目 次

オートマトンNOTE

後期期末試験の範囲分

1　言語と文法

1.1　自然言語 (natural language)

▷ 人間の文化の中で, 自然に発展して出来た言語を, **自然言語 (natural language)** という.

自然言語の例

- 日本語
- 英語
- ドイツ語

これらは, 誰かが「あ、日本語作ろう！」とか、「ドイツ語作ろう！」とか思って**意図的に作られたものではない**. どれも, 人間の生活の中で自然に発展し, 言語として確立したものである.

1.2　形式言語 (formal language)

▷ 特定の目的のために, 意図的に作られた言語を, **形式言語 (formal language)** という.

形式言語の例

- プログラミング言語 (C,Java,etc...)

形式言語は, 誰かが「こんな言語があったら便利だよなあ」と思って, **意図的に作った言語**である. まさか人間が生活する上で, 文化的かつ自然に C 言語が発展してきたワケがない. 誰かが作ったのである.

1.3　置き換えルール

▷1950 年代, 米国の言語学者**ノーム・チョムスキー**は, 次のように形式言語を定義する方法を提唱した.

『○○は□□である』

という, **置き換えルール**で形式言語を定義する.

これだけでは意味が分からないので, 実際に例を見てイメージを把握してみよう.

1.4　形式言語として英語を定義する

▷ 英語には, 色々な文がある. 例えば**肯定文, 否定文, 命令文, 疑問文**, などなど · · ·

さらに, これらの文の構造は, 必ずルールとして決まっている. 例として命令文について考えると,

命令文 = 動詞 目的語

例　Open (動詞)　the door. (目的語)　(そのドアを開けなさい)

という構造をしている. この事を, 次のように表そう.

命令文 → 動詞 目的語

(命令文は, 動詞と目的語が並んだものである)

このように,「命令文」というものは,「動詞」「目的語」というものに置き換えられる. これがチョムスキーが提唱した**置き換えルール**である.

さらに考えて行くと, **動詞は英単語に置き換えることができる**. 動詞となり得る英単語は無数にあるが, 例えば open,give,teach,buy などが挙げられる. このことを置き換えルールで表現すると,

動詞 → open | give | teach | buy | ⋯

("|" は,「または」という意味)

とできる. 目的語となり得る単語についても同様に,

目的語 → him | her | the door | it | ⋯

と定義できる. 今までに作ったルールを列挙すると,

- 命令文 → 動詞 目的語
- 動詞 → open | give | teach | buy | ⋯
- 目的語 → him | her | the door | it | ⋯

置き換えルールにおいて, 矢印の左側を**左辺**, 右側を**右辺**という.**左辺は右辺に置き換えることができる**.

実はこれで, **形式言語として命令文を定義**することができた. これらのルールを使って, 実際に命令文を１つ作ってみよう.

命令文 (スタート)
★ 命令文 → 動詞 目的語 より...
動詞 目的語
★ 動詞 → play より...
play 目的語
★ 目的語 → tennis より...
play tennis. ⇒ **完成！**
（テニスをしなさい）

このように, **置き換えルールを次々と適用**することによって, 命令文を完成させることができる. もちろん, 動詞と目的語は,play,tennis 以外の単語に書き換えて OK なので, 色んな命令文を作って「置き換えルール」のイメージを掴んでみてほしい.

1.5 終端記号 (terminal symbol)

▷ もう**それ以上書き換えることができない記号**を, **終端記号 (terminal symbol)** という. 先程の例では, **英単語** (play,tennis) に相当する.

1.6 非終端記号 (non-terminal symbol)

▷ **まだ書き換えることができる記号**を, **非終端記号 (non-terminal symbol)** という. 先程の例では,命令文,動詞,目的語に相当する.

1.7 文の導出

▷ **置き換えルールの左辺を右辺に置き換えることの繰り返し**によって１つの文を得ることを,
文を**導出** (deriving) するという. 先程, play tennis. という命令文を一つ作ったのは, まさに文の導出である. 当然, 導出し終えた文は, **終端記号のみから成る**.

1.8 文の解析

▷ 導出とは逆に, **置き換えルールの右辺を左辺に置き換えることの繰り返し**を, 文を**解析** (analysis) するという. 先程の例『play tennis.』を解析すると,

play tennis. (スタート)
★ 動詞→ play , 目的語→ tennis より...
動詞 目的語.
★ 命令文→動詞 目的語. より...
命令文 ⇒ **解析完了!!**

このように, **右辺を左辺に書き換える**ことの繰り返し（**解析**）によって,『**play tennis.**』**は命令文**であるということが分かったことになる.

1.9 最左導出, 最右導出

▷ 文を導出する際, **どれでも良いからとりあえず適当に置き換えを行おう**という方針だと, 何となくあいまいな感じがする. そこで, 文を導出する際の**明確な方針**として, 次のようなものがある.

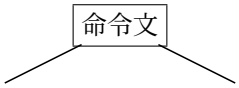
- **最左導出**
  最も左にあるものから順に書き換えを行う.
- **最右導出**
  最も右にあるものから順に書き換えを行う.

1.10 導出木 (deriving tree)

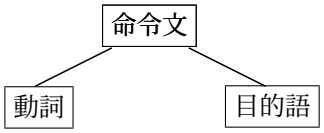
▷ 文の導出の過程を, 見やすく木構造として表現したものを, **導出木 (deriving tree)** という.
導出木を使った文の導出は, 次のように行う.

例　teach math (数学を教えなさい) の導出木.

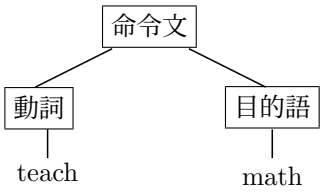
1. 木の**根**に, スタートの非終端記号 (**開始記号**) を書く.



2. その**非終端記号の書き換え先**を, 子とする.



3. **全て終端記号になるまで**繰り返す.



4. 終了！

1.11 形式文法の定義

▷ 形式文法は**置き換えルールと, 非終端記号, 終端記号**により決まることが分かったが, これらはもっと形式的, 数学的に定義できる.

※ 今後は, **置き換えルール**を**生成規則**と呼ぶ.

形式文法は,**4 項組** (4-tuple) により決まる.

- N : **非終端記号の集合**
  まだ置き換えができる記号の集合.

  例　$N = \{$命令文, 動詞, 目的語$\}$

- Σ : **終端記号の集合**
  置き換えができない記号の集合

  例　$\Sigma = \{\text{teach, him, her, tennis, play,} \cdots\}$

- P : **生成規則の集合**
  置き換えルール (生成規則) の集合.

  例　$P = \{$ 命令文 $\to$ 動詞 目的語 ,
  動詞 $\to$ teach $\mid$ play $\mid$ give $\mid \cdots$,
  目的語 $\to$ him $\mid$ her $\mid$ math $\mid \cdots\}$

- S : **開始記号**
  最初に置き換える記号 (導出木の根の部分).

  例　$S =$ 命令文

これら４つが与えられれば, 形式文法が定義されたことになる. これらによって定義される形式文法を, まとめて次のように表現する.

$$G = (N, \Sigma, P, S)$$

これが, 形式文法の数学的な定義である.

