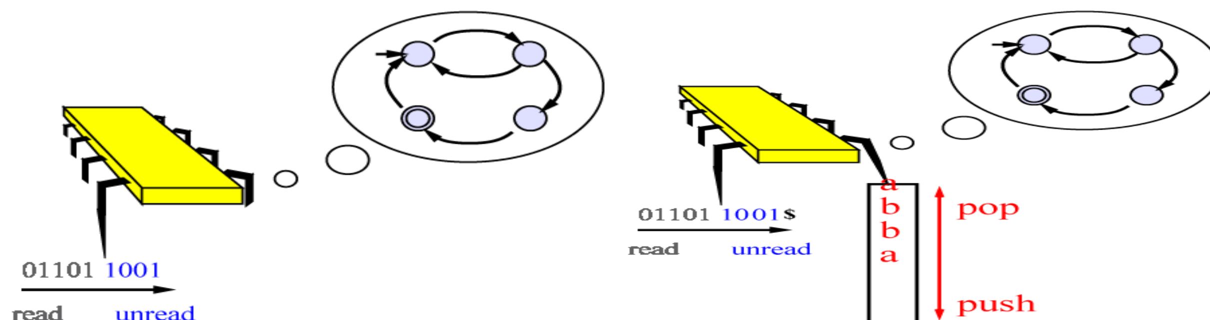
Computational Models - Lecture 5

- Context Free Grammars
- Pumping Lemma for context free languages
- Non context free languages
- Push Down Automata (PDA)
- Closure properties for CFL
- Algorithmic issues for CFL
- Equivalence of CFGs and PDAs



- Sipser's book, 2.1, 2.2 & 2.3

The CFG–PDA Equivalence Theorem

Theorem: A language is context free if and only if some pushdown automata accepts it.

CFL Closure Properties

- Are the context free languages closed under intersection?
- Suggested approach: Can we intersect two context free languages to get $0^n 1^n 2^n$?

CFL Closure Properties

- Are the context free languages closed under intersection?

$$S_1 \to A_1 B_1 \qquad S_2 \to A_2 B_2$$
$$A_1 \to 0A_1 1 | 01 \qquad A_2 \to 0A_2 | \varepsilon$$
$$B_1 \to 2B_1 | \varepsilon \qquad B_2 \to 1B_2 2 | 12$$

$$L_1 = 0^n 1^n 2^* \qquad L_2 = 0^* 1^n 2^n$$

- $L_1 \cap L_2 = 0^n 1^n 2^n$
- L_1 is a context free language, L_2 is a context free language, but $L_1 \cap L_2$ is not a context free language.

CFL Closure Properties

The fact that CFLs are not closed under intersection but are closed under union implies they are not closed under complementation, as $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$.

CFL Closure Properties

Can we give a simple, specific example, where L is not CFL but $\overline{L}$ is?

- Take $L = \{ww \mid w \in \{0,1\}^*\}$.
- For any $y \in \overline{L}$, either
  - y's length is odd.
  - y's length is even, 2ℓ, and there is an $i \geq 1$ such that $y_i \neq y_{\ell+i}$.
- PDA non-deterministically tries to verify one of the options. Employs stack for "matching locations". Accepts only on a successful branch (voluntary home assignment: fill in the details!).

CFL Closure Properties

- Are the context free languages context free languages closed under intersection with a regular language?
- That is, if L_1 is context free languages, and L_2 is regular, must $L_1 \cap L_2$ be context free languages?
- Run PDA L_1 and DFA L_2 "in parallel" (just like the intersection of two regular languages).
  - Formal details omitted (but you should be able to figure them out).
- Why does intersection work here but does not work for two PDAs?
  - Because the DFA does not try to access the stack, while the two PDAs may impose conflicting actions on the stack.

CFL Closure Properties: Example

Is $L = \{(0 \cup 1 \cup 2)^* :$ # of 0's = # of 1's = # of 2's $\}$ context free?

