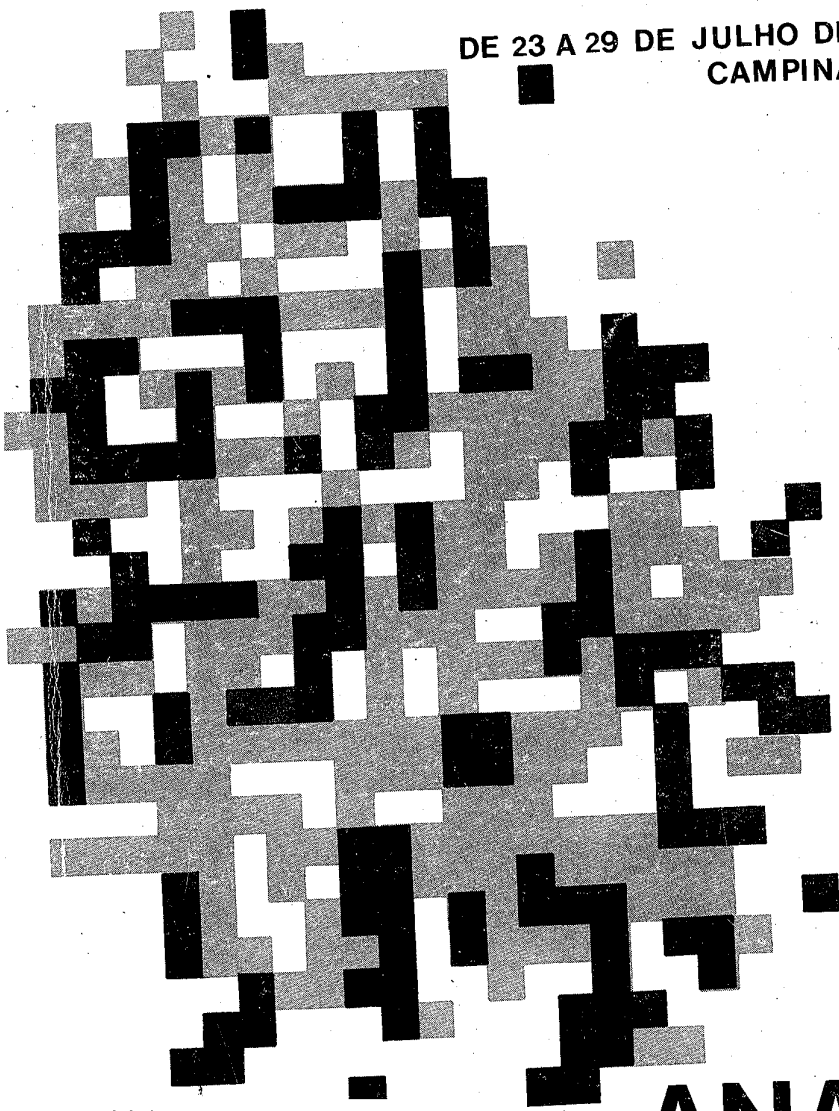
III CONGRESSO

DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO

DE 23 A 29 DE JULHO DE 1983
CAMPINAS - SP



ANAIS

VOL. I

III CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO
CAMPINAS 23 a 29 de julho de 1983

ANAIS
VOLUME I

TRABALHOS APRESENTADOS
X SEMINÁRIO INTEGRADO DE SOFTWARE E HARDWARE

EDITORES: N. MEISEL, L.J. BRAGA-FILHO

PROMOÇÃO: SOCIEDADE BRASILEIRA DE COMPUTAÇÃO - SBC
UNIVERSIDADE ESTADUAL DE CAMPINAS - UNICAMP

PATROCÍNIO: CAPES, CNPq, DIGIBRÁS, FINEP, SEI

CO-PATROCINADORES: CPqD/TELEBRAS, PUC-Campinas, UFV
PREFEITURA MUNICIPAL DE CAMPINAS

ANALISADORES SINTÁTICOS DESCENDENTES DIRIGIDOS POR TABELAS COM RECUPERAÇÃO AUTOMÁTICA DE ERROS

H.M.G. DE AGUIAR*, M.A.STANTON+

SUMÁRIO

O método de recuperação de erros em analisadores sintáticos descendentes recursivos proposto por A.C. Hartmann é adaptado para uso num analisador dirigido por tabelas baseadas em grafos de sintaxe. Mostra-se que é possível gerar automaticamente o analisador sintático com recuperação de erros dada somente a gramática.

