UMA EXPERIÊNCIA NA IMPLEMENTAÇÃO DE UM RECUPERADOR DE ERRO LR(1)

Mariza Andrade da Silva Bigonha
Roberto da Silva Bigonha

Departamento de Ciência da Computação
Instituto de Ciências Exatas
Universidade Federal de Minas Gerais
Caixa Postal 702
30.000 - Belo Horizonte - MG

SUMÁRIO

Apresentamos neste trabalho um esquema automático de recuperação de erros sintáticos dentro de um contexto de analisadores LR(1) baseado no método proposto por Burke-Fisher. O objetivo do método é diagnosticar todos os erros sintáticos encontrados durante a análise sintática sem apresentar erros que realmente não existam. O recuperador fornece mensagens de erro automaticamente, com possibilidade de substituição, remoção ou inserção de não-terminais na entrada ou na pilha. Além disso, a recuperação de erros é feita antes e independente de qualquer análise semântica do programa. Porém, o método não exclui o uso de informações semânticas na recuperação do erro. Este esquema automático de recuperação de erros sintáticos está incorporado a um Sistema de Implementação de Compiladores desenvolvido na UFMG. É inteiramente transparente ao usuário e produz resultados de boa qualidade sem aumentar significativamente o tamanho do compilador. Esse método não faz nenhuma restrição ao uso de reduções "default" e o uso das mesmas provou não prejudicar a recuperação de erro.

1 Objetivo e Abordagem Adotada

Anterior aos sistemas interativos para desenvolvimento de programas, era essencial a incorporação de algum método de recuperação de erro sintático nos compiladores profissionais. Isto porque era inadimissível que um compilador, funcionando em modalidade "batch", interrompesse a compilação após a detecção do primeiro erro de sintaxe.

Com a difusão de sistemas interativos, é, até certo ponto, aceitável um compilador sem recuperação de erro. Neste caso, a compilação pode ser interrompida após a detecção do primeiro erro,

devendo o usuário promover as correções necessárias e então reiniciar a compilação. Se o compilador for suficientemente rápido, o custo de re-compilação é perfeitamente suportável. Contudo, este enfoque apresenta a grande desvantagem de não permitir que a compilação continue, mesmo quando o usuário assim o deseje. Além disto, a ausência de recuperação de erro no compilador compromete a qualidade de suas mensagens de erro. O fato é que, embora existam aplicações em que se admita compiladores desprovidos de recuperação de erro, a disponibilidade desse recurso é certamente muito útil, mesmo nesses casos, porque no mínimo melhora-se a qualidade das mensagens e torna a operação desses sistemas mais flexível.

Tendo por objetivo um método que permitisse desenvolver um recuperador automático de erros sintáticos, que fosse ao mesmo tempo eficiente, prático e aplicável dentro de um contexto geral de analisadores sintáticos baseados em prefixo viável (Aho & Ullman, 1972), estudou-se vários métodos existentes, como por exemplo, (Burke & Fisher,1982), (Druseikis & Ripley, 1976), (Feyock & Lazurus, 1976), (Graham & Rhodes, 1975), (Graham & Haley & Joy, 1979), (Johnson, 1977), (Krol, 1981), (Levy, 1979), (Mickunas & Modry, 1979), (Poonen, 1977), (Rohrich, 1980), (Selzer & Melo, 1983), (Wirth, 1976). Dentre estes métodos, tomou-se como ponto de partida o proposto por Burke-Fisher (Burke & Fisher, 1982), o qual foi adaptado de forma a funcionar em conjunção com um analisador LR(1) baseado em tabela compactadas (Bigonha & Bigonha, 1983, Bigonha 1985).

2 Descrição do Método

2.1 INTRODUÇÃO

O método de Burke-Fisher supõe um contexto no qual um analisador LR, dirigido por tabela mantem um "buffer" de entrada, duas pilhas de estado (PS e PE). O "buffer" pode conter parte ou toda sequência de símbolos terminais ainda a serem processados na cadeia de entrada, e possivelmente símbolos da pilha. A pilha PE é uma pilha de trabalho. A pilha PS é usada para armazenar estados do reconhecedor sintático, sendo o estado corrente aquele que estiver no seu topo.

