# 文脈自由文法の拡張と それに基づく計算モデル

山下義行
日立マイクロコンピュータ・エンジニアリング(株)

中田育男
筑波大学電子情報工学系

## 1．はじめに

本報告では、文脈自由な生成規則を用いた形式文法とそれに基づく計算モデルについて報告する。

いわゆる計算理論としては、Turing Machine(TM)、帰納的関数、λ算法、組合せ子、述語論理、形式言語などがある。それぞれの計算理論の形式的な表現能力は等価であるから、たとえばTMで表現できることは必ず他の計算理論でも表現でき、また逆も成り立つ。

そこでこれらの計算理論に基づく計算モデルについて考えると、形式言語理論以外の計算理論ではそれぞれ下のような計算モデルが知られており、さらにはその計算モデルに基づく実際のプログラミング言語が開発されている。

Turing Machine⇒手続き型計算モデル
帰納的関数、λ算法、組合せ子⇒関数型計算モデル
述語論理⇒論理型計算モデル
形式言語⇒ ？

これに対し『形式文法を定義することが即プログラミング』であるような計算モデルはほとんど知られていない。これにはふたつの理由が考えられる。（注：ここでいう形式言語理論とは生成規則による書き換えで言語を生成するような理論であり、代数的言語[1]等は対象としない。）

最初の理由は、たとえば句構造文法では文脈依存な生成規則を許すが、これが人にとって理解しづらいという点である。

たとえば次の生成規則：

$$S \rightarrow a\,S\,A\,B$$
$$S \rightarrow a\,b\,B$$
$$B\,A \rightarrow A\,B$$
$$b\,A \rightarrow b\,b$$
$$b\,B \rightarrow b\,c$$
$$c\,B \rightarrow c\,c$$

を見て、この規則から生成される言語が

$$\{a^n b^n c^n \mid n \geq 1\}$$

であるとはすぐにはわかりづらい。まして逆に$a^n b^n c^n$を生成する書き換え規則を新たに考え出すことは非常に困難である。左辺に複数の記号を許す文脈依存な規則を使うとなんでも表現できるかわりに規則の適用が思わぬ副作用を招くことがあり、少なくとも句構造文法をプログラムと考えるには問題がある。

そこで文脈依存な規則は用いず、文脈自由な規則だけで何でもできるように形式文法を定義したいと考えるのが自然である。たとえばMatrix Grammar, Programmed Grammar[2][3]などはそのような試みであるが、これらは次の理由から必ずしも満足のいくものではない。

すなわちもうひとつの理由は、形式文法を既に我々が慣れ親しんでいるプログラミングに関する概念、たとえばデータ構造、アルゴリズム、手続き等、としてうまく解釈できていないという点である。文脈自由文法をデータ構造の記述に使うことはBNF記法、属性文法[4]など以前から行なわれているが、文脈自由文法による表現には制限があり、それだけで閉じた計算モデルとはなりえない。現状ではプログラムの一部を記述しているに過ぎない。これに対し論理型計算モデルは述語の手続き的解釈を通してHorn節がプログラムとみなせることを明らかにした。論理型計算モデルは任意の帰納的関数を表現できる[5]から、その点でも問題はない。

以上の議論から、形式言語を計算モデルとみなすには次の要求がでてくる。

(1) 文法：Gの各々の生成規則は文脈自由である。
(2) 言語：L(G)のクラスは句構造言語と等価である。
(3) 文法：Gはプログラムとして解釈できる。

我々はこの要求を次のようにして解決した。

(1),(2)に対しては、Matrix Grammar[2]が行っているような生成規則の適用方法にある制限を加えることで言語のクラスを拡張するという方法を採用し、Coupled Context Free Grammar(CCFG)とそのサブセットであるSingle tree-Coupled Context Free Grammar(S-CCFG)というものを新たに定義した。

(3)に対しては、S-CCFGをプログラムのデータ構造と解釈し、複数のS-CCFGをCouplingしたCCFGをデータ間の関係を記述したプログラムと解釈する方法を与えた。

このようにして得られた形式文法：CCFGに基づく計算モデルは、新しいプログラミング・パラダイムを提供すると思われる。

まず2章ではCCFGとS-CCFGを定義し、幾つかの性質を述べる。そして3章ではCCFGに基づく計算モデルについて構文、表示的意味、操作的意味を与え、幾つかの性質を述べる。

## 2．形式言語

この章では文脈自由な生成規則のみを用いた形式文法：Coupled Context Free Grammar(ＣＣＦＧ)とそのサブセットであるSingle tree-Coupled Context Free Grammar（S-CCFG）およびそれらが生成する言語を定義する。この文法は３章で述べる計算モデルの基礎となる。

