Contents

1 Tradução dirigida por sintaxe (Capítulo 4)

Tradução dirigida por sintaxe é uma técnica que permite realizar tradução (geração de código) concomitantemente com a análise sintática via ações semânticas que são associadas as regras de produção da gramática. A execução dessas ações pode gerar ou interpretar código, armazenar informações na tabela de símbolos, emitir mensages de erro e etc.

Os símbolos gramaticais passarão a conter atributos capazes de armazenar valores durante o processo de reconhecimento.

1.1 Esquemas de tradução

Um esquema de tradução é uma extensão de uma gramática livre de contexto, extensão essa realizada através da associação de atributos aos símbolos gramaticais e de ações semânticas as regras de produção. Um atributo de um símbolo pode conter um valor numérico, uma cadeia de caracteres, um tipo de dado, um endereço de memória e etc. Ações semânticas podem ser avaliações de atributos ou chamadas a procedimentos e funções. Com exceção dos atributos dos terminais que são calculados pelo analisador léxico,

os valores dos demais atributos e variáveis são calculados durante a execução das ações semânticas.

Atributos podem ser sintetizados ou herdados. Dado uma sentença de entrada, a ela irá corresponder uma árvore de derivação, a qual será construída durante o processo de análise. A execução das ação ssemânticas irá definir os valores dos atributos dos nós da árvore para essa sentença de entrada. Deve ser observado que todas as informações referentes a uma sentança estão contidas na sentença original. Isto é, na sequencia de tokens da cadeia de entrada. Como os tokens ficam nas folhas da árvore de derivação, isso significa que todas as informações para um analisador estão originalmente nessas folhas.

As regras semânticas computam valores de atributos a partir de outros atributos e isso estabelece uma dependência entre as regras, pois um atributo só pode ser calculado depois que todos os atributos dos quais ele depende tiverem sido calculados. Portanto, a ordem em que as regras de produção são usadas no processo de reconhecimento pode não ser necessariamente a mesma ordem em que as ação ssemânticas correspondentes são executadas. A ordenação correta para realizar os cálculos dos atributos é representada através de um grafo dirigido, chamado de grafo de dependência. Esse grafo determina uma sequência de avaliação (ou escalonamento) para as regras semânticas.

Os esquemas S-atribuídos e os esquemas L-atribuídos possuem a característica de executar os esquemas de traduções e suas ações semânticas num único passo. A árvore de derivação que mostra os valores dos atributos associados a cada nó é chamada de árvore de derivação anotada.

Mesmo que o cálculo dos atributos de um símbolo possa variar em fução da localização do símbolo na árvore, cada instanciaçao desse símbolo terá sempre a mesma configuração (mesmo conjunto de atributos). Isto é, o número de atributos e os seus tipos não variam em função do ponto em que o símbolo aparece na árvore de derivação.

1.2 Gramática de atributos

As ações semânticas podem produzir efeitos colaterais tais como imprimir um valor, armazenar um literal em uma tabela ou em um arquivo, atualizar uma variável global, etc. Quando o esquema de tradução não produz efeitos colaterais, ele é dito ser uma gramática de atributos. Nesse caso, as ações semânticas são atribuições ou funções envolvendo unicamente os atributos do esquema (não são usadas variáveis globais, arquivos, etc).

2 Geração de código intermediário (Capítulo 5)

A geração de código intermediário é a transformação da árvore de derivação em um segmento de código. Esse código pode, eventualmente, ser o código objeto final, mas, na maioria dos casos constitui-se num código intermediário, pois a tradução de código fonte para objeto em mais de um passo apresenta algumas vantagens, como:

- Possibilita a otimização do código intermediário, de modo a obter-se o código objeto final mais eficiente;
- simplifica a implementação do compilador, resolvendo, gradativamente, as dificuldades da passagem de código fonte para objeto, já que o código fonte pode ser visto como um texto condensado que "explode" em inúmeras instruções de baixo-nivel;
- possibilita a tradução de código intermediário para diversas máquinas.

A desvantagem de gerar código intermediário é que o computador requer um passo a mais. A tradução direta do código fonte para o objeto leva a uma compilação mais rápida.

