REGULAR LANGUAGES

BACKGROUND:

\( \Sigma \) a finite non-empty set called an alphabet.

Typically \( \Sigma = \{0, 1\} \).

A **string** \( w \) is a finite sequence of symbols (aka, alphabets) in \( \Sigma \).

\[ \varepsilon : = \text{zero symbols, } |w| = 0 \text{ of symbols of } w \text{ (rather positions)} \]

\[ \Sigma^k : = \{ w : |w| = k \text{ and } w \text{ a string over } \Sigma \} \]

\[ \Sigma^* : = \cup_{k \geq 0} \Sigma^k, \quad \Sigma^+ : = \cup_{k \geq 1} \Sigma^k \]

A **language** \( L \) over \( \Sigma \) is a subset of \( \Sigma^* \).

The concatenation of \( x = x_1 \ldots x_m \) and \( y = y_1 \ldots y_m \) is \( xy = x_1 \ldots x_m y_1 \ldots y_m \) . Note: \( xy \neq yx \).

(Find all \( x, y \in \Sigma^* \text{ with this property} \).

Note: A language is called a problem when strings are given some interpretation. (E.g., view \( w \in \Sigma^* \) as the binary rep. of an integer).

An **automaton** is an abstract model of computation:

![Diagram](attachment:diagram.png)

**Terminology:**
- **Transition diagram**
- **Start arrow**
- **Accept states**
It reads an input $w = w_1 \ldots w_n$ from left to right. It follows the arcs according to $w_i$ starting at the start state. It accepts iff it ends up in an accepting state after reading the whole input. It rejects otherwise.

Formally, a **deterministic finite automaton (DFA)** is a 5-tuple $D = (Q, \Sigma, \delta, q_0, F)$ where:

1. $Q$ is a finite set (states)
2. $\Sigma$ an alphabet
3. $\delta : Q \times \Sigma \rightarrow Q$ a transition function
4. $q_0 \in Q$ the start state
5. $F \subseteq Q$ the accept state.

We define the extended transition function $\hat{\delta}$ of $D$ as follows:

$$\hat{\delta}(q, \varepsilon) = q; \quad \hat{\delta}(q, x) = \delta(\hat{\delta}(q, x), x)$$

We define the language $L$ of $D$ as:

$$L(D) = \{ w \in \Sigma^* : \hat{\delta}(q_0, w) \in F \}$$

We say $D$ recognizes a language $L$ if $L = L(D)$.

We say $L \subseteq \Sigma^*$ is regular if $L = L(D)$ for some $D$.

**Example:** $D = (Q = \{q_0, q_1, q_2, q_3\}, \Sigma = \{0, 1\}, S, q_0, F = \{q_1\})$

$S$:

$$\begin{align*}
q_0 &\rightarrow 0, 1, q_1 \\
q_1 &\rightarrow 0, 1, q_2 \\
q_2 &\rightarrow 0, 1, q_3 \\
q_3 &\rightarrow 0, 1, q_0
\end{align*}$$

**Example:** $L = \{ w \in \{0, 1\}^* : w \text{ is seen as a binary expansion} \}$ is regular when $w$ is seen as a binary expansion. If $w$ starting with $0$ is invalid unless $w = 0$.

**Note:** $2x = 2x + a$

So if we are at state $i$ after reading $x$, we go to state $2i + a$ after reading $a$.

We ignore all leading $0$'s unless there is exactly one.

**Questions:** How can we decide, given regular $L_1, L_2$, if any of the following are regular:

$L_1 L_2$; $L_1 L_2$; $L_1 = \Sigma \setminus L_1$; ...

How can we show if a language is not regular?

We will develop appropriate machinery to answer such questions.

**Nondeterministic finite automata**

Nondeterminism: a general useful concept in studying computation.

Gives the power to be in several states at once or guess something about the input.

**Example:**

```
\begin{align*}
0, 1 &\rightarrow 0 \\
0 &\rightarrow 1
\end{align*}
```

What does this do? Accept all strings that end in $01$ at $q_0$ guess if 2 more symbols to go.
Let's look at its execution on \( w = 00101 \).

Formally, an NFA is a 5-tuple \( N = (Q, \Sigma, \delta, q_0, F) \) such that:
1. \( Q \), (2) \( \Sigma \), (4) \( q_0 \), (5) \( F \) as before.
2. \( \delta \colon Q \times \Sigma \to \mathcal{P}(Q) \) is the transition relation.

The extended trans. func. is:

\[
\hat{\delta}(q, \varepsilon) = \{ q \} \quad \hat{\delta}(q, x) = \bigcup_{p \in \delta(q, x)} \delta(p, x)
\]

Recall: \( \delta_N \):

\[
\begin{array}{c|c|c|c|c|c}
0 & 1 & 0 & 1 & 0 & 1 \\
q_0 & q_1 & q_0 & q_1 & q_0 & q_1 \\
q_1 & q_0 & q_0 & q_0 & q_0 & q_0 \\
q_2 & q_1 & q_1 & q_1 & q_1 & q_1 \\
\end{array}
\]

Example: \( N = (Q_N, \Sigma, \delta_N, q_0, F_N) \) with \( \delta_N \) as shown.

Equivalence of DFAs and NFAs:

Is there any \( L \) such that \( L = L(N) \) for some NFA \( N \) but \( L \neq L(D) \) for all DFA \( D \)?

No: we show how to convert any NFA \( N \) to an equivalent DFA \( D \) such that \( L(N) = L(D) \).

The subset construction:

Let \( N = (Q_N, \Sigma, \delta_N, q_0, F_N) \) consider:

\( \hat{\delta}_D = \delta(D) \) as follows.

\[ \delta(D) (q, \varepsilon) = \{ q \} \quad \delta(D) (q, x) = \bigcup_{p \in \delta(D)(q, x)} \delta(D)(p, x) \]

Example:

\[ \delta(D) (q, a) = U_{p \in \delta(N)(q, a)} \delta(N)(p, a) \]

All possible states we can go to.

Recall \( \delta_N \):

\[
\begin{array}{c|c|c|c|c|c}
0 & 1 & 0 & 1 & 0 & 1 \\
q_0 & q_0 & q_0 & q_0 & q_0 & q_0 \\
q_1 & q_1 & q_1 & q_1 & q_1 & q_1 \\
q_2 & q_2 & q_2 & q_2 & q_2 & q_2 \\
\end{array}
\]

(UNREACHABLE STATES NOT SHOWN)

Theorem: Given \( N \) and \( D \) as above, we have that \( L(D) = L(N) \).

Proof: We need to show: \( \forall w \in \Sigma^* \quad \delta_D (q_0, w) = \delta_N (q_0, w) \).

We do so by induction on \( w \).

Basis: \( |w| = 0 \), then \( \delta_D (q_0, \varepsilon) = \{ q_0 \} \) by defn. of \( \delta_D \).

Induction: let \( w = xa \). \( |w| = n+1 \).

\( \delta_D (q_0, x) = \delta_D (q_0, x) \) by defn. of \( \delta_N \).

