第 19 講 文脈自由言語

林　恒俊

文脈自由言語

- CFL は CFG で定義される言語をいい非常に広範囲な対象がこの言語に基づいて実体化されている。
- 一般に入れ子 (nesting) になった構造を備えた情報データは基本的に CFL として取扱うことが可能である。入れ子構造のデータには例えば
  - 自然言語
  - 数式
  - プログラミング言語
  - HTML や XML などのウェブページや文書記述

  などがある。このようなデータ構造を規定し解析し処理するために CFL を基盤とする手段が広く利用されている。
- 正規言語が連結、選択、繰返し操作に基づく対象データの取扱いの基礎であることに対して CFL は入れ子構造を持つ対象データを取扱う基礎をなしている。

文脈自由文法

- CFG は基本的に様々なプログラミング言語の定義に利用されている Backus Nauer form を形式化したものであり計算機科学の基礎として重要な位置を占めている。また Chomsky の主張によると主要な自然言語の文法の骨格は CFG で記述することが可能である。
- CFG では導出は常に 1 個の非終端記号について行われる。常に最も左端の非終端記号から導出を行って得られる導出列を**最左導出** (**leftmost derivation**) という。さらに最も右端の非終端記号から

© 2004–2013 Tsunetoshi Hayashi.

導出を行って得られる導出列を**最右導出** (**rightmost derivation**) という。

- CFG の生成規則は左辺の非終端記号を根とし右辺の記号列を枝とする木構造で表示することができる。そしてこれらの生成規則の木の根と枝を互いに接続した 1 つの木として導出過程を表現することが可能である。この木を**導出木** (**derivation tree**) あるいは**解析木** (**parse tree**) と呼ぶ。
- すべての文に対して最左導出ないし最右導出が 1 通りしかないかまたは解析木が 1 通りしか存在しない文法を**曖昧でない** (**unambigous**) 文法と呼ぶ。そうでない文法を**曖昧な** (**ambigous**) 文法という。ある CFL を定義する文法がすべて曖昧であるときその CFL は曖昧な言語と呼ばれる。

すこし複雑な CFG

- 次の

$$G_s = (\{S\}, \{a, b\}, \{S \to aSb, S \to SS, S \to \Lambda\}, S)$$

で定義される文法 G_s は明らかに CFG である。

$$L(G_s) = \{\Lambda, ab, aabb, abab, ababab, aabbab, \ldots\}$$

G_s が受理する言語は a と b が同数個でしかも ab が入れ子になった記号列から構成される。言換えると a を左括弧、b を右括弧で置換すると例えば (())() のように正しい括弧対になる記号列を要素とする。

- $aaabbabb$ は $L(G_s)$ の要素である。すなわち $S \stackrel{*}{\Rightarrow} aaabbabb$。この要素を生成する導出列は例えば

$$S \Rightarrow aSb \Rightarrow aSSb \Rightarrow aaSbSb \Rightarrow aaaSbbSb \Rightarrow \\ aaabbSb \Rightarrow aaabbaSbb \Rightarrow aaabbabb$$

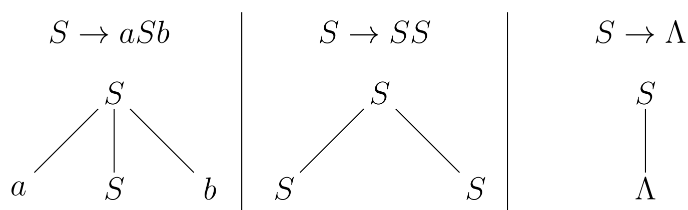
でありこれは最左導出になっている。確かめよ。

- 同一記号列を生成する導出はユニークではない。次の導出列もこの記号列を生成する。

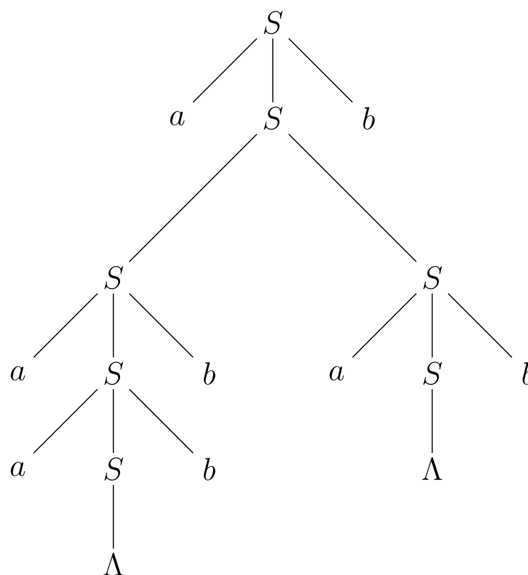
$$S \Rightarrow aSb \Rightarrow aSSb \Rightarrow aSaSbb \Rightarrow aSabb \Rightarrow \\ aaSbabb \Rightarrow aaaSbbabb \Rightarrow aaabbabb$$