A única diferença entre o código intermediário e o código objeto final é que o código intermediário não especifica detalhes da máquina alvo.

2.1 Linguagens intermediárias

Os diferentes tipos de linguagem intermediária fazem parte de uma das três categorias abaixo.

- representações gráficas: árvore e grafo de sintaxe;
- notações pré e pós fixadas;
- código de três-endereços, quadras e triplas.

2.2 Árvore e grafo de sintaxe

Um grafo de sintaxe além de incluir as simplificações da árvore de sintaxe, faz a fatoração das subexpressões comuns, eliminando-as, conforme a figura abaixo.

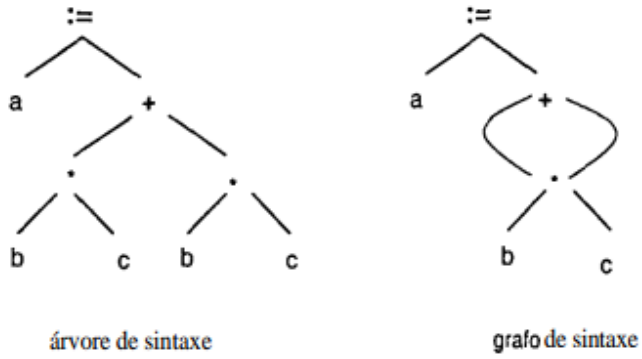

Figura 5.1 Árvore de sintaxe e grafo de sintaxe para a := (b * c) + (b * c).

2.2.1 Funções auxiliares

ptr geraFolha(Categ categ, String lexema) {...}
ptr geraNo(String opr, ptr node) {...}
ptr geraNo(String opr, ptr node1, ptr node2) {...}
Nota que dentro dessas funções se faz necessário verificar se os nós que estão querendo ser criados já existem, caso eles existam, basta retornar o ponteiro para esses nós já existentes.

2.2.2 Exemplo

Exemplo de gramática com ações semânticas que geram um grafo de sintaxe.
S = Ea '=' { s.ptr = geraNo(atr.lex, Ea.ptr); }
Ea = Ea1 'opa' Ta { Ea.ptr = geraNo(opa.lex, Ea1.ptr, Ta.ptr); }
Ea = Ta { Ea.ptr = Ta.ptr; }
Ta = Ta1 'opm' Fa { Ta.ptr = geraNo(opm.lex, Ta1.ptr, Fa.ptr); }
Ta = Fa { Ta.ptr = Fa.ptr; }
Fa = '(' Ea ')' { Fa.ptr = Ea.ptr; }
Fa = 'cten' { Fa.ptr = geraFolha('cten', cten.lex); }
Fa = 'id' { Fa.ptr = geraFolha('id', id.lex); }

2.3 Notações pré e pós-fixadas

Notações pré e pós-fixadas podem ser generalizadas para operadores **n-ários**. Para a avaliação de expressões desse tipo, pode-se utilizar uma pilha e um

processo que age do seguinte modo: lê a expressão da esquerda para a direita, empilhando cada operando até encontrar um operador. Encontrando um operador n-ário, aplica o operador aos n operandos do topo da pilha. Processamento semelhante pode ser aplicado para a avaliação de expressões pré-fixadas; nesse caso, a expressão é lida da direita para a esquerda.

Notação		
infixada	**pós–fixada**	**pré–fixada**
(a+b)*c	ab+c*	*+abc
a*(b+c)	abc+*	*a+bc
a+b*c	abc*+	+a*bc

Figura 5.2 Notações pós e pré fixadas

2.3.1 Exemplo

Exemplo de gramática com ações semânticas que geram uma notação pós-fixada.

S = Ea '=' { print(Ea.cod); }
Ea = Ea1 'opa' Ta { Ea.cod = Ea1.cod + Ta.cod + opa.lex; }
Ea = Ta { Ea.cod = Ta.cod }
Ta = Ta1 'opm' Fa { Ta.cod = Ta1.cod + Fa.cod + opm.lex; }
Ta = Fa { Ta.cod = Fa.cod; }
Fa = '(' Ea ')' { Fa.cod = Ea.cod; }
Fa = 'cten' { Fa.cod = cten.lex; }
Fa = 'id' { Fa.cod = id.lex; }

2.4 Código de 3 endereços e quadras

No código intermediário de 3 endereços, cada instrução faz referência a no máximo três variáveis(endereços de memória). O tipo de código intermediário que será utilizado na disciplina é o de quadras.

