Computational Models– Lecture 5

- One More PDAs Example
- Equivalence of PDAs and CFLs
- Nondeterminism adds power to PDAs (not in book)
- Closure Properties of CFLs
- Algorithmic Aspects of PDAs and CFLs
- DFAs and PDAs: Perspectives

- Sipser's book, 2.2 & 2.3

- Mid-term exam on Friday, April 13.
- Material is first five lectures (*i.e.* up to and including today, and Chomsky normal form from lecture 4).
- Closed books (no auxiliary material).
- 10 multiple choice (“closed, American”) questions.
- Duration 1:40 hrs.

Another PDA Example

A palindrome is a string w satisfying $w = w^{\mathcal{R}}$.

- "Madam I'm Adam"
- "Dennis and Edna sinned"
- "Red rum, sir, is murder"
- "Able was I ere I saw Elba"
- "In girum imus nocte et consumimur igni" (Latin: "we go into the circle by night, we are consumed by fire".)
- "$\nu\iota\psi o\nu\ \alpha\nu o\mu\eta\mu\alpha\tau\alpha\ \mu\eta\ \mu o\nu\alpha\nu\ o\psi\iota\nu$"
- Palindromes also appear in nature. For example as DNA restriction sites – short genomic strings over $\{A, C, T, G\}$, being cut by (naturally occurring) restriction enzymes.

Another PDA Example

- On input x, the PDA start pushing x into stack.
- At some point, PDA guesses that the mid point of x was reached.
- Pops and compares to input, letter by letter.
- This PDA accepts palindromes of *even length* over the alphabet.
- Again, non-determinism seems necessary.

Non-Deterministm Adds Power: Proof

Theorem: Let M be a PDA that accepts

$$L = \{x^n y^n | n \geq 0\} \cup \left\{x^n y^{2n} \mid n \geq 0\right\} \ .$$

Then M is non-deterministic.

Proof: [a] Suppose, by way of contradiction, that M is deterministic.

- Create two copies of this PDA, denoted M_1 and M_2.
- Two states in M_1 and M_2 are called "cousins" if they are copies of the same state in the original PDA.

[a] (prf modified from www.cs.may.ie/∼jpower/Courses/parsing/node38.html)

Non-Deterministm Adds Power (cont.)

- We now modify these PDA copies to make them into one PDA, M_0, over the alphabet $\{x, y, z\}$.
- States of the new M_0 are those of M_1 union M_2.
- Start state of the new M_0 is the start state of M_1.
- The accepting states of the new M_0 are the accepting states of M_2.

Non-Determinism Adds Power (cont.)

- Modifications:
  - Erase all x transitions of M_2.
  - Replace every existing y transition of M_2 by a new z transition.
  - At this point M_2 got only z transition (so x and y inputs lead immediately to rejection).
  - Erase all x transitions out of accept states of M_1.

Non-Deterministm Adds Power (cont.)

- The surgery is almost done, but if we don't connect the two halves of its brain, the patient will not function coherently.
- Replace every existing y transition leading out of accept states of M_1 by a new z transition, and redirect it to its "cousin" in M_2.
- Surgery over. Patient (a deterministic PDA) still alive. Let us now diagnose what, if anything, it can do.

Non-Determinism Adds Power (cont.)

- What language M_0 recognizes?
- Certainly if M_0 accepts a string, it must be of the form $(x \cup y)^* z^*$.
- But surely not all strings of that form are accepted by M_0.
- For example, the $(x \cup y)^*$ prefix must be accepted by the original M.
- Otherwise there will be no switch to M_2, and no acceptance by M_0. (think why is $L(M_0) \neq \emptyset$? Would this also be true for non deterministic M?)
- So the prefix of an accepted string is either of the form $x^n y^n$ or $x^n y^{2n}$.
- And the whole string is of the form $x^n y^n z^i$ or $x^n y^{2n} z^j$.

Non-Deterministm Adds Power (cont.)

