CSE4083 Formal Languages

Final Exam

Grant Butler | gbutler2020@my.fit.edu

Table of Contents

Deterministic Finite Automata

1. Definition of a DFA

Give a precise (mathematical) definition of a DFA machine `M` and how its parts work together to solve problems.

Solution

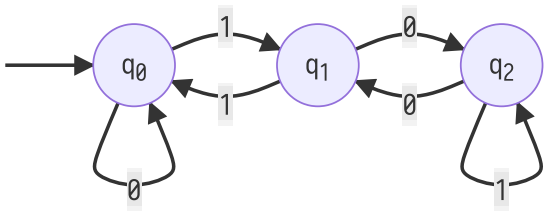
Deterministic Finite Automata (DFA) are 5 tuple, consisting of a set of states Q, a set of symbols Σ, a transition function δ, a starting state s, and an end state F. It is commonly written as:

$$M = (Q, \Sigma, \delta, s, F)$$

DFA's are mathematical models and read a string of symbols over an input alphabet. They **always** either accept or reject the input string.

Typically written as a state diagram, the different parts of a DFA are states Q are vertices, input characters which are written on arcs to show state transitions δ between those states, the initial state s maked with an arrow, and a final state F marked with a double circle.

Example State Diagram of a DFA



Nondeterministic Finite Automata

2. Definition of an NFA

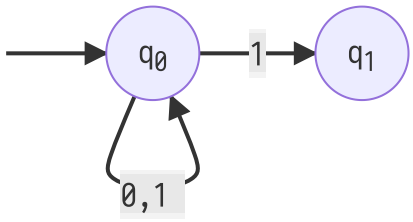
Give a precise (mathematical) definition of a NFA machine *M* and outline how the NFA can be converted into a DFA.

NFA Definition

Similar to a DFA, an NFA is also a 5-tuple, $(Q, \Sigma, \delta, q_0, F)$ as:

- Q: finite set of states
- Σ: finite set of input symbols
- δ: transition function
- q_0: initial state
- F: final (accepting) state

Example State Diagram of an NFA



Converting an NFA to a DFA

An NFA defined as:

$$A = (Q, \Sigma, \delta, q_0, f)$$

can be converted to DFA:

$$B = A' = (Q', \Sigma, \delta, q_0', f')$$

By perfoming the following tasks:

1. Replace states Q with the power set of Q, Q'.
2. Set starting state of B to q_0'.
3. Define all words $\sigma_0, \sigma_1, ..., \sigma_n \in \Sigma*$ and all states $C \in Q'$ with transition function:

$$\delta : (Q' \times \Sigma*) \to Q'$$

Essentially, the DFA B is equivalent in states $C \in Q'$ if and only if the word for the NFA A gives one of the states $s \in C \subseteq Q$. Final state f' is described by every $C \in Q'$ having at least one final state $c \in f$. Therefore, the DFA B accepts the same language and words as the NFA A, so the two are equivalent.

Regular Expressions

3. Regular Expressions in Programming

Describe how regular expressions are use in programming. For example, to define numbers, identifiers, keywords, etc.

Solution

Regular expressions are often used in different programming languages. In Python, for instance, the `re` package can be imported and used to do specific tasks relating to data. For instance, from my CSE3231, I used regex like this:

```
# now to parse the other link
link_regex = re.compile(r'HREF="(?:[^"]|"")*"')  # regex for the href
href_str = link_regex.search(main_html).group()  # getting the link
```

This allowed me to search for an `href` in an html file and find the link.

In the POSIX library, expressions like `[]`, `[:number:]`, `[:lower:]`, and `[:word:]` are used to find different types of characters. These can be used to help searching large amounts of data and filtering it for specific patterns.SS

Closure Properties of Language Classes and Operations

4. List of Closed Classes

Some operations are closed for certain language classes. List classes that are closed with respect to (common formal language operations).

Solution

Closure properties on regular language are defined as operations on regular languages that are garunteed to produce a regular language. Closure means that the operation will result in a language that is the same type as what it operated on.

Closure properties of Regular Languages:

- Kleen Closure
- Positive Closure
- Complement
- Reverse Operator
- Complement
- Union
- Intersection
- Set Difference
- Homomorphism
- Inverse Homomorphism

Decision Problems of Language Classes

5. Types of Questions

Some "yes/no" questions can be answered correctly by some algorithm. Other questions place constrains on parameters to filter the solutions and perhaps select a best answer depending on some criterion. Other questions have no known computable answer.

