ソフトウェア基礎論24−6
(1988. 2. 12)

演算子モデルによる自然言語処理

Prosessing of Natural Languages using the Operator Model

萩原正也、田中徹郎、徳田雄洋
Masaya HAGIWARA, Tetsuro TANAKA and Takehiro TOKUDA
山梨大学　工学部
Faculty of Engineering, Yamanashi University

1．はじめに

本論文では、日本語文を抽象的演算子のｱﾝﾊﾟｰｽ列と考え、この抽象演算子を５種類の演算子（前置・２項中置・後置・０項・一般）に分類し、構文構造をプログラミングシステムの分野で用いられている親が演算子・子が被演算子の関係にある木を用いて表すことを試みる。

本研究は、日本語インターフェースを簡易的に構成するための日本語処理の指針を定めることを目標に始まった[6]。

例えば、ＵＮＩＸコマンドの日本語インターフェースを構成する場合について考える。ＵＮＩＸコマンドの日本語インターフェースで入力される日本語文は、「ファイル /etc/passwdをプリンタpllに出力する」、「ユーザーtoorをホストコンピュータyufujiに登録する。」のような、文の形が命令形であるような文であり，又、’ファイル’，’プリンタ’，’ユーザー’など処理の手がかりとなるキーワードが使用されている。さらに，入力日本語文はコンピュータへの命令であるので意味的に曖昧であってはならず、そのため文自体の表現方法も、自由に表現できる人間同士の会話とは異なり、ある程度の制限がある。

このように、ある用途の決まった日本語インターフェースを作成する場合、対象となる日本語文もある形式の表現方法の文のみとなると考える。このある形式の表現方法のみの文の集合を、本論文では日本語サブセットと呼ぶことにする。例えば、ＵＮＩＸコマンドの日本語インターフェースで用いられる日本語文の集合は、ＵＮＩＸコマンド用日本語サブセットと呼ぶ。

本論文では、日本語サブセットの処理の際に、構文構造表現として、プログラミングシステムの分野で用いられている演算子モデルを用いることを提案する。この表現方法は、親が演算子・子が被演算子であるような単純な木構造で構文構造を表している。解析の結果、日本語文は木構造となり、この木構造をアンパースすることによって日本語文を生成することができる。

また、日本語文をこの演算子モデルに当てはめるために、日本語の単語を抽象的演算子のｱﾝﾊﾟｰｽ結果と考え、５種類の演算子（前置・中置・後置・０項・一般）に分類することについて述べる。

そして、演算子モデルを用いることにより簡易的に日本語サブセットの処理を行うことができることを、後置演算子文法と意味規則を使った属性文法を用いたり、日本語文を式と見て計算する方法を用いて日本語サブセットのインターフェースを実現することによって示す。

本論文の構成は次の通りである。第２章では、演算子モデルの説明と日本語を演算子モデルで表現することについて述べる。第３章では日本語の曖昧さを演算子モデルで説明する。第４章では、演算子モデルを用いて実際に構成したインターフェースの試作例としてＵＮＩＸの awkコマンド用日本語インターフェースとWinogradの積木の世界への日本語インターフェースについて述べる。

ＵＮＩＸは、ＡＴ＆Ｔ　ベル研究所の登録商標である。

２．日本語の演算子モデル

プログラミングシステムのコンパイラ生成系や構造エディタ生成系で取り扱っている基本モデルは、図１のようなモデルである。

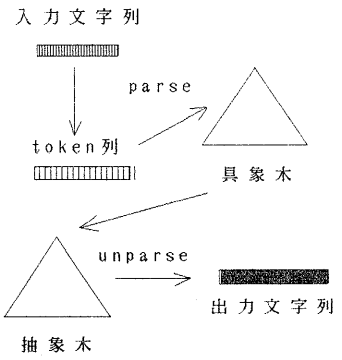


図１　基本モデル

例えば、カーネギーメロン大学のGandalfソフトウエア開発環境で使用されている構造エディタＡＬＯＥでは、次のように利用している[14]。

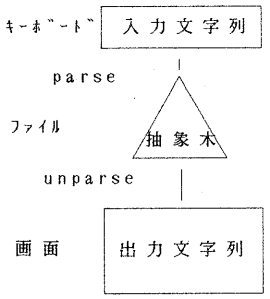


図２　ＡＬＯＥにおける抽象木の利用

ＡＬＯＥにおけるデータ構造は、単なる文字の集まりではなく、内部表現として木構造になっている。この木構造は、解析木から冗長な情報を消去した抽象的な木になっている。この抽象木は、編集の対象となるプログラミング言語に依存している。

