CS 536 Review

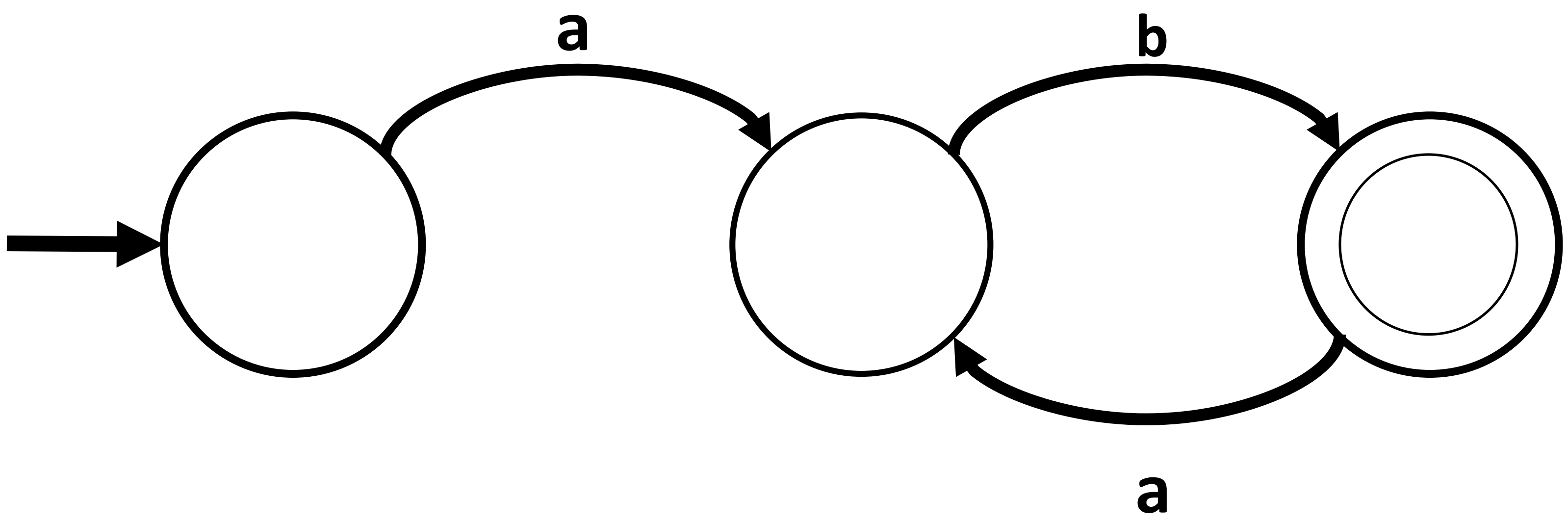
Finite State Automata

- A DFA can be defined by a quintuple (Q, Σ, δ, s, F) where
  - Q is a finite, non empty set of states.
  - Σ is the input alphabet.
  - δ is the transition function δ: Q x Σ -> Q
  - a ϵ Q is the initial state.
  - F c Q is a set of accepting states. Note this need not be non-empty!
- δ need can be a partial function, but δ as a total function is required for some algorithms in their default form. (See Project 2)

Finite State Automata Continued

- NFAs are similar to DFAs in that they are a quintuple (Q, Σ, δ, a, F) except δ: Q x (Σ U {ε}) -> P(Q) where P(Q) is the power set of Q
- Despite the added flexibility of epsilon transitions and non-determinism, they are no more powerful than DFAs!
- This symmetry is broken when moving to more expressive languages and complex automata
- Is the language $\{ (ab)^n \mid n \geq 1\}$ a regular language?

Yes it is!



Another Example

- Is the language $\{ a^n b^n \mid n \geq 1\}$ a regular language?
- It is not! DFAs have no way to “store” information such as the number of a’s written.
- If you don’t believe me, I challenge you to come up with a DFA that does accept the above language. I will pay you 100 dollars for the rights to use this automata in a paper I will then publish.
- This language is context free however.

