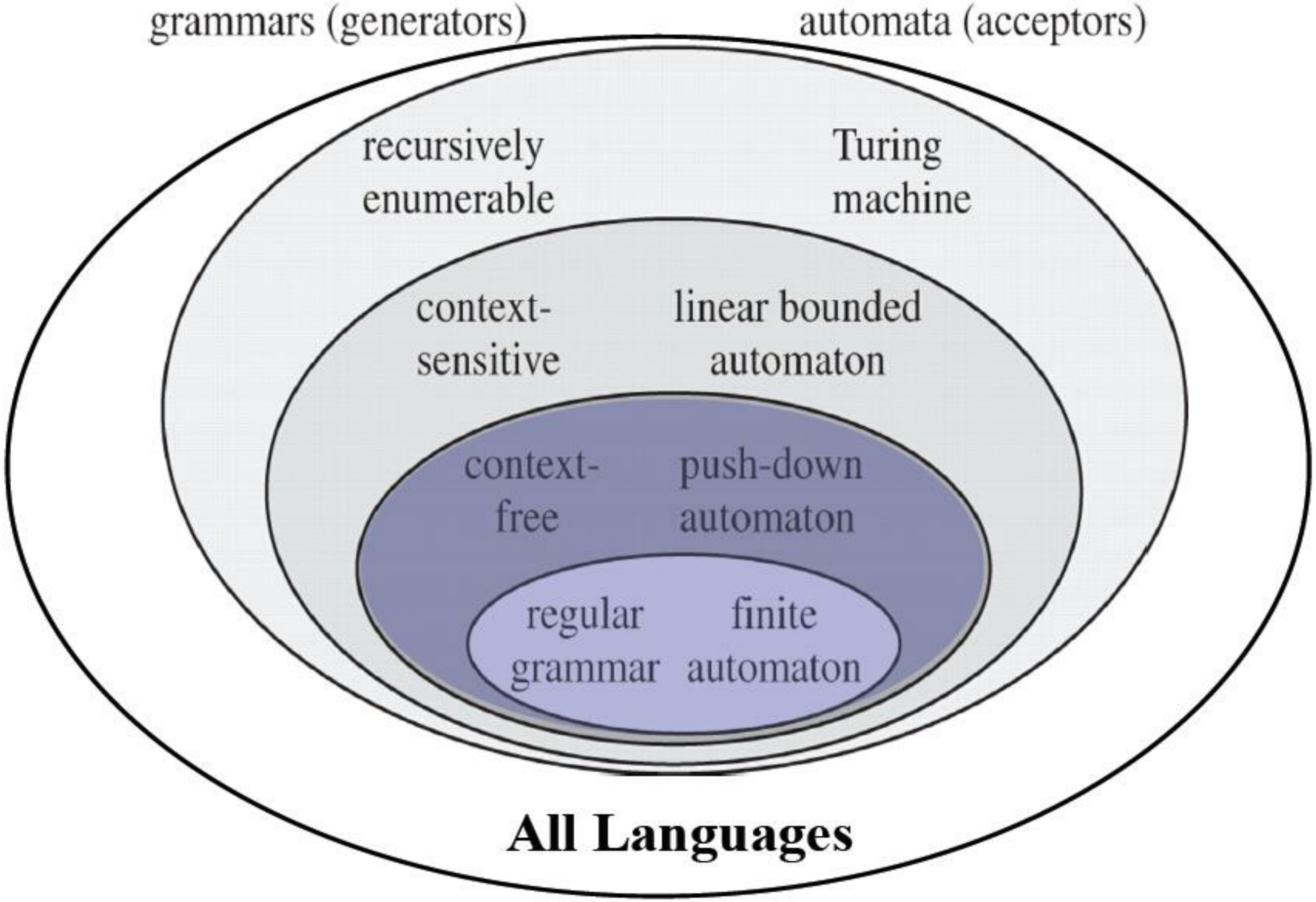
Languages and Computation (COMP2049/AE2LAC)