2　チョムスキー階層

▷ 形式言語の文法の定義について今まで考えてきた. 形式言語の文法には様々なものが考えられるが, チョムスキーはこれを４つの組に分類し, それぞれに

- 句構造文法 (0 型文法)
- 文脈依存文法 (1 型文法)
- 文脈自由文法 (2 型文法)
- 正則文法 (3 型文法)

と名づけた. この分類を,
チョムスキー階層 (Chomsky hierarchy) という.
チョムスキー階層による形式言語の文法の分類は, **その文法に, どのような形の生成規則が存在するか**によって決められたものである.

2.1 句構造文法 (0 型文法)

▷ 形式言語の文法の中で, 最も制限が少ない
(というか, 何ら制限がない. つまり, ルールが無い. 何でもアリ)
文法のことを, **句構造文法 (0 型文法)** という.0 型文法の生成規則は, 次のような形でなければならない.

$$\alpha \to \beta$$

(α, β は, 終端記号や非終端記号から成る任意の**記号列**)

つまり, 実質**「どんな生成規則があっても良い」**というのが 0 型文法である.

0 型文法は**自由度が高すぎる**ため, 言語として使うのが非常に困難である.
たとえば, 夏休みの自由研究は, 自由すぎて逆にテーマに困るようなものである.

2.2 文脈依存文法 (1 型文法)

▷ 0 型文法よりも制限がある文法, 具体的には, 次のような形の生成規則のみを持つ文法を, **文脈依存文法 (1 型文法,CSG)** という.

$$\alpha A\beta \to \alpha\gamma\beta$$

(α, β, γ は, 終端記号や非終端記号から成る任意の**記号列**, A は非終端記号.)

文脈依存文法という名前に見られる「文脈」とは,α, β のことである.α, β(任意の記号列) によって, 間に挟まれた A が置換できるかどうかを判断するので, **置換できるかどうかが文脈に依存している**といえる.

2.3 文脈自由文法 (2 型文法)

▷1 型文法よりも制限が強い文法, 具体的には次のような形の生成規則のみを持つ文法を, **文脈自由文法 (2 型文法,CFG)** という.

$$A \to \beta$$

(β は, 終端記号や非終端記号から成る任意の**記号列**, A は非終端記号.)

つまり, **左辺が単一の非終端記号のみから成る**生成規則のみを持つ文法を,2 型文法という. 実は, これから最も重要になるのは 2 型文法であり, **プログラミング言語は,2 型文法**で記述されている.

2.4 正則文法 (3 型文法)

▷ チョムスキー階層の中で最も制限が強い文法, 具体的には, 次のような形の生成規則のみを持った文法を**正則文法 (3 型文法,RG)** という.

$$A \to a\ ,\ A \to aB$$

(A, B は非終端記号,a は終端記号)

つまり, **非終端記号を終端記号に置き換える**, または, **非終端記号を, 終端記号の後ろに非終端記号をつけたもの**に置き換える, という生成規則のみを持つ文法を正則文法という. **正則表現** (*.gif とか) は, 正則文法によって記述されている.

2.5 ある文法が何型文法かを判定するには

▷ ある文法が何型文法なのかを判定する際には, **条件の厳しい文法 (3 型)** から順番に, 生成規則が条件に当てはまるかどうかを確かめて行くと良いだろう.

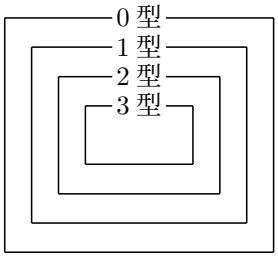
たとえば, ある文法が 3 型文法の条件に当てはまらなかったら, 今度は 2 型文法の条件にあてはまるかどうかを調べ, その次は 1 型……というふうに, **どんどん条件を緩めて行く**とやりやすい.

なぜかというと, 実は**どんな文法も 0 型文法の条件には当てはまる**のである **(0 型文法にはルールが無いから)**. だから, 条件の厳しいものから順に調べて行くのが最も確実で分かりやすい方法なのだ.

2.6 文法同士の関係

▷ 各文法の間には, 次のような関係が成り立つ.

3 型文法 $\subset$ 2 型文法 $\subset$ 1 型文法 $\subset$ 0 型文法

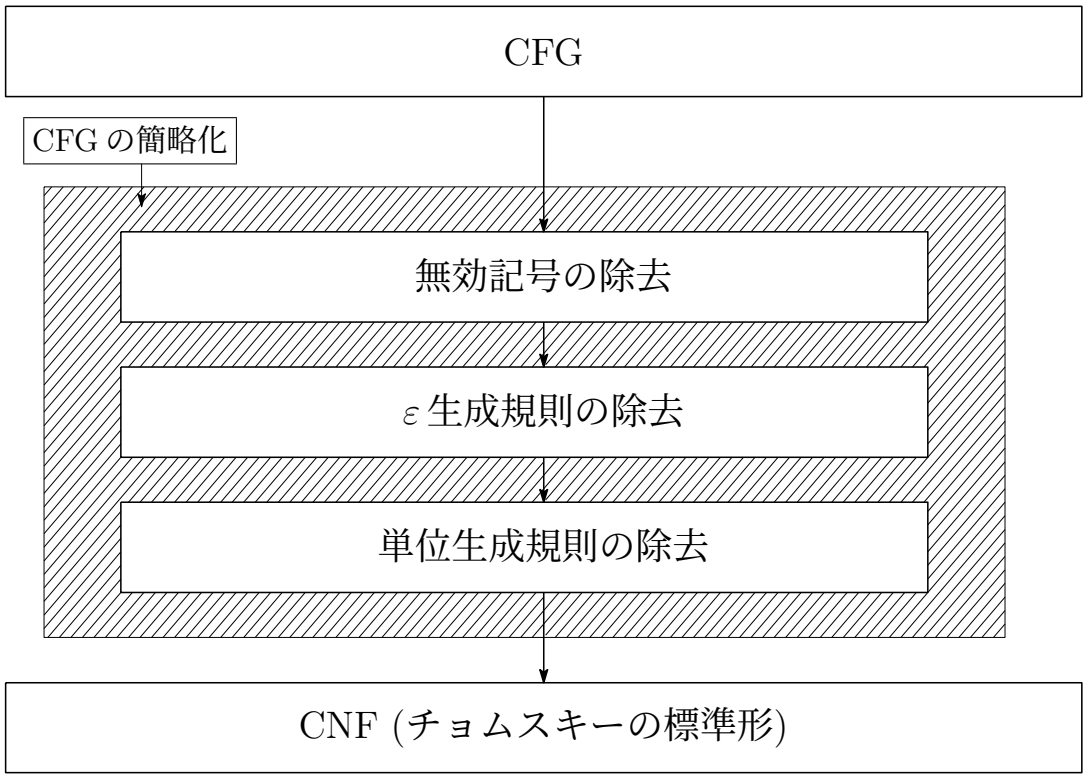


3 CFG → CNF

▷ これから, 主にプログラミング言語の記述に使われる **CFG(2 型文法)** を, 最も使いやすい形, **チョムスキーの標準形 (CNF)** に変形することを目標とする.CFG を CNF に変換するのは一般になかなか骨が折れる作業だが, 全体像を見渡すことさえ出来れば, **ほとんど難しい手順は無い**と思って良いだろう.(要するに, 単にやることが多いというだけなのである)

CFG を CNF に変換するためには, 比較的多数の手順が必要になる. まずは, どのような手順で CFG を CNF に変換できるのかを大まかに展望するため,**CFG→CNF のロードマップ (Road map)** を見てみよう. 変換の大まかな流れを掴んでみてほしい.

