利用者向き記述名表現法の名前管理向き表現への変換

2U－6

古宇田 フミ子
fumiko@mtl.t.u-tokyo.ac.jp
東京大学 工学部*

1 はじめに

既に提案した利用者向き記述表現法 [F94] は、一つの対象を異なる見方(表 4)により捉え、識別子だけでなく属性等の記述も可能である。この表現法の構造は、図 1からも読み取れるように、以下の特徴を持つ。1) 記述側面を単独又は、 and や or で、等位節として繋げたり、これらの関係を複合的に組合せたりできる。図 1の表現法の中で、$P\ Q$ は P and Q を、$P \mid Q$ は P or Q を表す。{・} は、記号を区別するための括弧で利用者表現には現れない。2) 一つの記述側面の要素としての記述要素も同様な and/or の並びの構造が可能となる (利用者表現法では、ST, Ph, CR, PA, Md が該当する)。3) 動作属性を説明する節の構造は、動作、関与者、周囲の三要素からなる (clause)。4) 関与者と周囲は、記述要素として、節の構造を再帰的に記述できる。p(・) や c(・) の中に現れ得る clause として表される。通常の文としてみると、これらは、関係代名詞節や to 不定詞節に相当する。5) 記述の省略が可能等、記述内容の柔軟性を持つ。図 1中、[・] で表される要素は省略可能であることを示す。

この表現法に対応するような管理向きの構造を、and/or の並びの構造、記述の省略等に対する柔軟性、再帰表現を持たせた形で、構成した。ここでは、利用者向き表現法から対応する内部表現への変換法について考察する。

2 表現法の変換時の問題点

利用者向き表現法では、括弧の対応、使われている記号の位置、使われている文字列としての単語の意味内容等から、表現全体を見れば、一意に区別でき、曖昧性がないことが分かる (ここでは詳細は触れない)。ところが、利用者表現から、入力文字列を見ながら「逐次的に」内部表現に変換しようとする場合、既に入力された文字からだけで意味を区別する必要がある。先が見えない制約がある。

利用者記述法では、なるべく用いる記号数を減らして、できるだけ簡単な記述で曖昧でなく表現できることを目指した。そのため、一つの記号で複数の意味を表しているものがある。例えば、括弧は、各記述要素を and/or で繋ぐときに用いられたものか、節 (clause) のまとまりの区切りとして用いられたのか両方の解釈が可能となる。又、他の例としては、関与者の記述 (p( ) と P( )) は、独立して属性機能に用いられる場合と、提供機能の要素として用いられる場合がある。両者は形式としては全く同じになる。

表 1: Descriptive Aspects and their corresponding symblos

	Aspects	Symbol
識別的記述側面	The Describing Aspects of Distinction	
識別子	identifier	id( )
使用法	usage	us( )
属性的記述側面	The Describing Aspects of Properties	
属性機能	attribute function	p( ) P( )
提供機能	providing function	(ST PA CR)
環境的記述側面	The environment Describing Aspect	
環境	environment	e( )

3 解決法

3.1 利用者向き表現から内部表現に変換するための方針

第一の問題、即ち、入力文字列を見ながら「逐次的に」内部表現に変換できるようにすることに対しては、入力文字を判別した段階でその文字がどの意味要素に対応するかを判断する方法をとる。

利用者が使用する文字は、記述側面を表す記号として、p(・), (・), id(・), us(・), e(・), c(・) があり、記述要素では、 Attr: や of:、また、役割を表す幾つかの名

*How to transform a user friendly descriptive name into its management form
Fumiko Kouda
Department of Information and Communication Engineering, Faculty of Engineering,
University of Tokyo
3-1 Hongo 7-chome, Bunkyo-ku, Tokyo 113, Japan

詞、冠詞(Al)、前置詞(Prep)、更に、 Ph や Cmd で用いられる文字列と |, > 等の演算子がある。

記号を見ただけで、意味が分かるものは多い。括弧は、clause の区別と and/or の組合せ表現の両方に用いられるので、見た時点では、一意に定まらない。未来の入力情報が必要になる。文字に関しては、文字の直前に示された Attr: 等の識別記号やそれ以前の記述要素や記述側面を表す記号で、区別できることが多い。

そこで、記述表現として入力された文字毎にその入力文字が持つ意味を判断し、後の入力で必要な場合に備えて、その意味を記録、保存する。特に、複数の意味を表す記号では、意味の可能性を列挙し、後の入力により分かった段階でその意味を判断する。ここでは述べてはいないが、利用者向き記述表現法は曖昧でなく一意に定まることから、この判断は必ず可能である。これは、第二の問題の解決法の一つとなる。

入力文字の値(記号)から記述の構造が分かる(例えば、入力が p(ならば、後に対応する) がくるはず)ことが多いので、記述の構造に対しても入力文字から分かる範囲で(構造の)記録をとる。このことで、例えば、現在見ている p(・) が clause の中の関与者なのか、独立した属性表現なのか区別可能となり、再帰構造も処理可能となる。

