例文からの生成規則の自動修正とその属性文法への適応

3AH－6

中澤 聡　　佐藤 真一　　浜田 喬
東京大学　　学術情報センター

1 はじめに

ある一定の形式に従って作成されたテキストや、コマンドラインなどを文脈自由文法パーサを用いて構文解析することはよく見られる状況である。しかし、入力され得る全てのテキストの形式を正しく記述する文法規則を予め用意しておくことは難しい場合が多い。また、仕様の変更や、内容的にはほとんど同じでありながらこれまでとは異なる形式の入力を構文解析する際に、手動で文法規則を修正するのは大きな労力を必要とする。

以上のような問題に対して、これまで筆者らは、予め基礎となる文法規則を元文法として与えておき、以後、元文法では正しく構文解析できない修正用の例文が入力されると、それを受理できるよう新たな生成規則を自動的に追加していく、という手法を研究してきた [1]。

本稿ではさらに修正する文法規則の対象を文脈自由文法から属性文法に拡張し、属性の型情報を利用することにより、この手法の効率化を計る。

2 属性文法

属性文法では、各非終端記号は属性と呼ばれる変数を持っており、各属性の値は、文脈自由文法の生成規則１つ１つに対応する意味規則において、属性間の関数計算として求められる。図１は文字列 id をキーとする表に与えられた数値を記録していき、最後に尋ねられた id に対する数値の総和を開始記号 S の属性 S.sum として計算する属性文法の例である。

・属性の種類と型

```
int      S.sum , NI.val;
String   NI.name , A.name;
Table    I.tab , NL.tab;
```

・生成規則と意味規則

$S \rightarrow I\ A$
　{ $S.sum = call(I.tab, A.name)$ }
$I \rightarrow input\ NL\ ;$
　{ $I.tab = NL.tab$ }
$NL \rightarrow NI$
　{ $NL.tab = add(\phi, NI.name, NI.val)$ }
$NL_1 \rightarrow NI\ ,\ NL_2$
　{ $NL_1.tab = add(NL_2.tab, NI.name, NI.val)$ }
$NI \rightarrow id\ ,\ num$
　{ $NI.name = id$
　$NI.val = num$ }
$A \rightarrow ask\ id\ ;$
　{ $A.name = id$ }

図 1: 属性文法の例

3 修正処理の流れ

修正用の例文が入力された際の、処理の大まかな流れは図２のようになる。ただし、元文法、すなわち修正の基準となる属性文法は、それ以前に与えられているとする。

1. 例文が入力されると、まず元文法だけを用いて初期構文解析を行なう。例文には修正箇所が含まれてい

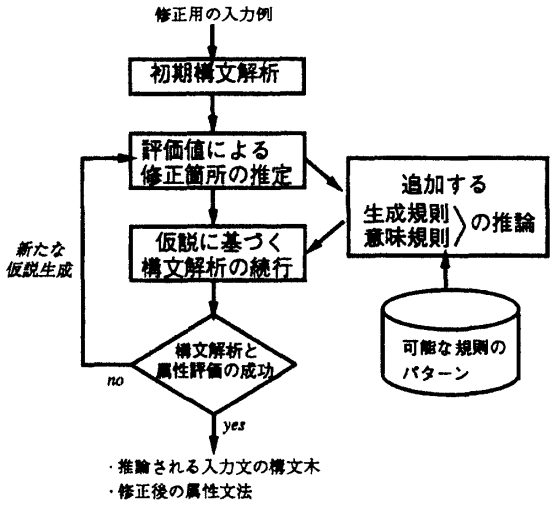

図 2: 例文からの修正処理の流れ

Automatic Modification of Production Rules by Example Sentences and the Application of the method to Attribute Grammars

Satoshi Nakazawa[1], Shin'ichi Satoh[2], Takashi Hamada[2]
[1]University of Tokyo
[2]National Center for Science Information Systems