Pushdown Automata

Dr. Tianxiang Cui

tianxiang.cui@nottingham.edu.cn

The Chomsky Hierarchy



Pushdown Automata

- **Definition**
- A language can be generated by a CFG if and only if it can be accepted by a **pushdown automaton**
- A pushdown automaton is similar to a finite automaton but has an auxiliary unlimited memory in the form of a stack
- LIFO access to the stack - sufficient to handle CFGs
  - Placing words onto the stack is called pushing
  - Taking words off the stack is called popping
- By default, pushdown automata are, nondeterministic. Unlike NFA, the nondeterminism cannot always be removed

Pushdown Automata

- Let's start with a simple example, the language $L = \{a^n b^n \mid n \geq 0\}$
  - This is not a regular language, but it is context-free
- In processing the first part of an input string that might be in L, all we need to remember is the number of a's
  - Saving the actual a's is a simple way to do this
  - So, the PDA will start by reading a's and pushing them onto the stack
- As soon as the PDA reads a b, two things should happen
  - It enters a new state in which only b's are legal inputs
  - It pops one a off the stack to cancel this b
- In the new state, the correct move on the input symbol b is to pop an a off the stack to cancel it. Once enough b's have been read to cancel the a's on the stack, the string read so far is accepted

Pushdown Automata

- The stack has no limit to its size, so the PDA can handle anything in L
- A single move of a PDA will depend on the **current state**, the **next input**, and the **symbol currently on top of the stack** (the only one the PDA can see)
- In the move, the PDA is allowed to **change states** and to **modify the top of the stack**
  - In many stack applications, the only legal stack moves are to push a symbol on and to pop one off
  - Here, to make things a little simpler, we will allow the PDA to replace the top symbol X by a string α of stack symbols
- Special cases are pushing a symbol Y (replacing X by YX) and popping X (replacing X by ε)

PDA Formal Definition

- A pushdown automaton is a 7-tuple $A = (Q, \Sigma, \Gamma, q_0, Z_0, F, \delta)$, where:

1. A finite set of states Q
2. A finite set of input symbols (alphabets) Σ
3. A finite set of stack symbols Γ
4. An initial state $q_0 \in Q$
5. An initial stack symbol $Z_0 \in \Gamma$
6. A set of accepting (or final) states $F \subseteq Q$
7. A transition function $\delta: Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \rightarrow P_{fin}(Q \times \Gamma^*)$
   - where $P_{fin}(S)$ are the finite subsets of a set
   - $P_{fin}(S) = \{ X \mid X \subseteq S \wedge X \text{ is finite}\}$

PDA: Transition Function

- Because values of δ are sets, PDA A may be nondeterministic
  - It may have a choice of transitions from any state
- A move requires that there be at least one symbol on the stack. Z_0 is the one on the stack initially
- Typically, the transition function is of the form:

$$\delta\ (q,\ a,\ X) = \{(p,\ Y),\ldots\}$$

1. Make a state transition from q to p
2. a is the **next** input symbol
   - Remove a from the front of the input, a can be ε
3. X is the **current** stack **top** symbol
4. Y is the **replacement** for X, it is in Γ^* (a string of stack symbols)
   - Replace X on the top of the stack by Y

PDA: Example

- Consider the PDA A that can recognize the language $L = \{a^n b^n \mid n \geq 0\}$ as following

$$A = (Q = \{q_0, q_1, q_2\}, \Sigma = \{a, b\}, \Gamma = \{a, \#\}, q_0, Z_0 = \#, F = \{q_2\}, \delta)$$

where

$$\delta(q_0, a, \#) = \{(q_0, a\#)\}$$
$$\delta(q_0, \varepsilon, \#) = \{(q_2, \#)\}$$
$$\delta(q_0, a, a) = \{(q_0, aa)\}$$
$$\delta(q_0, b, a) = \{(q_1, \varepsilon)\}$$
$$\delta(q_1, b, a) = \{(q_1, \varepsilon)\}$$
$$\delta(q_1, \varepsilon, \#) = \{(q_2, \#)\}$$
$$\delta(q, w, x) = \emptyset \text{ everywhere else}$$