しかし、ファイルの抽象木構造をそのまま端末でみることはできず、抽象木に予約語やセミコロン・かっこ・タブ・スペースなどを追加して端末上で見やすい形にする操作を行う。この見ることのできない形式の木構造表現を見える文字列形式に出力することをアンパースと言う。

このように、抽象木と具象木を区別する考え方は、プログラミングシステムの他の分野や自然言語処理の分野でも一般的である。例えばウィーン定義言語[20]や、変形生成文法[5]や、格文法[6]や、概念依存文法[15]などである。

図３は、このモデルを用いた算術式の中置記法から後置記法への変換例である。

算術式は、まず token列に変換され、中置式の文法による構文解析で具象木を生成する。具象木から不必要な情報を消去し抽象木を生成する。抽象木に対し後置式を生成するアンパース手続きを行うことによって後置式が生成される。

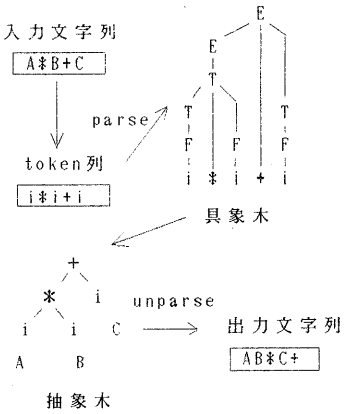


図３　中置記法から後置記法への翻訳

算術式の抽象木表現は、親が演算子・子が被演算子であるような簡単な構造を持つ木で表す。これを演算子モデルとよぶ。算術式は、この木のアンパースの仕方によって前置記法や中置記法や後置記法になる。算術式はこのような簡単な構造を持つ抽象木で構造表現される。

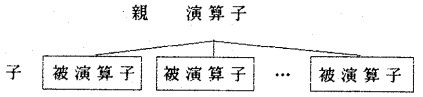


図４　演算子モデル

この演算子モデルは、単に算術式を表現するためだけではなく、プログラミングシステムの他の分野でも用いられている。

前述の構造エディタＡＬＯＥでは、エディタで作成されるＡＳＰＬＥプログラムを次のように演算子モデルの木構造を持つ内部表現に置き換えている。

```
while i <> 10 do
  a := a * i;
  i := i + 1
end
```

図５　ASPLEプログラムの１部分

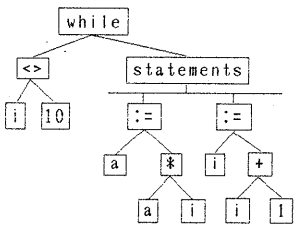


図６　ＡＬＯＥの内部表現

ここで、'while'，'<>'，'statements'，':='はオペレータと呼ばれ、それぞれが決まった形でオペランドを持っている。ＡＬＯＥでは、各オペレータに対して動作ルーティンを決めることによってインタプリタを実現している。

本論文では、日本語サブセットを、算術式やＡＬＯＥの内部表現のように簡単な構造を持つ抽象木にあてはめて取り扱う。

日本語の演算子モデルによる抽象木をつくるために、日本語文は抽象的演算子のｱﾝﾊﾟｰｽ列と考え、抽象的演算子をｱﾝﾊﾟｰｽの方法によって５つに分類する。（右肩の*は抽象的演算子の名前であることを示す。）