A principal diferença entre os dois é a ocupação de memória, que nos dias de hoje não é um problema, devido ao fato de que a maioria dos dispositivos tem memória o suficiente para suportar quadras.

2.4.1 Funções auxiliares

String geraTemp() {...}
String geraRot() {...}

void emiteRot() {...}

String geraCod(String operador, String operando1, String operando2, String destino) {...}

String geraCod(String operador, String operando, String destino) {...}

String geraGt(String rotulo) {...}

2.4.2 Exemplo

```
S = 'id' '=' Ea {
    S.cod = Ea.cod + geraCod('=', Ea.nome, id.lex);
    }
    Ea = Ea1 'opa' Ta {
    Ea.nome = geraTemp();
    Ea.cod = Ea1.cod + Ta.cod + geraCod(opa.lex, Ea1.nome, Ta.nome,
    Ea.nome);
    }
    Ea = Ta {
    Ea.nome = Ta.nome;
    Ea.cod = Ta.cod;
    }
    Ta = Ta1 'opm' Fa {
    Ta.nome = geraTemp();
    Ta.cod = Ta1.code + Fa.cod + geraCod(opm.lex, Ta1.nome,
    Fa.nome, Ta.nome);
    }
    Ta = Fa {
    Ta.nome = Fa.nome;
    Ta.cod = Fa.cod;
    }
    Fa = '(' Ea ')' {
    Fa.nome = Ea.nome;
    Fa.cod = Ea.cod;
    }
    Fa = '-' Fa1 {
    Fa.nome = geraTemp();
    Fa.cod = Fa1.cod + geraCod("-u", Fa1.nome, Fa.nome);
    }
    Fa = 'cten' {
    Fa.nome = cten.lex;
    Fa.cod = "";
```

}
Fa = 'id' {
Fa.nome = id.lex;
Fa.cod = "";
}

2.5 Ações semânticas para construção de tabela de símbolos

Esta seção apresenta esquemas de tradução que reconhecem declarações de variáveis e geram tabelas de símbolos. Inicialmente, é apresentado um esquema que gera tabelas de símbolos para programas monolíticos, isto é, formados por um único bloco. Posteriormente, esse esquema é estendido para permitir a geração de tabelas para programas bloco-estruturados com procedimentos aninhados.

2.6 Expressões lógicas e comandos de controle

Expressões lógicas são usadas como expressões condicionais em comandos de controle e em comandos de atribuição lógica. Nesta seção, apresentaremos dois tipos de traduções para expressões lógicas.

- Representação numérica: este método codifica numericamente as constantes true e false e avalia as expressões lógicas de forma numérica, ficando o resultado de avaliação numa variável temporária.
- Representação por fluxo de controle: este método traduz expressões lógicas para instruções if e goto que desviam a execução do programa para pontos distintos, caso o resultado da avaliação seja true ou false.

2.6.1 Representação numérica

Esquema de tradução para avaliação numérica de expressões lógicas. Note que o esquema de tradução abaixo gera código para expressões lógicas, supondo que as instruções geradas são armazenadas num vetor de quadruplas. A função geraCod, nesse caso, utiliza uma variável proxq para indicar o índice da próxima quadra disponível. Após gravar uma quadrupla, a função geraCod incrementa proxq.