まず2.1節でS-CCFGを定義する。そして続いて2.2節でS-CCFGを拡張してCCFGを定義し、最後に2.3節でその性質について検討する。

### 2．1．Single tree-Coupled Context Free Grammar

S-CCFGは以下のように定義される。

【定義１】 S-CCFGは４項組：G＝(N,T,P,S) で定義される。ここに、Nは非終端記号の有限集合、Tは終端記号の有限集合、S∈Nは開始記号である。Pは生成規則：

$$N \rightarrow V^{*} \qquad V = T \cup N$$

の有限集合（以下これを『規則集合』と呼ぶ）の有限集合である。□

【例１】 次の４項組：G＝(N,T,P,S) はS-CCFGである。

N={S,A,B,C}
T={a,b,c}
P={{S→ABC},
{A→a, B→b, C→c},
{A→aA,B→bB,C→cC}}

この文法による言語の生成は例２で述べる。□

S-CCFGの文形式、書き換え規則は次のように定義される。

【定義２】 S-CCFG：G＝(N,T,P,S)の文形式は次のように定義される。

(1) 開始記号：Sは文形式である。
(2) αが文形式であり、αが規則集合：p∈Pによってβに書き換えられるならば、βは文形式である。

特に終端記号のみからなる文形式を『文』という。□

【定義３】 S-CCFG：G＝(N,T,P,S)において、規則集合：p∈Pによる文形式：αの書き換えは次のように定義される。

pに含まれる全ての生成規則の左辺の非終端記号の集合を$N_p$とし、αに含まれる全ての非終端記号の集合を$N_\alpha$とするとき、この集合間に

$$N_p \subseteq N_\alpha$$

の関係が成り立つならば、pに含まれる生成規則：

$$X_i \rightarrow \omega_i \qquad i=1 \sim |p|$$

を用いてαの中の全ての$X_i$を$\omega_i$へ同時に書き換えることができる。(さもなくば書き換えることはできない。)ただし

(1) αの中に複数の同じ非終端記号：$X_i$が存在するならば、それら全てを$\omega_i$へ書き換える。
(2) pの中に左辺が同じ非終端記号：Yである複数の生成規則：

$$Y \rightarrow \omega_i, Y \rightarrow \omega_j, \cdots, Y \rightarrow \omega_k \quad i,j,k=1 \sim |p|$$

が存在するならば、αをこれら各々の生成規則で書き換えたものをそれぞれ$\beta_i, \beta_j, \cdots, \beta_k$とし、それらをメタ記号：『≈』で結んだもの：

$$\beta_i \approx \beta_j \approx \cdots \approx \beta_k$$

を文形式とする。

□

定義３(1)(2)はわかりづらいので、例を後ほど例３、４に示す。

定義３に従って文形式：αを文形式：βへ書き換えることを

$$\alpha \Rightarrow \beta$$

と表す。また⇒の反射的かつ推移的閉包を$\overset{*}{\Rightarrow}$で表す。

次にS-CCFG：Gの生成する言語：L(G)を定義する。L(G)はSingle tree-Coupled Context Free Language(S-CCFL)であるという。

【定義４】 S-CCFG：G＝(N,T,P,S)の生成する言語：L(G)は次のように定義される。

$$L(G)=\{\alpha \mid S \overset{*}{\Rightarrow} \alpha \approx \alpha \approx \cdots \approx \alpha,\ \alpha \in T^{*}\}$$

□

ここで、「$\alpha \approx \alpha \approx \cdots \approx \alpha$」は開始記号：Sから導出される文：

$$\alpha_1 \approx \alpha_2 \approx \cdots \approx \alpha_n \quad n \geqq 1$$

の中で≈で結ばれた全ての終端記号列：$\alpha_1, \alpha_2, \cdots, \alpha_n$が互いに等しい、すなわち

$$\alpha_1=\alpha_2=\cdots=\alpha_n$$

であるような文を意味する。
言語の生成例をいくつか述べる。

【例2】 例1の文法について考える。
まず文形式：Sを規則集合：p＝{S→ABC} で書き換えることを考える。このとき、

$$N_p=\{S\}\subseteq N_\alpha=\{S\}$$

が成り立つから、書き換えることができて、

$$S\Rightarrow ABC$$

となる。次に導出された文形式：ABCを規則集合：{A→aA,B→bB,C→cC} が書き換えることを考える。このとき、

$$N_p=\{A,B,C\}\subseteq N_\alpha=\{A,B,C\}$$

が成り立つから、書き換えることができて、

$$\Rightarrow aAbBcC \quad by\{A\rightarrow aA,B\rightarrow bB,C\rightarrow cC\}$$

となる。以下同様に、たとえば

$$\Rightarrow aaAbbBccC \quad by\{A\rightarrow aA,B\rightarrow bB,C\rightarrow cC\}$$
$$\Rightarrow aaabbbccc \quad by\{A\rightarrow a,B\rightarrow b,C\rightarrow c\}$$

となる。この場合は

$$S\overset{*}{\Rightarrow}aaabbbccc$$

である。
このようにして、一般に

$$S\overset{*}{\Rightarrow}a^nb^nc^n \quad n\geqq 1$$

であることがわかるから、この文法の生成する言語：L(G)は

$$L(G)=\{a^nb^nc^n \mid n\geqq 1\}$$

である。L(G)は文脈自由言語ではない。□

次に定義3(1)の条件を満たす場合について示す。

【例3】 S-CCFG：G＝(N,T,P,S) を次のように与える。

$$N=\{S,X,A,B\}$$
$$T=\{a,b\}$$
$$P=\{\{S\rightarrow XX\},$$
$$\{X\rightarrow AB\},$$
$$\{A\rightarrow a,B\rightarrow b\},$$
$$\{A\rightarrow aA,B\rightarrow bB\}\}$$

