PCS 3216 – Sistemas de Programação

Aula 20

Exemplo Completo - Construção de um Compilador

do cap. 5 do livro ***Introdução à Compilação*** 1ª Ed.

Parte 1 – Especificações e Análise Léxica

Introdução

Uma vez estudados diversos aspectos dos fundamentos pertinentes ao estudo dos compiladores, e levantadas as especificações relacionadas à sua implementação, a melhor forma de fixar todos os conceitos é aplicar na prática os métodos e técnicas estudados, com a finalidade de obter um compilador para uma linguagem desejada.

Para tanto, antes de mais nada é necessário que sejam efetuadas diversas decisões de projeto, especialmente em relação aos itens para os quais tenham sido levantadas múltiplas opções.

Uma vez escolhido o roteiro de implementação, é relativamente simples compor o programa final aplicando cuidadosamente os métodos de construção escolhidos para cada parte do compilador.

Integrar as partes do compilador também não é uma tarefa complexa, pois existem muitos algoritmos suficientemente conhecidos e divulgados na literatura que implementam as técnicas escolhidas.

Nesta aula, é desenvolvido um exemplo completo, procurando-se através dele apresentar um roteiro de elaboração para projetos semelhantes.

Descrição BNF da Linguagem-Exemplo

```
(1) <programa> ::= <seqüência de comandos> END
(2) <seqüência de comandos> ::= <comando> | <seqüência de comandos>; <comando>
(3) <comando>::= <rótulo>: <comando> | <atribuição> | <desvio> | <leitura>
                 | <impressão> | <decisão> | ε
(4) <atribuição> ::= LET <identificador> := <expressão>
(5) <expressão> ::= <expressão> + <termo>  | <expressão> - <termo> | <termo>
(6) <termo>::= <termo> * <fator>  | <termo> / <fator>  | <fator>
(7) <fator> ::= <identificador> | <número> | < <expressão> >
(8) <desvio> ::= GO TO <rótulo> | GO TO <identificador> OF <iista de rótulos>
(9) <lista de rótulos> ::= <rótulo> | <lista de rótulos> , <rótulo>
(10) <rótulo> ::= <identificador>
(11) <leitura> ::= READ <lista de identificadores>
(12) <lista de identificadores>::=<identificador>,<lista de identificadores>|ε
(13) <impressão> ::= PRINT <lista de expressões>
(14) <lista de expressões> ::= ε |  <expressão>, <lista de expressões>
(15) <decisão> ::= IF <comparação> THEN <comando> ELSE <comando>
(16) <comparação> ::= <expressão> <operador de comparação> <expressão>
(17) <operador de comparação>:: =   >  |  =  |  <
(18) <identificador>::=<letra>|<identificador><letra>|<identificador><dígito>
(19) <número> ::= <dígito> | <número> <dígito>
(20) <letra> ::= A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z
(21) <dígito> ::= 0|1|2|3|4|5|6|7|8|9
```

Sintaxe da Linguagem, em Notação de Wirth iterativa

```
(1) programa = seqüência-de-comandos "END".
(2) seqüência-de-comandos = comando { ";" comando }.
(3) comando ={ rótulo ":" }[ atribuição | desvio | leitura | impressão | decisão ].
(4) atribuição = "LET" identificador ":=" expressão.
(5) expressão = termo { ( "+" | "-" ) termo }.
(6) termo = fator { ( "*" | "/" ) fator }.
(7) fator = identificador | número | "(" expressão ")".
(8) desvio = "GO" "TO" ( rótulo | identificador "OF" lista-de-rótulos ).
(9) lista-de-rótulos = rótulo { "," rótulo }.
(10) rótulo = identificador.
(11) leitura = "READ" lista-de-identificadores.
(12) lista-de-identificadores = [ identificador { "," identificador } ].
(13) impressão = "PRINT" lista-de-expressões.
(14) lista-de-expressões = [ expressão { "," lista-de-expressões } ].
(15) decisão = "IF" comparação "THEN" comando "ELSE" comando.
(16) comparação = expressão operador-de-comparação expressão.
(17) operador-de-comparação = ">" | "=" | "<".
```

