UM GERADOR AUTOMÁTICO DE RECONHECEDORES SINTÁTICOS PARA O SPD

JOÃO JOSÉ NETO
MARIO EDUARDO S. MAGALHÃES

ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO
CAIXA POSTAL 11.455
01000 - SÃO PAULO - SP.

RESUMO

Este artigo apresenta os métodos de análise sintática e de geração de reconhecedores sintáticos, resultantes de um primeiro esforço em busca da inclusão de um sistema de apoio à construção de compiladores ("compiler-writting system") no SPD - um sistema automático de apoio ao projeto e desenvolvimento de sistemas digitais realizado no LSD - EPUSP. É feito um resumo dos resultados teóricos mais importantes em que se baseiam tais métodos, apresentando-se a seguir as diretrizes utilizadas no reconhecedor construído pelo sistema e no programa que o gera. Um esboço funcional dos algoritmos que implementam esta metodologia é apresentado, finalizando-se o trabalho com um exemplo que exercita o sistema.

ABSTRACT

This paper presents the results of a first effort towards the inclusion of compiler-writting systems facilities into the SPD - an automatic tool for digital systems research and development – , by describing methods of syntactic analysis and automatic generation of syntactical recognizers. The most important theoretical results are listed and the main guidelines of the recognizer's and of its generator's design are presented along with their algorithm and an application example.

1. INTRODUÇÃO

O SPD - sistema automático de apoio ao projeto e desenvolvimento de sistemas digitais - foi desenvolvido durante a década de setenta no Laboratório de Sistemas Digitais (LSD) da EPUSP, motivado pela necessidade da então emergente atividade de projeto de computadores no Brasil, visando dotar o LSD de recursos computacionais para a automação de alguns procedimentos trabalhosos e freqüentemente utilizados pela equipe de projeto. Com este espírito idealizou-se o SPD como um gerenciador de recursos que coloca à disposição do usuário ferramentas que simplificam a obtenção de "software" de apoio tais como: simuladores, montadores e outros programas auxiliares estruturalmente semelhantes. Em uma primeira aproximação o SPD pode ser visto como um sistema que aceita, como entrada, descrições em uma linguagem de alto nível, dita linguagem de descrição (LD), dos módulos de *software* desejados, gerando como saídas uma versão executável dos módulos descritos, cuja operação pode ser comandada através de uma linguagem de controle, dita linguagem de execução (LE), também oferecida pelo sistema.

Atualmente o SPD dispõe de recursos para a descrição, obtenção e execução de módulos tais como simuladores e montadores. Pretende-se com este trabalho descrever os primeiros resultados obtidos em estudos realizados no sentido de incluir no SPD recursos que o tornem também capaz de oferecer apoio à construção de processadores de linguagens de alto nível, o que se justifica pela conveniência de utilização destes graças à conseqüente simplificação obtida no desenvolvimento e manutenção de programas.

Apresenta-se neste trabalho a descrição da parte de tal sistema de apoio responsável pela geração automática de reconhecedores sintáticos que deverão constituir o núcleo dos compiladores a serem obtidos com o auxílio deste sistema.

O reconhecedor utilizado neste trabalho é baseado na teoria exposta em [4]. Qualitativamente, o modelo pode ser visualizado como um conjunto de máquinas de estados finitos, associadas através de uma pilha. Cada uma destas máquinas tem como tarefa o reconhecimento de uma dada classe de construções de linguagem. Em cada instante, existe apenas uma máquina ativa, a qual decide, a partir do próximo caracter de entrada a analisar, entre efetuar transições internas à máquina em questão, transitar para outra máquina salvando o estado de retorno na pilha, ou ainda retornar para uma eventual

máquina a partir da qual a máquina corrente foi ativada (com base em um estado de retorno previamente armazenado na pilha). Caso nenhuma das três situações (normais) acima for reconhecida, constata-se uma condição de erro.

A parte do sistema responsável pela geração de reconhecedores é, em uma primeira aproximação, um conjunto de rotinas que efetua, sobre uma descrição fornecida da linguagem desejada, testes de consistência, e transformações gramaticais, gerando diagnósticos, listagens e índices, relativos à gramática fornecida, além de estruturas de dados cuja interpretação implementa o reconhecedor desejado.

Neste artigo, apresenta-se inicialmente um resumo dos resultados teóricos em que se baseia o sistema de apoio; em seguida, é fornecida uma visão global do método de reconhecimento, da estrutura do reconhecedor e dos princípios de construção do reconhecedor a partir da gramática. Discute-se também os algoritmos utilizados, finalizando-se com um exemplo de uma pequena linguagem, do respectivo reconhecedor, tal como gerado pelo algoritmo, e um acompanhamento passo a passo da operação deste reconhecedor no processamento de um pequeno texto exemplo.