- What can we say about the z^i part? First, i must be greater than 0 for a transition to take place.
- By construction, M_2 on z^i imitates the actions of M on y^i from the same starting point.
- This means that if M_0 accepts $x^n y^n z^i$ then M accepts $x^n y^{n+i}$.
- Which is possible if either $i = n$, so M_0 accepts $x^n y^n z^n$, $n > 0$,
- or M_0 accepts $x^n y^{2n} z^j$, so M accepts $x^n y^{2n+j}$.
- But L contains no strings of this last form!

Conclusion of Proof

- We just showed that the PDA M_0 accepts the language $\{x^n y^n z^n | n \geq 1\}$. Contradiction. ♣
- Contradiction? What contradiction? What the $%&# does this contradict?
- It contradics the fact that by the so called $uvxyz$ pumping lemma, the language $\{x^n y^n z^n | n \geq 1\}$ is not context free, so is not accepted by a PDA.
- So our initial supposition that the language $\{x^n y^n\} \cup \{x^n y^{2n}\}$ is accepted by a deterministic PDA, does not hold.
- While thinking about the proof, where would it fail if the original M were non-deterministic?

PDA Languages vs. CFLs

The set of Push-Down Automata Languages, L_{PDA}, is the collection of all languages that are accepted by some PDA:

$$L_{PDA} = \{L : \exists\, PDA\, M \wedge L(M) = L\}\ .$$

- Natural questions:

- $L_{CFG} \subseteq L_{PDA}$?
- $L_{PDA} \subseteq L_{CFG}$?

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.

If Part

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

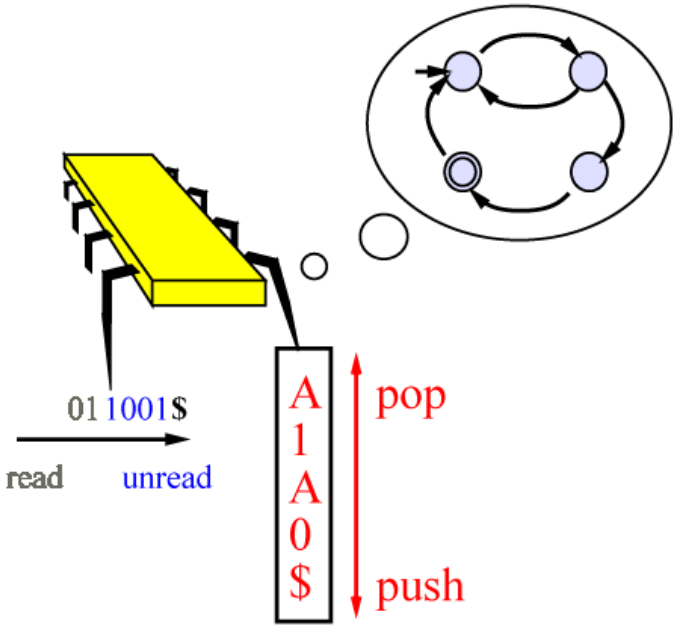
- Let A be a context-free language.
- By definition, A has a context-free grammar G generating it.
- On input w, the PDA P should figure out if there is a derivation of w using G.

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

Answer: It guesses.

CFL Implies PDA (cont.)

Where do we keep the intermediate string?



intermediate string: 01A1A0

- can’t put it all on the stack
- only strings whose first letter is a variable are kept on stack

CFL Implies PDA

Will be more convenient to use grammar in Chomsky normal form, due to compact derivation rules.

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 two symbols).
- if EOI reached when stack is empty, accept.

CFL Implies PDA (cont.)

Informal description:

- push $S\$$ on stack
- if top of stack is variable A, non-deterministically select rule and substitute.
- if top of stack is terminal a, read next input and compare. If they differ, reject.
- if top of stack and input symbol are both $\$$, enter accept state. (Namely accepts only if input has all been read and stack is empty!).

CFL Implies PDA (cont.)