Context Free Grammars

- CFGs are defined to be 4 tuple G=(V, Σ, R,S) where:
  - V is a finite set where each v ϵ V is a *Variable*. *Variables* are non terminal characters than define a sublanguage of G.
  - Σ is the set of *Terminal Characters* of G, which are disjoint from V. This is the actual content of the grammar.
  - R is a relation (V, (V U Σ)*) known as the *Production Rules* of G
  - S is the start variable. Is analogous to S in DFA's
- CFGs are more sophisticated than Regular Languages as
  - Tokens become grammatical phrases
  - Structure in the program can be accounted for

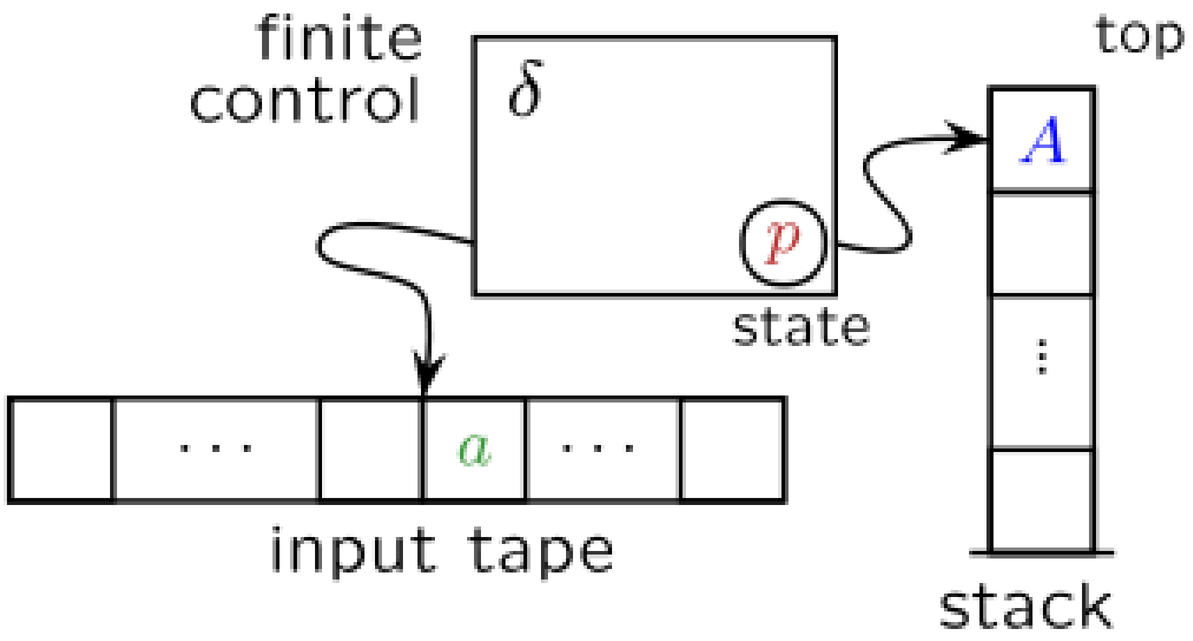
Grammar for $\{ a^n b^n \mid n \geq 1\}$

- G = (V, Σ, R, S) where:
  - V = {T}
  - Σ = {“a”, “b”}
  - R = (T, aTb | ab). When written as a production rule: T -> aTb | ab
  - S = T

CFGs Continued

- The Machines used to recognize CFGs are known as Nondeterministic Pushdown Automata (NPDA)
- This is mathematically formulated as a 7-tuple (Q, Σ, Γ, δ, a, Z, F)
  - Q is a finite set of states
  - Σ is the input alphabet
  - Γ is the stack alphabet
  - a ϵ Q is the start state
  - Z ϵ Γ is the initial stack symbol
  - F c Q is the set of accepting states
  - δ is a function δ: Q x (Σ U {ε}) x Γ -> Q x Γ* (don’t specify this by hand)

