BLG311E Formal Languages and Automata

Pushdown Automata(PDA) and Recognizing Context-free Languages

A.Emre Harmancı Tolga Ovatman Ö.Sinan Saraç

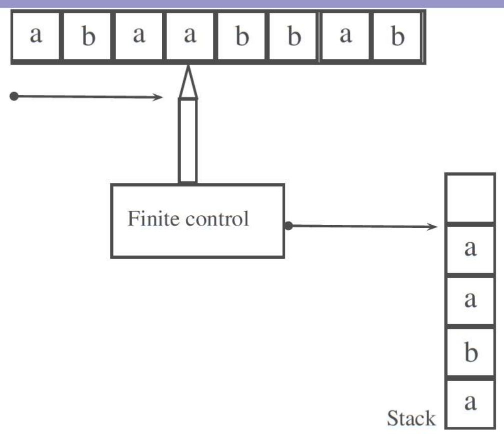
2015

Outline

It is not possible to design finite automata for every context-free language. For instance the recognizer for the language $\omega\omega^R|\omega \in \Sigma^*$ should contain a memory. We can design a pushdown automaton for every context-free language.

Pushdown Automata

A pushdown automaton is similar in some respects to a finite automaton but has an auxiliary memory that operates according to the rules of a stack. The default mode in a pushdown automaton (PDA) is to allow nondeterminism, and unlike the case of finite automata, the nondeterminism cannot always be eliminated.



PDAs are not deterministic. Input strip is only used to read input while the stack can be written and read from.

Formal Definition of a PDA

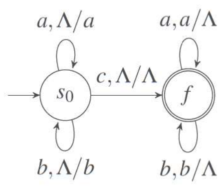
A pushdown automaton (PDA) is a 6-tuple $M = (S, \Sigma, \Gamma, \delta, s_0, F)$, where:

- S: A finite, non-empty set of states where $s \in S$.
- Σ: Input alphabet (a finite, non-empty set of symbols)
- Γ: Stack alphabet
- $s_0 \in S$: An initial state, an element of S.
- δ: The state-transition relation $\delta \subseteq (S \times \Sigma \cup \{\Lambda\} \times \Gamma \cup \{\Lambda\}) \times (S \times \Gamma^*)$
- F: The set of final states where $F \subseteq S$.

An example

$(\omega c \omega^R | \omega \in \{a,b\}^*)$
$M = (S, \Sigma, \Gamma, \delta, s_0, F)$
$S = \{s_0, f\}, \Sigma = \{a,b,c\}, \Gamma = \{a,b\}, F = \{f\}$
$\delta = \{[(s_0,a,\Lambda),(s_0,a)],[(s_0,b,\Lambda),(s_0,b)],[(s_0,c,\Lambda),(f,\Lambda)],$
$[(f,a,a),(f,\Lambda)],[(f,b,b),(f,\Lambda)]\}$

state	tape	stack	trans. rule
s_0	abb **c** bba	Λ	[(s_0,a,Λ),(s_0,a)]
s_0	bb **c** bba	a	[(s_0,b,Λ),(s_0,b)]
s_0	b **c** bba	ba	[(s_0,b,Λ),(s_0,b)]
s_0	**c** bba	bba	[(s_0,c,Λ),(f,Λ)]
f	bba	bba	[(f,b,b),(f,Λ)]
f	ba	ba	[(f,b,b),(f,Λ)]
f	a	a	[(f,a,a),(f,Λ)]
f	Λ	Λ	



An example

state	tape	stack	trans. rule
s	abb **c** bba	Λ	[(s,a,Λ),(s,a)]
s	bb **c** bba	a	[(s,b,Λ),(s,b)]
s	b **c** bba	ba	[(s,b,Λ),(s,b)]
s	**c** bba	bba	[(s,c,Λ),(f,Λ)]
f	bba	bba	[(f,b,b),(f,Λ)]
f	ba	ba	[(f,b,b),(f,Λ)]
f	a	a	[(f,a,a),(f,Λ)]
f	Λ	Λ	