Need shorthand to push strings of length 2 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 two 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

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

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

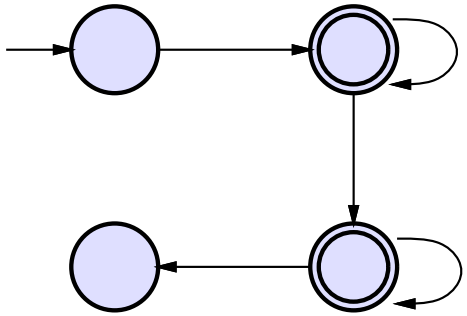
To make things easier, we slightly modify P

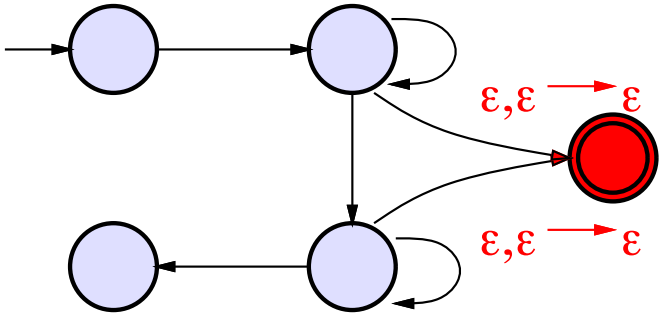
- Has single accept state q_a.
- It empties stack before accepting.
- Each transition either pushes a symbol on stack, or pops a symbol from stack, but not both.

PDA Implies CFL (2)

Modify P to make things easier

- single accept state q_a



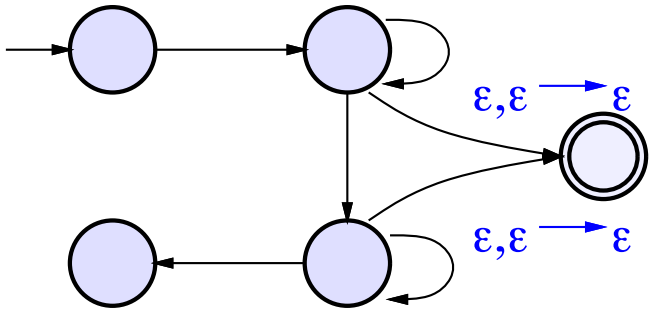


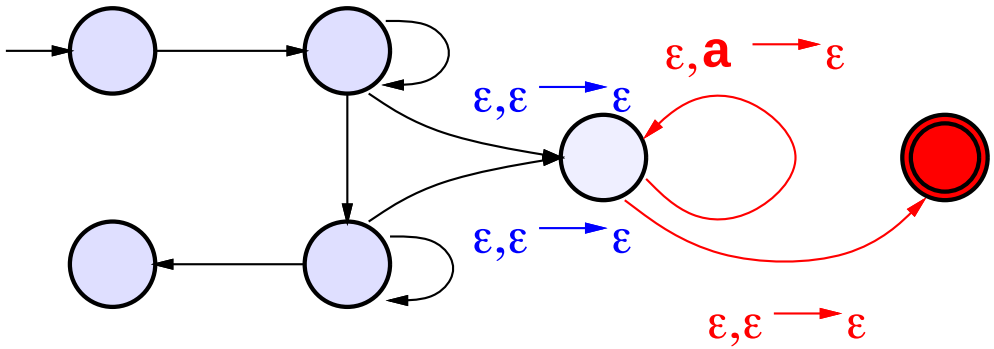
- empties stack before accepting
- each transition pushes or pops, but not both.

PDA Implies CFL (3)