2. RESUMO DAS BASES TEÓRICAS

Uma linguagem pode ser representada, de modo geral, de duas maneiras: através de dispositivos geradores ou gramáticas (que constam de uma coleção de regras (produções) que estabelecem maneiras de formar sentenças corretas a partir dos elementos do vocabulário da linguagem) ou através de dispositivos reconhecedores (que são também coleções de produções que estabelecem maneiras de verificar se a sentença fornecida faz parte ou não da linguagem em questão). Chomsky estabelece uma hierarquia entre as linguagens [1,3,5] segundo suas estruturas. Considerar-se-ão nesta publicação as linguagens pertencentes aos tipos 2 e 3 da hierarquia de Chomsky, ditas livres de contexto e regulares, respectivamente, e que são de grande interesse em computação. Dependências de contexto não serão tratadas neste trabalho, uma vez que podem ser consideradas fora dos procedimentos de reconhecimento de que trata este artigo.

Gramáticas. Uma gramática livre de contexto é uma quádrupla $G = (V, \Sigma, P, S)$ onde V, Σ e S denotam o vocabulário, os terminais e a raiz da gramática, e P é o conjunto de produções, do tipo $\alpha \rightarrow \beta$, com $\alpha \in N = V-\Sigma$, $\beta \in V^*$. (1)

Define-se gramáticas regulares ou lineares como sendo aquelas

em que as produções (1) são de duas formas apenas:

i) $\alpha \rightarrow a$

ii) $\alpha \rightarrow a\beta$ (neste caso, as gramáticas são chamadas lineares à direita)

ou $\alpha \rightarrow \beta a$ (neste, à esquerda)

onde: $a \in \Sigma$, $\alpha \in N$, $\beta \in V^*$

Uma gramática livre de contexto é *self-embedded* se e somente se existir algum não terminal $\alpha \in N$ tal que

$\alpha \Rightarrow^* \beta \alpha \gamma$ com $\beta, \gamma \in V+$

Sabe-se [3] que gramáticas livres de contexto que não sejam *self-embedded* geram linguagens regulares.

Reconhecedores. Define-se um autômato de pilha como sendo uma sêtupla:

$$A = (Q, \Sigma, \Gamma, P, q_0, Z_0, F)$$

onde Q é um conjunto finito não vazio de estados

Σ é um conjunto finito não vazio de terminais

Γ é um conjunto finito não vazio de símbolos de pilha

$q_0 \in Q$ é o estado inicial de A

$Z_0 \in \Gamma$ é o símbolo inicial da pilha

$F \subseteq Q$ é o conjunto do estados finais de A

P é um conjunto finito não vazio de produções, que são expressões p da forma:

$$p: \quad \gamma Z q_i \beta \rightarrow \gamma \delta q_j \alpha \qquad (2)$$

onde $\gamma \in \Gamma^*$ representa o conteúdo não visível da pilha

$Z \in \Gamma$ é o conteúdo do topo da pilha antes da aplicação de p

$\delta \in \Gamma^*$ é o conteúdo da pilha que substitui Z após a aplicação de p

$q_i, q_j \in Q$ são estados de A

$\beta \in \Sigma^*$ é a cadeia de entrada antes da aplicação de p

$\alpha \in \Sigma^*$ é a cadeia de entrada após a aplicação de p

$\beta \in \{\sigma\alpha, \alpha\}$ onde $\sigma \in \Sigma$ é o caracter de entrada consumido pela aplicação de p

Diz-se que A reconhece $\alpha_0 \in \Sigma^*$ por esgotamento da pilha e da cadeia de entrada quando, partindo de uma situação inicial $t_0 = (Z_0, q_0, \alpha_0\phi)$ for possível chegar a uma situação final $t_f = (Z_0, q_F, \phi)$ com $q_F \in F$ passando por uma cadeia de situações intermediárias $t_i = (\gamma_i, q_i, \alpha_i\phi)$, $0 < i < f$, onde em cada t_i tem-se:

$\phi \notin \Sigma$ é um indicador de fim de cadeia

γ_i representa o conteúdo de pilha

q_i representa o estado de A

α_i representa a cadeia ainda não analisada nesta situação.

Denota-se este reconhecimento por $t_o \Rightarrow^* t_f$

A classe de linguagens reconhecidas pelos autômatos de pilha corresponde às linguagens livres de contexto ou Chomsky tipo 2.

Os autômatos finitos, por sua vez, são definidos como quíntuplas:

$$A = (Q, \Sigma, P, q_o, F)$$

onde P contém produções do tipo

$$q_i \beta \rightarrow q_j \alpha \qquad (3)$$

Diz-se que A reconhece a cadeia $\alpha_o \in \Sigma$ se, partindo de situação inicial $t_o = (q_o, \alpha_o \phi)$ for possível chegar à situação final $t_f = (q_F, \phi)$ com $q_F \in F$, passando por situações intermediárias $t_i = (q_i, \alpha_i \phi)$, $0 < i < f$.

Todos os símbolos e as denotações mantêm seus significados habituais.

A classe de linguagens reconhecidas pelos autômatos finitos corresponde às linguagens regulares ou Chomsky tipo 3.

