4J－5

意味再解析を考慮したGHCによるLanguage-oriented Editorの研究

山田雅彦　武田正之　井上謙蔵
(東京理科大学)

１．はじめに

我々は、編集時に誤りを発見し、誤り訂正処理をユーザーと対話形式で行なう Language-oriented Editor (以下 L.o.E. と略す)を研究・開発してきた[1]。これは、特定の言語に限定されることなく、対象言語に特有の構文・意味規則などの知識を利用することによって、編集の段階で構文・意味誤りの発見やその回復支援を行なうことでプログラム開発環境を強化し、ソフトウェアの生産性の向上を目的とした言語指向エディタである。

これまでのシステムでは、属性文法を限定節文法ＤＣＧで表現した記述に誤り診断・回復記述を付加したものを知識として蓄え、これに基づいた Prologのメタ推論によって実現されていた。その為、誤り診断・回復についての知識記述における負担が大きいこと、逐次処理の為、バックトラックを用いた誤り回復処理により、解析空間が大きくなることなどが問題とされていた。

そこで、本システムでは並列論理型言語ＧＨＣを用いることにより、構文解析と意味解析の並列実行、複数の誤りに対する指摘、再解析の範囲の縮小などを実現できるようにした。ここでは、意味解析及び再解析を中心に報告する。

２．システム構成

本システムの構成は、図１のようになっている。L.o.E.は、主にエディタ部と知識作成更新部の２つのシステムから構成され、対象言語Ｌの知識を与えるシステム管理者と、エディタを使用するエディタ利用者の２種類のユーザーを対象としている。

知識作成更新部は、知識作成支援部とＫＤＬトランスレータからなり、システム管理者が、対象言語Ｌの知識を知識記述言語ＫＤＬによって記述し、知識作成支援部は、この記述に関する属性の絶対非循環性などのチェックを行なう。又、ＫＤＬトランスレータによって構文解析モジュール，意味解析モジュール，誤り回復モジュールがＧＨＣの節の集合として生成される。

エディタ部は、知識作成更新部によって生成されたモジュール群とこれらを制御するモジュール・マネージャなどからなり、モジュール群によって誤り発見を行ない、マネージャによってエディタ利用者がテキストの編集を対話形式で行なえるようにしている。

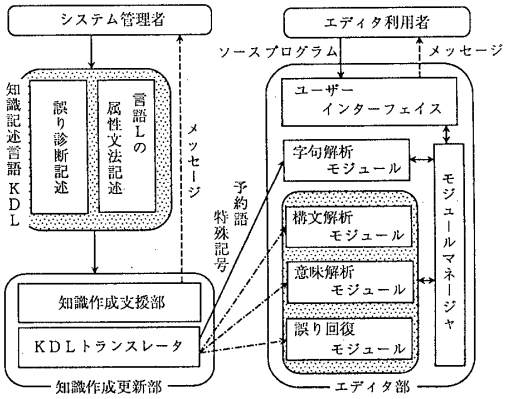

図１ システムの構成

３．解析の制御構造

このシステムでは、構文解析と意味解析が並列に、図２のような制御で行なわれる。意味解析は、構文解析と並列に生成される意味ネットワークによって行なわれ、構文解析，意味解析が終了すると誤りを発見し、その誤り診断を行なう。この時、構文誤り，意味誤りは、並列に発見される。エディタ利用者には、この診断結果を基に誤り原因箇所を指摘する。

その後、エディタ利用者によってテキストの変更が行なわれると、構文再解析及び意味ネットワークの再構築が行なわれ、新たな誤り発見及び診断を行なう。これらの再解析及び再構築は、必要最小限の範囲で行なわれる。

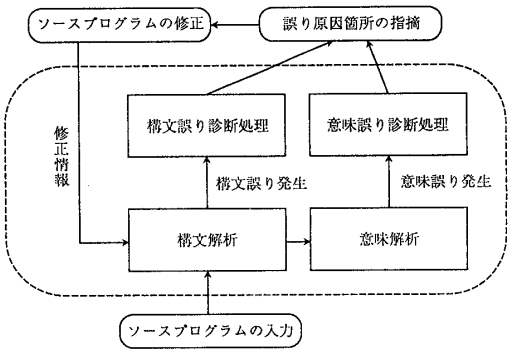

図２ 解析の制御構造

４．意味解析及び再解析

L.o.E.は、与えられたソースプログラムを解析することによって、構文と意味に関する誤りを発見し、その誤り原因箇所をエディタ利用者に指摘する。しかし、意味誤りについては、誤り発生箇所と誤り原因箇所が一致していないことがある。

Re-evaluation mechanism for Language-oriented Editor using GHC
Masahiko YAMADA, Masayuki TAKEDA, Kenzo INOUE
Science University of Tokyo

例えば、プログラム１のPascalのプログラムでは、①の他に②③が考えられる。

プログラム 1
```
program test ;
  var x : boolean ;
              ③
  begin
    x := 1 ;
    ②    ①
  end.
```

