

Pelotas, 4 de abril de 2008.

Il.^{mo} Sr.

Prof. Dr. Carlos Antônio Pereira Campani

M. D. Coordenador do Curso de Bacharelado em Ciência da Computação / UFPel

Sr. Coordenador:

Encaminhamos em anexo a Proposta para Projeto de Conclusão de Curso do aluno **ADRIANO BONAT ALVES**, para cumprimento do requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação.

O título provisório do Projeto de Conclusão do Curso é:

COMPILAÇÃO DE CÓDIGO FUNCIONAL PARA *BYTECODES* JAVA

Atenciosamente,

Adriano Bonat Alves

Prof. Cristiano Damiani Vasconcellos, Dr.
Orientador

ADRIANO BONAT ALVES

COMPILAÇÃO DE CÓDIGO FUNCIONAL PARA *BYTECODES* JAVA

Proposta para Projeto de Conclusão apresentada ao Curso de Bacharelado em Ciência da Computação, Instituto de Física e Matemática, Universidade Federal de Pelotas.

Orientador: Prof. Cristiano Damiani Vasconcellos, Dr.

PELOTAS, 2008

1. RESUMO DA PROPOSTA

Considerando a diversidade de linguagens de programação existentes e que existem linguagens que apresentam características que a tornam mais adequadas para a resolução de um determinado tipo de problema que outras, uma boa interface para a chamada de funções externas é uma característica importante na escolha de uma linguagem. Entretanto essa interoperabilidade entre linguagens nem sempre é algo simples, muitas vezes tipos de dados não podem ser mapeados trivialmente entre diferentes linguagens. Este trabalho propõe um estudo sobre as questões relacionadas à interoperabilidade entre linguagens de diferentes paradigmas e a implementação de uma linguagem funcional baseada na sintaxe de Haskell que possa ser executada na máquina virtual Java, que fornece um ambiente multi-plataforma, portanto altamente portátil, e com checagem de tipos segura.

2. INTRODUÇÃO

A grande maioria das linguagens de programação fornece algum suporte a chamadas externas, possibilitando a interoperabilidade entre linguagens, mas geralmente essas chamadas envolvem algumas limitações e uma dificuldade considerável nas conversões de tipos, uma solução mais flexível é o uso de componentes independentes de arquitetura para implementar essa interface, por exemplo o *Component Object Model* (COM) [COM, 2008], da Microsoft.

O *Common Language Runtime* (CLR) [CLR, 2008], também da Microsoft, dá um passo adiante na questão de interoperabilidade, fornecendo uma linguagem intermediária e um sistema de tipos neutro capaz de suportar múltiplas linguagens de programação. O *Common Language Runtime* é um ambiente de execução para uma linguagem intermediária independente de arquitetura (*Common Intermediate Language* ou *IL*), essa linguagem intermediária possui um sistema de tipos (*Common Type System*) que garante a não violação dos tipos no código gerado por qualquer linguagem para este ambiente. Embora a premissa no projeto de *IL* seja uma plataforma multi-linguagens o paradigma orientado a objeto foi o que teve maior influência e importância em sua concepção [HAMILTON, 2003]. Por essa razão, algumas características encontradas em linguagens funcionais, particularmente no sistema de tipos, muitas vezes apresentam dificuldades na tradução para *IL*, como contornar essas limitações têm sido objeto de várias pesquisas recentemente.

Código intermediário independente de arquitetura e máquinas virtuais foram popularizados pelo projeto da linguagem Java, esse projeto teve como premissa a idéia de permitir que programas possam ser executados em diversas plataformas, seguindo a sua filosofia "*Write once, run everywhere*", além de fornecer um sistema de tipos seguro. Para alcançar este objetivo, programas Java são compilados para uma linguagem intermediária (*bytecode*) que roda em uma máquina virtual chamada de *Java Virtual Machine* (JVM).

O sucesso da plataforma Java ocorreu principalmente por sua proposta de executar um programa em diversas plataformas, sem modificações e de maneira

segura, pois a linguagem não suporta ponteiros para áreas de memória explicitamente. Com isso primeiramente conquistou o ambiente Web, depois a área de aplicativos desktop e então dispositivos embarcados.

A linguagem de *bytecodes* da JVM é padronizada [LINDHOLM, 2008], possibilitando que outros compiladores gerem código para executar na JVM, porém esta linguagem foi pensada exclusivamente para a linguagem de programação Java, tornando difícil a implementação de suporte para outras linguagens, inclusive de paradigmas diferentes, que rodem sobre a JVM.

