マルチメディア通信と分散処理 76－31
（１９９６．５．１７）

構造化され記述に自由度を持つ記述名表現の入力解析構成法

古宇田 フミ子
fumiko@mtl.t.u-tokyo.ac.jp
東京大学 工学部 電子情報工学科

属性や動作の説明をすることで対象の識別に用いる記述名を簡単な構造を持つ表現法として提案している。この記述表現法を仮定して入力解析を構成することを目的とする。

この記述名の入力解析では、通常の識別子の場合とは異なり、記述表現が構造化されている点や、記述成分の省略や語順に自由度があること等、考慮する必要がある。

この問題に対し、入力文字列を構造を表す記号と内容を表す値とに区切り、それぞれに状態名と記述表現の構造を表すスタックから成る状態空間を対応させ、入力文字列の状態遷移と入力解析処理が対応するように構成した。この結果、構造化され、自由に記述された各種の記述名に対処可能な入力解析が実現できた。

A syntax analysis of the descriptive name notation with simple structures and with capability of providing users with flexible descriptions

Fumiko Kouda
Department of Information and Communication Engineering,
Faculty of Engineering,
University of Tokyo
3-1 Hongo 7-chome, Bunkyo-ku, Tokyo 113, Japan

We propose a new syntax analysis suitable for the descriptive notation, which we have already proposed. It has a simple structure to explain the meaning of the context in the descriptive names and provides users with writing unrestricted descriptions.

These features bring the analysiser some difficulties.

To solve them, we separate the marks and content-terms in the descriptive notation, and assign each of them a state. The state consists of cartesian product of a state name independent of the structure and a stack space which reflects the structure of a description.

By making correspondence of the state transition taken out by a descriptive notation with a process of the syntax analysis, we can find a way to conquer the difficulties when analysing the structure and flexibility of a descriptive notation.

1 はじめに

記述名は、識別子を拡張した名前であり、対象の属性や動作、等の説明をすることで対象を識別しようとする名前をも含む(表1参照)。このような名前の拡張が必要となる理由の一つは、計算機資源の識別子自体が分からない時でも、その資源対象の持つ属性や動作ならば、一般的にも詳細にも、何らかの形で説明できることが多いので、属性や動作を説明することで識別に用いる名前があれば、代替手段として便利であると考えるからである。

ところが、このような記述名管理では、通常の識別子の名前管理と異なり、記述名として何を書くかという問題や、記述名を用いる時の利用者向き記述名の表現法やその入力解析法の問題が生ずる。更に、記述の多様性に応じた名前解決の方法も重要な課題となる。これまでに、第一段階として、筆者らは記述に必要な対象識別法を検討し、記述側面の考え方を導入し、各記述側面の記述の持つ意味構造に沿って通常の文に近い簡単な構造を持つ利用者向き記述名表現法を提案してきた[F94]。

本論文では、このような記述名表現法を仮定して、利用者向き記述名表現法で表された記述名の入力解析に焦点を絞る。入力解析の目的は、与えられた記述名から記述の構文関係を解釈し、対応する内部の管理テーブルに値を正しく格納することにある。通常の名前ならばこの写像操作は簡単にできる。ところが、記述名の場合は記述表現に簡単な構造があり、中には、同じ形で異なる意味を持つ構文もあることや、利用者には必要な記述を比較的自由な語順で書くことができること等を考慮しなければならない。更に、入力文字列は前向きに読まれ、後戻りできないという条件下で入力解析を行なわなければならない問題がある。

本文では、最初に、利用者向き記述表現法と管理テーブルについての仮定を述べる。続いて、入力解析の問題解決法を述べる。

2 前提とする記述名の捉え方とその表現法

2.1 利用者向き記述表現法

対象を識別する時にどのような見方が可能かという観点に立ち、対象を異なる観点から識別する。各々の観点を記述側面と定義した。ここでは、表1の種類だけを前提とする。

属性的記述側面では、性質の表現が重要である。属性機能では、対象の持つ性質を修飾語として表現した。被修飾語としての対象の一般名も表現することとした。性質の表現としては一単語で書けるもの、タイプと値の組で書くもの(例、データ長 = 100 Bytes)、動作的な節による修飾(例、ソケットに結びついている(ディスクリプタ:被修飾語))、の種類を考慮した。属性自体が動作的な節(that 節 ～すること)で表現できる場合は別の構造とした。

