Universidade de São Paulo
Instituto de Matemática e Estatística

MAC0499 – Trabalho de Formatura Supervisionado

Um Compilador Em Java Para Fins Didáticos

Iniciação Científica

Aluno: Daniel Sguillaro
Orientador: Prof. Dr. Alan M. Durham

I – Parte objetiva

Introdução

Cursos de graduação em Ciência da Computação normalmente possuem em seu currículo uma disciplina que trata dos diferentes paradigmas de programação. Ao cursá-la, o aluno costuma ter dificuldades no aprendizado de tais paradigmas pois geralmente só teve contato com uma forma de programação até então.

Como os paradigmas são ensinados através das linguagens que os representam, surge uma dificuldade adicional: o aluno precisa se familiarizar com diferentes sintaxes, nomes de funções, uso de compiladores, etc.

Esta monografia apresenta a proposta de implementação de um compilador para o sistema de interpretação de linguagens utilizado no curso MAC0316 (Conceitos Fundamentais de Linguagens de Programação) do IME-USP. Esse sistema tem fins didáticos e visa resolver o problema descrito no parágrafo anterior. Foi desenvolvido em Java pelo Prof. Alan Durham, tendo sido baseado em sistema de funcionalidade semelhante originalmente desenvolvido por Samuel Kamin.

Para resolver o problema descrito acima, implementou-se uma série de interpretadores, um para paradigma de programação a ser estudado. Os interpretadores possuem a mesma sintaxe básica (da linguagem LISP) e podem ser facilmente extendidos para acomodar os diferentes conceitos presentes em cada paradigma diferente de programação. Por causa dessa facilidade em extender o sistema, consideramos que ele seja um arcabouço para construção de tradutores. Daqui em diante, chamaremos o sistema desenvolvido de ***arcabouço Java,*** ou simplesmente ***arcabouço***.

Um dos principais pontos fortes do arcabouço é a possibilidade do usuário poder estudar e alterar/extender a implementação da linguagens apresentadas. Além disso, sua arquitetura deixa bem claro as semelhanças e diferenças entre cada um dos interpretadores desenvolvidos.

Os paradigmas implementados são para linguagens funcionais (incluindo avaliação sob demanda), orientadas a objetos, e lógicas. Existe também um interpretador básico para que o usuário possa primeiro se familiarizar com a sintaxe, com a principal estrutura de dados disponível (as listas) e com o uso de recursão como principal método de controle de fluxo.

Inicialmente, iremos apresentar os conceitos básicos de compilação. Em seguida introduziremos o conceito de interpretação, e mostraremos sua diferença em relação à compilação. Feito isso, podemos mostrar a estrutura do arcabouço Java

e explicar o que deve ser modificado nele para adicionar as funcionalidades propostas.

O processo de compilação

Um ***compilador*** pode ser definido como um programa que lê outro programa escrito em uma certa linguagem (chamado ***programa-fonte***) e o traduz para outro programa equivalente, porém escrito em outra linguagem (chamado ***programa-objeto***). Independentemente dessas duas linguagens, os conceitos básicos de compilação são essencialmente os mesmos, e iremos expor esses conceitos aqui.

Compilar um programa não é uma tarefa fácil. Dividir o processo de compilação em partes é essencial. Esse processo pode ser dividido nas seguintes partes:

- ***análise:*** decompõe o programa-fonte em suas partes básicas. Essa fase pode por sua vez ser subdividida nas seguintes partes:

    - ***análise léxica:*** o analisador léxico recebe como entrada o próprio programa-fonte, que nada mais é que um stream de caracteres e lê o programa linearmente. O analisador léxico deve identificar os elementos mais básicos do programa, como nomes de variáveis, constantes, palavras-chave, constantes, operadores, etc, que constituem “classes” da linguagem fonte. Deve também remover

conteúdo irrelevante para o processo de compilação, como por exemplo os espaços em brancos e os comentários. O resultado deste processo é um stream de ***tokens***, onde cada um dos tokens é um representante de uma das classes descritas acima.

- ***análise sintática:*** o analisador sintático recebe como entrada o stream de tokens gerado pelo analisador léxico. Sua função é agrupar os tokens de uma forma hierárquica, determinando a estrutura do programa-fonte. O resultado deste processo é uma árvore sintática, representando a estrutura do programa.