文形式：Sから書き換えを行うと、

$$S\Rightarrow XX \quad by\{S\rightarrow XX\}$$

となる。文形式：XXには規則集合：{X→AB} が適用可能であるが、このときは定義3(1)の条件を満たすから、文形式：XXのふたつのXには同時に同じ生成規則を適用し

$$XX\Rightarrow ABAB \quad by\{X\rightarrow AB\}$$

となる。以下同様にして、たとえば

$$\Rightarrow aAbBaAbB \quad by\{A\rightarrow aA,B\rightarrow bB\}$$
$$\Rightarrow aabbaabb \quad by\{A\rightarrow a,B\rightarrow b\}$$

となる。結局、言語：L(G)は

$$L(G)=\{\alpha\alpha \mid \alpha=a^nb^n,\ n\geqq 1\}$$

である。□

次に定義3(2)の条件を満たす場合について示す。

【例4】 S-CCFG：G＝(N,T,P,S) を次のように与える。

$$N=\{S,X,Y,N\}$$
$$T=\{☆,|\}$$
$$P=\{\{S\rightarrow ☆X,\quad S\rightarrow Y\},$$
$$\{X\rightarrow |☆NX,\ Y\rightarrow N|Y\},$$
$$\{X\rightarrow \varepsilon,\quad Y\rightarrow N\}$$
$$\{N\rightarrow ☆\},$$
$$\{N\rightarrow ☆N\}\}$$

開始記号：Sの書き換えを考えると、規則集合：{S→☆X, S→Y} が適用可能であることがわかる。そしてこの適用は定義3(2)の条件を満たしているから、

$$S\Rightarrow ☆X\approx Y \quad by\{S\rightarrow ☆X,\ S\rightarrow Y\}$$

と書き換えられる。以下同様にして、たとえば

$$\Rightarrow ☆|☆NX\approx N|Y \quad by\{X\rightarrow |☆NX,\ Y\rightarrow N|Y\}$$
$$\overset{*}{\Rightarrow}☆|☆☆X\approx ☆|Y$$
$$\overset{*}{\Rightarrow}☆|☆☆|☆☆☆X\approx ☆|☆☆|Y$$

$$\overset{*}{\Rightarrow}☆ \mid ☆☆ \mid ☆☆☆ \approx ☆ \mid ☆☆ \mid ☆☆☆$$

となる。結局

$$L(G)=\{☆, ☆ \mid ☆☆, ☆ \mid ☆☆ \mid ☆☆☆, \cdots\}$$

である。もし☆,☆☆,☆☆☆・・・をそれぞれ自然数：１，２，３・・・とみなすならば、この文法は記号：｜で区切られた１からｎまで（ｎ≧１）の任意の自然数列を生成すると考えることができる。□

以上S-CCFG、S-CCFLについて述べた。S-CCFGは文脈自由な生成規則を規則集合として束縛し、規則集合に含まれる生成規則は同時に適用するという書き換え規則を与えたものである。もし規則集合に含まれる要素数を１に限定したならば、それは文脈自由文法と等価である。

## ２．２．Coupled Context Free Grammar

S-CCFGを拡張してCCFGの定義を与える。

【定義５】 CCFGは５項組：G＝(N,T,P,S,R) で定義される。ここにN、T、Pは定義１と同じである。SはNの部分集合で『開始記号集合』である。Sの要素は開始記号である。RはNの部分集合で『補助記号集合』である。Rの要素は『補助記号』である。SとRは

$$S \cap R = \phi$$

を満たねばならない。□

【例５】 次の５項組：G＝(N,T,P,S,R) はCCFGである。

$$N=\{P,Q,Y,Z\}$$
$$T=\{a,b,c\}$$
$$P=\{\{P\rightarrow YP, Q\rightarrow ZQ\},$$
$$\{P\rightarrow Y,\ Q\rightarrow Z\},$$
$$\{Y\rightarrow a, Z\rightarrow a\},$$
$$\{Y\rightarrow c, Z\rightarrow b\}\}$$
$$S=\{P,Q\}$$
$$R=\phi$$