3.1 CFG → CNF のロードマップ



このように,CFG はすぐに CNF に変換できるわけではなく,**CFG の簡略化**という手順を踏まなければ, CNF に変換することはできない. さらには, その「簡略化」は, 大まかに分けて 3 つの手順から成り立っている. これを見れば想像が付くとは思うが, **作業量はなかなか多い**ことは覚悟しなければならない. が, **個々の作業はそれほど難しいものではないので, 全く恐れる必要はない**ということも, あらかじめ言っておこう.

ちなみに, なぜこんなに面倒な手順が必要なのかというと, 文法を変換することによって**生成される言語が変わってしまってはいけない**ので, **生成される言語が変わらないように慎重に**変換しなければならないからである. これは俗に言う**等価変換**ってやつである.

このロードマップに示されている手順は,CFG を CNF に等価変換するために, 先人が考え出した**アルゴリズム (algorithm)** である. つまり, この手順に沿って変換を行うだけで,CFG は綺麗に CNF に変換することができる. もちろん, 個々の手順が「何を表しているか」「どうしてこの手順が必要か」について理解しているに越したことはないが, 例えば私たちが Java プログラミングをしながら「どうして System.out.println は動くんだろう？」とかをほとんど考えずに使っているように, このアルゴリズムについても, そんなに深く考えずとも, 変換作業はうまく行くから, あんまり深く考えたりしなくても大丈夫である.

アルゴリズムなんてのは, 結局のところ「凡人が何も考えずに使う」ために作られたものだと思うのだ.

これから CFG を CNF に変換するための具体的な手順の説明に入るが, **自分が今何をやっているのか**が分からなくなったら, いつでもこのロードマップに戻ってきて, **自分の居場所を確認しながら**進むと良いだろう.

4　CFG の簡略化

▷ さっそく,CFG を CNF に変換するための準備として,**CFG の簡略化**を行おう.

4.1　無効記号の除去

▷ 例えば, どうやっても終端記号に書き換えることのできない非終端記号や, 書き換えによってその記号に到達することができない記号など, そのような**不要な記号**を, **無効記号**という. まずは,CFG に含まれている**無効記号を除去**することから始めよう.

4.1.1　生記号の抽出

▷ **生記号**とは, 次の条件を満たす記号のことをいう.

- $A \to \omega$ (A は非終端記号 , ω は**終端記号のみ**から成る記号列) という生成規則があったとき,A を**生記号**という.
- $A \to \alpha$ (A は非終端記号,α は, **生記号と終端記号のみ**から成る記号列) という生成規則があったとき,A を**生記号**という.

生記号の抽出の手順

1. 生成規則の中から, 右辺が終端記号のみから成る規則を探し, 左辺 (**生記号**) を, **生記号リスト**に追加する.
2. さらに生成規則を見て, 右辺が終端記号と生記号のみから成る規則を探し, 左辺 (**生記号**) を, **生記号リスト**に追加する.
3. これを繰り返して, **生記号を全て抽出**し, **抽出されなかった記号 (死記号) と, 死記号を含む生成規則**を CFG から除去する.

現段階ではいまいちイメージが掴みにくいかもしれないが, 後で例題を通して理解できるだろう.

4.1.2　到達可能記号の抽出

▷ $S \to \alpha$ (S は開始記号,α は任意の記号列) という生成規則があったとき,α に含まれる全ての記号を**到達可能記号**という.

到達可能記号に含まれている**非終端記号**を A とすると,$A \to \alpha$ α は任意の記号列という生成規則があったとき,α に含まれる記号を**到達可能記号**という.

到達可能記号の抽出の手順

1. 生成規則 $S \to \alpha$ を見て,α に含まれる記号を全て**到達可能記号リスト**に入れる.
2. **到達可能記号リスト内の非終端記号**を左辺とする生成規則 $A \to \beta$ を見て,β に含まれる記号を全て**到達可能記号リスト**に入れる.
3. これを何度も繰り返し, 到達可能記号を全て抽出. **抽出されなかった記号と, その記号を含む生成規則**を CFG から除去する.

以上で, 第一手順**無効記号の除去**は完了である.

4.1.3　無効記号を除去する上での注意

▷ 無効記号の除去は, 必ず次の手順で行なわなければならない. **(順番が逆転してはいけない)**

1. 生記号の抽出, 除去
2. 到達可能記号の抽出, 除去

4.1.4　例題

▷ 次の CFG の無効記号を除去しよう.

▷ $G = (N, \Sigma, P, S)$
$N = \{S, A, B, C, D\}$
$\Sigma = \{a, b, c, d\}$
$P = \{S \to aAB,$
$A \to aBB,$
$B \to ab,$
$B \to cCD,$
$D \to d\ \}$
$S = S$

▷ まず, 生記号を抽出する.

右辺が**終端記号**のみから成る生成規則を探すと $\cdots$

$$B \to ab, D \to d$$

よって, 左辺 B, D は生記号なので, **生記号リストに追加**する.

生記号リスト

$$B, D$$

次に, 右辺が**生記号リスト内の非終端記号と, 終端記号**のみから成る生成規則を探すと $\cdots$

$$A \to aBB$$

$\therefore$ 左辺 A は生記号なので, **生記号リストに追加**する.

生記号リスト

$$B, D, A$$

同様に, 右辺が**生記号リスト内の非終端記号と, 終端記号**のみから成る生成規則を探すと $\cdots$

$$S \to aAB$$

$\therefore$ 左辺 S は生記号なので**生記号リストに追加**する.

生記号リスト

$$B, D, A, S$$

これ以上同様の手順を繰り返すことができなくなったので, これで生記号の抽出は終了である.

この手順により抽出されなかった非終端記号は C のみである. よって, **非終端記号の集合 N から C を除去し, 生成規則の集合 P から C を含む生成規則を全て除去**すると,CFG は次のように書き換えられる.

$$\begin{aligned} \triangleright\ & G = (N, \Sigma, P, S) \\ & N = \{S, A, B, D\} \\ & \Sigma = \{a, b, d\} \\ & P = \{S \to aAB, \\ & \qquad A \to aBB, \\ & \qquad B \to ab, \\ & \qquad D \to d\ \} \\ & S = S \end{aligned}$$

▷ 次は, 到達可能記号を抽出しよう. 開始記号を含む生成規則に着目すると,

$$S \to aAB$$

よって, 右辺の記号を**到達可能記号リストに追加**する.

※ただし, **開始記号は到達可能記号**であるとする！

到達可能記号リスト

$$a, A, B, S$$

次に, **到達可能記号リストに含まれる非終端記号を左辺とする生成規則**に着目すると,

$$A \to aBB, B \to ab$$

これらの生成規則の右辺は全て到達可能記号なので, **到達可能記号リストに追加**する.

到達可能記号リスト

$$a, b, A, B, S$$

これ以上この手順を繰り返すことができなくなったので, これで到達可能記号リストの更新は完了である. あとは, **抽出されなかった記号 (D, d) と, それらの記号を含む生成規則を除去**すると,

$$\begin{aligned} \triangleright\ & G = (N, \Sigma, P, S) \\ & N = \{S, A, B\} \\ & \Sigma = \{a, b\} \\ & P = \{S \to aAB, \\ & \qquad A \to aBB, \\ & \qquad B \to ab, \\ & S = S \end{aligned}$$

以上で,**CFG から無効記号を除去することができた**.