TH: \( \delta_D (q_0, x) = \delta_N (q_0, x) \).
NOW: \[ \hat{\delta}_D (q_0, a) = \hat{\delta}_D (\hat{\delta}_D (q_0, a), a) \]

\[ \begin{align*}
\hat{\delta}_D (q_0, a) &= \hat{\delta}_D (\hat{\delta}_N (q_0, a), a) \\
\hat{\delta}_N (q_0, a) &= \bigcup_{p \in \hat{\delta}_N (q_0)} \hat{\delta}_N (p, a) \\
\end{align*} \]

\[ \text{COROLLARY:} \quad \text{THEOREM L.2 \#1 FOR AN \textit{A DFA D} \quad \Leftrightarrow \quad \text{THEOREM L.2 \#1 FOR AN \textit{A NFA N}.} \]

\[ \text{PROOF:} \quad (\Rightarrow) \text{WE JUST PROVED} \\
(\Leftarrow) \text{EASY:} \quad \hat{\delta}_N (q_0, a) = \{ \delta_D (q_0, a) \}. \]

\[ \text{NOTE: WHEN CONVERTING AN \textit{A NFA TO A DFA THE NUMBER OF STATES CAN} BLOW UP EXPONENTIALLY.} \]

\[ \text{THIS IS INHERENT.} \]

\[ \text{LET} \quad L[w] : \text{\textit{n-th symbol of w from end is i}} \]

\[ \text{IF} \quad L[w] : \text{\textit{n-th symbol of w from end is 7}} \]

NFA: \[ \text{SUPPOSE WE BUILD A DFA \textit{WITH < 2^n STATES.}} \]

\[ \text{THEN} \quad q_i \ldots q_n \neq b_i \ldots b_m \ 	ext{ST. D IS IN STATE q AFTER READING n SYMBOLS. (PAPER HOLE)} \]

\[ \text{LET q_i \neq b_i: TAKE FIRST SUCH i.} \]

\[ \text{CASE 1:} \quad i = 1. \ \text{WITHOUT LOSS OF GENERALITY (WLOG)} \]

\[ a_1 = 1 \text{ AND } b_1 = 0 \Rightarrow q_1 \ldots q_n \in L = q \text{ ACCEPTING} \]

\[ \text{CASE 2:} \quad i \neq 1. \ \text{LOOK AT STATE p AFTER READING c_i z_i. WLOG q_i = 1, b_i = 0.} \]

\[ c_i \neq b_i \text{ AND \textit{LET}} \quad c_i \neq b_i \text{ \textit{AND \textit{LET}}} \]

\[ \text{E-CLOSED: THIS SHOWS FOR AN \textit{A N-FSA WE NEED} 2^n \text{ STATES FOR ANY EQUIVALENT DFA. \textIT{C}AN IMPROVE THIS TO 2^n \text{!}} \]

\[ \text{E-TRANSITIONS} \]

\[ \text{A NEW FEATURE THAT WILL BE HELPFUL IN PROGRAMMING NFAS.} \]

\[ \text{IDEA: ALLOW TRANSITIONS WITHOUT READING ANY INPUT SYMBOLS (IE. READING E).} \]

\[ \text{EXAMPLE:} \quad \text{SUPPOSE L_1 = L(N_1), L_2 = L(N_2) \quad \text{THEN} \quad L_1 \cup L_2 = L(E) \ \text{WHERE:} \]

\[ \text{FORMALLY, LET Z_E := Z_U \cup E} \quad \text{WLOG E \& Z TO AVOID CONFUSION) \quad \text{THEN} \quad E = (Q, \Sigma, Z_E, q_0, E, F) \text{ AS AN E-NFA IFP.} \]

\[ \text{Q, Z, q_0, F AS IN NFAS BUT NOW S := Q \times Z_E \rightarrow P(Q) \text{ IN ORDER TO DEFINE S FOR E-NFAS WE NEED A DEFINITION.} \quad \text{FOR q \in Q:} \]

\[ \text{ECLOSE(q)} := \{ q' \in Q ; q' \text{ CAN \ BE REACHED FROM q} \text{ BY JUST FOLLOWING E-TRANSITIONS OVER MULTIPLE TRANSITIONS} \} \]

\[ \text{EXAMPLE:} \quad \text{ECLOSE(1) = \{ 1, 2, 3, 4, 5 \}.} \]
THE EXTENDED TRANSITION FUNCTION FOR E-NFA:

\[ E = (Q, \Sigma, \delta, q_0, F) \]

\[ \delta'(q, \epsilon) = \text{CLOSE}(q) \]

\[ \delta'(q, \sigma a) = \bigcup_{j=1}^{m} \text{CLOSE}(r_j) \text{ where:} \]

\[ \{ r_1, \ldots, r_m \} = \bigcup_{p \in \delta'(q, \epsilon)} S(p, a) \text{ (as before in NFAs)} \]

(MAKE A TRANSITION THEN CLOSE IT.)

\[ L(E) = \{ w : \delta'(q_0, w) \cap F \neq \emptyset \} \]

EPSILON-ELIMINATION

WE CAN ALSO CONVERT E-NFAS TO DFA USING A SLIGHT TWEAK TO THE SUBSET CONSTRUCTION.

Let \( E = (Q_E, \Sigma, \delta_E, q_0, F_E) \). Define \( D = (Q_D, \Sigma, \delta_D, q_D, F_D) \) as follows:

1. \( Q_D = P(Q_E) \)
2. \( \Sigma = \Sigma \)
3. \( q_D = \text{CLOSE}(q_0) \)
4. \( F_D = \{ S \subseteq Q_E : S \cap F_E \neq \emptyset \} \)
5. For \( S \subseteq Q_E \) and \( \sigma \in \Sigma \):
   \[ \delta_D(S, \sigma) = \bigcup_{r_j \in S} \text{CLOSE}(r_j) \]
   where \( \{ r_1, \ldots, r_m \} = \bigcup_{p \in \delta'(q, \epsilon)} S(p, a) \) (as in subset)

THEOREM: FOR \( E \) AND \( D \) AS ABOVE, \( L(D) = L(E) \).

PROOF: (QUITE SIMILAR TO THAT FOR NFAS.)

PUZZLE: WHICH UNARY LANGUAGES ARE REGULAR?

REGULAR EXPRESSIONS

You are probably familiar with these:

\[ [A-Z][a-z]* \quad [ ] [a-z][A-Z] \]

matches "ens d\".

FORMALLY A REGULAR EXPRESSION (REG. EXP.) IS DEFINED INDUCTIVELY AS FOLLOWS.

1. \( \epsilon, \phi \) are reg. exps.
2. \( \alpha \) for \( \forall a \in \Sigma \) is a reg. exp.
3. If \( R_1 \) and \( R_2 \) are reg. exps, so are \( (R_1 \cup R_2) \) and \( (R_1^* \cup R_2^*) \).

We also define THE LANGUAGE ACCEPTED BY REG. EXP. INDUCTIONALLY:

\[ L(\alpha) = \{ w \} \quad L(\epsilon) = \{ \epsilon \} \quad L(\phi) = \emptyset \]

\[ L(R_1 \cup R_2) = L(R_1) \cup L(R_2) \]

\[ L(R_1^*) = (L(R_1))^* \]

(except for \( R = [a-z][A-Z][a-z]\))

EXAMPLE: WRITE A REG. EXP. FOR STRINGS WITH ALTERNATING 0'S AND 1'S:

\( (01)^* \) NOT QUITE: BEGINS WITH 0 ENDS WITH 1 ALWAYS.

So: \( (01)^* + (10)^* + 1(01)^* + 0(10)^* \)

\[ = (\epsilon + 1)(01)^*(\epsilon + 0) \]

NOTE: THE PRECEDENCE OF OPERATORS ARE: \( \cdot \rightarrow \text{CONCAT} \rightarrow + \)

So: \( (01)^* + = (0(1^*))^* \)
Theorem: \( L = L(CR) \) for some regex R \( \Leftrightarrow L = L(D) \) for some DFA D.

Proof: \( \Rightarrow \) We build an \( \epsilon \)-NFA for a given regex R as follows:

- \( R = \emptyset \)
- \( R = \epsilon \)
- \( R = \alpha \)
- \( R = R_1 \cup R_2 \)
- \( R = R_1 R_2 \)
- \( R = R_1^* \)

All accepting states.

(Example: \((ab+a)^*\) ~\( q_1 \) \( q_3 \) \( q_2 \) \( q_0 \) \( q_1 \) \( q_2 \) \( q_3 \) \( q_4 \) \( q_5 \)

\( ab + a \):

\( (ab+a)^* \)

(\( \Leftarrow \)) We start with a DFA and build a regex for it by eliminating its states one at a time. Here is the idea: suppose S is "on the way" from \( p \) to \( q \):

We are going to remove S, and label path

From \( p \) to \( q \) in a way that captures all strings that take \( p \) to \( q \) (i.e., repair the ripping!)

But there may be only many such strings.

