PCS 3216 – Sistemas de Programação

Aula 19

DETALHAMENTO DE UM MÉTODO DE COMPILAÇÃO

DETALHAMENTO DE UM MÉTODO DE COMPILAÇÃO

- Esboça-se aqui um **roteiro de projeto e de implementação** de um processador de linguagem de alto nível.
- Para isso, parte-se de uma **definição formal da sintaxe** da linguagem, utilizando-se para isso, como meta-linguagem, a **Notação de Wirth**, com algumas modificações que a tornam mais adequada à geração automática de reconhecedores sintáticos baseados em autômatos de pilha estruturados.
- A organização deste tópico segue a seguinte sequência:
  - **mapeamento** de gramáticas em reconhecedores
  - construção de **analisadores léxicos**
  - implementação de **analisadores sintáticos**
  - **análise semântica** e **geração de código**

MAPEAMENTO DE GRAMÁTICAS EM RECONHECEDORES

Mapeamento de gramáticas em reconhecedores

- A construção de um processador para uma linguagem definida formalmente através de uma gramática é efetuada, com frequência, na forma de compiladores ou interpretadores **dirigidos por sintaxe**.
- Neste esquema, o processador da linguagem é centrado no **reconhecedor sintático** da mesma, que assume desta forma um papel importante como elemento de controle da compilação.
- A compilação dirigida por sintaxe rege uma substancial parcela dos processadores de linguagens de programação existentes, e esse forte atrativo se deve à sua naturalidade e facilidade de compreensão.

- O método aqui apresentado, embora aplicável à obtenção de outros tipos de compiladores, tem uma grande afinidade com o esquema dos **compiladores dirigidos por sintaxe**, e por esta razão nossa primeira atenção está voltada à obtenção de reconhecedores sintáticos para a linguagem em estudo, de cuja gramática dispomos.
- Para isto, a providência inicial é a de promover uma **manipulação da gramática** que define a linguagem, de forma que ela se torne mais adequada à geração de um reconhecedor para a mesma.

- Descreve-se em seguida, em palavras, a **intuição do método** escolhido para uso neste estudo.
- Detalhes adicionais estão disponíveis aos interessados em nosso livro-texto ***Introdução à Compilação***, e são exemplificados de forma simplificada na seção “Exemplo Completo” deste material.
- A notação adequada para a aplicação do método adiante apresentado é a meta-linguagem denominada **Notação de Wirth Modificada**, conforme já foi anunciado anteriormente.

- Sendo as diversas metalinguagens usuais muito semelhantes entre si, se a linguagem estiver definida em qualquer outra notação, poderá ser convertida com muita facilidade para a **Notação de Wirth**.
- A seguir, a gramática deve ser manipulada de forma que sejam **eliminadas auto-recursões** que não caracterizem aninhamentos, redefinindo-se na **forma iterativa** todos os não-terminais que estejam representados usando esse tipo de recursão.

Uma Linguagem descrita em Notação de Wirth

(1) programa = seqüência-de-comandos "END".
(2) seqüência-de-comandos = comando { ";" comando }.
(3) comando = { rótulo ":" } [atribuição | desvio I leitura I impressão I decisão].
(4) atribuição = "LET" identificador ":=" expressão.
(5) expressão = termo { ("+" I "-") termo }.
(6) termo = fator { ("*" I "/") fator }.
(7) fator = identificador I número I "(" expressão ")".
(8) desvio = "GO" "TO" (rótulo I 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 = ">" I "=" | "<".

(18) identificador = letra { letra I 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.

- O passo seguinte consiste em **eliminar todos os não-terminais desnecessários** – são essenciais apenas a **raiz da gramática** e aqueles que definam **aninhamentos independentes** – preservando-se aqueles que, por opção de projeto, ou ainda por preferências do projetista, se desejem manter.
- A **preservação de não-terminais não-essenciais** deve ser efetuada **com critério**, de modo que não resulte prejuízo à clareza nem à eficiência do reconhecedor produzido.