4.2 ε 生成規則の除去

▷ ε 生成規則とは, 次の形の生成規則のことである.

$$A \to \varepsilon$$

(A は非終端記号, εは, 空列を表す.)

このような生成規則は,CFG から除去しておく必要がある. これから, 早速除去作業を行おう.

4.2.1　空白化記号の抽出

▷ **空白化記号**とは, 次のような条件に当てはまる記号のことをいう.

- $A \to \varepsilon$ (ε 生成規則) の左辺 A(非終端記号) は, **空白化記号**.
- $B \to \alpha$ (B : 非終端記号, α:**空白化記号の列**) の左辺 B は, **空白化記号**.

空白化記号の抽出の手順

1. $A \to \varepsilon$ (ε 生成規則) の形の生成規則を全て探し, 左辺 (非終端記号) をすべて**空白化記号リストに追加**.
2. $B \to A$ (B は非終端記号, A は**空白化記号**) という生成規則を全て探し, 左辺をすべて**空白化記号リストに追加**.
3. ε 生成規則を全て除去する.

ここでは理解できなくとも, 後で例題に触れれば感覚的に理解できるだろう.

4.2.2　規則の追加

▷ この手順が終われば, 第 2 段階「ε 生成規則の除去」は完了である.

規則の追加手順

1. 全ての生成規則 (ε生成規則は除去済) を見て, **右辺に空白化記号 (さっき抽出したやつ) を含まない規則**は, この先**そっとしておく**.
2. **右辺に空白化記号を含む規則**について, **次のような規則を追加**する.
   - **右辺が空列にならない**限り, 右辺から空白化記号を取り除いた**あらゆるパターン**の規則を新たに追加する.
3. もし, 開始記号 S が空白化記号の場合は, **新たな開始記号** S' を導入し,

$$S' \to S\ ,\ S \to \varepsilon \quad \text{とする.}$$

2 番の手順が少々分かりにくいので, 例をあげておく.

4.2.3　”あらゆるパターン”とは

▷ ある CFG において, 空白化記号の集合 (非終端記号) が $\{A, B, C\}$ であるとする. また, このとき, 次のような生成規則があるとする.

$$A \to ABC\ , A \to aBA \quad (a \text{ は終端記号})$$

右辺に着目すると, どちらの規則とも, 空白化記号を含んでいることがわかる.

まず, $\boxed{A \to ABC}$ について, 右辺が空列にならない限り, 空白化記号を取り除いたあらゆるパターンを列挙すると, 次のようになる.

- $A \to BC$ (A を取り除いた)
- $A \to AC$ (B を取り除いた)
- $A \to AB$ (C を取り除いた)
- $A \to C$ (A, B を取り除いた)
- $A \to A$ (B, C を取り除いた)
- $A \to B$ (A, C を取り除いた)

これらの規則を, **全て生成規則に追加すれば良い**.

次に $\boxed{A \to aBA}$ について, 右辺が空列にならない限り空白化記号を除去したパターンを列挙すると,

- $A \to aB$ (A を取り除いた)
- $A \to aA$ (B を取り除いた)
- $A \to a$ (A, B を取り除いた)

となる. これらを**全て生成規則に追加すれば良い**.

4.2.4 例題

▷ 次の CFG の ε生成規則 を除去しよう.

▷ $G = (N, \Sigma, P, S)$
$N = \{S, A, B, C\}$
$\Sigma = \{a, b, c\}$
$P = \{S \to aAC,$
$S \to BC,$
$A \to AA,$
$A \to B,$
$B \to b,$
$B \to C,$
$C \to \varepsilon\}$
$S = S$

まず, 空白化記号を全て抽出することから始めよう. ε 生成規則は $C \to \varepsilon$ のみなので, この左辺 C を空白化記号リストに追加する.

空白化記号リスト

C

次に, 右辺が空白化記号のみから成る生成規則を探すと, $B \to C$ が見つかるので, 左辺 B をリストに追加する.

空白化記号リスト

C, B

更に同様の手順を繰り返すと,$A \to B, S \to BC$ が見つかるので,

空白化記号リスト

C, B, A, S

最後に ε 生成規則を除去すると,

▷ $G = (N, \Sigma, P, S)$
$N = \{S, A, B, C\}$
$\Sigma = \{a, b, c\}$
$P = \{S \to aAC,$
$S \to BC, A \to AA,$
$A \to B, B \to b,$
$B \to C\ \}$
$S = S$

次に, 規則の追加を行う. まず, 右辺に空白化記号 (さっき抽出したやつ) を**含まない**規則

$$B \to b$$

は, この先そっとしておいてあげよう.

その他の, 右辺に空白化記号を含む規則について,**「右辺が空列にならない限り空白化記号を取り除いたあらゆるパターン」**を列挙すると, 次のようになる.

- $S \to aAC$ について
  - $\boxed{S \to aA}$, $\boxed{S \to aC}$, $\boxed{S \to a}$
- $S \to BC$ について
  - $\boxed{S \to B}$, $\boxed{S \to C}$

以上の規則を, 全て CFG に追加する.

また, 開始記号 S が空白化記号なので, 新しい開始記号 S' を導入し, 次のような規則を追加する.

$$S' \to S, S \to \varepsilon$$

以上の手順により, 最終的に CFG は次のような形に等価変換できた.

▷ $G = (N, \Sigma, P, S)$
$N = \{S', S, A, B, C\}$
$\Sigma = \{a, b, c\}$
$P = \{S \to aAC,$
$S \to BC, A \to AA,$
$A \to B, B \to b,$
$B \to C$
ここから下が追加した規則
$S \to aA, S \to aC, S \to a$
$S \to B, S \to C$
$S' \to S, S \to \varepsilon\ \}$
$S = S'$

以上で, 第 2 段階 ε 生成規則の除去は完了である.

4.3 単位生成規則の除去

▷ CFG の簡略化も, いよいよこれで最終段階である. この手順を終えれば, 目標としてきた **CNF** が目前に見えてくる.

▷ **単位生成規則**とは, 次のような形をした生成規則のことである.

$$A \to B \quad (A, B \text{ は非終端記号})$$

つまり, **ある非終端記号を, 別の非終端記号に置き換える**生成規則のことを, 単位生成規則と呼ぶ.

単位生成規則の除去の手順

1. 全ての非終端記号それぞれについて, **単位生成規則のみを使って遷移できる非終端記号の集合** (非終端記号 A に対し, 集合を N_A とする) を求める.
2. 単位生成規則を全て除去する.

ここではまだ理解できずとも, 後に例題に触れれば感覚的に理解できるだろう.

4.3.1 規則の追加

▷ 単位生成規則の除去を完全にするには, 新たな規則を追加する必要がある.

単位生成規則の除去の手順

1. 先程の手順で求めた集合 N_A について, **N_A の要素を左辺として持つ生成規則に対し,A を左辺とする規則を追加する.**

これについては少々分かりにくいので, 例を挙げて説明しておこう.

4.3.2 規則の追加の例

$N_A = \{A, B, C\}$ とし, 生成規則は

$$A \to ab, B \to BC, C \to cAD$$

とする. このとき, これらの生成規則の左辺は,N_A の要素である. このとき, これらの生成規則について, 新しく次のような規則を追加する.

