# 5C−3　並列構文解析における関係節等の痕跡を含む文の処理

春野 雅彦　松本 裕治　長尾 真
京都大学工学部電気工学第二学科

## 1　はじめに

関係代名詞や疑問詞などを含む文の内部には痕跡が存在する。痕跡を探索しながら行なう構文解析法にはXG [1]、あるいはBUP-XG [2] などが考えられてきた。これらはバックトラックを行ないながら、一つのパスのみについて痕跡の探索を行なうものである。本稿では同時に多くのパスについて解析を進めるアルゴリズムを持つSAX (Sequential Analyzer for syntaX and semantiCS)[3] 上で痕跡を探索しながら構文解析を進める方法について論じる。

## 2　SAXの基本的動作

構文解析システムSAXはDCGで記述された文法をPrologのプログラムに変換し高速の上昇型構文解析を実現する。これは本来、GHCのような並列論理型言語上での並列構文解析システムとして考案されたものであるが、Prolog上の逐次型構文解析システムとしても効率よく働く。本節ではSAXの基本的動作について説明する。SAXトランスレーターは各文法規則を読み込むとその右辺に、それぞれ文法中での場所を示す識別詞を順に付けてゆく。SAXでは終端記号および非終端記号が全てPrologの述語として表現されており、上のように割り当てられた識別子をそれらがお互いに受け渡しすることにより徐々に大きな非終端記号を作りあげてゆく。一つのプログラムとしての(非)終端記号が複数の識別子を出すことから並行構文解析が可能となるのである。この動作を詳しく調べてみるとボトムアップチャート法と同じアルゴリズムを用いていることが分かる。SAXを用いて文を解析するには、例えば次のようなゴールを呼べば良い。ここでは隣接する語は共有変数で結ばれこれらが識別子の送受のためのストリームとして働く。

```
the([begin],X,[]),man(X,Y,[]),
walks(Y,Z,[]),fin(Z).
```

ここに、beginは文の先頭を表すための識別子である。beginという識別子はsentenceのみに理解されてendという識別子を生成する。finという述語はこのendという識別子を理解するための述語である。文の解析が成功するのはfinがendを受けとる時である。さらにSAXトランスレーターはトップダウン予測を行なうことによって不要な識別子の受渡しを押えている。以下で痕跡の位置を探索するのにこのトップダウン予測を用いている。

## 3　痕跡の処理方法

### 3.1　文法記述とその変換法

例として関係文に関する文法記述の概略を示す。

```
np --> np,relmarker(P),<sentence/P>.}
relmarker(np) --> np([wh=+]).
relmarker(pp) --> pp([wh=+]).
```

これら文法は関係詞relmarkerの後の文が痕跡としてそれぞれnp,ppをもつことを示す。ただし(wh=+)はnp,ppの中にそれぞれ関係詞が含まれていることを示す。この文法を読み込んだトランスレーターはrelmarkerから特殊な識別子slash(Cat)が出力されるようにすると同時に、入力されてくる全ての終端記号の間にprocess(図1参照)とよばれるprologプログラムをはさみこみ初期ゴールをつくる。これは前節であげたゴールの例で全ての単語の間にprocessをはさみこんだものである。slashについては例えばslash(np)なら、その後に痕跡としてnpが存在することを示す。またprocessはslash(Cat)を理解し痕跡を探すためのものであるがこれについては次節で詳しくのべる。参考のためrelmarkerのSAXプログラム例を次に示す。

```
rel_marker([],_) -->
        [].
rel_marker([Id|Tail],P) --> rel_marker_h(Id,P), !,
          rel_marker(Tail,P).
rel_marker([_|Tail],P) -->  rel_marker(Tail,P).
rel_marker_h(id7(InS),P) -->
             [id8(InS),slash(P)].
```

この場合、id7は上の文法規則の右辺の先頭のnpが解析し終ったことに対応する識別子でありrelmarkerはその直後に割り当てられた識別子id8以外にslashを出している。

### 3.2　痕跡の探索

この節で上記のprocessの動作について説明する。processは各単語の間に入って識別子の流れを見ている。またそれぞれprocessは次の単語を自分が担当する単語として記憶している。このprocessと単語の対応は、入力文から初期ゴールを作る時に自動的にとられるようになっている。プロセスは通常の識別子(slash(Cat)でない識別子)を受けとると最初に述べたSAXの動作を行なうためその識別子をそのまま次の単語(processが