CFL Closure Properties

- $L \triangleq \{(0 \cup 1 \cup 2)^* :$ # 0's = # 1's = # 2's $\}$
- Is L context free?
  - $L \cap 00^*11^*22^* = \{0^n1^n2^n : n > 0\}$ which is not context free.
  - Context free languages intersected with a regular languages are context free.
  - $00^*11^*22^*$ is regular.
  - So L is not a context free language!

- This could also be established using pumping lemma, but proof above is much more elegant.

More CFL Closure Properties

- Assignments
- Homomorphism
- Inverse Homomorphism: $h^{-1}(L) = \{x | h(x) \in L\}$
  - Idea: read $a \in \Sigma$ and simulate $h(a)$
  - Problem: The PDA might make an unbounded number of moves when simulating $h(a)$
  - Solution: encode in the state the value of $h(a)$ and simulate it like a buffer

More CFL Closure Properties: Inverse Homomorphism

Theorem: The CFL are closed under inverse homomorphism.

- Define $M' = (Q', \Sigma, \Gamma, \delta', [q_0, \epsilon], F \times \{\epsilon\})$
- States $Q' = \{[q, x] \mid q \in Q,$ x prefix of some $h(a)\}$
- Transition function δ'
  - If $(p, Z) \in \delta(q, \epsilon, Y)$ then $([p, x], Z) \in \delta'([q, x], \epsilon, Y)$
    ϵ-moves of original M
  - If $(p, Z) \in \delta(q, a, Y)$ then $([p, x], Z) \in \delta'([q, ax], \epsilon, Y)$
    simulate original M from buffer
  - $([p, h(a)], Y) \in \delta'([q, \epsilon], a, Y)$ load buffer by $h(a)$

Algorithmic Questions Regarding CFGs

Given a CFG, G, and a string w, does G generate w?

Initial Idea: Design an algorithm that tries all derivations.

Problem: If G does not generate w, we'll never stop.

Chomsky Normal Form

A simplified, canonical form of context free grammers. Elegant by itself, useful (but not crucial) in proving equivalence theorem. Can also be used to slightly simplify proof of pumping lemma.
Every rule has the form

$$
\begin{aligned}
A &\rightarrow BC \\
A &\rightarrow a \\
S &\rightarrow \varepsilon
\end{aligned}
$$

where S is the start symbol, A, B and C are any variable, except B and C not the start symbol, and A can be the start symbol.

CFN: Theorem

Theorem: Any context-free language is generated by a context-free grammar in Chomsky normal form.
Basic idea:

- Add new start symbol S_0.
- Eliminate all ε rules of the form $A \to \varepsilon$.
- Eliminate all "unit" rules of the form $A \to B$.
- Patch up rules so that grammar generates the same language.
- Convert remaining "long rules" to proper form.

Algorithmic Questions for CFGs (2)

Lemma: If G is in Chomsky normal form, $|w| = n$, and w is generated by G, then w has a derivation of length $2n - 1$ or less.

Algorithm's idea:

- First, convert G to Chomsky normal form.
- Now need only consider a finite number of derivations – those of length $2n - 1$ or less.

Algorithmic Questions for CFGs: membership

- Build a function $derive(A, x)$ that returns TRUE if $A \stackrel{*}{\Rightarrow} x$
- PROCEDURE $derive(A, x)$.
  - If $|x| = 1$ then if $A \to x \in R$ return TRUE, otherwise return FALSE.
  - For each $A \to BC$ and each partition $x = x_1 x_2$,
    - Call $derive(B, x_1)$ and $derive(C, x_2)$.
    - If both return true, exit and return TRUE.
  - Return FALSE.
- Test membership by calling $derive(S, w)$.
- Why did we need CNF ?!