提供機能では、動作、関与者(participant)、周囲(circumstance)をvpc構造として表わすこととした。関与者の表現はその性質が重要となることもあるので、形態的には属性機能と同じものとなった。周囲は、動作を修飾する語(句、節)である。周囲も性質表現を用いて、属性機能の形態に状況を示す語が加わった形となった。

動作の主体の記述が必要な場合も生ずる。この場合、記述内容は関与者の記述と同様になるので、形態的には属性機能と同じものを用いた。

環境的記述側面は環境のタイプとその値を書く形(例、OS = Unix)とした。識別的記述側面はどの範囲で一意になるかの条件が必要な場合も考慮して、環境的記述側面を選択的に書けるような表現法とした。

2.2 記述表現法の構造

記述表現法の基本構造は、図1で表される。表現の要素として、「単語: 値」からなるプリミティブと、関数記号と左括弧、右括弧で囲まれた中に他の記述成分が含まれる複合構造がある。プリミティブの値には、一単語または、タイプと値の組がある。プリミティブの記号を表す「単語: 」には Attr: や of: がある。 Partname や Circname はこの記号だけで用いられる。値だけで表現されるものには、動作、被修飾語としての対象の一般名、がある。

複合構造には、記述側面と記述側面の構成成分となる記述要素がある。これらの「枠」は(・), P(・), p(・), c(・), e(・), id(・), us(・)の形で表される。各々、vpc構造、属性機能で主体、属性機能で関与者、周囲の記述要素、環境的記述側面、識別子、使用法、を表す。括弧の中には、プリミティブな構造や他の記述要素や記述側面が入る。

ここで、 P(・)は p(・) で、Partname = Agentive: の場合、つまり、属性として示される対象が主体となる特別の場合を表す。具体例として、速度が9600のプリンタは p(printer of:(speed 9600)) のように表される。

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( )

```
SFe ::= clause | PA | id | us | Env    /* 記述側面 */
clause ::= {( ST [PA] [CR] )}          /* 記述側面 提供機能 */
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) /* 記述側面 属性機能 */
CRe ::=  c([Circname][Prep] Ph [Attr:Md] [of:Ph] [(Partname clause)]) |
         c( Circname[Prep] clause )  /* 記述要素 */
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    */
Phe ::=  noun | ( noun  Value )      /*  Ph の基本要素 */
atre ::=  adjective phrase           /* 修飾語, Md の基本要素  */
```

図 1: Basic Components of Descriptive Aspects

```
P ::= Pe | Pa | Po | Pc
Pe ::= Basic Component
Pa ::= Pe Pe |  Pa Pe  |  Pe Pa
Po ::= { Pe | Pe }|{ Po | Pe }|{ Pe | Po }
Pc ::= {( Pa ) | Pe } | ( Po ) Pe | { Pe | ( Pa )} | Pe ( Po )
       | ( Pc ) ( Pc ) | {( Pc ) | ( Pc )}
```

図 2: Compound Components of Descriptive Aspects, Elements or Primitives

一つの記述側面の記述要素として他の記述側面と同じ構造を含むことがある。構造としては、再帰構造の形となる。ここで再帰的な記述とは通常の文における to 不定詞、動名詞、関係代名詞、等の従属節を構造化したものであり、関与者の記述や周囲の記述の記述要素となる。

記述要素、記述側面、プリミティブ、それぞれの記述成分同士が and で接続される場合や or でつながる場合を考慮している。and と or が混在する場合は括弧を用いて、その範囲を示している。区切り記号は、(), 接続詞は and は空白、or は | が用いられる (図 2)。

3 記述構造に対応した記述管理テーブル構成

記述管理テーブルの関係を図 3に示した。個々の記述側面の構造の関係を示す表を SFcp とした。記述側面が属性機能の場合は関与者の記述と同じ構造 (後述) になる。記述側面が提供機能の場合は動作、関与者の記述、周囲の記述からなる。この管理テーブルを Pf と呼ぶ。識別的記述側面は、環境と、識別子または使用法から構成される。管理テーブルをそれぞれ Ev, Id, Us とする。