- $B \to BC$ に対して $\boxed{A \to BC}$
  (左辺を $N_{\mathbf{A}}$ の A で置き換えた)
- $C \to cAD$ に対して $\boxed{A \to cAD}$
  (左辺を $N_{\mathbf{A}}$ の A で置き換えた)

ちなみに,$A \to ab$ については, **左辺が既に** A なので, 改めて規則を追加する必要はない.

4.3.3 例題

▷ 次の CFG の単位生成規則を除去しよう.

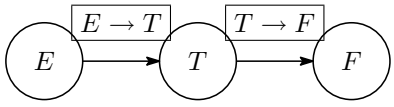
$$\begin{aligned} \triangleright\ G &= (N, \Sigma, P, S) \\ N &= \{E, T, F\} \\ \Sigma &= \{a, +, *, (,)\} \\ P &= \{E \to E + T, \\ &\quad E \to T, \\ &\quad T \to T * F, \\ &\quad T \to F, \\ &\quad F \to (E), \\ &\quad F \to a\} \\ S &= E \end{aligned}$$

まず, 単位生成規則は $T \to F, E \to T$ の 2 つである. 非終端記号は $\{E, T, F\}$ であるので, それぞれについて**単位生成規則のみを使って遷移できる非終端記号**の集合 N_E, N_T, N_F を求める.

- E について
  ▷ $E \to T$ より,T に遷移できる.
  ▷ $T \to F$ より,F に遷移できる.

  $\therefore N_E = \{E, T, F\}$
- T について
  ▷ $T \to F$ より,F に遷移できる.

  $\therefore N_T = \{T, F\}$
- F について
  $\therefore N_F = \{F\}$



この手順を終えた時点で, **単位生成規則は生成規則から全て除去**する.

次に, **規則の追加**に入ろう. まずは N_E の要素を左辺として持つ規則は以下のとおりである.

$$E \to E + T, T \to T * F, \\ F \to (E), F \to a$$

これらの左辺を $E(N_{\underline{E}})$ で置き換えた規則を追加すれば良いので, 追加する規則は

- $E \to T * F$
- $E \to (E)$
- $E \to a$

の 3 つである. 同様の手順を N_T, N_F についても適用すると, 残りの追加する規則は

- $T \to (E)$
- $T \to a$

である. これらを追加した結果,CFG は以下のように等価変換できる.

$$
\begin{aligned}
\triangleright\ & G = (N, \Sigma, P, S) \\
& N = \{E, T, F\} \\
& \Sigma = \{a, +, *, (,)\} \\
& P = \{E \to E + T, T \to T * F, \\
& \qquad F \to (E), F \to a \\
& \qquad \boxed{\text{ここから下が追加した規則}} \\
& \qquad E \to T * F \\
& \qquad E \to (E), E \to a, \\
& \qquad T \to (E), T \to a\} \\
& S = E
\end{aligned}
$$

以上で, 単位生成規則の除去が完了した.

これで,CFG の簡略化についての説明は終了である. つまり,**CFG を CNF に変換するための準備が, いよいよ整った**ということだ.

5 CNF への変換

▷ 3 つの手順を経て, 私たちは CFG(2 型文法) を CNF(チョムスキーの標準形) に変換するための道具立てをしてきた. そして準備が整った今, いよいよ **CFG を CNF に変換**することについて考えよう.

5.1 CNF とは何か

▷ CNF とは, 次のような生成規則のみを持つ CFG のことである.

$$A \to a\ ,\ A \to BC\ ,\ S \to \varepsilon$$

(A, B, C, S は非終端記号,a は終端記号,S は開始記号)

簡略化済みの CFG は, うまく CNF に等価変換することができる.

5.2 大まかな変換手順

簡略化済みという前提の意味

▷ CFG が簡略化済みであるとは,

- 無効記号が除去されている
- ε 生成規則が除去されている
- 単位生成規則が除去されている

ということである.

簡略化済み CFG を CNF に変換する手順は以下の通りである.

CFG → CNF の手順

1. 既に CNF 型になっている生成規則については, そっとしておく.
2. CNF 型になっていない生成規則について, **新たな非終端記号を導入し,CNF の形に直す**.

2 の手順について, これから詳しく説明しよう.

5.3 生成規則の CNF 型への変換

▷ CNF 型でない生成規則については,CNF 型への変形が必要となる. これは例を通したほうが分かりやすいので, 例を挙げて考えてみよう.

例 $B \to aA$

これは,$A \to BC$ という形 (CNF 型) になっていないので, 変形を行わなければならない. 今欲しいのは,$A \to BC$ (**左辺が一つの非終端記号, 右辺が二つの非終端記号**) という形なので, この形を常に意識しながら変形を行おう.

$B \to aA$ という生成規則について, **新たな非終端記号** $< a >$ **を導入**する. そして,
a を $< a >$ で置き換えて $\boxed{B \to < a > A}$ とする.

これは,**CNF 型 (右辺がふたつの非終端記号) の規則**である.

が, **もともとは終端記号**であった a を**非終端記号** $< a >$ に置き換えて, はいおしまいというワケには行かない. そこで, **次のような規則**を追加する.

$$< a > \rightarrow a$$

これにより,CNF 型ではなかった規則 $\boxed{B \rightarrow aA}$ が, 2 つの CNF 型規則

$\boxed{B \rightarrow < a > A}$, $\boxed{< a > \rightarrow a}$ に変形できた.

($< a >$ は, 新たに導入した非終端記号)

変形の基本は, **右辺の長さが 3 の場合も,4 の場合も, それ以上の場合も, これと同じ**である.

変形の手順

1. CNF 型でない規則の右辺を, **先頭と, それ以外**に分け, **新たな非終端記号を導入**し, それに応じた**新たな規則を追加**する.
   (1 つの非終端記号の場合は, そのままで良い.)

   例　$B \rightarrow baB$ を変形すると,

   $$B \rightarrow \underbrace{< b >}_{\text{先頭}} \underbrace{< aB >}_{\text{先頭以外}}$$

   追加する規則：$< b > \rightarrow b$, $< aB > \rightarrow aB$

2. 新たに追加した規則について, 同様のことを繰り返す (右辺の長さが 3 以上の場合, この繰り返し回数が多くなる).

   例　$< aB > \rightarrow \underbrace{< a >}_{\text{先頭}} \underbrace{B}_{\text{先頭以外}}$

   追加する規則：$< a > \rightarrow a$.

この作業により, 簡略化済み CFG は,CNF に変換することができる. 実際に例を通して体験してみよう.