Pushdown Automata and Context-Free Languages

6. Definition of a PDA

Give a precise (mathematical) definition of a PDA and how its parts work together to solve problems.

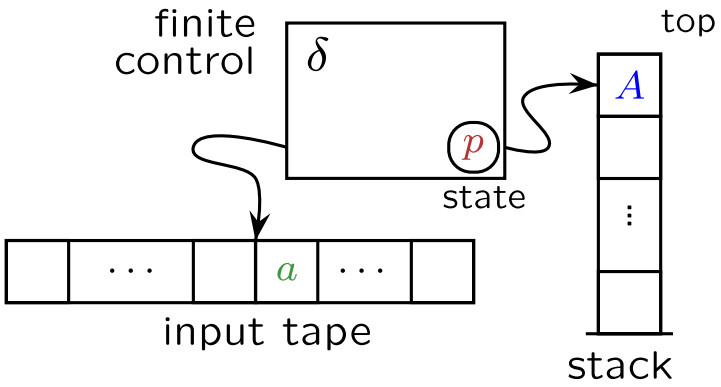
Solution

A pushdown automata (PDA) is described as a 7-tuple:

$$M = (Q, \Sigma, \Gamma, \delta, q_0, Z, F) \text{ where}$$

- Q: a finite set of states
- Σ: a finite set called *input alphabet*
- Γ: a finite set called *stack alphabet*
- δ: a finite subset of $Q \times (\Sigma \cup \varepsilon) \times \Gamma \times Q \times \Gamma^*$, the *transition relation*
- $q_0 \in Q$: start state
- $Z \in \Gamma$: the *initial stack symbol*
- $F \subseteq Q$: the set of *accepting states*

Example of PDA



Production Rule for Context-Free Grammars

7. Ambiguity in Parsing

What is *ambiguity* with respect to parsing a string w for membership in a language L generated by a context-free grammar G? Why does it master? Can it be mitigated?

Solution

In context free grammars, derivation is done by replacing non-terminal symbols with terminal symbols until all non terminal symbols are replaced by terminal ones. Thus, a context free grammar G has:

V: finite set of terminal symbols
T: a finite set of non terminal symbols
P: a set of production rules
S: the start symbol

Ambiguity is when a grammar G manages to have more than one leftmost derivation tree or more than one right hand derivation tree. Ambiguity matters because it causes confusion, and should be removed because each parsing tree will have some meaning, and if a grammar has more than one parsing tree, then the mentioned grammar has more than one meaning. Ambiguity then, will compromise the accuracy of the grammar.

Ambiguity can be removed/mitigated by using precedence rules, semantics, grouping rules, or by improving the grammar itself.

8. Chomsky Normal Form

Most useful grammars can be translated into a *normal form* making answers to questions about the language more transparent.

Solution

A context free grammar is in Chomsky Normal Form if all production guidlines fulfill one of the following situations:

- a non-terminal generating a terminal
- a non-terminal generating two non-terminals
- begin image generating ε

Pumping Lemmas

The Pumping lemma for regular languages is easy to visualize. (if a FSM for a regular language has fewer states than it's input string length, then by the pigeonhole principle, there must be a loop in the machine that can pump a sub-string that can be pumped.)

9. How Pumping Lemma Works

How is this pumping lemma used (what is its proof technique)? Describe the structure of its pre-conditions, and give an example of it's use.

The Pumping lemma for Context-Free languages is more difficult for me to visualize.

(if a PDA for a context-free language has accepts a sufficiently long string then there is an early sub-string and a later sub-string each of which may be pumped.)

If w is a efficiently long string in a CFL then its parse tree can be pruned into 5 parts: A prefix tree, a sub-tree that can be repeated, a middle tree, another repeatable sub-tree, and a boundary tree.