So we consider a "generalized" automaton whose arrows are labeled with regex.

We will operate on such GDFA, as follows:

Is converted to:

To make the conversion slightly easier assume:

1) Only one \( q_{start} \), only one \( q_{accept} \), \( q_{start} \neq q_{accept} \)
2) No incoming arrows to \( q_{start} \)
3) No outgoing arrows from \( q_{accept} \).

So the above procedure will leave us with

For some regex R, this is what we wanted!
EXAMPLE:

RIPI: (3,2), (5,3)
(2,2), (2,3)
(3,2), (3,3)

RIP2: (5,a), (5,3)
(3,a), (3,3)

RIP3: (5,a)
(a (a+b) a+b) (b a+b) (a+b) (a+b) ab+b

OBSERVATIONS ABOUT REG EXP.

L + M = M + L ; \( L(M)N = L(MN) \)
\( \varnothing + L = L = L_0l = L ; B L = L_0l ; \varnothing \varnothing = \varnothing ; \varnothing = \varepsilon = E = E \)
\( L(M + N) = L(M) + L(N) \)
\( L(M) + N = M + L(N) \)
\( L + L = L \)
\( (L)^* = L^* \)
\( L^* + L^* = L^* \)

CLOSURE PROPERTIES OF REGULAR LANGUAGES

1) \( L, M \text{ REGULAR} \implies L \cup M, L \cdot M, L^* \text{ REGULAR.} \)
   (FOLLOWS FROM EQUIVALENCE WITH REG EXP)

2) \( L = \varepsilon \implies L \text{ is regular.} \)
   \textbf{PROOF:} \( L = L(L) \) FOR DFA D. WITH ACCEPTING STATES \( F_D \).
   CONSIDER DFA D WITH \( F_D = \varepsilon - F_D \).
   THEN \( 3 \)
   \( \delta_D(q_0, w) \in F_D \iff \delta_D(q_0, w) \in E - F_D \iff w \notin L \).

3) \( L \cup M \text{ is regular.} \)
   \textbf{PROOF:} \( L \cup M = L \cup M \)
   \( L \cup M \) is regular.

4) \( \text{FOR A STRING } w = w_1 \ldots w_n \text{ DEFINE ITS REVERSAL } w^R = w_n \ldots w_1 \)
   \( L \text{ is regular} \implies L^R \text{ is regular.} \)
   \textbf{PROOF:} \( L = L(E) \) FOR REG EXP E. WE BUILD \( E^R \).
   \( E^R = \varepsilon, E \)
   \( E = E_1 + E_2 \)
   \( E = E_1 \cdot E_2 \)
   \( E^R = E_1^R + E_2^R \)
   \( E^R = E_1^R \cdot E_2^R \)
   \( E^R = (E^R)^* \)
   \( E^R = (E^R)^* \)
   \( E^R = (E^R)^* \)
   \( E^R = (E^R)^* \)

5) \( \Sigma, \Delta \text{ BE ALPHABETS; A HOMOMORPHISM IS A MAP } h : \Sigma^* \to \Delta^* \).
   \( h(x) = h(x) \).
   \( h(xy) = h(x)h(y) \).
   \textbf{EASY TO SHOW IN ALL CASES: } w \in L(E) \iff w^R \in L(E^R)
   \text{ AND THEN AN INDUCTION.} \)

6) \( \Sigma, \Delta \text{ BE ALPHABETS; A HOMOMORPHISM IS A MAP } h : \Sigma^* \to \Delta^* \).
   \( h(x) = h(x) \).
   \( h(xy) = h(x)h(y) \).
   \textbf{EASY TO SHOW IN ALL CASES: } w \in L(E) \iff w^R \in L(E^R)
   \text{ AND THEN AN INDUCTION.} \)
5.1) If L is regular and h a homomorphism then h(L) is regular.

Proof: Let L=LE(E) for some reg exp E.

For a reg exp E:

h(E) := replace a in E with h(a) for all aεΣ.

Claim: L(h(E)) = h(L(E))

Proof: If E = ∅ both sides = ∅.

E = E both sides = h(E).

E = a both sides = h(a).

Induction: Suppose E = F+G.

By construction h(E) = h(F)+h(G).

Also: L(F) = L(F)∪L(G) h(L(F)) = h(L(F))∪h(L(G)) because h is applied element-wise.

By IH: h(L(F)) = L(h(F)) and h(L(G)) = L(h(G)), so done for E = F+G.

The other cases are similar.

5.2) h⁻¹(L) is also regular.

Let L = LE(D) for some DFA D = (Q, Σ, δ, q₀, F).

Consider DFA B = (Q, Σ, δ, q₀, F) where

\[ \gamma(q, a) = \delta(q, h(a)) \]

(\(B\) on input a runs D on h(a).)

By induction:

\[ \gamma(q₀, x_a) = h(δ(q₀, h(xₐ)), a) \]

\[ = h(δ(q₀, h(xₐ)), a) \]

\[ = h(δ(q₀, h(xₐ)), a) = h(δ(q₀, h(xₐ))) = h(δ(q₀, h(xₐ))) \]

So if k∈L(C) then \(xₐ\) ∈ L(B).

THE PUMPING LEMMA.

Is \(L₀ = \{0^n1^n : n ≥ 3\}\) regular?

Suppose a k-state DFA accepts it. Consider words \(C₀,n, \ldots, \alpha, k\).

By pigeonhole principle \(k ≥ j\) st. After \(C₀ \& C₁\) DFA is in same state q. But all that DFA remembers is q.

So if it accepts \(C\), it will also accept \(C₀\).

Theorem: Let L = LC(D) for a DFA D. Then \(∃\) pump (the pumping length) \(v ≤ wεL\) with \(1w \geq p\).

We can break w into \(w = yzx\).

(1) \(y \neq ε\) (2) \(|y| ≤ p\) (3) \(q_i ≥ yₗ \) \(z \neq ε\).

Non-trivial. \(xy\) near the beginning. \(w\) can be "pumped" beginning.

Proof: Let \(p = |q|\). \(D = (Q, Σ, S, q₀, F)\).

Let \(w = a₁ \ldots aₘ\) with \(m ≥ p\). (If no such \(w\) exists we are vacuously done)

Let \(q_i = δ(q_i, a_i, \ldots, a_q); i = 0 \ldots m\) (at least \(p+1\) of these).

By pigeonhole: \(∃ i ≠ j \) \(p \neq a_i, q_i = q_j\). WLOG \(i < j\).

Let \(x = a₁ \ldots a_q \neq q_j; y = a_i \ldots a_j\); \(z = a_j \ldots a_m\) \(= \|xy\| \leq p\).

\(x\) takes us from \(q_i \) to \(q_i\); \(y\) from \(q_i\) to \(q_j\); \(z\) from \(q_j\) to \(q_m\) an accept state.

\(x \neq ε\), \(z\) may be \(ε\), but \(y \neq ε\) since \(p < j\).

Hence \(x \neq ε\) and \(xy\) \(= \neq ε\) for any \(k≥0\).
EXAMPLES:

(1) $L_1 = \{0^n1^m : m \geq n \}$ is not regular.

Suppose it is. Let $p = \text{the pumping length}.$

Let $w = 0^p1^p.$ Note $|w| \geq p.$

Then $\exists x, y, z \in L. w = xyz,$ $|xy| \leq p,$ $y \neq \varepsilon.$

And $xy^iz \in L$ for $i \geq 0.$

Since $|xy| \leq p \Rightarrow y$ only has $0$'s.

$\Rightarrow$ all $1$'s in $z.$ $xy^iz$ has $p$ $0$'s and $p$ $1$'s.

Then $xz$ has less $0$'s than $1$'s since $y \neq \varepsilon.$

(2) $L_2 = \{ w : w \in \{0, 1\}^* \}.$

$p$ is in the lemma. $w = 0^p1^p \in L.$ $|w| \geq p.$

Write $w = xyz.$ Since $|xy| \leq p,$ $y$ only has $0$'s.