演算子名　：　ｱﾝﾊﾟｰｽの順序

1. 前置演算子：自身のｱﾝﾊﾟｰｽ，部分木１のｱﾝﾊﾟｰｽ，…，部分木ｎのｱﾝﾊﾟｰｽ　（ｎ≧１）

例　演算子　赤い･

被演算子　名詞をｱﾝﾊﾟｰｽする部分木

ｱﾝﾊﾟｰｽ結果　赤い　箱

演算子[赤い･]は、被演算子を１つ持ち、被演算子の色属性を「赤」にする演算を行い、ｱﾝﾊﾟｰｽの結果「赤い」を出力する前置演算子であると考えられる。

2. ２項中置演算子：左部分木のｱﾝﾊﾟｰｽ，自身のｱﾝﾊﾟｰｽ，右部分木のｱﾝﾊﾟｰｽ

例　演算子　と･

被演算子　名詞をｱﾝﾊﾟｰｽする部分木

ｱﾝﾊﾟｰｽ結果　箱　と　四角すい

演算子[と･]は、前後に被演算子を１つずつ持ち、二つの被演算子を１つのものと考える演算を行い、ｱﾝﾊﾟｰｽの結果「と」を出力する中置演算子であると考えられる。

3. 後置演算子：部分木１のｱﾝﾊﾟｰｽ，…，部分木ｎのｱﾝﾊﾟｰｽ，自身のｱﾝﾊﾟｰｽ

例　演算子　を･　（ｎ≧１）

被演算子　名詞をｱﾝﾊﾟｰｽする部分木

ｱﾝﾊﾟｰｽ結果　箱　を

演算子[を･]は、被演算子を１つ持ち、被演算子の対文属性値を「対象」にする演算を行い、ｱﾝﾊﾟｰｽの結果「を」を出力する後置演算子であると考えられる。

例　演算子　のせる･

被演算子　名詞句をｱﾝﾊﾟｰｽする部分木

ｱﾝﾊﾟｰｽ結果　箱を　台に　のせる

演算子[のせる･]は、被演算子をいくつか取り、被演算子をもとに他へ「のせる」働きかけをする演算を行い、ｱﾝﾊﾟｰｽの結果「乗せる」を出力する後置演算子であると考えられる。

4. ０項演算子：自身のｱﾝﾊﾟｰｽ

例　演算子　箱･　被演算子　なし

ｱﾝﾊﾟｰｽ結果　箱

演算子[箱･]は、被演算子を取らず、自分自身の属性を定数として返す演算を行い、ｱﾝﾊﾟｰｽの結果「箱」を出力する０項演算子であると考えられる。

5. 一般演算子：自身のｱﾝﾊﾞｰｽの前、途中、後に部分木のｱﾝﾊﾞｰｽをつなげる

例　演算子　もし･

被演算子　条件文、文をｱﾝﾊﾞｰｽする部分木

ｱﾝﾊﾞｰｽ結果　もし　箱　ならば　台に　のせる

演算子[もし･]は、間と後に被演算子を１つずつ取り、間の被演算子が真ならば、後ろの被演算子を値として返す演算を行い、ｱﾝﾊﾞｰｽの結果「もし…ならば…」を出力する一般演算子であると考えられる。

このように、抽象的演算子は５種類に分類することができる。又、日本語の単語は、この５種類の抽象的演算子のｱﾝﾊﾞｰｽの結果であると考える。

上記の定義を用いて、例文「赤い箱を青い箱にのせる」を演算子に対して被演算子をかっこでくくる形で表すと次のようになる。

([赤い･[箱･()]]を･[青い･[箱･()]]に･)のせる･

この記述を演算子モデルで表すと図７のようになる。

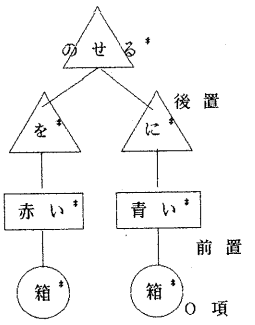


図７　日本語の演算子モデル

このように、日本語文を抽象的な演算子のｱﾝﾊﾞｰｽ列と考え、簡単な構造をもつ抽象木で表すことができる。

ここで、本論文で提案する演算子モデルによる自然言語処理における５原則について述べる。

１．具象木と抽象木の区別

1.1 具象木と抽象木を区別する。

プログラミングシステムの構造エディタ生成系や自然言語処理の格文法で用いられているように具象木と抽象木を区別する。ここで、具象木とは、文を構成する要素がすべて葉の位置に並ぶ木のことを言い、抽象木とは、要素が各ノードの位置にある木を言う。

1.2 抽象木は、親が演算子・子が被演算子の構造を持つ木とする。

演算子・被演算子の関係は、ｱﾝﾊﾞｰｽしたときｱﾝﾊﾞｰｽ列の中を移動できるものが被演算子である。抽象木を作るに当たっては、抽象的演算子の名前と種類と被演算子の個数、優先順位、結合性、ｱﾝﾊﾞｰｽ法などを決めなければならない。演算子には、被演算子の並ぶ順序が自由なものや被演算子の個数が不定個のものもある。又、被演算子の型には順序型や引数型がある。