るため、元文法だけでは構文解析は成功せず、途中で停止する。このとき、構文解析は例文中の全ての場所から可能な限りボトムアップに部分木を作成していくという手法をとる。

2. ついで例文中の**修正箇所の大まかな推定**を行なう。推定は、初期構文解析によって作成された不完全な部分木のおのおのに対して評価値を求め、評価値の高い部分木の足りないノードに当たる場所を修正箇所と判断する。もちろん、このような候補は複数存在する。
3. 推定された修正箇所の評価値の高いものに対して、**追加する生成規則、およびそれに対応する意味規則の仮説をたてる。**追加する生成規則の仮説は、既存の生成規則を元に右辺記号列の削除、位置交換、挿入、置換の操作を何度か施すことにより、局所的に修正箇所を満足させるようなものを選ぶ。意味規則も同様に、例文の属性値を局所的に満足させるような関数の組合せを選ぶ。
   推定された修正箇所に対して何通りもの生成規則が考えられるので、上記の削除、位置交換などの各操作ごとにコスト関数を設定しておき、修正箇所に適する生成規則の仮説のうち、このコスト関数が小さいものから順にパーサに渡していく。
4. パーサは渡された仮説ごとに ID をつけて、構文解析および属性評価を続行し、その仮説が局所的だけでなく大局的にも例文に適しているか確かめる。新たな文法規則の仮説を用いてもあまり構文解析が進まなければ、すなわち ID をつけられた仮説によって生成された部分木が増えなければ、処理を中断して、2. の手順まで戻り、また別の仮説を試みる。
5. その仮説を用いて構文解析および属性評価がうまく終了すれば、解析結果と追加する生成規則と意味規則の組を出力する。

この手順で用いている評価値とコスト関数は、最初に与えられる元文法から予め設定しておく[3]。

ここで手順 2. と 4 はループを形成する。このループにおけるバックトラックの回数が少ないほど望ましいため、次の節では、3. で推論する文法規則に制限を加える手法について述べる。

4 追加する文法規則への制限

本手法で取り扱う属性文法は、修正を通して

1. 非終端記号の種類
2. 各非終端記号が持つ属性の種類と型
3. 意味規則中に用いられる関数の種類

これらは不変であるという前提をとる。また、以上の情報は最初に元文法とともに与えられているとする。

一般の文脈自由文法においては、追加する生成規則の形に制限を設けることはできなかった。しかし、属性文法を修正対象としたときはこの前提から、生成規則に現れる記号の組に制限を加えることができる。

たとえば、図 1 の例では意味規則に

```
int    call(Table, String);
Table add(Table, String, int);
```

といったシグネチャをとる関数が用いられている。これらの関数は引数としてとる属性の型の組から、戻り値の型への変換手段と見なされる。意味規則で使用される関数の種類は有限であるという前提から、ある属性の組にこれらの関数を任意の回数、任意の順番で働かせて得られる属性の型も限られていることになる。これと 2 の前提から、生成規則の左辺の非終端記号の持つ各属性と右辺の記号列の持つ各属性とは、種類・数ともにこれらの関数で変換可能な組合せでなければならない。

実際、この制限は元文法が与えられた時点で、ある非終端記号を左辺とする生成規則の右辺に出現し得る記号列の組合せとして、完全に求めておくことができる。これにより、修正対象が文脈自由文法であったときに比べて、追加する生成規則の仮説数を押えることができる。

5 おわりに

実際、この属性の型の制限がどの程度有効かは、与えられた文法規則で宣言してある属性の型の種類と、関数の種類によって大きく影響する。今後は、属性評価によって増大する計算量とこの制限との関係を調べるとともに、等価な無数の文法規則の中から望ましい修正結果を選択する、他の有効な手法について検討する。

参考文献

[1] 中澤聡, 浜田喬. 適応パーサによる文脈自由文法の自動修正. 第 53 回全国大会講演論文集 (2), pp. 189–190. 情報処理学会, 1996.

[3] 詳しくは [1] 参照のこと。