$G = (N, \Sigma, n_0, \mapsto)$
$N = \{S\}$
$\Sigma = \{a, b, c\}$
$n_0 = S$
$< S > ::= a < S > a \mid b < S > b \mid c$

Definitions

Push: To add a symbol to the stack $[(p, u, \Lambda), (q, a)]$

Pop: To remove a symbol from the stack $[(p, u, a), (q, \Lambda)]$

Configuration: An element of $S \times \Sigma^* \times \Gamma^*$. For instance (q, xyz, abc) where a is the top of the stack, c is the bottom of the stack.
Instantaneous description (to yield in one step):

Let $[(p, u, \beta), (q, \gamma)] \in \delta$ and $\forall x \in \Sigma^* \wedge \forall \alpha \in \Gamma^*$

$(p, ux, \beta\alpha) \vdash_M (q, x, \gamma\alpha)$
Here u is read from the input tape and β is read from the stack while γ is written to the stack.

Definitions

$(p, ux, \beta\alpha) \vdash_M (q, x, \gamma\alpha)$

Let $\vdash_M{}^*$ be the reflexive transitive closure of $\vdash_M$ and let $\omega \in \Sigma^*$ and s_0 be the initial state. For M automaton to accept ω string:

$(s, \omega, \Lambda) \vdash_M{}^* (p, \Lambda, \Lambda)$ and $p \in F$
$C_0 = (s, \omega, \Lambda)$ and $C_n = (p, k, \Lambda)$ where
$C_0 \vdash_M C_1 \vdash_M \ldots \vdash_M C_{n-1} \vdash_M C_n$

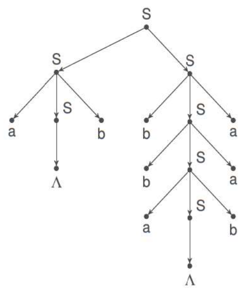
This operation is called *computation* of automaton M, this computation invloves n steps.

Let $L(M)$ be the set of string accepted by M.
$L(M) = \{\omega \mid (s, \omega, \Lambda) \vdash_M{}^* (p, \Lambda, \Lambda) \wedge p \in F\}$

Example 1

$\omega \in \{\{a, b\}^* \mid \#(a) = \#(b)\}$
$M = (S, \Sigma, \Gamma, \delta, s_0, F)$
$\delta = \{[(s, \Lambda, \Lambda), (q, c)], [(q, a, c), (q, ac)], [(q, a, a), (q, aa)],$
$[(q, a, b), (q, \Lambda)], [(q, b, c), (q, bc)], [(q, b, b), (q, bb)],$
$[(q, b, a), (q, \Lambda)], [(q, \Lambda, c), (f, \Lambda)]\}$

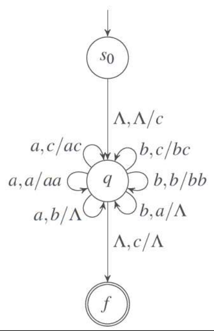
state	tape	stack	trans. rule
s	abbbabaa	Λ	[(s,Λ,Λ),(q,c)]
q	abbbabaa	c	[(q,a,c),(q,ac)]
q	bbbabaa	ac	[(q,b,a),(q,Λ)]
q	bbabaa	c	[(q,b,c),(q,bc)]
q	babaa	bc	[(q,b,b),(q,bb)]
q	abaa	bbc	[(q,a,b),(q,Λ)]
q	baa	bc	[(q,b,b),(q,bb)]
q	aa	bbc	[(q,a,b),(q,Λ)]
q	a	bc	[(q,a,b),(q,Λ)]
q	Λ	c	[(q,Λ,c),(f,Λ)]
f	Λ	Λ	