２．具象文法とｱﾝﾊﾞｰｽ文法の区別

2.1 具象文法や具象木はとりあえず問題にしない。

具象文法や具象木は直接構成要素の構文構造表現である。

2.2 抽象木のｱﾝﾊﾞｰｽにより文が生成される。

2.3 抽象木のｱﾝﾊﾞｰｽ列を生成するｱﾝﾊﾞｰｽ文法を問題とする。

３．演算子のｱﾝﾊﾞｰｽ法による分類

3.1 単純演算子の５分類

演算子名　：　ｱﾝﾊﾞｰｽの順序

（@n は部分木のｱﾝﾊﾞｰｽ結果）

・０項演算子：自身のｱﾝﾊﾞｰｽ

ｱﾝﾊﾞｰｽ結果　string

・前置演算子：自身のｱﾝﾊﾞｰｽ，部分木のｱﾝﾊﾞｰｽ

ｱﾝﾊﾞｰｽ結果string @1 @2 ... @n (n≧1)

・２項中置演算子：左部分木のｱﾝﾊﾞｰｽ，自身のｱﾝﾊﾞｰｽ，右部分木のｱﾝﾊﾞｰｽ

ｱﾝﾊﾞｰｽ結果 @1 string @2

・後置演算子：部分木のｱﾝﾊﾞｰｽ，自身のｱﾝﾊﾞｰｽ

ｱﾝﾊﾞｰｽ結果@1 @2 ... @n string (n≧1)

・一般演算子：自身のｱﾝﾊﾞｰｽに部分木のｱﾝﾊﾞｰｽを埋め込む

ｱﾝﾊﾞｰｽ結果string0 @1 string1 ... @n stringn (n≧1)

3.2 複合演算子の単純演算子からの定義

複合演算子とは、いくつかの単純演算子からな

る式が１つの演算子としての働きを持つもの。

- ０項複合演算子　@body
- 前置複合演算子　@body @1 @2 ... @n (n≧1)
- 後置複合演算子　@1 @2 ... @n @body (n≧1)

4．抽象木の表示方法

4.1 １個の木による表示

単純演算子や、複合演算子による抽象木を１つの木で表示を行う。

4.2 １個以上の木による表示

複合演算子の@bodyにあたる部分木を別の抽象木を用いて表現したり、同一ｱﾝﾊﾞｰｽ列を持つ抽象森の数え上げに、１個以上の木による表示を行う

5．構文的曖昧さの定義

5.1 同一の入力列が２つ以上の異なる具象木を持つこと

5.2 ２つ以上の異なる具象木が同一のｱﾝﾊﾞｰｽ列を持つこと

最後に、図７の例文を用いて、演算子モデルによる構造表現を依存構造表現と表層格構造表現の２つの構造表現方法と比較する。

図８は、例文を依存構造表現で表したものである。

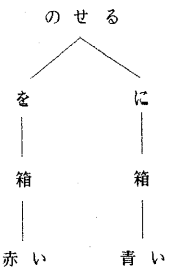


図８　依存構造表現

依存構造表現は、文の構造を語と語の間の依存関係によって捕らえたものである。子の位置にある単語は親の位置にある単語に依存していることを示す。

この依存関係は、日本語においては係り受けと呼ばれるものに等しい。　依存構造表現も、演算子モデルによる構造表現も、各枝の両端にある単語（演算子モデルでは、抽象演算子名である）の組合せが同じである。しかし、依存構造表現では、依存する単語が依存される単語の下にくるため、被修飾語の下に修飾語が位置する形になるのに対し、演算子モデルによる構造表現では、演算子の下に被演算子がくるため、被修飾語の上に修飾語が位置する形になる。

図９は、例文を表層格構造表現で表したものである。

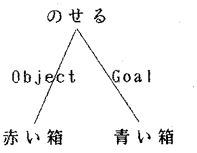


図９　表層格構造表現

格構造表現は、文中の動詞に対して他の単語がどのような役割（格）があるかを表したものである。この役割は、意味の上から各単語の立場を考えるものである。

表層格構造表現も、演算子モデルによる構造表現も、単語がどのような役割があるかを表している。表層格構造表現では、枝についているラベルが格を示す。演算子モデルでは、格演算子が演算動作を持っていることから属性操作が示す。この意味から、両者とも意味構造表現の世界へ立ち入っている。しかし、格構造表現では文中の単語がすべてノードとして現れていないため、例えば「僕は学校へいく」と「僕が学校へいく」のような心理的要素を含んだ文の構造表現にも差がでない。演算子モデルでは、単語を表示する抽象演算子がノード中に存在するため、演算の定義によって、心理的な要素等の拡張も可能である。