$\Rightarrow xy^iz \text{ has more } 0 \text{'s than } 1 \text{'s.}$

(Note: If we took $w = 0^p0^p$ then it could be

That $x = 0^m,$ $y = 0^n,$ $z = 0^{2p - (m + n)}.$ With $n$ even.

Then $0^m(0^m)^i(0^{2p - (m + n)}) \in L \text{ for all } i \geq 0.$

(3) $L = \{ 1^n : n \geq 0 \}.$

Take $w = 1^{p^2}.$ Note $w = xyz.$ Then $|xy^iz| \leq p^2 < p(p + 1)^2$ but we also have $p^2 < p^2 + 2p < |xy^iz| \text{ not a perfect square.}$

(4) $L = \{ 0^i1^j : i \geq 0, j > i \}.$

Take $w = 0^i1^j.$ Write $w = xyz.$ $|xy| \leq p \Rightarrow y$ has only $0$'s.

$\Rightarrow xz$ has $p$ or less $0$'s.

(Note: If we took $w = 0^p1^p$ then it could be

That $x = 0^m,$ $y = 0^n,$ $z = 0^{2p - (m + n)}.$ With $n$ even.

Then $0^m(0^m)^i(0^{2p - (m + n)}) \in L \text{ for all } i \geq 0.$

(5) $L = \{ w : \#_0(w) \neq \#_1(w) \}.$

Note $L_3 = \{ 0^i1^j : i \geq 0 \}$ is regular.

If $L$ were regular, then so is $L \cap L_3 = \{ 0^i1^j : i \geq j \}.$

But then so is $L \cap L_3 = \{ 0^m1^n : m \geq 0 \}.$

(6) $L_{\text{prime}} = \{ 1^n : n \text{ prime} \}.$

Let $\overline{p} = \text{next prime } (p) \text{ in pumping}$

$w = \overline{p}$ $w = xy \in L.$ $|xy| \leq p$.

Let $m = |y|.$ By pumping $xy^{p-m}z \in L.$

But $|xy^{p-m}z| = |xz| + (p - m)|y|$

$= p - m + (p - m)m = m^2 + (p - m).$

Now $m + 1 \geq 2 \text{ since } y \neq \varepsilon.$

$p - m \geq 2 \text{ since } \overline{p} \geq p + 2 \geq m + 2.$

So the length of $|xy^{p-m}z|$ is not a prime.

TESTING EMPTINESS/MEMBERSHIP:

Given $w \in \Sigma^*, \text{ ask } w \in L(D) ?$ $w \in L(E) ?$ $w \in L(R) ?$

L(D) $\neq \emptyset ?$ $L(E) \neq \emptyset ?$ $L(R) \neq \emptyset ?$

\[ \begin{array}{|c|c|c|c|} \hline L & \text{DFA} & \text{E-NFA} & \text{Reg. Exp.} \\ \hline L = \emptyset ? & \text{GRAPH REACHABILITY} O(n^2) & \text{GRAPH REACHABILITY} O(n^2) & L(R) = \emptyset \text{ or } L(E) = \emptyset \\ \hline L = \emptyset ? & \text{SIMULATE} O(n) & \text{SIMULATE} O(n^2) & \text{CONVERT TO E-NFA} \\ \hline w \in L ? & \text{GRAPH REACHABILITY} O(n^2) & \text{CONVERT TO E-NFA} \text{ (INDUCTIVE METHOD TRICKY FOR R*)} \\ \hline \end{array} \]
MINIMIZATION

**Given a DFA D, does it have any redundant states? How can we find the minimal DFA that is equivalent to D?**

**Definition:** We say two states p, q are equivalent iff for all w, S(p, w) ∈ F ⇔ S(q, w) ∈ F. We write p ≡ q.

**Note:**
- p ≡ p (reflexive)
- p ≡ q ⇔ q ≡ p (symmetric)
- p ≡ q ∧ q ≡ r ⇒ p ≡ r (transitive)

Thus ≡ is an equivalence relation. Hence it partitions the states into equivalence classes.

Two equivalent states can be merged into one.

**Table-Filling Algorithm**

Our strategy to find all equivalent pairs: find distinguishable (non-equivalent) states on E.

Then inductively work our way through the DFA to find all other distinguishable states.

We'll show all remaining states are equivalent.

We use: if S(p, w) ∈ p ≡ q then p ≡ s.

**Example:**

\[
\begin{array}{c}
\text{1. } \{c, A, E, F\}, \ldots \text{ t.e.} \\
\text{2. } \{g, H\} \rightarrow \{f, F, E\}, \{D, E\} \\
\text{3. } \{c, G\} \rightarrow \{d, G\}, \{f, G\}, \{d, B\}, \{f, B\} \\
\text{4. } \{b, F, E\}, \{b, D\} \\
\text{5. } \{B, F, E\} + 1 \rightarrow \{B, E\}, \{B, A\} \\
\end{array}
\]

**Notes:**
- No more states that we need (discovered)
- Running time: O(n^2)

*Occasional bugs (not found)
**Theorem:** Two states are not distinguished by table-filling if they are equivalent.

**Proof:** Let \( \{p, q\} \) be such that

1. \( \exists w \in W \) such that \( \hat{s}(p, w) \neq \hat{s}(q, w) \)
2. Also does not find \( \{p, q\} \) as distinguishable.

Call such \( \{p, q\} \) bad.

Take a bad \( \{p, q\} \) with shortest \( w \).

\( w = q_1q_2 \ldots q_n \), \( \hat{s}(p, w) \in F \land \hat{s}(q, w) \notin F \). (wlog)

Let \( r = s(p, q_1) \), \( s = (q, w) \rightarrow \hat{s}(r, q_1q_2 \ldots q_n) \), \( s = (q, w) \land \hat{s}(r, q_1q_2 \ldots q_n) \notin F \)

\( \Rightarrow \{r, s\} \) with shorter \( q_2 \ldots q_n \) distinguishable.

\( \Rightarrow \) algo finds \( \{r, s\} \) as distinguishable.

\( \Rightarrow \) it also finds \( \{p, q\} \) as distinguishable in next step.

Since \( s(p, q_1) \neq r \land s(q, q_1) = s \). \( \therefore \).

**Theorem:** If the algo marks \( \{p, q\} \) then \( p, q \) are not equivalent.

**Proof:** Induction on stages of the algo.

**Basis:** \( i = 1 \). If algo marks \( \{p, q\} \) in the first stage then clearly \( \{p, q\} \) are not equivalent.

**Induction:** Suppose true for stages \( i \leq n \). We prove for \( n + 1 \).

If algo marks \( \{p, q\} \), then this is because \( 3q \rightarrow \).

\( r = \hat{s}(p, q) \) and \( s = \hat{s}(q, q) \) and \( \{r, s\} \) was marked in a previous step.

By induction \( \exists w \), s.t. \( \hat{s}(p, w) \in F \land \hat{s}(q, w) \notin F \) (wlog)

Then \( \hat{s}(r, w) \in F \land \hat{s}(q, w) \notin F \). \( \therefore \).

**Note:** If \( p \neq q \) for some DFA, then for all \( q \in \Sigma \), \( \hat{s}(p, a) \neq \hat{s}(q, a) \). That is equivalent states go to equivalent states. (Follows directly from def)

**Minimization:** Let \( \alpha = \{S_1, \ldots, S_m\} \) be a partition of the states using the table-filling algorithm.

1. Eliminate all unreachable states.
2. Let the start state be the one set \( S_i \) containing the start state of \( D \). (wlog \( S_i = S_1 \))
3. Let accept states be all \( S_i \) that contain an accept state of \( D \). (there could be many)

**Eg.**

![Diagram](image)

- \( B \times C \xrightarrow{} B \times C \xrightarrow{} \{A, B \} \) + 1 \rightarrow \{A, B \}

For \( i, q \in \Sigma \) define the transition function:

\[ \hat{s}_i^{M}(q, a) = \left[ \hat{s}_q^{D}(q, a) \right] \]

For any \( q \in S_i \), \( \hat{s}_i^{M} \) is an equivalence class of \( \hat{s}_q^{D}(q, a) \)

This is well-defined: For all \( q_1, q_2 \in S_i \), \( q_1 \neq q_2 \) \( \Rightarrow \)

![Diagram](image)
The minimized DFA cannot be beaten

Let D be a DFA. Let M be its minimization. Let N be another DFA such that L(M) = L(N).