Example 1

$\omega \in \{\{a, b\}^* \mid \#(a) = \#(b)\}$
$M = (S, \Sigma, \Gamma, \delta, s_0, F)$
$\delta = \{[(s, \Lambda, \Lambda), (q, c)], [(q, a, c), (q, ac)], [(q, a, a), (q, aa)],$
$[(q, a, b), (q, \Lambda)], [(q, b, c), (q, bc)], [(q, b, b), (q, bb)],$
$[(q, b, a), (q, \Lambda)], [(q, \Lambda, c), (f, \Lambda)]\}$

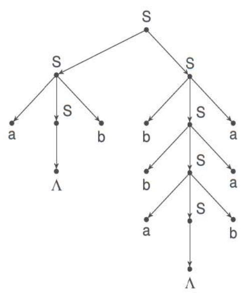
state	tape	stack	trans. rule
s	abbbabaa	Λ	[(s,Λ,Λ),(q,c)]
q	abbbabaa	c	[(q,a,c),(q,ac)]
q	bbbabaa	ac	[(q,b,a),(q,Λ)]
q	bbabaa	c	[(q,b,c),(q,bc)]
q	babaa	bc	[(q,b,b),(q,bb)]
q	abaa	bbc	[(q,a,b),(q,Λ)]
q	baa	bc	[(q,b,b),(q,bb)]
q	aa	bbc	[(q,a,b),(q,Λ)]
q	a	bc	[(q,a,b),(q,Λ)]
q	Λ	c	[(q,Λ,c),(f,Λ)]
f	Λ	Λ	



Example 1

$\omega \in \{\{a, b\}^* \mid \#(a) = \#(b)\}$
$M = (S, \Sigma, \Gamma, \delta, s_0, F)$
$\delta = \{[(s, \Lambda, \Lambda), (q, c)], [(q, a, c), (q, ac)], [(q, a, a), (q, aa)],$
$[(q, a, b), (q, \Lambda)], [(q, b, c), (q, bc)], [(q, b, b), (q, bb)],$
$[(q, b, a), (q, \Lambda)], [(q, \Lambda, c), (f, \Lambda)]\}$

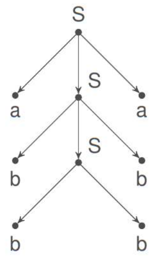
$G = (N, \Sigma, n_0, \mapsto)$
$N = \{S\}$
$\Sigma = \{a, b\}$
$n_0 = S$
$< S > ::= a < S > b \mid b < S > a \mid < S >< S > \mid \Lambda$



Example 2

$\omega \in \{xx^R | x \in \{a,b\}^*\}$
$M = (S, \Sigma, \Gamma, \delta, s_0, F)$
$\delta =$
$\{[(s,a,\Lambda),(s,a)],[(s,b,\Lambda),(s,b)],[(s,\Lambda,\Lambda),(f,\Lambda)],[(f,a,a),(f,\Lambda)],[(f,b,b),(f,\Lambda)]\}$

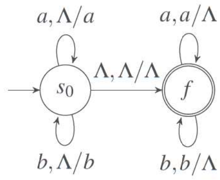
state	tape	stack	trans. rule
s	abbbba	Λ	[(s,a,Λ),(s,a)]
s	bbbba	a	[(s,b,Λ),(s,b)]
s	bbba	ba	[(s,b,Λ),(s,b)]
s	bba	bba	[(s,Λ,Λ),(f,Λ)]
f	bba	bba	[(f,b,b),(f,Λ)]
f	ba	ba	[(f,b,b),(f,Λ)]
f	a	a	[(f,a,a),(f,Λ)]
f	Λ	Λ	



Example 2

$\omega \in \{xx^R | x \in \{a,b\}^*\}$
$M = (S, \Sigma, \Gamma, \delta, s_0, F)$
$\delta =$
$\{[(s,a,\Lambda),(s,a)],[(s,b,\Lambda),(s,b)],[(s,\Lambda,\Lambda),(f,\Lambda)],[(f,a,a),(f,\Lambda)],[(f,b,b),(f,\Lambda)]\}$

