ITS64304 Theory of Computation

School of Computer Science

Taylor's University Lakeside Campus

Lecture 3: Pumping Lemma and Pushdown Automata (PDA)

Dr Raja..

Learning outcomes

At the end of this topic you should be able to:

- Use pumping test to determine a given language is regular or not.
- Design a PDA for a given specification

Aligns to Course Learning Outcome 2.

Regular Languages

Theorem 1: A language *L* is accepted by a DFA **iff** *L* is accepted by an NFA **iff** there is a regular expression for *L*

- A language *L* as above is known as a **regular language**
- A language is regular, if it can be represented as:
  - a regular expression
  - an NFA
  - a DFA

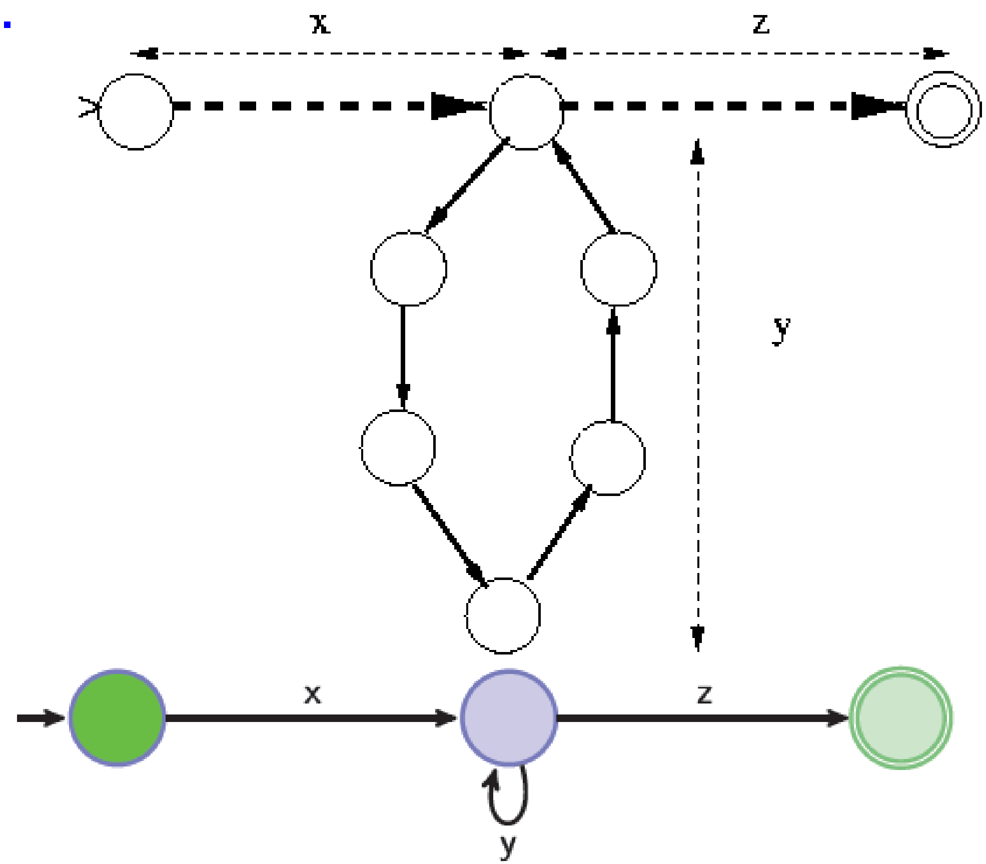
Limitations of FSAs

We have seen:

- FSAs have a finite number of *states*
  - Thus have a finite amount of 'memory'
  - thus.. can't accept $a^n b^n$ for unlimited $n > 0$
  - There is **NO FSA/FSM for $\{a^n b^n \mid n \geq 0\}$**
  - as number of a's to remember may be larger than finite number of states in the FSA
  - So, not all languages are regular
- But we know FSA can accept some unlimited strings... e.g. ab^na
- What is special about these strings or regular languages?

FSAs & 'loops'

- Consider FSA *M* for accepting some *w*
- If |*w*| >= *n* (number of states), then *M* must contain a *loop* in order for it to accept *w*
- For e.g.