Solution

There is pumping lemmas for ordinary languages and setting free languages. If a language L is context free, then there exists some interger $p \leq 1$ called pumping length, such that every string s in L that has a length of p or more symbols can be written as

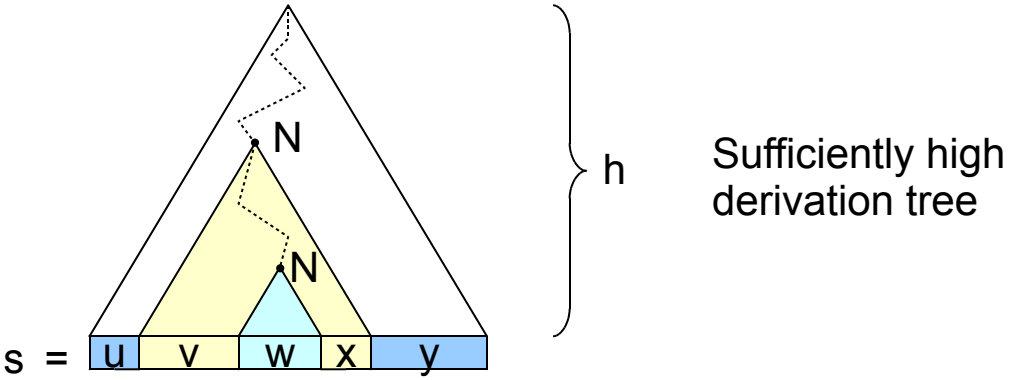
$$s = uvwxy$$

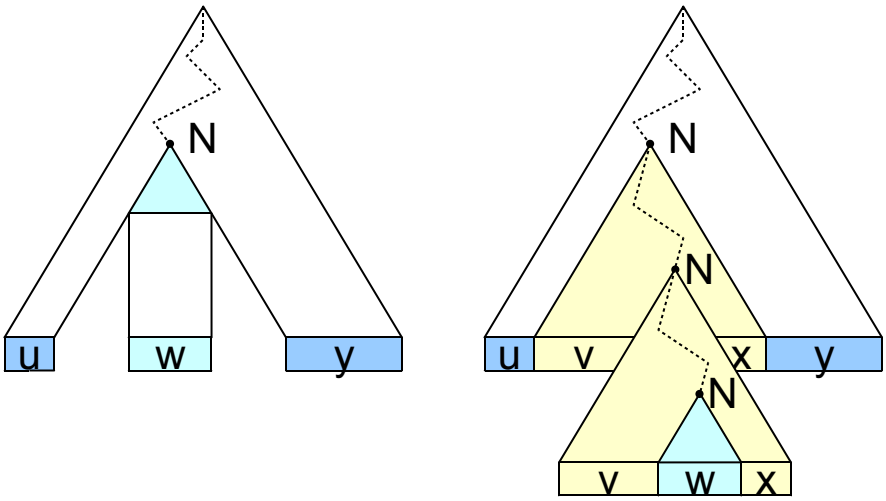
with substrings u, v, w, x, and y such that:

1. $|vx| \geq 1$
2. $|vwx| \leq p$
3. $uv^n wx^n y \in L$ for all $n \geq 0$

Pumping lemma for context free languages is a property all context free languages are garunteed to have. Pumping lemma states that a string s can be split into five substrings, as described above, that can produce a string that is still in the language. The process of "pumping up" the copies of strings v and x is what gives it it's name.

Diagram





Generating $uv^0wx\,y^0$ Generating $uv^2wx\,y^2$

10. Pumping Lemma In Use

How is this pumping lemma used (proof technique)? That is, describe the structure of its pre-condition, and give an example of it's use.

Solution

Pumping lemma is used to prove that a language L is non-context-free by showing that arbitrarily long strings s are in L that cannot be pumped without producing strings outside.

Example

A language $L = \{a^n b^n c^n \mid n > 0\}$ can be proven non-context-free with pumping lemma. A string $s = a^p b^p c^p$ is in L. Using pumping lemma, we can write s as $s = uvwxy$, and then we can use the descibed process to then find that substring vwx can have no more than two distinct symbols. There are five possibilities:

1. $vwx = a^j$ for some $j \leq p$
2. $vwx = a^j b^k$ for some j and k with $j + k \leq p$
3. $vwx = b^j$ for some $j \leq p$
4. $vwx = b^j c^k$ for some j and k with $j + k \leq p$
5. $vwx = c^j$ for some $j \leq p$

It is verifiable that $uv^i wx^i y$ does not contain equal numbers of each letter for any $i \neq 1$. Therefore, $uv^2 wx^2 y$ does not have the form $a^i b^i c^i$, so then the string is not in language L. This means that language L is not context free.

11. Context-Free Grammars in Chomsky Normal Form

When is a Context Free Grammar in Chomsky Normal Form?