Gramática Léxica, em Notação de Wirth

```
Tokens não triviais:
(18) identificador = letra { letra | dígito }.
(19) número = dígito { dígito }.
(20) letra = A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z.
(21) dígito = 0|1|2|3|4|5|6|7|8|9.
```

```
Palavras reservadas:
"END", "LET", "GO", "TO", "OF", "READ", "PRINT", "IF", "THEN", "ELSE"
```

```
Operadores e outros sinais:
";", ":", "+", "-", "*", "/", "(", ")", ",", ">", "=", "<"
```

```
Símbolos compostos:
":="
```

Descrição informal dos comandos

Programa

Um programa qualquer, denotado na linguagem desejada, deve ser formado através da construção de uma seqüência de comandos, separados por ponto e vírgula, sendo o comando vazio uma alternativa válida.

Os rótulos que precedem um comando dão nome ao mesmo para que possam ser referenciados em comandos de desvio. Cada comando pode receber mais de um nome.

Ao longo de todo o programa, cada rótulo só pode aparecer uma única vez como nome de algum comando.

Não há restrições quanto ao número de instâncias dos rótulos-destino dos desvios permitidos na linguagem.

Além do comando vazio, cinco outros estão disponíveis: o comando de atribuição de valor a uma variável, o comando de desvio incondicional, o comando de leitura de valores, o comando de impressão de valores, e o comando de decisão.

Comando vazio

- O **comando vazio** não tem função alguma no programa, podendo ser livremente utilizado como mero recurso lingüístico de programação.

Comando de atribuição

- O comando de **atribuição de valor a uma variável** envolve o cálculo do valor associado a uma expressão, valor esse que é depositado em uma variável, explicitada através do seu identificador à esquerda da expressão.

Declaração de variáveis

- A linguagem não possui declarações explícitas para variáveis, sendo estas consideradas como declaradas na primeira ocasião em que forem referenciadas.
- Cabe portanto ao programador cuidar para que suas variáveis sejam adequadamente utilizadas ao longo do programa.

Expressões

- A avaliação de **expressões** segue as convenções usuais, sendo efetuada da esquerda para a direita, tendo produtos e divisões precedência sobre somas e subtrações, precedência essa que pode ser alterada através da utilização de parênteses.
- Apenas números e identificadores de variáveis são aceitos em expressões.
- Sobre os valores que representam, incidem as quatro operações aritméticas habituais.
- Essa linguagem utiliza aritmética inteira, com a precisão que esteja disponível na máquina na qual o programa-objeto deve ser executado.

Comandos de desvio

- Os **comandos de desvio** apresentam-se em dois formatos:
  - no primeiro (desvio direto), é especificado um desvio incondicional para um rótulo explicitamente indicado;
  - no segundo (desvio múltiplo), um identificador, que representa uma variável inteira, é utilizado como índice para a seleção de um dos rótulos de uma lista, e é para o rótulo assim selecionado que se executa o desvio. O primeiro rótulo desta lista associa-se ao valor zero da variável indexadora, o segundo, ao valor um, e assim por diante. Valores indevidos para a variável indexadora tornam inócuo o comando.

Comando de leitura

- O comando de **leitura** promove a entrada de dados de um meio externo (por exemplo, o teclado) e, com os valores lidos, preenche as variáveis de uma lista especificada no comando.
- No momento da execução deste comando, cada um dos dados deve ser fornecido como um elemento separado (por exemplo, um dado por linha).
- Assim, para um comando que especifica a leitura de n variáveis, deverá ser promovida a leitura de n linhas de dados, cada qual referente a uma das variáveis, e fornecidos na ordem especificada pela lista de variáveis especificada no comando.
- Caso essa lista seja vazia, o comando de leitura ignora (salta) a próxima linha de dados fornecida.

Comando de impressão

- O comando de **impressão** opera de forma análoga, porém age no sentido inverso: cada uma das expressões da lista que consta no comando é avaliada, e os valores obtidos são impressos, um por linha, na mesma ordem em que as expressões correspondentes figuram na lista especificada.
- Caso a lista encontrada no comando seja vazia, apenas uma linha em branco é impressa.