FSAs & 'pumping'

- FSA can be built to accept strings having predictable cycles
  - thus regular languages can have cycles

- *Pumping a string refers to constructing new strings by repeating (pumping) substrings in the original string*
  - e.g. FSA for ab^na
  - pump-up *aba* to *abba* - still accepted

FSAs and ‘pumping’

- The “pumping” observation can be used to determine if a language is regular or not!
  - FSA accept regular languages
  - regular strings can be “pumped” into a given string and still have it accepted by the machine

Pumping Lemma

Theorem: **The Pumping Lemma**

- Let L be a regular language.
- Then there exists a $n \geq 1$ such that
  - any string $w \in L$ with $|w| \geq n$ can be written as $w = xyz$ such that:
    - $|xy| \leq n$
    - $|y| > 0$
    - $xy^iz \in L$ for all $i \geq 0$ (i.e. we can "pump" y)

  - NOTE: *$|w|$ represents length of string s, y^i – I copies of y are concatenated together and y^0 equals λ*

Pumping Lemma

- Intuitively, this means that
  - if a string is accepted by a DFA, and the string is longer than the number of transitions in that path,
    - then that DFA path must contain a cycle
- Pumping Lemma does not tell us where the cycle is;
  - just that there must be at least one somewhere
- We can use the pumping test to prove a language is **not** regular - if string w violates the pumping test

- But if the string w passes the pumping test, that is no guarantee the language is regular
  - perhaps we haven't found the 'right' w

Pumping Lemma – Example 1

- Example: $L = ab^n a \; n>0$
  - We already know it is regular, so we would expect it to pass the pumping test

- Let $n = p - 1$
- Let $s = ab^{p-1}a$, thus $|s| \geq p$ is true
- Pumping lemma guarantees s can be split into xyz where:
  - $|y| > 0$
  - $xy^iz \in L$ for all $i \geq 0$
  - $|xy| \leq p$

Pumping Lemma

- Have:
  - $x = a$, $y = b^{p-1}$, $z = ba$
- Is there a p such that $|y| > 0$?
  - Yes: $p = 2$ gives $|b^{2-1}| > 0$
- and that has $|xy| \leq p$?
  - Yes: $|ab^{p-1}| \leq p$
- and that has $xy^iz \in L$ for all $i \geq 0$?
  - Yes: $xz = aba$, $xyz = abba$, $xyyz = abbba$, $xyyyz = abbbba$ etc for all i. All xy^iz are $\in L$
- Thus ab^na passes the pumping test and might be regular

Pumping Lemma – Example 2

- Example: Consider $(aa)^n \quad n \geq 0$
- Set $n = p-1$ and have:
  - $x = e$, $y = (aa)^{p-1}$, $z = aa$
- Is there a p such that $|y| > 0$?
  - Yes: $p = 2$ gives $|aa| > 0$
- and that p gives $|xy| \leq p$?
  - Yes: $p = 2$ would give $|aa| \leq p$
- and has $xy^iz \in L$ for all $i \geq 0$?
  - Yes: xz = aa, xyz = aaaa, xyyz = aaaaaa, xyyyz = aaaaaaaa etc for all *i*. All xy^iz are $\in L$
- Thus $(aa)^n$ $n \geq 0$ might be regular

Pumping Lemma – Example 3

- Example: is $L = \{ a^n b^m c^n \mid n,m \geq 0\}$ regular?
- Let n = m and have $s = a^p b^p c^p$
- Pumping lemma says $|y| > 0$ and $|xy| \leq p$
- So y can contain only a's
- $y = a^q$ for some $1 \leq q \leq p$
- $x = a^{p-q}$
- $y = a^q$, $x = a^{p-q}$ for some $1 \leq q \leq p$
- $Z = b^p c^p$

- Can we pump y?
- $xyyz = a^{p-q} a^q a^q z = a^{p+q} b^p c^p \notin L$

- Contradicts pumping lemma thus L not regular

Conclusion

- DFA is an efficient algorithm for testing membership of L(M)
- FSMs are not powerful enough as some simple languages cannot be represented
- We need to add memory
- PDA and TM are more powerful than FSMs

Pushdown Automata

- For example, consider $\{ww^R \mid w \in \{a,b\}^*\}$

- Needs to “remember” 1st half of the string
- requires some memory “in reverse”

- Another application: how to balance parentheses?

