記 号 処 理 51－2
（１９８９．６．２）

# 格文法を用いたLogoの拡張

酒井公治
慶応義塾大学理工学研究科

数式処理において、+や－のような可換性、多義性を持つ演算子の取り扱い、および、それらの性質を持つ演算子を新たに定義することには様々な難しさが存在する．本研究はプログラミング言語Logoが本来持っている特長を考慮して、そこに格文法の考え方、およびオブジェクト指向の技術を応用することでLogoを拡張し、可換性、多義性を持つ演算子の取り扱いを容易にすることを試みた．

# Improving Logo with case grammar

Kimiharu Sakai
Faculty of science and Technology, Keio University
(Nakanishi Lab., Keio Univ., 3-14-1, Hiyoshi, Kouhoku-ku, Yokohama-shi, 223, Japan)

It is very difficult to manipulate the operator and define new one with commutativity or ambiguity, like + and -, in Computer Algebra. In this report, I make the new improving Logo with case grammar and object oriented technique by thinking the characteristic of Logo, and it become easier for it to manipulate the operator with commutativity or ambiguity easier.

**あらまし**
可換性、多義性を持つ数式を処理するのが容易なように、格文法の考え方を導入し、それをオブジェクト指向の技術を用いて実現した、プログラミング言語Logoの拡張を試みた。

## 1　まえがき

数式処理の技術やアルゴリズムは現在広く知られ、多くの分野で実質的な成果が挙がってきた。数式処理には、Lispに代表される記号処理に向いた機能を豊富に持つプログラミング言語が用いられるのが普通であるが、それらの多くは文法要素の出現順序に依存した意味解析を行なうので、四則演算の中に存在する可換性や多義性を実現するのに特別な技術を必要とする。自然言語処理の中では、可換性や多義性を実現するのが容易な文法として、格文法が知られている。そこで、格文法の考え方を用いて、それをオブジェクト指向の技術で実現し、Logoを拡張することを試みる。この拡張により、本研究で試作したプログラミング言語を用いれば、容易に可換性、多義性を持つ演算を定義することが可能になる。

## 2　Logoの特長

プログラミング言語Logoには、筆者が数式処理に向くと考える幾つかの特長が存在する。以下に本研究で注目するLogoの特長を挙げる。

1）Lispに似た記号処理用の命令を持つ。

例
```
FPUT、FIRST、BUTFIRST
```

故に記号処理を容易に実現できる。

2）中間置二項演算子が存在する。

例
```
1+2、2-3、3*4、4/5
```

故に数学の歴史的な記述方法に近い表記が可能である。

3）関数（オペレータ）や手続き（コマンド）の実引き数の有効範囲を示す区切り記号がない。

例
```
FD 100 RT 90
```

故に新しい中間置、または前置演算子を定義することが可能である。また、ある種の数学上の関数に対しては、歴史的な記法に近い表記が可能になる。

例
```
SIN 90、LOG 2 4
```

これらの特長は、可換な二項演算子を新たに定義するときに大きな意味を持つ。

4）関数名や手続き名と変数やデータの記法が異なる。

例
```
FPUT "X :Y
```

故に処理系は容易に変数と関数を区別できるが、これは数学の歴史的な記述法に対しては異質になる。

5）必要に応じてカッコで結合順序を変更できる。

例
```
FD(100)
```

故に従来のプログラミング言語に見られるような表現を行なうことも可能である。

## 3　数式処理のためのデータ構造

数式処理にLispが適した側面を多く持っていることはよく知られている。しかし、現在知られているアルゴリズムを純粋にLispのリスト処理だけで実現することを考えると、その柔軟な記述能力がかえってシステムの安定性を損なうこともある。近年のLispの中には、非常に多くのデータ型を持つものもあり、特に構造体の導入は近年の数式処理用のアルゴリズムを実現するのには有効である。反面、構造体を用いることは本来Lispが持つ柔軟で動的なデータ処理に大きな制限を与えることもまた事実である。

簡単な3次多項式のリスト表現と構造体による表現の例を以下に挙げる。

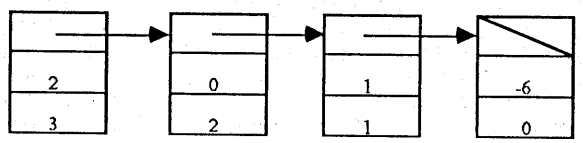
例

$2X^3+X-6$

リスト表現

(+ (* 2 (EXPT X 3))(* 0 (EXPT X 2))(* 1 (EXPT X 1))(* -6 (EXPT X 0)))

