Finite State Automata

COMP360
“It has been said that man is a rational animal. All my life I have been searching for evidence which could support this.”

Bertrand Russell
Reading

Read sections 2.1 – 2.4 in the textbook
Assembler Assignment

• The assembler program is due before we meet again

• Upload the source code of your assembler to Blackboard before 1:00 on Monday, January 27, 2020
Language Recognition

• We say a theoretical machine can recognize a language if it can correctly identify a string of characters as being a syntactically correct program.

• If a theoretical machine can recognize a language, it can identify input that is not in the proper form, i.e., missing a semicolon or improper brackets.
Chomsky Hierarchy

• There are four classes of languages
• Each class is a subset of another class
• Regular languages are the simplest while recursively enumerable (RE) languages are the most complex
## Languages and Machines

<table>
<thead>
<tr>
<th>Language</th>
<th>Machine</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regular Expressions</td>
<td>Deterministic Finite State Automata (DFA)</td>
<td>(a b^* a)</td>
</tr>
<tr>
<td>Context Free</td>
<td>Push Down Automata (PDA)</td>
<td>(a^n b^n)</td>
</tr>
<tr>
<td>Context Sensitive</td>
<td>Bounded Turing Machine</td>
<td>(a^n b^n c^n)</td>
</tr>
<tr>
<td>Recursively Enumerable</td>
<td>Turing Machine</td>
<td>Anything that can be recognized</td>
</tr>
</tbody>
</table>
Finite State Automata

• A FSA has circles to represent states and arrows to represent transitions between states

• Arrows are labeled with a character. That transition is taken if the FSA is in that state and the next input character matches the arrow label

• There is one or more final states (double circle) that indicate the input matches the language

• If there is no matching transition for an input, the FSA rejects the input as a member of the language
FSA for ending in 3 Consecutive 1’s

- State A : no 1s detected
- State B : one 1 detected
- State C : two 1s detected
- State D : three 1s detected

Note that each state has 2 output arrows
What string is NOT accepted by this FSA?

A. accb
B. aac
C. aab
D. c
E. cbab
Nondeterministic FSA

• In Computer Science theory, a Nondeterministic FSA (NFSA) can have multiple out arrows with the same label
• An NFSA magically guesses which transition to take
• Since an NFSA is no more powerful than a deterministic FSA (*and we don’t have magical programs*) we will only use deterministic FSA
• Our FSA cannot have duplicate out arrows with the same label
Deterministic FSAs

• Deterministic FSA: For each state and for each input symbol, there is exactly one transition
• Non-deterministic FSA (NDFSA): Remove this restriction
• At each node there is 0, 1, or more than one transition for each alphabet symbol
• A string is accepted if there is some path from the start state to some final state
• Example nondeterministic FSA (NDFSA): 01 is accepted via path: ABD even though 01 also can take the paths: ACC or ABC where C is not a final state
Equivalence of FSA and NDFSA

• Important early result: NDFSA = DFSA

• Let subsets of states be states in DFSA.
• Keep track of which subset you can be in.

• Any string from \{A\} to either \{D\} or \{CD\} represents a path from A to D in the original NDFSA.
Draw a FSA

• Draw a FSA that accepts money, such as $123 or $665.95
Possible Solution
Regular Expressions

• Regular expressions are often used to specify a regular language
• A regular expression defines a series of symbols in the order they can appear in the language
• Consider a very simple language with only the letters “a” and “b”
• \texttt{aabb} is a regular expression for a string with two letters \texttt{a} followed by two \texttt{b}’s
Regular Expression Alternatives

• A regular expression can specify alternatives using the vertical bar character, |

• `aabb | ba` defines a regular language that accepts two a’s then two b’s or a string with just a b and an a

• You can use parenthesis to group strings

• `(aabb | ba)a` defines a regular language that accepts `aabba` or `baa`
Regular Expression Repetitions

• A symbol or (group) might repeat multiple times in a valid string of the language

• A “*” indicates that a symbol or (group) repeats zero, one or many times

• A “+” indicates that a symbol or (group) repeats once or many times, but at least once

• a$^{0..1}$ mean the symbol may appear zero or 1 time

• $(aabb | ba)^+ a$ defines the strings

  aabbaabba, aabbbbaa, baa and more
Which string is **NOT** \((aa \mid bb)^* a^+\)

A. a
B. bbaabba
C. aaa
D. aabb
E. bbbbbbbaaaaa
Write a Regular Expression

• Write a regular expression to define a string that begins and ends with an a and can have an even number of b’s between them.
Possible Solution

• Write a regular expression to define a string that begins and ends with an \texttt{a} and can have an even number of \texttt{b}'s between them