この文法による言語の生成は例６で示す。□

CCFGにおける書き換えの対象は『文形式集合』である。文形式集合とその書き換え規則は次のように定義される。

【定義６】 CCFG：G＝(N,T,P,S,R)の文形式集合は次のように定義される。

(1) Sは文形式集合である。
(2) Aが文形式集合であり、Aが規則集合：p∈Pによって Bに書き換えられるならば、Bは文形式集合である。

文形式集合の要素は文形式である。□

【定義７】 CCFG：G＝(N,T,P,S,R)において、規則集合：p∈Pによる文形式集合：Aの書き換えは次のように定義される。

pに含まれる全ての生成規則の左辺の非終端記号の集合を$N_p$とし、Aに含まれる全ての文形式に含まれる全ての非終端記号の集合を$N_A$とする。

このとき、もし

$$N_p - R \subseteq N_A$$

が成り立つならば、

$$A' = A \cup (R \cap N_p - N_A)$$

として、A'に含まれる全ての文形式をpを用いて書き換える。ただしそれぞれの文形式の書き換えは定義３に従う。□

定義７のA'の計算は、$N_p$には含まれるが$N_A$には含まれない必要最少限の非終端記号をAに加え、それをA'としようという意味である。Rが空でない例は例７、８に示す。

定義７に従って文形式集合：Aを文形式集合：Bへ書き換えることを

$$A \Rightarrow B$$

と表す。また⇒の反射的かつ推移的閉包を$\overset{*}{\Rightarrow}$と表す。

文形式集合の書き換えをその要素の書き換えに立ち入ってみた場合に、

$$A \Rightarrow B$$

であるときにもしα∈Aがβ∈Bへ書き換えられるならば、それを

$$\alpha \Rightarrow \beta \,/\, A \Rightarrow B$$

と表す。同じく$\alpha_1, \alpha_2, \cdots, \alpha_j$∈Aがそれぞれ$\beta_1, \beta_2, \cdots, \beta_j$∈Bへ書き換えられるならば、それを

$$\alpha_1 \Rightarrow \beta_1, \alpha_2 \Rightarrow \beta_2, \cdots, \alpha_j \Rightarrow \beta_j \,/\, A \Rightarrow B$$

と表す。またこの反射的かつ推移的閉包を

$$\alpha_1 \overset{*}{\Rightarrow} \beta_1, \alpha_2 \overset{*}{\Rightarrow} \beta_2, \cdots, \alpha_j \overset{*}{\Rightarrow} \beta_j \,/\, A \overset{*}{\Rightarrow} B$$

と表す。

次にCCFG：Gの生成する言語を定義する。

【定義8】 CCFG：$G=(N,T,P,S,R)$が全ての開始記号：$X_1,X_2,\cdots,X_n\in S$に関して生成する言語：$L(G,X_1,X_2,\cdots,X_n)$は次のように定義される。

$$L(G,X_1,X_2,\cdots,X_n)=\{(x_1,x_2,\cdots,x_n)\mid X_1\overset{*}{\Rightarrow}x_1,\ X_2\overset{*}{\Rightarrow}x_2,\ \cdots,\ X_n\overset{*}{\Rightarrow}x_n\ /S\overset{*}{\Rightarrow}\{\beta_1,\beta_2,\cdots,\beta_m\},\ \beta_i=\alpha_i\approx\alpha_i\approx\cdots\approx\alpha_i,\ \alpha_i\in T^*,\ i=1\sim m,m\geq 1\}$$

□

$L(G,X_1,X_2,\cdots,X_n)$は、開始記号集合：Sから導出される文：$x_1,x_2,\cdots,x_n$のn項組の全体である。$L(G,X_1,X_2,\cdots,X_n)$はCoupled Context Free Language(CCFL)であるという。

言語の生成例を示す。

【例6】 例5について考える。

まず開始記号集合：$S=\{P,Q\}$を規則集合：$p=\{P\to YP,Q\to ZQ\}$で書き換えることを考える。このとき、

$$N_p-R=\{P,Q\}\subseteq N_S=\{P,Q\}$$

が成り立つから書き換えることができて、

$$S'=S\cup(R\cap N_p-N_S)=S$$

であるから、Sに対し規則集合を適用し

$$S\Rightarrow\{YP,ZQ\}\quad by\{P\to YP,Q\to ZQ\}$$

となる。以下同様に書き換えて、たとえば

$$\overset{*}{\Rightarrow}\{aP,aQ\}$$
$$\overset{*}{\Rightarrow}\{acP,abQ\}$$
$$\overset{*}{\Rightarrow}\{acc,abb\}$$

となる。このようにして、Gが生成する言語：$L(G,P,Q)$は2項組の集合：

$$L(G,P,Q)=\{(a,a),(c,b),(aa,aa),(ac,ab),(ca,ba),(aaa,aaa),\cdots\}$$

である。□

次に補助記号による言語生成の制御について例を示す。

【例7】 CCFG：$G=(N,T,P,S,R)$ を次のように与える。この例は例5、6と異なりRが$\phi$でない。

$N=\{X,Y,A\}$
$T=\{a,b\}$
$P=\{\{X\to A,Y\to A,Y\to a\},\ \{A\to a\},\ \{A\to b\}\}$
$S=\{X\}$
$R=\{Y\}$