ABSTRACT

A.C.Hartmann's automatic error recovery method for recursive descent parsers is adapted for use in a table-driven top-down parser based on syntax graphs. The automatic generation of table-driven parsers with error recovery is shown to be possible given only the grammars as input.

* Banorte Sistemas e Métodos, Recife, Pernambuco

+ Departamento de Informática, PUC/RJ, Rio de Janeiro

1. INTRODUÇÃO

Hartmann [HAR77] descreveu um analisador sintático descendente recursivo incorporando recuperação de erros para uma gramática LL(1), e mostrou como o esquema de recuperação de erros poderia ser gerado diretamente a partir da gramática, expressa no formato de grafos de sintaxe (veja, por exemplo, [WIR76]). O método de Hartmann foi analisado por Pemberton [PEM80], que sugeriu diversas melhorias. Como foi demonstrado por Wirth [WIR76], além do analisador recursivo, também podemos derivar de grafos de sintaxe um analisador descendente dirigido por tabelas no qual aquilo que depende da gramática sendo analisada é contido nas tabelas, que são neste caso uma representação direta dos grafos de sintaxe. É natural perguntar se o esquema de recuperação de erro de Hartmann não pode também ser aplicado aos analisadores dirigidos por tabelas, e mais, se esta aplicação não poderá ser gerada automaticamente, através de uma extensão do gerador de analisadores sintáticos de Wirth [WIR76]. O principal objetivo deste trabalho é responder no afirmativo a estas duas perguntas.

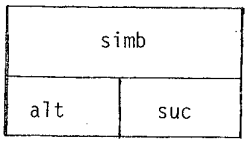
2. ANALISADORES DESCENDENTES DIRIGIDOS POR TABELAS

Wirth [WIR76] discutiu a análise sintática de linguagens de programação, e mostrou a equivalência de grafos de sintaxe e gramáticas na forma estendida de Backus e Naur ("extended BNF"), que é também conhecido como do lado direito regular ("right regular part" ou RRP). Ele também derivou analisadores recursivos e dirigidos por tabelas diretamente da gramática. Em termos práticos, os analisadores dirigidos por tabelas têm a vantagem de serem mais fáceis de adaptar a mudanças na gramática, uma vez que toda dependência da gramática está concentrada em tabelas, parametrizadas para cada gramática, enquanto num analisador recursivo uma mudança na gramática implica numa mudança na estrutura do programa do analisador. Logo, para um gerador de analisadores sintáticos, a forma dirigida por tabelas exibe uma clara superioridade. Finalmente, Wirth sugeriu uma representação concreta das tabelas sintáticas e um algoritmo recursivo para o analisador, que percorre os grafos de sintaxe, casando a entrada com a gramática. Setzer [SET79] apresentou uma forma iterativa (não recursiva) deste algoritmo, que usa uma pilha explícita para registrar quais não terminais da gramática estão ativos correntemente. Seu algoritmo é inteiramente equivalente ao de Wirth, e ambos utilizam a mesma representação das tabelas sintáticas, que resumimos a seguir.