Suppose |Q_M| < |Q_N|.

We run MINIMIZE on M and N together.

Assuming Q_M ∩ Q_N ≠ ∅ so \( \hat{S}_{MN} = \{ \hat{S}_{M}(q) : q \in Q_M \} \) \( q \in Q_N \).

\( Q_{MN} = Q_M \cup Q_N \).

Let \( p \in Q_M \rightarrow \exists w \). \( \hat{S}_N(q^0, w) = p \). (\( * \) states removed

Let \( q^0 = \hat{S}_N(q^0, w) \).

Since L(M) = L(N), \( \forall w \) \( \hat{S}_M(q^0, w) = \hat{S}_N(q^0, w) \).

Thus \( \forall w \in w \). \( \hat{S}_{MN}(q^0\cdot w') = \hat{S}_N(q^0, w') \)

\( \equiv w' \hat{S}_N(p, w') = \hat{S}_M(q, w') \) \( \equiv w' \).

So \( p \) and \( q \) are equivalent under \( S_{MN} \).

So we showed: every state of M is equiv.

To a state of N. Under \( S_{MN} \).

If \( \# \) states \( (M) > \# \) states \( (N) \) then

\( \exists \) two states \( p \neq p' \) of M equiv. To some

state \( q \) of N. But then \( p \) and \( p' \)

are equiv. Under \( S_{MN} \), and hence also under \( S_M \).

\( \{q\} \) to minimization

Since both in DFA M.

Similarly every state of \( N \) is equiv. To a state of M.

Let \( \{q_i\} \) be states of \( M \). Suppose \( q_i = p_i \) (under \( S_{MN} \))

\( \{p_i\} \) be states of \( N \).

We show: \( \forall a \in \Sigma \) \( S_M(q_i, a) = q_j \Rightarrow S_N(p_i, a) = p_j \).

Indeed: if \( S_M(q_i, a) = q_j \Rightarrow S_{MN}(q_i, a) = q_j \).

Let \( S_N(p_i, a) = p_k \Rightarrow S_{MN}(p_i, a) = p_k \).

But we know if \( q_i = p_i \) then \( S_{MN}(q_i, a) = S_{MN}(p_i, a) \).

Hence \( q_j = p_k \) and thus \( j = k \).

So M and N are the same DFA up to relabeling.

NFA minimization?

Consider:

\[ \begin{array}{ccc}
\bullet & \bullet & \bullet \\
\bullet & \circ & \circ \\
\end{array} \]

\[ \{ w : w \text{ ends in } \circ \} \]

Running MINIMIZE on this gives:

\[ \begin{array}{ccc}
B & X \\
C & X \\
A & \emptyset \\
\end{array} \]

But now take \( \{A, C\} \rightarrow \{B, A\} \).

So \( \{A, C\} \) distinguishable.

So this NFA is "MINIMIZED." But it is not:

Can minimize this using exhaustive search. \( O(2^{2^n}) \).

But not in time polynomial in \( \# \) states.

(That will imply \( P = \text{PSPACE} \). We might see this later.)
EQUVALENCE: RUN MINIMIZATION ON JTM M N.
CHECK IF START STATES ARE EQUIVALENT.

D₁

D₂

DEC under S₂

\[ E + E^* \]

\[ \text{E in } L; \text{if } A \text{ in } L, \text{ then } OA1 \text{ in } L \]

\[ (A₀ = E; A_{n+1} = OA₁) \]

\[ \text{OR: \ CONSIDER \ EXPRESSIONS \ in } "+" \ \text{AND} \ "\cdot" \]

- \( a \) is in \( L \)
- IF \( A, B \) are in \( L \), so is \( A + B \)
- IF \( A, B \) are in \( L \), so is \( A \cdot B \).

\[ (A₀ = a; Aₙ = Aₙ \cdot Aₖ \text{ m, k } \in \text{N} \]

\[ Aₙ = Aₙ \cdot Aₖ \text{ m, k } \in \text{N} \]

WE CAN WRITE THESE, MORE COMPACTLY AS:

\[ A \rightarrow OA₁ \]

\[ A \rightarrow E \]

\[ A \rightarrow a \]

\[ A \rightarrow A + A \]

\[ A \rightarrow A \cdot A \]

\[ A \rightarrow a \text{ (FROZEN CHOMSKY) } \]

\[ \text{S } \rightarrow \text{NP VP} \]

\[ \text{NP } \rightarrow \text{CN} | \text{CN PP} \]

\[ \text{VP } \rightarrow \text{CV} | \text{CV NP} \]

\[ \text{PP } \rightarrow \text{P CN} \]

\[ \text{CN } \rightarrow \text{A N} \]

\[ \text{CV } \rightarrow \text{V | V NP} \]

\[ A \rightarrow a | \text{the } N \rightarrow \text{boy } | \text{girl } | \text{flower } \]

\[ V \rightarrow \text{touched } | \text{tells } \]

\[ a \text{ girl with a flower likes the boy (PAN TREE) } \]
Formally, a context-free grammar is a 4-tuple
\[ G = (V, \Sigma, R, S) \]
where:
1) \( V \) is a finite set of variables
2) \( \Sigma \) is a finite set of terminals, \( \Sigma \cap V = \emptyset \)
3) \( R \) is a finite set of rules (or productions)
   each rule is a pair of the form \( A \rightarrow w \) (\( A \in V \) \( \wedge w \in (\Sigma U V)^* \)), \( \emptyset \) called head, \( w \) body
4) \( S \in V \) is the start variable.

**Example:**
\[ G = (\{S\}, \{(, )\}, R, S) \]

**Grammar:**

\[ S \rightarrow (S) | SS | \epsilon \]

Body \( \in \{S,(,)\}^* \)

3 rules/productions

This grammar generates all properly nested parentheses.

**Note:** CFGs are similar to Backus-Naur form for syntax of programming languages.

**Language of a Grammar:**

Let \( u,v,w \in (\Sigma U V)^* \). Let \( A \rightarrow w \) be a rule. We say \( uA \rightarrow v \) yields \( uv \) and write \( uA \Rightarrow uv \).

We say \( u \) derives \( v \), written \( u \Rightarrow^* v \), if \( u = v \) or \( \exists \) a sequence \( u_1, u_2, \ldots, u_k \in (\Sigma U V)^* \) such that

\[ u \Rightarrow u_1 \Rightarrow u_2 \Rightarrow \cdots \Rightarrow u_k \Rightarrow v. \]

We define:

\[ L(G) = \{ w \in \Sigma^* : S \Rightarrow w \} \]

**Example:**

Let \( D \) be a DFA with transition function \( S(\delta(q,f)) = \delta \). Then there is a CFG \( G \) that generates \( L(D) \). Add rules \( R_i \rightarrow qR_j ; R_i \rightarrow \epsilon \) for \( q \in F \)

\[ S = R_0 \]

Thus in a \( CFG \):

- **Variables** \( \rightarrow \) **States**
- **Terminals** \( \rightarrow \) **Alphabet**
- **Rules** \( \rightarrow \) **Transition Function**

**Note:** We have shown every REG. LANG. is context-free, but not every CFL is REG. \( \{0^n1^n \mid n \geq 0 \} \)

**Leftmost Derivations:**

Derivations can proceed by replacing variables in arbitrary order.

In leftmost derivations we require that at each step we replace the leftmost variable by one of its production bodies. (Note by def. every variable has a production.)

Formally: say \( uA \Rightarrow uv \) iff \( w \in \Sigma^* \) (terminals only)

And \( A \rightarrow w \) is a production.

Say \( u \Rightarrow v \) if \( u \Rightarrow u_1 \Rightarrow \cdots \Rightarrow u_k \Rightarrow v \).