- ***análise semântica:*** o analisador semântico recebe como entrada a árvore sintática gerada pelo analisador sintático. Sua função é verificar o significado do programa-fonte, por exemplo verificando se um operador recebeu o número correto de operando. O resultado deste processo é um código em algum formato intermediário, como por exemplo o formato de quádruplas, descrito a seguir.

- ***síntese:*** usa a decomposição feita no processo de análise para gerar o programa-objeto. Pode ser subdividida nas seguintes partes

- ***geração de código:*** recebe como entrada o código em formato intermediário e gera (geralmente) código assembler ou código de máquina.

- ***otimização de código:*** recebe como entrada o código gerado e faz otimizações nesse código com o intuito de tornar o programa-objeto mais eficiente.

Compilação vs. Interpretação

A compilação não é a única forma de tradução de código. Existem também uma forma chamada ***interpretação***. No processo de interpretação, o tradutor processa o código fonte juntamente com os dados fornecidos pelo usuário. Assim, ele atua de maneira interativa, executando o programa na medida que os comandos e/ou expressões são lidos (fig. 1).

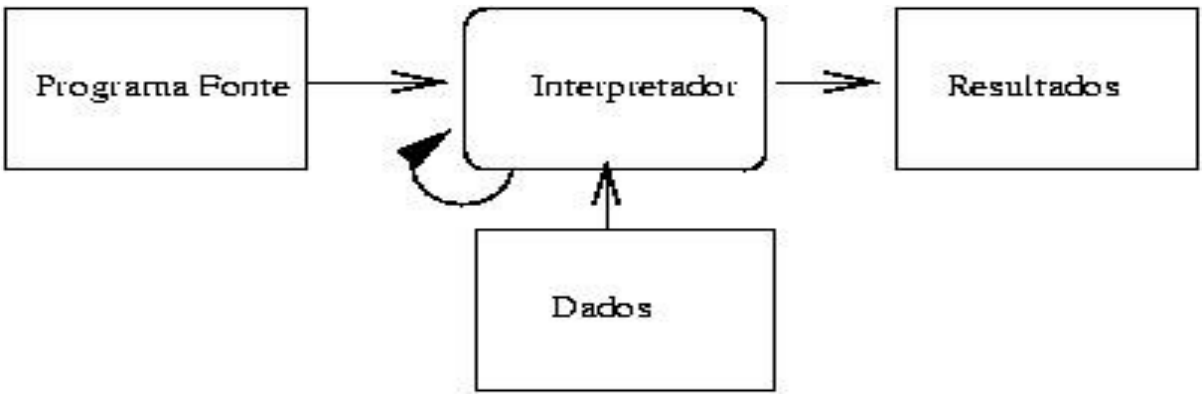

Figura 1 – Processo de interpretação

Já na compilação. o código é traduzido em um primeiro momento (chamado ***tempo de compilação***), gerando o programa-objeto. Em um segundo momento (chamado ***tempo de execução***) o programa-objeto é executado diretamente pelo processador da máquina (fig. 2).