Cada elemento dos grafos de sintaxe é representado por uma estrutura do tipo nó onde, com o uso de Pascal como linguagem de definição, temos

```
type ponteiro = ↑nó;
          nó =  record suc,alt:ponteiro;
                  case terminal:boolean of
                    verdadeiro:(tsimb:simbterm);
                       falso  :(nsimb:ponteiro)
                  end;
```

onde simbterm é um tipo enumerado que corresponde aos símbolos terminais da gramática. Podemos representar o nó graficamente por uma caixa com três divisões: um valor simb e dois ponteiros, suc e alt, que apontam, respectivamente, ao elemento sucessor, ou a uma lista de elementos alternativos.



(Note que simb tem duas variantes, tsimb e nsimb).
Os elementos nos grafos de sintaxe são símbolos terminais ou não terminais. Um símbolo terminal é representado pelo próprio valor, enquanto um símbolo não terminal é representado por (um ponteiro a) um grafo de sintaxe.
É fácil mostrar que os três construtores de expressões regulares (concatenação, alternação e fechamento) têm seus correspondentes em termos de grafos de sintaxe (sequência, alternação e iteração), e logo na representação considerada aqui (veja figura 1).

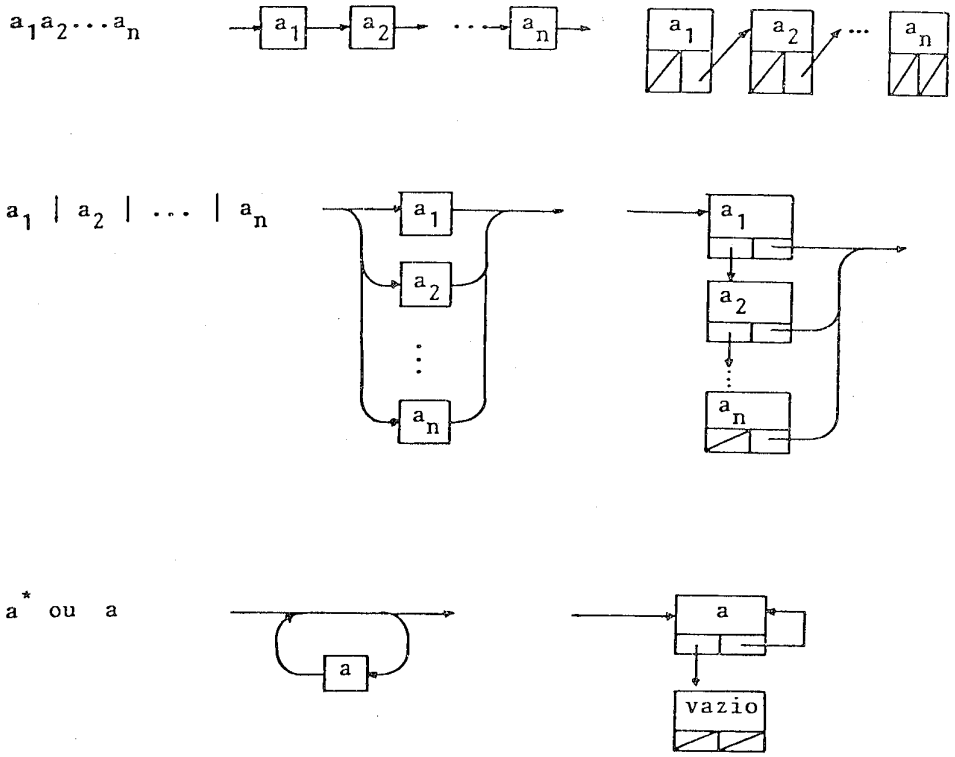


Figura 1: a correspondência entre expressões regulares, grafos sintáticos e tabelas de derivação.