関与者の記述では、二通りの異なる構成法がある。関与者の記述の管理テーブルを Pa として、Pa には (union を用いて異なる構造を指せる) ポインタを導入し、どちらでも指せるようにする。節 (clause) から成る構造は、RPf とする。RPf は、役割語、動作 ST、関与者の記述 Pa, 周囲の記述を成分として持つ。Ph と修飾語と役割語からなる構成では、節 (clause) の構成を除いて、一つの表 SPh とする。修飾語としての節 (clause) は、それ

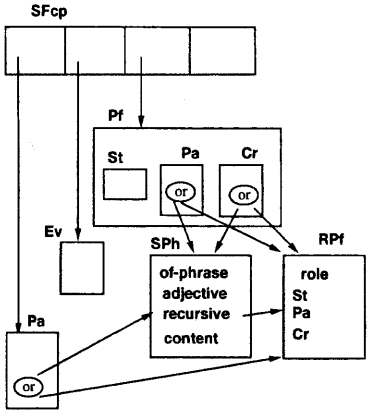

図 3: The table structure

自体構造を持つので、RPf を利用することとし、そのポインタを SPh に含める。SPh は、of 句、attr 句、Ph, RPf へのポインタから成る。

周囲の記述の管理テーブルを Cr とする。Cr は属性記述に状況を示す語が加わったものとなる。

これらの要素には、省略可能な要素もあるので、使用状況のフラグを管理テーブルに設ける。いずれのテーブルも同種のものが複数ある。

4 入力文字解析 — 利用者向き記述表現法から内部構造へ

4.1 入力文字列の区切り方

入力解析では、管理テーブルには文字列だけを書き込めばよい。記述表現に使われた記号は記述の構造を示すのでテーブルに書き込むための指標となる。このことから、入力文字列では、記号と (内容を表す) 値を区別して見る必要がある。

記述表現法では、記号の種類を少なくしたために値の意味が二重にとれることがある。このような場合には、入力時に値の意味を調べ、値の種類を分類することで解決を図る。

記述表現法では、Ph の記述に関して (A B) という並びがあった場合、1) A と B が同種類で A and B を表すのか、2) A がタイプで B がその値 (A = B) となるかの差が意味としての区別がつかない。これを避けるためタイプを表す語は入力文字列では区別する。

二番目として、入力文字列の並びが $p(Partname(A$ の場合、ここまでの入力文字列からは 1) A は Ph になるのか,2) A が動詞を表すのか区別がつかないという問題が生ずる。この問題には、入力文字列を区切る時に、動詞の値も区別することで対処する。

4.2 記述表現と状態空間

記述名表現の記号や値の繋がり方の関係や現在の立場を示すために、記述名表現を有限状態空間として捉えることを試みる。

入力文字列では、記号と値を区別し各々を異なるものとして見ているので、入力文字列の区切りごとに一つの状態を与える。

4.2.1 記述の束縛された関係と状態名

記述成分の構成に注目する。プリミティブの記号と値の繋がり方や、関数記号と左括弧、右括弧の対応関係の構成は分解できないし順序を変えることもできない。そこで、記述成分の「束縛された関係」を、記述表現の記号と文字列(単語)との組合せからなり、全体として一つのプリミティブ、記述要素、又は、記述側面を構成しアトミックな意味を持つ記述成分と定義する。記述の自由度は束縛された関係(以下、束縛態(bound group)と呼ぶ)を一つの単位として順序や省略の有無が自由にできることである。

入力文字列は、記号と値の文字列が別々に区切られるので、通常、一つの束縛態は複数の状態を持つ。状態は少なくともその束縛態に入った(入っている)、とその束縛態が終わった、の二種類を当てることができる。また、関数記号と括弧で囲まれた記述成分の場合は括弧の中には異なる束縛態が入ることもある。この構造は入れ子の関係となる。