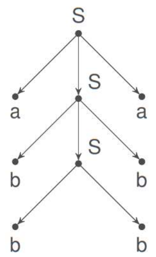
state	tape	stack	trans. rule
s	abbbba	Λ	[(s,a,Λ),(s,a)]
s	bbbba	a	[(s,b,Λ),(s,b)]
s	bbba	ba	[(s,b,Λ),(s,b)]
s	bba	bba	[(s,Λ,Λ),(f,Λ)]
f	bba	bba	[(f,b,b),(f,Λ)]
f	ba	ba	[(f,b,b),(f,Λ)]
f	a	a	[(f,a,a),(f,Λ)]
f	Λ	Λ	



Example 2

$\omega \in \{xx^R | x \in \{a,b\}^*\}$
$M = (S, \Sigma, \Gamma, \delta, s_0, F)$
$\delta =$
$\{[(s,a,\Lambda),(s,a)],[(s,b,\Lambda),(s,b)],[(s,\Lambda,\Lambda),(f,\Lambda)],[(f,a,a),(f,\Lambda)],[(f,b,b),(f,\Lambda)]\}$

$G = (N, \Sigma, n_0, \mapsto)$
$N = \{S\}$
$\Sigma = \{a,b\}$
$< S > ::= a < S > a \mid b < S > b \mid aa \mid bb$



Deterministic PDA

Deterministic PDA

1) $\forall s \in S \wedge \forall \gamma \in \Gamma$ if $\delta(s,\Lambda,\gamma) \neq \varnothing \Rightarrow \delta(s,\sigma,\gamma) = \varnothing; \forall \sigma \in \Sigma$
2) If $a \in \Sigma \cup \{\Lambda\}$ then $\forall s, \forall \gamma$ and $\forall a$ Card$(\delta(s,a,\gamma)) \leq 1$

- (1) If there exists a lambda transition(yielding in one step) in a configuration no other transitions should be present for any other input. (2) There should be a unique transition for any (state,symbol,stack symbol) tuple
- For nondeterministic PDA, the equivalence problem to deterministic PDA is proven to be undecidable[1].
- For instance $\omega\omega^R$ can be accepted by a non-deteministic PDA but there doesn't exist any deterministic PDA that accepts this language.

[1] An undecidable problem is a decision problem for which it is impossible to construct a single algorithm that always leads to a correct yes-or-no answer

Chomsky Hierarchy

Type	Language (Grammars)	Form of Productions in Grammar	Accepting Device
0	Recursively enumerable (unrestricted)	$\alpha \to \beta$ $(\alpha, \beta \in (N \cup \Sigma)^*$, α contains a variable)	Turing machine
1	Context-sensitive	$\alpha \to \beta$ $(\alpha, \beta \in (N \cup \Sigma)^*, \lvert\beta\rvert \geq \lvert\alpha\rvert$, α contains a variable)	Linear-bounded automaton
2	Context-free	$A \to \alpha$ $(A \in N, \alpha \in (N \cup \Sigma)^*)$	Pushdown automaton
3	Regular	$A \to aB, A \to \Lambda$ $(A, B \in N, a \in \Sigma)$	Finite automaton

- Questions about the strings generated by a context-free grammar G are sometimes easier to answer if we know something about the form of the productions.
- Sometimes this means knowing that certain types of productions never occur, and sometimes it means knowing that every production has a certain simple form.
- For example, suppose we want to know whether a string x is generated by G, and we look for an answer by trying all derivations.
  If we don't find a derivation that produces x, how long do we have to keep trying?

Definition

A context-free grammar is said to be in *Chomsky normal form*(CNF) if every production is of one of these two types:

- $A \to BC$ (where B and C are non-terminals)
- $A \to \sigma$ (where σ is a terminal symbol)

Theorem

For every context-free grammar G, there is another CFG G_1 in Chomsky normal form such that $L(G_1) = L(G) - \Lambda$.