Comparações

- Comparações entre expressões são efetuadas através dos operadores > (maior que), = (igual a) e < (menor que).
- As expressões são avaliadas e comparadas da maneira habitual, de acordo com o operador escolhido, sendo todos os valores tratados como números inteiros relativos.
- Operações de comparação são usadas em comandos de decisão.

Comando de decisão

- O **comando de decisão** efetua inicialmente uma comparação entre duas expressões.
- Caso a comparação resulte verdadeira, será executado o comando que estiver presente, no comando de decisão, entre as palavras reservadas THEN e ELSE.
- Caso contrário, executa-se o comando que figurar após a palavra reservada ELSE.
- Note-se que ambos esses comandos são obrigatórios em todos os comandos de decisão desta linguagem.

Observações

- O compilador deverá aceitar todos os programas sintaticamente corretos, convertê-los para o formato de programa-objeto e prepará-los para a execução.
- Não está indicada, na especificação desta linguagem, a detecção de nenhum erro de execução, ficando portanto a inclusão desta função a critério de quem elaborar o compilador.

Casos anormais

Casos anormais deverão sofrer um tratamento pré-determinado:

- **em expressões que causem "overflow", esta ocorrência é ignorada.**
- **variáveis não preenchidas pelo programa são associadas ao valor que estiver contido na posição de memória na qual tenham sido alocadas.**
- **uma operação de divisão por zero não é executada, e nesse caso o próprio dividendo será considerado como resultado.**
- **em uma operação de divisão cujo resultado seja fracionário, apenas a parte inteira do quociente é considerada como sendo o resultado.**
- **nas operações de multiplicação, os resultados ocupam o dobro dos bits ocupados pelos fatores, mas nesta implementação apenas a palavra menos significativa resultante dessa operação é considerada como sendo o resultado.**
- **desvios indexados com índice inválido são ignorados, resultando um comando inócuo.**
- **na leitura, são considerados apenas os bits correspondentes à palavra menos significativa do dado fornecido, caso este tenha amplitude muito grande.**

Especificação da linguagem-objeto

Especificação da linguagem de saída

Para determinar o conjunto de instruções da máquina-alvo a ser empregado pelo compilador, optou-se por reduzir a um pequeno número as instruções utilizadas, de tal modo que o subconjunto assim escolhido possa ser facilmente adaptado para alguma arquitetura existente, a ser utilizada em uma implementação.

Escolhe-se ainda um conjunto muito reduzido de pseudo-instruções, composto de pseudo-instruções típicas dos montadores usualmente disponíveis.

Para uma implementação real, eventuais discrepâncias entre o conjunto de códigos aqui proposto e o disponível na máquina podem ser contornados com facilidade através de adaptações triviais.

As instruções de máquina a serem utilizadas são todas instruções de referência à memória, com endereçamento direto ou imediato, e estão relacionadas na tabela seguinte:

Linguagem da máquina virtual

Conjunto minimalista de instruções da máquina-alvo

Instrução		Interpretação
LDA	X	Obtém o conteúdo da posição X, e deposita no acumulador.
STA	X	Armazena o conteúdo do acumulador na posição X.
ADA	X	Soma ao acumulador o conteúdo da posição X.
SUB	X	Subtrai, no acumulador, o conteúdo da posição X.
MUL	X	Multiplica, no acumulador, o conteúdo da posição X.
DIV	X	Divide o acumulador pelo conteúdo da posição X.
CALL	X	Chama a subrotina X (da biblioteca de execução).
BRU	X	Desvia incondicionalmente para a posição X.
BRN	X	Desvia para a posição X se o conteúdo do acumulador for negativo.
BRZ	X	Desvia para a posição X se o conteúdo do acumulador for zero.
BRP	X	Desvia para a posição X se o conteúdo do acumulador for positivo.

