Push-Down Automata

PDA class of machines corresponding to CFG's (accepting exactly the context-free languages) — useful in designing parser's based on CFG.

Clearly we must give PDA's unbounded memory to allow it to handle non-regular languages.

However we will restrict its access to the memory!

Set-up

Finite State Control?

Essentially an ε-NFA

Stack notation

Stack notation

Top

Bottom

Contents

ABBAC

ABBAC

ABBAC

Pop returns A; new content BBAC

Push (xyz) new content XYZBBAC

Transitions

Determined by

Effect

Input or ε-move

Current state

Stack top

New state

Pop

Disc new string
Formally, \( PDA \ G = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F) \)

- **States** — \( \{ q_1, q_2, \ldots \} \)
- **Input Alphabet** — \( \{ a, b, c, \ldots \} \)
- **Stack Alphabet** — \( \{ A_1, B_1, C_1, \ldots \} \)
- **Start State** \( (q_0 \in Q) \)
- **Start Stack-Symbol** \( (z_0 \in \Gamma) \)
- **Final States** \( (F \subseteq Q) \)

**Transitions**

\[ \delta(q, \alpha, X) = \{ (q_1, \alpha_1), (q_2, \alpha_2), \ldots \} \]

**Where**

\[ \begin{cases} q_1, q_i \in Q \\ \alpha \in \Sigma \text{ OR } \alpha = \epsilon \\ X \in \Gamma \\ \alpha_i \in \Gamma^* \end{cases} \]

**Action**

First PDA "pops" stack-top to determine \( X \), reads input to determine \( \alpha \) (unless it is an \( \epsilon \)-transition), then knowing \( q, \alpha, X \) it selects non-deterministically one of the possibilities \( (q_i, \alpha_i) \).

**Finally**

State goes from \( q \) to \( q_i \);

Input scans past \( \alpha \) (unless \( \alpha = \epsilon \))

Stack loses old ("top") symbol \( X \) but has \( \alpha_i \) "pushed" onto it.

Note: We need \( z_0 \) initially on stack to allow the first transition to pop the stack.
CONVENTION: \( \text{STRINGS IN } \Sigma^* \ldots \omega, x, y, z \)

\( \text{STRINGS IN } \Pi^* \ldots q, p, y, \ldots \)

**EXAMPLE**

\[ L = \{ 0^n1^n \mid n \geq 1 \} \]

**PDA**

\( \Sigma = \{ 0, 1 \} \quad \Pi = \{ x, z \} \quad Q = \{ q_0, q_1, q_2 \} \quad F = \{ q_2 \} \)

**TRANSITIONS**

\[ \delta(q_0, 0, z_0) = \{ (q_0, xz_0) \} \quad \text{ON INPUT 0, ADD X TO STACK} \]

\[ \delta(q_0, 0, x) = \{ (q_0, xx) \} \]

\[ \delta(q_0, 1, x) = \{ (q_1, \epsilon) \} \quad \text{ON INPUT 1, SWITCH TO q_1 & CONSUME X} \]

\[ \delta(q_1, 1, x) = \{ (q_1, \epsilon) \} \quad \text{ON INPUT 1, KEEP CONSUMING X'S} \]

\[ \delta(q_1, \epsilon, z_0) = \{ (q_2, \epsilon) \} \quad \text{WHEN Z_0 IS EXPOSED, CONSUME IT & MOVE TO FINAL STATE q_2} \]

**TRANSITION DIAGRAM**

\[ \begin{array}{ccc}
0, z_0/xz_0 & \vdash & 1, x/\epsilon \\
\rightarrow & q_0 & \rightarrow & q_1 \\
0, x/xx & \rightarrow & \epsilon, z_0/\epsilon & \rightarrow & q_2
\end{array} \]

**SIMILAR TO**

\( \epsilon \)-NFA WITH STACK'S CHANGES ADDED IN

**REMARKS**

1) **WILL REJECT INPUTS NOT OF THE FORMAT** \( 0^*1^* \)
   **BY NOT HAVING A TRANSITION DEFINED (AT SOME POINT)**

2) **IF TOO FEW 1's**, **WILL NEVER GO TO** \( q_2 \)