まず開始記号集合：$S=\{X\}$を規則集合：$p=\{X\to A,Y\to A,Y\to a\}$ で書き換えることを考える。このとき、

$$N_p-R=\{X,Y\}-\{Y\}=\{X\}\subseteq N_S=\{X\}$$

が成り立つから書き換えることができる。そして

$$S'=S\cup(R\cap N_p-N_S)=\{X\}\cup(\{Y\}\cap\{X,Y\}-\{X\})=\{X,Y\}$$

であるから、$\{X,Y\}$ に対し規則集合を適用し

$$S\Rightarrow\{A,A\approx a\}\quad by\{X\to A,Y\to A,Y\to a\}$$

となる。以下同様に書き換えて、

$$\Rightarrow\{a,a\approx a\}\quad by\ \{A\to a\}$$

または

$$\Rightarrow\{b,b\approx a\}\quad by\ \{A\to b\}$$

となる。よって$L(G,X)$は、定義8から

$$L(G,X)=\{a\}$$

である。もし規則集合：$\{X\to A,Y\to A,Y\to a\}$を$\{X\to A,Y\to A,Y\to b\}$へ変更したならば、

$$L(G,X)=\{b\}$$

となる。このように$L(G,X)$は補助記号：Yの書き換えで制御されている。□

例7では、補助記号は最初の書き換えで1回だけ文形式集合に加えられた。しかし一般には補助記号は導出過程で動的に複数回文形式集合に加えることができる。その例を次に示す。

【例8】 CCFG：$G=(N,T,P,S,R)$ を次のように与える。

$N=\{X,Y,A\}$
$T=\{a,b\}$

P={{X→AX,Y→A,Y→a},
{X→A, Y→A,Y→b},
{A→a},
{A→b}}
S={X}
R={Y}

例6同様に書換を行うと、たとえば

S⇒{AX,A≈a} by{X→AX,Y→A,Y→a}
⇒{aX,a≈a} by {A→a}
⇒{aaX,a≈a,a≈a}
⇒{aab,a≈a,a≈a,b≈b}

となる。このようにして

$$L(G,X)=\{a^nb \mid n\geqq 0\}$$

である。$a^nb$が導出されるときには、補助記号：Yは文形式集合にn＋1回加えられる。□

以上ＣＣＦＧ、ＣＣＦＬについて述べた。ＣＣＦＧは複数のS-ＣＣＦＧをCouplingした形になっている。もしS-ＣＣＦＧによる導出過程を一本の導出木の成長で表すならば、ＣＣＦＧによる導出過程は複数本の導出木が互いに干渉しながら同時に成長していくものとみなすことができる。

## ２．３．Coupled Context Free Languageの性質

この節ではＣＣＦＬのいくつかの性質について述べる。

まず最初にＣＣＦＬの表現能力についてふたつの定理を示す。証明は繁雑なので省略するが、定理１に関しては証明の方針を例９に示す。

【定理１】 帰納的に加算な言語はＣＣＦＬである。
(証明) 任意の半無限長１テープ決定性ＴＭが受理する言語を生成するＣＣＦＧが存在することを示せばよい。
以下省略。□

【定理２】 ＣＣＦＬは帰納的に加算である。
(証明) ＣＣＦＬに含まれる任意の文を受理する非決定性ＴＭが存在することを示せばよい。
以下省略。□

定理１の証明の代りにＴＭのシミュレーション例を示す。

【例９】 １からｎ（ｎ≧１）までの自然数の和を求める関数：Ｆ(ｎ)について、引数値：ｎと関数値：Ｆ(ｎ)の２項組：(ｎ,Ｆ(ｎ))を受理するＴＭを考える。ただし簡単のために自然数の表現は

１＝☆、
２＝☆☆、
…
ｎ＝☆☆…☆

とする。

ＴＭのテープ上にはたとえば初期記号列：

(☆☆,☆☆☆)

を用意し、
(1) 関数値から引数値を引く。
(2) 引数値から１を引く。
を繰り返して、最終的にテープ上に

(☆,☆)

のみが残れば、初期記号列はこのＴＭによって受理されるとする。上の例ではテープの状態はつぎのように遷移していく。

(☆☆,☆☆☆)
(☆,☆)

よって（☆☆,☆☆☆）は受理された。これはＦ(２)＝３の関係を表している。

そこで全ての（ｎ,Ｆ(ｎ)）（ｎ≧１）を生成するＣＣＦＧ：Ｇを与え、上のＴＭの動きを文形式集合の書き換えでシミュレートする。

文法：Ｇ={Ｔ,Ｎ,Ｐ,Ｓ,Ｒ} は次のようになる。

Ｔ={ (,|,) ,☆}
Ｎ={Arg,Val,C,M,N,F,Any}
Ｐ={{Arg→N,Val→F,C→(N|F)Any,C→M},
{M→(☆|☆)},
{M→(☆N|NF)(N|F)Any,
M→(☆N|NF)M},
{Any→ε},
{Any→(N|F)Any},
{N→☆},
{N→☆N},
{F→☆},
{F→☆F}}
Ｓ={Arg,Val}
Ｒ={C}