Pseudo-instruções

Conjunto de pseudo-instruções do montador

Pseudo-instrução	Interpretação
LBL X	**Atribui ao endereço da próxima instrução o rótulo X.**
DS X	**Define posição de memória com conteúdo inicial igual ao número X.**
END	**Demarca o final físico do programa-fonte.**
EXT X	**Define o rótulo X como nome de rotina externa, chamada pelo código-objeto (rotina de biblioteca).**

Formatação do código simbólico

- Cada linha do código simbólico pode conter uma ou mais instruções e/ou pseudo-instruções separados entre si por um ponto e vírgula, ordenadas da esquerda para a direita, e as linhas são ordenadas de cima para baixo.
- Comentários podem ser inseridos na forma de um texto, introduzido por %, que se estende até o final da linha.
- Operandos referentes a posições de memória são representados por identificadores.
- Identificadores representados por # seguido de um inteiro simbolizam endereços de memória conhecidas apenas pelo compilador.
- Operandos imediatos representam valores e não endereços, e são denotados por = seguido de um inteiro.
- Como no texto-fonte, inteiros são denotados em decimal

Linguagem da máquina abstrata MVN

Mnemônicos da ling. máquina MVN

Operação **0** **Jump** Mnemônico **JP**	Operação **1** Jump if **Zero** Mnemônico **JZ**	Operação **2** Jump if **Negative** Mnemônico **JN**	Operação **3** Load **Value** Mnemônico **LV**
Operação **4** **Add** Mnemônico **+**	Operação **5** **Subtract** Mnemônico **–**	Operação **6** **Multiply** Mnemônico *****	Operação **7** **Divide** Mnemônico **/**
Operação **8** **Load** Mnemônico **LD**	Operação **9** Move to **Memory** Mnemônico **MD**	Operação **A** **Subroutine** Call Mnemônico **SC**	Operação **B** **Return** from Sub. Mnemônico **RS**
Operação **C** **Halt** Machine Mnemônico **HM**	Operação **D** **Get** Data Mnemônico **GD**	Operação **E** **Put** Data Mnemônico **PD**	Operação **F** **Operating** System Mnemônico **OS**

Mnemônicos disponíveis para a MVN

```
; MNEMÔNICOS    CÓDIGO    INSTRUÇÃO / PSEUDO-INSTRUÇÃO
;
;    JP    J      /0xxx     JUMP INCONDICIONAL
;    JZ    Z      /1xxx     JUMP IF ZERO
;    JN    N      /2xxx     JUMP IF NEGATIVE
;    LV    V      /3xxx     LOAD VALUE
;    +     +      /4xxx     ADD
;    -     -      /5xxx     SUBTRACT
;    *     *      /6xxx     MULTIPLY
;    /     /      /7xxx     DIVIDE
;    LD    L      /8xxx     LOAD FROM MEMORY
;    MM    M      /9xxx     MOVE TO MEMORY
;    SC    S      /Axxx     SUBROUTINE CALL
;    RS    R      /Bxxx     RETURN FROM SUBROUTINE
;    HM    H      /Cxxx     HALT MACHINE
;    GD    G      /Dxxx     GET DATA
;    PD    P      /Exxx     PUT DATA
;    OS    O      /Fxxx     OPERATING SYSTEM CALL
;
;    @     @                ORIGIN
;    #     #                END
;    K     K                CONSTANT
```

Listagem

Na listagem, é impresso o número da linha do texto-fonte, seguido de uma cópia da linha em questão. Eventuais erros de compilação, provenientes da análise daquela linha, são impressos logo a seguir.

A supressão da listagem não deve suprimir as mensagens que reportam os erros detectados pelo compilador.

Em uma implementação desses mecanismos, as variáveis LIST e MIX seriam controladas pela rotina de leitura de programa-fonte: a cada linha lida, consulta-se a variável LIST, imprimindo-se a imagem da linha no dispositivo de saída de listagens caso LIST=TRUE.

Consulta-se a variável MIX, e, caso MIX=TRUE, emite-se uma cópia da linha, na forma de um comentário, no dispositivo de saída do código-objeto.

Controles de Geração de Código

As variáveis PACK e CODE são controladas pelas rotinas de geração de código: a cada vez que uma dessas rotinas é chamada para gerar algum código, consulta-se inicialmente a variável CODE.

Caso CODE=FALSE, nada é gerado.