A rotina de recuperação de erro, é ativada quando, dado o estado do topo da pilha e o próximo símbolo (terminal) da cadeia de entrada, nenhuma ação sintática for legalmente possível. Neste caso, o símbolo terminal em análise, ou seja, o símbolo corrente, passa a ser visto como o terminal no qual o erro foi detectado.

A principal preocupação deste método é determinar a natureza do erro, ou seja, se o erro é simples ou não. Denomina-se erro simples, aquele que pode ser corrigido pela modificação de apenas

um símbolo na cadeia de entrada. Esta modificação pode assumir a forma de inserção ou remoção de um único símbolo, ou a substituição de um símbolo terminal por outro. Este tipo de correção é chamado correção simples. Se o erro não for simples, a sua correção pode envolver a remoção ou a inserção de um trecho de programa. O trecho removido pode preceder, seguir ou estar em volta do símbolo errado. O trecho inserido, se houver algum, é composto de uma sequência de terminais necessários para fechar um ou mais escopos. Este tipo de recuperação de erro é denominado recuperação de escopo.

Denomina-se escopo, construções aninhadas, tais como procedimentos, blocos, estruturas de controle e expressões parentetizadas. Escopos são delimitados por abridores e fechadores, os quais são pares de símbolos que iniciam e terminam, respectivamente construções sintáticas. Por exemplo, "(" - ")", "begin" - "end", "if" - "end if" são delimitadores típicos de escopo.

A recuperação do erro está dividida em três fases. A primeira fase trata a correção simples. Na segunda fase, a correção é feita via recuperação de escopo. E na terceira fase, a correção é feita através da remoção de um trecho do programa em conjunto com a inserção, remoção, substituição de um símbolo terminal. Antes de se iniciar cada fase copia-se o conteúdo de PS para a pilha de trabalho PE.

2.2 PRIMEIRA FASE DE RECUPERAÇÃO

O processo de se tentar corrigir um erro em um dado ponto é chamado de tentativa. As correções ou estratégias testadas numa tentativa obedecem a seguinte ordem: inserção de um símbolo, remoção de um símbolo, e substituição de um símbolo por outro.

Inserção refere-se à inserção de um símbolo terminal ou não-terminal antes do corrente. Substituição refere-se à substituição do símbolo corrente por um símbolo terminal ou não-terminal. Remoção refere-se à remoção do símbolo terminal corrente (somente terminais são removidos).

Inicialmente, tenta-se inserir, remover e substituir um símbolo com objetivo de se recuperar do erro a partir do símbolo corrente. Se nenhuma correção for possível, símbolos são retirados da pilha PE e colocados de volta em "buffer". Para cada um desses símbolos, testa-se novamente as três estratégias. Este processo se repete até que uma correção seja possível ou que o conteúdo do topo da pilha PE seja um abridor de escopo. O abridor de escopo encontrado neste caso marca o início de uma unidade gramatical que provou estar sintaticamente errada. Como o texto a ser corrigido está dentro desta unidade não há necessidade e não é desejavel que se desempilhe mais elementos da pilha além do abridor de escopo.

O conjunto de símbolos terminais que servirão como candidatos à inserção ou substituição são todos aqueles símbolos que poderão ser lidos a partir da configuração corrente do analisador sintático, ou sejam os "lookaheads" (Aho & Ullman, 1982) do estado corrente. Cada um destes candidados determina uma correção possível.

O critério usado para determinar se uma correção foi bem sucedida é baseado na distância que o candidato permitiu ao analisador sintático avançar na cadeia de entrada. Nesta implementação a distância máxima analisada de cada vez é dada por uma constante MAXCHECK. Para a inserção, substituição, todas as tentativas com cada candidato são sempre consideradas. Por exemplo, ao testar um conjunto de candidatos à inserção, se alguns destes candidatos permitirem ao analisador avançar a distância máxima MAXCHECK, para todos estes candidatos as correções são consideradas válidas e relatadas como tal. Entretanto, para a recuperação do erro, somente uma é escolhida. Por outro lado, pode acontecer que nenhum candidato permita ao analisador avançar a distância MAXCHECK. Neste caso, escolhe-se o candidato, se houver algum, que permitiu ao analisador avançar mais, desde que uma distância mínima estabelecida, MINCHECK, tenha sido observada. As distâncias mínima e máxima devem ser definidas de acordo com o contexto em que se está trabalhando. A distância mínima não pode ser muito pequena e deve garantir que o analisador avance sobre a entrada depois da recuperação. Dado que erros podem ocorrer próximos um do outro, a distância máxima não deve ser muito grande. Os valores 3 para MINCHECK e 24 para MAXCHECK têm produzido bons resultados (Burke & Fisher,1982, Bigonha, 1985).