3.2 実現法

利用者向き記述表現の解釈には、字句解析と構文解析として、それぞれ lex と yacc を利用することとした。この理由は、lex を用いると文字の解釈が容易になること、yacc では、文字列の解釈として選択や繰り返しの構文規則を組み合わせることで記述要素の並び方の順序の不定性や自由な記述に対処できると考えられるためである。

yacc と lex に割り当てる処理は以下のように分けた。

lex 処理の段階で記述要素の and/or の並びの構造、節の再帰構造の記述、等の記述構造の解釈を行なう。このためにスタックと処理中の内容を示すフラグを用いる。

記述内容(即ち、記述要素の並び順や省略の自由度)は yacc の選択や繰り返しの構文規則を利用して解析する。

4 考察と今後の課題

当初、lex では、文字の識別だけを行ない、yacc において記述表現法のすべての(文字や記号の)並び方を判断させ、必要な情報を内部構造に書かせる方針でプログラミングを行なった。

その結果は、yacc で、315 grammar rules, 659states, となったが、同時に、clause に関する再帰構造の所や and/or の組合せの場所を中心として、conflict が生じてしまった(42 shift/reduce, 45reduce/reduce conflicts reported)。

後者については一組の内容が終りとなる右括弧の所に終了記号を入れることで、conflict を減らすことができたが、前者については良い方法が見つけられなかったので、3章で述べたような別の考え方を採り入れた。

現在、新たな版のコーディング中であり、 yacc 980 行、lex 480 行程度の大きさとなっている。このプログラムを完成させ、評価を行なうことが残されている。

参考文献

[S88] SUN Programming Utilities and Libraries lex, yacc pp.205 - 267, 1988

[F94] F. Kouda A new notation of user-friendly descriptive names, pp.493-498, ICON-9, 1994

```
SF ::= SFe | SFa | SFo | SFc          /*  記述表現 */

SFe ::= clause | PA | id | us | Env   /* 記述側面 */
clause ::= {( ST [PA] [CR] )} /* 提供機能に関する記述側面 */
PAe ::=  p([Al][Partname] Ph [Attr:Md] [of:Ph] [(Partname clause)]
       P([Al][Agentive:] Ph [Attr:Md] [of:Ph] [(Partname clause)] )
       p(Partname clause) | P([Agentive:]  clause)
id( Env: Ph )   /* 識別的記述側面: 識別子 */
us( Env: Cmd )  /* 識別的記述側面: 使用法 */
Env ::=  e( noun, Value ) | Env  Env /*  環境的記述側面   */

SFa ::= SFe SFe | SFe SFa | SFa SFe
SFo ::= { SFe | SFe } | { SFe | SFo } | { SFo | SFe }
SFc ::= {( SFa )|SFe }|( SFo )SFe |{ SFe|( SFa )}| SFe ( SFo )
        |( SFc )( SFc )|{( SFc )|( SFc )}

ST ::=  v | STa | STo | STc   /* v は一つの動作表現 */
STa ::= v v | STa v | v STa    /* v 同士の and 接続 */
STo ::= { v | v }|{ v | STo }|{ STo | v } /* v 同士の or 接続 */
STc ::= {( STa )| v }| ( STo ) v |{ v | ( STa )}| v ( STo )
       | ( STc ) ( STc ) | {( STc )|( STc )} /* and と or の組合せ */

Partname ::= Affected:|Agentive:|Recipient:|Resultant:|Attribute:|
      /* (以下、省略)  関与者の役割の種類 を表す */
Circname::= Locative:|Temporal:|Process:|Respect:|Contingency:|Degr
      /* (以下、省略)  周囲の役割の種類 を表す */
Prep ::= in:|at:|between:|from:|to:
Al ::= a | the |  | one
Ph ::= Phe | Pha | Pho |Phc     /* 記述要素:    Pha は and接続、
             Pho は or接続、Phc はこれらの混合、SF や ST と同様 */
Phe ::=  noun | ( noun  Value )
Value ::=  string | number | op(value)
op ::=  < | = | >
Md ::= atre | atra | atro | atrc /* 記述要素:    atra は and接続、
             atro は or接続、atrc はこれらの混合、SF や ST と同様 */
atre ::=  adjective phrase /* 修飾語 */

PA ::=  PAe | PAa | PAo | PAc /* 属性機能に関する記述側面 */
/* PAa は and接続、PAo は or接続、PAc はこれらの混合、SF や ST と同様 */

CR ::=  CRe | CRa | CRo | CRc  /*  記述要素: 周囲  */
/* CRa は and接続、CRo は or接続、CRc はこれらの混合、SF や ST と同様 */
CRe ::=  c([Circname][Prep] Ph [Attr:Md] [of:Ph] [(Partname clause)]
       c( Circname[Prep] clause )
Cmd ::=  command | command-line
```

図 1: A proposed notation of descriptive names