CNF Transformation

Following algorithm that can be used to construct the CNF grammar G_1 from a Type-2 grammar G:

1. Eliminate unit productions **and then** Λ productions.
2. Break-down productions whose right side has at least two terminal symbols.
3. Replace each production having more than two non-terminal occurrences on the right by an equivalent set of double-non-terminal productions.

Example CNF Transformation

CNF Transformation

CNF grammar G_1 from a Type-2 grammar G:

A Type-2 Grammar

$S = a, b, c$
$N = S, T, U, V, W$
$\mapsto = \{$
S → TU | V
T → aTb | Λ
U → cU | Λ
V → aVc | W
W → bW | Λ
$\}$
which generates the language $a^i b^j c^k | i = j$ or $i = k$.

CNF Transformation

1. Eliminate unit productions **and then** Λ productions.
   Unit production ex.: $V \to W$
   Λ production ex.: $T \to \Lambda$

A Type-2 Grammar

S → TU | V
T → aTb | Λ
U → cU | Λ
V → aVc | W
W → bW | Λ

Unit productions eliminated

S → TU | aVc | bW | Λ
T → aTb | Λ
U → cU | Λ
V → aVc | bW | Λ
W → bW | Λ

CNF Transformation

1. Eliminate unit productions **and then** Λ productions.
   Unit production ex.: $V \to W$
   Λ production ex.: $T \to \Lambda$

Unit productions eliminated

S → TU | aVc | bW
T → aTb | Λ
U → cU | Λ
V → aVc | bW | Λ
W → bW | Λ

Unit productions eliminated

S → TU | aVc | bW | aTb | ab | cU | c | b | ac
T → aTb | ab
U → cU | c
V → aVc | bW | b | ac
W → bW | b

CNF Transformation

2. Break-down productions whose right side has at least two terminal symbols.
   Introduce for every terminal symbol σ a variable X_σ and a production rule $X_\sigma \to \sigma$

A Type-2 Grammar

S → TU | aTb | aVc | cU
S → ab | ac | c | b | bW
T → aTb | ab
U → cU | c
V → aVc | ac | bW | b
W → bW | b

Non-single-terminals eliminated

S → TU | X_aTX_b | X_aVX_c | X_cU
S → X_a X_b | X_a X_c | c | b | X_bW
T → X_aTX_b | X_a X_b
U → X_cU | c
V → X_aVX_c | X_a X_c | X_bW | b
W → X_bW | b
$X_a \to$ a
$X_b \to$ b
$X_c \to$ c

CNF Transformation

3. Replace each production having more than two non-terminal occurrences on the right by an equivalent set of double-non-terminal productions.

A Type-2 Grammar

S → TU | X_aTX_b | X_cU | X_aVX_c
S → X_a X_b | X_a X_c | c | b | X_bW
T → X_aTX_b | X_a X_b
U → X_cU | c
V → X_aVX_c | X_a X_c | X_bW | b
W → X_bW | b
X_a → a
X_b → b
X_c → c

Non-double-non-terminals elim.

S → TU | X_a Y_1 | X_cU | X_a Y_2
Y_1 → TX_b
Y_2 → VX_c
S → X_a X_b | X_a X_c | X_bW | b | c
T → X_a Y_1 | X_a X_b
U → X_cU | c
V → X_a Y_2 | X_a X_c | X_bW | b
W → X_bW | b
X_a → a
X_b → b
X_c → c

Another Example

1. Eliminate unit productions

A Type-2 Grammar

S → AaA | bA | BaB
A → aaBa | CDA | aa | DC
B → bB | bAB | bb | aS
C → Ca | bC | D
D → bD | Λ

Unit productions partly eliminated

S → AaA | bA | BaB
A → aaBa | CDA | aa | DC
B → bB | bAB | bb | aS
C → Ca | bC | bD | Λ
D → bD | Λ

1. Eliminate Λ productions.

A Type-2 Grammar

S → AaA | bA | BaB
A → aaBa | CDA | aa | DC
B → bB | bAB | bb | aS
C → Ca | bC | bD | Λ
D → bD | Λ