2.2.1 ESTRATÉGIA 1: INSERÇÃO DE UM SÍMBOLO

Nesta estratégia, supõe-se que o usuário cometeu o engano de omitir um símbolo. Para testar essa hipótese, procede-se da seguinte maneira:

Para cada símbolo do conjunto "lookahead" da configuração corrente do analisador LR(1), insira-o em "buffer" antes do símbolo corrente e tenta-se avançar no analisador sintático. Se a distância percorrida com a inserção deste símbolo for superior ou igual a distância que se avançou com um outro símbolo do "lookahead", deve-se salvá-lo como o melhor candidato à correção até o momento.

Como exemplo do uso desta estratégia, considere a linguagem cuja gramática está descrita na seção 6. Para o trecho do programa abaixo o recuperador implementado produz as seguintes mensagens:

```
5    P : PROCEDURE(i : INTEGER)
6        a : BOOLEAN
7        BEGIN
8            i :=  * 1
9        END;
10   Q : PROCEDURE(h : INTEGER)
11       j : INTEGER;
12       i : INTEGER;
13       R : PROCEDURE(i : INTEGER)

-------------------!
*****    Inserido "id" antes do simbolo "*" da linha 8
*****    Outras correcoes possiveis:
*****    Inserir "cte"
*****    Remover "*"
*****    Substituir por "-"
```

2.2.2 ESTRATÉGIA 2: REMOÇÃO DE UM SÍMBOLO

Nesta estratégia supõe-se que o usuário cometeu o engano de escrever um símbolo a mais. Para testar esta hipótese, o seguinte critério deve ser usado:

Em primeiro lugar, verifica se o símbolo corrente é um terminal, pois, somente terminais podem ser removidos. Em seguida, partindo do símbolo à direita do corrente, tenta-se avançar na cadeia de entrada. Se, ao avançar, pelo menos a distância mínima MINCHECK for percorrida, considera-se o símbolo como um candidato à correção.

A análise sintática do trecho do programa abaixo, produz as seguintes mensagens de tratamento de erros.

```
18       BEGIN
19           i := j;
20           IF h = 0 THEN P(J)
21           ELSE IF h = 1 THEN THEN P(i)
22                ELSE R(k)
23       END
24   BEGIN
25       i := 0;
26       j := 1;
27       k := 2;
28       Q(k);
-------------------------------!
*****    Removido "then" antes de "id" na linha 21
```


2.2.3 ESTRATÉGIA 3: SUBSTITUIÇÃO DE UM SÍMBOLO

Nesta estratégia supõe-se que o usuário escreveu um símbolo no lugar de outro. Para testar essa hipótese, basta que se aplique o algoritmo da estratégia 1 descrito na seção 2.2.1 para o símbolo corrente, substituindo-o por cada um dos símbolos do "lookahead" da configuração corrente do analisador.

Por exemplo, a análise sintática o trecho do programa abaixo produz as seguintes mensagens correspondentes ao tratamento do erro segundo esta estratégia.

```
12   Q = PROCEDURE(h : INTEGER)
13       j : INTEGER;
14       i : INTEGER;
15
16   R : PROCEDURE(i : INTEGER)
17        b: INTEGER

--------!
***** Substituido "=" por  ":" da linha 12
```

2.3 SEGUNDA FASE DA RECUPERAÇÃO

A segunda fase de recuperação, somente é ativada se a primeira fase não obtiver sucesso. Esta fase de recuperação trata da recuperação de escopo, ou seja, ela cuida de fechar um ou mais abridores de escopo através da inserção de uma apropriada sequência de fechadores.

O conjunto de abridores de escopo e suas sequências de fechadores é dependente de linguagem devendo o usuário prover ao recuperador a definição desse conjunto para cada linguagem.