5.4　例題

▷ 次の簡略化済み CFG を,CNF に変換しよう.

$$G = (N, \Sigma, P, S)$$
$$N = \{A, B, S\}\ ,\ \Sigma = \{a, b\}\ ,\ S = S,$$
$$P = \{S \rightarrow AB, A \rightarrow aABB, B \rightarrow BAa,$$
$$A \rightarrow a, B \rightarrow aBaB, B \rightarrow b\}$$

まず,CNF 型になっていない生成規則をさがす.

$$A \rightarrow aABB\ ,\ B \rightarrow BAa\ ,\ B \rightarrow aBaB$$

- $B \rightarrow BAa$
  - $\boxed{B \rightarrow B < Aa >}$　(先頭と先頭以外に分割)
  - $< Aa > \rightarrow Aa$　(新たな規則を追加)
  - $\boxed{< Aa > \rightarrow A < a >}$ (先頭と先頭以外に分割)
  - $\boxed{< a > \rightarrow a}$　(新たな規則を追加)
- $A \rightarrow aABB$
  - $\boxed{A \rightarrow < a >< ABB >}$
    (先頭と先頭以外に分割)
  - $\boxed{< a > \rightarrow a}$, $< ABB > \rightarrow ABB$
    (新たな規則を追加)
  - $\boxed{< ABB > \rightarrow A < BB >}$
    (先頭と先頭以外に分割)
  - $\boxed{< BB > \rightarrow BB}$　(新たな規則を追加)
- $B \rightarrow aBaB$
  - $\boxed{B \rightarrow < a >< BaB >}$
    (先頭と先頭以外に分割)
  - $\boxed{< a > \rightarrow a}$, $< BaB > \rightarrow BaB$
    (新たな規則を追加)
  - $\boxed{< BaB > \rightarrow B < aB >}$
    (先頭と先頭以外に分割)
  - $< aB > \rightarrow aB$　(新たな規則を追加)
  - $\boxed{< aB > \rightarrow < a > B}$ (先頭と先頭以外に分割)

    ($< a > \rightarrow a$ は既に最初に追加してあるので, ここで改めて追加する必要はない)

これで,CNF 型でなかった規則が,**CNF 型に変形**された. あとは, **元の CFG から CNF 型でなかった規則を全て除去**し, かわりに **CNF 型に変形後の規則 (四角で囲ってあるもの)** を全て追加すれば, CFG→CNF はやっとのことで完了である.

6　グライバッハの標準形 (GNF)

▷ CFG を CNF に無事に変換することができた. そこで, 次の目標となるのが, **グライバッハの標準系 (GNF)** である. **任意の CNF は,GNF に変換することができる.** 変換は少々長い手順から成るが, 根気よく練習すれば必ずできるようになる.

6.1　GNF の定義

▷ GNF とは, 以下のような生成規則のみを持つ CFG のことである.

$$A \to aB_1B_2\cdots B_n\ ,\ A \to a\ ,\ S \to \varepsilon$$

(A, B_k, S：非終端記号 , S：開始記号 , a：終端記号)

6.2　変換の道具立て

▷ CNF→ GNF の変換に向けて, **必要な武器を揃えなければならない**. 武器を準備しないと狩りができないように,CNF を GNF に変換する際にも, きちんと準備をしておかなければいけないのである.

変換に必要な武器

- 生成規則の置換
- 左再帰性の除去

6.2.1　生成規則の置換

▷ 生成規則の置換は, とっても簡単である.

$$\boxed{A \to B\gamma}$$

(A, B：非終端記号, γ: 任意の記号列)

$$\boxed{B \to \beta_1, B \to \beta_2, \cdots, B \to \beta_n}$$

(各β_k: 任意の記号列)

という生成規則があるとしよう.$A \to A\gamma$ は GNF の形 (以下 GNF 型と呼ぶ) になっていないので, どうにかして除去したい. そのために用いるのが, **生成規則の置換**というテクニックである.

生成規則の置換

1. $A \to B\gamma$ を, 次のような規則に変換する.
   - $A \to \beta_1\gamma, A \to \beta_2\gamma, \cdots, A \to \beta_n\gamma$.
2. $A \to B\gamma$ を除去.

つまり,$\boldsymbol{A \to B\gamma}$ **の,$\boldsymbol{B}$ の部分を置き換えるだけ**の話である. 簡単なテクニックだが, これまた大いに役に立つのだ. 例題は後に示す.

6.2.2　左再帰性の除去

▷ これも, 生成規則の置換よりは若干面倒だが, 大して難しくない.

$$\boxed{A \to A\gamma}$$

(A, B：非終端記号, γ: 任意の記号列)

このように, 左辺の非終端記号が, 右辺の先頭になってしまっている生成規則は, **左再帰性を持つ**という. このような規則が残っていると都合が悪いので, 除去しておかなければならない.

次の規則について, 左再帰性の除去手順を示す.

$$\boxed{A \to A\gamma}$$

(A：非終端記号, γ: 任意の記号列)

$$\boxed{A \to \beta_1, A \to \beta_2, \cdots, A \to \beta_n}$$

(各β_k: 任意の記号列)

左再帰性の除去

1. $A \to A\gamma$ を, 次のような規則に変換する
   - $A \to \beta_1 Z, A \to \beta_2 Z, \cdots A \to \beta_n Z$.
   - $Z \to \gamma, Z \to \gamma Z$.

   ※ Z は, **新しく導入した非終端記号**である.
2. $A \to A\gamma$ を除去する.

この 2 つのテクニックを使いこなせば,CNF は GNF に変換することができる.

6.2.3 例題

1. 次の生成規則に, **生成規則の置換**を施そう.

$$A \to Bab, B \to a, B \to bAA$$

解答 (太字は, 置き換えた部分)

$$A \to \boldsymbol{a}ab, A \to \boldsymbol{bAA}ab. \quad //$$

2. 次の生成規則の**左再帰性を除去**しよう.

$$A \to Ab, A \to B, A \to abA$$

解答 (太字は, 新たに導入した非終端記号)

$$A \to B\boldsymbol{Z}, A \to abA\boldsymbol{Z}, \boldsymbol{Z} \to b, \boldsymbol{Z} \to b\boldsymbol{Z}. \quad //$$

6.3 CNF → GNF

▷ これで道具は揃ったので, いよいよ CNF→GNF の変換作業を行おう.

変換の流れ

1. 非終端記号に**順序付け**を行う.
   (たとえば, $N = \{S, A, B\}$ なら,
   $S = \boldsymbol{A}_1, A = \boldsymbol{A}_2, B = \boldsymbol{A}_3$ みたいに)
2. 生成規則の中で, $\boxed{A_i \to A_j\gamma\ (i > j)}$ の形のものを探す. (**右辺のほうが番号が小さい！**)
   ※ $i \leq j$ の規則は, 現段階ではそっとしておく.
3. 2. で見つけた生成規則に対して, **生成規則の置換**を行う.
4. 3. で,GNF 型の規則が生まれたら, そっとしておく. 左再帰性を持つ規則が生まれたら, **左再帰性を除去**する.
5. 非終端記号の, **番号の大きい方** ($A_n \to A_1$の順) から, **生成規則の置換**によって GNF 型に変換する.
6. 4. の**左再帰性の除去**で,GNF 型でないものが生まれていたら, **生成規則の置換**で GNF 型に変換する.

これだけ読んでも意味が分かりにくいと思うので, 例で体験してみよう.

6.4 例題

▷ 次の CNF を GNF に変換しよう.

$$N = \{S, A\}, \Sigma = \{1, 2\}, S = S$$
$$P = \{S \to AA, S \to 1, A \to SS, A \to 2\}$$

1. 非終端記号の順序づけ

   ▷ $S = A_1, A = A_2$ とする.

$$N = \{A_1, A_2\}, \Sigma = \{1, 2\}, S = A_1$$
$$P = \{A_1 \to A_2A_2\ ,\ A_1 \to 1\ ,$$
$$A_2 \to A_1A_1\ ,\ A_2 \to 2\}$$

2. $A_i \to A_j\gamma$ の形で, **右辺の先頭のほうが番号が小さいもの**を探す.
   ▷ $A_2 \to A_1A_1$ が見つかった.