3) **IF TOO MANY 1's**, **WILL GET STUCK IN** \( q_2 \)
   **WITHOUT REACHING END OF INPUT.**
INSTANTANEOUS DESCRIPTIONS (ID)

IDEA: SUCINCT NOTATION FOR DESCRIBING THE ENTIRE CONFIGURATION OF A PDA MID-STREAM IN AN EXECUTION ("SNAPSHOT" OR "CORE DUMP").

\[
\text{ID} = <q, x, \alpha>
\]

\[
\begin{align*}
\text{CURRENT STATE} & \quad (q \in Q) \\
\text{UNREAD INPUT} & \quad (x \in \Sigma^*) \\
\text{STACK CONTENTS} & \quad (\alpha \in \Gamma^*)
\end{align*}
\]

TRANSITION IN ID:

SUPPOSE \((p_i, a_i) \in \delta(q, a, x)\)

THEN WE CAN SAY

\[
<q, a \alpha, x \beta> \xrightarrow{*} <p_i, x, a_i \beta>
\]

"GOES TO" IN ONE TRANSITION

IN GENERAL CAN SAY \(\text{ID}_1 \xrightarrow{*} \text{ID}_n\) IF

\(\text{ID}_1 \xrightarrow{*} \text{ID}_2 \xrightarrow{*} \text{ID}_3 \xrightarrow{*} \ldots \xrightarrow{*} \text{ID}_n\).

EXAMPLE:

PREVIOUS PDA ON INPUT 0011

INITIAL ID \(\text{ID}_0 = <q_0, 0011, z_0>\)

EXECUTION

\[
\begin{align*}
<q_0,0011,z_0> & \xrightarrow{*} <q_0,011,xz_0> \xrightarrow{*} <q_0,11,xxz_0> \xrightarrow{*} <q_1,1,1,xz_0> \xrightarrow{*} <q_1,1,1,xz_0> \xrightarrow{*} <q_2,\epsilon,\epsilon>
\end{align*}
\]

\[
\text{THUS} \quad <q_0,0011,z_0> \xrightarrow{*} <q_2,\epsilon,\epsilon>.
\]
Remark: At end the stack was empty, but this cannot happen anywhere in between as the PDA must "pop" before doing a transition — in fact, this is precisely why we initialize stack to $Z_0$. Thus while there can be an $e$-move which ignores the input, same is not true for stack-top.

Acceptance? PDA accepts input $w$ if there is at least one execution trace which leads to a final state when input-end is reached.

Rejection? \( \begin{align*} &\text{IF no transition is possible ("stuck")} \\ &\text{IF input not over, but stack is empty} \\ &\text{IF input over, but in non-final state} \end{align*} \)

Of course this must happen on every execution trace to reject $w$.

Language of PDA $M = (Q, \Sigma, \Gamma, s, q_0, Z_0, F)$

\[
L(M) = \left\{ w \in \Sigma^* \mid \langle q_0, w, Z_0 \rangle \xrightarrow{*} \langle q, \varepsilon, \alpha \rangle \right\}
\]

For $q \in F$, and any $q' \in Q$.

Observe $\xrightarrow{*}$ just says $\langle q_0, w, Z_0 \rangle$ can possibly lead to $\langle q, \varepsilon, \alpha \rangle$, but that is only one of many possible execution traces.

Remark: That is called "final state language" since there is another kind of language defined for PDAs.
EMPTY STACK LANGUAGE \( N(M) \)

\[ N(M) = \{ \omega \in \Sigma^* \mid \langle q_0, \omega, z_0 \rangle \xrightarrow{\ast} \langle p, \epsilon, \epsilon \rangle, \text{ for any } p \} \]

**Idea:** In \( N(M) \) we ignore final states completely and consider the stack being empty as a sign of acceptance — of course, as usual we say that \( \omega \) is accepted if for at least one execution trace, when \( \omega \) is consumed the stack is empty.