Nesta estratégia supõe-se que o usuário cometeu o engano de omitir um ou mais fechadores de escopo os quais devem ser inseridos. O fechamento de um ou mais escopos é feito da seguinte maneira: em primeiro lugar determina-se o conjunto de possíveis fechadores para cada abridor presente na pilha PE. Cada um destes fechadores constitue um candidato à correção.

Se o analisador não conseguir avançar sobre o candidato, o mesmo é rejeitado. Se o analisador avançar pelo menos a distância mínima MINCHECK, o fechador poderá ser usado para a correção do texto.

Se o analisador avançar somente sobre o candidado, então, deve-se, recursivamente, tentar fechar o proximo abridor na pilha. Este processo se repete até que não haja mais fechadores para o abridor em questão.

Como exemplo, para o trecho do programa abaixo, o recuperador produz as seguintes mensagens correspondentes ao tratamento de erro segundo esta estratégia.

```
16    R: PROCEDURE(i : INTEGER)
17         b: INTEGER
18       BEGIN
19              i := i + 1;
20              b := i
21
22    BEGIN

-------------¦
***** Inserido "end" antes da linha 22
      para fechar o "begin" da linha 18
```

2.4 TERCEIRA FASE DA RECUPERAÇÃO

Esta fase só é ativada se a segunda fase não obtiver sucesso. Esta fase é bem parecida com a primeira descrita na seção 2.2. Basicamente, a diferença entre as duas reside no fato de que nesta fase tanto os elementos da pilha como os símbolos contidos em "buffer" são descartados. Ou seja, a terceira fase não devolve símbolos desempilhados ao "buffer" e ainda remove símbolos terminais e não-terminais de "buffer" numa tentativa de se recuperar do erro.

Começando se com o terminal no qual o erro foi detectado, isto é, com o símbolo corrente, verifica-se se o analisador consegue recuperar-se simplesmente com a remoção do elemento do topo da pilha sintática PE em conjunto com uma das estratégias de inserção, remoção ou substituição. Não tendo sucesso, desempilha-se mais um elemento de PE, sem devolvê-lo à entrada, e repete-se o processo até que o topo de PE seja um abridor de escopo ou que uma correção tenha tido sucesso. Como na primeira fase, o abridor de escopo delimita o desempilhamento sobre o contexto à esquerda.

É importante ressaltar que nesta fase, por motivos de eficiência, basta que o analisador avance a distância minima MINCHECK+2, para que se considere recuperado do erro.

Se nenhuma correção foi possível com o símbolo corrente, este é removido, ou seja, o próximo símbolo em "buffer" passa a ser o símbolo corrente e restaura-se a pilha para a configuração inicial e repete-se o procedimento descrito acima.

Para o trecho de programa abaixo, o recuperador produz as seguintes mensagens correspondentes ao tratamento de erro ocorrido durante a terceira fase de recuperação.

```
12    Q : PROCEDURE(h : INTEGER)
13     j : INTEGER;
14     i : INTEGER
15
16    BEGIN (* nao e permitido comentario nesta linguagem*)
17         i := j;
18         IF h = 0 THEN P(j)
19       ELSE IF h = 1 THEN P(i)
20            ELSE P(k)

            ¦-----------------------------------------¦
***** Removido o texto da linha 16 coluna 14
      ate linha 16 coluna 60
***** Removido ")" antes de "id" na linha 16
```

3 Analisador de Erro

Como foi visto, o processo de recuperação de erro requer que se tente avançar sobre a entrada diversas vezes com o objetivo de determinar a correção mais apropriada. Neste caso, ao encontrar um erro de sintaxe, o analisador deve apenas informar situação de erro ao invés de invocar o recuperador, encerrar a análise quando MAXCHECK símbolos já houverem sido processados e nunca ativar as rotinas semânticas. De forma a tornar mais eficiente a análise sintática, criou-se um novo analisador, denominado DETECTAERRO, cuja função é determinar a distância percorrida por cada tentativa de correção de erro.

O analisador DETECTAERRO é essencialmente um analisador sintático LR(1) de onde foram eliminados os mecanismos de se atrasar reduções (Veja seção 4) e incorporado um método para determinação dos rótulos de transições eliminadas pelas reduções "default". Ressalta-se que símbolos não-terminais presentes em "buffer" são processados corretamente pelo analisador LR(1) apresentado em (Bigonha & Bigonha, 1983).