3. **生成規則の置換**を行う.
   ▷A_1についての生成規則は,
   $A_1 \to 1, A_1 \to A_2A_2$ なので,

$$\underbrace{A_2 \to 1A_1}_{\text{GNF 型なので, そっとしておく}} \quad , \quad \underbrace{A_2 \to A_2A_2A_1}_{\text{左再帰性を持ってしまっている}}$$

★ この時点で,$A_2 \to A_1A_1$ は除去しておく！

4. **左再帰性を除去**する.
   ▷$A_2 \to A_2A_2A_1$

$$\underbrace{A_2 \to 1A_1Z}_{\text{GNF 型}}\ ,\ \underbrace{A_2 \to 2Z}_{\text{GNF 型}}\ ,$$
$$\underbrace{Z \to A_2A_1}_{\text{GNF 型でない}}\ ,\ \underbrace{Z \to A_2A_1Z}_{\text{GNF 型でない}}$$

★ この時点で,$A_2 \to A_2A_2A_1$ は除去しておく！

5. 非終端記号の**番号の大きい方**から, **生成規則の置換**を行う (GNF 化).
   ▷ もっとも番号が大きい A_2に関して,
   GNF 型でないものを探す. ⇒ ナシ.
   ▷ その次に番号が大きい A_1に関して,
   GNF 型でないものを探す. ⇒ $A_1 \to A_2A_2$.

   ▷ **生成規則を置換**する.
   $A_1 \to 2A_2 \mid 1A_1A_2 \mid 1A_1ZA_2 \mid 2ZA_2$.

★ この時点で,$A_1 \to A_2A_2$ は除去しておく！

6. Z について, **生成規則の置換**を行う (GNF 化).

▷ $Z \to A_2A_1$, $Z \to A_2A_1Z$

▷ $Z \to A_2A_1$について $\cdots$

$Z \to 2A_1 \mid 1A_1A_1 \mid 1A_1ZA_1 \mid 2ZA_1$

▷ $Z \to A_2A_1Z$ について $\cdots$

$Z \to 2A_1Z \mid 1A_1A_1Z \mid 1A_1ZA_1Z \mid 2ZA_1Z$

★ $Z \to A_2A_1, Z \to A_2A_1Z$ は除去！

$N = \{A_1, A_2\}, \Sigma = \{1, 2\}, S = A_1$

$P =$

$\{\ A_1 \to 1 \mid 2A_2 \mid 1A_1A_2 \mid 1A_1ZA_2 \mid 2ZA_2.$

$A_2 \to 2 \mid 1A_1Z \mid 2Z \mid 1A_1.$

$Z \to 2A_1|1A_1ZA_1|2ZA_1|$

$1A_1A_1|2A_1Z|1A_1A_1Z|1A_1ZA_1Z|2ZA_1Z.$

以上で,CNF の GNF 化が完了した. かなり面倒な手順だったと思うが, 何度も練習してマスターしよう.

7 プッシュダウンオートマトン (PDA)

▷ 今までわざわざ面倒な手順を踏んで,CFG を CNF に変換し, そして何とか CNF を GNF に変換するところまで辿りつけた.

実は CNF を GNF に変換したことにはワケがあって, そのワケというのが, **プッシュダウンオートマトン (PDA)** を構成するということなのである.

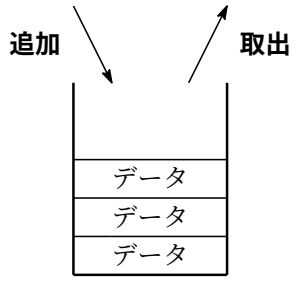
7.1 PDA とは

▷ PDA とは, オートマトンの授業の前半で学んできた**有限オートマトン (FA)** と,2 年の情報工学基礎で学んだ抽象データ型, **スタック (stack)** の夢の共演によって実現できる**新たなオートマトン**である.

7.2 スタック

▷ とは言っても, スタックというものを学んだのがそもそも数年前なので, まずはスタックとは何か？ということについての復習から始めよう.

スタックとは, 基本的な**データ構造**のひとつであり, 現在, 非常に広範囲で用いられている大変重要なものである. スタックは**後入れ先出し (Last in first out / LIFO)** によるデータ構造を持っている. つまり, 後に追加したデータから順番に取り出すことができるということである. スタックは, 分かりやすく次のような図をつかって表現できる.



この例えの他に, たとえば**積み重ねた皿**は, スタック構造を作る.（上にある皿から順番に取り出すから）

7.3 push と pop

- **push**
  スタックにデータを追加すること.
- **pop**
  スタックからデータを取り出すこと.

7.4 スタックトップ

▷ スタックの最も上段にあるデータを, **スタックトップ (stack top)** という. スタックトップは, **一番先に取り出すことができるデータ**のことである.

7.5 PDA の定義

▷ さて, では復習もこの辺にして, 早速 PDA の定義をしてみよう. 今まで学んできた**有限オートマトン**や, **形式文法**は, いくつかの要素の組として表現できた. 例えば $\cdots$

- 例 **DFA（決定性有限オートマトン）**の定義

  $M = (Q, \Sigma, \delta, q_0, F)$ による 5 項組 (5-tuple)

- 例 **形式文法**の定義

  $G = (N, \Sigma, P, S)$ による 4 項組 (4-tuple)

PDA もこれらと同様に, **要素の組**により定義することができる.PDA は **7 項組 (7-tuple)** で定義される.

- Q : **状態の有限集合**
- Γ : **スタック記号の有限集合**
- Σ : **入力記号の有限集合**
- δ : **遷移規則の有限集合**
- q_0 : **初期状態** $(q_0 \in Q)$
- z_0 : **初期スタック** $(z_0 \in \Gamma)$
- F : **最終状態の有限集合** $(F \subseteq Q)$

PDA は, これらによる 7 項組

$M = (Q, \Gamma, \Sigma, \delta, q_0, z_0, F)$ により定義される.

このように, 新たに**スタック**に関する要素が増えていることがわかる. このことについて理解するには, 遷移規則 δ について考えるのが分かりやすい.

7.6 PDA の遷移規則 δ

▷ PDA の遷移規則 (状態遷移関数)δ は, 以下のような形をしている.

$$\delta(p, a, A) \to (q, \alpha)$$

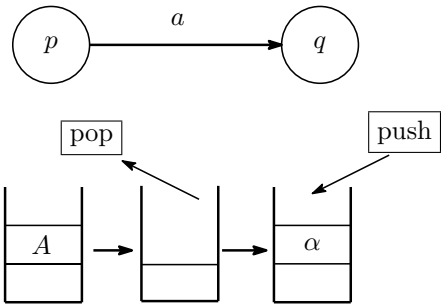
- p : 現状態
- a : 入力記号
- A : スタックトップ
- q : 次状態
- α : スタックへの入力

これだけを見ても, 有限オートマトンに加えて, 新たに**スタックの状態**を考慮に入れなければならないことがわかる. この状態遷移関数を日本語に翻訳してみよう.

$$\delta(p, a, A) \to (q, \alpha)$$

現状態 p, スタックトップ A の PDA に
a を入力したら, 状態 q に遷移して,
スタックトップ A を pop し, α を push する.