Algorithmic Questions for CFGs: membership

- What is the time complexity of $derive(S, w)$?
- Simple Recursive implementation analysis:
  - each time test $|R|$ rules and n partitions.
  - $T(n) \leq |R|n \cdot 2T(n-1)$
  - $T(n) = O((|R|n)^n)$.
- Still exponential

Dynamic Programming for CFG membership

- What is the time complexity of $derive(S, w)$?
- Better implementation: keep in memory the results of $derive(A, x)$.
  - Number of different inputs: $|V| \cdot n^2$.
  - Only $|V| \cdot n^2$ calls, each takes $O(n|R|)$.
  - $T(n) = O(|R|n^3|V|)$.
- Polynomial rime!

Emptiness of CFGs

Given a CFG, G, is $L(G) = \emptyset$?

In other words, is there any string w, such that G generate w?

Theorem: There is an algorithm that solves this problem (and always halts).

Possible approaches for a proof:

Bad Idea: We know how to test whether $w \in L(G)$ for any string w, so just try it for each w. (criticize this...)

Better Idea: Can the start variable generate a string of terminals?

Even Better Idea: Can a particular variable generate a string of terminals?

Removing redundant variables and terminals of a CFG

1. Mark all terminal symbols in G.
2. Repeat until no new variable become marked:
   Mark any A where $A \to U_1 U_2 \ldots U_k$ and all U_i have already been marked.
4. Remove all unmarked variables, and any rule they appear in.
5. If S is removed, then $L(G) = \emptyset$.
6. Remove any variable A not reachable from S.
7. Remove any terminal which does not appear in some rule.

CFG Emptiness (2)

Algorithm: On input G (a CFG),

1. Remove redundant variables and terminals, get G'.
2. $L(G') = \emptyset$ iff S is removed

Finiteness of CFGs

Given a CFG, G, is $|L(G)|$ finite?

1. Remove redundant variables and terminals.
2. Create a graph where nodes are variables and directed edges are derivations.
3. $L(G)$ is infinite iff the graph has a cycle.

CFGs "Fullness"

Given a CFG, G, is $L(G) = \Sigma^*$?

We just saw an algorithm to determine, given a CFG, G, if $L(G) = \emptyset$

$L(G) = \Sigma^*$ iff $\overline{L(G)} = \emptyset$. Why not modify the algorithm so it determines emptiness of the complement?

Unfortunately, CFGs are not closed under complement.

Fact: There is no algorithm to solve this problem.

We are not prepared to prove this remarkable fact (yet).

CFGs Inherent Ambiguity

Given a CFG, G, is $L(G)$ inherently ambiguous?

This means that for any CFG that generates $L(G)$, there is a word in the language with two different parse trees.

Fact: There is no algorithm to solve this problem.

We will not prove this fact, yet you want to know it to put things in context.

When Are Two CFGs equivalent?

Given two CFGs, G, H, is $L(G) = L(H)$?

Hey, we did this already for equivalence of DFAs!

We constructed C from A and B:

$$L(C) = \left(L(A) \cap \overline{L(B)}\right) \cup \left(\overline{L(A)} \cap L(B)\right).$$

and tested whether $L(C)$ is empty.

When Are Two CFGs equivalent?

This approach was fine for DFAs, but not for CFLs!

The class of context-free languages is not closed under complementation or intersection.

Fact: There is no algorithm to solve this problem.

We are not prepared to prove this remarkable fact (yet).

The CFG–PDA Equivalence Theorem

Theorem: A language is context free if and only if some pushdown automata accepts it.

This time (unlike the regular expression vs. regular languages theorem), the proofs of both the "if" part and the "only if" part are non trivial.

Proof sketch follows.

If Part

Theorem: If a language is context free, then some pushdown automaton accepts it.

