Compiladores

Prof. Marc Antonio Vieira de Queiroz

Ciência da Computação - UNIFIL
LAB 7
marc.queiroz@unifil.br

30/04/2014

Roteiro I

Análise Sintática I

Por definição as linguagens de programação possuem regras precisas para descrever a estrutura sintática de programas bem formados.

Exemplo: Em C um programa é composto de funções, uma função é composta de declarações e comandos, um comando é formado a parti de expressões e assim por diante.

A estrutura sintática das construções de uma linguagem de programação é especificada pelas regras de uma gramática livre de contexto ou notação Backus-Naur Form (BNF).

Benefícios significativos para projetistas de linguagens e compiladores:

- Uma gramática provê uma especificação sintática precisa e fácil de entender para as linguagens de programação.

Análise Sintática II

- A partir de determinadas classes de gramáticas podemos construir automaticamente um analisador sintático eficiente que descreve a estrutura de um programa-fonte. Também podem ser detectas ambiguidades sintáticas, e outros erros.
- A estrutura imposta a uma linguagem por uma gramática devidamente projetada facilita a tradução de programas fonte para código objeto correto e detecção de erros.
- Uma gramática permite o desenvolvimento de uma linguagem iterativamente, possibilitando acrescentar novas construções para realizar novas tarefas.

O papel do analisador sintático I

- Recebe do analisador léxico uma cadeia de tokens representando o programa-fonte, e verifica se essa cadeia de token pertence à linguagem gerada pela gramática.
- Emitir mensagens para quaisquer erros de sintaxe, e se recuperar destes erros, a fim de continuar processando todo o programa.
- Programas bem formados: gera a árvore de derivação que é utilizada nos processos seguintes do compilador.

Existem três estratégias gerais de análise sintática para o processamento de gramáticas:

1. Universal: Métodos baseados nesta alternativa como o algoritmo Cocke-Younger-Yasami e o algoritmo de Earley podem analisar qualquer gramática. Porém são ineficientes para serem usados em compiladores de produção.
2. Descendentes: constroem as árvores de derivação de cima para baixo (raiz $\rightarrow$ folhas).
3. Ascendentes: fazem a análise de baixo para cima (folhas $\rightarrow$ raiz).

Em ambas as estratégias descendentes e ascendentes a entrada do analisador é consumida da esquerda para a direita, um símbolo de cada vez.

Obs: Os métodos ascendentes e descendentes mais eficientes funcionam apenas para subclasses de gramáticas (LL e LR) que são suficientes para descrever a maioria das construções sintáticas das linguagens de programação modernas.

Outras tarefas que podem ser executadas durante a análise sintática:

1. coleta de informações sobre tokens na tabela de símbolos.
2. verificação de tipo e análise semântica
3. geração do código intermediário

Gramáticas I

G1: Ver quadro (produções).
G2: (sem recursão à esquerda).
G3: Geração de árvores ambíguas.

Erros de sintaxe I

- Incluem ponto-e-vírgula mal colocados ou chaves extras ou faltando:
- Em C e Java a definição de um case sem um switch delimitando-o é um erro sintático.
- O recuperador de erros em um analisador sintático possui objetivos simples, mas desafiadores.

Recuperação no modo pânico I

- O analisador sintático descarta um símbolo de entrada de cada vez até que um dentre um conjunto de **tokens de sincronismo** seja encontrado.
- **tokens de sincronismo:** delimitadores cujo papel no programa-fonte é claro e não-ambíguo, variam com a linguagem.
- Exemplo: ; , { , }, $\cdots$
- Embora ignore uma quantidade considerável de símbolos terminais do programa-fonte sem se preocupar com erros adicionais, é simples e tem-se a garantia de não entrar em um loop infinito.

Recuperação em nível de frase I

- Ao detectar o erro, o analisador sintático pode realizar a correção local sobre o restante da entrada, ou seja, pode substituir o prefixo da entrada por uma cadeia que permita a análise.
- Ex: Substituir uma "vírgula" por "ponto-e-vírgula"
- excluir "ponto-e-vírgula" que sejam desnecessários
- inserção de "ponto-e-vírgula"
- **cuidado:** não criar loops infinitos - sempre inserindo símbolos na frente do símbolo de entrada corrente.
- **desvantagem:** dificuldade de lidar com situações que o erro real aconteceu antes do ponto de detecção.

Produções de erro I

Estende-se a gramática da linguagem com produções que geram construções erradas, antecipando os erros mais comuns. Um Analisador Sintático criado a partir de uma gramática estendida por estas produções de erro detecta os erros quando uma das produções é usada durante a análise, gerando o diagnóstico de erro apropriado sobre o que foi lido.

Correção Global I

Idealmente, o compilador deveria fazer o mínimo de mudanças possível no processamento de uma cadeia de entrada incorreta. Existem algoritmos que auxiliam na escolha de uma sequência de mudanças a fim de obter uma correção com custo global menor.

Desvantagem: métodos muito caros em termos de tempo e espaço de implementação. Técnicas de interesse meramente teórico.