O modelo a ser utilizado pode ser definido através da óctupla:

$$M = (Q, A, \Sigma, \Gamma, P, z_o, q_o, F) \qquad (4)$$

onde A é o conjunto finito não vazio de submáquinas de M. Uma submáquina $a_i \in A$ é uma quíntupla :

$$a_i = (Q_i, \Sigma_i, P_i, E_i, S_i) \qquad (5)$$

onde $Q_i = \{q_{ij} | q_{ij} \in Q\} \subseteq Q$ é o conjunto finito não vazio de estados da submáquina a_i

$\Sigma_i \subseteq \Sigma$ é o conjunto finito não vazio de terminais correspondente ao alfabeto de entrada de a_i

$P_i \subseteq P$ é o conjunto finito não vazio de produções p_{ij} de a_i que são expressões das formas

$$p_{ij}: \gamma q_{ik} \beta \rightarrow \gamma q_{ij} \alpha \qquad (6a)$$

$$p_{ij}: \gamma q_{ik} \beta \rightarrow \gamma (i,\ell) q_{mn} \alpha \qquad (6b)$$

$$p_{ij}: \gamma (m,n)\, q_{ik} \beta \rightarrow \gamma q_{mn} \alpha \qquad (6c)$$

onde $\gamma \in \Gamma^*$ representa o conteúdo não visível da pilha

$(i,\ell) \in \Gamma$ representa a alteração da pilha

$(m,n) \in \Gamma$ é o antigo conteúdo do topo da pilha $\forall (m,n) \in \Gamma$

$q_{xy} \in Q_x$ são estados da submáquina $a_x \in A$

$\beta \in \Sigma^*$ representa a cadeia de entrada antes da aplicação de p_{ij}.

$\alpha \in \Sigma^*$ representa a cadeia de entrada após a aplicação de p_{ij}

$\beta \in \{\sigma\alpha, \alpha\}$, onde

$\sigma \in \Sigma_i$ é o caracter de entrada consumido pela aplicação de p_{ij}

$E_i \subseteq Q_i$ é o conjunto não vazio de estados de entrada de a_i

$S_i \subseteq Q_i$ é o conjunto não vazio de estados de saída de a_i

Q é o conjunto finito não vazio de estado de M

$$Q = \{q_{ij} \in Q_i | a_i = (Q_i, \Sigma_i, P_i, E_i, S_i) \in A\} \quad (7)$$

$q_0 \in Q$ é o estado inicial de M com

$q_0 = q_{\infty} \in E_0$ onde $a_0 = (Q_0, \Sigma_0, P_0, E_0, S_0) \in A$ é a submáquina inicial de M

Σ é o conjunto finito não vazio de terminais correspondente ao alfabeto de entrada de M

$$\Sigma = \{\sigma \in \Sigma_i | a_i = (Q_i, \Sigma_i, P_i, E_i, S_i) \in A\} \quad (8)$$

Γ é o conjunto finito não vazio de símbolos da pilha de M, ou seja, é o alfabeto de pilha de M

$$\Gamma = \{Z_0\} \cup \{\gamma_{ij} = (i,j) | q_{ij} \in Q_i \text{ para algum } a_i = (Q_i, \Sigma_i, P_i, E_i, S_i) \in A\} \quad (9)$$

$Z_0 \in \Gamma$ é o símbolo inicial da pilha

P é o conjunto finito não vazio de produções de M dado por:

$$P = \{p_{ij} \in P_i | a_i = (Q_i, \Sigma_i, P_i, E_i, S_i) \in A\} \quad (10)$$

F é o conjunto finito não vazio de estados finais de M

$$F = \{q_{oj} \in S_0 | a_0 = (Q_0, \Sigma_0, P_0, E_0, S_0) \in A\} \quad (11)$$

Diz-se que M reconhece $\alpha_0 \in \Sigma^*$ quando, partindo de uma situação inicial $t_0 = (Z_0, q_0, \alpha_0\phi)$ (12) for possível chegar a uma situação final $t_f = (Z_0, q_F, \phi)$ (13) com $q_F \in F$ passando por uma cadeia de situações intermediárias $t_i = (\gamma_i, q_i, \alpha_i\phi)$ (14) com $0 < i < f$ onde em cada t_i tem-se

- $\phi \notin \Sigma$ é um indicador de fim de cadeia
- γ_i representa o conteúdo da pilha
- q_i representa o estado de M
- α_i representa a cadeia de entrada ainda a ser analisada

Denota-se este reconhecimento por $t_0 \Rightarrow^* t_f$

Este reconhecedor é detalhado mais profundamente na referência [4], em que é demonstrada sua equipotência ao autômato de pilha, de onde se pode concluir que a classe de linguagem por ele reconhecida também corresponde às linguagens livres de contexto ou Chomsky tipo 2, o que será explorado adiante.

Tal modelo define uma máquina M (exp 4) constituída basicamente de um conjunto A de submáquinas a_i (exp 5), as quais operando em conjunto, efetuam a análise de uma cadeia de entrada α_o formada por elementos de Σ (exp 8), utilizando-se de uma pilha auxiliar onde são armazenados elementos de Γ. Estes (exp 9) correspondem univocamente, com exceção do símbolo inicial da pilha Z_o, aos estados $q_{ij} \in Q$ (exp 7) entre os quais a máquina M transita. Observe-se que Z_o é um elemento particular de Γ que não guarda correspondência com nenhum estado da máquina, tendo como única função indicar, quando no topo da pilha, a ausência de outros elementos armazenados na mesma.

Um estado particular q_o é o estado inicial da máquina M. Assim a operação desta sempre se inicia com a pilha vazia (contendo somente Z_o), na máquina inicial a_o à qual pertence o estado $q_o = q_{oo}$.