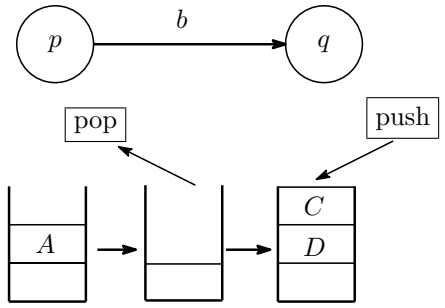
こんな感じである. つまり, 状態が遷移する際には, 同時に push と pop が行われ, スタックの状態も変わるということである.



状態の遷移とともに, スタックの状態も変わる!!

例　$\delta(p, b, A) \to (q, CD)$ はどんな遷移を表すか？

現状態 p , スタックトップ A, 入力 b
$\Rightarrow q$ に遷移し , A を pop , CD を push.



※ あ,CD みたいに**複数の記号**を push するときは, **後ろから順番に push** すると覚えておいてね！

7.7 計算状況

▷ PDA の状態を考える際, 毎回いちいち「この PDA の状態と, スタックの状態が $\cdots$」と言うのは, 結構めんどくさい. このめんどくささをどうにかするために, **新たな用語**を導入しよう.

- **計算状況**
  PDA の状態と, スタックの状態のこと.

例えば**「PDA の状態とスタックの状態を示せ」**と言われるのと, **「PDA の計算状況を示せ」**と呼ばれるのは同じことである.

また, スタックの要素の個数のことを,
計算状況の**高さ (height)** という.

7.8 空スタック受理

▷ PDA の受理には, 特殊なパターンがある. それが**空スタック受理**というパターンである.

PDA が

$$M = (Q, \Gamma, \Sigma, \delta, q_0, z_0, \phi) \quad (F = \phi)$$

のとき,M は**空スタック受理**するという.

空スタック受理する M は, **最終状態を持たない**.
よって,M の受理, 却下に**状態は関係ない**のである.
では, 何によって受理と却下を判定するのかというと, **スタックが空かどうか**によって判定する.

- 入力終了時にスタックが空
  ⇒ 入力を**受理**する.
- 入力終了時にスタックが空でない
  ⇒ 入力を**却下**する.

PDA が空スタック受理かを判断するときには,
***F*=ϕ かどうか**を見れば判断することができる.
***F*=ϕ かどうか**を見れば判断することができる.
大事なことなので,2 回言いましたよ.

7.9 DPDA と NPDA

▷ PDA には, 大きく分けて 2 種類がある.

- **決定性プッシュダウンオートマトン (DPDA)**
  ▷ 遷移先が**常に 1 つ**のみ!
- **非決定性プッシュダウンオートマトン (NPDA)**
  ▷ 遷移先が**いくつあっても**良い!

DPDA の場合は, もちろん, **入力が終わった時点で最終状態に止まっていたら**, その入力が受理される.
が,NPDA は**遷移するパターンがいくつもあり得る (遷移先が 1 つとは限らないから)** ので, 簡単には行かない. そこで,NPDA の受理を次のように定義する.

- NPDA では, **最終状態に止まる遷移方法が 1 つでもあるなら, 入力は受理**と定義

※非決定性有限オートマトン (NFA) と同じ定義!!

7.10 CFG と PDA の等価性

▷ オートマトンの授業の前半で学んだが, **任意の有限オートマトン (FA) は, 等価な正則表現 (RE) に変換することができた**. つまり,

$$FA \equiv RE$$

実は, これと全く同じ関係が, 何と **CFG と PDA の間でも成立**するのである. つまり, **任意の CFG は等価な PDA に変換できるし, その逆も可能なのだ**.

$$CFG \equiv PDA$$

7.11 CFG→ NPDA の変換

▷ では早速,**CFG を NPDA に変換**してみよう. 実はこの変換作業はとっても簡単なので, 全く恐れることはない.

まず, 変換を行う上での**前提**は,**CFG は GNF に変換済み**であることである. (ref : p14〜) GNF に変換済みということは, つまり, 生成規則が以下の形をしたもののみから成るということを表す.

$A \to aB_1B_2B_3 \cdots B_n$
  (A, 各B_k : 非終端記号 , a : 終端記号)
$A \to a$ (a : 終端記号)

また, 次のようなことも念頭に置いておこう.

- 変換後の NPDA が**持つ状態は 1 つだけ**.
- 変換後の NPDA は**空スタック受理**.

では, 早速変換作業を行おう.

変換の流れ

1. まず, 生成規則に注目.
   $A \to aB_1B_2 \cdots B_n$ という生成規則を,
   次のような**状態遷移関数に変換**する.

   $$\delta(q, a, A) = \{(q, B_1B_2 \cdots B_n)\}$$

2. 全ての生成規則についてこの変換を行い,
   得られた状態遷移関数を全て持つ NPDA を構成する.

これだけだとイマイチ分かりにくいと思うので, 例で実際に体験してみよう.

7.12 例題

▷ 次の GNF を等価な NPDA に変換しよう.

$N = \{S, A, B\}, \Sigma = \{a, b, c, d\}, S = S,$
$P = \{S \to a, S \to bAB, A \to c, B \to d\}$

全ての生成規則について, 状態遷移関数をつくる.

▷ $S \to a\ (= a\boldsymbol{\varepsilon})$ について $\cdots$

$$\delta(q, a, S) = \{(q, \boldsymbol{\varepsilon})\}$$

▷ $S \to b\boldsymbol{AB}$ について $\cdots$

$$\delta(q, b, S) = \{(q, \boldsymbol{AB})\}$$

▷ $A \to c\ (= c\boldsymbol{\varepsilon})$ について $\cdots$

$$\delta(q, c, A) = \{(q, \boldsymbol{\varepsilon})\}$$

▷ $B \to d\ (= d\boldsymbol{\varepsilon})$ について $\cdots$

$$\delta(q, d, B) = \{(q, \boldsymbol{\varepsilon})\}$$

非終端記号をスタック記号, **終端記号を入力記号**としてみなし, これらの**状態遷移関数を全て持つ**, **空スタック受理**の NPDA を作ると $\cdots$

$M = (\{q\}, \{S, A, B\}, \{a, b, c, d\}, \delta, q, S, \phi)$

δとして,以下の状態遷移関数を持つ.

$$\delta(q, a, S) = \{(q, \varepsilon)\},$$
$$\delta(q, b, S) = \{(a, AB)\},$$
$$\delta(q, c, A) = \{(q, \varepsilon)\},$$
$$\delta(q, d, B) = \{(q, \varepsilon)\}.$$

$M = (\{q\}, \{S, A, B\}, \{a, b, c, d\}, \delta, q, S, \phi)$ の左の要素から順に, 意味を説明しよう.

- $\{q\}$

  変換後の NPDA は状態が 1 つだけと**約束してあった**ので q という**ただ１つの状態**を作った.

- $\{S, A, B\}$

  GNF の非終端記号を, スタック記号とした.

- $\{a, b, c, d\}$

  GNF の終端記号を, 入力記号とした.

- δ

  生成規則から作った状態遷移関数全てを表す.

- q

  初期状態のこと. 状態は q のみなので, 当然初期状態は q 以外にありえない.

- S

  GNF の開始記号を, 初期スタック状態とした.

- ϕ

  変換後の NPDA は空スタック受理と**約束してあった**ので,$F = \phi$ とした.