3.1 ESCOLHA DE UM CANDIDATO A CORREÇÃO

Na tentativa de se recuperar de um erro sintático, pode acontecer que mais de um símbolo ou mais de uma estratégia faça com que o analisador se recupere do erro em questão como foi mostrado na seção 2.2.

Para escolha da "melhor" estratégia, adotou-se o seguinte critério: Se o primeiro candidato encontrado for candidato à inserção, à remoção ou à substituição de um símbolo, então ele é escolhido para reportar a correção. Os demais candidatos, se houverem, são indicados como outras possíveis correções para àquele erro. Entretanto, se o primeiro candidado refere-se à estratégia recuperação de escopo, todos os fechadores de escopo inseridos com sucesso são reportados como correções possíveis. Neste caso, apenas um candidato é determinado pelo método.

3.2 ORDEM DE PREFERENCIA ENTRE ESTRATÉGIAS DE CORREÇÃO

Têm-se observado que a melhor ordem de se tentar corrigir um erro de sintaxe é: inserção, remoção e substituição (Burke & Fisher, 1982). Se a preferência for dada a substituição sobre a inserção, em geral a recuperação obtida não é satisfatória. Experiências mostram que mudando a ordem destas três estratégias a qualidade da recuperação em geral degrada. A causa provável para este tipo de resultado é que o erro mais frequente é omissão de símbolos. Contudo, deve-se desenvolver maiores estudos nesta área no sentido de estabelecer a ordem mais adequada.

3.3 INSERÇÃO E SUBSTITUIÇÃO DE SÍMBOLOS NÃO-TERMINAIS

No método de recuperação presente, considera-se a inserção e substituição por um símbolo não-terminal como desejável, pois, estas correções podem simplificar a recuperação. Entretanto, nem todo não-terminal deve ser considerado para não degradar a qualidade das mensagens emitidas pelo compilador. Especificamente, só se deve permitir a substituição ou inserção dos não-terminais autorizados pelo usuário.

4 Efeito da Recuperação de Erro no Analisador Sintático

Devido ao fato de reduções poderem ser feitas na pilha do analisador mesmo quando existe um erro de sintaxe na cadeia de entrada

e ao fato de reduções "default" serem usadas pelo método de compactação de tabelas (Bigonha & Bigonha, 1983) surgem os seguintes problemas:

a. Como restaurar a pilha sintática para a configuração existente após o último "shift", dado que reduções podem ter sido feitas indevidamente; em outras palavras, como neutralizar o efeito dessas reduções indevidas de forma a ter uma recuperação do erro mais precisa.

b. E como recuperar informações referentes aos rótulos de transições eliminadas nos estados com ações de redução, uma vez que reduções "default" podem ser usadas. Os rótulos dessas transições são necessários quando tenta-se recuperar do erro sintático.

Uma solução para o primeiro problema é atrasar as reduções como em Burke-Fisher (Burke & Fisher, 1982), ou seja, durante a análise sintática atrasa-se as reduções até que o analisador seja capaz de ler o próximo símbolo da entrada. A técnica usada para atrasar as reduções será discutida em detalhe na seção 4.1.

A solução para o segundo problema consiste em recuperar os rótulos eliminados através de análise do autômato de estados finitos correspondente à coleção canônica de ítens LR(0) (Bigonha, 1985).

Isto é possível porque rótulos de transições para estado de redução (Bigonha & Bigonha, 1983) são também rótulos de transições de estados que puderem ser alcançados após uma sequência de reduções "default". Em outras palavras, esses rótulos são símbolos que necessariamente deverão ser lidos após as reduções terem sido efetuadas. Assim, para recuperar rótulos eliminados pela introdução de reduções "default", deve-se deixar o analisador LR(1) avançar, escolhendo-se sempre o caminho via redução "default", até que se atinja um estado só de leitura. Os rótulos das transições que deixam todos os estados pelos quais o analisador passou pertencem ao conjunto dos rótulos procurados.