Pode-se visualisar a operação da máquina como a evolução entre diversas situações (exp 14), a partir de uma situação inicial (exp 12) em direção a uma situação final (exp 13) conveniente.

Tal evolução decorre da aplicação do conjunto de produções P (exp 10). Este é constituído por grupos $P_i \subseteq P$ de produções que regem o funcionamento da respectiva submáquina a_i quando esta estiver ativa. (Obviamente, neste modelo não há simultaneidade de operação entre duas máquinas quaisquer).

Durante a operação de M através de uma máquina a_i, três classes de movimento podem ocorrer, dependendo do tipo de produção aplicada. Tal aplicação é decidida em função apenas do estado atual e eventualmente do caracter de entrada devendo-se notar que o conteúdo da pilha não influi na decisão.

Se, partindo da situação inicial (exp 12) após transitar pelos estados convenientes q_{ij} das respectivas submáquinas a_i, a máquina M atingir algum estado $q_{oj} \in F$ tendo então consumido todos os caracteres $\sigma \in \Sigma$ que compunham a cadeia de entrada α_o, e encontrando vazia a pilha (Z_o no topo), (exp 13) a cadeia de entrada α_o é dita reconhecida pela máquina M, isto é, pertence à linguagem cujo dispositivo reconhecedor a máquina M implementa. Sendo impossível atingir estas condições diz-se que α_o não pode ser reconhecido por M, ou seja α_o não pertence à linguagem que M reconhece.

3. O ALGORITMO

Pretende-se apresentar um algoritmo que permita a obtenção de um reconhecedor para uma linguagem livre de contexto a partir de

uma descrição formal de sua sintaxe, em BNF.

Observe-se que o reconhecedor sintático obedece ao modelo teórico apresentado anteriormente, devendo portanto ser constituído de um conjunto de regras, correspondente ao conjunto P de produções, juntamente com um mecanismo de armazenamento que simula a pilha de controle utilizada. Para o funcionamento dinâmico do modelo são necessárias uma rotina encarregada da extração de átomos da cadeia de entrada (analisador léxico) e rotinas que interpretam as regras definidas pelas produções.

Note-se que este modelo corresponde ao núcleo de um compilador voltado para sintaxe e que certas partes deste esquema permanecem fixas independentemente da linguagem a ser reconhecida (analisador léxico e rotinas de interpretação das produções). Para cada reconhecedor sintático desejado, é necessário construir um conjunto diferente de produções, admitindo-se que tenham sido feitas as devidas adaptações nas rotinas de análise léxica. Assim, a partir do BNF, serão construídas as produções para cada linguagem cujo reconhecedor se deseja. O que se pretende descrever adiante é o conjunto de regras que mapeiam as produções BNF nas produções do autômato proposto e que serão interpretadas pelo bloco de controle do reconhecedor gerado. As adaptações do analisador léxico são triviais e não serão tratadas neste texto.

Procedimento para a contrução da máquina M a partir da gramática G = (V, Σ, P, S) definida em BNF.

1) Testa-se se a gramática é reduzida
   - se não for, são fornecidas ao usuário indicações sobre as causas da não redução e para-se o processo.
2) Agrupam-se em produções únicas todas as opções de definição de cada não terminal, obtendo-se para cada $n_i \in N$ uma única produção da forma:

   p_i: $n_i ::= d_{i1}|d_{i2}|\ldots|d_{ip}$ com $d_{ij} \in V^*, 1 \leq j \leq p$

3) Abrevia-se as produções de acordo com as seguintes regras:
   i) $n_i ::= \alpha_o\beta_1|\alpha_o\beta_2|\ldots|\alpha_o\beta_m$ em $n_i ::= \alpha_o(\beta_1|\beta_2|\ldots|\beta_m)$
   ii) $n_i ::= \alpha_o|\alpha_o\beta_1|\alpha_o\beta_2|\ldots|\alpha_o\beta_m$ em $n_i ::= \alpha_o[\beta_1|\beta_2|\ldots|\beta_m]$
   iii) $n_i ::= \alpha_1|\alpha_2|\ldots|\alpha_p|\beta_1n_i|\beta_2n_i|\ldots|\beta_mn_i|n_i\gamma_1|n_i\gamma_2|\ldots|n_i\gamma_q$
   em $n_i ::= \{\beta_1\beta_2|\ldots|\beta_m\}(\alpha_1|\alpha_2|\ldots|\alpha_p)\{\gamma_1|\gamma_2|\ldots|\gamma_q\}$
   iv) $n_i ::= \alpha_1|\alpha_2|\ldots|\alpha_p|\beta_1n_i|\beta_2n_i|\ldots|\beta_mn_i|n_i\gamma_1|n_i\gamma_2|\ldots|n_i\gamma_2|\ \lambda$

em $n_i ::= \{\beta_1|\beta_2|\ldots|\beta_m\}\ [\alpha_1|\alpha_2|\ldots|\alpha_p]\ \{\gamma_1|\gamma_2|\ldots|\gamma_q\}$

onde $n_i \in N = V - \Sigma$ é um não terminal

α_j, $0 \leq j \leq p$; γ_j, $1 \leq j \leq q$; β_j, $1 \leq j \leq m$ são elementos do conjunto Δ definido como:

$\Delta = \{\delta|\delta$ é de uma das seguintes formas λ; v_i; (δ_k); $[\delta_k]$; $\{\delta_k\}$; $\delta_i\delta_j$, denominadas fatores, ou na forma de opções: $\delta_j|\delta_k$, onde λ é a cadeia vazia de caracteres; $v_i \in V$; δ_i, δ_j são fatores; $\delta_k \in \Delta\}$

Observe-se que até este ponto a gramática $G = (V, \Sigma, P, S)$ não foi modificada, tendo sido efetuadas meras mudanças de notação. Construir-se-á a seguir uma gramática $G' = (V', \Sigma, P', S)$ equivalente, através de modificações visando obter-se uma forma que facilite a geração automática da máquina M proposta. Sem perda de generalidade, pode-se supor que a raiz da gramática G não seja *self-embedded*. (Caso isto não ocorra, acrescenta-se às produções orignais uma produção adicional $S ::= S'$ onde S' era a antiga raiz da gramática).

4) Constrói-se G' equivalente a G, aplicando-se as regras abaixo.

i) Constrói-se o conjunto N'_s contendo todos os elementos de N que sejam *self-embedded*.

$$N'_s = \{n_i \in N | n_i \Rightarrow^* \alpha\, n_i \beta, \quad \text{com} \quad \alpha, \beta \in V^*\}$$

ii) Obtém-se do usuário o conjunto N'_u de não-terminais, elementos do conjunto N que se deseja preservar na gramática G' final. Observe-se que todos os demais não-terminais, isto é, nem *self-embedded* nem diferenciados no conjunto N'_u, poderão ser eliminados na gramática final.

Constrói-se assim os conjuntos:

$$N' = N'_a \cup N'_u \cup \{S\} \quad \text{e} \quad V' = N' \cup \Sigma$$

iii) Para cada $p_i \in P$, da forma:

$$p_i: \quad n_i ::= \delta_i, \quad \text{com } n_i \in N', \delta_i \in \Delta,$$

Para cada ocorrência, em δ_i de não terminais $n_j \in N$ tais que $n_j \neq n_i$ iniciando opção (que não seja única), ou $n_j \in N - N'$ nos demais casos, substituir n_j por sua definição entre parênteses (δ_i), desde que, se $n_j \in N'$, a presente substituição de n_j não seja decorrente de alguma substituição prévia do mesmo não terminal. Repete-se o procedimento, enquanto possível, nas expressões resultantes, obtendo-se finalmente p''_i.

A coleção de produções resultante constitui o conjunto P".

$$P'' = \{p''_i | n_i \in N'\}$$

iv) Aplica-se sobre o conjunto P" o seguinte procedimento, cuja função é detetar e eliminar sempre que possível a geração de produções não-determinísticas na máquina M que se pretende construir:

⌈- Cria-se o conjunto P' vazio
- Para cada produção p_i'' : $n_i ::= \delta_i$ em P"
  ⌈- Cria-se os conjuntos I, F, I' e F' vazios e R = {i}
  |- Deteção-de-não-determinismos (δ_i, I, F) /* *detalhada adiante neste artigo* */
  - Se I = ∅
    então p_i'' é incluída no conjunto P'
    senão ⌈- substituir toda a ocorrência, em δ_i, de elementos "n_j" pertencentes a I, com $n_j ::= \delta_j$, por "(δ_j)", obtendo-se a expressão δ_i'
      - Deteção-de-não-determinismos (δ_i', I', F')
      - Se I ≠ I' ou F ≠ F'
        então ⌈- retirar a produção p_i'' do conjunto P"
              - incluir a produção $n_i ::= \delta_i'$ no conjunto P" ⌋
        senão ⌈/* *não determinismo inevitável* */
              - repetir o seguinte procedimento quantas vezes se desejar:
                ⌈- substituir todas as ocorrências, em δ_i', de elementos "n_k" pertencentes a I, com $n_k ::= \delta_k$, por "(δ_k)" obtendo-se um novo δ_i'
                - incluir a produção $n_i ::= \delta_i'$ no conjunto P' ⌋ ⌋ ⌋ ⌋ ⌋

Apesar de, em alguns casos, ser impossível construir uma gramática cujas produções acarretam a geração exclusiva de produções determinísticas na máquina M (para deteção de construções repetitivas) é possível, à custa do aumento do número de estados de M, impor o ponto a partir do qual a máquina M passa a ser controlada por produções não determinísticas, o que é obtido fixando-se arbitrariamente o número de substituições de não-terminais "n_k" detetados no algoritmo acima.