プリミティブな記述成分の束縛態には他の種類の束縛態が中に入ることはない。括弧で囲まれた構造の場合 $f(s_1..s_i..s_n)$ は、f の束縛態の状態に入ったまま $s_i, 1 \leq i \leq n$ の各束縛態に次々と入ることになる。入力解析処理の関心は現在の入力の区切りにあるので、この場合の状態名は「f に入った」から「s_1 に入った」,「s_i に入った」に変わる。

ところが、中には、これまでの入力値からだけではどの束縛態に入ったかが明確でない状況が存在する。このような場合には二つの可能性のうち一つの状態を与え、後の入力で異なっていることが分かった場合だけ状態を修正することとした(図4)。

$f(s_1..s_i..s_n)$ の構造で s_i が終わった時の状態を見る。
I) 束縛態 s_i がプリミティブの場合は、外の束縛態 f に戻るので、「f に入っている」状態となる。

束縛態 s_i が関数記号と括弧からなる場合は、束縛態 s_i が終わったことの状態名の割り当て方には以下の二通りの可能性がある。

ある束縛態 S_j で右括弧が来て記述の入れ子構造の一つ外に出た場合、束縛態 S_j は終わったように見える。
II-a) その外の束縛態 S_p の構造が S_j と同じ束縛態になる可能性がある場合(スタック空間を見ると分かる。次節参照)は、束縛態 S_j と次に来る可能性のある記述成分 S_{j+1} が関連するので「束縛態 S_j が(部分的に)終わった」と言う状態名を当てる。束縛態 S_j 毎に異なる状態名になる。
II-b) 状態 S_j が終わり、外の束縛態 S_p が束縛態 S_j と異なることがスタック空間等から分かる時は、次に来る記述成分が束縛態 S_j に対するものと全く独立なので束縛態 S_j が終わった状態として「外の束縛態の構造 S_p の状態にある」と言う状態を与える。この場合、異なる束縛態 S_k に対しても同じ終了状態が割り当てられる。I)の場合もこれに当たる。

$f(s_1..s_i..s_n)$ の構造で f(に対応する)が来ると f の束縛態が終わるので、上記の II-a) または、II-b) の状態名となる。

4.2.2 記述の入れ子の関係とスタック空間

$f(s_1..s_i..s_n)$ の構造では関数記号と左括弧の記号の組と右括弧で囲まれた中に他の記述成分が含まれた入れ子構造になっている。入力解析では入力文字の区切りを次々と一方向に読み込む。逆戻りできないので過去の入力状況、即ち、束縛態の入れ子の前後関係、を記憶しておくものが必要となる。このような入れ子構造を反映させる構造としてスタック空間を対応させる。

スタックに積むものは入力の区切りすべてではなく(これは状態名で表現できる)、束縛態の種類を表すものだけとする。記述表現では記述成分の始まりを表す種々の記号に当たる。

スタックの積み降ろしは入力順に行なわれる。束縛態は記号で始まる。このような記号が来た場合、束縛態がプリミティブか関数記号と括弧の構文かを反映させて、スタックにおいてもプリミティブな記述成分ではその記号が来た時に一段積み、括弧が用いられた $f(s_1..s_i..s_n)$ の構造のものは、括弧を用いていることと記述成分の種類を明確にするために関数記号で表される種類と括弧の記号(f と( )を二段に積む。

束縛態がプリミティブの形の場合の終わりは値があることで分かるが、値がないこともあるので、次に異なる束縛態が来た時にスタックから取り出す。束縛態が関数記号と括弧の組からなる時は右括弧が来たら(と f をスタックから取り出す。入れ子関係なので)だけで対応する関数記号が分かる。同時に終了した束縛態の種類も分かる。

前節にあるように入力だけでは、どの束縛態に入ったかが一意に定まらない場合は、後の入力により、定まることを利用し、束縛態の種類が異なっていた場合には、スタックの値も修正する(図4)。