この２つの構文構造表現は、文の構造を表現してはいるが文の再生に関しての決まりがない。演算子モデルによる構造表現は、依存構造表現で表される単語と単語の関係を保持しながら、格構造表現で表される格の概念を演算子が演算を行うことで表しており、又、構文表現から元の文を再生することもｱﾝﾊﾞｰｽの手続きを用いて行うことができる。

3．演算子モデルの応用 I

第２章では、日本語文を５種類の演算子のｱﾝﾊﾞｰｽ列と考えることによって簡単な抽象木で構文構造表現できることを示した。

本章では、この日本語の演算子モデルの理論的応用として日本語の曖昧さについて演算子モデルを用いて説明する。

プログラミングシステムの分野では、３つの演算子（前置・中置・後置）の組合せは、一般には曖昧な文を生成することが分かっている[8]。

1)中置演算子の存在は単独で曖昧となる。

例　文法　S → S + S
　　S → a
　文　a＋a＋a

2)前置演算子と後置演算子の共存は曖昧となる

例　文法　S → f S
　　S → S g
　　S → a
　文　f a g

3)同一の前置演算子（または後置演算子）が異なる個数の被演算子を持つ場合は曖昧となる

例　文法　S → S －　（負の符号）
　　S → S S －　（減算）
　　S → a
　文　a a－－

このことは、日本語の演算子モデルの上でも成り立つ。例として次の文をあげる。

黒い目の女の子

例文の解釈としては、

(1)黒い目をした女が、生んだ子供

(2)黒い目をした、性が女である子供

のふたつがあげられ、直観的に意味的に曖昧であることがわかる。

ここで、黒い゛を「黒い」という単語をｱﾝﾊﾞｰｽする前置演算子、目゛，女゛，子゛をそれぞれ「目」，「女」，「子」という単語をｱﾝﾊﾞｰｽする０項演算子、の゛を「の」という単語をｱﾝﾊﾞｰｽする中置演算子と考えると、１）の条件からこの文は構文的に曖昧であるといえる。又、抽象木の数は、中置演算子の゛と前置演算子黒い゛の組合せであることから、「（黒い目）の女の子」で２つ、「黒い（目の女の子）」で２つの合わせて４つがあることが計算できる。

実際に、例の抽象木を書くと図１０のようになる。

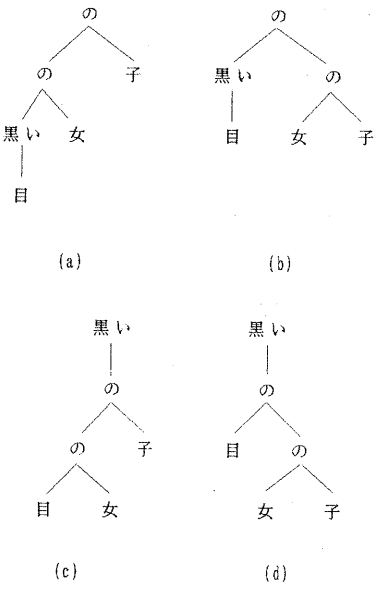


図１０　曖昧な文の抽象木

演算子モデルによる抽象木は、４通りできる。このことから、第２章の演算子モデルによる自然言語処理における５原則の5.1より、この文は曖昧であるという。４通りの木のうち、(a)と(b)が例文の解釈の(1)，(2)に対応する。また(c)と(d)は、意味の解釈ができない。実際の自然言語処理では、このような木の生成を抑える工夫がされる。

このように、演算子モデルを用いると日本語の曖昧さを説明することができ、また構文上の曖昧さの数を計算することができる。

4．演算子モデルの応用 II

本章では、実際に演算子モデルを用いてインタプリタを構成する方法について、属性文法を用いる方法とLispのＳ式を利用する方法の２つについて述べる。

4.1　属性文法によるＵＮＩＸコマンドawkの日本語インターフェースの作成

第３章では、日本語の構文的な曖昧さの理由を演算子モデルを用いて説明した。実際の自然言語処理では、この曖昧な文の処理についてDepth first searchとバックトラックを用いる方法や、Breadth first searchと並列スタックを用いる方法など、さまざまな方法が考えられている。

