Universidade Federal de Goiás
Bacharelado em Ciências da Computação
Compiladores 2019 – 2

Compilador para a Linguagem Cafezinho
Especificação dos trabalhos:
T2 (Geração da Representação Intermediária e Análise Semântica)
e
T3 (Geração de Código)

prof. Thierson Couto Rosa

1 Objetivos

Esta etapa do projeto de construção de um compilador para a linguagem Cafezinho consiste na geração da representação intermediária do programa fonte, na utilização desta representação para a análise semântica e a geração do código na linguagem *assembly* do processador MIPS. As seções seguintes descrevem cada uma das fases desta etapa do trabalho.

2 Geração da Representação Intermediária

A representação intermediária corresponde a uma árvore sintática abstrata que é gerada durante a fase de análise sintática (ver Seção 2.8 e 5.3 do livro-texto[1]). Ações semânticas devem ser acrescentadas às produções da gramática no arquivo de entrada para o Bison ou Yacc, formando assim, o esquema de tradução necessário para gerar a árvore abstrata para programas de entrada escritos em Cafezinho. A árvore abstrata criada deve ser armazenada na memória principal do computador e servirá de entrada para as fases seguintes que correspondem à análise semântica e à geração de código em *assembly*. Os alunos devem se orientar pelo exemplo de construção de árvore sintática abstrata apresentado na Seção 2.8 do livro-texto.

[1] Alfred Aho et al. Compiladores - Princípios, Técnicas e Ferramentas. segunda edição, Pearson Addison Wesley, São Paulo, 2007.

É importante que a árvore abstrata seja a mais compacta possível, pois durante a fases seguintes será necessário fazer percursos na árvore para detectar erros semânticos e gerar o código em linguagem *assembly*. Os itens léxicos (identificadores, nomes de comandos, operadores, etc) devem ser armazenados na árvore juntamente com a linha onde eles ocorrem no programa fonte para que as mensagens de erro possam se referir à linha onde o erro foi detectado.

3 Analisador Semântico

O segundo objetivo desta etapa do trabalho corresponde à implementação do analisador semântico do compilador. Entretanto, para que o analisador semântico possa funcionar, é necessário a implementação prévia da tabela de símbolos para que os diversos identificadores e seus atributos que aparecem em um programa fonte possam ser armazenados e utilizados durante a análise semântica e a geração de código.

Aspectos Semânticos da Linguagem Cafezinho

Declaração de variáveis e parâmetros formais

As regras de escopo e de tempo de vida das variáveis, funções e parâmetros de funções da linguagem Cafezinho são semelhantes às regras da linguagem C, porém há as seguintes restrições:

- Não há o conceito de protótipo de função. Todas funções utilizadas no programa principal devem ser completamente declaradas (cabeçalho e corpo) antes da declaração programa que representa o bloco principal do programa.
- Todas as variáveis declaradas antes da declaração bloco principal (bloco identificado pela palavra-chave programa) são globais às funções declaradas após as declarações dessas variáveis e também são globais ao bloco principal.
- Não há o conceito de variáveis static como na linguagem C. O tempo de vida de uma variável local a um bloco corresponde à duração do bloco.
- Todo programa em Cafezinho ocupa apenas um arquivo e não há compilação separada. Portanto, não há declarações do tipo extern como na linguagem C.
- Assim como na linguagem C, uma variável global ao programa é uma variável declarada fora de qualquer função. Uma variável declarada em um bloco é local a esse bloco e global a todos os blocos internos a esse bloco, em qualquer profundidade de aninhamento de blocos. Entretanto, se uma variável local a um bloco tem o mesmo nome de uma variável global a esse bloco, a variável local se sobrepõe à variável global durante a duração do bloco.

- Os parâmetros formais têm escopo de variável local. Não é permitido uma variável local com mesmo nome de um parâmetro formal no bloco mais externo que forma a função. Entretanto, é permitido criar uma variável com mesmo nome de um parâmetro formal, em um bloco aninhado. Por exemplo:

```
int f(int a, int b){ int a, b; ....} -- não permitido.
int f(int a, int b) {....{int a,b; .....}  .... } -- permitido
```

- A passagem de um tipo vetor como parâmetro é feita passando-se o endereço de início do vetor (passagem por referência) e qualquer alteração feita em algum elemento do vetor pela função é mantida no vetor passado pela chamada à função quando essa termina sua execução.

- Ao contrário da linguagem C, na linguagem Cafezinho o número de parâmetros formais de uma função e o número de parâmetros de uma chamada à função deve ser o mesmo.

- A declaração de qualquer nome deve preceder o seu uso em um escopo válido, isto é, um nome pode ser utilizado em um comando somente se ele tiver sido declarado previamente e se o comando estiver no escopo de sua declaração.

Tipos e Checagem de Tipos

A linguagem Cafezinho possui apenas quatro tipos distintos: int, car, int [] e car []. A linguagem não permite operações entre objetos ou expressões de tipo simples (car ou int) distintos ou entre objetos e expressões de tipo simples e objetos do tipo vetor ([]). Contudo, são permitidas operações entre um elemento de um vetor e uma variável simples desde que sejam ambos do mesmo tipo básico (int ou car). O tipo vetor corresponde a um ponteiro para uma sequência de objetos de algum tipo básico. A atribuição de uma variável vetor X a outra variável vetor Y de mesmo tipo básico, corresponde à cópia do endereço da variável X para a variável Y. Essa atribuição é permitida somente se os vetores forem do mesmo tipo básico.