Λ productions partly eliminated

S → AaA | bA | BaB
A → aaBa | CDA | aa | DC | CA
A → DA | C | D | Λ
B → bB | bAB | bb | aS
C → Ca | bC | bD | b | a
D → bD | b

1. Eliminate Λ productions.

A Type-2 Grammar

S → AaA | bA | BaB
A → aaBa | CDA | aa | DC | CA
A → DA | C | D | Λ
B → bB | bAB | bb | aS
C → Ca | bC | bD | b | a
D → bD | b

Λ productions eliminated

S → AaA | bA | BaB | a | b
A → aaBa | CDA | aa | DC | CA
A → DA | C | D | CD
B → bB | bAB | bb | aS
C → Ca | bC | bD | b | a
D → bD | b

1. Eliminate unit productions.

A Type-2 Grammar

S → AaA | bA | BaB | a | b
A → aaBa | CDA | aa | DC | CA
A → DA | CD | C | D
B → bB | bAB | bb | aS
C → Ca | bC | bD | b | a
D → bD | b

Unit productions eliminated

S → AaA | bA | BaB | a | b
A → aaBa | CDA | DC | CA | aa
A → DA | CD | Ca | bC | bD | a | b
B → bB | bAB | aS | bb
C → Ca | bC | bD | a | b
D → bD | b

Transformation continues with the 2nd and 3rd steps.

Consider the following languages. Can you design a PDA to recgonize them.

Example 1

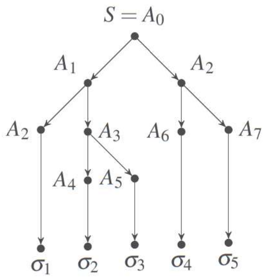
$L(G_1) = \{a^n b^n c^n | n \geq 0\}$

Example 2

$L(G_2) = \{xx \mid x \in \{a,b\}^*\}$

Some languages require more capable memory architectures than a stack to be recognized!

Pumping Lemma for Context-Free Languages



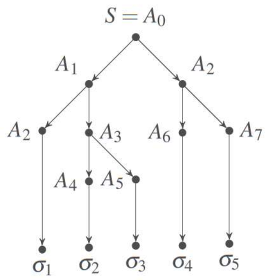
Chomsky Normal Form

The parse tree of a CNF defined context free language is a binary tree. When the leaves of the parse tree is traversed from left to right a word of the language is formed.

Theorem

Gor every context fee grammar G, there is another grammar in CNF such as $L(G_1) = L(G) - \Lambda$. In G_1 grammar rules there exists either a couple of non-terminals or a single terminal.

Pumping Lemma for Context-Free Languages



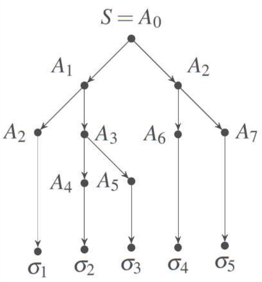
Theorem

Let's assume we produce the string $u = \sigma_1\sigma_2\ldots\sigma_i\ldots\sigma_n \quad (\sigma_i \in \Sigma)$ for a parse tree produced by a CNF grammar such as G_{CNF}.

If we write u using only the non-terminals that directly produce terminals $u = A_1B_2\ldots A_n$

e.g. $S = \sigma_1\sigma_2\sigma_3\sigma_4\sigma_5$ ya da
e.g. $S = A_2A_4A_5A_6A_7$

Pumping Lemma for Context-Free Languages



The parse tree of the grammar can be a complete binary tree for the most extreme case. In such a situation all the paths that navigate to the leaves of this tree are equals to the depth[a] of this tree. In a complete binary tree if the depth is n than the number of leaves is equal to 2^n.

In a CNF grammar's parse tree each leaf is a single child of a non-terminal parent increasing the tree's depth by one ($h = n+1$). On the other hand the length of the produced word is the number of leaves in a complete binary tree of depth n, which is ($|w| = 2^{h-1} = 2^n$).