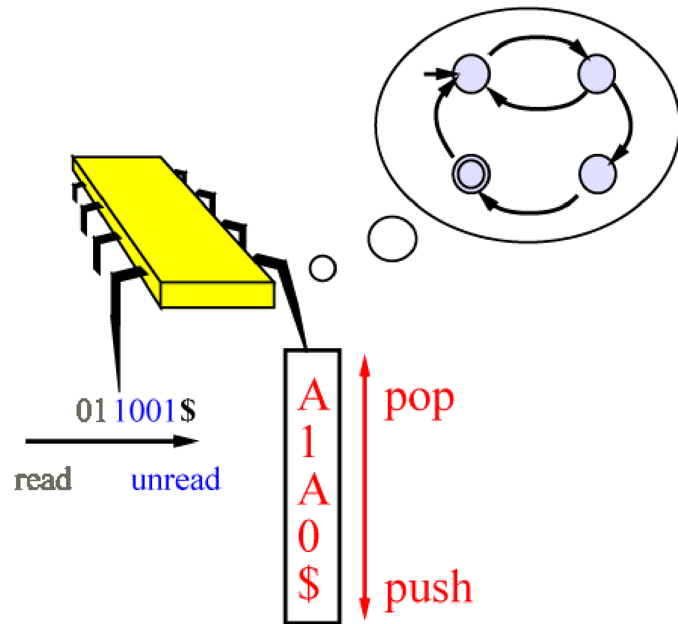
- Let A be a context-free language.
- By definition, A has a context-free grammar G generating it.
- On input w, the PDA P should figure out if there is a derivation of w using G.

Question: How does P figure out which substitution to make?

Answer: It guesses.

CFL Implies PDA (cont.)

Where do we keep the intermediate string ($01A1A0$ in the example below))?



- can't put it all on the stack
- partial strings starting by a variable are kept on stack

CFL Implies PDA

Informally, on input string $w \in \Sigma^*$:

- P pushes start variable S on stack
- keeps making substitutions
- when popping a terminal, P checks equality with current input string
- rejects if not equal
- when popping a variable, P pushes to top of stack a right hand side of some rule corresponding to variable (zero, one, or more symbols).
- if stack is empty, allow to accept and terminate.

CFL Implies PDA (cont.)

Informal description:

- push $S\$$ on stack
- if top of stack is variable A, non-deterministically select rule $A \rightarrow \alpha$ and substitute.
- if top of stack is terminal a, read next input and compare. If they differ, reject.
- if top of stack is $\$$, allow to enter accept state.

CFL Implies PDA (cont.)

Need shorthand to push strings of finite length onto stack. For example, suppose

$$A \to BC$$

is a derivation of the CFG.
Then we add a "shorthand state", q_e, and the two transitions

$$(q_e, C) \in \delta(q_\ell, A, \varepsilon), \quad \delta(q_e, \varepsilon, \varepsilon) = \{(q_\ell, B)\}$$

Notice that the second transition is deterministic (the first one may or may not be). Also notice order: Push C first, then B.
These intermediate states are different for different derivations.

CFL Implies PDA (cont.)

States of P are

- start state q_s
- accept state q_a
- loop state q_ℓ
- q_e states, needed for shorthand of right hand sides of rules

Transition Function

Initialize stack

$$\delta(q_s, \varepsilon, \varepsilon) = \{q_\ell, S\$\}$$

Top of stack is variable (shorthand for multiple transitions)

$$\delta(q_\ell, \varepsilon, A) = \{(q_\ell, w) | \text{ where} A \to w \text{ is a rule } \}$$

Top of stack is terminal

$$\delta(q_\ell, a, a) = \{(q_\ell, \varepsilon)\}$$

End of Stack and End of Input

$$\delta(q_\ell, \varepsilon, \$) = \{(q_a, \varepsilon)\}$$

Example

$$
\begin{aligned}
S &\rightarrow AT|\varepsilon \\
A &\rightarrow AB|AA|a \\
B &\rightarrow b \\
T &\rightarrow TT|t
\end{aligned}
$$

Transition rules for PDA: On black/white board.

