コンパイラにおけるインタラクティブなエラー処理方法の提案

5C−2

中井　央　　中田　育男
{nakai, nakata}@ulis.ac.jp

図書館情報大学

1. はじめに

従来、コンパイラのエラー処理の研究では、エラーの自動修正が目標であった。具体的には次の二つのことを行うものである。1) エラー発見後、残りの入力の中に更なるエラーがないかどうかを調べるためにエラーから回復すること、2) 現在のエラーに対し、ユーザに適切なエラーメッセージを提示すること。

通常、多くのプログラミング言語では、プログラムはファイルの先頭から末尾に向かって (左から右へ) 処理すればよいように設計されている。このため、実際に我々がエラーメッセージを見た場合は、経験に基づいて次のように処理を行う。

まず、最初のエラー発見個所の近辺をエディタ上で眺め、エラーメッセージを解釈し、修正を行う。

二つ目以降のエラーについても同様に修正を行う。しかし、各エラーメッセージは、コンパイラが引き起こした副次的エラーによるものである可能性がある。したがって、それまでの修正によって副次的エラーが解消されている可能性もあるので、すべてのエラーメッセージに対して修正を行わずに修正作業を一旦切り上げて、コンパイルを行う場合もある。

一般的なコンパイラでは、エラーが発見されても残りの入力の中にさらなるエラーがないか調べ、エラーメッセージを出力してコンパイルを終了する。上記のような左から右への解析を行うコンパイラではエラー発見個所までの解析は正しいものであると考えられるが、それらの結果は破棄されてしまう。このため、エディタによる修正後、再コンパイルの必要がある。

本研究では、エラーの発見時にユーザとインタラクティブにエラー修正を行い、それまでの解析結果と修正にしたがって解析を続行し、修正を施したソースプログラムに対する解析結果を出力するエラー処理方法を提案する。本稿ではその際の方針と問題点について述べる。

A Proposal for Interactive Error Processing Method on Compilers
Hisashi Nakai, Ikuo Nakata
University of Library and Information Science

2. インタラクティブなエラー処理

本研究では、次のようなインタラクティブなエラー処理方法を提案する。

まず、エラーを発見したら、そこで解析を中断する。そこまでに得られた解析結果とコンパイラの持つ情報から、そのエラーの修正の候補を提示する。ユーザはその候補の中から選んで修正を行う。ユーザの修正を受けて解析を再開する。ソースファイルをすべて処理し終わったら修正を施したファイルを別名で出力する。

この結果、プログラムの最後まで解析を行った場合、修正されたプログラムの解析結果を得ることができる。これが目的コードの場合、直ちに実行するあるいは次のフェーズへ渡すことが可能である。

エラー処理の観点から見ると、エラーの修正は次の二つに分けられる。

(i)　エラー発見時の先読み記号 (以降、これをエラートークンと呼ぶ) に対して何か操作を行うもの
(ii)　エラートークン以外の個所の修正

この節では、(i) の場合のインタラクティブなエラー処理について述べる。

以下では、LR(1) 構文解析を仮定する。

LR(1) 構文解析法は、プッシュダウンオートマトンによる解析法であり、各状態は LR(1) 項によって表現されている。LR(1) 項は、構文規則中のどの位置を現在解析しているかという情報と先読み記号の情報の組によって成り立っている。

コンパイラにおいて、エラーが発見されたとき、もっとも単純に考えられるエラーの処理の方法は、エラートークンに対して、なにか操作をすることである。その操作は次のものが考えられる。

(1)　エラートークンの直前にトークンを挿入する
(2)　エラートークンを他のトークンで置換する
(3)　エラートークンを削除する

これらの処理を行うためには、エラートークンの直前のトークンのシフト時の構文解析スタックを復元しなければならない。これはエラートークンによって影響を受けない最近の解析スタックが、エラートークン

の直前のトークンをシフトした時点であるからである。

LR 構文解析法では、現在の状態には、先読み記号として来てもよいトークンの種類の情報がある。(1) や (2) では、これらを提示し、ユーザが選択をすれば、エラーからの回復、コンパイルの続行ができるようにすべきである。実際の実装では、GUI などを用いてメニューの中からユーザが選択する形式となる。

