3N-3

GHCによる 学習機能をもつ Language-oriented Editor の実現

小原 忍、 武田 正之、 井上 謙蔵
(東京理科大学)

我々は、編集時に誤りを検出し、誤り処理をユーザとの対話形式で進められる Language-oriented Editor (以下 L.o.E.と略) を研究・開発し、成果を挙げている[1]。これは Prolog で実現されていたが、文法記述においてバックトラックを意識しなければならない等の問題点もあった。それらの問題点を解決すべく、並列処理を有効に用いた、学習機能を持つ L.o.E.を実現したので、ここに報告する。

1. L.o.E.の特徴

①対象言語の汎用性： L.o.E.は言語に対する知識を属性文法によって記述された文法から得ることで対象言語を特定せず、汎用性を実現している。

②GHCによる並列処理： L.o.E.は並列論理型言語GHC[2]によって実現されており、並列マシンによる実行速度の向上が期待される。更に、並列処理によって、1つの文に対して複数の解釈が存在するような文法に対する処理も容易に実現されている。

③文法記述の容易さ： L.o.E.は、対象言語の知識を与えるシステム管理者と、L.o.E.をエディタとして使用する言語利用者の2ユーザを対象としている。システム管理者が言語の知識を記述するのには属性文法を用いる。属性文法は言語の構文に意味規則を付加して拡張した文脈自由文法であり、知識が言語の生成規則ごとに独立に表現されるため、これら知識のモジュラリティ・拡張性が高く保たれる。また、属性評価器を機械的に生成することができるのでシステムの自動生成に向いている。システム管理者はL.o.E.の制御法などを意識せずに属性文法によって文法を記述することができる。

④強力な意味誤り回復： L.o.E.は解析途中で構文・意味誤りを発見すると、その箇所をユーザに指摘する。しかし意味誤りでは発見箇所と誤り原因箇所が一致していないことがあり得る。例えば、(例1)の Pascal のプログラムでは①の箇所で意味誤りが発見されるが、誤りの原因としては①の他に②③が考えられる。L.o.E.はこの原因箇所の究明をユーザとの対話形式で行っている。実際には、属性文法における属性評価には方向(合成・相続)があるので、その意味木を逆にたどることで実現している。

⑤学習機能： 意味木の探索で誤り原因箇所を究明した場合、その候補箇所が複数個存在したり、原因とはなり得ない箇所まで候補として抽出してしまうことがある。L.o.E.はユーザが行った誤り回復の結果、どの候補箇所から指摘するのが最も効果的かを学習している。

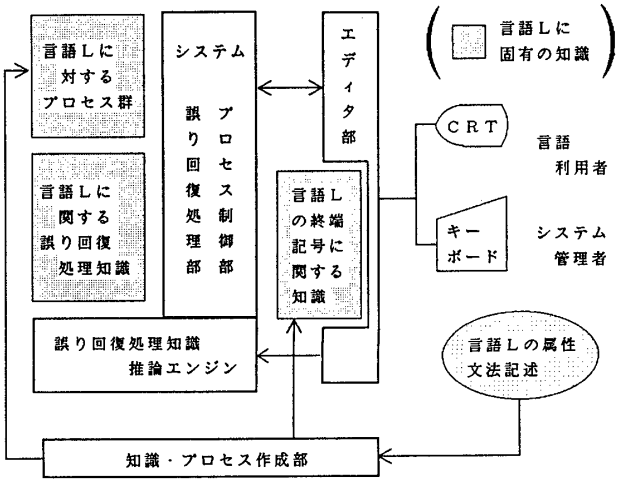
⑥誤り回復処理の例示からの帰納推論： L.o.E.では学習機能による誤り原因指摘箇所の優先順位の決定の他に、システム管理者が誤り回復処理情報を例示によって与えることができる。これによって、原因箇所が終端記号ではなく、ルールに依存している場合にも対応することができる(後述)。

⑦エキスパート・システム： 編集対象言語に対して未熟な言語利用者が、習熟者であるシステム管理者の与えた知識と対話形式で誤り回復ができることから、本システムはエキスパート・システムとしての役割を果たすこともできる。

(例1) 意味誤りの例 (Pascal)

```
program test ;
var x : integer ;
            ③
begin
    x := true ;
   ②    ①
end.
```