3. O ESQUEMA DE RECUPERAÇÃO DE ERROS DE HARTMANN

A idéia básica é de recuperar de um erro de sintaxe lendo símbolos da entrada até encontrar um símbolo que pertence a um conjunto de símbolos admissíveis, chamados pontos de recuperação. Depois da ressincronização do analisador com este símbolo da entrada, a análise prossegue normalmente. O essencial do esquema é a escolha dos pontos de recuperação, que são derivados diretamente dos grafos de sintaxe da gramática. Num certo sentido, que esclareceremos na seção seguinte, os pontos de recuperação consistem de símbolos terminais que podem ser derivados de elementos nos grafos de sintaxe, elementos esses que podem ser alcançados a partir da posição corrente. Alguns destes pontos podem ser obtidos estaticamente a partir do grafo de sintaxe corrente. Outros, contidos em outros grafos de sintaxe, são determinados dinamicamente através do estado atual do analisador sintático. Estes últimos pontos de recuperação compõem o que Hartmann chama o contexto.

Detetamos um erro de sintaxe quando o próximo símbolo da entrada não pode iniciar uma cadeia derivada da posição corrente do grafo. Quando isto ocorre, fazemos recuperação de erro como descrita acima.

Hartmann introduziu seu esquema de recuperação de erro no contexto de um analisador descendente recursivo, e, nesta forma, várias melhorias foram propostas por Pemberton [PEM 80]. Parece natural aplicar o esquema aos analisadores dirigidos por tabelas que são derivados dos grafos de sintaxe. Na próxima seção, descrevemos as modificações a um analisador dirigido por tabelas, que são necessárias para incorporar o esquema de Hartmann.

4. O ESQUEMA DE HARTMANN APLICADO A ANALISADORES DIRIGIDOS POR TABELAS

Antes de apresentar o analisador dirigido por tabelas incorporando recuperação de erros, torna-se conveniente introduzir uma notação que facilite a discussão e represente claramente nossas idéias.

Consideramos os grafos de sintaxe compostos de nós que são ou simples (correspondendo a elementos terminais e não terminais) ou compostos (correspondendo a alternações e iterações na gramática). Chamamos os nós simples de elementos e os nós compostos de componentes.

Suponhamos que, num dado momento durante análise sintática, alcançamos o nó N. Definimos A(N), o alcance de N, que consiste daqueles nós do conjunto de grafos de sintaxe, que podem ser alcançados a partir de N. Observamos aqui, que, no caso específico de uma alternação ou iteração, consideramos o subgrafo correspondente como um único nó composto para os fins desta definição. Deveríamos notar que A(N) é composto de $A_{local}(N)$, o alcance local de N, que consiste daquela parte de A(N) contida no grafo de sintaxe corrente, o qual corresponde ao elemento não terminal no topo da pilha, e $A_{global}(N)$, o alcance global de N, que consiste da união dos alcances locais de todos os outros elementos não terminais na pilha. $A_{local}(N)$ é determinado estaticamente a partir da gramática, enquanto $A_{global}(N)$ é determinado dinamicamente durante o processo de análise.

Para cada elemento (ou componente) N dos grafos de sintaxe, estamos interessados principalmente em dois conjuntos de símbolos associados a N: o conjunto diretor e o conjunto de recuperação. O conjunto diretor (também chamado FIRST(N) por Aho e Ullman [AHO77]) con

têm aqueles símbolos que podem iniciar cadeias que são derivadas a partir do componente N dos grafos de sintaxe. O conjunto de recuperação consiste da união do conjunto diretor de N, com os conjuntos diretor de todos os elementos (ou componentes) do alcance de N.

Sejam D(N) o conjunto diretor de N, e R(N) o conjunto de recuperação de N. Logo temos

$$R(N) = D(N) + R(A(N)) \tag{4.1}$$

onde

$$R(A(N)) = R(A_{local}(N)) + R(A_{global}(N)),$$

e + indica união de conjuntos.

$R(A_{local}(N))$ é definido ser a união dos conjuntos diretor para todos os nós contidos no alcance local de N, isto é

$$R(A_{local}(N)) = \bigcup_{M \in A_{local}(N)} D(M)$$

observamos que

$$C = R(A_{global}(N))$$

é o que Hartmann chama do contexto do grafo de sintaxe corrente.