**Example:**

\[ S \Rightarrow SS \Rightarrow (S)S \Rightarrow (S)(S) \Rightarrow S \Rightarrow \epsilon. \]

Is a derivation, but not leftmost.

\[ S \Rightarrow 3S \Rightarrow (S)S \Rightarrow (S)(S) \Rightarrow (S)(S) \Rightarrow \epsilon \]

Is leftmost.

**Note:** Can define rightmost derivations analogously. The 1st derivation above is neither leftmost nor rightmost.
PARSE TREES:

Let G be a CFG. A tree is a parse tree for G if:
1) The root of the tree is labeled with S.
2) The interior nodes are labeled with variables.
3) The leaf nodes are labeled with terminals or E if with E then only child.
4) If an interior node is labeled with A and its children with $X_1, X_2, \ldots, X_k$ (from left to right), then $A \rightarrow X_1 X_2 \ldots X_k$ is a production.

The yield of a tree is concatenation of leaves from left.

Example: $(1) \quad S \rightarrow SS | (S) | E$

yield = ((1)) (preorder traversal of the tree)

$(2) \quad E \rightarrow E + E | (E) | E$

yield = a + axa.

Ambiguity:

A CFG G is ambiguous if some w E L(G) has two or more parse trees. (= meaning $\equiv$)

Example: $S \rightarrow SS | (S) | E$. Then $w = ox(\_)$ E L(G).

Two parse trees:

Example: $E \rightarrow E + E | (E) | E$

Then $w = a + axa E L(G)$.

Two parse trees:

Natural examples:

The curl touches the boy with flower.
La fille touche le garçon avec la fleur.
Das Mädchen berührt den Jungen mit der Blume.

Parse trees & leftmost derivations:

1) For every parse tree there is a leftmost derivation.
2) For every leftmost derivation there is a parse tree.

The proofs of these are by induction.

1) Suppose $A_i$ of height $h$. Then $A \rightarrow a_1 \ldots a_m$ a production.

Thus $A \rightarrow a_1 \ldots a_m$ (generalized it with any root label)

By IH: $X_i \rightarrow w_i$

Thus $A \rightarrow X_1 X_2 \ldots X_k \rightarrow w_1 w_2 \ldots w_k a_1 \ldots a_m$. Thus $A \rightarrow w$.

2) If $A \rightarrow a_1 \ldots a_m$ in 1 step, then it is a production $A \rightarrow a_1 \ldots a_m$ a parse tree.
Let $A \xrightarrow{\alpha} W$ be a k-step LR(1) derivation.

First step is $A \xrightarrow{\alpha} X_1 \cdots X_k$.

But there can write $W=W_1 \cdots W_n$ in S. $X_i \xrightarrow{\alpha} W_i$.

And all these have $\leq k-1$ steps. So $X_i$ have parse trees.

So have parse tree.

![Parse Tree Diagram]

Which has yield $W$.

(Also: Applies to Rightmost and General Derivations)

Note: For LMDs and PTS:

- LMDs differ when a different production used.
- Different trees to different LR(1) derivations.
- LM Der. = Parse Trees.
- (Trees differ at the point)

So: $G$ ambiguous if some $w \in L(G)$ has $\geq 2$ leftmost derivations.

Inherent Ambiguity:

Suppose $L(G)$ and $G$ ambiguous. This does not rule out the possibility that there is some other $G'$ S.

$L(G') = L(G)$ and $G'$ is unambiguous.

If there is no such $G'$ we call $L$ inherently ambiguous.

Example: Let $G$ be $S \rightarrow S \mid (S) \mid E$

$G'$ be $B \rightarrow (RB \mid E \mid \epsilon)$.

$R \rightarrow ) \mid (RR$

$L(G) = L(G')$, $G$ ambiguous $G'$ unambiguous.

(Exercise: Look at (1))

A always appears at match $A$ is wart.
CHOMSKY NORMAL FORM

When working with CFGs it is often convenient to have them in a simplified form (e.g. in testing membership).

A CFG is in Chomsky Normal Form if every rule is of the form:

\[ A \rightarrow BC \quad \text{or} \quad A \rightarrow a \quad \text{or} \quad S \rightarrow \varepsilon. \]

where \( a, b \in \Sigma \) A TERMINAL \( a \neq \varepsilon \),
\( A, B, C \in \mathbb{V} \) VARIABLES and \( B \neq S \) and \( C \neq S \).

Theorem: Any CFL is generated by a CFG in the Chomsky Normal Form.

Proof: 1) Add a new start variable \( S_0 \): add rule \( S_0 \rightarrow S \) (S in the original CFG), so start symbol never on the right.

2) Eliminate \( \varepsilon \)-rules:

2.1) Remove \( A \rightarrow \varepsilon \) for a non-start variable.

2.2) For each occurrence of \( A \) in body of a production, add new rule with that occurrence deleted, so if we have \( R_1AuAv \) we add rules: \( R_1uAuv; R_1uAv; R_1uv \).

Note: if we have \( R \rightarrow A \) we add \( R \rightarrow \varepsilon \) unless we already removed \( R \rightarrow \varepsilon \).

2.3) Repeat for all non-start \( A \).

3) Eliminate unit rules:

3.1) For each \( B \rightarrow A \) remove it.

3.2) For each \( B \rightarrow uA \) add \( A \rightarrow u \).

Note: if \( A \rightarrow \varepsilon \) already deleted, we won’t add it.

3.3) Repeat.

4) Replace each \( A \rightarrow u_1 \ldots u_k \), \( k \geq 3 \) with \( A \rightarrow u_1u_1; A \rightarrow u_1u_2; \ldots; A \rightarrow u_{k-1}u_k \).

4.2) Replace any rule \( A \rightarrow u \varepsilon \) with \( A \rightarrow u \varepsilon \).

Example:

\[ S \rightarrow ASA | aB \]
\[ A \rightarrow BIS \]
\[ B \rightarrow \varepsilon \]

1) Add \( S_0 \rightarrow S \).

2) Remove \( B \rightarrow \varepsilon \):

\[ S_0 \rightarrow S \]
\[ S \rightarrow ASA | aB | a \]
\[ A \rightarrow BIS \]
\[ B \rightarrow \varepsilon \]

3) Remove \( S \rightarrow S \):

\[ S_0 \rightarrow S \]
\[ S \rightarrow ASA | aB | a | A | S \]
\[ A \rightarrow BIS \]
\[ B \rightarrow \varepsilon \]

5) Remove \( A \rightarrow \varepsilon \):

\[ S_0 \rightarrow S \]
\[ S \rightarrow ASA | aB | a | A | S \]
\[ A \rightarrow BIS \]
\[ B \rightarrow \varepsilon \]

6) Remove \( A \rightarrow \varepsilon \):

\[ S_0 \rightarrow S \]
\[ S \rightarrow ASA | aB | a | A | S \]
\[ A \rightarrow BIS \]
\[ B \rightarrow \varepsilon \]

(No More \( \varepsilon \)-Rules)

3a) Remove \( S \rightarrow S \):

\[ S_0 \rightarrow S \]
\[ S \rightarrow ASA | aB | a | A | S \]
\[ A \rightarrow BIS \]
\[ B \rightarrow \varepsilon \]

3b) Remove \( A \rightarrow S \):

\[ S_0 \rightarrow S \]
\[ S \rightarrow ASA | aB | a | A | S \]
\[ A \rightarrow BIS \]
\[ B \rightarrow \varepsilon \]

(No More \( \varepsilon \)-Rules)

4a) Split:

\[ S_0 \rightarrow S; S \rightarrow ASA; A \rightarrow ASA \]

\[ S_0 \rightarrow S_1 | S_2 | S_3 \]
\[ S \rightarrow A_1 | A_2 | A_3 | A_4 \]
\[ A \rightarrow BIS \]
\[ B \rightarrow \varepsilon \]

(No More \( A \rightarrow \varepsilon \))

4b) Replace:

\[ S_0 \rightarrow A_1 | B | A | A_2 | A_3 | A_4 \]
\[ S \rightarrow A_1 | B | A | A_2 | A_3 | A_4 \]
\[ A \rightarrow BIS \]
\[ B \rightarrow \varepsilon \]

(Done!