- Pode-se a seguir efetuar uma **busca de definições cíclicas**, ou seja, de iterações sintáticas, denotando-as de forma tal que se torne explícita essa característica, para tornar mais imediata a geração direta de um reconhecedor eficiente.
- A gramática, neste ponto, estará preparada para ser **convertida em um autômato de pilha estruturado**.

- No caso particular de ser a linguagem representada **apenas em função de terminais**, ou seja, se a raiz da gramática não for auto-recursiva, e se for também o único não-terminal restante, tem-se em mãos uma linguagem regular, e o resultado do mapeamento será um **autômato finito**.
- Neste caso, com facilidade é possível aplicar a esse autômato **procedimentos clássicos de minimização**, para obter sua forma mínima única, preconizada pela teoria.

- Para a obtenção do autômato de pilha estruturado a partir da gramática, deve ser efetuada a **identificação dos estados** através da análise das construções da metalinguagem, e aplicadas regras de **mapeamento direto** de cada construção da notação **para um padrão correspondente de autômato**, obtendo-se assim o projeto esquemático do reconhecedor que se deseja.

- Os **detalhes** desse procedimento construtivo são apresentados adiante, **em exemplo ilustrativo**.
- Uma **otimização** subsequente das diversas submáquinas assim obtidas, efetuada através de **métodos clássicos de redução de autômatos** finitos, pode levar a formas mais eficientes do reconhecedor, embora pela aplicação das regras de mapeamento indicadas isto só seja necessário em casos especiais.

CONSTRUÇÃO DE ANALISADORES LÉXICOS

Construção de analisadores léxicos

- A construção de um **núcleo de processador de linguagem** dirigido por sintaxe depende fundamentalmente da obtenção de um reconhecedor sintático que seja capaz de formar um **substrato** para as ações de **geração de código**.
- O reconhecedor sintático, por sua vez, tem seu funcionamento vinculado ao **recebimento dos átomos** contidos no texto-fonte, em função dos quais é capaz de **promover as transições** que efetuarão o **reconhecimento** propriamente dito daquele texto.

- O **analisador léxico** é exatamente o elemento do processador de linguagem que se responsabiliza pela tarefa de **extrair do texto** de entrada os seus **átomos**, **classificá-los** e fornecê-los ao reconhecedor sintático.
- Do ponto de vista da análise léxica, o texto-fonte é considerado como uma simples cadeia de caracteres.

- Cadeias simbólicas especiais, formadas pela concatenação dos caracteres do texto-fonte segundo leis de formação bem determinadas, constituem os **átomos**, que o reconhecedor sintático utiliza como partículas **indivisíveis**.
- Tipicamente são considerados átomos: **identificadores**, números **inteiros sem sinal**, sinais de **operação** ou de **pontuação**, palavras-chave ou **palavras reservadas**, caracteres **especiais**, etc.

- Os átomos são encontrados justapostos ou então separados entre si por meio de **separadores** - espaços, comentários, marcas de final de linha, etc.
- Separadores não contribuem para o processo de compilação, e por isso são, em geral, simplesmente **descartados pelo analisador léxico**.
- As leis de formação que regem a **sintaxe dos átomos**, neste nível de análise, são sempre **muito simples**, e podem ser definidas como **gramáticas regulares**, logo é possível tratar a extração dos átomos por meio de um mecanismo dirigido por sintaxe, controlado por um **autômato finito**.

- Isto é providencial, uma vez que os **analisadores léxicos** são procedimentos ativados milhares de vezes durante a compilação de um programa típico, e portanto representam um forte **gargalo** no processamento de uma linguagem.
- Assim sendo, convém que os analisadores léxicos sejam construídos de uma forma bastante **cuidadosa e otimizada**, para que não venham a comprometer a eficiência de todo o processador da linguagem.