Only If Part

Theorem: If a PDA accepts a language, L, then L is context-free.

- For each pair of states p and q in P, we will have a variable A_{pq} in the grammar G.
- This variable, A_{pq}, generates all strings that take P from p with an empty stack to q with empty stack.
- Same string also takes p with any stack to q with same stack!
- Start variable is A_{q_0,q_a} (assuming a single accept state q_a).

PDA Implies CFL

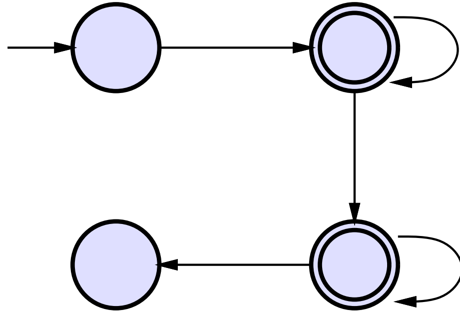
To make things easier, we slightly modify P

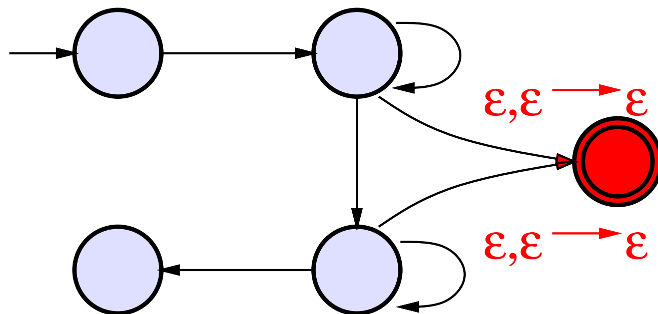
- Has single accept state q_a.
- It empties stack before accepting.
- Each transition either pushes a symbol on stack, or pops a symbol from stack, but not both.

PDA Implies CFL (2)

Modify P to make things easier

- single accept state q_a



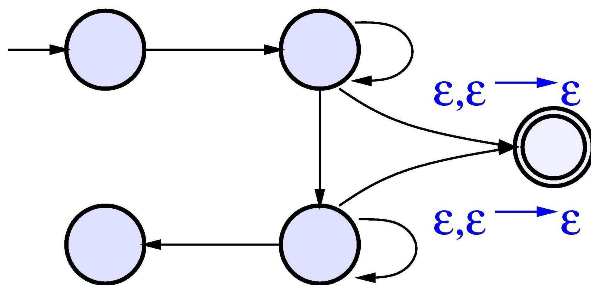


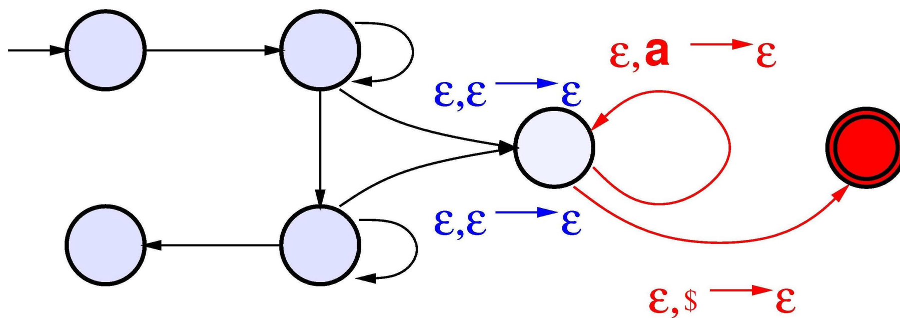
- empties stack before accepting
- each transition pushes or pops, but not both.

