Plano de Trabalho de Conclusão de Curso
Interpretando Efeitos Algébricos Por Meio de Mônadas

UDESC – Centro de Ciências Tecnológicas
Departamento de Ciência da Computação
Bacharelado em Ciência da Computação
Turma 2023/2 – Joinville/SC

Karla Alexsandra de Souza Joriatti – karla.joriatti@edu.udesc.br
Cristiano Damiani Vasconcellos – cristiano.vasconcellos@udesc.br *(orientador)*
Paulo Henrique Torrens – paulotorrens@gnu.org *(coorientador)*

Agosto de 2023

Resumo

Ao longo do desenvolvimento de um compilador, durante a etapa de otimização, a escolha de uma estrutura de dados intermediária impacta diretamente na capacidade e eficácia do compilador. Um exemplo bastante utilizado de representação intermediária é a forma de atribuição estática única (SSA). A forma SSA (*Single static assignment*) é um modelo de representação intermediária onde cada uso de uma variável pelo programa esteja ligado a uma única atribuição desta mesma variável. Essa propriedade torna a forma SSA equivalente a uma representação funcional, o que leva a considerar a possibilidade de explorar tal equivalência durante a análise do código. Um possível exemplo é a transformação de SSA para uma representação funcional com sistemas de efeitos, como já abordado em trabalhos anteriores. O objetivo deste trabalho é a formalização e implementação da tradução de uma representação funcional, obtida a partir de uma representação na forma SSA, que apresenta sistema de efeitos para uma representação funcional que utilize mônadas.

Palavras-chave: ***SSA, Mônadas, Efeitos***

1 Introdução e Justificativa

Durante o processo de criação de um compilador, em sua etapa de otimização, a escolha adequada de uma estrutura de dados para representação intermediária terá influência direta no poder e eficiência do compilador. A forma de atribuição estática única (*SSA - single static assingment*) é um exemplo de representação intermediária utilizada predominantemente nos compiladores de linguagens imperativas (e.g., GCC e Clang) para aplicação de otimizações baseadas em análise de fluxo de dados (CYTRON et al., 1989). Para estar na forma SSA é necessário que o programa seja modificado

de modo que cada uso de uma variável pelo programa esteja ligado a uma única atribuição desta mesma variável (WEGMAN; ZADECK, 1991).

Como observado por (APPEL, 1998), a forma de representação intermediária SSA é equivalente a um programa em linguagem funcional. Tais linguagens utilizam as definições de cálculo lambda apresentadas por (CHURCH, 1932) como base, programando por meio de funções aninhadas. Ademais, programas em linguagens funcionais apresentam uma série de vantagens que podem ser aproveitadas na etapa de otimização, como, por exemplo, códigos funcionais puros separados de códigos com efeitos (Haskell e Miranda), modularidade (e.g., utilização de funções de ordem superior para "colar"funções e método de avaliação preguiçosa que permite "colar"programas), garantias comuns a linguagens funcionais (principalmente linguagens funcionais puras), entre outros (HUGHES, 1989).

Diversos trabalhos se basearam na descoberta de Appel sobre a equivalência entre SSA e linguagens funcionais. Dentre tais contribuições, vale mencionar o trabalho de (TORRENS; VASCONCELLOS; GONÇALVES, 2017), que sugerem uma linguagem intermediária capaz de ser interpretada tanto como SSA quanto como uma representação funcional pura. Além do trabalho supracitado, (RIGON; TORRENS; VASCONCELLOS, 2020) também criam uma representação funcional a partir de uma representação na forma SSA com inferência e tratamento de efeitos por meio de um sistema de efeitos colaterais. O sistema de tipos e efeitos apresentado por Rigon, Torrens e Vaconcellos, baseado no trabalho de (LEIJEN, 2014), possui as vantagens de uma linguagem impura, porém ainda permite uma menor incidência de erros devido a identificação de trechos de código que causam efeitos colaterais (i.e., alterações em estados de memória). O sistema de efeitos impõe que as generalizações sejam aplicadas somente a computações puras e ainda garante que expressões com efeitos colaterais estejam encapsuladas de forma a não afetar o restante do programa. Além do sistema de efeitos, também foi aplicada a transformação de SSA para uma representação ANF (*A-normal-form*) capaz de deixar um programa mais natural e intuitivo e permitir a compilação de efeitos, diferente da técnica mais comumente utilizada, a passagem para CPS (*continuation-passing-style*), onde os efeitos serão removidos após a transformação (FLANAGAN et al., 1993). O objetivo deste trabalho é apresentar uma versão de representação funcional intermediária equivalente a apresentada por Rigon, Torrens e Vasconcellos, porém com tratamento de efeitos por meio do uso de mônadas (WADLER, 1995). A tradução de código funcional com sistema de efeitos para um código funcional que utiliza mônadas será baseado no trabalho de (VAZOU; LEIJEN, 2016), que apresentam a formalização da tradução de sistema de efeitos para mônadas.