---

**Handling gaps in wh-constructions with a concurrent parsing system**

*Masahiko HARUNO, Yuji MATSUMOTO, Makoto NAGAO*
Kyoto University

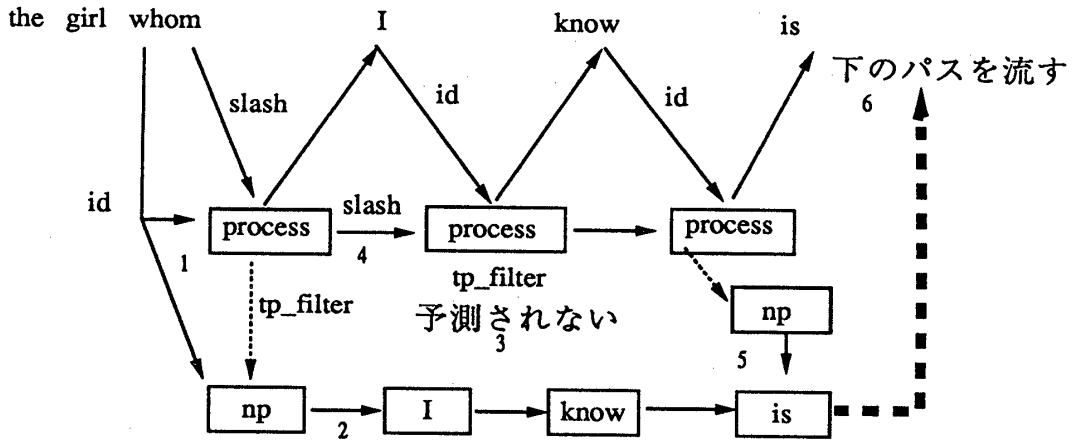


図1　processの概要

担当している単語) に流す。この場合は、間に process が入っているだけで普通の SAX の動作が行なわれている。次に process が識別子として slash(Cat) を受けとった場合について説明する。まず process はトップダウン予測を行なってその位置に Cat が続き得るかどうかしらべる。続けば通常の解析とは別に Cat に対応する非終端記号を生成し、二つの処理を同時並行に行なえるようにする (図1の1)。この時 process は Cat が出す識別子を自分が担当している単語に渡して解析を続けさせる (図1の2)。もちろんトップダウンに予測されていない時は何もしない (図1の3)。さらに slash(Cat) を受けとったプロセスは痕跡があることを次に伝えるために受けとった識別子 slash(Cat) をそのまま次のプロセスに送る。ここで process が自分の担当している単語ではなく次の process に slash(Cat) を伝えていることに注意する必要がある (図1の4)。こうすることで slash に関与するのは実質的に process だけになる。また二つ目以上の痕跡が予測された時はその出力を、それ以前に痕跡を挿入したパスとマージする (図1の5)。これはそれぞれのパスを独立に保ちながら不必要に枝分かれさせない働きがある。あらゆるパスを独立にしておくことで一つのパスには一つの痕跡しか含まれないことが分かる。このようにしておくことは他の文法ルール (特に並列句のものなど) との整合性を考える時に非常に重要となる。また痕跡の入っていない通常の解析がすべて失敗して slash 以外の識別子が出なくなった時には痕跡を入れないと解析できないということであるから process は二本のパスを一本にまとめる (図1の6)。process は上で述べてきたような動作を行なうことによって痕跡の位置を正しく推測してさらにそれを正しい結果として次に伝えることができる訳である。

### 3.3　解析結果

この節では文法ルール 20 程の簡単な文法を用いて痕跡を含む文を解析した結果をいくつか示す。

```
|: The man who gave it to the girl
   is a doctor.
parsed
execution time          = 20 msec
|: The man whom the girl gave it
   to is a doctor.
parsed
execution time          = 22 msec
|: I know who stole the purse.
parsed
execution time          = 19 msec
```

## 4　おわりに

以上 SAX おける痕跡の処理について述べてきたが最後にこれから考えなければならない問題について触れておく。

- 関係詞の省略の問題。目的格の関係代名詞や関係副詞はしばしば省略される。今のところ [ ] も関係詞扱いすることで処理しているがこれは効率の点からいって好ましくない。名詞連続を発見した時に関係詞のルールを起動できないかという点について研究を進めている。
- Ross の複合名詞句制約に関する問題。関係節の埋め込みの問題と合わせて考える。

## 参考文献

[1] Pereira,F: Extrposition grammars , AJCL,Vol.7, No.4 (1981)

[2] 今野 聡 田中 穂積: 左外置を考慮した構文解析システム, コンピューターソフトウエア, Vol.3, No.2 (1986)

[3] 松本 裕治 杉村 領一: 論理型言語に基づく構文解析システム SAX, コンピューターソフトウエア, Vol.3, No.4 (1986)