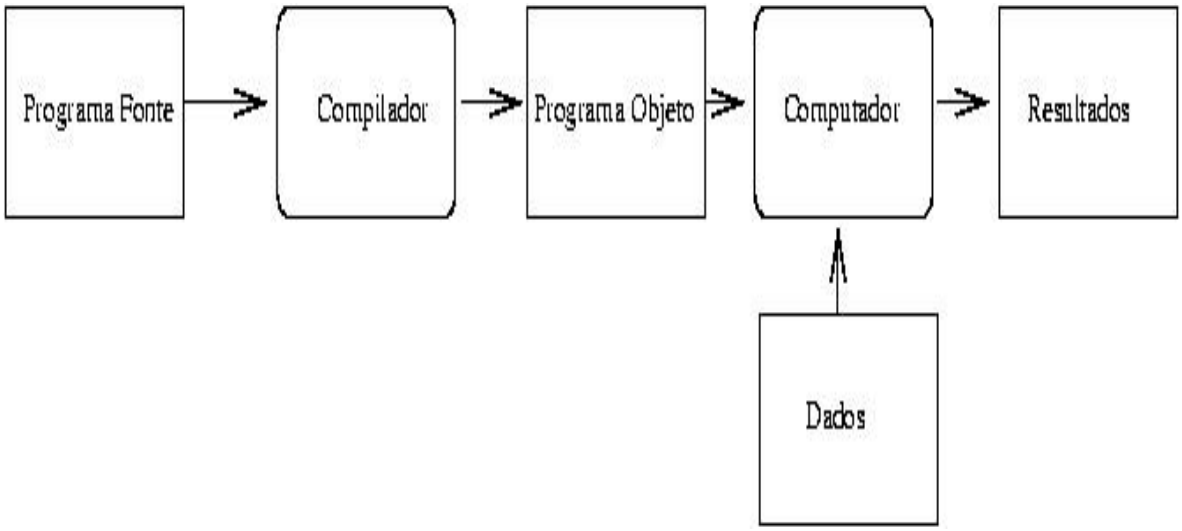

Figura 2 – Processo de compilação

Ambas as formas são de tradução possuem suas vantagens e desvantagens, e são amplamente utilizadas na prática.

O arcabouço Java

O arcabouço Java traduz código usando a interpretação. Sua implementação foi feita da seguinte forma:

- As classes *TokenStream* e *Token* (e suas subclasses) implementam o módulo de análise léxica. A classe *ExpressionParser* implementa o módulo de análise sintática

- A classe *Environment* é responsável por armazenar o estado de execução do interpretador. Essa classe contém os valores associados às variáveis, escopo de nomes, etc. São implementados usando estruturas que contém elementos do tipo par chave-valor, onde a chave é o nome do identificador (tipo *String*) e o valor é um *Object* que pode ser qualquer valor associado àquele nome em um dado momento da execução.

- A classe *Expression* e suas subclasses são implementada usando o padrão *Interpreter*, onde cada classe implementa a semântica de um elemento da árvore abstrata gerada pelo analisador sintático. A hierarquia de classes com raiz em *Expression* é mostrada na fig. 3. Essas subclasses representam números (*NumberExp*), cadeias de caracteres (*SymbolExp*), listas (*CellExp*), operações primitivas (*OperationClosure*), funções de usuário (*Closure*), chamadas de função (*ApplicationExp*), nomes de identificadores (*VariableExp*), etc.

- O interpretador avalia a árvore abstrata gerada chamando o método Expression.evaluate() do elemento na raiz da árvore abstrata. Esse método

retorna para o interpretador um *Expression* que é impresso usando o seu método toString().

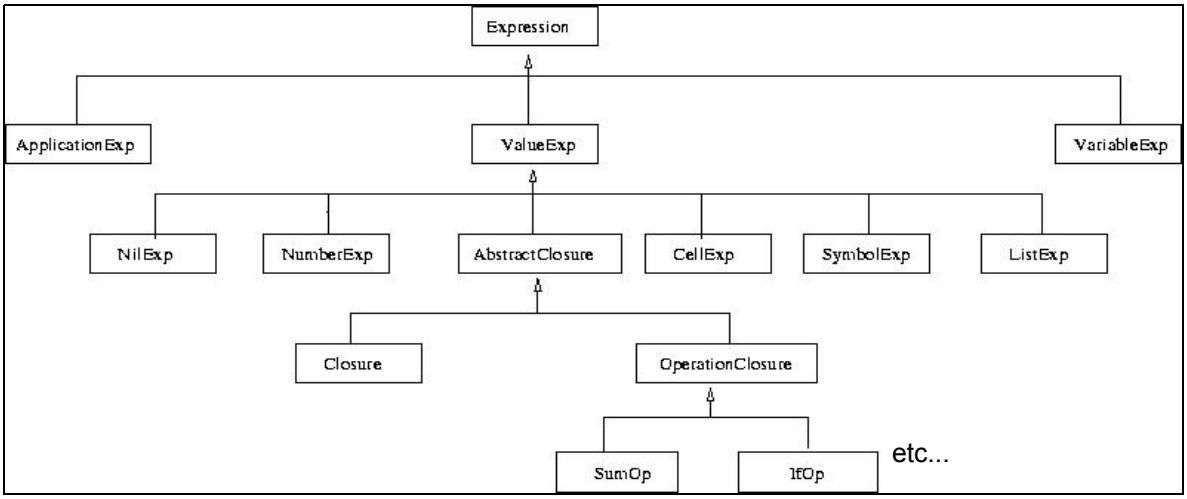

Figura 3 – Hierarquia de classes com raiz em Expression

O compilador

Como a compilação é uma forma de tradução de código muito usada, seria interessante que o usuário do arcabouço também tivesse contato com essa forma de tradução. Por isso, estamos extendendo-o de forma que também possa compilar o código-fonte.