Caso CODE=TRUE, efetua-se a emissão do código solicitado no dispositivo de saída do código-objeto.

Consulta-se então a variável PACK.

Caso PACK=TRUE, aguarda-se que a linha de código-objeto seja totalmente preenchida antes que seja emitida uma mudança de linha.

Caso PACK=FALSE, gera-se uma nova linha a cada código gerado.

No exemplo a ser detalhado a seguir estas variáveis não serão utilizadas, uma vez que correspondem a um nível de detalhamento do compilador mais refinado que o adotado para o restante do exemplo.

Convém, entretanto, mencionar que os valores dessas quatro variáveis podem ser fornecidos pelo programador ao compilador através da leitura de uma linha especial de controle (por exemplo, a linha que precede a primeira das linhas que compõem o programa a ser compilado).

Analisador Léxico

Projeto

Através da inspeção da gramática BNF da linguagem a ser compilada, podem-se levantar os terminais correspondentes às palavras reservadas, sinais de pontuação, letras e dígitos.

Convém observar que é grande o número de ativações do analisador léxico durante a compilação, e a necessidade de uso de "look-ahead" para a distinção entre identificadores e palavras reservadas, e por isso, buscar a obtenção de uma boa eficiência.

Ocorre também um grande uso de identificadores e números na gramática, e por isso consideram-se como terminais tanto os identificadores como os números, distinguindo, a posteriori, por consulta a tabela, entre palavras reservadas e identificadores.

Tratamento semelhante pode ser dado aos terminais correspondentes a símbolos compostos.

Gramática Léxica, em Notação de Wirth

(repetida)

```
Tokens não triviais:
identificador, número
```

```
(18) identificador = letra { letra | dígito }.
(19) número = dígito { dígito }.
(20) letra = A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z.
(21) dígito = 0|1|2|3|4|5|6|7|8|9.
```

```
Palavras reservadas:
"END", "LET", "GO", "TO", "OF", "READ", "PRINT", "IF", "THEN", "ELSE"
```

```
Operadores, separadores e outros sinais:
";", ":", "+", "-", "*", "/", "(", ")", ",", ">", "=", "<"
```

```
Símbolos compostos:
":=", "GO" "TO"
```

```
Símbolos filtrados:
Branco, fim de linha, comentário
```

Observações

Brancos, sinais de mudança de linha e comentários inseridos no texto-fonte serão eliminados, e desta maneira o analisador léxico terá o comportamento de um fIltro.

A divisão do texto em seus componentes básicos utilizará o critério do comprimento máximo, ou seja, só será considerado identificado um átomo no instante em que o próximo caractere de entrada, ainda não analisado, não puder ser integrado ao mesmo.

Assim sendo, não há detecção de erros léxicos: todo símbolo que não puder ser acoplado a um átomo será considerado como símbolo inicial do próximo átomo a ser extraído, ou de algum separador.

Para efetuar a análise léxica, será desenvolvido um transdutor finito, cujo alfabeto de entrada é o conjunto de todos os caracteres da linguagem, bem como todos os caracteres que poderão figurar nos comentários, e o seu alfabeto de saída, o conjunto dos átomos relacionados anteriormente.

Representação dos átomos (*tokens*)

Cada átomo será representado por um par de números inteiros, o primeiro correspondente ao tipo de átomo, e o segundo, a uma informação complementar, conforme as convenções mostradas na seguinte tabela:

Terminal	Tipo do Átomo	Informação Complementar
palavra reservada	a própria palavra reservada	irrelevante
identificador	<<identificador>>	índice do identificador na tabela de símbolos
número inteiro	<<número>>	o valor numérico associado ao número
sinal	o próprio sinal	irrelevante
outro símbolo	o próprio símbolo	irrelevante

Observações

Uma tabela de símbolos é utilizada para coletar todos os identificadores declarados no programa, e possui uma correspondência direta com uma tabela de atributos, utilizada pelas ações semânticas.

Pode-se admitir a disponibilidade de rotinas de acesso às tabelas de símbolos e de palavras reservadas.

Uma das rotinas se encarrega de efetuar buscas de identificadores nestas tabelas, retornando a informação do local em que o identificador foi encontrado, ou então a informação de que o identificador não consta na tabela.