  これは最右導出である。確かめよ。

- G_s の生成規則は次のように木構造で表示することができる。



- 記号列 $aaabbabb$ の解析木はユニークで次に示される。



- この言語と同内容の言語が DPDA で定義されることが DPDA の例題ですでに示された。PDA と CFL が深く関連していることが推測される。

実用的な CFG

- ほとんどのプログラミング言語は式で値を表現して計算するようになっている。このような式には数値定数や変数、演算子、括弧や区切り記号などを使うことができる。典型的な式は次のようなものである。

  $$a,\ a+a\times a,\ a\times(a+a),\ a\times a,\ (a),\ \ldots$$

  式の書き方 (構文) はプログラミング言語が厳密に定義している。そして構文定義には普通 BNF のような CFG かそれと類似の表現が利用される。ここではこの式の構文について考察する。

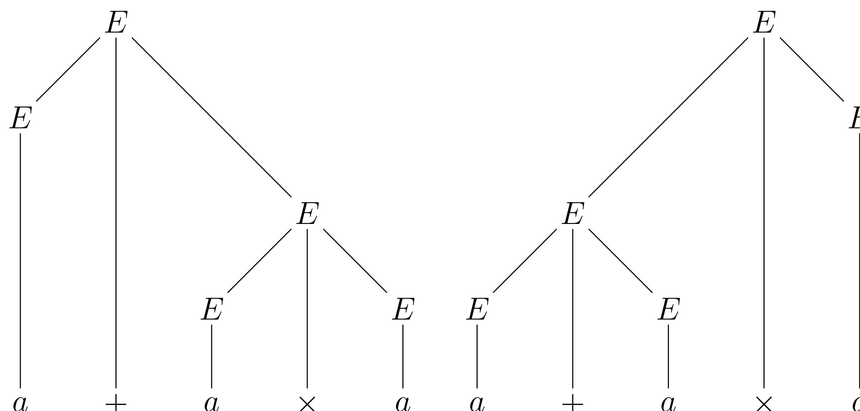
- 文法 $G_a = (\{E\}, \{+, \times, a, (,)\}, P, E)$、ただし

  $$\begin{aligned} P = \{&E \to E+E,\\ &E \to E\times E,\\ &E \to (E),\\ &E \to a\} \end{aligned}$$

  は CFG であり、このような数式風記号列を定義している。例えば導出列

  $$E \Rightarrow E+E \Rightarrow a+E \Rightarrow a+E\times E \Rightarrow a+a\times E \Rightarrow a+a\times a$$

  は $a+a\times a$ という式を生成する。しかしこの文法は曖昧であり次のようにこの式について 2 個の解析木が存在する。



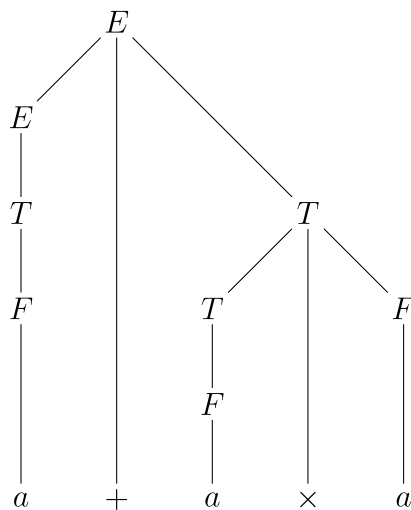
- 次の文法 $G_u = (\{E, T, F\}, \{+, \times, a, (,)\}, P, E)$、ただし

$$\begin{aligned} P = \{&E \to E + T, \\ &E \to T, \\ &T \to T \times F, \\ &T \to F, \\ &F \to (E), \\ &F \to a\} \end{aligned}$$

  は CFG であり、同一の数式風記号列を定義している。$a + a \times a$ を生成する導出は

$$\begin{aligned} &E \Rightarrow E + T \Rightarrow T + T \Rightarrow F + T \Rightarrow a + T \Rightarrow \\ &\quad a + T \times F \Rightarrow a + F \times F \Rightarrow a + a \times F \Rightarrow a + a \times a \end{aligned}$$

