NLPA Tutorial Language Classes

Mayce Al Azawi

Language Classes

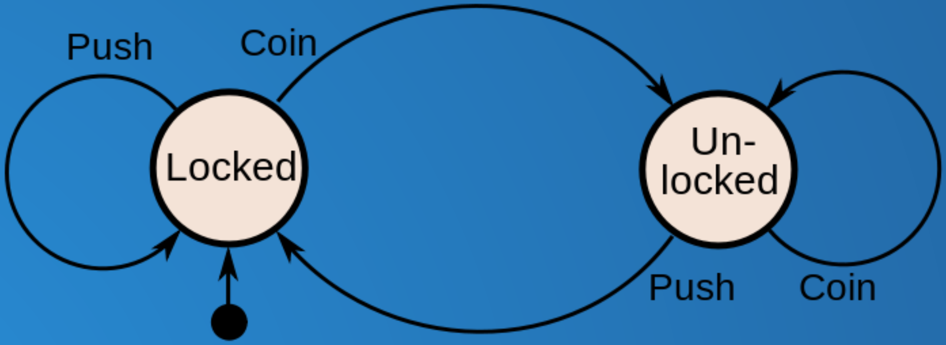
Overview of the following, but you need to read more:

- Machine
  - finite state, pushdown automata, linear bounded automata, turing machine
- Language
  - regular, context-free, context-sensitive, phrase structure, uncomputable
- Algorithms
  - recursive descendant parsing, LR parsing

Finite State Automata

A finite-state automaton (FSA) is a 5-tuple $M = (Q, \Sigma, \delta, i, F)$, where Q is a finite set of states, $i \in Q$ is the initial state, $F \subseteq Q$ is a set of final states, Σ is a finite alphabet and $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow 2^Q$ is the transition function.

Example: Turnstile



Example of FSA

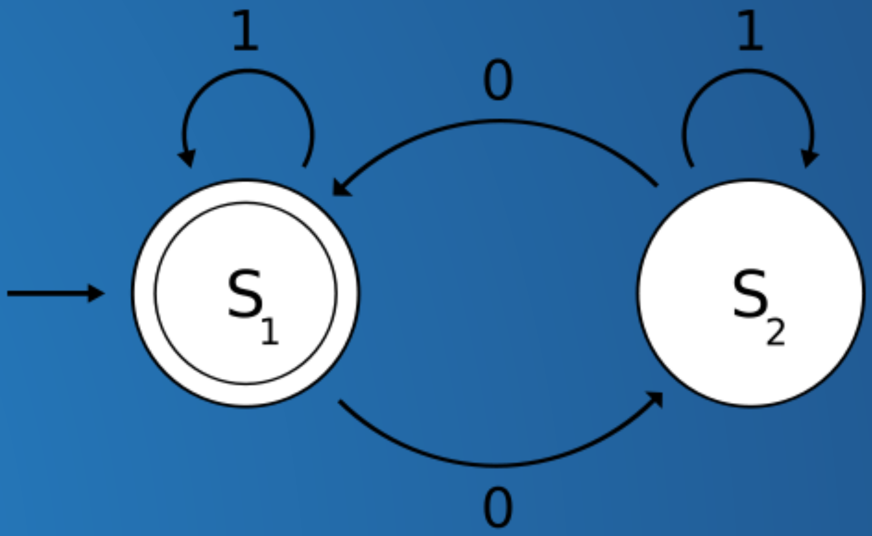
The example shows one that determines whether a binary number has an odd or even number of 0's

Initial and Final state is S_1

Accept binary string has even number of 0's

Example:
1, 11, 11..., 00, 010, 1010, 10110, etc...

later FST


$$\begin{tikzcd} \arrow[r] & S_1 \arrow[loop above, "1"] \arrow[r, bend right, "0"'] & S_2 \arrow[loop above, "1"] \arrow[l, bend right, "0"'] \end{tikzcd}$$

Pushdown automaton PDA

- a type of automaton that employs a **stack**
- PDA has:
  - a finite set which is called the stack alphabet
  - an initial stack symbol
- PDAs can use the top of the stack to decide which transition to take.
- They can manipulate the stack as part of performing a transition.

PDA

- can be used in parser design.
- The deterministic pushdown automaton can handle all deterministic **context-free** languages
- The nondeterministic version can handle all context-free languages.

Turing machine

- is a device that manipulates symbols on a strip of tape according to a table of rules
- on this tape are symbols which the machine can read and write, one at a time, using a tape head.