たとえば(☆☆,☆☆☆)の導出は

{Arg,Val}⇒{N,F,(N|F)Any≈M}
⇒{☆☆,☆☆☆,
(☆☆|☆☆☆)Any≈M}
⇒{☆☆,☆☆☆,
(☆☆|☆☆☆)(☆|☆)≈M}

$\overset{*}{\Rightarrow}${☆☆,☆☆☆,
(☆☆|☆☆☆)(☆|☆)
≈(☆N|NF)(N|F)Any
≈(☆N|NF)M }
$\overset{*}{\Rightarrow}${☆☆,☆☆☆,
(☆☆|☆☆☆)(☆|☆)
≈(☆☆|☆☆☆)(☆|☆)
≈(☆☆|☆☆☆)M }
$\overset{*}{\Rightarrow}${☆☆,☆☆☆,
(☆☆|☆☆☆)(☆|☆)
≈(☆☆|☆☆☆)(☆|☆)
≈(☆☆|☆☆☆)(☆|☆) }

となる。よって

$$(☆☆,☆☆☆)\in L(G,Arg,Val)$$

である。このようにして一般に

$$L(G,Arg,Val)=\{(n,F(n)) \mid n\geq 1\}$$

がわかる。□

例9は簡単な全域的な関数に関するTMの動きをCCFGでシミュレーションしたものであったが、同様にしてTMの任意の動きをCCFGでシミュレーションできることを示せば、それは定理1の証明になっている。

次に、計算モデルの意味論に直結する定理を与える。

【定義9】『CCFG：$G=(N,T,P,S,R)$から$X\in S$に関して作られるS-CCFG：$G_X$』は、Pに含まれる全ての規則集合について

$$X\overset{*}{\Rightarrow}\cdots Y\cdots$$

であるようなYを左辺に持つ生成規則のみを残して得られた規則集合の全体を$P_X$とするとき

$$G_X=(N,T,P_X,X)$$

である。□

CCFG：GとGから作られるS-CCFG：$G_X$との間には次の関係が成り立つ。

【定理3】 CCFG：$G=\{N,T,P,S,R\}$に対し、Gから全ての開始記号：$X_1,X_2,\cdots,X_n\in S$について作られるS-CCFGをそれぞれ$G_{X1},G_{X2},\cdots,G_{Xn}$とするとき

$$L(G,X_1,X_2,\cdots,X_n)\subseteq L(G_{X1})\times L(G_{X2})\times\cdots\times L(G_{Xn})$$

（証明）左辺に含まれるが右辺には含まれないようなn項組が少なくともひとつ存在するとして背理法を用いる。□

## 3．計算モデル

この章では、CCFGをデータ構造を指向したプログラムとして解釈する計算モデルについて述べる。

### 3．1．構文

プログラムは定義5を満たすCCFGの5項組：$(N,T,P,S,R)$である。

### 3．2．意味

この節では、はじめにCCFG：$G=(N,T,P,S,R)$の5項組にプログラミングの用語を対応つける。これはプログラムの意味論とは直接には関係ないが、Gの内容をプログラムとして理解する助けとなる。

そしてその後にプログラムの表示的意味と操作的意味を与える。

$(N,T,P,S,R)$を次のように解釈する。

(1) データ

$a\in T$ならば、aは『基底データ』である。$A\in N$ならば、Aは『構造データ』である。両者を『データ』という。

(2) 局所的なデータ構造

ひとつの生成規則：

$$A\rightarrow B_1,B_2,\cdots,B_k \qquad A\in N,\ B_i\in V$$

は構造データ：Aの局所的な『構造』を表す。これは「構造データ：Aはデータ：$B_1,B_2,\cdots,B_k$の列から構成されている」と解釈する。

(3) データ構造の局所的な関係

生成規則の集まりである規則集合はデータ構造間の局所的な『関係』を表す。次の(a),(b),(c)がその基本的な解釈である。

(a) もし規則集合が

$$\{A\rightarrow\alpha、B\rightarrow\beta、\cdots\}$$
$$A\neq B、\quad \alpha,\beta\in V^*$$

ならば、これは「構造データ：Aが$\alpha$から構成されているならば、構造データ：Bは$\beta$から構成されている。逆にBが$\beta$から構成されているならば、Aは

αから構成されている」と解釈する。

(b) もし規則集合が

$$\{A\to\alpha、A\to\beta、\cdots\}$$

ならば、これは「構造データ：Aはαかつβから構成されている、すなわちAは『多重な構造』を有する」と解釈する。

(c) もし規則集合が

$$\{A\to\alpha_1 C\alpha_2、B\to\beta_1 C\beta_2、\cdots\}$$
$$C\in N,\ \alpha_1,\alpha_2,\beta_1,\beta_2\in V^*$$