  であり、解析木は 1 種類で



  である。この文法は G_a と異なって曖昧ではない。

- プログラミング言語で書かれたプログラムを実行するためにはプログラムを計算機の機械語に変換しなければならない。プログラムは記号列と見なされるのでこれには記号列から逆にその導出過程を求める必要がある。この処理を**構文解析** (**syntax analysis**, **parsing**) という。

- 実用的な言語処理系を設計するためには構文解析を効率的に実現しなければならない。それには文法が曖昧でないことや決定的な構文

解析が可能であることが要求される。プログラミング言語を定義する文法はこれらの点を十分にふまえて構成する必要がある。

- 文法 G_u はこれらの問題を考慮した上で作られたものである。しかし G_u は曖昧ではないが決定的な解析にはすこし問題があることが知られている。

- 次の文法 $G_L = (\{E, E', T, T', F\}, \{+, \times, a, (,)\}, P, E)$、ただし

$$
\begin{aligned}
P = \{E &\to TE', \\
E' &\to +\, TE', \\
E' &\to \Lambda, \\
T &\to FT', \\
T' &\to \times\, FT', \\
T' &\to \Lambda, \\
F &\to (E), \\
F &\to a\}
\end{aligned}
$$

  は CFG であり、同一数式風記号列を定義している。この文法 G_L は曖昧ではなくかつ決定的な解析が可能である。

自然言語の文法

- 次の構文定義は英文の骨格を示している。

$$
\begin{array}{lcl}
\langle \text{sentence} \rangle & \to & \langle \text{subject} \rangle \langle \text{verb phrase} \rangle \\
\langle \text{subject} \rangle & \to & \langle \text{noun phrase} \rangle \\
\langle \text{noun phrase} \rangle & \to & \langle \text{article} \rangle \langle \text{noun} \rangle \\
\langle \text{article} \rangle & \to & \text{a, an, the, } \ldots \\
\langle \text{noun} \rangle & \to & \text{bird, frog, } \ldots \\
\langle \text{verb phrase} \rangle & \to & \langle \text{verb} \rangle \langle \text{adverb} \rangle \\
\langle \text{verb} \rangle & \to & \text{fly, jump, } \ldots \\
\langle \text{adverb} \rangle & \to & \text{quickly, slowly, } \ldots
\end{array}
$$

  $\langle\,\rangle$ に囲まれた記号が非終端記号、それ以外の単語は終端記号を示している。開始記号は $\langle \text{sentence} \rangle$ である。

- この文法から以下のような文が生成される。

a bird fly(ies) slowly
the frog jump(s) quickly

- 英文は語の機能を語順に依存する割合が多いため、このような構文定義には都合がよい。語尾変化が盛んで語順に依存しない言語では構文定義が必ずしもうまくできるとは限らない。しかしこのような文法を確立してあれば、機械翻訳等の技術を発展させる上で一助となるかもしれない。

CFL の性質

- 以前に正規言語は集合和、連結、Kleene の $*$ 演算、補集合、集合積、逆列などの操作に対して閉じていることを説明した。ここでは CFL について正規言語と同様な性質が成立するかどうか考察する。

- なお正規言語では FSA を手段として利用したが、CFL では以下のように CFG を手段として活用する。

- CFL L_1, L_2 を定義する文法 G_1, G_2 次のように定義する。ただし G_1, G_2 は終端記号集合は共有し非終端記号集合には共通部分がないものとする。

$$G_1 = (N_1, \Sigma, P_1, S_1),$$
$$G_2 = (N_2, \Sigma, P_2, S_2)$$

このような仮定で一般性を失うことはない。また S を N_1, N_2 のいずれにも属さない記号とする。

  - 次の文法 $G_\cup$ は明らかに CFG であり

$$G_\cup = (N_1 \cup N_2 \cup \{S\}, \Sigma, P_1 \cup P_2 \cup \{S \to S_1, S \to S_2\}, S)$$

  $G_\cup$ が生成する言語は $L_1 \cup L_2$ である。理由を考えよ。

  - 次の文法 $G_\circ$ は明らかに CFG であり

$$G_\circ = (N_1 \cup N_2 \cup \{S\}, \Sigma, P_1 \cup P_2 \cup \{S \to S_1 S_2\}, S)$$