[a]Tree depth: The longest path from a leaf to the root

Pumping Lemma for Context-Free Languages

In that case we can at most produce a word of length 2^n from a tree of depth $n+1$. In this tree

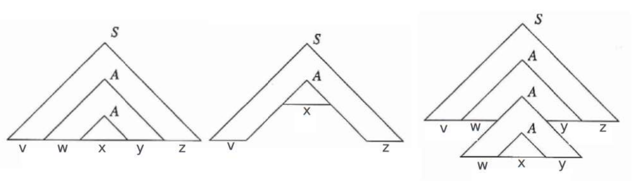
1. If each inner node does not correspond to a different non-terminal, in other words there exists a rule repetition, there exists substrings in the word that can be repeated (pumped).
2. If each inner node corresponds to a different non-terminal then there will be a repetition if it is possible to produce a word of length larger than 2^n.

Theorem

Let L be a context-free language. For all integers n that satisfy $u \in L$ and $|u| \geq n$, string u can be written as $u = vwxyz$ where v, w, x, y, and z satisfies the following:

1. $|wy| > 0$
2. $|wxy| \leq n$
3. For all $m \geq 0$, $vw^mxy^mz \in L$.

Pumping Lemma for Context-Free Languages



Example 1

$$L = \{a^mb^mc^m | m \geq 0\}$$

Assumptions:

- L has a CFG
- $u = vwxyz$ and $|u| \geq n$.
- n corresponds to the number of non-terminals.
- $|wxy| \leq n$

$$\begin{array}{c|c|c|c|c} a^{n-1} & a & b^{n-2} & b & c^n \\ v & w & x & y & z \end{array} \qquad \begin{array}{c} vw^2xy^2z = a^{n+1}b^nc^n \\ vxz = a^{n-1}b^{n-2}c^n \end{array}$$

Example 2

$$L = \{xx \mid x \in \{a,b\}^*\}$$

Assumptions:

- L has a CFG
- $u = a^n b^n a^n b^n$ and $|u| \geq n$
- n corresponds to the number of non-terminals.
- $|wxy| \leq n$

1. $\underset{v}{a^n b^3} \mid \underset{w}{b} \mid \underset{x}{b^{n-4}} \mid \underset{y}{a} \mid \underset{z}{a^{n-1} b^n} \quad \underset{vw^0xy^0z}{a^n b^3 b^{n-4} a^{n-1} b^n}$
2. or $\underset{v}{a^n b} \mid \underset{w}{b} \mid \underset{x}{b^{n-3}} \mid \underset{y}{b} \mid \underset{z}{a^n b^n} \quad \underset{vw^0xy^0z}{a^n b^{n-2} a^n b^n}$
3. or $\underset{v}{a^{n-1}} \mid \underset{w}{a} \mid \underset{x}{b^{n-3}} \mid \underset{y}{b} \mid \underset{z}{b^2 a^n b^n} \quad \underset{vw^0xy^0z}{a^{n-1} b^{n-1} a^n b^n}$

Example 3

$$L = \{x \in \{a,b,c\}^* \mid \#(a) < \#(b) \text{ and } \#(a) < \#(c)\}$$

Assumptions:

- L has a CFG
- $u = a^n b^{n+1} c^{n+1}$ and $|u| \geq n$
- n corresponds to the number of non-terminals.
- $|wxy| \leq n$

1. $\underset{v}{a^{n-1}} \mid \underset{w}{a} \mid \underset{x}{b^{n-3}} \mid \underset{y}{b} \mid \underset{z}{b^3 c^{n+1}} \quad \underset{vw^2xy^2z}{a^{n+1} b^{n+2} c^{n+1}}$
2. ya da $\underset{v}{a^n b^5} \mid \underset{w}{b} \mid \underset{x}{b^{n-5}} \mid \underset{y}{c} \mid \underset{z}{c^n} \quad \underset{vw^0xy^0z}{a^n b^n c^n}$