Neste caso, outra rotina deve ser utilizada para inserir um identificador na tabela de símbolos.

Para uma primeira implementação, não há necessidade de algoritmos rebuscados, de modo que, para as especificações do projeto em questão, pode-se adotar um método de busca seqüencial, de fácil implementação.

Otimizações do compilador podem alterar, posteriormente, a organização de tais estruturas de dados e os métodos de busca a serem aplicados.

Com as considerações até aqui efetuadas, pode-se conceber o autômato finito que servirá como base para o transdutor que implementa o analisador léxico.

Com as considerações até aqui efetuadas, pode-se conceber o autômato finito que servirá como base para o transdutor que implementa o analisador léxico.

Cada um dos átomos formados por mais de um símbolo pode ser reconhecido através dos seguintes autômatos particulares: identificadores, números e o sinal composto de atribuição.

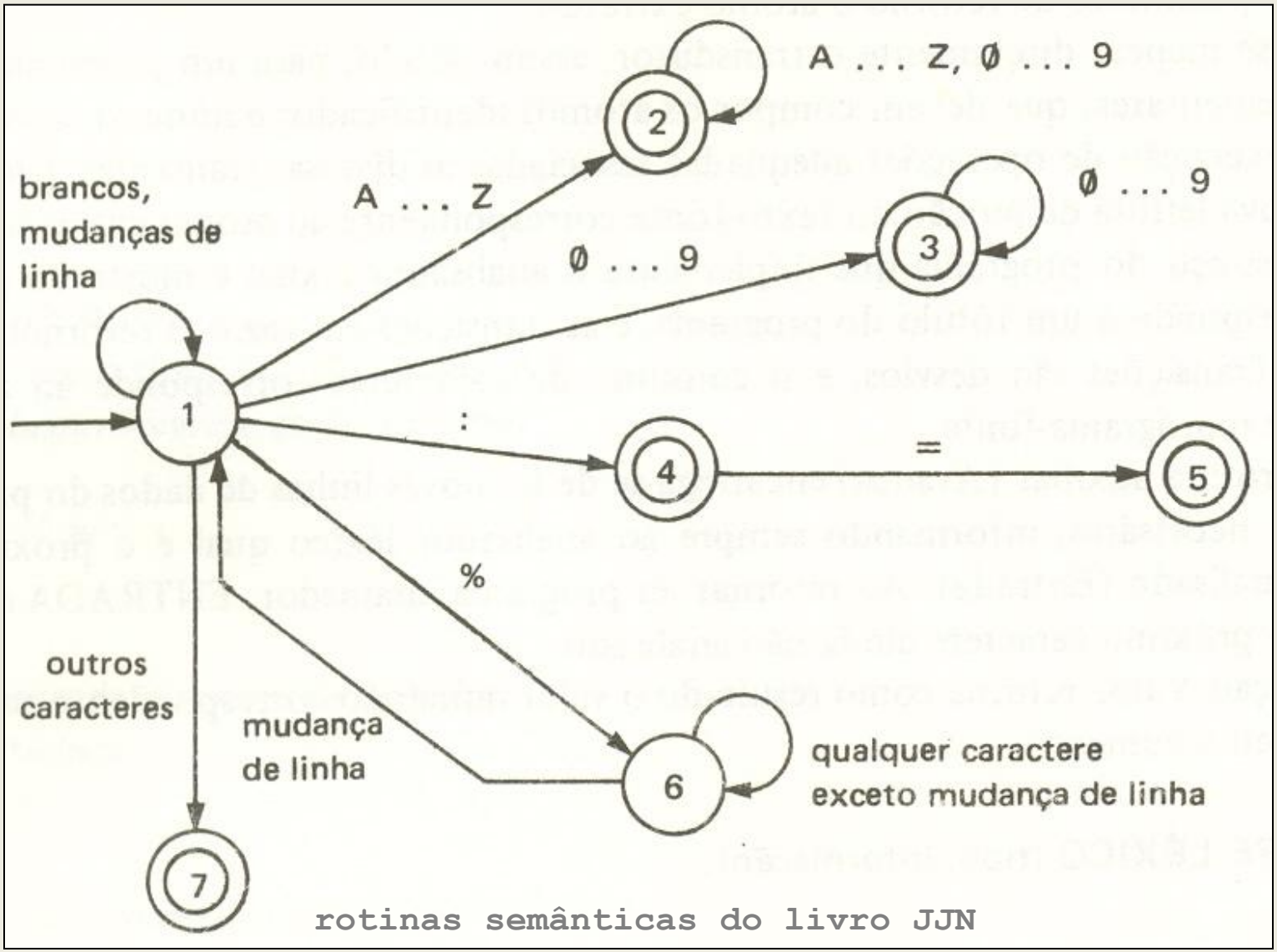
Separadores (comentários, brancos e mudanças de linha) podem ser descartados através de um filtro instalado no estado inicial do autômato que implementará o núcleo do analisador léxico.