Para se determinar o primeiro estado de leitura atingido após reduções "default", basta inserir na entrada um "token" especial, não pertencente a linguagem, e ativar o analisador de erro. Quando um erro de sintaxe for detectado, o estado corrente no topo da pilha é o estado desejado e os estados visitados são exatamente aqueles desejados.

4.1 Novo Analisador LR(1)

O analisador LR(1) apresentado na Figura 1 incorpora os efeitos da compactação da tabela LR(1) e implementa a técnica de atraso de redução até o empilhamento do próximo símbolo.

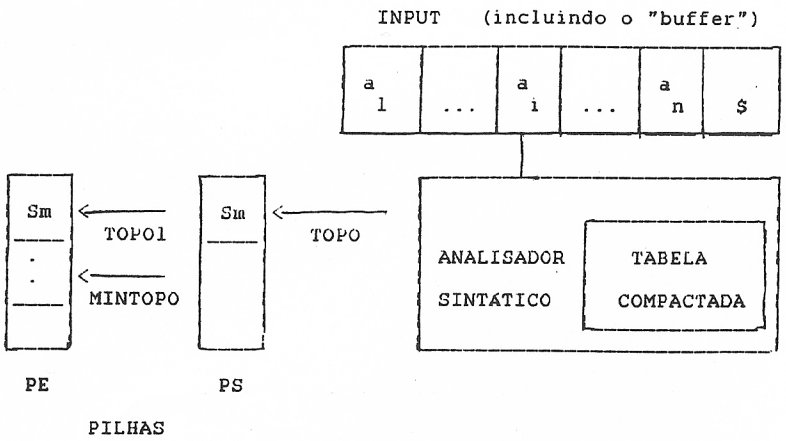

Figura 1 :Novo Analisador LR(1)

A pilha PE é uma pilha de trabalho. Como a pilha PS, ela é usada para armazenar estados do reconhecedor, sendo o estado corrente aquele que estiver no seu topo.

Durante toda a análise sintática, a pilha PE reflete exatamente a situação da análise. A pilha PS só é atualizada no momento em que se realiza empilhamento. Portanto, nem sempre reflete a situação corrente da análise. Reflete, porém, a configuração da pilha imediatamente após o último empilhamento.

Associadas a pilha PE têm-se as variáveis TOPO1 e MINTOPO controlando respectivamente o topo de PE e o menor valor que TOPO1 assumiu entre dois empilhamentos.

Ao ser detectada uma ação de redução, o número da produção que define a redução deve ser salvo e a pilha de trabalho PE passa a refletir de imediato a aplicação desta redução. Entretanto, a redução não é aplicada à pilha sintática PS e a rotina semântica associada também não é ativada.

Ao ser encontrada uma ação de "SHIFT", aplicam-se as reduções salvas desde o último empilhamento e as respectivas ações semânticas à pilha sintática PS antes de executar a ação de empilhar, tornando PS igual a PE. Deve-se ressaltar que todo símbolo lido é sempre empilhado simultaneamente nas duas pilhas PS e PE.

Quando uma situação de erro for detectada, o analisador sintático

deve voltar à configuração existente no momento em que o símbolo precedendo o terminal que detectou o erro foi empilhado. Ou seja, PE deve ter a mesma configuração de PS. Fazer PE igual a PS corresponde a desfazer as reduções feitas após o último empilhamento, eliminando, portanto, os efeitos de redução "default" incluídas.

Para se otimizar a restauração de PE, MINTOPO sempre aponta para o menor valor que o topo de PE assumiu após o último empilhamento. Como abaixo deste ponto as duas pilhas já são iguais, PE é aí cortada e apenas sua parte superior precisa ser restaurada.

5 Conclusão

O método proposto por Burke-Fisher serviu de base para a implementação do mecanismo de recuperação de erro de um Sistema de Implementação de Compiladores, o SIC, desenvolvido na UFMG. A parte deste sistema devotada a recuperação de erro compreende 1400 linhas de código Pascal. O usuário do SIC necessita apenas especificar a gramática da linguagem para a qual deseja-se implementar o compilador, sem ter que se preocupar com erros de sintaxes e respectivas mensagens de erro. Tanto a recuperação como a emissão de mensagens são inteiramente automáticas. Não se dispõe ainda de dados conclusivos em relação à velocidade do recuperador e a qualidade da recuperação que se pode obter. Até o presente momento os testes têm se mostrado bastante satisfatórios.