- O método para a obtenção do autômato finito a ser usado como núcleo do analisador léxico pode ser o mesmo que é adotado para a parte sintática.
- Para que isto seja possível, é necessário dispor de uma **descrição formal da linguagem**, vista sob o enfoque da análise léxica.
- Assim, devem ser levantadas as formas gerais dos componentes básicos da linguagem, considerando-se como alfabeto o conjunto de caracteres disponíveis ao programador, e definindo-se sintaticamente todos os outros átomos da linguagem, comentários, separadores etc.

- De posse da **gramática léxica**, constrói-se o **autômato** correspondente, complementando-se tal reconhecedor com **rotinas de extração** das informações sintáticas e semânticas que são inerentemente associadas aos diversos átomos da linguagem.
- A construção física de um analisador léxico será posteriormente delineada, no **exemplo ilustrativo** da aplicação do método que estamos estudando.

Gramática Léxica

átomo = reservada | sinal | pontuação | identificador | número.

reservada = “END” | “LET” | “GO” | “TO” | “OF” |

“READ” | “PRINT” | “IF” | “THEN” | “ELSE” .

sinal = “:=” | “+” I “-” | “*” I “/” | “(” | “)” | “>” I “=” | “<” .

pontuação = “;” | “:” | “,” .

identificador = letra { letra I dígito } .

número = dígito { dígito } .

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” .

dígito = “0” | “1” | “2” | “3” | “4” | “5” | “6” | “7” | “8” | “9” .