Este filtro não deve apresentar estados finais, uma vez que nenhum átomo é reconhecido por ele, ao contrário do que ocorre em outros casos.

Outros símbolos encontrados são tratados isoladamente como átomos.

Compondo-se estes autômatos parciais, obtém-se um reconhecedor-base para o transdutor finito desejado.

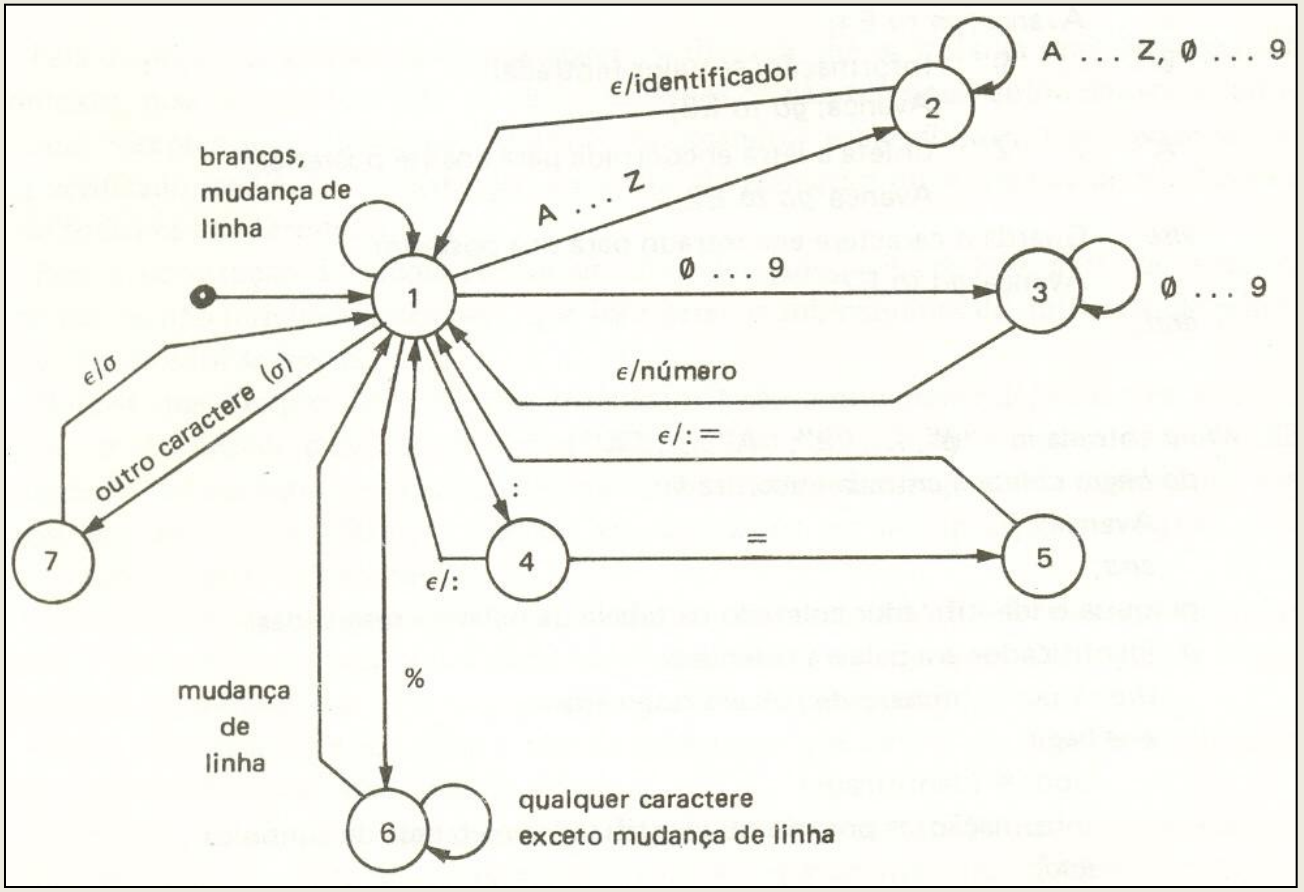
Analisador léxico – diagrama de transições do transdutor finito