Um dos principais objetivos da elaboração inicial do arcabouço em Java foi mostrar de forma clara quais as semelhanças e diferenças na implementação de cada uma das linguagens apresentadas. Seguindo essa mesma filosofia, o

compilador está sendo implementado de forma que sejam aproveitados vários módulos do arcabouço. Isso irá mostrar ao aluno quais as semelhanças e diferenças entre a compilação e a interpretação. Assim, todo o “front-end”, do qual fazem parte os módulos de análise léxica, sintática e semântica pôde ser usado sem modificações.

A compilação se dá através da adição do método compile() às classes da hierarquia de *Expression.*

O código objeto que estamos gerando não é um código de máquina específico, nem linguagem de montagem (“assembly language”). O formato escolhido foi o de quádruplas. As instruções tem o seguinte formato: <operador>, <operando1>, <operando2>, <resultado>. Esse formato foi escolhido por dois motivos.

- Portabilidade do código gerado: podemos traduzir facilmente quádruplas para qualquer linguagem assembly, já que elas são quase uma forma neutra de assembly. Alternativamente, podemos implementar máquinas virtuais que interpretam as quádruplas. A vantagem em usarmos máquinas virtuais é que caso se queira mover o compilador para outra máquina, é muito mais fácil reescrever a máquina virtual do que o compilador inteiro. Esta é a abordagem que escolhemos para executar o código.

- Otimização de código: o formato de quádruplas é ideal para realizarmos otimizações

Tabelas de símbolos

As ***tabelas de símbolos (TS)*** são a estrutura de dados do compilador que substituem os ambientes do interpretador. Sua função é auxiliar o compilador na geração de código objeto.

Na verdade, essas tabelas são bem semelhantes aos ambientes. Assim como os ambientes, elas também contém elementos do tipo par chave-valor; as operações também são semelhantes - inserção, alteração, busca e remoção; e finalmente, elas também possuem uma estrutura hierárquica, ou seja, tabelas podem conter sub-tabelas.

O que as difere dos ambientes é conteúdo que armazenam. Enquanto nos interpretadores os ambientes armazenam o estado da execução do programa, ou seja, o valor associado a cada identificador em cada momento, nos compiladores isso não faz sentido já que o programa não será executado e sim apenas traduzido. Assim, as TS irão armazenar uma série de atributos associados às variáveis usadas no programa.

No caso de variáveis globais, a TS irá armazenar o endereço da variável, ou seja, o local da memória onde seu valor será ramazenado. No caso de palavras reservadas para as operações primitivas será armazenado o objeto que

implementa essa primitiva já que este sabe se compilar através do método compile(). Para variáveis locais a funções, armazenamos a sua posição no registro de ativação, pois o espaço destinado a essa variável será alocado somente em tempo de execução, não tendo posição fixa de memória.

Como foi dito anteriormente, as tabelas de símbolo contém objetos do tipo par chave-valor. A chave é um objeto do tipo String, representando o nome do identificador. O valor é um objeto do tipo SymbolTableEntry que contém os atributos descritos acima.

Máquina Virtual

O modelo da máquina virtual ainda está sendo estudado pelo autor, e ainda não foi implementado.

A divisão da memória provavelmente será feita da seguinte forma: existirão duas áreas distintas, uma para o código que armazena as instruções do programa, e outra para os dados do programa. A área de dados por sua vez, é dividida em três partes: as áreas de dados estáticos, pilha e heap.

Haverão três registradores, dois acumuladores genéricos (acc1 e acc2) e um registrador para gerenciar a pilha (topo_pilha).

Finalmente, a linguagem da máquina virtual será a linguagem de quádruplas descrita anteriormente.