2 Objetivos

O trabalho propõe verificar a viabilidade de criar um código equivalente a representação funcional apresentada por (RIGON; TORRENS; VASCONCELLOS, 2020), porém utilizando mônadas para lidar com efeitos. Dessa forma, isolar efeitos no código monádico pode ajudar nas análises intermediárias e, futuramente, poderá ser usado como base para auxiliar trabalhos focados na extração de informações diretamente da representação na forma SSA.

3 Metodologia

O desenvolvimento do trabalho será feito em duas fases; a primeira fase abrangerá uma ampla revisão da literatura sobre o tema e a segunda fase se dará pela formalização e implementação da tradução da representação funcional com sistemas de efeitos (RIGON; TORRENS; VASCONCELLOS, 2020) para uma representação que trate efeitos com mônadas. Destacam-se as seguintes etapas:

1. Estudo sobre representações intermediárias, sistemas de efeito e mônadas
2. Formalização da tradução de código funcional com sistema de efeitos para mônadas
3. Implementação da tradução formalizada na segunda etapa
4. Análise dos resultados
5. Escrita do texto

A análise dos resultados será feita a partir de testes sobre os códigos utilizados por Rigon, Torrens e Vasconcellos, verificando se a geração de código monádico equivalente à representação funcional com sistema de efeitos foi possível.

4 Cronograma proposto

Etapas	**2023/2**					**2024/1**					
	Ago	**Set**	**Out**	**Nov**	**Dez**	**Fev**	**Mar**	**Abr**	**Maio**	**jun**	**Jul**
1	■	■	■								
2			■	■	■	■					
3				■	■	■	■	■	■		
4								■	■	■	■
5		■	■	■	■	■	■	■	■	■	■

5 Linha e Grupo de Pesquisa

O tema proposto para o trabalho se enquadra nos temas pesquisados pelo Grupo de Pesquisa em Fundamentos da Computação (FUNÇÃO). O tema é pertinente à linha de pesquisa em linguagens funcionais e sistema de tipos.

6 Forma de Acompanhamento/Orientação

O acompanhamento das atividades desenvolvidas será realizada em reuniões semanais entre orientador, coorientador e aluno, presenciais e/ou via chat, com até uma hora de duração e possibilidade de agendar encontros extras, caso necessário. Também serão utilizados recursos como correio eletrônico para, caso necessário, orientação ao longo da semana.

Para acompanhamento do progresso, serão entregues aos orientadores artefatos com os resultados obtidos em cada etapa (quando aplicável), para que assim exista uma avaliação contínua com mais ciclos de retorno.

Referências

APPEL, A. W. Ssa is functional programming. **Acm Sigplan Notices**, ACM New York, NY, USA, v. 33, n. 4, p. 17–20, 1998.

CHURCH, A. A set of postulates for the foundation of logic. **Annals of mathematics**, JSTOR, p. 346–366, 1932.

CYTRON, R. et al. An efficient method of computing static single assignment form. In: **Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages**. [S.l.: s.n.], 1989. p. 25–35.

FLANAGAN, C. et al. The essence of compiling with continuations. In: **Proceedings of the ACM SIGPLAN 1993 conference on Programming language design and implementation**. [S.l.: s.n.], 1993. p. 237–247.

HUGHES, J. Why functional programming matters. **The computer journal**, Oxford University Press, v. 32, n. 2, p. 98–107, 1989.

LEIJEN, D. Koka: Programming with row polymorphic effect types. **arXiv preprint arXiv:1406.2061**, 2014.

RIGON, L. F.; TORRENS, P.; VASCONCELLOS, C. Inferring types and effects via static single assignment. In: **Proceedings of the 35th Annual ACM Symposium on Applied Computing**. [S.l.: s.n.], 2020. p. 1314–1321.

TORRENS, P.; VASCONCELLOS, C.; GONÇALVES, J. A hybrid intermediate language between ssa and cps. In: **Proceedings of the 21st Brazilian Symposium on Programming Languages**. [S.l.: s.n.], 2017. p. 1–3.

VAZOU, N.; LEIJEN, D. From monads to effects and back. In: SPRINGER. **Practical Aspects of Declarative Languages: 18th International Symposium, PADL 2016, St. Petersburg, FL, USA, January 18-19, 2016. Proceedings 18**. [S.l.], 2016. p. 169–186.

WADLER, P. Monads for functional programming. In: SPRINGER. **Advanced Functional Programming: First International Spring School on Advanced Functional Programming Techniques Båstad, Sweden, May 24–30, 1995 Tutorial Text 1**. [S.l.], 1995. p. 24–52.

WEGMAN, M. N.; ZADECK, F. K. Constant propagation with conditional branches. **ACM Transactions on Programming Languages and Systems (TOPLAS)**, ACM New York, NY, USA, v. 13, n. 2, p. 181–210, 1991.

Cristiano Damiani Vasconcellos	***Paulo Henrique Torrens***
Orientador	*Coorientador*

Karina Girardi Roggia
Líder do Grupo de Pesquisa em Fundamentos da Computação

Karla Alexsandra de Souza Joriatti