本節では、文法そのものを曖昧さの発生しないものにすることで日本語インターフェースを簡易的に構成することを示す。例として、ＵＮＩＸコマンドの１つである awk用の日本語インターフェースを、曖昧でない日本語サブセットを構成する後置演算子文法に属性と意味規則を加えた属性文法を用いることによって実現する。

後置演算子文法は、ｱﾝﾊﾞｰｽ列生成文法である。文法の構成方法は、つぎの通りである[8]。

＝後置演算子文法の構成方法＝

条件１）文法規則の形式は、次の形式のいずれかである。ただし、以下の文法規則のスキームにおいて、非終端記号は１つの特定の非終端記号Ｓであり、終端記号は、var_i, op_iである。また、a_{i1}, a_{i2}, …, a_{ik}は、終端記号列である。$(1 \leq i, j, k)$

非終端記号　Ｓ

終端記号の集合　ＶＡＲ∪ＤＥＬ∪ＯＰ

- VAR:変数の集合　$(VAR \cap DEL = \Phi)$
- DEL:区切り記号の集合　$(VAR \cap OP = \Phi)$
- OP:演算子の集合　$(DEL \cap OP = \Phi)$

文法規則

$S \rightarrow var_i$

$S \rightarrow S\ a_{i1}\ S\ a_{i2} \cdots S\ a_{ik}\ op_i$

$var_i \in VAR$

$a_{i1}, a_{i2}, \cdots, a_{ik} \in DEL^*$

$op_i \in OP$

すなわち、被演算子はa_{i1}, a_{i2}, …, a_{ik}という終端記号列で区切られ、被演算子と区切り記号列の並びの最後には演算子op_iが存在する。

条件２）各終端記号op_iは、その被演算子の個数が一意的に定まる。すなわち、同一の終端記号op_iに対し、その被演算子の個数はどの文法規則でも同一である。

このとき３つの集合ＶＡＲ，ＤＥＬ，ＯＰは互いに共通の要素を持たず、区切り記号の列 a_{i1}, a_{i2}, …, a_{ik} は特に制約はない。文法全体ですべて同一の終端記号でもよいし、また空列でもよい。

この後置演算子文法構成方法を用いて文法を構成する際、演算子として動詞やキーワードとなる名詞を用いる。図１１は、上記の構成方法を用いて構成された、曖昧さを含まない awkコマンド用日本語サブセット構成文法である。

```
1)  S → S の S を 出力する
2)  S → S の フィールド
3)  S → S 番目
4)  S → S と S の 和
5)  S → num
6)  S → string
```

図１１　曖昧でない日本語サブセット構成文法

この文法規則に意味規則を与え、属性文法としたものが図１２である。

```
1)  S → S の S を 出力する
    code(s0)=code(s1)+"cat"+place(s1)
          +"| awk '{print"+part(s2)+"}'¯%"
2)  S → S の フィールド
    part(s0)="$"+num(s1)
    pattern(s0)=part(s0)
3)  S → S 番目
    num(s0)=num(s1)
    pattern(s0)="NR == "+num(s1)
4)  S → S と S の 和
    part(s0)=part(s1)+" + "+part(s2)
5)  S → num
    num(s0)=pop(*num-stack*)
6)  S → string
    place(s0)=pop(*string-stack*)
    pattern(s0)=place(s0)
```

図１２　awk用日本語サブセットの属性文法

この属性文法を用いて、日本語文をawkコマンドに変換した後に実行した例が図１３である。

```
ﾃﾞｰﾀﾌｧｲﾙ file1
 1 2 3
 4 5 6      入力日本語文
 7 8 9      ・file1の１番目のフィールドを出力する
 9 8 7      ・file1の１番目のフィールドと３番目の
              フィールドの和を出力する
実行結果

  入力ローマ字日本文分かち書き
(file1 no 1 banme no field wo syuturyokusuru)

 awkコマンド   cat file1 | awk ' {print $1}'

 実行       1
            4
            7
            9

  入力ローマ字日本文分かち書き
(file1 no 1 banme no field to 3 banme no field
 no wa wo syuturyokusuru)

 awkコマンド   cat file1 | awk ' {print $1 + $3}'

 実行       4
            10
            16
            16
```