**Rejection?** Along every execution trace, one of the following happens:

- Before \( \omega \) is over, PDA gets stuck
- Before \( \omega \) is over, stack is empty
- When \( \omega \) is over, stack is non-empty

**Example**

\[ L = \{ \omega \omega^R \mid \omega \in \{a, b\}^* \} \]

**PDA** \( M \)

\[ Q = \{q_0, q_1\}, F = \emptyset \]

\[ \Sigma = \{a, b\}, \Gamma = \{A, B, z_0\} \]

**Goal:** Accept by empty stack \( (N(M) = L) \)

[Diagram of PDA]
IDEA
1) \( q_0 \) pushes \( w \) onto stack one-by-one
2) Guess mid-point and move to \( q_1 \) (\( \epsilon \)-move)
3) In \( q_1 \), match input with stack one-by-one
4) At end, \( Z_0 \) should be on top so remove it to halt and accept.

**KEY** In step 3, stack pops \( w \) in reverse order!

**Example**
Input = aabbaa

**Execution Trace**

\[
\begin{align*}
&<q_0, aabbaa, Z_0> \\
\text{Guess} &<q_0, abbaa, A Z_0> \\
&<q_0, bbaa, AA Z_0> \\
&<q_0, ba, BAA Z_0> \\
\text{End of } w &<q_1, ba, BAA Z_0> \\
&<q_1, a, AA Z_0> \\
&<q_1, q_1, AZ_0> \\
&<q_1, \epsilon, Z_0> \\
&<q_1, \epsilon, \epsilon> \quad \text{Accept by empty stack}
\end{align*}
\]

**Remark** Since PDA is non-deterministic, other execution traces are possible:

\[
\begin{align*}
&<q_0, aabbaa, Z_0> \quad \text{---} \quad <q_0, abbaa, AZ_0> \\
\text{Bad Guess} &<q_1, abbaa, AZ_0> \\
&<q_1, bbaa, Z_0> \quad \text{---} \quad <q_1, bbaa, \epsilon>
\end{align*}
\]
THEOREM \[ L = L(M_1) \text{ for some PDA } M_1 \iff L = N(M_2) \text{ for some PDA } M_2. \]

**Proof (\(\Rightarrow\))**

**Given** \( M_1 = (\mathcal{Q}, \Sigma, \Gamma, S, q_0, Z_0, F) \)

**Construct** \( M_2 \) such that \( N(M_2) = L(M_1) \).

**Idea** \( M_2 = (\mathcal{Q}_2, \Sigma, \Gamma_2, S_2, q_0, X_0, \emptyset) \)

with \( \mathcal{Q}_2 = \mathcal{Q} \cup \{p_0, q_1\} \)

\( \Gamma_2 = \Gamma \cup \{X_0\} \)

**Diagram**

- **\( s_0 \)**? **Idea**
  1) \text{ Start in } p_0 \text{ with } X_0 \text{ on stack}
  2) \text{ Move to } q_0 \text{ with } Z_0X_0 \text{ on stack}
  3) \text{ Simulate } M_1
  4) \text{ From any final state (of } M_1) \text{ add transition to } q_1 \text{ which will just empty the stack.}

- **\( x_0 \)**? **Prevents "accidental" acceptance by } M_2: \text{ when } M_1 \text{ empties its stack & rejects.}
PROOF ($\Leftarrow$)  

**Given** \( M_2 = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, \emptyset) \)  

**Construct** \( M_1 \) such that \( L(M_1) = N(M_2) \)  

**Idea** \( M_1 = (Q_1, \Sigma, \Gamma_1, \delta_1, p_0, x_0, F_1) \)  

with \( Q_1 = Q \cup \{p_0, \theta\} \)  
\( \Gamma_1 = \Gamma \cup \{x_0\} \)  
\( F_1 = \{\theta\} \)  

---  

\( S_1 \) idea  

1) **Start in** \( p_0 \) with \( x_0 \) in stack  
2) **Move to** \( q_0 \) with \( Z_0x_0 \) on stack  
3) **Simulate** \( M_2 \)  
4) **When** \( M_2 \) empties its stack, it exposes \( x_0 \)  
5) **From all states**, add an \( \epsilon \)-move to \( \theta \) whenever \( x_0 \) is on top of stack.  