Pseudo-código de um 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 E1;
E7: tipo := caractere anteriormente guardado; return;
end léxico;
```

CONSTRUÇÃO DE RECONHECEDORES SINTÁTICOS

Implementação de reconhecedores sintáticos

- A construção de um **reconhecedor sintático** a partir de uma gramática segue a técnica descrita no início deste tópico, e consta de uma sequência de etapas que pode ser esquematizada como segue.
- Admite-se, para começar, que a linguagem esteja descrita formalmente na **Notação de Wirth**, a metalinguagem anteriormente mencionada.
- Se não for esse o caso, será preciso manipular inicialmente a gramática disponível para expressá-la na Notação de Wirth.

- Detalhes operacionais da construção física deste tipo de reconhecedores sintáticos podem ser encontrados mais adiante neste material, no qual é resumida uma variante do método apresentado na íntegra no livro ***Introdução à Compilação***.
- Examina-se inicialmente a raiz da gramática. Caso o não-terminal que a representa exiba algum aninhamento sintático, explícito ou implícito, deve ser adicionado um novo não-terminal para substituí-lo como raiz da gramática.
- Esse não-terminal deve ser formalizado através de uma produção única, que o defina como sinônimo da antiga raiz da gramática.

Uma das regras gramaticais de uma gramática na Notação de Wirth

expressão =

(identificador I número I "(" expressão ")")

{ ("*" I "/") (identificador I número I "(" expressão ")") }

{ ("+" | "-") (identificador | número I "(" expressão ")")

{ ("*" I "/") (identificador I número I "(" expressão ")" } }.

Notar que se trata de uma regra que depende apenas de terminais e do não-terminal **expressão**, de forma auto-recursiva central, portanto pode ser manipulada isoladamente sem a necessidade de considerar outras regras sintáticas da gramática.

- Caso a gramática esteja expressa em qualquer outra notação, deve ser convertida para a Notação de Wirth.
- Determinam-se todos os **não-terminais**, direta ou indiretamente **auto-recursivos**, montando-se uma árvore das dependências que guardam entre si na gramática, e verificando-se quais deles podem vir a reaparecer, após uma cadeia de derivações, em alguma situação aninhada.

- Determinam-se quais dos **não-terminais** da gramática são **essenciais**, já que podem ocorrer casos de dependência cíclica entre os membros do conjunto de não-terminais auto-recursivos.
- Além da **raiz da gramática**, que sempre é essencial, dentre os elementos de um ciclo, são considerados essenciais os não-terminais que figurem como nós da árvore gramatical, na posição mais próxima à raiz da árvore.
- A critério do projetista, podem ser incluídos nesse conjunto todos os **não-terminais adicionais** cuja preservação seja por qualquer razão considerada conveniente para o projeto.

- Através de substituições sucessivas, **eliminam-se** todos os **não-terminais que não sejam auto-recursivos.**
- **Eliminam-se auto-recursões que não sejam centrais**, convertendo em iterações as construções expressas recursivamente sem aninhamentos.