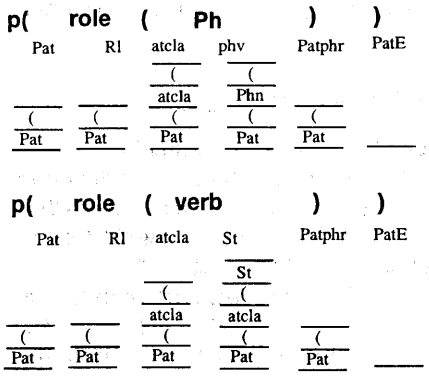

図 4: Amending states

4.2.3 入力文字の区切りと状態遷移

入力文字の区切りのをそれぞれを一つの状態に対応させると、入力文字の区切りの列に対応させられた状態は全体として状態遷移となる。状態遷移が起こるための入力は次の入力文字の区切りであり、次の状態はこの入力文字の区切りに対応する状態となる。

ここで状態は過去の入力の記述の構造とも関係するので、状態は4.2節で述べた現在を示す状態名だけでなく、過去の入力の記述構造を示すスタック空間との直積で構成されることとする。

状態名により現在の記述成分の種類が分かり、スタック空間により記述成分の入れ子の関係が分かることから、正しい状態遷移により束縛態の構成が逐次的に復元(対応付け)され得る。例えば、ある「束縛態に入った」状態で次の入力で文字が来た場合はその入力値はスタックで示される記述成分の入れ子の中にあり、その束縛態の記述成分の値を表すことが分かる。

4.3 入力文字解析処理法

状態遷移を利用して記述表現から文字列だけを目的のテーブルに格納する。処理としては書き込みテーブルのサーチとテーブルへの書き込みが必要になる。

入力文字の区切りとして記号が来た場合は記号がある束縛態の始まりならば、対応するテーブルの確保等をする。

入力文字の区切りが記述成分の終わりを示す記号の場合はテーブルの使用個数等の使用状態を確定する。

入力文字の区切りが記述表現法の値の場合は入力記号の処理で指定されたテーブルにその値を書き込む。

5 実装法と例題

5.1 実装法

実装には、字句解析 lex と構文解析 yacc[S88] を用いた。4.1節で述べた入力文字の区切り方に対応させて、トークン (token) を定めた [F96]。字句解析 lex では、文字列をトークンに分け、これを yacc に渡す。入力文字の区切りが値である場合はトークンとともに文字列そのものも yacc に渡すこととした。構文解析 yacc では lex で作られたトークンと (入力文字列) を受けてトークンに対応する処理をする。lex と yacc は対となり、トークンを作りだす毎に交互に動作する。yacc プログラムは繰り返し構文の中に選択構文を入れた簡単な構造とした。繰り返し構文の中では、選択の構文を用いて該当するトークンが選択され、このトークンに対する処理が実行される。このことが文の終わりまで繰り返される。このような処理の進め方により状態遷移が実現される。yacc での各トークンに対する処理は、4.3節に対応するものとなった。

5.2 例題

属性的な記述で動作を表す記述 (提供機能) を行ない、その使用法を問い合わせる記述名を取り上げ記述の語順について確認する。

通常の文で書くと、「What is the usage name with the following activity attributes ? transfer the header file of Unix OS, which relates to calculation, to a tape」となる記述文を調べる。

これを、(1) 英語的語順、「transfer the file to a tape」、(2) 日本語的語順、「ファイルをテープに送る」、(3) 日本語的語順、「テープにファイルを送る」、各々で表現し、入力解析の結果で生ずるテーブル構成を比較する。各々の記述表現は以下のように表される。但し、関数記号と括弧で囲まれた中が空の場合はこの項目に対する質問を表す記述表現である。

(transfer p(the file Attr: header of:(OS = Unix) (Agentive: (relate p(calculation)))) c(to a tape)) us( )... (1)

(p(the file Attr: header of:(OS = Unix) (Agentive: (relate p(calculation)))) c(to a tape) transfer) us( )... (2)

(c(to a tape) p(the file Attr: header of:(OS = Unix) (Agentive: (relate p(calculation)))) transfer) us( )... (3)

この結果は図5のようになった。テーブル構成は 1) と 2) は同じ構成、3) は用いたテーブルの順番は異なるが構成形態は同一のもので SFcp が 2 個、Pf は 1 個、SPh は 3 個、RPf は 1 個使われた。順番が異なる理由は入

