking – Type Systems – Specification of a simple type checker
Lab Practice: Construction of LR parsing table.Implementation of syntax analysis using YACC, Construction of Shift Reduce Parser.</td></tr>
<tr><td>UNIT IV</td><td>RUN TIME ENVIRONEMENT AND INTERMEDIATE CODE GENERATION</td><td>9+6</td></tr>
<tr><td colspan="3">Run-Time Environments – Source language issues – Storage organization – Storage-allocation strategies – Intermediate languages – Declarations – Assignment statements – Boolean expressions – Case statements – Back patching – Procedure calls.
Lab Practice:
Generation of code for a given intermediate code generator.</td></tr>
<tr><td>UNIT V</td><td>CODE GENERATION AND CODE OPTIMIZATION</td><td>9+6</td></tr>
<tr><td colspan="3">Issues in the design of a code generator – The target machine – Run-time storage management – Basic blocks and flow graphs – Next-use information – A simple code generator – Register allocation and assignment – The DAG representation of basic blocks – Generating code from DAGs.
Introduction to optimization techniques – The principle sources of optimization – Peephole optimization – Optimization of basic blocks – Loops in flow graphs – Introduction to global data-flow analysis – Code improving transformations.
Lab Practice:
Implementation of DAG representation.</td></tr>
</table>


Thank You