構造体による表現




即ち数式処理においては、歴史的な表記法と安定で確実なアルゴリズムを必要とするためにデータ表現に制限を設ける必要がある一方で、柔軟で動的なデータ表現も要求される。この両者の要求は時に深刻な対立を招くことになる。

先に挙げた特長からLogoは以上の対立を吸収する可能性を持っていると筆者は考えた。たとえば、Logoは本来二項演算子の中間置表現を持っているので、あるアルゴリズムの実現の容易なデータ構造を用いたいというような何か特別な理由がないかぎり、数式をそのままの形で処理することができる。

ただし、現在のLogoにはデータ型がほとんど存在しないのでそのままの機能では十分実用に耐えるようにするのは難しい。ここで簡単に考えてしまえば、Logoの仕様の中に豊富なデータ型を導入したり、新たな型の定義機能を導入すればよいかのようにもみえる。しかし、その方法では柔軟な操作に支障をきたすという側面を補うことができず、なによりも他のプログラミング言語を使用するほうが容易である。そこで近年の研究の成果を考慮して、オブジェクト指向の技術を一挙に導入してしまう方が良いと考える。元来、Logoの特長とされるタートル・グラフィックスの考え方は、オブジェクト指向を取り入れやすいものであり、Logoにオブジェクト指向を導入する試みは幾らか知られている。ただし、これまでの試みでは総称関数 (Generic function)の技術が導入されたものは知られていない。

そこで本研究ではLogoの特長を十分に考慮したうえで、数式処理に向いたLogoの拡張を考え、Logoの処理系に格文法の考え方を導入し、それをオブジェクト指向の技術を用いて、特に総称関数の技術で実現することを試みる。

## 4 格文法の考え方

自然言語処理などで行なわれる格文法の考え方は概ね以下のようになっている。

例えば、

私は彼女に本をあげた。

のような文章は、

私は本を彼女にあげた。

や

彼女に私は本をあげた。

とほぼ同一の意味を表わしていると考えることができる。即ち、

主語 ◀——▶ 私は
間接目的語 ◀—▶ 彼女に
直接目的語 ◀—▶ 本を
動詞 ◀——▶ 挙げた。

と考えれば文を文脈解析できたことになり、以上の各要素をどのように並べても不自然なことはあっても人間には意味が通じることになる。

つまり、可換性を持つ言語に対して格文法の考え方は有効である。

## 5 演算子と格文法

通常の算術演算子はその特長として独自の結合優先順位を持ち、演算子によっては可換性を持っている。通常のプログラミング言語は文法要素の出現順序に依存して文脈解析が行なわれるので、可換性を実現するのは難しい。現在知られている文法理論の中では格文法が文法要素の出現順序に依存しない文脈解析に適している。以下に本研究における格文法の考え方を述べる。

まず初めに二つの集合を考える。一つはデータの集合であり、もう一つは格記述子の集合である。データと格記述子が結合したものを格と呼ぶ。データの集合は、プログラミング言語の設計と実際のプログラムでの処理によって分けられる複数の部分集合の合併集合になる。逆に、格記述子の集合は、それらによって生成が可能になる格の種類によって分けられる複数の部分集合の共通集合であり、それらの部分集合を格記述子の属性集合と呼ぶ。格文法の考え方では最小の文法要素は格、またはデータである。格の文法上および意味上の働きは、その格を構成する格記述子の属する属性集合によって決定される。

## 6　オブジェクト指向による格文法の実現

以上のような格文法の考え方において基礎となる集合のそれぞれをオブジェクト指向のクラスとして定義し、それらを継承して行くことで属性集合を拡張させる。オペレータまたはコマンドはそれらを表わす格記述子の属性集合を継承させることで区別する。故に本研究においてはLogoの関数にあたるオペレータ、および手続きにあたるコマンドを表わすワードを格記述子と考え、可換性を陽に記述するには格記述子のオーバーロードによって実現する。またオーバーロードするべき定義が存在しない場合は格文法の考え方より、処理系が自動的に既存の定義の中で仮引き数の属する型を考慮して、それらの型の中に可換性を表わすクラスを継承しているものがあれば、それを継承しているものの間で引き数の順番を変更し、適用できるかどうか、実行できるかどうかを判断し、もし可能なら変更した順番で実引き数と仮引き数を束縛して定義を実行する。