Consideramos agora as diversas alternativas para N

(a) N um elemento terminal

D(N) é o conjunto que contém somente o símbolo terminal correspondente

(b) N um elemento não terminal

D(N) é o conjunto de símbolos terminais que iniciam cadeias derivadas a partir do não terminal correspondente.

(c) N uma sequência

Seja N a sequência $N_1;N_2;\ldots;N_m$ de elementos ou componentes. Então $D(N) = D(N_1)$.

(d) N uma alternação

N corresponde a um conjunto $\{N_1,N_2,\ldots,N_m\}$ de sequências alternativas, sendo no máximo uma destas vazia. Então

$$D(N) = \bigcup_{1 \le i \le m} D(N_i;A(N)) \tag{4.2}$$

onde ; indica o operador de sequenciamento. Note que se N_i for vazia, então $D(N_i;A(N)) = D(A(N))$ e no outro caso $D(N_i;A(N)) = D(N_i)$.

(e) N uma iteração

suponhamos que a iteração contém uma sequência E no elo de iteração. Então

$$D(N) = D(E) + D(A(N)) \tag{4.3}$$

e neste caso, como o elo poderá ser percorrido várias vezes, temos

$$\begin{aligned} R(N) &= D(N) + R(E) + R(A(N)) \\ &= R(E) + R(A(N)) \end{aligned}$$

por causa da definição de D(N).

Agora temos condições de explicar como incorporar o esquema de Hartmann no analisador dirigido por tabelas. A cada nõ onde seria possível detetar um erro de sintaxe, associamos os conjuntos diretor e de recuperação. Portanto estes conjuntos serão associados aos nõs que correspondem aos elementos terminais, às alterações e às iterações. Para facilitar o tratamento dos nõs compostos, introduzimos para cada alternação um elemento de alternação, e a cada iteração um elemento de iteração, como mostrado na figura 2.

A cada elemento de alternação e de iteração associamos seu conjunto diretor e seu conjunto de recuperação. Adicionalmente, como indicado por Hartmann e Pemberton, devemos associar a cada elemento de iteração um terceiro conjunto de símbolos terminais, usado para determinar se saímos da iteração depois de um erro de sintaxe. Denotamos este conjunto por S(N), o conjunto de saída de N, definido a seguir

$$S(N) = R(A(N)) - D(E) \qquad (4.4)$$

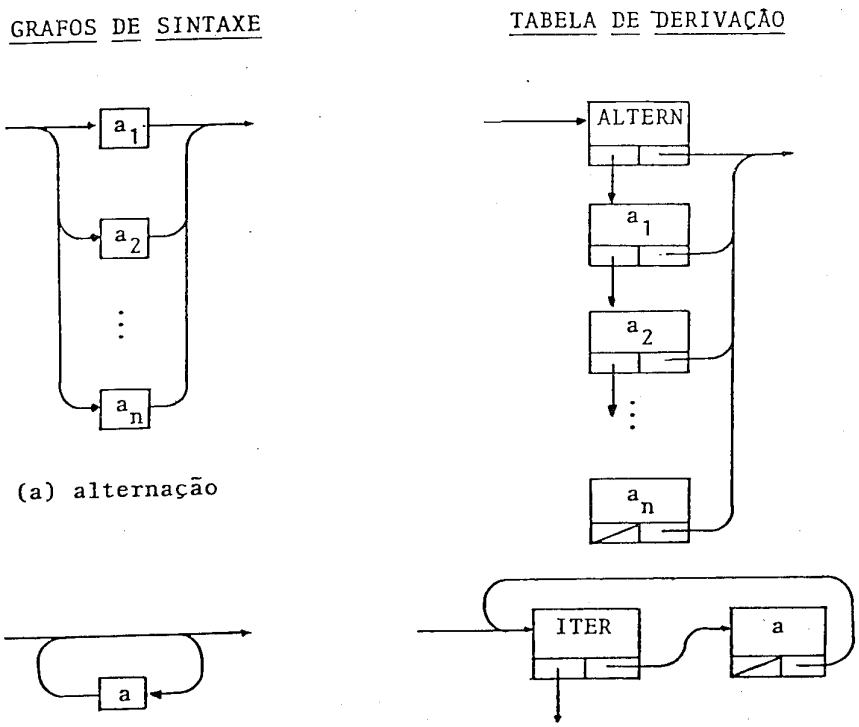


Figura 2: Estruturas da tabela de derivação para a recuperação de erros.