PDA

- Count the parentheses from left to right
  - +1 for each '('
  - -1 for each ')'
- If the count becomes negative, reject the string
- Accept if the count is 0 at the end of the string

((( ) ((( ))) ( ))) √

1 2 3 2 3 4 5 4 3 2 3 2 1 0

(((((( )))) ( ))) √

1 2 3 4 5 6 5 4 3 2 3 2 1 0

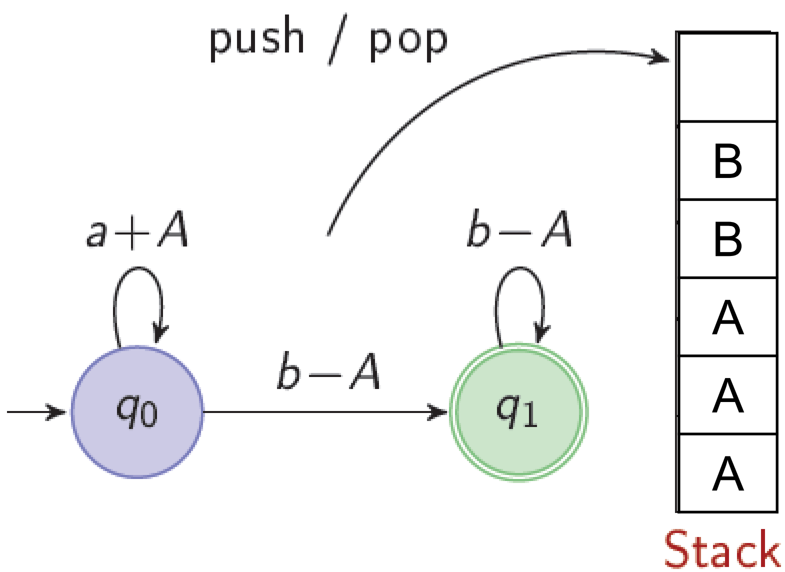
((( ) ((( )))))) (X

1 2 3 2 3 4 5 4 3 2 1 0 -1 0

- Need to store the sequence not the count
- L = strings over ∑= { (,) } with balanced brackets.
- We add a stack to FSM to get PushDown Automaton (PDA)

PDA

PDA for accepting L = $\{a^n b^n \mid n \geq 0\}$



Input: aaabbb √   Input: aaabb X

- Stack Operations
  - Push
  - Pop
- Initially the stack is empty
- String accepted if
  - It finishes in a final state
  - Stack is empty
- String rejected
  - Stack not empty at end
  - Not in a final state at end
- Empty stack popped (violation)
- PDAs can be nondeterministic, similar to context-free grammars

PDA

Definition 8: A push-down automata is a sextuple M = (Q, ∑, δ, Γ, s, F) where

- (same) Q is a finite set of states
- (same) ∑ is an alphabet
- (new) Γ is the stack alphabet
- (same) $q_0 \in$ Q is the initial state
- (same) F $\subseteq$ Q is the set of final states
- (new) δ, the transition relation is a finite subset of

  (Q x (∑ u {e}) x Γ*) x (Q x Γ*)

PDA

The transition:

(p,a,b) (q,r)

- Current state = p
- Current input symbol = a
- Current stack top = b
- New state = q
- New stack top = r *(stack replacement)*

PDA

- The empty symbol *e* is allowed as input symbol and stack top
  - An *e* as input means input is not read.
  - An *e* as stack top in (x, y, λ) means ignore stack
  - An *e* as stack top in (w, λ) means pop stack

(p, λ, λ) (p, a)

  - Push *a* onto stack without reading input or changing state

(p, λ, a) (p, λ)

  - Pop *a* from stack without reading input or changing state

(p, a, λ) (q, λ)

  - Read input *a* and change state from *p* to *q* (ignore stack)
  - This is equivalent to a finite automata transition

PDA

Example: PDA for $\{wcw^R \mid w \in \{a, b\}^*\}$

M = (Q, ∑, δ, Γ, s, F) where Q = {s, f}, ∑ = {a, b, c}, Γ = {a, b}, F = {f} and δ is :

(s, a, e) (s, a)

(s, b, e) (s, b)

(s, c, e) (f, e)

(f, a, a) (f, e)

(f, b, b) (f, e)

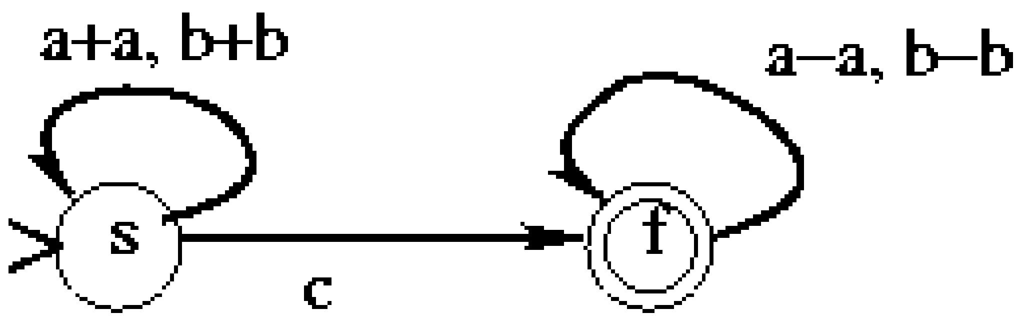
PDA

State	Unread Input	Stack
s	abbcbba	λ
s	bbcbba	a
s	bcbba	ba
s	cbba	bba
f	bba	bba
f	ba	ba
f	a	a
f	e	λ