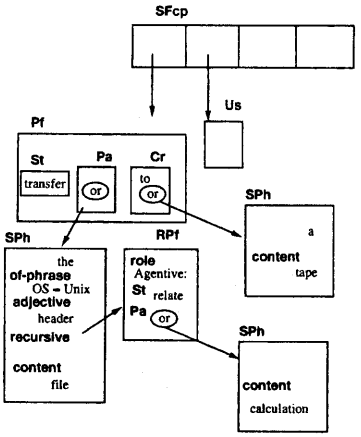

図 5: An Example

力順に文字の区切りに対応するテーブルが作られるためである。vpc 構造に関しては、入力解析は記述要素の語順に依らないことが分かる。

6 入力解析時の問題解決法に関する考察

6.1 状態空間の選び方と記述の自由度

入力解析処理用に記述表現に有限状態空間を導入した。この時、各々の状態を小さく分け、一まとまりの構造は状態遷移で、表されるように構成したので、いろいろなタイプの記述の語順や記述の省略に対しても対応可能となった。

状態名を記述構成の他の記述成分との関係とは独立に付けたので、どの記述構造であっても同じ状態名が用いられる。記述の入れ子構造はスタックを見ることで分かる。このことで記述の自由度に対処できる。

仮に記述構造に依存した形で状態名を付けたとすると、例えば、同じ動詞に対する状態名が、再帰構造の中と外、また、再帰の中の再帰で使われた動詞とで異なるものとなり、記述の自由度を実現するためには、状態名が大量に必要になることになる。同時に、入力解析処理で場合分け等が複雑になる。

6.2 記述の構造化と入力解析

記述表現を構造化したことで利用者側には記述の語順の自由度が実現できるが、入力解析においても構造化を利用できる。入力文字の区切りが記号の場合は、記号の種類に応じて、後から来る値を書き込むためのテーブル探しを行なっている。構造化していなければ、値の意味づけとテーブルサーチと書き込みとを同時に行なう必要がある。記述の構造化は入力解析にとってプラスに働くと考えられる。

6.3 入力順が一方向となること

入力文字の区切りが記号で、ある束縛態の始まりを示していれば、それをスタックに積むことで入力された記述の構造を記憶する。このことで、後からその束縛態の終わりを示す記号 (右括弧または、次の束縛態の始まり) が入力されるとその対応づけが可能となる。

スタックにはすべての入力情報を積むのではなく、記述の構造に関係する部分だけを積んでいるので、スタック空間を見ると、現在の記述構造の入れ子の状態が直ちに分かる。

スタックを利用したことで、記述の構造の処理には入力の一方向性は影響しない。

入力文字の区切りを記号と値とで分離し、記号でテーブルサーチをしているので、入力文字の区切りとして値が来た場合、直ちに、テーブルに書き込み可能となるので、入力が一方向性であっても問題はない。

6.4 同じ構造で異なる意味を持つ記述表現の自由度の支援

記述要素に文中の役割を示す記号を用いて構造化したのでこの記号化により束縛された関係を単位として記述の要素の語順に自由度が生まれる。例題で示したように英語的な語順で、動詞、目的語、状況を表す語の順としてもよいし、日本語的な状況語、目的語、動詞の語順でも書ける。

記述要素の語順で、まず、p(・) が来た場合はこれを独立した記述側面と捉えている。続いて、c(・) 又は、動詞が来ると記述全体は提供機能であるのが正解であることが分かる。入力解析では、p(・) は記述要素であるべきことが状態空間から判断されるので、いくつかのテーブルのポインタを付け替えることで、修正される。

7 おわりに

記述名の入力解析を行なう場合に問題となる記述の構造化、記述の自由度、同じ構造で異なる意味を持つ場合、入力順が一方向となること等に関しては、状態名とスタックの直積から成る有限状態空間を導入し、状態遷移と入力解析を対応づけることで解決を図った。各状態は小さな単位で表され、状態名は記述構造と独立につけられ、記述構造はスタックで表したので、種々の形態の記述表現に対処できるものとなった。

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

[F96] 古宇田 簡単な構造を持つ記述名の入力解析 第 52 回情報処理全国大会 1Aa-8, 1996