Modify P to make things easier

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

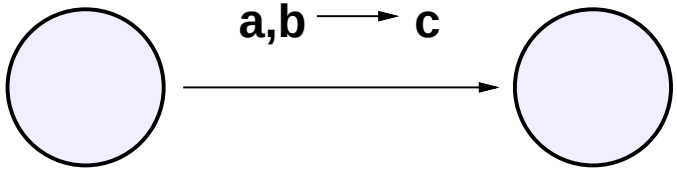


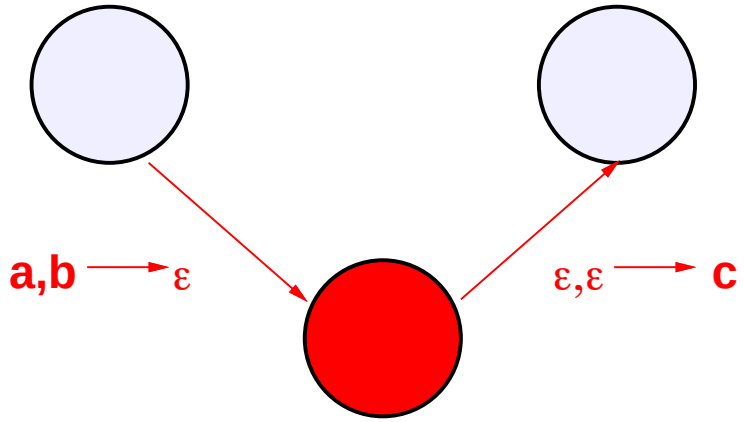


PDA Implies CFL (4)

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: $A_{pq} \rightarrow aA_{rs}b$, 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: $A_{pq} \rightarrow A_{pr}A_{rq}$, 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 $A_{pq} \rightarrow a A_{rs} b$.
- for every $p, q, r \in Q$, add rule $A_{pq} \rightarrow A_{pr} A_{rq}$.
- for each $p \in Q$, add rule $A_{pp} \rightarrow \varepsilon$.

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, and/or slides from fall 2003/4.

CFL Closure Properties

- We saw that Context-Free Languages are closed under union, concatenation, and star?
- It is time we resolve closure with respect to complementation and intersection.

CFL Closure Properties

- Are the context free languages context free languages closed under intersection?
- Suggested approach: Can we intersect two context free languages langauges 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 languages

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).

CFL Closure Properties: Example

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

CFL Closure Properties

- $L \triangleq \{(0 \cup 1 \cup 2)^* : \#\ 0\text{'s} = \#\ 1\text{'s} = \#\ 2\text{'s}\ \}$
- Is L context free?
  - $L \cap 00^*11^*22^* = \{0^n 1^n 2^n : n > 0\}$ 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!

Algorithmic Questions Regarding DFAs

Given a regular expression, R, find the smallest DFA (minimum number of states) that accepts $L(R)$.

- Initial Idea: Use the algorithm describe in class to transform R into an NFA. Then transform this NFA into a DFA, M.
- That's very nice, but how do we know M is minimal?
- It need not be!

Algorithmic Questions for DFAs (2)

Given a regular expression, R, find the smallest DFA that accepts $L(R)$ (minimum number of states).

- We can enumerate all DFAs that are strictly smaller than M.
- For each such M_i, test if $L(M_i) = L(M)$ (we saw an algorithm for this).
- Take the smallest such M_i.
- Algorithm is very inefficient. If smallest M has n states, algorithm will take time that is exponential in n.
- More efficient algorithm is known, using the Myhill-Nerode theorem.

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.

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.

We won't prove this (go ahead — try it at home!).

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 (3)

Theorem: There is an algorithm (that halts on every inputs) $\mathcal{A}$, that on inputs G and w, decides if G generates w.

On input $\langle G, w \rangle$, where G is a grammar and w a string,

1. Convert G to Chomsky normal form.
2. List all derivations with $2n - 1$ steps, were $n = |w|$.
3. If any generates w, accept, otherwise reject. ♣

Algorithmic Questions for CFGs (4)

Theorem: There is an algorithm (that halts on every inputs) $\mathcal{A}$, that on inputs G and w, decides if G generates w.

Remarks:

- Related to problem of compiling prog. languages.
- Would you want to use this algorithm at work?
- Every theorem about CFLs is also about PDAs.

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?

CFG Emptiness (2)

Algorithm: On input G (a CFG),

1. Mark all terminal symbols in G.
2. Repeat until no new variables become marked.
3. Mark any A where $A \rightarrow U_1 U_2 \ldots U_k$ and all U_i have already been marked.
4. If start symbol marked, accept, else reject. ♣

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).

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.

Stop! Danger! Abyss ahead!