- Expressam-se os não-terminais auto-recursivos centrais não-essenciais **em função dos essenciais**.

- Eliminam-se todos os não-terminais não-essenciais restantes, por meio de **substituições sucessivas** de suas definições em todos os pontos em que ocorrem.
- Determina-se a **correspondência entre as construções sintáticas** que as expressões da metalinguagem representam **e as partes do autômato** que se está construindo, que sejam responsáveis pelo seu reconhecimento.

Associação de estados à regra

expressão

. (	. I	. |	. N	. |	. ⟨	. E	. ⟩	.)	.
0	0	2	0	3	0	4	5	6	1

. {	. (	. *	. |	. /	.)	. (	. I	. |	. N	. |	. ⟨	. E	. ⟩	.)	. }	.
1	7	7	9	7	10	8	8	12	8	13	8	14	15	16	11	7	

. {	. (	. +	. |	. -	.)	. (	. I	. |	. N	. |	. ⟨	. E	. ⟩	.)	.
7	17	17	19	17	20	18	18	22	18	23	18	24	25	26	21	

. {	. (	.*	. |	. /	.)	. (	. I	. |	. N	. |	. ⟨	. E	. ⟩	.)	. }	. }	.
21	27	27	29	27	30	28	28	32	28	33	28	34	35	36	31	27	17	

Diagrama de estados correspondente

(c) expressão

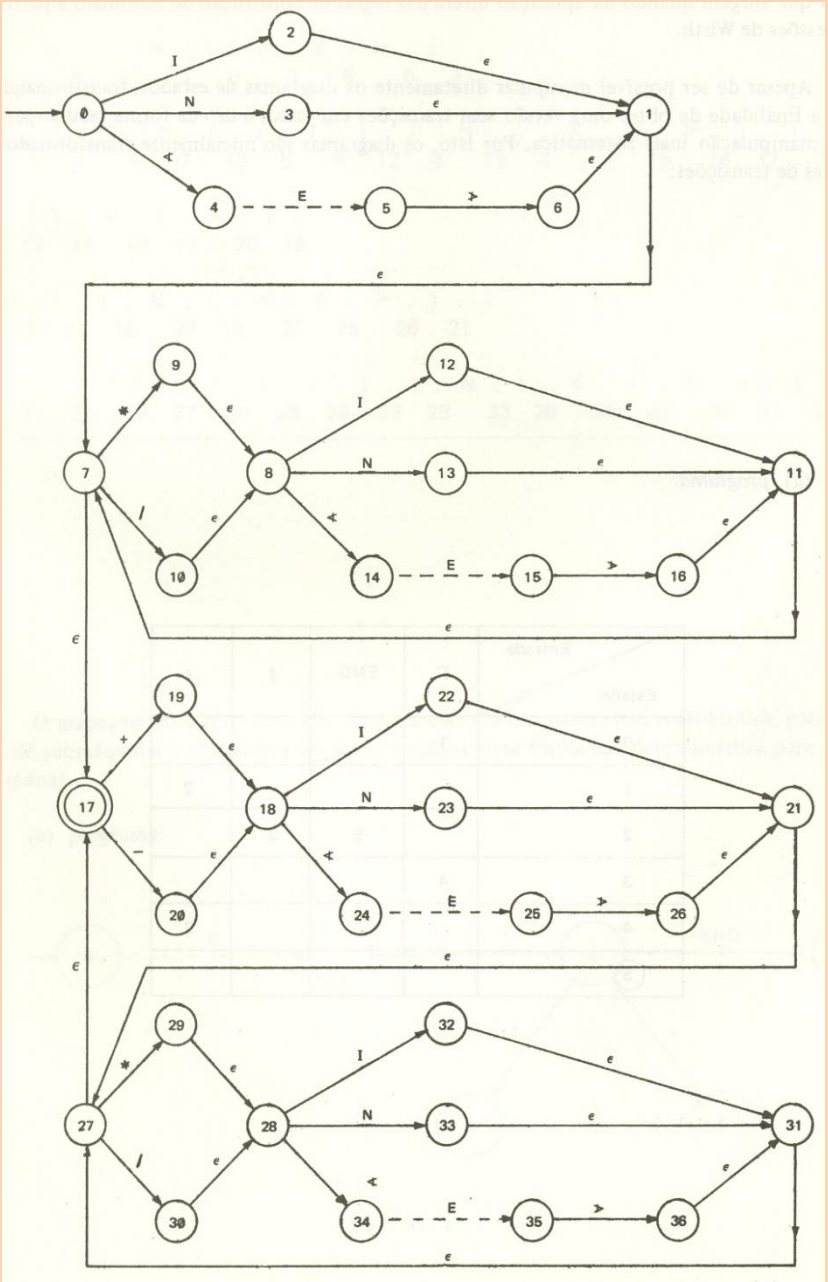


Tabela de Transições

expressão:

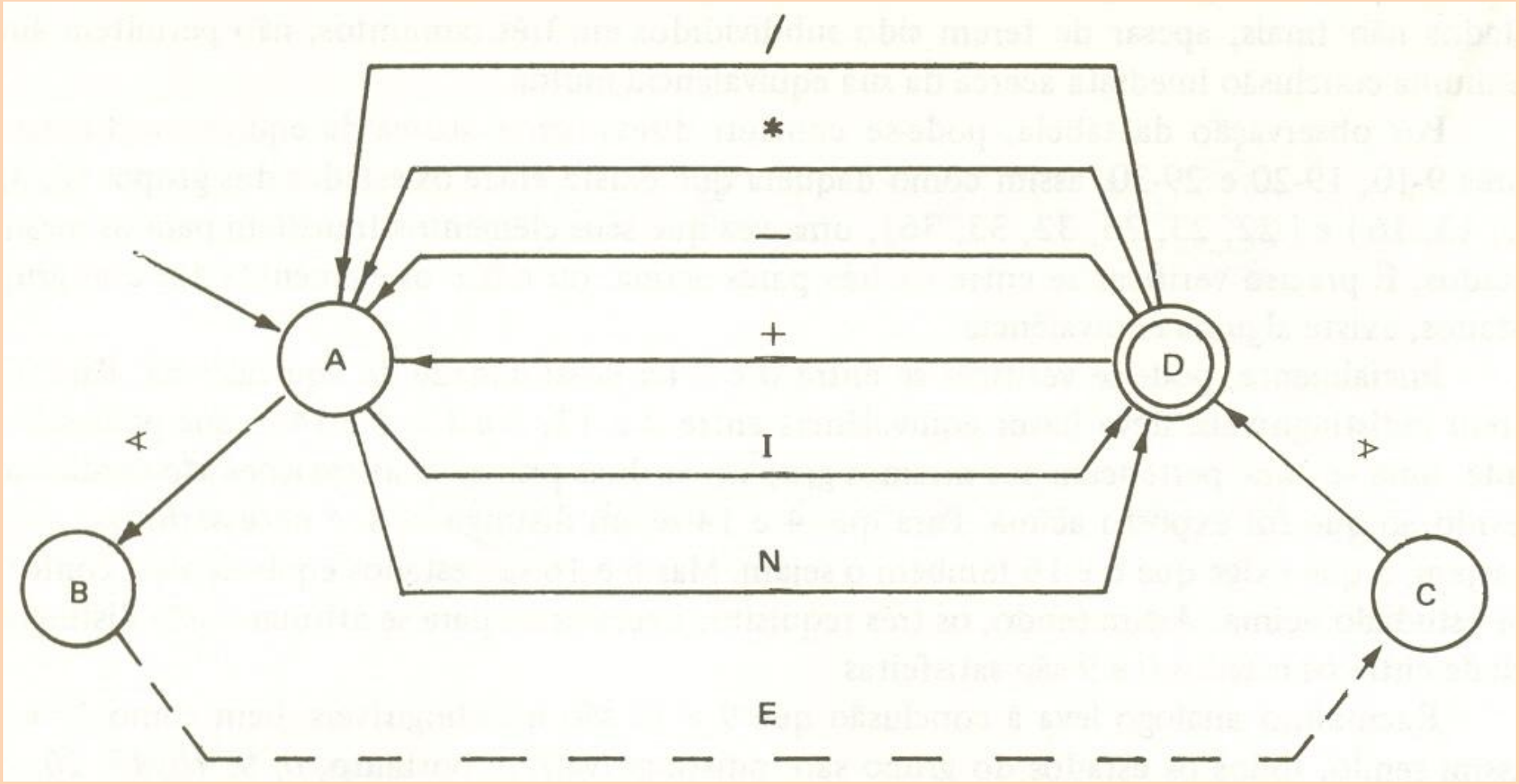
	I	N	<	E	>	+	-	*	/	ε
→0	2	3	4							
1										7
2										1
3										1
4				5						
5					6					
6										1
7								9	10	17
8	12	13	14							
9										8
10										8
11										7
12										11
13										11
14				15						
15					16					
16										11
17→						19	20			
18	22	23	24							
19										18
20										18
21										27
22						.				21
23										21
24				25						
25					26					
26										21
27								29	30	17
28	32	33	34							
29										28
30										28
31										27
32										31
33										31
34				35						
35					36					
36										31

- Determinam-se **equivalências entre os estados** das diversas partes assim construídas do autômato, para evitar a priori estados redundantes, transições em vazio e outras fontes desnecessárias de ineficiências.
- Se possível, necessário ou conveniente, identificam-se e **eliminam-se** eventuais fontes remanescentes de **não-determinismos**.

Expressão:

	I	N	<	E	>	+	-	*	/
→A	**D**	**D**	**B**						
B				**C**					
C					**D**				
D→						**A**	**A**	**A**	**A**

Expressão:



- **Minimizam-se as sub-máquinas** obtidas, se isso for considerado necessário ou conveniente.
- Incorporam-se **procedimentos de recuperação de erros** às sub-máquinas, se isso for considerado necessário ou conveniente.
- **Integram-se as sub-máquinas** assim obtidas, obtendo-se como resultado o reconhecedor final desejado.

- O autômato assim obtido corresponde a um **projeto conceitual completo do reconhecedor sintático** desejado.
- Resta transformá-lo mais uma vez, de tal forma que se obtenha uma **versão executável** em um computador.
- Para tanto, entre outras opções, é possível implementá-lo por meio de **tabelas de transições**, uma para cada sub-máquina, com a ajuda de um programa **interpretador dessas tabelas**, ou então escrevendo diretamente um programa que implemente o **algoritmo representado pelo autômato.**