Combine: \( u, v, w, x \)
NOTE: THE ORDER OF THE OPERATIONS IS IMPORTANT.

E.G. ELIMINATING E-RULES MIGHT CREATE UNIT RULES.

SO E-ELIMINATE THEN UNIT ELIMINATE.

BUT SOME CAN BE INTERCHANGED AND THIS AFFECTS

THE SIZE OF THE FINAL GRAMMAR.

THERE ARE OTHER "NORMAL FORMS." E.G., GREIBACH NORMAL

FORM PUTS THE PRODUCTIONS IN THE FORM \( A \rightarrow \alpha x \)

WHERE \( \alpha \in \Sigma \) AND \( \epsilon \notin \alpha \). HENCE A STRING OF LENGTH

\( n \) HAS A DERIVATION OF LENGTH \( n \).

CLAIM: If \( \Sigma^* \leq L(G) \), \( |w| \geq 1 \), THEN ANY DERIVATION OF \( w \)

FROM ANY VARIABLE \( X \) HAS LENGTH EXACTLY \( 2n-1 \).

PROOF: IF \( |w| = 1 \) USE RULE \( \epsilon \rightarrow C \). \( 2n-1 = 1 \).

SUPPOSE TRUE FOR \( |w| < m \).

LOOK AT THE FIRST STEP OF DERIVATION.

IF \( X \rightarrow \epsilon \) THEN \( |w| = 1 \), AND WE DID THIS CASE.

IF \( X \rightarrow X_1X_2 \): WE CAN WRITE \( w = w_1w_2 \) WITH

\( X_1 \rightarrow W_1 \) AND \( X_2 \rightarrow W_2 \).

AND \( w_1 \epsilon \) AND \( w_2 \epsilon \).

\( (X_1, X_2 \) ARE NOT START SYMBOLS \) \( \Rightarrow |w| \leq n-1 \), \( |w| \leq n-1 \).

SO BY IH LENGTH OF DERIVATION = \( 1 + 2|w_1|-1 + 2|w_2|-1 \)

= \( 2|w| - 1 \).

PROVE OR DISPROVE: (I) THE CNF PROCEDURE PRESERVES

UNAMBIGUITY. (2) THE CNF PROCEDURE PRESERVES

AMBIGUITY.

CLOSURE PROPERTIES:

LET \( G_1 \) AND \( G_2 \) BE CFGs. WITH START VARIABLES \( S_1 \) AND \( S_2 \),

UNION: \( S \rightarrow S_1 | S_2 \) \( \rightarrow \) REVERSAL: \( \epsilon \rightarrow \epsilon \).

CONCAT: \( S_1 \rightarrow S_1 \epsilon \) \( \rightarrow \) REVERSAL: \( \epsilon \rightarrow \epsilon \).

STAR: \( S \rightarrow S_1 \epsilon \) \( \rightarrow \) REVERSAL: \( \epsilon \rightarrow \epsilon \).

NEW TYPE OF COMPUTATIONAL MODEL WITH AN EXTRA

COMPONENT CALLED A STACK.

A STACK IS A LAST-IN-FIRST OUT (LIFO) TYPE OF MEMORY.

INPUT \( \rightarrow \) STATE \( \rightarrow \) ACC/REJ \( \rightarrow \) INPUT \( \rightarrow \)

CONTRL \( \rightarrow \) ACC/REJ \( \rightarrow \) STACK.

WRITING: "PUSHES DOWN" SYMBOLS AT THE TOP.

READING: "POPS OUT" SYMBOLS AT THE TOP.

(A BIT LIKE PLATES IN A CAFETERIA!)

RECALL: \( \{0^n \} \): NOT REGULAR SINCE A DFA CANNOT

STORE ARBITRARY NUMBERS. \( 3 \) \( \epsilon \) IT. DFA IN STATE AFTER \( 0^n \).

WITH A PDA: READ INPUT AND PUSH O'S ONTO STACK.

AS SOON AS 1'S APPEAR, POP O'S FROM STACK.

ACCEPT IF STACK IS EMPTY.

PDAs ALSO COME IN (AT LEAST!) TWO FLAVORS:

DETERMINISTIC AND NON-DETERMINISTIC. WE'LL LOOK AT

NON-DETERMINISTIC HERE AND LEAVE PDPA'S TO A PROJECT

WHY? AS WE'LL SEE. PDA'S AND CFG'S ARE EQUIVALENT

IN POWER.

EXAMPLE: RECALL \( L = \{0^n w^n : w \in \Sigma^* \} \) WAS NOT REGULAR.

IDEA FOR A PDA:

1) PUSH SYMBOLS ONTO STACK.

2) NONDETERMINISTICALLY GUESS THE MIDDLE HAS BEEN READ.

3) POP SYMBOLS CHECKING THEY MATCH INPUT.

4) ACCEPT IF STACK EMPTY.
FORMALLY, A (NONDETERMINISTIC) PUSHDOWN AUTOMaton
IS A 6-TUPLE $P = (Q, \Sigma, \Gamma, S, q_0, F)$ WHERE:

1) $Q$ A FINITE SET OF STATES.
2) $\Sigma$, FINITE INPUT ALPHABET.
3) $\Gamma$, THE STACK ALPHABET.
4) $\delta$: $Q \times \Sigma \times \Gamma \rightarrow P(Q \times \Gamma^*)$ THE TRANSITION FUNCTION
5) $q_0 \in Q$ START STATE
6) $F \subseteq Q$ SET OF ACCEPTING STATES

EXAMPLE: $L = \{0^n1^n : n \geq 0\}$

$P = (Q, \Sigma, \Gamma, S, q_0, F) = (\{q_0, q_1, q_2, q_3, q_4\}, \{0, 1\}, \{\epsilon\}, q_0, F)$

**GRAPHICAL NOTATION / TRANSITION DIAGRAM:**

WRITE $a \rightarrow b \rightarrow c$ FROM STATE $q_i$ TO STATE $q_j$ TO SIGNIFY: $(q_i, c) \in \delta(q_i, a, b)$

**PDA $P$ ACCEPTS $w = w_1 \ldots w_m$ WHERE $w_i \in \Sigma^*$

IF $\exists \{r_0, r_1, \ldots, r_m \in Q\}$ SUCH THAT:

1) $(r_0, S_0) = (q_0, \epsilon)$
2) For $i = 0, \ldots, m-1$: $(r_{i+1}, b) \in \delta(r_i, w_{i+1}, a)$

WHERE $S_i = q^+ \atop{a, b} \in \Gamma^*$

3) $r_m \in F$.

1) START OUT WITH EMPTY STACK AT THE START STATE.
2) $P$ MOVES PROPERLY: AT STATE $q_i$, READ $w_{i+1}$ AND A SYMBOL ON STACK (BOTH COULD BE $\epsilon$).
3) GOTO SOME STATE $q_{i+1}$ AS GIVEN BY $\delta$.
4) POP $a$ AND PUSH $b$ (AS GIVEN BY $\delta$). UPDATE STACK FROM $S_i = q^+ \atop{a, b} \in \Gamma^*$.
5) ACCEPT IF AN ACCEPT STATE IS REACHED.

EXAMPLE: $L = \{ww^R : w \in \{0, 1\}^*\}$
**Example:**

\[ L = \{ a^i b^j c^k \mid i, j, k \geq 0 \} \]

**Theorem:** A language is context-free if and only if some PDA recognizes it.

\((\Rightarrow)\) Have a CFG \(\Rightarrow\) have a PDA.