Finalmente, para permitir controlar o fluxo de análise nos componentes de alternação e iteração, ao primeiro elemento em cada alternativa de uma alternação, e ao primeiro elemento do elo de uma iteração, associamos seu conjunto diretor. Observamos que, em geral, isto só afeta os elementos não terminais, uma vez que a todos os outros elementos já associamos o conjunto diretor.

O algoritmo de análise sintática agora é razoavelmente simples. Numa linguagem do estilo Pascal, o elo principal é descrito a seguir

```
while (*1*)(pilha não estiver vazia) and (entrada não estiver vazia)
   do begin
    (*verificamos se tem erro de sintaxe*)
     while (*2*) (tipodeelemento(N)<>não terminal  and not (proxsimb in D(N))
        do begin
          (*lê entrada até encontrar ponto de recuperação*)
          while (*3*) not (proxsimb in R(N))
             do obtém (proxsimb);
          (*podemos emitir uma mensagem de erro aqui*)
          (*agora ressincronizamos a análise com a entrada*)
          while (*4*) not (proxsimb in D(N))
             do if tipo de elemento(N)=iteração
                  then if proxsimb in S(N) then N:=alt(N) (*sai do elo*)
                                           else N:=suc(N)
                  else if suc(N)=nil then desempilha(N)
                       (*volta ao não terminal anterior*)
                                     else N:=suc(N)
        end; (*while 2*)
        (*agora temos proxsimb ∈ D(N), e podemos processar o nó*)
      case tipo de elemento(N) of
           terminal: begin obtém (proxsimb); N:=suc(N) end;
       não terminal: begin empilha(N); N:=nsimb(N) end;
         alternação: repeat N:=alt(N) until proxsimb in D(N);
           iteração: if proxsimb in D(suc(N)) then N:=suc(N)
                                              else N:=alt(N)
        end (*case*)
   end (*while 1*)
```

5. CONSIDERAÇÕES SOBRE A IMPLEMENTAÇÃO

A formulação do algoritmo da análise sintática dada na seção anterior esconde alguns detalhes da implementação. Discutimos estes agora.

Representação dos nós

As funções suc, alt, tipo de elemento e nsimb são todas funções estaticamente determinável de N, o nó corrente. Logo são todos candidatos ideais para representação por tabelas. As funções D, R e S têm uma parte estática e uma parte dinâmica (veja equações (4.1) a (4.4)). A parte estática pode ser determinada de uma vez por todas e inserida na tabela sintática, enquanto a parte dinâmica é determinada pelo algoritmo de análise, e é unida com a parte estática toda vez que isto se torna necessário. Portanto, incluimos em nossa tabela sintática os três campos, diretor, recuperação e saída, que correspondem às partes estáticas de D, R e S, respectivamente. Adicionalmente, precisamos associar aos elementos de iteração, e aos elementos de alternação que tiverem uma alternativa vazia, um campo booleano chamado últimoi e últimoa, respectivamente, que indicam se $A_{local}(N)$ contêm um caminho vazio. Uma possível representação do nó é apresentada em seguida.

```
type símbolos = set of simbterm;
    tipo de nó = (terminal,não terminal,alternação,iteração);
     ponteiro = ↑nó;
           nó = record
                 suc,alt:ponteiro;
                 diretor,recuperação:símbolos;
                 case tipo de elemento:tipo de nó of
                 não terminal: (nsimb:ponteiro);
                   iteração  : (saída:símbolos;
                                últimoi:boolean);
                  alternação : (últimoa:boolean)
                 end
```

Podemos melhorar o aproveitamento de espaço, observando que os conjuntos diretor dos nós terminais contêm um só elemento.