  $G_\circ$ が生成する言語は $L_1 \circ L_2$ である。理由を考えよ。

- 次の文法 G_* は明らかに CFG であり

$$G_* = (N_1 \cup \{S\}, \Sigma, P_1 \cup \{S \to S_1 S, S \to \Lambda\}, S)$$

G_* が生成する言語は L_1^* である。理由を考えよ。

- 以上の考察により CFL は集合和、連結及び Kleene の $*$ 演算に対して閉じていることが確かめられた。すなわち 2 個の CFL の集合和や連結操作で得られる言語も CFL である。またある CFL の Kleene の $*$ をとったものも CFL である。
- CFL は集合積演算に対して閉じていない。2 個の CFL の集合積が CFL でない場合がある。また補集合演算についても閉じていない。
- これはアルファベット $\{a, b, c\}$ 上のつぎの言語を考察するとよい。

$$L_1 = \{a^m b^m c^n \mid m \geq 0, n \geq 0\}$$
$$L_2 = \{a^m b^n c^n \mid m \geq 0, n \geq 0\}$$

これらの言語は共に CFG で定義可能であるため CFL である。しかしその集合積 $L_1 \cap L_2$ は $\{a^n b^n c^n \mid n \geq 0\}$ でこれは後述のように CFL ではない。

- CFL の逆列演算に関して考察してみよ。

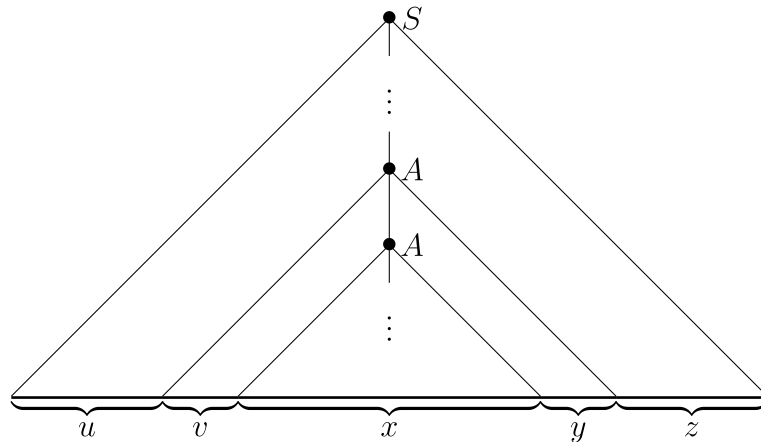
考察課題　CFL の逆列演算に関してはどうか。

CFL のポンピング定理

- 正規言語の場合と同様に CFL についてもポンピング定理が成立する。CFL のポンピング定理は解析木について考察することから得られる。
- CFL L の文を w (ただし $w \in L$) とすると
  - $|w|$ が十分に大きい場合には $w = uvxyz$ と分解できて
  - v か y のいずれかが空列ではなくて
  - $k \geq 0$ の k について $uv^k xy^k z \in L$ とすることができる

すなわち文の 2 カ所を同時に取去ったり繰返したりしても言語の要素になるように文を分解することができる。

- この定理は次のように説明することができる。つぎの解析木において



開始記号 S から文 $w \in \Sigma^*$ まで導出が n 回行われるとし生成規則の右辺の長さの最大を l とすると、w の長さの最大は $|w| = l^n$ である。いま $|w| > l^{|N|}$ になるような w を考えると w の導出に必要な導出回数は $|N|$ すなわち非終端記号の数よりも多くなる。開始記号から w に至るパスの途中にある非終端記号が重複する。重複した非終端記号を A とするとこのパスから A の中間の部分パスを取去っても正しい導出であるし繰返しても正しい導出である。この部分パスから生成される記号列を v と y に割当てればよい。

- この定理を利用すると言語 $L = \{a^n b^n c^n \mid n \geq 0\}$ が CFL でないことを証明することができる。手法は基本的に正規言語の場合と同様である。詳細は省略する。

CFL と PDA

- 正規言語と FSA が同一言語クラスを定義しているのと同様に CFL を定義する抽象機械は NPDA である。これを証明するためにはやはり FSA の場合と同様に

    1. 与えられた CFL を受理する NPDA を構成する

2. 与えられた NPDA が受理する言語の CFG を構成する

の 2 点が説明できればよい。

- 前半の CFL を受理する NPDA はその CFL を定義する CFG の構文解析器を構成すればよい。実際には非決定的 LL(1) 構文解析器で実現可能である。詳細は省略する。
- 後半の NPDA から文法を構成するためには NPDA をシミュレートする CFG を構成する。詳細は省略する。

自習課題　上記証明について調査し正当性を確かめよ。