PDA Implies CFL (3)

Modify P to make things easier

- single accept state q_a $\surd$
- empties stack before accepting

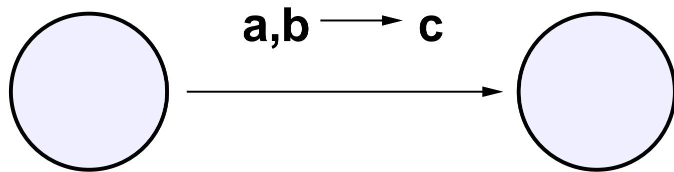


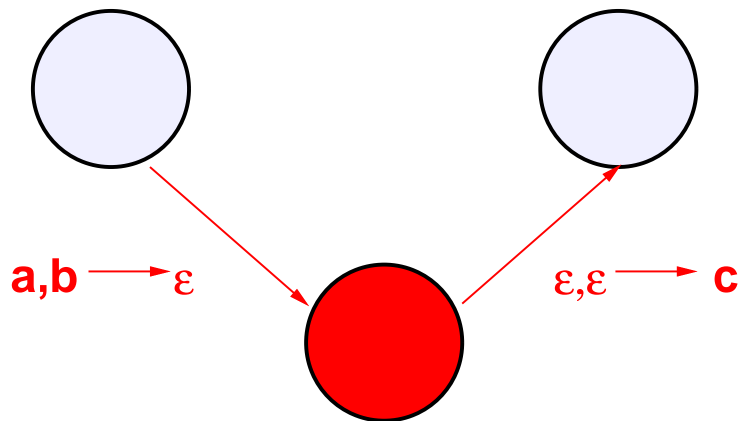


PDA Implies CFL (4)

Modify P to make things easier

- single accept state q_a $\surd$
- empties stack before accepting $\surd$
- transition either pushes or pops, but not both





Proof Idea

Suppose string x takes P from p with empty stack to q with empty stack.

First move that touches the stack must be a push, last must be a pop.

In between, two possibilities:

- Stack is empty only at start and finish, but not in middle.
- Stack was also empty at some point in between.

Proof Idea (2)

Suppose string x takes P from p with empty stack to q with empty stack.

First move that touches the stack must be a push, last must be a pop.

In between, two possibilities:

- Stack is empty only at start and finish, but not in middle.

  Simulate by: $A_{pq} \to aA_{rs}b$, where a, b are first and last symbols in x (shorter x will be taken care of too), r follows p, and s precedes q.

- Stack was also empty at some point in between. Simulate by: $A_{pq} \to A_{pr}A_{rq}$, r is intermediate state where P has empty stack.

Details of Simulating Grammar

Given PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, \{q_a\})$, construct grammar G. Variables are $\{A_{pq} \mid p, q \in Q\}$.

Start variable is $A_{q_0 q_a}$.

Rules:

- For $p, q, r, s \in Q$, $t \in \Gamma$, and $a, b \in \Sigma$, if $(r, t) \in \delta(p, a, \varepsilon)$ and $(q, \varepsilon) \in \delta(s, b, t)$, add rule $A_{pq} \to a A_{rs} b$.
- for every $p, q, r \in Q$, add rule $A_{pq} \to A_{pr} A_{rq}$.
- for each $p \in Q$, add rule $A_{pp} \to \varepsilon$.

Overall Structure

Should now prove

Claim: A_{pq} generates x if and only if x brings P from p with empty stack to q with empty stack.

Only If Part

Theorem: If a PDA accepts a language, L, then L is context-free.

Proof After constructing the grammar G , should prove it generates exactly the same language accepted by the PDA. This is done by induction on the length of any computation of P on any input string x.

The induction argument is a bit lengthy and tedious, and we'll skip it.

Diehards are welcome to consult pp. 106–114 in Sipser's book.

A Short Summary

- Regular Languages $\equiv$ Finite Automata.
- Context Free Languages $\equiv$ Push Down Automata.
- Closure properties of regular languages and of CFLs.
- Most algorithmic problems for finite automata are solvable.
- Some algorithmic problems for finite automata are not solvable.
- Pumping lemmata for both classes of languages.
- There are additional languages out there.

The View Over The Horizon