3. OBJETIVOS

Esse projeto visa estudar questões relacionadas à interoperabilidade entre linguagens de diferentes paradigmas e as dificuldades encontradas em gerar código funcional para uma linguagem intermediária projetada para o suporte a linguagens orientadas a objetos. Para isso será implementado um compilador para uma pequena linguagem funcional que terá como código objeto *bytecodes* Java.

4. CONTRIBUIÇÃO ESPERADA

A expectativa, com este trabalho, além do estudo das questões sobre interoperabilidade entre diferentes paradigmas e das limitações da JVM como um ambiente multi-linguagens, é que seja implementado um *front-end* analisador sintático e inferidor de tipos para o núcleo de uma linguagem funcional baseada na sintaxe de Haskell, excluindo o tratamento a sobrecarga. Também se espera a implementação do *back-end* para esta linguagem que tenha como código objeto *bytecodes* Java, possibilitando esta linguagem acessar funcionalidades de programas que rodem na JVM, assim como outras linguagens acessarem as funcionalidades expostas pela linguagem implementada.

5. METODOLOGIA E PLANO DE TRABALHO

Inicialmente, será feito um estudo da especificação da máquina virtual Java. Posteriormente será feita uma revisão bibliográfica sobre interoperabilidade entre linguagens e da correspondência de tipos entre linguagens funcionais e o sistema de tipos da JVM [BENTON, 1999].

Para a implementação do front-end do compilador deverá ser feito um estudo das ferramentas de geração de compiladores, como provavelmente a implementação será em linguagem Haskell, as opções a serem analisadas são o *Happy* [HAPPY, 2008], que é um gerador similar ao *yacc*, que tem como entrada uma gramática livre de contexto e gera um módulo Haskell contendo o analisador sintático, outra opção é o *Lucky* [LUCKY, 2008], que se utiliza de combinadores monádicos para a implementação do analisador.

Como a linguagem a ser compilada é funcional, com tipagem estática sem a necessidade de declarações de tipos, um estudo sobre inferência de tipos e geração de código para este paradigma será necessário [JONES, 1987].

Posteriormente irá ser implementado o *front-end* e o *back-end* da linguagem baseada em Haskell gerando *bytecodes* Java.

As principais atividades do plano de trabalho estão listadas a seguir:

1. Revisão bibliográfica;
2. Estudo da especificação da JVM;
3. Análise e definição das ferramentas de geração do compilador;
4. Estudo sobre inferência de tipos e geração de código para linguagens funcionais;
5. Implementação do *front-end* (analisador sintático e inferidor de tipos);
6. Implementação do *back-end*;
7. Estudo das improváveis incompatibilidades de tipos entre as duas linguagens;
8. Escrita da Monografia;
9. Impressão e entrega dos exemplares;

10. Preparação da apresentação;
11. Defesa do trabalho de conclusão.

6. CRONOGRAMA

O cronograma abaixo ilustra as atividades do plano de trabalho descrito na seção anterior divididos em semanas.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33		
	Abril				Maio				Junho				Julho				Agosto				Setembro				Outubro		Nov.								
1	■	■	■	■																															
2					■	■	■																												
3								■																											
4								■	■	■																									
5								■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
6															■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
7																									■	■	■	■	■	■	■	■	■	■	■
8																																			
9																																			
10																																			
11																																			

7. REFERÊNCIAS

HAMILTON, JENNIFER. **Language Integration in the Common Language Runtime**. v. 38, n. 2, p. 19-28, fevereiro 2003.

BENTON, N.; KENNEDY, A. **Interlanguage working without tears: blending SML with Java**. v. 34, n. 9, p. 126-137, setembro 1999.

JONES, P. **The Implementation of Functional Programming Languages**. Prentice Hall, 1987.

LINDHOLM, T.; YELLIN, F. **The Java Virtual Machine Specification**. Disponível em: <http://java.sun.com/docs/books/jvms/second_edition/html/VMSpecTOC.doc.html>. Acesso em: abril 2008.

COM. Microsoft Corporation. **The Component Object Model**. Disponível em: <<http://msdn2.microsoft.com/en-us/library/ms694363.aspx>>. Acesso em: abril 2008.

CLR. Microsoft Corporation. **Common Language Runtime Overview**. Disponível em: <[http://msdn2.microsoft.com/en-us/library/ddk909ch\(VS.71\).aspx](http://msdn2.microsoft.com/en-us/library/ddk909ch(VS.71).aspx)>. Acesso em: abril 2008.

HAPPY. **Happy: The Parser Generator for Haskell**. Disponível em: <<http://www.haskell.org/happy/>>. Acesso em: abril 2008.

LUCKY. **Lucky: A Haskell parser generator using monadic parser combinators**. Disponível em: <<http://www.ki.informatik.uni-frankfurt.de/~klose/lucky.html>>. Acesso em: abril 2008.