図１３　実行例

このように、awkコマンド用日本語インターフェースが属性文法を用いて簡単に構成可能である。

ここで、構文規則中の終端記号を tokenと考え、日本語文を一度、字句解析部で token列に変換することによって、同義語による表現の柔軟な処理が得られる。

例えば「出力する」，「書き出す」，「出す」は同じtoken 「出力する」に変換する。その結果、

file1の１番目のフィールドを出力する
file1の１番目のフィールドを書き出す
file1の１番目のフィールドを出す

は同じtoken列となり図１３と同じ実行結果が得られる。

4.2. LispのＳ式を用いたWinogradの積木の世界の日本語インターフェースへの応用

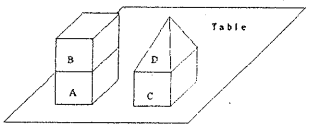
日本語の文は演算子のｱﾝﾊﾞｰｽ列であると考えて、日本語の文を算術式形式に変換し、その式から文の持つ意味動作・意味記述・質疑応答を計算することが可能である。　日本語の文の計算方法の１つとして、プログラミング言語Lispを用いる方法がある。

図７の木を適当に括弧をつけながら前置記法に変換すると次のようになる。

(のせる' (を' (赤い' (箱')))(に' (青い' (箱')))))

この形はLispのＳ式と同じ形で、このことから日本語の文はLispのＳ式に変換できることが分かる。そこで抽象演算子名を関数名とする関数をLispで定義すれば、このＳ式をLispインタプリタで評価することができる。

このことを、Winogradの積木の世界に適用してみる。



Ａ：赤い箱
Ｂ：白い箱
Ｃ：青い箱
Ｄ：赤い
四角すい

図１４　Winogradの積木の世界

辞書は、構文解析用と動作実行用の２つを定義する。

構文解析用の辞書には、例えば、「赤い' 」という演算子は、「前置演算子　被演算子－１つ」といったように、各演算子ごとにその種類と被演算子の数が記録されている。

```
(akai pre 1)      (aoi pre 1)
(hako zero)       (ni post 1)
(wo post 1)       (noseru post 2)
```

図１５　構文解析用辞書の例

動作実行用の辞書には、例えば、「赤い' 」という演算子は、「被演算子の色属性に対して、{赤}という属性値を代入する演算を行う」といったように、各演算子がどういう演算を行うかをLispのプログラムの形で定義している。

```
(defun akai (s) (cons '(color aka) s))
(defun aoi (s) (cons '(color ao) s))
(defun hako () '((katachi hako)))
(defun ni (s) (list 'basyo s))
(defun wo (s) (list 'taisyo-butsu s))
(defun noseru (&rest s)
  (let ((obj (second (assoc 'taisyo-butsu s)))
        (sup (second (assoc 'basyo s))))
       (if (listp obj)
          (setq obj (know-name object)))
       (if (listp sup)
          (setq support (know-name support)))
       (setq plan nil)
       (put-at object (get-space obj sup))
       (reverse plan))))
```

図１６　動作実行用辞書の例

この辞書を使って積木の世界を再現する。

日本語文

”赤い箱を青い箱にのせる”

は、

(1)構文解析部で辞書を用いて中間表現のＳ式に変換。

(2)そのＳ式を動作実行部で評価し、積木を動かす。

の二つの部分で処理される。

実行結果

日本語　(akai hako wo aoi hako ni noseru)

(1) 中間表現：LispのＳ式

(NOSERU (NI (AOI (HAKO))) (WO (AKAI (HAKO))))

(2) 積木の動き[23]

```
(C SUPPORT D)
(TYPE T IF FIND-SPACE SHULD SUCCEED)nil
((D WO TSUKAMU)
 (D WO (D NO TAME NO TABLE NO UE NO KUHAKU) E UGOKASU)
 (D WO HANASU)
 (B WO TSUKAMU)
 (B WO (B NO TAME NO TABLE NO UE NO KUHAKU) E UGOKASU)
 (B WO HANASU)
 (A WO TSUKAMU)
 (A WO (A NO TAME NO C NO UE NO KUHAKU) E UGOKASU)
 (A WO HANASU))
```