(図1) システム構成図



Language-oriented Editor with Knowledge Acquisition for Error Recovery
Shinobu KOBARA, Masayuki TAKEDA, Kenzou INOUE
Science University of Tokyo

2．システム構成

L.o.E.のシステム構成は（図1）のようになっている。

システム管理者が言語Lの属性文法記述を「知識・プロセス作成部」に与えると、そこでは言語Lを解析するのに必要なＧＨＣのプロセスを生成し、属性値を持つ終端記号の知識などを、語彙解析部を含む、エディタ部へ与える。このとき言語Lのルール1つに対し、1つのプロセスを対応させる。但し、ある非終端記号を左辺に持つルールが複数個存在する場合には、それらを管理するプロセスを1つ生成する。更に、終端記号、属性評価、文脈条件にもそれぞれ1つのプロセスを対応付ける。「プロセス制御部」は各プロセスと、直接または間接的に通信ストリームを共有している。構文・意味解析はこれらプロセスによって top-down に行われる。誤りが発見されると「誤り回復処理部」で意味木の探索などが行われ、ユーザとの対話形式で誤り処理が進められる。その際、言語Lに関する誤り回復処理知識が有効に用いられる。また、実行された処理の結果を更に知識として蓄える。回復処理が終了すると、必要な部分から再解析が実行される。

システム管理者が与える誤り回復処理の知識は、指摘された誤りに対しての原因候補箇所を例示として与える。この例示に対して「誤り回復処理知識推論エンジン」が働き、誤り回復処理知識を更新する。

3．プロセスの制御

解析が top-down に並列に実行されているとき、ある解析木が誤りを発見した場合、他の解析木が実行中ならばそのプロセスは中断しなければならない。そしてすべての解析木が中断状態に陥ると「誤り回復処理部」が処理を開始する。このときすべての誤り発見プロセスを中断させておくことは効率が悪いだけでなく、非実用的である。特にε規則が存在する箇所では中断プロセスが膨大な数になってしまう。そこで本システムでは、中断状態のプロセスが保持しているトークンの位置に対し、他の解析木のプロセスが成功すればその位置における誤り表示の必要はないと仮定し、中断状態のプロセスを消滅させる。例えば（例2）では 'begin expected' という誤りで中断しているプロセス④は、プロセス②が同じトークン位置で [var] に成功するので消滅してもよいと判断される。

4．誤り回復処理の例示

すでに述べたように、システム管理者が誤り回復処理情報を例示によって与えることができる。これによって学習機能では得られない誤り回復処理知識を得ることが可能である。学習機能では単に意味木をたどる方向の優先順位を決定するだけなので、意味木からはずれた位置に誤り原因がある場合は指摘することはできない。例えば（例3）ではルール②ではなく③を採択したためにルール④の文脈条件で誤りが発見されている。これはルールに依存する誤りなので意味木をたどって原因箇所を発見することはできない。このような誤りを例示として L.o.E. に与えることで、特別な誤り回復処理知識を得ることができる。

（例2）プロセス消滅の例

＜文法＞

```
program  -> [program], var_decl, block.   ①
var_decl -> [var], ident, [:], type, [;].  ②
var_decl.                                  ③
block    -> [begin], ...                   ④
```

＜プログラム＞

```
program
  var x:integer; ...
```

但し、[ ]は終端記号を表す。

（例3）ルールに依存した誤りの例

＜文法＞

```
program -> var_decl, statement.            ①
  where
    statement.flag:=var_decl.flag.
var_decl -> [var], ...                     ②
  where
    var_decl.flag:=true.
var_decl                                   ③
  where
    var_decl.flag:=false.
statement -> [use], id, [;], more_statement ④
  condition
    statement.flag = true.
```

＜プログラム＞

```
program
  use x;
```

但し、where と condition はそれぞれ属性文法、文脈条件を表す。

5．終わりに

本システムはＵＮＩＸのＫ－Ｐｒｏｌｏｇの ＧＨＣインタプリタ上で実現されている。例示からの誤り回復処理推論は1つの試みであり、実験による改良の余地はかなりあると思われる。

参考文献

[1] 武田正之：
知識ベースに基づく Language-oriented Editor
情報処理学会論文誌 Vol.28 No.11(1987)

[2] 古川・溝口 共編：
並列論理型言語ＧＨＣとその応用
共立出版(1987)