- has a finite, non-empty set of the tape alphabet/symbols
- has a blank symbol occur on the tape

TM

- is simple
- can be adapted to simulate the logic of any computer **algorithm**
- is particularly useful in explaining the **functions of a CPU**

Linear Bounded Automaton LBA

- is a restricted form of nondeterministic Turing machine
- with a **single tape** and a single tape head
- tape is bounded by (some linear function of) the length of the input, while it is unlimited in the Turning machine
- is acceptors for the class of **context-sensitive** languages

Must know the differences

- FSMs less memory, limited number of states, less computational power than the other
- PDA is more capable than FSM and less than Turing machine
- More advantages and disadvantages: Read more
- Applications: Building LM in MT, ASR, OCR, spell checker, and using in morphological analysis

Languages: Chomsky hierarchy of classes of formal grammars

- Regular Grammar
- Context-Free Grammar
- Context-Sensitive Grammar
- Phrase Structure Grammar

Each formal grammar consists of:

- a finite set of **production rules** (left-hand side and right-hand side) where each side consists of a sequence of these symbols
- a finite set of **nonterminal symbols** (indicating that some production rule can yet be applied)
- a finite set of **terminal symbols** (indicating that no production rule can be applied)
- a **start symbol** (a distinguished nonterminal symbol)

Languages

Grammar	Languages	Automaton Recognizer	Production rules Constraint
Type-0	Unrestricted / Recursively enumerable	Turing machine	a --> b no restrictions
Type-1	Contest-Sensitive	Linear-bounded nondeterministic turing machine	aAb --> acb
Type-2	Context-Free	Nondeterministic pushdown automaton	A --> a
Type-3	Regular	Finite state machine	A --> a A --> aB

Q: differences between R and CF is the stack? Why!

Example CFG

Here is a context-free grammar for syntactically correct infix algebraic expressions in the variables x, y and z:

1. S → x
2. S → y
3. S → z
4. S → S + S
5. S → S - S
6. S → S * S
7. S → S / S
8. S → (S)

This grammar can, for example, generate the string

(x + y) * x - z * y / (x + x)

as follows:

S (the start symbol)
→ S - S (by rule 5)
→ S * S - S (by rule 6, applied to the leftmost S)
→ S * S - S / S (by rule 7, applied to the rightmost S)
→ (S) * S - S / S (by rule 8, applied to the leftmost S)
→ (S) * S - S / (S) (by rule 8, applied to the rightmost S)
→ (S + S) * S - S / (S) (etc.)
→ (S + S) * S - S * S / (S)
→ (S + S) * S - S * S / (S + S)
→ (x + S) * S - S * S / (S + S)
→ (x + y) * S - S * S / (S + S)
→ (x + y) * x - S * y / (S + S)
→ (x + y) * x - S * y / (x + S)
→ (x + y) * x - z * y / (x + S)
→ (x + y) * x - z * y / (x + x)

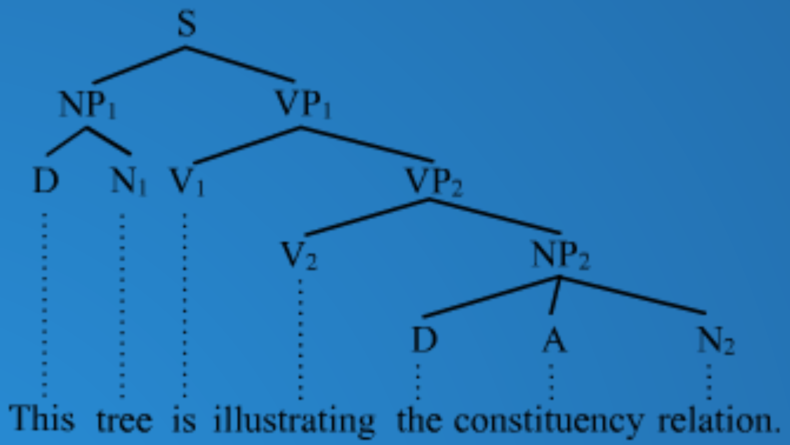
parse tree?

Phrase Structure

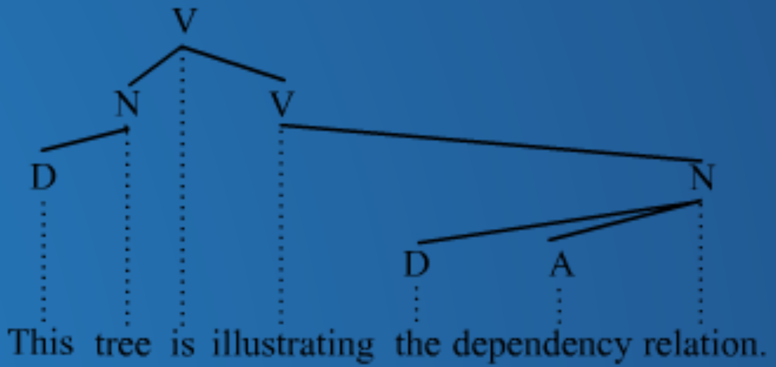
- term for grammars as defined by **phrase structure rules**
- Constituency relation
    - word or group of words that represent a unit, can make up **larger grammatical units**
- Dependency relation
    - one-to-one correspondence
    - for every element (word or morph) in a sentence, there is just one node in the syntactic structure

Example

The constituency tree on the left could be generated by phrase structure rules. The sentence S is broken down into smaller and smaller constituent parts. The dependency tree on the right could not, in contrast, be generated by phrase structure rules (at least not as they are commonly interpreted).



Constituency relation (PSG)



Dependency relation

Algorithms: Recursive Descent Parsing

- top-down parser
- predictive parser
  - no backtracking
  - LL(k) grammar
  - **context free**
  - no left recursion
- Recursive descent with backup technique
  - backtracking
  - may not terminate at all, unless the grammar is LL(k)
  - may take exponential time

LR Parser

- bottom-up parser
- efficiently handle **deterministic context free language**
- linear time

For the exam, you need to know the differences!