- Let's try the input string $aaabbb$

Instantaneous Description (ID)

- At any time instance, the state of the computation of a PDA $A = (Q, \Sigma, \Gamma, q_0, Z_0, F, \delta)$ is given by
  1. The state $q \in Q$ the PDA is in (i.e. the **current state**)
  2. The portion of the input string $w \in \Sigma^*$ that has not yet been read (i.e. the **remaining input**)
  3. The **contents of the stack** $\gamma \in \Gamma^*$ (the convention will be that the top corresponds to the leftmost symbol)
- Such a triple $(q, w, \gamma) \in Q \times \Sigma^* \times \Gamma^*$ is called **Instantaneous Description** (**ID**)
  - Or sometimes it is called the **configuration** of a PDA
- We define a relation $\vdash_A$ between IDs, $(q, x, \alpha) \vdash_A (p, y, \beta)$ means that PDA A can move in one step from one ID to the next one
  - $\vdash_A$ can be read as "goes-to"
  - Or, one of the possible moves in the first configuration takes A to the second
- $\vdash_A^n$ and $\vdash_A^*$ refer to n moves and zero or more moves, respectively

Instantaneous Description (ID)

- If we have $id_1 \vdash_A id_2$, meaning that PDA A can move in one step from id_1 to id_2. Since PDAs in general are nondeterministic, there are two possibilities:

1. $(q, xw, z\gamma) \vdash_A (p, w, \alpha\gamma)$ if $(p, \alpha) \in \delta(q, x, z)$
2. $(q, w, z\gamma) \vdash_A (p, w, \alpha\gamma)$ if $(p, \alpha) \in \delta(q, \varepsilon, z)$

   where $q, p \in Q$, $x \in \sum$, $w \in \sum^*$, $z \in \Gamma$, $\alpha, \gamma \in \Gamma^*$

- In the first case, the PDA reads an input symbol x, and consults the transition δ function to calculate a possible new state p and a sequence of stack symbols α that replaces the current top symbol z on the stack
- In the second case, the PDA ignores the input, moves into a new state and modifies the stack as above. The input is unchanged

Example

- Using the previous example PDA that recognizes the language $L = \{a^n b^n \mid n \geq 0\}$, we can describe the sequence of moves for the string *aaabbb* as follows:

$$(q_0, aaabbb, \#)$$

$$\vdash_A (q_0, aabbb, a\#)$$

$$\vdash_A (q_0, abbb, aa\#)$$

$$\vdash_A (q_0, bbb, aaa\#)$$

$$\vdash_A (q_1, bb, aa\#)$$

$$\vdash_A (q_1, b, a\#)$$

$$\vdash_A (q_1, \varepsilon, \#)$$

$$\vdash_A (q_2, \varepsilon, \#)$$

- What happens with the input string *aaabbbb*?

The Language of A PDA

- Generally, there are two "flavors" of PDAs
- **Acceptance by final state**

$$L(A) = \{\, w \mid (q_0, w, Z_0) \vdash_A^* (q, \varepsilon, \gamma) \wedge q \in F \,\}$$

- This says that the language of PDA A is equal to all words, w, such that

1. A can move from (q_0, w, Z_0) to (q, ε, γ) in zero or more steps, **and**
2. q is a final state

- (q_0, w, Z_0) to (q, ε, γ) are both IDs:
  - (q_0, w, Z_0) is the starting ID
  - (q, ε, γ) is the final ID

The Language of A PDA

- (q_0, w, Z_0)
  - PDA A is at **starting** state q_0, the remaining input string is the **entire word w**, and the content of the stack is Z_0
- (q, ε, γ)
  - PDA A is at state q, which is a **final** state. The remaining input string is **empty (ε)**, and the contents of the stack are γ (we don't care about this, it can be **anything**)
- So, as long as PDA A reaches a final state, with **no** remaining input string, the word is **accepted** – **even if the stack is not empty**