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.

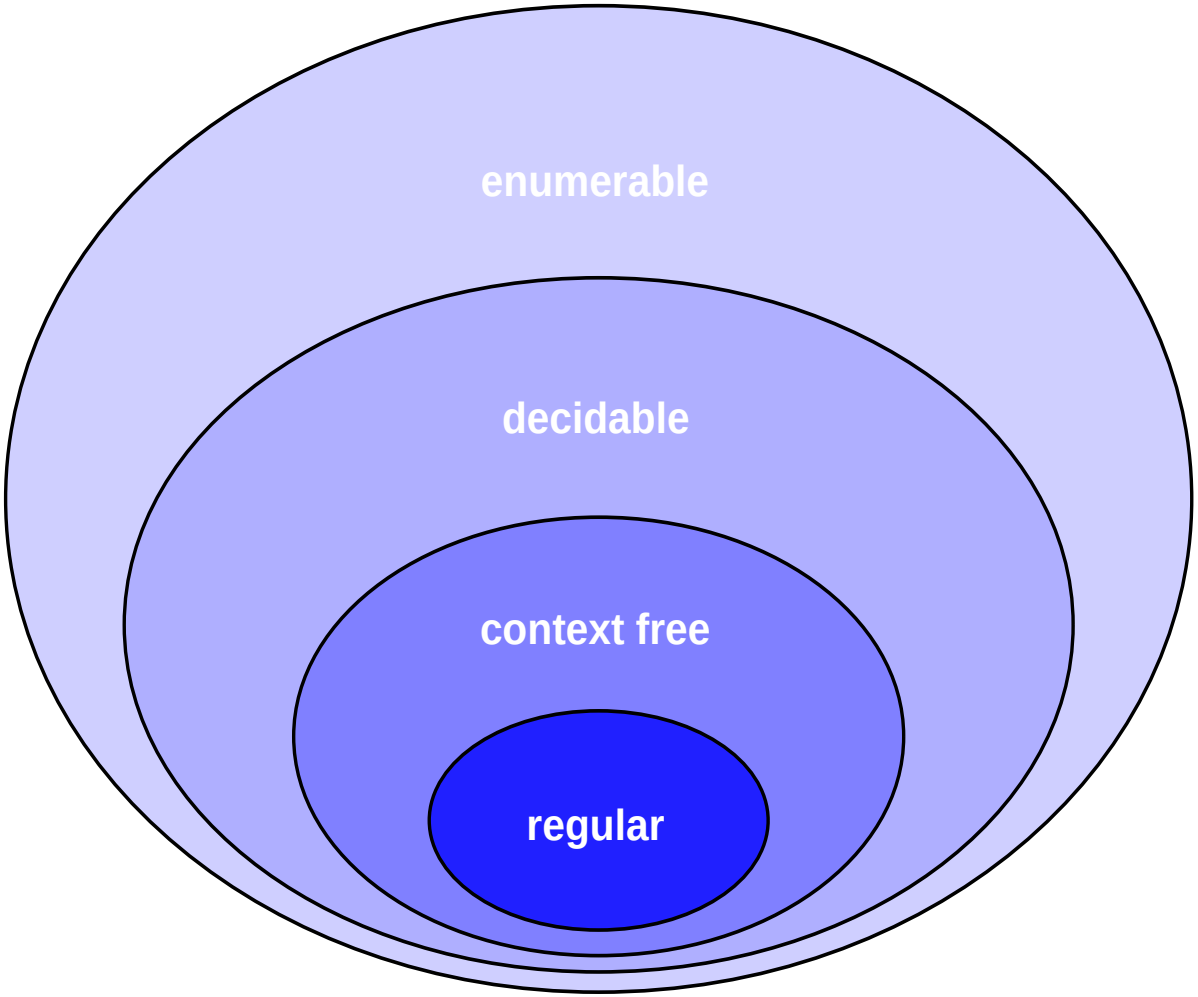
Fact: There is no algorithm to solve this problem.

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

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



Happy passover (& kosher for legumes eaters)!