---  

**Picture**  

---  

**Remark**  

In simulating CFG/CFL on PDAs, it is much more convenient to use \( N(M) \) than \( L(M) \) — hence the new definition.
EQUIVALENCE OF CFG'S AND PDA'S.

CLAIM

\[ \text{POWER (PDA)} = \text{CFL} \]

\[ \Leftrightarrow \]

\{ 
  \( \cdot \) EVERY CFL IS ACCEPTED BY SOME PDA 
  \( \cdot \) EVERY PDA ACCEPTS SOME CFL. 
\}

THEOREM

\[ L \text{ IS CFL} \implies L = N(M) \text{ FOR SOME PDA M.} \]

PROOF

\[ \text{SUPPOSE } G \text{ IS A CFG FOR } L. \]

\[ \text{OUR GOAL } \text{CONSTRUCT A PDA M SUCH THAT } \]

\[ L(G) = N(M). \]

IDEA

PDA M SIMULATES L.M. DERIVATIONS IN G FOR INPUT \( \omega \) SUCH THAT AT ANY STEP THE DERIVATION SENTENTIAL FORM IS REPRESENTED BY:

a) SEQUENCE OF SYMBOLS "CONSUMED" FROM INPUT \( \omega \) BY M,

b) FOLLOWED BY CONTENTS OF M'S STACK.

\[ \text{THUS } \quad \text{IN } G \quad S \xrightarrow{L,M} abXycz \xrightarrow{\ast} abXycz \]

\[ \text{IN } M \quad \langle q_0, abXycz, S \rangle \]

\[ \xrightarrow{\ast} \langle q_0, \varepsilon, Xycz \rangle \]

\[ \xrightarrow{\ast} \langle q_0, \varepsilon, \varepsilon \rangle \]

NOTE WE USE ACCEPTANCE BY EMPTY STACK.
Formally, given CFG $G = (V, T, P, S)$

Construct PDA $M = (Q, \Sigma, \Pi, S, q_0, Z_0, \emptyset)$

with $Q = \{q\}$, $q_0 = q$

$\Sigma = T$

$\Pi = V \cup T$

$Z_0 = S$

Defining $S$? Two types

1) If terminal $a$ is on stack-top, then expect to see an $a$ in input and consume both —
   Note no change in sentential form

2) If variable $A$ is on stack-top, then replace by R.H.S. of any production rule in $P$ —
   Note no change in input consumed.

Thus 2 types of transitions.

1) $\forall A \in V$
   $s(q, \epsilon, A) = \{(q, x_1), (q, x_2), \ldots, (q, x_k)\}$
   Where $A \rightarrow x_1 \mid x_2 \mid \ldots \mid x_k$ are in $P$.

2) $\forall a \in T$
   $s(q, a, a) = \{(q, \epsilon)\}$.

Example

Consider $G$

$S \rightarrow A S \mid \epsilon$

$A \rightarrow C A 1 / A 1 / 0 1$
PDA

\[ M = \left( \{ q \}, \{ 0, 1 \}, \{ 0, 1, a, s \}, s, q, s, \emptyset \right) \]

\[ S \]

\[ S(q, e, s) = \{ (q, As) \} \]
\[ S(q, e, A) = \{ (q, A1) \} \]
\[ S(q, 0, 1) = \{ (q, e) \} \]
\[ S(q, 1, 1) = \{ (q, e) \} \]

**EXECUTION?**

**CONSIDER** \( w = 011 \)

**IN G**

\[ S \Rightarrow As \Rightarrow A1s \Rightarrow 011s \Rightarrow 011 \]

**IN M**

\[ \langle q, 011, s \rangle \]
\[ \vdash \langle q, 011, As \rangle \]
\[ \vdash \langle q, 011, A1s \rangle \]
\[ \vdash \langle q, 011, 011s \rangle \]
\[ \vdash \langle q, 11, 11s \rangle \]
\[ \vdash \langle q, 1, 1s \rangle \]
\[ \vdash \langle q, e, s \rangle \]
\[ \vdash \langle q, e, e \rangle \]