すなわちα、βが同じ非終端記号：Cを持つならば、これは「構造データ：A、Bは同じ構造データ：Cを含む」と解釈する。

(4) 入出力データ

下記の定義10参照。

(5) 補助データ

A∈Rならば、Aはプログラムの実行を助ける『補助データ』である。

次に表示的意味と操作的意味を述べる。

プログラム：G＝(N,T,P,S,R)の表示的意味を与える前に、Gに対して入出力データを定義する。

【定義10】 X∈Sならば、Xは『入出力データ』である。

入出力データ：Xの『データ構造』はGからXに関して作られるS-CCFG：$G_X$ (定義9参照)である。

入出力データ：Xの『データ領域：$D_X$』はXのデータ構造：$G_X$の生成する言語：

$$D_X=L(G_X)$$

である。

v∈$D_X$ならば、vはXの『値』である。

Gの『データ領域：$D_G$』は、全ての入出力データ：X1,X2,…,Xn∈Sのデータ領域：$D_{X1},D_{X2},\cdots,D_{Xn}$の直積：

$$D_G=D_{X1}\times D_{X2}\times\cdots\times D_{Xn}$$

である。□

プログラムの表示的意味を次のように与える。

【定義11】 意味関数：SFは、GからGのデータ領域：$D_G$の部分集合への関数：

$$SF(G)=L(G,X_1,X_2,\cdots,X_n)$$

である。□

定理3より

$$SF(G)\subseteq D_G$$

は保証されている。

SF(G)をプログラム：Gの『解』という。もし

$$SF(G)\subsetneq D_G$$

が成り立つならば、Gは『意味のあるプログラム』であるという。もし

$$SF(G)=D_G$$

が成り立つならば、Gは『入出力データの宣言』であるという。

次に操作的意味のひとつとして、次のような理想的なインタプリタを与える。

【定義12】 プログラム：G＝(N,T,P,S,R)の『計算』とは、文形式集合の系列：

$$\Omega_0,\Omega_1,\Omega_2,\cdots,\Omega_m$$

$\Omega_0=S$,
$\Omega_i\Rightarrow\Omega_{i+1}$ $1\leqq i\leqq m-1$
$\Omega_m$に適用可能な規則集合は存在しない。

である。もしこの系列が

$$\Omega_m=\{\beta_1,\beta_2,\cdots,\beta_n\}$$
$$\beta_i=\alpha_i\approx\alpha_i\approx\cdots\approx\alpha_i,$$
$$\alpha_i\in T^*,1\leqq i\leqq n,n\geqq 1$$

を満たすならば、この計算は『成功』であるといい、この計算の解は$(x_1,x_2,\cdots,x_n)$である。ただし

$$X_1\overset{*}{\Rightarrow}x_1,\ X_2\overset{*}{\Rightarrow}x_2,\ \cdots,\ X_n\overset{*}{\Rightarrow}x_n\ /\ S\Rightarrow\Omega_m$$

□

【定義13】 『インタプリタ』は、プログラム：Gの全ての成功する計算とその解を、文形式集合の書き換えを通して求めることができる。□

幾つかのプログラム例を示す。

【例10】 次の文法：G＝{N,T,P,S,R}は記号列の反転プログラムである。

N={I,O,A,B,X}
T={a,b}
P={{I→A, O→B},

{A→XA, B→BX},
{A→X　B→X　},
{X→a},
{X→b}}
S={I,O}
R=φ

ここにa、bは基底データ、I,O,A,B,Xは構造データである。規則集合：{I→A,O→B}は、『構造データ：Iが構造データ：Aであるとき、構造データ：Oは構造データ：Bである(逆も成り立つ)』と解釈する。規則集合：{A→XA,B→BX}は、『構造データ：Aが構造データ：X,Aの並びからなるとき、構造データ：Bは構造データ：B,Xの並びからなる(逆も成り立つ)』と解釈する。以下同様である。

またI、Oは入出力データであり、データ構造：$G_I$、$G_0$、データ領域：$D_I$、$D_0$はそれぞれ

$G_I=(N,T,P_I,I)$　$G_0=(N,T,P_0,O)$

$P_I=\{\{I\to A\},\{A\to XA\},\{A\to X\},\{X\to a\},\{X\to b\}\}$

$P_0=\{\{O\to B\},\{B\to BX\},\{B\to X\},\{X\to a\},\{X\to b\}\}$

$D_I=T^+$　　$D_0=T^+$

である。

Gの解は

$$L(G,I,O)=\{(a,a),(ab,ba),(aab,baa)\cdots\}$$

である。解は確かに記号列が互いに反転した関係になっている。このとき

$$L(G,I,O)\subsetneq D_I\times D_0$$

であるから、Gは意味のあるプログラムである。□

【例11】 例10のプログラムで規則集合：

$$\{I\to A,\ O\to B\}$$

を

$$\{I\to A,\ I\to aab,\ O\to B\},$$

と変更すると、変更されたプログラムでは

$D_I=\{aab\}$、$D_0=T^+$

$L(G,I,O)=\{(aab,baa)\}\subsetneq D_I\times D_0$

となる。これはひとつの特定例に関する計算と考えることができる。□

次に補助データの使用法のひとつを示す。

【例12】 記号列に含まれる記号：cを記号：bに書き換え(例5、例6参照)、さらにそれを反転する(例10参照)プログラムを示す。この場合には入出力データの関係を直接記述できないから、中間ファイルに相当する補助データ：Mを設ける。

