UFRGS - INF01147 - Compiladores - 2012/1
Turma B - Prof. Marcelo Johann

Trabalho Prático - Especificação da Etapa 3:
Geração de Árvore Sintática Abstrata - AST

Resumo:
Na terceira etapa do trabalho de implementação de um compilador para a linguagem **de** é preciso guardar a árvore de derivações encontrada pelo analisador sintático da etapa2, gerado pelo *yacc* (ou *bizon*), a qual será usada posteriormente para análise semântica e geração de código. Esta árevore será verificada visualmente e também pelo processo de geração de um programa-fonte equivalente ao original, à partir da árvore encontrada.

Funcionalidades necessárias:
a. Corrijir a especificação das suas expressões em relação à etapa anterior, simplificando-as de forma que as produções utilizem a forma mais simples que é ambígua, e retirando a ambiguidade pela declaração da precedência correta dos operadores com as definições %left, %right ou %nonassoc ;
b. Implementar a estrutura da árvore de sintaxe, com funções de criação de nodo, impressão de um nodo e impressão da árvore ;
c. Colocar ações sintáticas ao lado das regras de produção descritas no arquivo para o *yacc*, as quais criam ou propagam os nodos da árvore, montando-a recursivamente segundo a análise. A maior parte das ações serão chamadas à rotina de criação de nodo;
d. Finalmente, você deve implemetar uma rotina que percorre a árvore e gera um programa fonte bastate semelhante, funcionalmente equivalente ao programa analisado;

Descrição da Árvore
A Árvore Sintática Abstrata, ou *AST*, do inglês, *Abstract Syntax Tree*, é uma árvore n-ária onde os nodos intermediários representam símbolos não terminais, os nodos folha representam *tokens* presentes no programa fonte, e a raiz representa o programa corretamente analisado. Essa árvore registra as derivações reconhecidas pelo analisador sintático, e torna mais fáceis as etapas posteriores de verificação e síntese, já que permite consultas em qualquer ordem. A árvore é abstrata porque não precisa representar detalhadamente todas as derivações. Tipicamente serão omitidas derivações intermediárias onde um símbolo não terminal gera somente um outro símbolo terminal, *tokens* que são palavras reservadas, e todos os símbolos "de sincronismo" ou identificação do código, os quais estão implícitos na estrutura

reconhecida. Os nodos da árvores serão de tipos relacionados aos símbolos não terminais, ou a nodos que representam operações diferentes, no caso das expressões.

Nodo da AST

O nodo da AST armazena primeiramente o seu tipo, que pode ser comando (de uma lista) ou assinalamento, ou soma (uma das operações de expressão) ou identificador (nodo folha que apenas aponta para a tabela de símbolos e representa variável ou constante em forma de nodo da AST. Por isso o nodo da AST deve ter um ponteiro para a tabela *hash*, além de ponteiros para os seus *n* filhos (aconselha-se 4, que é suficiente).

Especificações que devem ser feitas

Cada grupo precisa definir quais tipos de nodos existirão na AST, e podem haver diversos modelos um pouco distintos e igualmente válidos. Um elemento chave que irá ser usado na AST é a lista encadeada. Sempre que puder haver uma lista na linguagem, como lista de parâmetros, declarações ou comandos separados por ponto-e-vírgula, a árvore será desbalanceada (tipicamente para a direita), representando essa lista. Cada uma dessas listas podem ser feitas pelo encadeamento de um ponteiro de filho específico, e seus nodos podem já ser os parâmetros, declarações ou comandos encadeados, ou, como é mais aconselhável, pode haver um nodo específico para representar a lista, onde o primeiro filho é o parâmetro declaração ou comando, e o segundo filho é o próximo elemento da lista.

Geração de Código Fonte

O processo de geração de codigo fonte, ou descompilação, será percorrer a árvore gerada em ordem (ou na ordem apropriada a determinado nodo), à partir da raiz, e imprimir para cada nodo os símbolos que representam esse nodo, incluindo as chamadas recursivas às sub árvores, de forma que seja gerado um programa equivalente ao analisado. Ele não será idêntico, pois caracteres como espaços e estruturas como comentários, entre outros, foram ignorados. Mas a árvore deve ter as informações necessárias para gerar facilmente esse programa equivalente, e pode ter alguns nodos específicos para facilitar essa geração, como um nodo “bloco”, que gera a sequência de abre-chaves, lista de comandos e fecha-chaves.

Teste em laço

O executável gerado nessa etapa, chamado de etapa3, deve fazer um laco de compilação e descompilação, e para isso deve receber como parâmetros dois nomes de arquivos, o arquivo fonte a ser analisado e o arquivo fonte equivalente a ser gerado. Como o arquivo gerado nunca será idêntico ao original, para testes o laço será chamado duas vezes e seu resultado comparado, desta forma:

```
./etapa3 source.txt eq1.txt
./etapa3 eq1.txt eq2.txt
diff eq1.txt eq2.txt
```

Controle e organização do seu código fonte

Você deve seguir as mesmas regras das etapas anteriores para organizar o código, permitir compilação com **make**, permitir que o código seja rodado com **./etapa3**, e esteja disponível como **etapa3.tgz**.

Porto Alegre,16 de Abril de 2012