**ACCEPT**

**OBSERVE** 1-1 CORRESPONDENCE BETWEEN L.M. DERIVATION AND EXECUTION TRACE.

**OF COURSE** THERE ARE MANY OTHER EXECUTION TRACES POSSIBLE — EACH CORRESPONDING TO A DISTINCT DERIVATIONS.

**ASIDE** OBSERVE IF TWO DISTINCT EXECUTIONS ACCEPT \( w \)

\[ \Rightarrow \exists 2 \text{ DISTINCT L.M. DERIVATION} \]

\[ \Rightarrow \text{COMMAND IS AMRIGI00081} \]
NEED TO SHOW \( N(M) = L(G) \)

INTUITION PDA \( M \) just "guesses" each step in \( L(M) \).
DERIVATION OF \( \omega \) AND SIMULATES THE DERIVATION STEP-BY-STEP.

**Lemma 1**

\[ S \xrightarrow{yB} \text{ with } \begin{cases} y \in \Sigma^* \\ B \text{ begins with a variable} \end{cases} \]

THEN IN \( M \), FOR ANY \( x \in \Sigma^* \)

\[ \langle q_0, yx, s \rangle \xrightarrow{\#} \langle q, x, B \rangle \]

**Proof** SIMPLE INDUCTION ON DERIVATION LENGTH (SEE BOOK)

**Lemma 2**

\[ \langle q_0, yx, s \rangle \xrightarrow{\#} \langle q, x, B \rangle \]

THEN IN \( G \), \( S \xrightarrow{yB} \)

**Proof** SIMILAR INDUCTION!

CONSEQUENTLY for any \( y \in L(G) \),

\[ S \xrightarrow{y} \]

which implies (by Lemma 1)

\[ \langle q_0, y, s \rangle \xrightarrow{\#} \langle q, \varepsilon, \varepsilon \rangle \]

Converse follows from Lemma 2.

**Thus**

\[ y \in L(G) \iff y \in N(M) \]

DONE
**Theorem 2** \( L = N(M) \) for PDA \( M \implies L \) is CFL.

**Proof** — **Idea** construct CFG which simulates \( M \)
— **However** since I have never figured out a good way to explain this, I won't even try to cover this in class.

**In Theorem 1** our construction heavily relied on the power of non-determinism to allow the machine to "guess" the correct derivation.

But in real life (or, in parsers/yacc) we don't have non-det. power.

**So can we convert PDAs to deterministic PDAs (DPDA)?**

**Define** DPDA is a PDA with 2 restrictions:

1. \( S(q, a, z) \) has \( \leq 1 \) possibility
2. if \( S(q, e, z) \) is defined, then for all \( a \in \Sigma \)
   \( S(q, a, z) \) is empty.

**Remark** Programming language syntax can be efficiently parsed only if corresponding language has a DPDA.

**Question** is DPDA as powerful as PDA?

**No!** \( L = \{ w w^R \mid w \in \Sigma^* \} \) cannot have a DPDA
(Intuitively because it is impossible to guess where \( w \) ends and \( w^R \) begins)
**DEFN** DCFL — class of languages accepted by DPDAs

**FACT** REGULAR C DCFL C CFL.

**CLAIM** IF L IS A DCFL, THEN IT CANNOT BE INHERENTLY AMBIGUOUS

**WHY?** BECAUSE IN CONVERTING PDAS TO CFGS, DPDAS ALWAYS YIELD AN UNAMBIGUOUS GRAMMAR.

**THUS** IN PROGRAMMING LANGUAGES WE USUALLY TRY TO DEFINE LANGUAGES/GRAMMARS WHICH YIELD DCFLS — WE GET EFFICIENT PARSING AS WELL AS NON-AMBIGUITY (SEE NOTION OF LR-GRAMMARS/PARSING IN BOOKS)

---

**CURIOS FACT**

**PREFIX PROPERTY** LANGUAGE L HAS PREFIX PROPERTY IF

\[ \forall w \in L \] THERE IS NO STRING IN L WHICH HAS W AS A PREFIX.

**CONSIDER** DPDA M

**THEN** N(M) MUST HAVE THE PREFIX PROPERTY

(ONCE IT EMPTIES STACK ON W, IT DIES AND CANNOT ACCEPT ANY EXTENSION OF W).