PDA Diagram

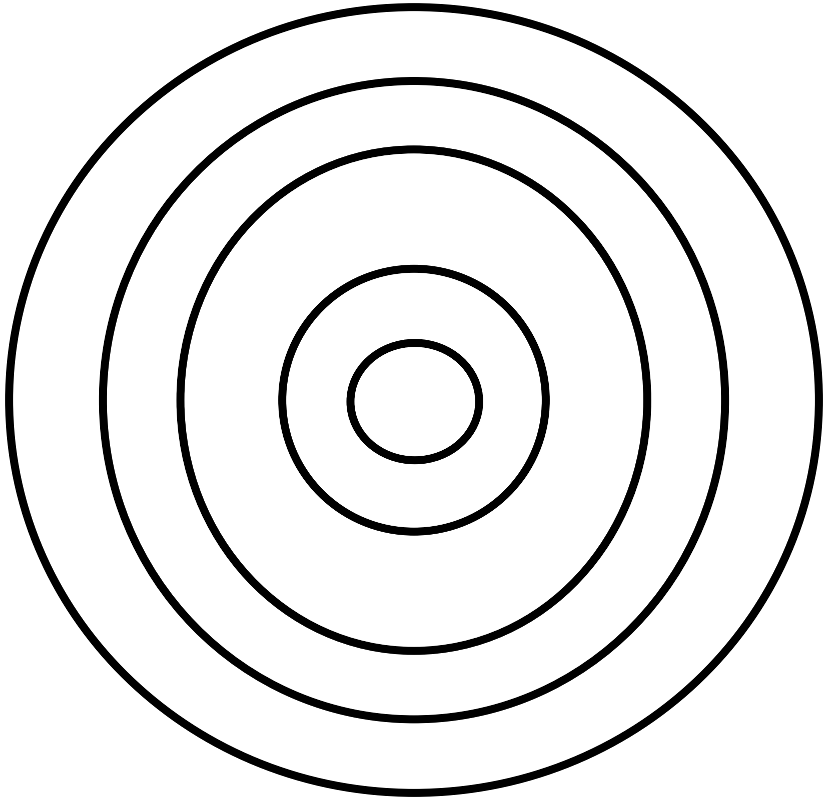


In English: the transition function looks at the current state s, the current input i, and the current stack symbol s and then decide to change from state s to s', pop s and push s'.

PDAs Continued

- You might be wondering if there is a Determinstic Pushdown Automata or DPDA
- There are DPDAs but they do **not** recognize the same class of langauges that NPDAs do. They are a strictly weaker class of machines!
- The above implies that not every CFG you come up with has a deterministic machine that recognizes it. Thus it may have no efficiently computable realization.
- This fact forces us to design context free languages in such a way that there is an efficient way of recognizing them.

Relationships between languages



- Outer Ring: Context Free Languages (NPDA)
- Next: LR(K) grammars (DPDA)
  - Parse bottom up with k steps of look ahead allowed
- Next: Simple LR (simpler parsing tables than above)
- Next: LL(K) grammars
  - Parse top down with k steps of look ahead allowed
- …
- Innermost: Regular Languages

JLex

- Code Demonstration of JLex

Makefile Examples (in C)

Homework 2 Solutions

- Jlex demonstration

HW 3

- Up on the website. Due 10/2 (next Tuesday) at the start of class.
- Generating the language of regular expressions using a CFG!
  - Limited set of operators and terminals (no escape characters etc)
- Question 1:
  - Writing an unambiguous CFG. (Remember the precedence and associativity of regular expressions. (), *, +, |, .
- Question 2:
  - Draw a parse tree for the expression ab+|c*df|ε
  - Use LTR(a) to mean “the LTR token for the letter a” and similarly for the others
  - This should be unambiguous

Project 2 Questions?

- Due 10/2 by 11:59