Depuração

Programas executados em qualquer um dos interpretadores do arcabouço podem ser depurados com qualquer depurador Java. Isto não acontece com os programas compilados, pois estes são executados em uma máquina virtual criada pelo autor. Por isso, será criado um depurador para os programas compilados. Infelizmente, até o momento o depurador não foi implementado.

Otimização de código

Idealmente um compilador deveria gerar código tão bom quanto código escrito à mão. Porém, isso é quase impossível de ser alcançado. Por isso, precisamos lançar mão de técnicas de otimização de código. O arcabouço terá um otimizador de código compilado mas, novamente, infelizmente este ainda não foi implementado.

Conclusões

O arcabouço java é uma ecelente ferramenta didática para o aprendizado dos diferentes paradigmas de programação pelos alunos. Conhecer e alterar sua implementação o torna ainda mais interessante, e este projeto está sendo especificamente importante nesse ponto. O usuário pode conhecer a implementação de um compilador e ver as diferenças e semelhanças entre a compilação e a interpretação. Além disso, pode conhecer as técnicas para otimizar o código gerado.

Referências bibliográficas

[1] DURHAM, A. M. ; CONCEICAO, A. F. ; SUSSUMU, E. ; LIMA, A. M. ; TEGLIA, J. ; SANTOS, M. B. - A Framework for Building Language Interpreters. In: Educator's Symposium - OOPSLA' 03, 2003, Anaheim. OOPSLA'03 Companion, 2003. p. 191-196.

[2] KAMIN, SAMUEL N. - Programming Languages: An Interpreter Based Approach. Addison-Wesley, 1990.

[3] AHO, A. V. ; SETHI, R. ; ULLMAN, J. D. - Compilers: Principles, Techniques, and Tools. Addison-Wesley, 1986.

[4] TREMBLAY, J. P. ; SORENSON P. G. - The Theory And Practice Of Compiler Writing. McGraw-Hill, 1985

[5] GAMMA, E. ; HELM, R. ; JOHNSON, R. ; VLISSIDES, J. - Design Patterns: Elements of Reusable Software. Addison Wesley, 1995.

II - Parte subjetiva

Escolha do Tema

Ao ingressar em um estágio no início de 2005 imaginava fazer o trabalho de formatura baseado nessa experiência. Porém, conforme o tempo ia passando vi que a área de desenvolvimento era um pouco tediosa por não oferecer muitos desafios, ou pelo menos desafios que não me causavam muito interesse. Comecei a considerar a opção de seguir carreira acadêmica, e para ter certeza que essa esta a opção correta decidi fazer uma iniciação científica (IC).

Nos últimos anos tive a curiosidade de estudar compiladores, principalmente por sua complexidade e por o tema ter um ótimo balanço entre teoria e prática. Esse

desejo não foi saciado pois a disciplina não é oferecida há alguns anos, já que é optativa.

Assim, a escolha desse tema para a IC foi algo natural. Procurei o Prof. Alan Durham para ser meu orientador por saber do seu interesse na área e por ser seu aluno na disciplina MAC0316 (Conceitos Fundamentais de Linguagens de Programação) na época.

Desafios e frustrações

Um dos principais desafios do projeto foi ter que aprender um tema tão complexo quanto compiladores. Consequentemente, implementar o compilador também está sendo uma tarefa complexa e certamente é um desafio.

Outro desafio foi ter que extender o arcabouço (ao invés de fazer uma implementação completa) pois a arquitetura do arcabouço existente é excelente, o que fez com que eu quisesse manter essa qualidade. Assim, por muitas vezes me vi gastando horas e horas analisando se a arquitetura do compilador estava em um nível tão bom quanto o restante do arcabouço. Em uma situação dessas, você não segue tanto os instintos, está sempre analisando cada pequena coisa que faz, e isso faz a sua produtividade diminuir. Mas certamente a qualidade final é melhor.

Como principais frustrações, posso citar o fato de não ter conseguido dedicar o tempo que gostaria ao projeto no segundo semestre. Como consequência, não consegui terminar a implementação no tempo proposto. Mas tenho certeza que terminarei o projeto até a data da recuperação.

Interação com o orientador

A leitura dos textos sugeridos por meu orientador foram essencias para que o projeto tomasse forma, pois a princípio o projeto seria fazer uma refatoração total no interpretador Prolog. A leitura e a discussão desses textos com ele foi o que nos fez mudar de idéia e trocar o tema para o que estamos fazendo hoje. Além disso, os textos lidos foram e estão sendo de enorme ajuda para minha implementação.