**BUT** PARSERS TYPICALLY DO ACCEPT BY EMPTY STACK

**THUS** WE ARTIFICIALLY INSERT "END-OF-INPUT" CHARACTERS AND ASK PARSERS TO ACCEPT W$ WHEN WE ARE REALLY INTERESTED IN W ONLY.
Simplifying Grammars

**Goal**
Remove constructs from grammars which slow down parsers — thereby establish a normal form.

**Useful Symbols**
Any \( x \in V \cup T \) such that
\[
S \xrightarrow{\ast} \alpha(xB) \xrightarrow{\ast} w
\]
with \( w \in \Sigma^* \) and \( \alpha, B \in (V \cup T)^* \)

Thus useless symbols are ones which don’t participate in derivations of strings in languages — can safely eliminate without changing the language.

**Defn**
\( x \in V \cup T \) is **generating** if \( x \xrightarrow{\ast} w \), for \( w \in \Sigma^* \)

**Defn**
\( x \in V \cup T \) is **reachable** if \( S \xrightarrow{\ast} \alpha(xB) \), for \( \alpha, B \in (V \cup T)^* \)

Clearly useful \( x \) is both generating & reachable.

**Idea**
Identify useless symbols by removing:

1. **Step 1** Non-generating \( x \)'s
2. **Step 2** Unreachable \( x \)'s

and all their productions.

**Observe** Most do it in this order

**Example**
\[
\begin{align*}
S & \rightarrow AB \\
A & \rightarrow b
\end{align*}
\]

Suppose we do step 2 first — all symbols are reachable, so when we do step 1 next we eliminate \( B \) as being non-gen.

But if we do it in their right order, we first eliminate \( B \) in step 1, and also the production \( S \rightarrow AB \).
NOW IN STEP 2, WE NOW FIND THAT A IS NOT REACHABLE
AND HENCE ALSO ELIMINATE A.

IN GENERAL WE PERFORM BOTH STEPS RECURSIVELY

STEP 1 [ELIMINATE NON-GENERATING SYMBOLS]

BASIS LABEL ALL TERMINALS IN T AS GENERATING.

INDUCTION FOR ALL PRODUCTIONS X → X₁X₂...Xₖ
IF EACH Xᵢ ON RHS IS GENERATING
THEN LABEL X AS GENERATING.

TERMINATE WHEN NO NEW GENERATING SYMBOLS ARE FOUND.

STEP 2 [ELIMINATE NON-REACHABLE SYMBOLS]

BASIS S IS REACHABLE

INDUCTION FOR ALL PRODUCTIONS X → X₁...Xₖ
IF X IS REACHABLE
THEN LABEL X₁, X₂,..., Xₖ AS BEING REACHABLE

EXAMPLE

\[
G_1 \begin{cases}
S \rightarrow AB | AC | CD \\
A \rightarrow BB \\
B \rightarrow AC | ab \\
C \rightarrow Ca | CC \\
D \rightarrow BC | b | d
\end{cases}
\]

STEP 1 GENERATING?

BASE \{a₁,b,d\}
\{a₁,b,d, B, D\}
\{a₁,b,d, B, D, A\}
\{a₁,b,d, B, D, A, C\}

REMOVE C AND ALL THE PRODUCTIONS THAT CONTAIN C ON EITHER SIDE.
STEP 2 REACHABLE?

BASE \{ S \}
\{ S, A, B \}
\{ S, A, B, c, d \}

FINALLY

\[
G_3 \begin{cases}
S \rightarrow A B \\
A \rightarrow B B \\
B \rightarrow a b
\end{cases}
\]

REMARK IF WE DID STEPS 1 & 2 IN REVERSE ORDER

\[ S \rightarrow C D \] (D REACHABLE)
\[ D \rightarrow b \] (D GENERATING)

THUS REMOVING C FIRST HELPED US REMOVE D LATER.

REMOVING E-PRODUCTIONS.

E-PRODUCTION \( A \rightarrow \epsilon \) SLOWS DOWN PARSERS.