G={N,T,P,S,R}
T={a,b,c}
N={I,M,O,P,Q,Y,Z,A,B,X}
P={{I→P,M→Q, M→A,O→B},
{P→YP,Q→ZQ},
{P→Y,　Q→Z　},
{Y→a,Z→a},
{Y→c,Z→b}
{A→XA,B→BX},
{A→X,　B→X　},
{X→a},
{X→b}}
S={I,O}
R={M}

このプログラムでは

$D_I=\{a,c\}^+$,　$D_0=\{a,b\}^+$

$$L(G,I,O)=\{(a,a),(c,b),(ac,ba),(acc,bba),\cdots\}\subsetneq D_I\times D_0$$

となる。□

例12では、入出力データ：I,O間にデータ構造の関係を直接プログラミングできない『構造の不一致(structure clash[6])』がある。そこで、補助的なデータ：Mを用いてIとM、MとOの関係を記述するふたつのプログラムをあらかじめ作成し、Mを多重なデータ構造にして、ふたつのプログラムを結合、IとOの関係を与えている。これはプログラミング・テクニックのひとつである。

## 3．3．いくつかの性質

ＣＣＦＧに基づく計算モデルには次のような特徴がある。

(1) 計算の双方向性

表示的意味(定義11)からすぐにわかるように、この計算モデルはデータ間の関係記述を行う。よって入出力

データ間には本質的に計算の方向がない。

(2) 非決定性

プログラムの中のある複数の規則集合：$p_1, p_2, \cdots, p_k$に含まれる生成規則の左辺の非終端記号の集合をそれぞれ$N_{p1}, N_{p2}, \cdots, N_{pk}$とし、ある文形式集合：Aに含まれる非終端記号の集合を$N_A$とするとき

$$\phi \subsetneq N_{pi} \subseteq N_A \qquad i=1\sim k$$

が成り立つならば、インタプリタの動きは非決定的になる。

これまで例にあげたプログラムは全て非決定的であった。

(3) 十分な計算能力

定理1から任意の帰納的関数をプログラミングすることができる。これはこの計算モデルに基づくプログラミング言語が汎用言語であることを保証する。

(4) データ構造の検証の容易さ

ある終端記号列がS-CCFL：L(G)に属するかどうかの判定は決定可能な問題である。よってS-CCFGで記述された入出力データ構造が正しく設計されているかどうかの検証は比較的容易であると思われる。

## 4．おわりに

本報告では文脈自由な生成規則を用いた形式文法：CCFGとそれに基づく計算モデルを述べた。

この計算モデルでは、CCFG：Gをプログラムし、Gから開始記号：Xに関して作られるS-CCFG：$G_X$をプログラムの入出力データ構造と解釈する。プログラミングはデータ構造およびその関係の記述であるから、これはJacksonの方法[6]と親和性がよい。

今後の課題はこの計算モデルに基づく実用的なプログラミング言語の設計、その処理系開発である。

今回は計算モデルのみに絞って述べたが、言語設計においては次のような問題がある。

(1) これは計算モデルの話にも関係するが、ストリームなどの『無限列』は今回は扱わなかった。有効なプログラミング手法として理論的にも実際的にもストリームを扱えるようにしたい。

(2) CCFGはデータとして記号列のみを扱うが、記号列は低次のフラットなデータ構造である。実際のプログラミングの局面には『高次なデータ構造』が必要になる。たとえば列を要素とする列などを扱えるべきである。

(3) 物理的な入出力デバイスとプログラムとの通信機能が必要である。もちろんその際に『副作用』はあるべきでない。

処理系開発では『効率』が最大の問題である。CCFGは文脈自由文法を拡張したものであるから、処理系は複数の文脈自由文法のパーザーが並行に(concurrently)実行するものと考えることができる。パーザーの実現方式には大きく分けてトップ・ダウン法とボトム・アップ法があるが、それぞれに対して次のような問題がある。

(1) トップ・ダウン法の場合には解の予測が可能であるから、虫食い算的な、これまで論理型言語が得意としていた問題を解くことができる。しかしプログラム(文法)中に『再帰性』があると決定的な計算はできないばかりか、本質的でない無限ループに陥る可能性もでてくる。非決定的な計算が入ることは我慢するとしても、本質的でない無限ループに陥ることは避けたい。

(2) ボトム・アップ法の場合には再帰性に関しては問題がない。しかしボトム・アップであるが故に解の予測ができなくなり、計算によっては実行できないものが生じる恐れがある。

現在は上にあげた課題を検討中である。

## 参考文献


[1] Gruss, M., Lentin, A. : Notions sur les grammaries formelles, Gauthier-Villars Editeur (1967)

[2] Salomaa, A. : Formal Languages, Academic Press (1973) Chap. 5

[3] 安本美典, 野崎昭弘 : 言語の数理, 筑摩書房 (1976) 11章

[4] 佐々政孝 : 属性文法, コンピュータソフトウェア, Vol.2, No.3 (1985) pp.560-562

[5] Tarnlund, S-A. : Horn Clause Computability, BIT, Vol.17 (1977) pp.215-226

[6] Jackson, M.A. : Principles of Program Design, Academic Press (1975)