The Language of A PDA

- **Acceptance by empty stack**

$$L(A) = \{ w \mid (q_0, w, Z_0) \vdash_A^* (q, \varepsilon, \varepsilon) \}$$

- This says that the language of PDA A is equal to all words, w, such that

1. A can move from (q_0, w, Z_0) to $(q, \varepsilon, \varepsilon)$ in zero or more steps

- (q_0, w, Z_0) to $(q, \varepsilon, \varepsilon)$ are both IDs:
  - (q_0, w, Z_0) is the starting ID
  - $(q, \varepsilon, \varepsilon)$ is the final ID

The Language of A PDA

- (q_0, w, Z_0)
  - PDA A is at **starting** state q_0, the remaining input string is the **entire word w**, and the content of the stack is Z_0
- $(q, \varepsilon, \varepsilon)$
  - PDA A is at state q, which can be **any** state. The remaining input string is **empty (ε)**, and the stack is also **empty (ε)**
- A PDA that accepts by final state can be converted to an equivalent PDA that accepts by empty stack and vice versa
- Both types of PDAs thus describe the same class of languages, the **Context-Free Languages** (CFLs).
- Note: under both conditions, input word w is accepted **only if all** the input string is read in

PDAs And CFGs

- **Theorem**
- For a language $L \subseteq \Sigma^*$, $L = L(G)$ for a CFG G *iff* $L = L(A)$ for a PDA A
- i.e. the CFGs and the PDAs describe the same class of languages
- Proof: By constructing a PDA A from a CFG G and vice versa such that $L(A) = L(G)$
- We will look at constructing a PDA from a CFG

Translating A CFG Into A PDA

- Given the CFG G=(N, T, S, P), we can construct the PDA as follows:

$$A = (Q = \{q_0\}, \Sigma = T, \Gamma = N \cup T, q_0, Z_0 = S, F = \emptyset\ , \delta)$$

where

$$\text{For every } E \in N,\ \delta(q_0, \varepsilon, E) = \{(q_0, \alpha) \mid E \rightarrow \alpha \in P\}$$

$$\text{For every } t \in T,\ \delta(q_0, t, t) = \{(q_0, \varepsilon)\}$$

$$\delta(q, w, \gamma) = \emptyset \text{ everywhere else}$$

- **Acceptance by empty stack**
- Allow expansion of non-terminal by pushing right-hand side of production onto the stack
- Remove terminal symbols from the stack as they are produced and match them with input symbols

Example

- Consider the grammar G:

$$E \rightarrow 0E0 \mid 1E1 \mid \varepsilon$$

- Construct the PDA A from G
- Show the input string 0110 is accepted by A
  - We may only need to show that at least one accepting configuration can be reached
- Might be a little bit harder to show the non-acceptance as we must show there are **no** accepting sequence
  - Try **all** the possibilities

Deterministic PDAs (DPDAs)

- A DPDA is a PDA that has **no** choice:
- A PDA $A = (Q, \Sigma, \Gamma, q_0, Z_0, F, \delta)$ is deterministic iff

$$|\delta(q, x, z)| + |\delta(q, \varepsilon, z)| \leq 1$$

$$\text{for all } q \in Q, x \in \Sigma, \; z \in \Gamma$$

- Example: the PDA for the previous example is not a DPDA

$$\text{e.g., } |\delta(q_0, 0, E)| + |\delta(q_0, 1, E)| + |\delta(q_0, \varepsilon, E)| = 3 > 1$$

Deterministic PDAs (DPDAs)

- DPDAs are important because they can be implemented efficiently
- But unfortunately:
- **Theorem**
- The set of languages accepted by the DPDAs is a **strict subset** of the languages accepted by PDAs: $L(\text{DPDA}) \subset L(\text{PDA}) = \text{CFL}$
  - This is in contrast to the case for finite automata, $L(\text{DFA}) = L(\text{NFA}) = $ Regular language
- However, most context-free languages of practical importance can be described by DPDAs

Concluding Remarks

- Pushdown automaton
  - Similar to a finite automaton, but with unlimited memories
  - Transition function
  - Instantaneous description / configuration
- The language of a PDA
- PDAs and CFGs
  - Translate a CFG into a PDA
- Deterministic PDAs