4º Trabalho de Introdução à Compilação

–

SCC-206

Grupo 4:

Arthur F. M. Nascimento	5634455
Paulo Cesar Antonio Jr.	5634890
Paulo R. C. Diniz	5634816
Felipe V. Perez	5634692

Sobre o Trabalho

O objetivo deste trabalho é implementar a geração de código de um compilador de Pascal Simplificado para MEPA (Máquina de Execução de Pascal) usando a ferramenta JavaCC. Para isso usamos o compilador feito no terceiro trabalho e adicionamos os comandos de geração de código (chamados *GERA* e *Proximo_Rotulo*).

O PS é uma versão simplificada do Pascal padrão mas contém também algumas extensões que a aproximam da linguagem fonte original. Estas extensões são: declaração de tipos através do comando *type*; variáveis enumeradas (que são declaradas através de uma lista de identificadores); e o comando condicional *case*.

A MEPA é uma máquina hipotética desenvolvida para fins educacionais que executa bem programas escritos em Pascal por ter sido desenvolvida com essa finalidade.

O JavaCC é um *Compiler Compiler* que gera o código fonte de um compilador a partir de descrições de alto nível dos tokens (para o Analisador Léxico), das produções da gramática (para o Analisador Sintático) e permite que se insiram trechos de código que constituem o Analisador Semântico.

Detalhes de Implementação

Usamos os métodos GERA e Proximo_Rotulo apresentados em aula para este trabalho. O primeiro cria uma instrução da MEPA usando os argumentos passados. O segundo retorna um novo rótulo único para ser usado em desvios. Os rótulos usados para procedimentos e para o início do programa principal, por simplicidade, são os nomes dos respectivos procedimentos e do programa principal.

Para cada bloco processado, o compilador deve gerar código que entre e saia do bloco corretamente. Se o bloco for o programa principal, então é necessário o rótulo, assim como AMEM e DMEM se houverem variáveis no escopo. Se o bloco for um procedimento, então além das instruções já citadas, é necessário gerar ENPR e RTPR.

Encontramos um problema na geração de código relacionado ao processamento de expressões e chamadas do procedimento especial *read*. Isso pois a gramática estabelece que uma chamada a procedimento é um identificador seguido possivelmente por uma lista de expressões entre parênteses. De acordo com a nossa implementação, ao processar uma expressão, o resultado final é deixado no topo da pilha para ser usado posteriormente. Contudo, não queremos o resultado das expressões para o procedimento *read* – apenas as variáveis a serem lidas. A solução mais elegante seria reescrever a gramática para permitir que *read* fosse seguido por uma lista de variáveis, e não de expressões. Mas como reconhecemos que alterar a gramática neste momento não é adequado, então passamos um parâmetro para o processamento das expressões que suprime o carregamento do valor da variável para a pilha.

O procedimento write é processado quase da mesma forma que os outros procedimentos, excetuando que não é feita a checagem de tipo para as expressões passadas e para cada parâmetro informado, uma instrução IMPR é gerada.

A chamada de procedimentos definidos pelo usuário são tratados da seguinte forma: ao se processar a lista de parâmetros, os resultados das expressões são mantidos no topo da pilha e uma lista com os tipos

estimados de cada expressão é retornada; esta lista é comparada com a lista de parâmetros do procedimento chamado e então a chamada é gerada se os tipos forem compatíveis de acordo com a checagem de tipos por nome.

As expressões são lidas no formato *em-ordem* mas as instruções geradas devem estar no formato *pós-ordem*. Para isso, a operação entre cada operando é guardada e a sua instrução é gerada apenas após o processamento do segundo operando.

Instruções

Para compilar, execute:

```
javacc pascal.jj
javac *.java
```

Para executar o programa que explora as extensões implementadas:

```
java Pascal <entrada.pas
```

Para executar os casos de teste do livro do Kowaltowski:

```
java Pascal <kow-8.4.pas
```

(analogamente para os exercícios 8.5, 8.9, 8.11, 8.13, 8.15, 8.21)