\texttt{a (bb\textsuperscript{*}) a}
Regular Expressions for Searching

• Regular expressions are often used to define a search target
• The Unix grep utility (globally search a regular expression and print) uses regular expressions
• The Java Pattern class uses regular expressions to define the pattern
Equivalent

• A FSA and a regular expression can define a regular language
• Regular expressions can be recognized by a regular language
Converting a Regular Expression to a FSA

• Consider the regular expression
  \[ a \ (bb) \ * \ a \]

• It can be recognized by the FSA
Try it

• Write a regular expression that defines a double number, such as 12.3 or -12.3
• Use n to represent a numerical digit, 0..9
Possible Solution

• Write a regular expression that defines a double number, such as 12.3 or -12.3
• Use \( n \) to represent a numerical digit, 0..9

\[-0..1 \quad n^+ \quad . \quad n^*\]
Degenerate Case

• 123. and .123 are both valid numbers
• A single period is not a valid number
• There must be a digit before or after the decimal point
• A possible solution might be

\[-0..1 \ (n^+ \ . \ n^*) \ | \ (n^* \ . \ n^+)\]
With Scientific Notation

• Write a regular expression that defines a double number, such as 12.3 or -1.23e1
• Use \( n \) to represent a numerical digit, 0..9

\[-0..1 \ n^+ \ . \ n^* \ (e \ -0..1 \ n^+) \ 0..1\]
FSA to Recognize a Double Number
Draw a FSA

• Draw a Finite State Automata that recognizes comments in the /* form */
• Use “other” to represent any other character
Possible Solution

• Draw a Finite State Automata that recognizes comments in the /* form */
• Use “other” to represent any other character
FSA Limits

• The only “memory” that a FSA has is the current state
• As a FSA inputs symbols, it moves to different states
• A FSA cannot count or compare an arbitrary number of symbols
Push Down Automata

• A PDA has a FSA and a stack memory
• The top of the stack, input symbol and FSA state determine what a PDA will do
• A PDA can:
  • Push a new symbol on the stack
  • Pop the top symbol from the stack
  • Change the FSA state
Push Down Automata Languages

• A Push Down Automata (PDA) can recognize context free grammars
• Almost all modern programming languages are context free grammars
• A PDA can “count” things
• Nobody has developed a programming language more complicated than a context free grammar
Defining a Language

• For a program to be able to recognize a member of a language, you have to be able to accurately and unambiguously define the language

• A regular language can be defined by a regular expression

• A context free language can be defined by a BNF
Compiler’s Purpose

• A compiler converts program source code into a form that can be executed by the hardware
• A compiler works with the language libraries and the system linker
Stages of a Compiler

• Source preprocessing
• Lexical Analysis (scanning)
• Syntactic Analysis (parsing)
• Semantic Analysis
• Optimization
• Code Generation
• Link to libraries
Source Preprocessing

• In C and C++, preprocessor statements begin with a #
• The preprocessor edits the source code based on the preprocessor statements
• `#include` is the same as copying the included file at that point with the editor
• The output of the preprocessor is expanded source code with no # statements
• Old C compilers had a separate preprocessor program
Lexical Analysis

• Lexical Analysis or scanning reads the source code (or expanded source code)
• It removes all comments and white space
• The output of the scanner is a stream of tokens
• Tokens can be words, symbols or character strings
• A scanner can be a finite state automata
I have a laptop with an i5, not a FSA

• You can write a simple program to simulate a FSA
• The simple program can create a list of tokens from source code input
• Simulating a FSA is a very easy way to implement a scanner
Syntactic Analysis

• Syntactic Analysis or parsing reads the stream of tokens created by the scanner
• It checks that the language syntax is correct
• The output of the Syntactic Analyzer is a parse tree
• The parser can be implemented by a context free grammar stack machine
Semantic Analysis

• The Semantic Analysis inputs the parse tree from the parser

• Semantic Analysis checks that the operations are valid for the given operands (*i.e. cannot divide a String*)

• This stage determines what the program is to do

• The output of the Semantic Analysis is intermediate code. This is similar to assembler language, but may include higher level operations
## Output of Compiler Stages

<table>
<thead>
<tr>
<th>Step</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lexical Analysis (scanning)</td>
<td>List of tokens</td>
</tr>
<tr>
<td>Syntactic Analysis (parsing)</td>
<td>Parse tree</td>
</tr>
<tr>
<td>Semantic Analysis</td>
<td>Intermediate code</td>
</tr>
<tr>
<td>Optimization</td>
<td>Better Intermediate code</td>
</tr>
<tr>
<td>Code Generation</td>
<td>Machine language</td>
</tr>
</tbody>
</table>
Reading

Read sections 2.1 – 2.4 in the textbook
Assembler Assignment

• The assembler program due before we meet again
• Upload the source code of your assembler to Blackboard before 1:00 on Monday, January 27, 2020