O procedimento abaixo, Deteção-de-não-determinismos (δ,I,F) utilizado, analisa a expressão δ, construindo o conjunto I de não-terminais que acarretam geração de produções não determinísticas a partir da expressão δ, e o conjunto F que reune, para cada opção

de δ, as sub-expressões mais à esquerda, tais que:

a) a opção não contenha não-terminais. Neste caso, a sub-expressão corresponde à própria opção.

b) a opção contenha não-terminais. Neste caso, aplica-se estas duas regras à expressão que define o não-terminal mais à esquerda, obtendo-se F'. Inclui-se em F todas as cadeias correspondentes à sub-expressão que precede o não-terminal em questão, tendo concatenado à direita cada elemento do conjunto F'.

```
Procedimento Deteção-de-não-determinismos (δ, I, F)
⌈- Para cada opção da cadeia δ
    ⌈- Criar conjuntos I", F" vazios
     - Para cada fator da opção
        ⌈- Criar conjuntos I', F' vazios
         - Conforme o tipo do fator, escolher
           a) cadeia de terminais σ:
              - Se F" = ∅
                  então incluir a cadeia em F"
                  senão ⌈concatenar σ à direita de cada elemento de F" obtendo o novo conjunto F" ⌋
           b) não terminal  n_j: /* definido como n_j ::= δ_j */
         - Se j ∉ R
             então ⌈- incluir j no conjunto R
                    - Deteção-de-não-determinismos (δ_j, I', F')
                    - Se F" ≠ ∅
                        então ⌈- concatenar cada elemento de F' à direita de cada elemento de F" obtendo o novo conjunto F' ⌋
                    - Se F' ∩ F" ≠ ∅ então incluir n_j em I"
                    - Incluir em F" todos os elementos de F' ⌋
         - Busca-se o final da opção ⌋
           c) fator do tipo "[δ]" ou do tipo "{δ}"
              ⌈- Deteção-de-não-determinismos (δ, I', F')
               - Incluir a cadeia vazia λ em F'
               - Se F" ≠ ∅
                   então ⌈- concatenar cada elemento de F' à direita de cada elemento de F" obtendo o novo conjunto F'
                          - Fazer I" = I" ∪ I'  se  F' ∩ F" ≠ ∅ ⌋
               - Incluir em F" todos os elementos de F' ⌋
```

d) fator do tipo "(δ)"

⌈- Executa-se a operação descrita em c), a menos da inclusão da cadeia vazia λ

- Busca-se fim da opção ⌋ ⌋

- Incluir em I todos os elementos de I"

- Incluir em F todos os elementos de F" ⌋ ⌋

Pela aplicação dos procedimentos acima descritos, obtêm-se o conjunto P', o que completa a construção da gramática G'.

5) Construção da máquina M = (Q, A, Σ, Γ, $\overline{P}$, q_o, Z_o, F) a partir da gramática G' = (V', Σ, P', S)

i) A cada produção p_i do conjunto P' criar produções conforme o seguinte algoritmo:

- Para cada produção p_i: $n_i ::= \delta_i$ pertencente a P'

⌈- Faz-se j = 0, k = 0, j' = 0, k' = 0, esvaziar a pilha π

- Se n_i = S então faz-se $\mu = Z_o$ senão faz-se $\mu = \gamma$

- Para cada símbolo encontrado em δ_i /* *varredura da esquerda para a direita* */

⌈- Conforme o símbolo escolhe-se:

a) início da expressão δ_i : - Empilha-se (k,k')

b) terminal $\sigma \in \Sigma$: ⌈- Gera-se produção:

$\mu\, q_{ik}\, \sigma\, \alpha \rightarrow \mu\, q_{ij+1}{}^{\alpha}$

- incrementa-se j' ⌋

- faz-se k := j = j' ⌋

c) não terminal: $n_t \in N$: ⌈- Gerar produção:

$\mu q_{ik}{}^{\alpha} \rightarrow \mu(i,j+1) q_{t,0}{}^{\alpha}$

- incrementa-se j'

- faz-se k = j

d) meta símbolo "|":

⌈- Copia-se o topo da pilha em k,k' sem alterá-lo

- Se k = k' = 0

então gera produção: $\mu(m,n)q_{ij}{}^{\alpha} \rightarrow \mu q_{m,n}{}^{\alpha}$

senão gera produção: $\mu q_{ij}{}^{\alpha} \rightarrow \mu q_{ik'}{}^{\alpha}$ ⌋

e) fim da expressão δ_i

⌈- Gera produção: $\mu(m,n)q_{ik}{}^{\alpha} \rightarrow \mu q_{mn}{}^{\alpha}$ se $\mu \neq Z_o$

f) meta símbolo "("; "["; "{"

- faz-se j=j'=j'+1; se "{", então empilhar (j,k) senão (k,j)

g) meta símbolo ")":

⌈- Copia-se o topo da pilha em k,k' desempilhando-o

- Gera produção: $\mu q_{ij}\alpha \rightarrow \mu q_{ik'}\alpha$

- faz-se $k = k'$ ⌋

h) meta símbolo "]":

⌈- Copia-se o topo da pilha em k,k' desempilhando-o

- Gera produções: $\mu q_{ij}\alpha \rightarrow \mu q_{ik'}\alpha$

$\mu q_{ik}\alpha \rightarrow \mu q_{ik'}\alpha$

- faz-se $k = k'$⌋

i) meta símbolo "}"

⌈- Copia-se topo da pilha em k',k desempilhando-o

- Gera produções: $\mu q_{ij}\alpha \rightarrow \mu q_{ik}\alpha$

$\mu q_{ik}\alpha \rightarrow \mu q_{ik'}\alpha$ ⌋⌋

A partir do conjunto $\overline{P}$ das produções construídas, pode-se obter todos os componentes da máquina $M = (Q, A, \Sigma, \Gamma, \overline{P}, Z_o, q_o, F)$ por inspeção das produções:

$Q = \{q_{ij} | q_{ij}$ aparece em alguma produção de $\overline{P}\}$

$A = \{a_i = (Q_i, \Sigma_i, P_i, E_i, S_i) | n_i \in N'\}$

onde Q_i é o subconjunto de Q cujos estados são da forma q_{im}

Σ_i é o subconjunto de Σ cujos elementos aparecem nas produções de P_i

P_i é o subconjunto de $\overline{P}$, cujas produções assumem uma das formas seguintes:

i) $\gamma q_{ik}\beta \rightarrow \gamma q_{ij}\alpha$

ii) $\gamma q_{ik}\beta \rightarrow \gamma(i,\ell)q_{mn}\alpha$

iii) $\gamma(m,n)q_{it}\beta \rightarrow \gamma q_{mn}\alpha$

E_i é formado do único estado de entrada $q_{io} \in Q_i$

S_i é o conjunto dos estados q_{it} encontrados nas produções do tipo iii) de P_i

Γ é o conjunto dos pares (i,ℓ) ou (m,n) encontrados nas produções dos tipos ii) e iii) de $\overline{P}$, além do símbolo Z_o inicial da pilha.

q_0 é o estado inicial (único) da máquina a_0, correspondente a $n_0 = S \in N'$

F é idêntico ao conjunto S_0 de estados de saída de a_0.

4. EXEMPLO

Considere-se a gramática G, variante do exemplo da referência [4], descrita abaixo em notação BNF (não-terminais correspondem a letras maiúsculas):

E ::= T E ::= λ T ::= T+F T ::= F

F ::= P * P F ::= P P ::= n|<E>

Transformando este conjunto de produções segundo os passos 1, 2 e 3 do algoritmo obtêm-se:

E ::= [T] T ::= F{+F} F ::= P{* P}

P ::= n|<E> S ::= E

Obtem-se o conjunto $N'_s = \{E,S\}$. Supondo $N_u = \emptyset$, ter-se-á o conjunto P" para a gramática G', formado pelas produções: (passo 4)

S ::= E E ::= [((n|<E>){*(n|<E>)}){+((n|<E>){*(n|<E>)})}]

Na tentativa (frustrada) de se detetar não determinismos, foram construídos os conjuntos: I_s e I_E vazios;

$F_s = \{n, \lambda\}$; $F_E = \{n, \lambda\}$

Logo, o conjunto P' desejado será o próprio conjunto P". Obtém-se a seguir a especificação da máquina M, através do seu conjunto de produções $\bar{P}$, pela aplicação do passo 5.

Para simplificar o conjunto de produções a ser construído pode-se eliminar parêntese supérfulos das expressões resultantes. São considerados supérfulos parênteses que envolvem fatores, bem como os que envolvem expressões desde que o conjunto seja envolvido por parênteses, colchetes ou chaves. Assim temos as produções BNF:

S ::= E E ::= [(n|<E>){*(n|<E>)}{+(n|<E>){*(n|<E>)}}]

As quais geram o conjunto $\bar{P}$ formado pelas produções

$P_s = \{p_0 : Z_0\, q_{So}\alpha \rightarrow Z_0\, (S,1) q_{Eo}\alpha\}$ onde a máquina é a_s, com $\Sigma_s = \Sigma$ pois a é qualquer elemento do vocabulário, $q_0 \in E_s = \{q_{So}\}$; $F = S_s = \{q_{S1}\}$

Para a máquina a_E tem-se:

$\Sigma_E = \Sigma$

$E_E = \{q_{Eo}\}$; $S_E = \{q_{E1}\}$ e conjunto P_E dado por:

a) produções do tipo p_k: $\gamma(m,n) q_{Ei}\alpha \rightarrow \gamma q_{mn}\alpha$, para i = 1 (k = 1)

b) produções do tipo p_k: $\gamma q_{Ei}\alpha \rightarrow \gamma(E,\ell) q_{Fo}\alpha$

para: $(k,i,\ell) \in \{(2,4,5); (3,11,12); (4,18,19); (5,25,26)\}$

c) produções do tipo p_k: $\gamma q_{Ei}\alpha \rightarrow \gamma q_{Ej}\alpha$ para:

k , i , j	k , i , j	k , i , j	k , i , j
6 , 0 , 1	10 , 7 , 14	14 , 14 , 1	18 , 21 , 7
7 , 2 , 7	11 , 9 , 2	15 , 16 , 21	19 , 23 , 16
8 , 3 , 2	12 , 10 , 9	16 , 17 , 16	20 , 24 , 23
9 , 6 , 2	13 , 11 , 9	17 , 20 , 16	21 , 27 , 23

d) produções do tipo p_k: $\gamma q_{Ei}\ \sigma\alpha \rightarrow \gamma q_{Ej}\alpha$

k , i , j , σ	k , i , j , σ	k , i , j , σ	k , i , j , σ
22 , 0 , 3 , n	26 , 8 , 10 , n	30 , 15 , 17 , n	34 , 22 , 24 , n
23 , 0 , 4 , <	27 , 8 , 11 , <	31 , 15 , 18 , <	35 , 22 , 25 , <
24 , 5 , 6 , >	28 , 12 , 13 , >	32 , 19 , 20 , >	36 , 26 , 27 , >
25 , 2 , 8 , *	29 , 7 , 15 , +	33 , 16 , 22 , *	--

As transições em vazio são justificadas nesta fase uma vez que o algoritmo gera produções que mantêm a estrutura inicial de todos os não terminais fornecidos pelo usuário no conjunto N_u, além dos *"self-embedded"* e da raiz da gramática, preservados necessariamente pelo algoritmo. Esta característica deverá ser explorada para a inclusão de rotinas de análise de sintática dinâmica e geração de código.