6 Definição da Gramática de Exemplo

```
program   =  prophead dcls cmdc;
prophead  =  "program" ;
dcls      =  dcl ;
dcls      =  dcls ";" dcl ;
dcl       =  "id" ":" "integer" ;
dcl       =  "id" ":" "boolean" ;
dcl       =  prochead  "(" par  ")" dcls cmdc ;
par       =  "id"  ":"  "integer" ;
par       =  "id"  ":"  "boolean" ;
prochead  =  "id"  ":"  "procedure" ;
cmdc      =  "begin" cmds "end" ;
cmds      =  cmd ;
cmds      =  cmds  ";"  cmd ;
cmd       =  "id"  ":="  exp ;
cmd       =  "id"  "(" exp  ")" ;
cmd       =  "if"  cond  "then"  cmd  "else"  cmd ;
cmd       =  "while"  cond  "do"  cmd ;
cmd       =  cmdc ;
```

```
cond    =  exp ;
exp     =  exp "+" exp   |  exp "=" exp    |  exp "/" exp
        |  exp "*" exp   |  exp "**" exp   |  exp "-" exp
        |  "-" exp       |  "(" exp ")"    |  "id"
        |  "cte" ;
```

7 Referências

Aho,A.V.,Ullman,J.D., The Theory of Parsing Translation and Compiling, Prentice-Hall,INC,1972.

Aho, A.V., Johnson,S.C., and Ullman, J.D. "Deterministic Parsing of Ambiguous Grammars". Comm. Assoc. Comp. Mach. vol. 18(8) 441-452 , August 1977.

Aho,A.V.,Ullman,J.D., Principles of Compiler Design, Addison-Wesley Publishing Company Co. (1973).

Bigonha, Roberto.S & Bigonha, Mariza A. S. "Compactação de Tabelas LR(1)", Anais do III Simpósio sobre Desenvolvimento de Software para Micros, 141-151, COPPE UFRJ - 1983.

Bigonha, Mariza A.S, "SIC: Sistema de Implementação de Compiladores". Tese de Mestrado, Belo Horizonte, DCC-ICEX UFMG, 1985.

Burke, M., Fisher Jr.,G.A.,"A Practical Method for Syntatic Error Diagnosis and Recovery",ACM ,1982.

Druseikis,F.C., Ripley,G.D.,"Error Recovery for Simple LR(K) Parsers", Proc. Nate. Conf. of ACM, Houston, TX,1976.

Feyock, S., Lazurus, P., "Sintax-directed Correction of Syntax Errors", Software Practice and Experience, Vol. 6, 1976.

Graham,S.L., Rhodes,S.P. "Practical Syntactic Error Recovery", CACM, November 1975.

Graham,S.L., Haley,C.B., Joy,W.N., "Practical LR Error Recovery",SIGPLAN Notices, August 1979.

Johnson,S.C., "YACC - Yet Another Compiler Compiler". Bell Laboratories, Murray Hill, 1977.

Krol,J.S.,"Simple Error Recovery Scheme for Optimized LR Parsers",TR 81-456, March 1981.

Lecaume, O., Bochmann, G.V. " A (truly) Usable and Portable C Compiler Writing System", IFIP74, 1974.

Levy, J. P.,"Automatic Correction of Syntax Errors in Programming Languages", Acta Informatica 4, 271-292, 1979.

Mickunas,M.D., Modry,J.A.,"Automatic Error Recovery for LR Parsers",CACM , June 1978,Volume 21,Number 6.

Poonen, G., "Error Recovery for LR(K) Parsers", Information Processing. August 1977.

Rohrich, Johannes, "Methods for the Automatic Construction of Error Correcting Parsers",Acta Informatica 13, 115 139,1980.

Setzer, V.W. e Melo, I.S.H., A Construção de um Compilador, Livros Científicos e Técnicos, Editora Campus Ltda.,1983.

Wirth, Nicklaus, Algorithms + Data Structures = Programs, 1976.

Wirth, Nicklaus, and Ammann, PASCAL = The Language and its Implementation, Edited by D.W. Barron; 65-73 1981.