Os conjuntos diretor, de recuperação e de saída

Como observamos acima, estes conjuntos têm uma parte estática e (possivelmente) uma parte dinâmica (veja equações (4.1) a (4.4)). As partes dinâmicas podem ser escritas

$$D(A_{global}(N)) \qquad \text{e} \qquad R(A_{global}(N))$$

e podem ser obtidas a partir da relação

$$D(A_{global}(N)) = D(M)$$

onde M é o elemento (ou componente) sucessor do elemento não terminal no topo da pilha. Se não existe nenhum sucessor deste não terminal, então examinamos o elemento imediatamente abaixo do topo da pilha (o topo da pilha anterior), e assim em diante, até encontrar um sucessor.

Já mostramos que $R(A_{global}(N))$ (o contexto de Hartmann) é obtido por união de

$R(A_{local}(M'))$, onde M' é o elemento não terminal no topo da pilha, com o valor anterior do contexto global.

Notamos que em ambos os casos deveríamos empilhar também os valores antigos de $D(A_{global})$ e $R(A_{global})$ quando empilhamos um elemento não terminal, para permitir a restauração desses valores ao desempilhar o elemento não terminal.

Portanto, quando precisarmos o valor de R(A(N)), devemos unir $R(A_{local}(N))$ ao contexto global corrente. Do mesmo modo, quando for necessário o valor de D(A(N)), devemos determinar primeiro se $A_{local}(N)$ contém um caminho nulo. Neste caso, $D(A_{global}(N))$ deve ser unido a $D(A_{local}(N))$.

Ações semânticas

Para fazer parte de um compilador, o analisador sintático deve produzir saída, através das chamadas ações semânticas. Pode-se fazer isto introduzindo mais um tipo de elemento, do tipodeelemento semântica, tendo dois campos, suc e ação. Ação pode ser o número ordinal de uma rotina semântica. Como alternativa, podemos acrescentar um campo ação aos outros tipos de elemento. Ou podemos fazer ambas estas coisas. Uma idéia muito promissora é a geração automática dos elementos semânticos através de extensões à gramática para incluir ações de tradução, assim produzindo um esquema de tradução dirigido pela sintaxe ("sintax directed translation scheme") como descrito por Barrett e Couch [BAR79].

Conjuntos

Uma maneira natural de implementar conjuntos é como vetores de bits. Algumas linguagens de programação, tais como Pascal [JEN75] e diversas das suas derivadas, implementam conjuntos como construtores básicos, e estes podem ser usados para as estruturas de dados descritas neste artigo.

6. A CONSTRUÇÃO AUTOMÁTICA DAS TABELAS SINTÁTICAS ESTENDIDAS

Wirth [WIR76] sugeriu um algoritmo para gerar as tabelas sintáticas na representação da seção 2 usando como entrada uma gramática na forma estendida de Backus e Naur ("extended BNF"). É uma tarefa razoavelmente simples estender este algoritmo para gerar a informação adicional necessária para implementar o esquema descrito aqui. Os passos adicionais que são necessários para alcançar este objetivo incluem:

(a) geração dos elementos de alternação e iteração

(b) determinação dos conjuntos diretor dos elementos não terminais. Algoritmos para isto são bem conhecidos (veja, por exemplo, Aho e Ullman [AHO77])

(c) determinação dos conjuntos de recuperação locais. Isto é feito melhor por meio de uma travessia regressiva de cada grafo de sintaxe, pois, como já notamos, temos

$$R(A_{local}(N)) = \bigcup_{M \in A_{local}(N)} D(M)$$

e evidentemente o conjunto de recuperação de um grafo nulo é vazio.

