Analisador Sintático parte III

Compiladores

Mariella Berger

A análise Bottom-Up

- Análise de empilhar e reduzir;
- Tenta construir uma árvore gramatical para uma cadeia de entrada começando pelas folhas e trabalhando árvore acima em direção à raiz;
- "Reduzir" uma cadeia w ao símbolo de partida de uma gramática.

A análise Bottom-Up

- Considere a gramática:

S -> aABe

A -> Abc | b

B -> d

- A sentença abbcde pode ser reduzida a S pelos seguintes passos:

abbcde

aAbcde

aAde

aABe

S

Pilha da Análise Sintática

- Quatro ações possíveis:
  - Empilhar
  - Reduzir
  - Aceitar
  - Erro.

Reduzir e Empilhar

	PILHA	ENTRADA	AÇÃO
(1)	$	$\mathbf{id}_1 + \mathbf{id}_2 * \mathbf{id}_3\$$	empilhar
(2)	$\$\mathbf{id}_1$	$+ \mathbf{id}_2 * \mathbf{id}_3\$$	reduzir por $E \rightarrow \mathbf{id}$
(3)	$\$E$	$+ \mathbf{id}_2 * \mathbf{id}_3\$$	empilhar
(4)	$\$E +$	$\mathbf{id}_2 * \mathbf{id}_3\$$	empilhar
(5)	$\$E + \mathbf{id}_2$	$* \mathbf{id}_3\$$	reduzir por $E \rightarrow \mathbf{id}$
(6)	$\$E + E$	$* \mathbf{id}_3\$$	empilhar
(7)	$\$E + E *$	$\mathbf{id}_3\$$	empilhar
(8)	$\$E + E * \mathbf{id}_3$	$\$$	reduzir por $E \rightarrow \mathbf{id}$
(9)	$\$E + E * E$	$\$$	reduzir por $E \rightarrow E * E$
(10)	$\$E + E$	$\$$	reduzir por $E \rightarrow E + E$
(11)	$\$E$	$\$$	aceitar

Fig. 4.22. Configurações de um analisador sintático de empilhar e reduzir para a entrada $\mathbf{id}_1 + \mathbf{id}_2 * \mathbf{id}_3$.

Introdução

- O programa Bison recebe um texto com a descrição de uma gramática e ações associadas.
- A gramática em questão é uma série de regras de formação de strings.
- As regras descrevem como formar strings válidas (produções) a partir do alfabeto da linguagem (tokens).

Introdução

- A cada possível produção da gramática está associada uma ação.
- As ações são escritas em C e indicam o que é que se deve fazer cada vez que se reconhece uma produção.
- O bison transforma esta descrição da gramática e ações associadas em um parser (programa capaz de analisar uma sequência de tokens de entrada, detectar produções e agir sobre elas).

Usando o Bison

- São 4 passos para criar um parser:
  - Escrever uma especificação de uma gramática no formato do bison. O arquivo terá a extensão .y.
  - Escrever uma especificação de um analisador léxico que pode produzir tokens; extensão .l.
  - Executar o bison sobre a especificação .y e o flex sobre a especificação .l.
  - Compilar e linkar os códigos fontes do parser, do analisador léxico e suas rotinas auxiliares.

Usando o Bison

- A saída do bison, **yy.tab.c,** contém a rotina **yyparse** que chama a rotina **yylex** para obter tokens.
- Como o Lex, o Bison não gera programas completos.
- A **yyparse** deve ser chamada a partir de uma rotina **main**.
- Um programa completo também requer uma rotina de erro chamada **yyerror**.

Escrevendo uma Especificação Bison

- Uma especificação bison descreve uma gramática livre do contexto que pode ser usada para gerar um parser.
- Ela possui elementos membros de 4 classes:
  - Tokens, que é o conjunto de símbolos terminais;
  - Elementos sintáticos, que são símbolos não terminais.
  - Regras de produção, que definem símbolos não terminais em termos de seqüência de terminais e não terminais;
  - Uma regra **start**, que reduz todos os elementos da gramática para uma regra.

Símbolos

- A cada regra está associado um símbolo não terminal (lado esquerdo).
- As definições (lado direito) consistem de zero ou mais símbolos terminais (tokens ou caracteres literais) ou não terminais.
- Tokens são símbolos terminais reconhecidos pelo analisador léxico, e só aparecem no lado direito das regras.
- A cada regra pode ser associada uma ação em C. Estas ações ficam entre chaves ({ }).

Símbolos

- Os nomes de símbolos podem ter qualquer tamanho, consistindo de letras, ponto, sublinhado e números (exceto na primeira posição).
- Caracteres maiúsculos e minúsculos são distintos.
- Os nomes de não terminais são usualmente especificados em minúsculos.
- Tokens, em maiúsculos.

Formato da especificação

- Uma especificação mínima em yacc consiste de uma seção de regras, precedida de uma seção de declaração de tokens reconhecidos pela gramática.
- O formato é similar ao formato do flex.

declarações

%%

regras de produção

%%

rotinas em C do usuário

Declarações

- Códigos em C entre `%{ %}` (como no flex)
- Definição de tokens: `%token T1`
- Definição de regras auxiliares para solução de ambiguidades:
  - `%left MAIS MENOS`
  - `%left VEZES DIVIDIR`
  - `%left MENOS_UNARIO`

Regras de Produção

/* uma gramática definida assim: */
<lado esquerdo> **:=** <alt1>|<alt2>|...|<altN>

/* transforma-se na seguinte especificação yacc */
<lado esquerdo>: <alt1> {/*ação 1*/}
| <alt2> {/*ação 2*/}
...
| <alt3> {/*ação N*/}
;

Regras de Produção

- Cada símbolo (terminal ou não) tem associado a ele uma pseudo variável;
  - O símbolo do lado esquerdo tem associada a ele a pseudo variável `$$`;
  - Cada símbolo i da direita tem associada a ele a variável `$i`;
  - `$i` contém o valor do token (retornado por `yylex()`) se o símbolo i for terminal, caso contrário contém o valor do `$$` do não terminal;
  - Ações podem ser executadas sobre estas variáveis.