rotinas semânticas do livro JJN

Para obter, finalmente, o transdutor, eliminam-se os estados finais, acrescentando-se uma transição em vazio partindo de cada um deles para o estado inicial.

Associa-se a tal transição a emissão do átomo que acaba de ser reconhecido e extraído



Para a operação deste transdutor, convenciona-se que, ao ser executada uma transição em vazio, é efetuado um retorno da subrotina que o implementa para o programa chamador, passando como parâmetro de retorno o átomo extraído.

Pode-se mapear diretamente o transdutor, assim obtido, para a forma de um programa.

As informações complementares, que devem compor os átomos identificador e número, devem ser obtidas através da execução de operações adequadas associadas às diversas transições, para que seja evitada uma nova leitura da porção do texto-fonte correspondente ao átomo extraído.

A seguir, esboça-se a lógica do analisador léxico.

Cada estado corresponde a um rótulo do programa, e cada transição em vazio, a um retorno ao programa chamador.

Transições não-vazias correspondem a desvios condicionais, e o consumo de caracteres corresponde ao avanço de um cursor sobre o programa-fonte, e à extração de um novo caractere.

Uma rotina auxiliar (Avança) encarrega-se de ler novas linhas de dados do programa-fonte sempre que necessário, informando sempre ao analisador léxico qual é o próximo caractere ainda não analisado (Entrada).

Ao retornar ao programa chamador, a variável ENTRADA está sempre se referindo ao próximo caractere ainda não analisado.

A função Valor retorna como resultado o valor numérico correspondente ao dígito fornecido como seu argumento.

O esboço de um programa que implementa o analisador léxico conforme acima convencionado é mostrado a seguir.

Pseudo-código do analisador léxico

```
procedure léxico (tipo, informação);
begin

E1: while entrada = branco or entrada = mudança de linha do
      Avança;
Case entrada of
begin "%" :  Avança; go to E6;
":" :    Avança; go to E4;
"0" : ... "9" : Informação := Valor(entrada); Avança; go to E3;
"A" : ... "Z" : Salva a letra, para uso posterior; Avança; go to E2;
                else: Salva o caractere, para uso posterior;
      Avança; go to E7;
end;

E2: while entrada in { "0", ... , "9", "A", ... , "Z" }
do begin coleta a entrada encontrada; Avança; end;
  pesquisa o identificador coletado na tabela de palavras
      reservadas;
  íf identificador era palavra reservada
     then tipo: = número da palavra reservada
     else begin
 tipo: = identificador
 informação: = posição do identificador na tabela de símbolos
 end;
  return ;
```

```
E3: while entrada in { "0", ... , "9" }
do begin  Informação: = Informação*10 + Valor(entrada);
      Avança; end;
tipo: = número;
return ;

E4: if entrada ≠ "="
then begin tipo := ":"; return; end
else begin Avança; go to E5; end;

E5: tipo := ":="; return ;

E6: while entrada ≠ mudança de linha do Avança; go to E 1 ;

E7: tipo := caractere anteriormente guardado; return;

end léxico;
```

(Esta versão do analisador léxico está projetada para ser usada como subrotina do analisador sintático.)

FIM da PARTE 1