Durante o primeiro semestre tivemos reuniões semanais sempre em um horário fixo, pois eu tinha pouco tempo disponível já que trabalhava em tempo integral. Essas reuniões eram praticamente o único momento disponível para discutirmos os temas estudados, e eu precisava aproveitar essas reuniões da melhor forma possível. Com a minha saída do estágio no meio do ano as reuniões passaram a ter caráter mais informal, já que o meu tempo na faculdade já não era mais escasso. O fato de poder passar em sua sala quando estava emperrado em alguma parte da implementação, ou mesmo para tirar pequenas dúvidas está sendo de grande ajuda.

Meu orientador está sendo um grande incentivador para que o projeto termine. O fato de ele desejar usar meu trabalho na disciplina MAC0316 também é um grande incentivo, e até um motivo de orgulho para mim. Colocando um pouco o projeto de lado, gostaria de mencionar que também estou recebendo seu incentivo para ingressar no mestrado (a princípio na área de bioinformática), algo que estou considerando seriamente.

BCC vs. IC

Conciliar o BCC com a IC não é uma tarefa fácil. As disciplinas que cursamos exigem uma grande dedicação, principalmente no que diz respeitos aos exercícios-programa (EPs). Fazer IC exige uma dedicação tão grande quanto as disciplinas. Mesmo assim não tenho reclamações, pois é uma experiência gratificante.

A forma de trabalho na IC é bem diferente em relação às matérias do curso. No BCC temos que nos dedicar a várias disciplinas ao mesmo tempo enquanto na IC nos dedicamos a um único tema. Outra diferença é que no BCC temos o professor explicando toda a matéria e passando o conteúdo na lousa, enquanto na IC temos que ler uma boa quantidade de material por conta própria. Porém nas disciplinas o contato para tirar dúvidas com o professor geralmente é pequeno, enquanto na IC o contato com o orientador é bem maior.

O grau de exigência na IC também é bem maior. No curso podemos entregar um EP com qualidade duvidosa e mesmo assim tirar uma nota alta. Na IC a qualidade dos trabalhos deve ser sempre excelente.

Disciplinas do BCC mais relevantes para o projeto

MAC0323 - Estrutura de Dados: o aprendizado de estruturas como árvores e tabelas de símbolos foi fundamental para o entendimento do arcabouço existente e sua ampliação. Fica apenas uma crítica: cursei a disciplina duas vezes pois na primeira vez não tive tempo de fazer todos os exercícios-programa, e posso afirmar que o conteúdo da segunda vez foi um pequeno subconjunto do conteúdo do ano anterior. Na minha opinião os alunos que cursaram a disciplina nesse ano tiveram um grande buraco em sua formação.

MAC0211 – Laboratório de Programação I: O contato com uma linguagem de baixo nível como o assembler me ajudou muito no projeto, já que o nosso compilador traduz o código para uma linguagem muito parecida com esta.

MAC0242 – Laboratório de Programação II: Nesta disciplina, assim como em MAC0211, há um grande projeto em grupo em várias fases. No ano que cursei o projeto consistia de três módulos: compilador, máquina virtual e ambiente gráfico.

A experiência que tive desenvolvendo os dois primeiros módulos mencionados foi de grande ajuda no projeto, apesar de ter sido algo bem mais simples.
Também tive contato com analisadores léxicos e sintáticos, que apesar de não serem usados no nosso projeto, são de grande ajuda no desenvolvimento de compiladores em geral.

MAC0316 – Conceitos Fundamentais de Linguagens de Programação: Tive contato com o arcabouço e com as linguagens que devem ser compiladas. Além disso é uma disciplina essencial por mostrar ao aluno as diferentes formas de programar.

MAC0441 – Programação Orientada a Objetos: O arcabouço faz grande uso das boas práticas de POO, como o uso de padrões de projeto. Tive contato com várias dessas técnicas nesta disciplina.

MAC0414 – Linguagens Formais e Autômatos: Expressões regulares, linguagens regulares e linguagens livres de contexto são fortemente usadas na construção de compiladores.