É feito a seguir o reconhecimento sintático da sentença:

$$n + \langle\langle n * n\rangle * n + \langle n\rangle\rangle$$

Por questões de concisão, transições que não consomem átomos da cadeia de entrada e/ou não alteram a pilha serão referenciadas apenas por seu índice. Caso algum átomo seja consumido, ou a pilha alterada, o fato será denotado pelo índice da produção, átomo consumido (quando houver) a alteração da pilha (quando houver) representadas pela operação realizada (+ : PUSH, - : POP). Os estados e os caracteres de pilha serão dados por pares ordenados.

PRODUÇÃO	OPERAÇÃO NA PILHA	ESTADO SEGUINTE	ATOMO	PRODUÇÃO	OPERAÇÃO NA PILHA	ESTADO SEGUINTE	ATOMO
0	+ (S,1)	(E,0)		7		(E,7)	
22		(E,3)	n	29		(E,15)	+
8		(E,2)		31		(E,18)	<
7		(E,7)		4	+(E,19)	(E,0)	
29		(E,15)	+	22		(E,3)	n
31		(E,18)	<	8		(E,2)	
4	+(E,19)	(E,0)		7		(E,7)	
23		(E,4)	<	10		(E,14)	
2	+ (E,5)	(E,0)		14		(E,1)	
22		(E,3)	n	1	-(E,19)	(E,19)	
8		(E,2)		32		(E,20)	>
25		(E,8)	*	17		(E,16)	
26		(E,10)	n	15		(E,21)	
12		(E,9)		18		(E,7)	
11		(E,2)		10		(E,14)	
7		(E,7)		14		(E,1)	
10		(E,14)		1	-(E,19)	(E,19)	
14		(E,1)		32		(E,20)	>
1	- (E,5)	(E,5)		17		(E,16)	
24		(E,6)	>	15		(E,21)	
9		(E,2)		18		(E,7)	
25		(E,8)	*	10		(E,14)	
26		(E,10)	n	14		(E,1)	
12		(E,9)		1	-(S,1)	(S,1)	FIM
11		(E,2)					

REFERÊNCIAS

[1] AHO, A.V., J.D. ULLMAN, "*The Theory of Parsing, Translation and Compiling*", 2 vols., Prentice-Hall, Englewood Cliffs, N.J., 1972.

[2] BARNES, B.H., "A programmer's view of Automata", *ACM Computing Surveys*, vol. 4, nº 2, december, 1972.

[3] HARRISON, M.A., "*Introduction to formal Language Theory*", Addison Wesley, 1978.

[4] JOSÉ NETO, J. e M.E.S. MAGALHÃES, "Reconhecedores Sintáticos - Uma alternativa didática para o uso em cursos de engenharia", Submetido ao *XIV Congresso Nacional de Informática*, São Paulo, 1981.

[5] SALOMAA, A., "*Formal Languages*", Academic Press, 1973

O MÉTODO DE GERAÇÃO DE CÓDIGO DO SISTEMA DE COMPILADORES LPS

IVAN LUÍS G. DE OLIVEIRA LIMA
COBRA S/A.
Av.GB-8, Eixo do Centro Metropolitano, 447 - RIO, RJ

ESTEVAM DE SIMONE
COPPE E INST.MATEMÁTICA/UFRJ
COPPE/SISTEMAS
CAIXA POSTAL 68511 - RIO, RJ

LUIZ CARLOS MONTEZ MONTE
COBRA S/A.
Av.GB-8, Eixo do Centro Metropolitano, 447, RIO, RJ

SUMÁRIO

Mediante a introdução de um segundo conjunto de símbolos terminais em uma gramática livre de contexto são definidas as gramáticas de tradução livres de contexto, estendidas a seguir através da adição de atributos. Desta forma é simplificada a implementação de um analisador sintático que comande o tradutor do programa fonte para uma sequência de chamadas a rotinas semânticas. Estas rotinas geram o código intermediário e efetuam o cálculo dos atributos semânticos. A geração de código se completa pela transformação das intruções intermediárias em instruções do processador e é proposto um esquema de transformação para operações binárias. O método foi empregado na construção do sistema de compiladores LPS para os computadores COBRA, e esta aplicação é parcialmente descrita à guisa de exemplo.

ABSTRACT

By adding a second set of terminal symbols to context-free grammars, we define context-free translation grammars, which are then extended to include attributes. This provides an easy way of constructing a parser-driven translator from source code to a sequence of semantic routine calls. These routines generate intermediate code and accomplish semantic attributes evaluation. Code generation will be completed by transforming intermediate instructions into processor instructions, and a scheme is given to binary operations transformation. The whole method was used in COBRA's LPS compiler system and its application is partially described within the examples.