DEFN \( x \in \nu \) IS NULLABLE IF \( x \xrightarrow{*} \epsilon \).

IDEA FIND NULLABLE SYMBOLS RECURSIVELY

BASIS IF \( P \) CONTAINS \( A \rightarrow \epsilon \), THEN LABEL A NULLABLE

INDUCTION + PRODUCTIONS \( X \rightarrow X_1 X_2 \ldots X_k \)
IF EACH \( X_i \) IS NULLABLE
THEN LABEL \( X \) NULLABLE

TERMINATE WHEN NO NEW NULLABLE SYMBOLS FOUND.
EXAMPLE

\[ G_1 \]
\[
S \rightarrow ABC \mid BCB \\
A \rightarrow aB \mid \epsilon \\
B \rightarrow CC \mid \epsilon \\
C \rightarrow S \mid \epsilon
\]

FINDING NULLABLE

BASE
\[
\{ C \} \\
\{ B, C \} \\
\{ S, B, C \}
\]

SUPPOSE WE ELIMINATE \( C \rightarrow \epsilon \)

ORIGINALLY WE COULD DERIVE

\[
S \not\Rightarrow BCB, \quad S \not\Rightarrow CB, \quad S \not\Rightarrow BC \\
S \not\Rightarrow BB, \quad S \not\Rightarrow C, \quad S \not\Rightarrow B
\]

BUT NOW WE CAN'T SO WE MUST ADD IN ALL THESE EFFECTS VIA DIRECT PRODUCTIONS

OVERALL ALGORITHM

\begin{enumerate}
\item \underline{IDENTIFY ALL NULLABLE SYMBOLS}
\item \underline{REPLACE ANY PROD.} \( X \rightarrow X_1 X_2 \ldots X_k \) \underline{BY SET OF PRODUCTIONS OF FORM}
\[ X \rightarrow \alpha_1 \alpha_2 \ldots \alpha_k \]
\underline{WHERE}
\begin{enumerate}
\item \( \alpha_i = X_i \) \underline{IF} \( X_i \) NON-NULLABLE
\item \( \alpha_i = X_i \) \underline{OR} \( \epsilon \) \underline{IF} \( X_i \) NULLABLE
\end{enumerate}
\item \underline{REMOVE ALL \( \epsilon \)-PRODUCTIONS.}
\end{enumerate}

IN EXAMPLE

\[
\begin{align*}
S &\rightarrow ABC \mid AB \mid AC \mid A \mid BCB \mid BC \mid CB \mid BB \mid B \mid C \mid \epsilon \\
A &\rightarrow aB \mid \epsilon \\
B &\rightarrow CC \mid C \mid \epsilon \mid \epsilon \\
C &\rightarrow S \mid \epsilon
\end{align*}
\]

FINALLY REMOVE ALL \( \epsilon \)-PRODS.
Glitch Originally $s \rightarrow e$ was possible, but after final step we do lose $e$ from $L(G)$ — this is unavoidable.

UNIT PRODUCTIONS

**DEFN** UNIT PRODUCTIONS $A \rightarrow B$ (slow down parsers).

**Algorithm**

**Step 1** Remove $e$-productions

**Step 2** $\forall x, y \in V$

\[ \text{If } x \xrightarrow{*} y \text{ and } y \rightarrow \alpha \text{ is not unit} \]

Then add $x \rightarrow \alpha$

**Step 3** Eliminate all unit productions

Finding $x \rightarrow* y$?

Since no $e$-productions, $x \rightarrow* y$ only if

$x \Rightarrow y_l \Rightarrow y_2 \Rightarrow \cdots \Rightarrow y_R \Rightarrow y$

with all $y_i$ being distinct.

Thus $r \leq |V|$.