- Para o **caso particular** de se obter uma única submáquina não-recursiva, ou seja, um **autômato finito**, a escrita direta de um código equivalente proporciona reconhecedores muito eficientes.
- Para o **caso geral**, implementações que sigam a tendência dos **reconhecedores descendentes recursivos** oferecem uma grande flexibilidade, dando margem à obtenção de compiladores muito **eficientes e de fácil manutenção**.

- A implementação através da **interpretação de tabelas** é de grande simplicidade e muito **prática**, sendo particularmente indicada para a **construção de protótipos** de processadores de linguagens, em aplicações didáticas, e durante a fase de desenvolvimento do compilador.

- Em qualquer das opções, deve-se notar que existe sempre a possibilidade de serem construídas **ferramentas que automatizem o procedimento**, delegando à máquina a tarefa da obtenção física dos programas e tabelas, liberando o projetista para a execução de atividades mais nobres e efetuando de forma mais confiável e rápida as operações de implementação física, que são sempre muito repetitivas e susceptíveis a falhas humanas.

ANÁLISE SEMÂNTICA E GERAÇÃO DE CÓDIGO

Análise semântica e geração de código

- Todas as atividades ligadas às **análises léxica e sintática** são, como foi mencionado, suficientemente **simples** e bem conhecidas, a ponto de permitirem uma fácil **formalização**, uma confortável **implementação** e até mesmo a **automatização** desses procedimentos de implementação.

- Embora possa ser também formalizada, a **semântica** das linguagens de programação **dificilmente apresenta uma simplicidade adequada** para permitir um tratamento similar.
- Por esta razão, apesar dos progressos alcançados nos estudos ligados à semântica, as partes do processador de linguagem a ela ligados têm sido geralmente **produzidas de forma artesanal**.

- As maiores dificuldades encontradas na automatização das atividades semânticas de um compilador ou interpretador residem na **complexidade** exigida das **metalinguagens** necessárias à descrição dos aspectos semânticos da linguagem cujo processador se deseja obter.

- Assim, enquanto não for inventada alguma notação que, de forma clara, simples e compacta, seja capaz de representar conjuntamente a sintaxe e a semântica de uma linguagem, a **geração automática de um compilador completo**, a partir de uma especificação formal da linguagem, continuará sendo uma **meta onerosa e pouco prática**.

- À parte dessa discussão, convém estudar um modo de **elaboração para as partes semânticas** do processador de linguagem, de forma que, integradas aos analisadores léxico e sintático, já discutidos, componham o compilador ou o interpretador que se esteja buscando.

- Do ponto de vista prático, todas as **necessidades do compilador**, que não sejam atendidas pelos analisadores léxico e sintático, têm sido **agrupadas nas denominadas rotinas semânticas.**
- Este nome não é adequado, porque **muitas dessas ações,** usualmente incluídas nessas rotinas, **não têm**, de fato, **cunho semântico** autêntico**.**