A context free grammar is in Chomsky Normal Form if all production guidlines fulfill one of the following situations:

- a non-terminal generating a terminal
- a non-terminal generating two non-terminals
- begin image generating ε

Transition Functions and Relations

12. Definitions of Function and Relation

What are precise (mathematical) definitions of *function* and *relation*?

Solution

A **relation** R from non-empty set B is a subset of the cartesian product $A \times B$. The subset is derived by descrbing a relationship between the first and second elements in the ordered pairs in $A \times B$.

A relation F from a set A to a set B is a **function** if every element of set A has one and only one image in set B.

Functions are a subset of relations.

13. Differences Between Function and Relation

Why is the (major) difference between function and relation interesting as related to classifying machines as deterministic or non-deterministic?

Solution

A deterministic algorithm is one which an outcome is determined by a unique formula. Algorithms where the outcome is not specific is called a non-deterministic algorithm. A non-deterministic algorithm cannot be executed in polynomial-time, while a deterministic algorithm can be executed in polynomial time.

14. State Transition Diagrams

In the study of formal languages state-transition diagrams are often usedto visualize changes in a machine's configuration as it acts on input. To visualize a machine's configuration think of its parts: Commonly a finite set of states, a finite input alphabet Σ, Perhaps storage devices (a stack, input/output tapes). The states are pictured as named circles sometimes decorated with symbols to denote special states, e.g. start and final states. Changes in configuration are denoted by labeled edges andperhaps changes in storage.

Edges in:

a. Finite Automata

Describe how edges are labeled and their meaning for finite state machines.

Solution

Edges in finite automata are used to show transitions from one state to another. Each state is labelled with the input that triggers that transition.

b. Pushdown Automata

Describe how edges are labeled and their meaning for pushdown automata.

Solution

Edges in pushdown automata are used to convey different meanings. An edge can denote three different meanings:

- push character into the stack
- pop a character from the stack
- skip the character

c. Turing Machines

Describe how edges are lebeled and their meaning for Turing machines.

Solution

Edges in turing machines are used to read a character and either go left or right and reach another state or stay in the same state.

Turing Machines

15. Definition of a Turing Machine

Give a precise (mathematical) definition of a Turing machine (TM) and how its parts work together to solve problems. Perhaps, not surprisingly (maybe surprisingly?), this most powerful machine envisioned is also the most simple known. (Other methods of computation have been developed, but each have been shown to be *equivalent* in *expressible* power as Turing machines). What does does this statement mean?

Solution

A turing machine is a 7-tuple:

$$M = \langle Q, \Gamma, b, \Sigma, \delta, q_0, F \rangle$$

where:

- Γ: finite non-empty set of *tape alphabet symbols*
- $b \in \Gamma$: the *blank symbol* (only symbol allowed to occur infinitely at any step)
- $\Sigma \subseteq \Gamma / \{b\}$: set of *input symbols*
- Q: finite non-empty set of *states*
- $q_0 \in Q$: initial state
- $F \subseteq Q$: set of *final* or *accepting* states
- $\delta = (Q \backslash F) \times \Gamma \nrightarrow Q \times \Gamma \times \{L, R\}$: partial function called the transition function, where L is left shift and R is right shift.

There can also be a reject state that makes rejection more specific in a Turing machine. In that case, there are three possibilities: accepting, rejecting, and running forever.

No other type of computational machine can defeat the Turing machine because all computing devices are Turing machines at their core.

16. Problems Unsolvable by Turing Machines

Describe a problem that cannot be solved by a TM. What does *undecidability* mean to you *philosophically*?

Solution

The halting problem is a problem that cannot be solved with the Turing machine. It asks: given an arbitrary Turing machine M over alphabet $\Sigma = \{a, b\}$ and arbitrary string w over Σ, does M halt when it is given w as an input?

That problem is unsolvable with a Turing machine, as it is not decideable, and hence is unsolvable.

Undecidability means something that is impossible to find an answer to because there is always another step to check. Philosphically, I relate this to analysis paralysis, where having more options ends up leading me to a point where I can't even make a decision.

17. Pumping Lemma for Turing Machines

Is there a pumping lemma for Turing machines? If so, what is it? If not, why not?

Solution

There no is pumping lemma for Turing machines, because they can go backwards on their tapes. Because of this, making a pumping is extremely trivial. A Turing machine can deal with the string no matter what combination, so there is no way to pump lemma for a Turing machine.