O tipo de cada parâmetro formal deve ser o mesmo de cada parâmetro de chamada correspondente. No caso em que o parâmetro de chamada corresponde a um elemento de um vetor, o parâmetro formal correspondente deve ser uma variável simples de mesmo tipo básico do vetor. Se a chamada passar um vetor como parâmetro, o parâmetro formal correspondente da função chamada também deve ser um vetor, sem o tamanho especificado (apenas o par de colchetes sem um valor entre eles). Além disso, o tipos básicos dos vetores do parâmetro formal e do parâmetro de chamada correspondente devem ser os mesmos.

Comandos e Expressões

Não existe o tipo lógico explícito na linguagem Cafezinho, entretanto nos comandos se e enquanto é necessário avaliar se uma expressão é verdadeira ou falsa para a tomada de uma

decisão quanto ao fluxo de execução dos internos a estes dois comandos. Assim como a linguagem C, a linguagem Cafezinho trata como verdadeira uma expressão que cujo valor é diferente de zero e como falsa, uma expressão cujo valor é zero. Obviamente, essa avaliação é possível somente em tempo de execução do programa fonte. Entretanto, a nível de análise semântica, uma expressão lógica (com operações OR, AND ou NEGAÇÃO) ou relacional (com operadores relacionais – >, <,<=, etc) devem receber tipo int.

Uma expressão retornada por uma função deve ter o mesmo tipo da função. As expressão do lado direito de uma atribuição deve ter o mesmo tipo da variável que recebe a expressão. Os operadores aritméticos devem ser aplicados em expressões do tipo int. Os operadores relacionais devem ser aplicados a operandos de mesmo tipo.

Tabela de Símbolos

A tabela de símbolos a ser implementada deve ter funções (métodos) que permitam as seguintes operações:

- Criação de um novo escopo.
- Inserção de um identificador na tabela de símbolos após a criação de um novo escopo.
- Pesquisar o lexema de um identificador na tabela de símbolos. Caso o lexema seja encontrado na tabela de símbolos, o endereço do identificador apropriado deve ser retornado para que os demais módulos do compilador possam atualizar os atributos do identificador.
- Remover da tabela de símbolos um escopo e todos os identificadores a ele associados.

O analisador semântico deve percorrer a árvore abstrata gerada para uma cadeia de entrada e checar, gradualmente, a correção semântica de cada construção presente na árvore, seguindo as regras semânticas apresentadas nesta seção. Qualquer ocorrência de uma construção no programa fonte que não é permitida na linguagem Cafezinho deve ser considerada um erro semântico. Nesse caso, o analisador semântico deve reportar o erro semântico encontrado, o número da linha onde ele ocorreu e terminar a compilação.

Geração de Código

Uma vez que o analisador semântico detecta que o programa de entrada (representado internamente pela árvore abstrata) não possui erros semânticos, ele deve ativar o gerador de código. O módulo gerador de código dever percorrer novamente a árvore abstrata e gerar instruções em linguagem assembly, à medida em que caminha na árvore abstrata.

O código será gerado na linguagem assembly do processador MIPS para que possa ser executado no SPIM que é um simulador arquitetura MIPS. Uma descrição da linguagem assembly a ser utilizada pode ser encontrada no Capítulo 3 e no Apêndice A do livro Organização e projeto de computadores, Dad A. Patterson e John L. Hennessy.

4 Informações Sobre Implementação e Entrega

Representação intermediária e análise semântica

O esquema de tradução para gerar a árvore abstrata, implementado no arquivo de entrada para o Bison/Yacc, e o analisador semântico devem ser entregues até o dia 15/11/2019. Especificamente deve ser entregue o seguinte:

1. O arquivo de especificações para o gerador de analisador léxico utilizado.
2. O código que implementa a tabela de símbolos.
3. O arquivo com comandos para o gerador de analisador sintático expandido com ações semânticas necessárias para a geração de árvore sintática abstrata.
4. O código do analisador semântico que irá utilizar a árvore abstrata e a tabela de símbolos para detectar a ocorrência de erros semânticos em arquivos contendo programas escritos na linguagem Cafezinho.
5. O código que irá utilizar a árvore abstrata e a tabela de símbolos para gerar o código objeto em linguagem *assembly*.
6. Um arquivo Makefile com comandos para:
    - Gerar o código do analisador léxico a partir do arquivo contendo as especificações para o gerador de analisador léxico.
    - Gerar o código do analisador sintático e construtor da árvore abstrata a partir do arquivo contendo as especificações para o gerador de analisador sintático.
    - Compilar os arquivos gerados e os escritos na linguagem de programação adotada no projeto (C, C++, Java ou Python)
    - Gerar o executável (O professor deve ser capaz de obter o executável do trabalho digitando apenas o comando *make* em uma *shell*).

Os itens acima devem estar agrupados em um arquivo do tipo *tar* compactado com o utilitário *gzip*. O referido arquivo deve ser submetido como uma tarefa a ser criada no ambiente Moodle da disciplina. Os códigos gerados devem estar preparados para executarem no sistema operacional Linux (ambiente onde os trabalhos serão avaliados).

Geração de Código

A segunda fase do trabalho corresponde à geração de código. A data limite de entrega desta fase é 09/12/2019. Devem ser entregues os mesmos itens requeridos na fase anterior e, em adição, o módulo do compilador responsável por gerar o código objeto. Deve ser entregue também um arquivo Makefle capaz de compilar todas os módulos produzidos

e gerar um programa executável. Os arquivos devem ser agrupados em um arquivo do tipo *tar*, compactado com o utilitário *gzip* e deve ser submetido como uma tarefa no sistema Moodle. O trabalho de completo deve ser apresentado em uma das aulas dos dias 09/12/2019 a 19/12/2019.

IMPORTANTE

Cópia de partes de código implicam em nota ZERO na NOTA FINAL do trabalho para todos os alunos envolvidos (os que copiarem e os que cederem o trabalho para cópia).