(d) Determinação dos elementos de alternação com uma alternativa vazia e sem sucessor local

(e) determinação dos elementos de iteração sem sucessor local

Trabalhando em grupos de três ou quatro alunos, estas extensões foram feitas ao algoritmo de Wirth como trabalho prático num curso sobre compiladores no programa de pós-graduação na PUC/RJ no segundo semestre de 1982.

7. CONCLUSÕES

Mostramos como o esquema de recuperação de erros de Hartmann, que Pemberton chamou de definitiva para analisadores descendentes recursivos, pode ser estendido facilmente a analisadores dirigidos por tabelas. A facilidade desta extensão se deve à simplicidade dos conceitos envolvidos, tanto de grafos de sintaxe, como de pontos de recuperação. O analisador resultante foi usado num compilador desenvolvido na PUC/RJ para a linguagem Edison [AGU83,BH81,STA82]. Também indicamos como as estruturas de dados adicionais para recuperação de erros podem ser geradas automaticamente da gramática, indicando também o trabalho necessário para sua implementação. Trabalhos futuros incluirão o desenvolvimento de um gerador de analisadores sintáticos baseado nas idéias aqui apresentadas.
Devemos mencionar aqui o método de recuperação de erros sintáticos de V.W.Setzer [SET81], descrito no contexto de análise sintática de gramáticas ESLL(1). Estas gramáticas são mais restritas que as gramáticas consideradas aqui, e portanto requerem maior cuidados na formulação. Quanto ao método de recuperação, o método de Setzer não trata de modo distinto os componentes de alternação e iteração, podendo somente reconhecer e recuperar de erros sintáticos nos nós terminais. A recuperação envolve a aplicação sucessiva de quatro estratégias diferentes de correção: eliminação, inserção e substituição de um símbolo terminal, ou a busca de um delimitador. É nossa opinião que as quatro estratégias de Setzer estão simultaneamente incorporadas na nossa busca de um ponto de recuperação. Resta comparar experimentalmente a eficácia relativa dos dois métodos.

* * *

Os autores agradecem o apoio financeiro da Finep e do CNPq, e as facilidades computacionais gentilmente cedidas pela UFPe. Valdemar Setzer contribuiu ao aprimoramento da descrição dos algoritmos através de longas conversas tidas com os autores, que lhe agradecem.

8. BIBLIOGRAFIA

[AGU83] Aguiar,H.M.G., "Projeto de um compilador para a linguagem Edison e implementação dos módulos de análise léxica e sintática", tese de mestrado, PUC/RJ, 1983.

[AHO77] Aho,A.H. and Ullman,J.D., "Pinciples of Compiler Design", Addison Wesley, Reading, Massachussetts, 1977.

[BAR79] Barrett,W.A.and Couch,J.D., Compiler Construction: Theory and Practice", Science Research Associates, 1979.

[BH81] Brinch Hansen,P., "Edison - A Multiprocessor Language", Software-Practice and Experience, 11(4), 325-361, 1981.

[HAR77] Hartmann,A.C., A Concurrent Pascal Compiler for Minicomputers", Springer-Verlag, Berlin, 1977.

[JEN75] Jensen,K. and Wirth,N., "Pascal User Manual and Report", Springer-Verlag, New York, 1975.

[PEM80] Pemberton,S., "Comments on an Error-recovery Scheme by Hartmann", Software-Practice and Experience, 10(3), 231-240.

[SET79] Setzer,V.W., "Non-recursive Top-down Syntax Analysis", Software-Practice and Experience, 9(3), 237-245, 1979.

[SET81] Setzer,V.W. e Homem de Melo,I.S., "A Construção de um Compilador", 2ª Escola de Computação, Campinas, 1981.

[STA82] Stanton,M.A., et al., "Projeto de um compilador portátil para a linguagem Edison", Anais do XV Congresso Nacional de Informática, Rio de Janeiro, outubro de 1982.

[WIR76] Wirth,N., "Algorithms + Data Structures = Programs", Prentice-Hall, Englewood Cliffs, New Jersey, 1976.