L.o.E.は、こういった誤り原因候補をエディタ利用者に提示し、エディタ利用者と対話形式によって誤りの原因を究明するものである。ここでは、以下に示すＫＤＬ記述に基づき、この意味に関する誤りを発見し、その原因候補を求め、テキスト修正後の再解析方法について述べる。

ＫＤＬ記述
```
program --> dcllist, stlist
  where stlist.env := dcllist.env.
dcllist(1) --> dcllist(2), [declare], id, [;]
  where dcllist(1).env := dcllist(2).env∪{id.tag}
  condition id.tag ∉ dcllist(2).env,
            {'二重宣言のエラー'}.
dcllist
  where dcllist.env := {}.
stlist(1) --> stlist(2), [use], id, [;]
  where stlist(2).env := stlist(1).env
  condition id.tag ∈ stlist(1).env,
            {'未宣言変数使用のエラー'}.
stlist.
```

まず初めに、エディタ利用者によって入力されたソースプログラム（Ｓ１）の構文木を生成する。この構文解析は、bottom-upに行なわれる。と同時に、属性の依存関係によって意味ネットワークを構築する（図３）。この意味ネットワークとは、意味規則，文脈条件などを１つのプロセスとみなし、属性の依存関係を表したプロセス・ネットワークのことである。このようにすることで、構文誤りは構文木作成の段階で発見され、意味誤りは、意味ネットワーク内で文脈条件を満たさないものから発見される。図３では、★印が意味誤り発生箇所となる。

次に、この意味誤り原因候補は、意味誤りの発生した文脈条件から意味ネットワーク内を逆にたどることで発見することができる。図３の場合、①,②,③が意味誤り原因候補となる。

そして、誤り原因候補が発見されるとそれをエディタ利用者に提示し、誤り訂正処理を行なわせる。エディタ利用者によってソースプログラムの修正（Ｓ２）が行なわれると、構文木変更処理が行なわれ、それに伴い意味ネットワークの変更が行なわれる（図４）。この時の意味ネットワークの付替点は、構文木変更処理によって発見され、再解析範囲（斜線部分）は、付替点から再解析を行なうことを意味ネットワーク内のプロセスに知らせることで決定する。この後、再解析が始まり誤り発見を行なう。

これまでは、ＫＤＬ記述中に誤り診断記述がない場合について述べてきた。誤り診断記述がある場合は、それに基づいて誤り原因候補を発見し、それをエディタ利用者に指摘する。

５．おわりに

本研究は、ＧＨＣを用いて Language-oriented Editor を実現することを目的としたものである。この Language-oriented Editor は、言語Ｌの知識を利用して、編集時に誤りを発見し、誤り訂正処理をユーザーと対話的に行なうものである。ここでは、その中の誤り原因候補の発見及び再解析範囲について述べた。また、本システムでは、構文解析において再解析の為に端記号プロセスのみを残し、プロセスの縮小なども行なっている。

並列処理による利点としては、１回の解析で複数の誤り原因の発見が行なえ、その誤り原因候補に対して誤り原因の可能性の強い順に提示することが容易にできることがあげられる。このシステムは、ＵＮＩＸのＫ－Ｐｒｏｌｏｇ のＧＨＣインタプリタ上で実現されている。

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)

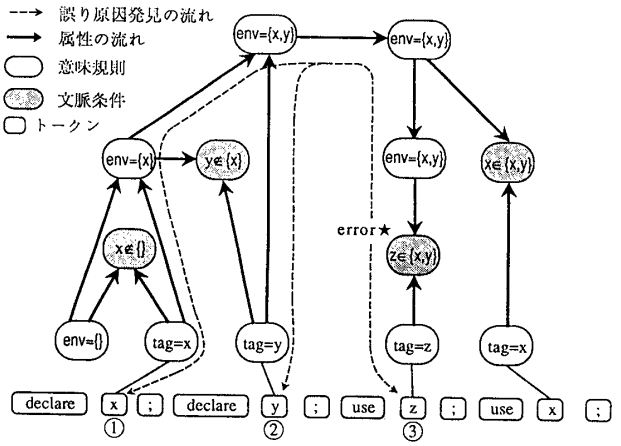

図３ ソースプログラム（Ｓ１）に対する意味ネットワーク

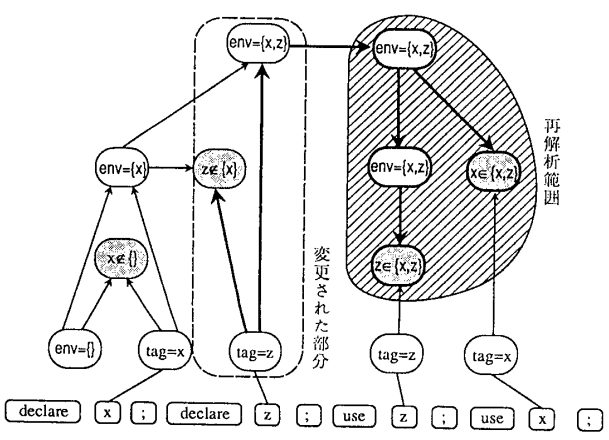

図４ ソースプログラム（Ｓ２）に対する意味ネットワーク