Can use reachability in directed graphs

**Example**

\[
G_1 = \begin{cases} 
S \rightarrow A | B \\
A \rightarrow S_a | a \\
B \rightarrow S | b 
\end{cases}
\]

**Algorithm**

\[ s \rightarrow* a, s \rightarrow* b \\
b \rightarrow* s, b \rightarrow* a \]
\[ \begin{align*}
S & \rightarrow Sa | a | b | S | A | B \\
A & \rightarrow Sa | a \\
B & \rightarrow Sa | a | b | S | A | B 
\end{align*} \]

**REMOVING UNIT PROD.**
\[ \begin{align*}
S & \rightarrow Sa | a | b \\
A & \rightarrow Sa | a \\
B & \rightarrow Sa | a | b 
\end{align*} \]

**OBSERVE** A, B are now useless as not reachable.

**QUESTION** to remove useless \( e \)-prod | unit-prod all together, does order matter?

**OBSERVE**

a) **REMOVING USELESS** only removing stuff, so cannot add \( e \)-prod | unit-prod

b) **REMOVING** \( e \)-prod could add unit-prod

c) **REMOVING** unit-prod

need to remove \( e \)-prod first could create useless \& symbols but not \( e \)-prod.

**Thus** use following order

- \( A \) \( e \)-productions (no \( e \)-prods added)
- \( B \) unit productions (no productions added)
- \( C \) useless symbols (no productions added)
**CHOMSKY NORMAL FORM.**

**DEFN**  CFG G IS IN CHOMSKY NORMAL FORM (CNF) IF ALL ITS PRODUCTIONS ARE OF THE FORM:

* $A \rightarrow a$
* $A \rightarrow XY$  
  WHERE $A, X, Y \in V$
  $a \in T$.

**THEOREM**  GIVEN ANY CFG G1 WITH $E \notin L(G)$ CAN FIND CNF GRAMMAR G2 SUCH THAT $L(G_2) = L(G_1)$.

**CONSTRUCTION**  3 STEP PROCESS

**STEP 1**  ELIMINATE UNIT-PROD & $E$-PROD.

NOW ALL PRODUCTIONS ARE OF THE FORM

* $A \rightarrow a$
* $A \rightarrow X_1 X_2 \ldots X_R$  $(k \geq 2)$
  WITH $X_1, X_2, \ldots, X_R \in V \cup T$.

**STEP 2**  REMOVE "MIXED BODIES"

FOR EACH $a \in T$ ADD NEW VARIABLE $V_a$ AND $V_a \rightarrow a$

IN EACH PROD $A \rightarrow X_1 \ldots X_R$ REPLACE $a$ BY $V_a$

NOW ALL PRODUCTIONS OF THE FORM

$A \rightarrow a$
$A \rightarrow A_1 \ldots A_R$  $(k \geq 2)$
  WITH $A_1, A_2, \ldots, A_R \in V$.

**STEP 3**  "FACTOR" LONG PRODUCTIONS

FOR $A \rightarrow A_1 A_2 \ldots A_R$ WITH $k \geq 3$
  ADD NEW VARIABLES $B_1, B_2, \ldots, B_{k-2}$

REPLACE
BY
A \rightarrow A_1 \ldots A_r
A \rightarrow A_1 B_1
B_1 \rightarrow A_2 B_2
B_2 \rightarrow A_3 B_3
\vdots
B_{r-2} \rightarrow A_{r-1} A_r.

VERIFY
- GET CNF GRAMMAR
- LANGUAGE IS PRESERVED.

EXAMPLE

\begin{align*}
G_1 = \left\{ \\
S & \rightarrow ABB \mid ab \\
A & \rightarrow Ba \mid bq \\
B & \rightarrow aAbB \\
\right. \\
\end{align*}

STEP 2

\begin{align*}
\begin{cases}
V_a \rightarrow a \\
V_b \rightarrow b \\
S & \rightarrow ABB \mid V_a V_b \\
A & \rightarrow BV_a \mid V_b V_q \\
B & \rightarrow V_a A V_b B \\
\end{cases}
\end{align*}

STEP 3

\begin{align*}
\begin{cases}
V_a \rightarrow a \\
V_b \rightarrow b \\
S & \rightarrow AX_1 \mid V_a V_b \\
X_1 & \rightarrow BB \\
A & \rightarrow BV_a \mid V_b V_q \\
B & \rightarrow V_a Y_1 \\
Y_1 & \rightarrow A Y_2 \\
Y_2 & \rightarrow V_b B \\
\end{cases}
\end{align*}