Exemplo:
Eb = Eb1 'ou' Tb {
NTerm Eb;
Eb.nome = geraTemp();

Eb.cod = Eb1.cod + Tb.cod + geraCod('or', Eb1.nome, Tb.nome,
Eb.nome);
}
Eb = Tb { NTerm Eb = Tb; }
Tb = Tb1 'e' Fb {
NTerm tb;
tb.nome = geraTemp();
tb.cod = Tb1.cod + Fb.cod + geraCod('e', Tb1.nome, Fb.nome,
tb.nome);
}
Tb = Fb { NTerm Tb = Fb; }
Fb = 'nao' Fb1 {
NTerm Fb;
Fb.nome = geraTemp();
Fb.cod = Fb1.cod + geraCod('nao', Fb1.cod, Fb.nome);
}
Fb = '(' Eb ')' { NTerm Fb = Eb; }
Fb = Ea1 'opr' Ea2 {
NTerm Fb;
Fb.nome = geraTemp();
Fb.cod = Ea1.cod + Ea2.cod + geraCod(opr.lex, Ea1.nome, Ea2.nome,
geraRot(proxq+3)) + geraCod('=', '0', Fb.nome) +
geraGT(geraRot(proxq+2)) + geraCod('=', '1', Fb.nome);
}
Fb = 'verd' { NTerm fb; fb.nome = geraTemp(); fb.cod =
geraCod('=', '1', Fb.nome);
}
Fb = 'falso' { NTerm fb; fb.nome = geraTemp(); fb.cod =
geraCod('=', '0', Fb.nome);
}
Esquema de tradução para gramáticas com while
Sent = 'enquanto' Eb 'faça' LSent 'fim' {
String inicio = geraRot();
String fim = geraRot();
Sent.cod = emiteRot(inicio) + Eb.cod + geraCod('==', Eb.nome, '0',
fim) + LSent.cod + geraGT(inicio) + emiteRot(fim);
}

2.6.2 Representação por fluxo de controle

Este método traduz expressões lógicas para um código formado por instruções **if** e **goto**. São gerados rótulos que serão atributes chamados **true** e **false** para onde a execução deve ser transferida em caso de avaliação verdadeira ou falsa.

Exemplo:

Sent = 'enquanto' Eb 'repita' LSent 'fim' {
NTBool Eb; Eb.true = geraRot(); Eb.false = Sent.prox;
NTSent LSent; LSent.prox = Sent.prox;
Sent.cod = emiteRot(LSent.prox) + Eb.cod + emiteRot(Eb.true) + LSent.cod + geraGT(LSent.prox);
}
Sent = 'se' Eb 'entao' LSent Senao {
NTBool Eb; Eb.true = geraRot(); Eb.false = geraRot();
NTSent LSent; Lsent.prox = Sent.prox;
NTSent Senao; Senao.prox = Sent.prox;
Sent.cod = Eb.cod + emiteRot(Eb.true) + LSent.cod +
geraGT(LSent.prox) + emiteRot(Eb.false) + Senao.cod + geraGT(Senao.prox);
}
Senao = 'senao' LSent 'fim' {
NTSent LSent; LSent.prox = Senao.prox;
Senao.cod = Lsent.cod + geraGT(Lsent.prox);
}
Senao = 'fim' {
Senao.cod = "";
}
Eb = Eb1 'ou' Tb {
NTBool Eb1; Eb1.true = Eb.true; Eb1.false = geraRot();
NTBool Tb; Tb.true = Eb.true; Tb.false = Eb.false;
Eb.cod = Eb1.cod + emiteRot(Eb1.false) + Tb.cod;
}
Eb = Tb { NTBool Tb = Eb; }
Tb = Tb1 'e' Fb {
NTBool Tb1; Tb1.false = Tb.false; Tb1.true = geraRot();
NTBool Fb; Fb.false = Tb.false; Fb.true = Tb.true;
Tb.cod = Tb1.cod + emiteRot(Tb1.true) + Fb.cod;
}
Tb = Fb { NTBool Fb = Tb; }
Fb = 'nao' Fb1 {

```
NTBool Fb1; Fb1.true = Fb.false; Fb1.false = Fb.true;
Fb.cod = Fb1.cod;
}
Fb = Ea1 'opr' Ea2 {
NTerm Ea1, Ea2;
Fb.cod = Ea1.cod + Ea2.cod + geraCod(opr.lex, Ea1.nome,
Ea2.nome, Fb.true) + geraGT(Fb.false);
}
Fb = 'true' { Fb.cod = geraGT(Fb.true); }
Fb = 'false' { Fb.cod = geraGT(Fb.falso); }
```