(s, a, λ) (s, a)
(s, b, λ) (s, b)
(s, c, λ) (f, λ)
(f, b, b) (f, λ)
(f, a, a) (f, λ)

PDA

- We can also represent PDAs via state diagrams

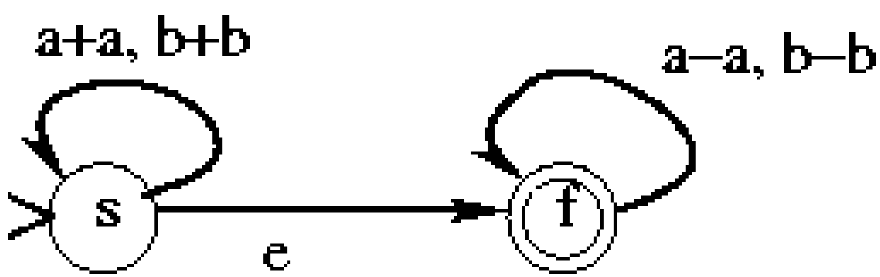


$L = \{wcw^R \mid w$ is a string over $\{a, b\}\}$

- In general, a transition is represented as a - α + β, meaning pop α from the stack and push β onto it

PDA

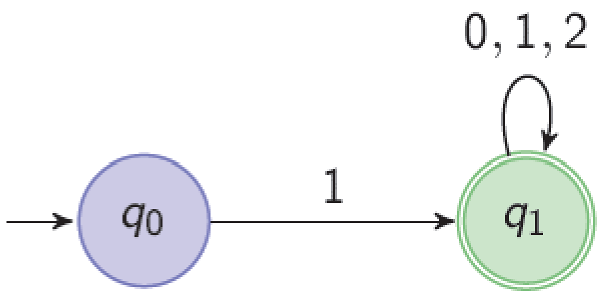
- Example: Let L = {ww^R | w $\in$ {a,b}*}
- We have the same transitions as before except that ((s,c,e), (f,e)) becomes ((s,e,e), (f,e))



- As before, some computations will not accept the string, but as long as at least one does, the string is accepted overall

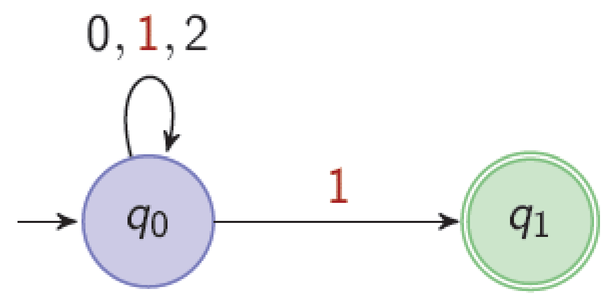
PDA: Non-determinism

- PDAs are by definition non-deterministic



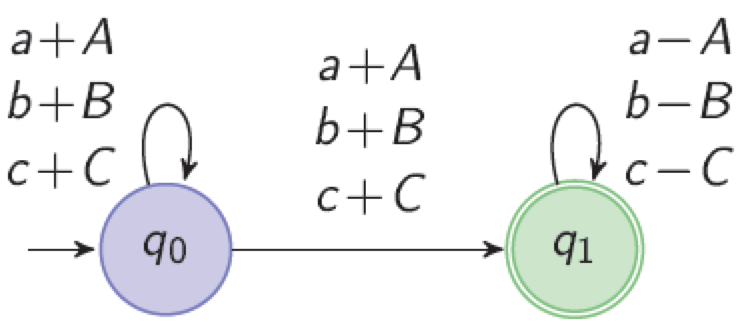
L = {strings over {0,1,2} starting with 1}

Deterministic



L = {strings over {0,1,2} ending with 1}

Non-deterministic



$L = \{w \mid w \in (a \mid b \mid c)^+, w = w^R\}$

Conclusion

- FSA and PDA as language acceptors
  - Input is a string
    - Processed one symbol at a time
    - From left to right
  - Output is either ‘yes’ or ‘no’
  - Memory
    - FSM = current state
    - PDA = current state + stack
  - Acceptance condition
    - FSM = Final state
    - PDA = final state + empty stack

Read your lesson materials

Look at the Tutorials, prepare for it…..