また、実際の選択の候補中には、識別子や数字も含まれる。これらが選択された場合には、さらに実際の字面をユーザに入力してもらう必要がある。このとき、これまでの解析情報を用いることによって、現在のスコープから可視な識別子をユーザに提示することが可能である。逆にそのような提示をすることで、打ち間違いを含め、更なるエラーを引き起こさないよう配慮できる。

3. インクリメンタルなコンパイラによるインタラクティブなエラー処理

前節の最初に述べた、インタラクティブなエラー処理の二つ目、(ii) エラートークン以外の個所の修正、について述べる。

以下の二つの例はどちらもエラートークンより前のトークンが実際に修正される場所になる。

```
precedure foo()

procedure bar(var i:integer):integer;
```

両方とも Pascal のプログラム断片である。

前者は、procedure と打つところを打ち間違えたため、単なる識別子として扱われた。このため、識別子が二つ続くような文法規則がないため、エラーとなった。

もし、後者のエラーで:integer を取り除くことがプログラマの意図であれば、前節のエラートークンの削除を繰り返せばよい。ここでは後者は、プログラマの意図として、procedure ではなく function としたかった場合であるとする。この場合、エラーの修正を自動で行うのは難しく、:integer を取り除いたとしても、プログラマが function のつもりで書いたものであれば、値を返す時点で意味エラーを引き起こしてしまう。前方に戻って procedure を function に変更するのは難しい。

上記のエラーの類は、k 個前を修正するものといえる。ここでは左から右への解析を仮定したが、一般的に考えれば、エラートークンよりも前のすべての解析結果を対象に変更を加える場合であるといえる。

このような場合にはインクリメンタルな構文解析の技法 [2] を適用するとよい。インクリメンタルな構文解析とは、すでに解析された情報をコンパイラが持っていて、ソースプログラムに変更があった場合には、変更箇所から影響を受ける範囲のみを再解析の対象とする構文解析方法である。

4. 実　　現

属性評価器生成系 Rie[3] に手を加え、インタラクティブなエラー処理を行う属性評価器を出力するように改造した。ただし、インクリメンタルな構文解析機能は未実装である。また、Pascal-P4 コードを出力する Pascal-S コンパイラをこの生成系によって作成した。

実際にエラーを含んだソースファイルを上記の Pascal-S コンパイラに与え、対話的に修正を行い、コードの生成を行わせた。この結果によって出力されたコードを実行することができた。これにより、インタラクティブなエラー処理方法が有効であることを確認した。

5. おわりに

本発表では、コンパイラにおけるインタラクティブなエラー処理方法を提案した。

インタラクティブなエラー処理方法の利点は次のものである。

(1) エラー発見個所でユーザからの修正を受け、その修正に基づいてコンパイルが続行できる。

(2) このため、すべての入力が終わった段階で、修正を施したソースプログラムに対するコンパイル結果を得ることができる。

今後の課題は大きく二つある。一つはインクリメンタルな解析方法の実装である。もう一つは意味エラー処理の強化である。意味情報をどのように利用すれば、インタラクティブなエラー処理に有効であるかについてさらなる研究が必要である。すなわち、エラートークンよりも前のトークンの修正をしなければならないであろうことの判断と、その際にどの位置を修正すべきかの提示のために意味情報をどのように利用するか、である。

参 考 文 献

1) Aho, A. V., Sethi, R., and Ullman, D.: *Compilers: Principles, Techniques, and Tools*, Addison-Wesley, Reading, Mass., 1986.

2) 中井央, 山下義行, 中田育男: インクリメンタルな LR 構文解析の一方式の提案とその評価, 情報処理学会論文誌, Vol. 37,No. 3(1996), pp. 371–383.

3) 佐々政孝, 石塚治志, 中田育男: 1 パス型属性文法に基づくコンパイラ生成系 Rie, コンピュータソフトウェア, Vol. 10,No. 3(1993), pp. 20–36.