例

定義

オペレータ名　可換なクラス１　X　可換なクラス２　Y

実行

オペレータ名　可換なクラス２の実体　可換なクラス１の実体

束縛状況

X◀▶ 可換なクラス１の実体
Y◀▶ 可換なクラス２の実体

これは

オペレータ名　可換なクラス１の実体　可換なクラス２の実体

と同じ束縛になる。

ここにオペレータは文の要素であるが実行単位とは考えず、コマンドが存在して初めて文となり実行可能であると見なす。これは、コマンドが文の区切りを表わすことを意味する。

## 7　Logoの拡張

数式処理においては歴史的な数式の記法上、中間置表現ができると便利である。そして二項演算子で可換性を持つものを定義できることが望ましい。歴史的な記述法を強く意識するならば、前置形の単項演算子や後置形の演算子も定義できると都合が良い。他に上添え字や下添え字が記述できることが望ましいが本論文では論じない。

Logoの特長の一つとして引き数の有効範囲を示すカッコが存在しないことがある。そのため、引き数が一つのオペレータと前置型単項演算子の区別は見かけ上は存在しないので現在のLogoでも生成可能であるように思えるが、結合順位を指定できないのが問題になる。

例

1 + MINUS 1 * 2

前述のようにLogoには引き数の有効範囲を指示するカッコが存在しないので二項演算子も定義可能に思えるが、現在のオペレータの定義方法ではそのオペレータより以前に実行されたオペレータの実行結果を得る方法がないので、二項演算子の定義は不可能になる。

以上よりLogoに新しい演算子の定義機能を追加するには、可換性および多義性の実現結合順位の指定と以前の結果を参照する機能を追加すればよいことになる。そこで以下のようにそれらの機能を実現する。

可換な演算子は前述の処理系による自動的な引き数の順番の並べ変えによって実現される。もちろん、減算や除算のように可換性を持たない演算子を定義することもできなくてはならないが、位置依存の引き数束縛にしたい場合は可換性を表わすクラスを継承しなければよい。多義性の実現は総称関数のオーバーロードによって可能になる。

結合順位の強さは定義されたオペレータの属するクラスの継承の深さを比較し、深い方をより結合が強いと考える。同等の深さの場合は左結合とする。

また直前のオペレータの実行結果はLEFTというクラスを継承する変数によって参照可能とする。

## 8　クラスの実現

本研究では試みの一つとして定義や宣言も格文法、およびオブジェクト指向の考えを適用して解釈を行なう。

クラス定義もそれらに従って従来のプログラミング言語とは異なる方法を用いる。

クラスの定義はクラス型のクラスを継承しているオペレータの定義によって実現され、以後そのオペレータ名がクラス名を表わすことになる。即ち従来のLogoとは異なり、オペレータの定義を行なうにはそのオペレータの値の

属するクラスのリストも記述しなければならない。逆に、既存のクラス名を要素とするリストが、コマンドが現われるべきところに現われたならば、それがクラス定義であると見なされる。つまり、クラス型のクラスを継承しているワードを要素とするリストの出現は、以後、変数、オペレータ、またはコマンドの型の宣言と見なされ、直後にあるワードを以後の文脈でそれらの実体を表わす名前として使用可能にする。

例
[クラス型のクラスを継承しているワード]新しいクラスを表わすワード

クラス定義における実体変数（Instance Variable）は、クラス型のクラスを継承したオペレータの仮引き数の宣言として記述される。この仮引き数宣言のところでもオペレータのクラスと同様に、既存のクラス名の後ろに現われるワードを変数として、その定義の中で使用可能にする。

例
[クラス型のクラスを継承しているワード]新しいクラスを表わすワード　クラス名１　変数名　...

ここで以上のオペレータの定義は総称関数として実現するので、実体変数への外部からの操作は参照、変更を含めて利用者が独自に総称関数として定義する必要がある。

## ９　実体の生成

NEWと言うコマンドが実行されるとその後ろにあるワードが示すクラスの実体（Instance）を新しく生成する。本研究で試作したLogoは代入コマンドMAKEを持たないので、クラスの実体の命名は実引き数と仮引き数の束縛によって実現する。

例

定義
[A-CLASS] NEW-CLASS SOME-CLASS X

実行
NEW-CLASS NEW "SOME-CLASS

束縛状況
X ◀▶ SOME-CLASSの実体

## １０　むすび

本研究は数式処理に必要と考えられる可換性、多義性を持つ演算子が新たに容易に導入できるようにプログラミング言語Logoを拡張改良した。その他オブジェクト指向の技術の導入により、柔軟で、堅実な型を導入することも可能になった。今後は本研究で試作した拡張されたLogoの処理系を用いて数式処理システムを構築し、その有効性を実証して行くことが必要である。