- Por exemplo, uma das mais importantes **atividades não-semânticas** que são usualmente executadas pelas rotinas de análise semântica encontradas na prática é aquela relacionada com os aspectos de **dependências de contexto** das linguagens de programação.

- Esta classe de atividades, nitidamente caracterizadas como sendo de **natureza sintática**, só não residem no módulo de análise sintática porque, por razões econômicas e de simplicidade, as **gramáticas** usadas na formalização das linguagens geralmente **restringem-se** a descrever apenas os aspectos da componente **livre de contexto** da linguagem.

- Entre as **tarefas sintáticas** (dependentes de contexto) executadas pelas rotinas semânticas do processador de linguagem, **destacam-se**:
  - Gerenciamento dos escopos dos identificadores, implementação da estrutura de blocos, passagem de parâmetros, modularização de programas, etc.
  - Criação e manutenção de **tabelas dos atributos** dos identificadores, com a finalidade de registrar **informações contextuais** ligadas a cada um dos identificadores no particular ponto do programa em que ocorre.

- Verificação da **compatibilidade** da utilização dos **tipos** dos objetos ao longo do programa, em função do contexto da sua **declaração** e das particulares construções sintáticas em que são empregados.
- **Limitação do número de ocorrências** de construções sintáticas nos comandos da linguagem, nos casos em que a representação livre de contexto permita um número arbitrário de tais construções.
- Verificação de **condições impostas pela implementação particular** e não pela linguagem propriamente dita, especialmente em relação a características da máquina ou do sistema operacional hospedeiro.

- Além dessas tarefas sintáticas, outras atividades de cunho não propriamente semântico são também executadas pelas chamadas rotinas semânticas:
  - A criação das **tabelas de símbolos** e sua manutenção, atividade que, a rigor, é de natureza léxica, uma vez que se refere ao gerenciamento de nomes e não propriamente dos objetos a que tais nomes se referem.

- A localização e o tratamento de **declarações contextuais** de objetos da linguagem, no caso em que esta disponibilize, através de sintaxe própria, a implicitação de certas declarações.
- A **alteração forçada do estado** em que se encontram os analisadores léxico e sintático, especialmente na ocasião da ocorrência de erros de sintaxe, visto que em geral um tratamento puramente sintático da recuperação de erros não costuma ser economicamente viável em grande número de casos da prática.

- Em adição às tarefas mencionadas, cabe naturalmente a essas rotinas a execução de ações **genuinamente semânticas**, tais como:
  - Escolher e gerenciar **representações para os diversos tipos de dados** disponibilizados pela linguagem, dando assim interpretação física aos objetos lógicos representáveis através dessa linguagem.
  - Efetuar o **gerenciamento do espaço de armazenamento** a ser utilizado, através do controle da utilização das áreas de memória lógica disponíveis por parte das estruturas de dados que representam os modelos matemáticos dos diversos objetos da linguagem.