このようにして、演算子モデルを用いた日本語インターフェースを通して、Winogradの積木の世界を再現することができる。

５．おわりに

日本語文の構文構造を演算子モデルを用いて表現し、属性文法やLispのＳ式に変換することによって評価・実行することができることを示した。

この方法では、抽象的演算子名にプログラムの断片を与えるため、語句が多義である一般の日本語処理には不向きであると考えられるが、逆にコマンドインターフェースのような、語句の意味や用途が限定された範囲での応用に対して有効である。

現段階では、一般の場合の抽象的演算子の名前・被演算子の数・優先順位・結合性の決定、曖昧な文の処理などまだ多くの課題が残されている。

《参考文献》

[1] Aho,A.V. and Ullman,J.D. : The Theory of Parsing,Translation,and Compiling,Vol.1 and 2, Prentice-Hall (1972 and 1973).

[2] Aho,A.V. and Ullman,J.D. : Principles of Compiler Design,Addision-Wesley (1977).

[3] Aho,A.V.,Sethi,R. and Ullman,J.D.:Compilers principles,Techniques,and Tools,Addison-Wesley Publishing Company (1986).

[4] Aho,A.V., Kernighan,B.W. and Weinberger, P.J. : Awk,-a pattern scanning and processing language,Unix User's Reference Manual, 4.3 BSD University of California Berkeley

[5] Chomsky,N.:Aspects of the Theory of Syntax, MIT press (1957).

[6] Fillmore,C.: The Case for Case,in Bach,E. and Harms,R. (eds) Universals in linguistic Theory,Holt,Reinhart and Winston (1968).

[7] 淵一博　監修　古川康一・溝口文雄　共編 自然言語の基礎理論，共立出版，(1986)

[8] 萩原正也，徳田雄洋： 入力インターフェース用日本語サブセットの構成法とその応用，情報処理学会プログラム言語研究会87-PL-13，(1987)

[9] Knuth,D.E. :On the translation of Languages from Left to Right, Inf.Cont.,Vol.8,No.6,pp.607-639 (1965).
[10] Knuth,D.E. : Semantics of context-free languages, Math. Systems Theory, Vol.2, No.2 pp.127-145 (1968).
[11] Lewis,P.M.II and Stearns,R.E. : Syntax directed transduction,J.ACM,Vol.15,No.3,pp.465-488 (1968).
[12] Montegue,R: The Proper Treatment of Quantification in Ordinary English, Approaches to Natural Language. D.Reidel Pub. (1973).
[13] 長尾 真：言語工学、昭晃堂 (1983)
[14] Notkin,D. : The GANDALF Project,The Journal of Systems and Software, Vol.5, no.2, pp.91-106 (1985).
[15] Shank,R.C., Abelson,R.P. : Scripts, Plans and Knowledge, IJCAI 4 (1975)
[16] 田村直良、高倉伸、片山卓也：自然言語処理を目的とした属性文法評価システム、コンピュータソフトウェア、Vol.3,No.3 (1986).
[17] Teitelbaum,T. and Reps,T. : The Cornell Program Synthesizer, A syntax-directed Programming Environment,Comm,ACM,Vol.24,No.9,September, pp.563-573 (1981).
[18] Teitelbaum,T, Reps,T. and Horwitz,S. : The Why and Wherefore of the Cornell Program Synthesizer,SIGPLAN Notices, Vol.16, No.6, pp8-16, The Proceedings of the ACM SIGPLAN/SIGOA,Symposium on Text Manipulation (1981).
[19] Tomita, M.: An Efficient Context-Free Parsing Algorithm for Natural Languages and Its Applications,Computer Science Department Carnegie-Mellon University (1985).
[20] Wegner,P.: The Vienna definition language Computing Surveys 4:1,5-63.
[21] Winograd,T.:Language as a Cognitive Process ,Addison-Wesley (1983).
[22] Winograd,T.:Understanding Natural Language Academic Press,Inc, (1972)
[23] Winston,P.H. and Horn,B.K.P.: LISP,Addison-Wesley Publishing Company,Inc.,Rading,Mass., U.S.A.(1981)
[24] Woods,W. : Augumented transition network grammar for natural language analysis,CACM, Vol.13,pp.589-602 (1970).

盛光印刷所