**Idea:** Non-deterministically guess the sequence of correct substitutions.

**Problem:** We need to store intermediate strings, but a stack does not allow access to variables not at top.

**Conclusion:** Guess a leftmost derivation and match any terminals before the first variable immediately with inputs.

**Sketch Solution:**

1. Place \(S\$\) on stack (\(S = \text{Start Variable}\)).
2. Repeat:
   1. If top stack symbol is a variable, guess rule \(A \rightarrow \text{symbol} \) and substitute \(A \) with \(W\) (note: we can write many symbols).
   2. If top stack symbol is a terminal \(a\), compare it to next input symbol: match; repeat. Else: reject.
   3. If top stack symbol = \$, \(\Rightarrow\) accept.

**DPDAs are less powerful than PDAs, but more powerful than DFAs.**

**Languages accepted by PDAs have unambiguous grammars.** We'll leave the details to a project.

**PDAs are important in, e.g., parsers in compilers.**

**Programming Languages.**
Example: \( S \rightarrow aTb \mid b \)

\( T \rightarrow Ta \mid e \)

\( L = \{ a^n b : n \geq 0 \} \).

1. Add rule \( Apq \rightarrow aPq \) where \( \Delta \) is \( q \) in state after \( p \).
2. Add rule \( Apq \rightarrow Apq \) where \( q \) is state when stack is empty.
3. For each \( pq, r \in Q \), \( a, b \in \Sigma \):
   - For \( S(p, q, e) \in (\tau \eta) \) and \( S(q, b, w) \in (q, e) \)
     - Add rule: \( Apq \rightarrow aApq \).
   - For each \( p, q, r \in Q \):
     - Add rule: \( Apq \rightarrow Apq \).
   - For each \( p, q \in Q \):
     - Add rule: \( Apq \rightarrow e \).

Start variable is \( A_{0, q, 0} \) where \( q_0 \) start of \( p \).
CLAIM 1: IF $A_{pq} \Rightarrow z$ THEN $x$ CAN BE BRING P FROM P TO q WITH EMPTY STACK.

Proof: Induction Basis: 1 Step Derivation $A_{pq} \Rightarrow \epsilon$: $\epsilon$ Takes P to $\epsilon$.

Basis $A_{pq} \Rightarrow z$ with 1st Step $A_{pq} \Rightarrow A_{pq} b$ or $A_{pq} \Rightarrow z A_{pq} q$. 

(1) By IH: y on the tag P from $\epsilon$ on empty Stack.

Since $A_{pq} \Rightarrow A_{pq} b$ & Rule 8, $b(\epsilon, b, \epsilon, c) \Rightarrow (\epsilon, \epsilon)$

So $\delta$ takes P to $\epsilon$ starting with $\epsilon$-Stack, leaving u.

y takes r to s and leave u on the stack.

r takes s to q with stack $\epsilon$, popping u.

$\Rightarrow x$ takes p to q on empty Stack.

CLAIM 2: IF $x$ CAN BE BRING P TO q WITH EMPTY STACKS.

Then $A_{pq}$ generates $x$.

Proof: $\epsilon$-Steps $\Rightarrow x = \epsilon$, have rule $A_{pq} \Rightarrow \epsilon$.

Proof for $k!$:

Assume for $k$: $x$ takes P to q on empty Stack.

Prove for $k! + 1$:

(1) Stack empty only at beginning and end. ($k+1 \geq 2$)

(2) Some where else too:

Then $\delta(s, q, x, \epsilon) \Rightarrow (x, b)$. Write $x$ as $\epsilon = A_{pq} b$.

$\delta(s, p, \epsilon, b) \Rightarrow (\epsilon, c)$

So has rule $A_{pq} \Rightarrow A_{pq} b$.

So y brings r to s without touching $\epsilon$. (Since only at ends empty)

$\Rightarrow y$ brings $\epsilon$ to $\epsilon$ on empty.

By IH: $\epsilon = A_{pq} x$. Hence $A_{pq} x$ is also well.

Let r be an intermediate step where stack is empty. $x = \epsilon$.

On go from $\epsilon$ to $\epsilon$ with $k$ steps. So $A_{pq} b \Rightarrow A_{pq} b$.
Proof: \( G = (V, \Sigma, R, S) \)

\[ b = \max_{A \rightarrow w} (|w|) \quad \text{(in CNF \( b = 2 \)).} \]

Note: A string of length \( \geq b^h \) has all parse trees of height \( \geq h \).

Let \( p = b^{h+1} \), so \( L, 1 \leq p \).

Take a parse tree for \( S \).

It has height \( \geq |v| + 1 \). (Since \( |u| \geq b^{h+1} \))

Choose tree with smallest #nodes.

Take longest path from \( S \) to leaves.

It has length \( \geq |v| + 1 \Rightarrow \geq 1 + |v| + 2 \) nodes.

Leaf is a terminal. \( \Rightarrow |v| + 1 \) variables along it.

\( \Rightarrow \) some variable \( R \) appears twice.

We get condition (1).

For (3): If \( u = v = \epsilon \). \( \Rightarrow uvxy = uvx \) has a smaller parse tree, \( \times \).

For (3): Take \( R \) among the lowest \( |v| + 1 \) on the path. \( \Rightarrow |vxy| \leq b^{h+1} = p \).

example: \( L = \{ a^ib^jc^k : 0 \leq i \leq j \leq k \} \) is not a CFL.

Take \( S = a^ib^jc^k = uvxyz \), \( |vxy| > 1 \), \( |vxy| \leq p \).

\( Vxy \) is short \( \Rightarrow \) misses \( a \)'s or \( c \)'s.

Misses \( a \)'s: \( VY \) has \( A_b \) or \( A_c \). But then \( uXXG \leq L \) has same \( \{ a \}'s \) as \( S \) (ie. \( p \)) but \( < \) \( p \) \( V's \) or \( c \)'s.

Misses \( c \)'s: \( VY \) has \( a \) or \( c \). \( \Rightarrow \) uvxy^2z has more \( c \)'s or \( a \) than \( c \)'s.

Example: PDA to CFG Conversion

\[ L = \{ wuw^R : w \notin \{0,1\}^* \} \]

5 states, 1 accept, empties stack, either pushes or pops.

\( V = \{ A_{ij} : i, j \in \{0, \ldots, 4\} \} \) (25 states)

\( \Sigma = \{0, 1\} \), \( S = A_{00} \)

\( R: \)

- \( A_{00} \rightarrow \epsilon \) \( ; A_{11} \rightarrow \epsilon \) \( ; \ldots \) \( ; A_{44} \rightarrow \epsilon \) (5 rules)

- \( A_{ij} \rightarrow A_{ik}A_{kj} \) \( \forall i, j, k \in \{0, \ldots, 4\} \) (53 = 125 rules)

(eg: \( A_{00} \rightarrow A_{00}A_{00} \) \( \mid A_{00}A_{00} \) \( \mid A_{00}A_{00} \) \( \mid A_{00}A_{00} \) \( \mid A_{00}A_{00} \) \( \mid A_{00}A_{00} \)

\( \cdots \) \( \) \( \) \( \) \( \) \( \)

- \( A_{04} \rightarrow A_{04}A_{04} \) \( \mid A_{04}A_{04} \) \( \mid A_{04}A_{04} \) \( \mid A_{04}A_{04} \)

- \( A_{04} \rightarrow A_{00}A_{00} \) \( \mid A_{00}A_{00} \) \( \mid A_{00}A_{00} \) \( \mid A_{00}A_{00} \)

(4 rules)

(\( \text{Total:} \) 139 rules)
OTHER LANGUAGES:

CONTEXT-SENSITIVE: PRODUCTIONS ARE OF THE FORM

$$\alpha A \gamma \rightarrow \alpha \beta \gamma.$$  
$$\alpha, \gamma \in (ZU^\ast)^*, \beta \in (ZU^\ast)^+; A \in V.$$ 

$$L = \{a^n b^n c^n : n \geq 1\}$$ IS CONTEXT SENSITIVE.

UNRESTRICTED GRAMMARS: PRODUCTIONS:

$$\alpha \rightarrow \beta$$

$$\alpha \beta \in (ZU^\ast)^*, \alpha \neq \varepsilon.$$ (EQUIVALENT TO TURING MACHINES)