– Representar o **ambiente de execução** da linguagem, implementando todas as rotinas auxiliares que se façam necessárias à execução correta do programa-objeto:
  - rotinas matemáticas em geral
  - administradores dos recursos da linguagem
  - manipulação dos objetos
  - utilização dinâmica de memória
  - formatação de dados, etc.

- Promover a **passagem de parâmetros** de e para os procedimentos, nas diversas formas eventualmente permitidas pela linguagem:
  - passagem de parâmetros por nome,
  - passagem por valor,
  - passagem por referência, etc.
- Promover a **comunicação** do programa **com o sistema operacional**, com a finalidade de tornar disponível ao programador os recursos disponíveis no sistema hospedeiro.

– Efetuar a **interpretação ou a tradução** do programa-fonte para a forma de **código-objeto**: esta é, de fato, a atividade mais autenticamente semântica do processador de linguagem, uma vez que se propõe a dar um significado global às construções sintáticas da linguagem.

- Promover a **comunicação entre ambientes**, especialmente em casos de linguagens para processamento concorrente.
- Promover a **otimização de código**, através:
  - da **compactação** do mesmo,
  - da transformação do programa em equivalentes **mais rápidos,**
  - da **escolha adequada entre recursos** funcionalmente equivalentes do ambiente hospedeiro, quer em nível local quer envolvendo todo o programa.

Tabela de transições com as ações semânticas

Expressão

	I	N	<	E	>	+	-	*	/	Ação de Saída
→A	D/1	D/2	8/3							11
B				C/4						12
C					D/5					13
D→						A/6	A/7	A/8	A/9	10

1	empilha (pilha de operandos, identificador encontrado)
2	empilha (pilha de operandos, número encontrado)
3	empilha (pilha de operadores, "(")
4	nada executa
5	X5: consulta (pilha de operadores, Y); Se Y ≠ "(" : executa GERACÓDIGO, detalhada adiante; GO TO X5. Se Y = "(" : desempilha (pilha de operadores, Y);
6	X6: consulta (pilha de operadores, Y); Se Y for "+", "-", "*" ou "/": executa GERACÓDIGO, detalhada adiante; GO TO X6; Caso contrário: empilha (pilha de operadores, "+");
7	7 - X7: consulta (pilha de operadores, Y); Se Y for "+", "-", "*", ou "/": executa GERACÓDIGO, detalhada adiante; GO TO X7; Caso contrário: empilha (pilha de operadores, "-");
8	X8: consulta (pilha de operadores, Y); Se Y for "*" ou "/": executa GERACÓDIGO, detalhada adiante; GO TO X8; Caso contrário: empilha (pilha de operadores, "*");
9	X9: consulta (pilha de operadores, Y); Se Y for "*" ou "/": executa GERACÓDIGO, detalhada adiante; GO TO X9; Caso contrário: empilha (pilha de operadores, "/");
10	(* Esta rotina é associada ao final do reconhecimento da expressão *) X10: consulta (pilha de operadores, Y); Se Y não for "1": executa GERACÓDIGO, detalhada adiante, GO TO X10;
11	ERRO ("esperava-se identificador, número ou '(' neste ponto").
12	ERRO ("esperava-se uma expressão correta neste ponto").
13	ERRO ("esperava-se ')' neste ponto").

FIM