Implementação e análise de uma engine para expressões regulares em Coq via testes baseados em propriedades

Marcos Benevides[1], **Sérgio Souza Costa**[2], **Rodrigo Geraldo Ribeiro**[3]

[1]Departamento de Informática – Universidade Federal do Maranhão (UFMA)

[2]Engenharia da Computação – Universidade Federal do Maranhão (UFMA)

[3]Departamento de Sistemas e Computação – Universidade Federal de Ouro Preto (UFOP)

marcos.schonfinkel@gmail.com,

sergio.costa@ufma.br, rodrigo.ribeiro@ufop.edu.br

Resumo. *Este artigo descreve um resumo de um trabalho de conclusão de curso de mesmo nome.*

1. Introdução

Um provador interativo de teoremas (ou assistentes de provas) é uma ferramenta que auxilia o desenvolvimento de provas formais. Este trabalho buscou analisar um algoritmo descrito na *functional pearl* "A Play on Regular Expressions"[Fischer et al. 2010], explorando as metodologias usadas na criação de sistemas corretos de acordo com sua especificação, utilizando técnicas como os testes baseados em propriedades e o assistente de provas COQ [Team 2020].

COQ é nomeado em homenagem um de seus criadores, Thierry Coquand e foi criado para desenvolver provas matemáticas, escrever especificações formais de programas e verificar sua corretude com respeito à especificação. Os termos dessa linguagem podem representar programas, propriedades e provas de propriedades.

2. O Algoritmo

[Fischer et al. 2010] apresentaram o algoritmo em três atos, cada qual subdividido em 2 ou 3 cenas. Cada cena descreve uma implementação diferente para expressões regulares, que aumentam gradualmente em nível de complexidade e generalidade.

O primeiro ato divide-se em duas cenas, que focam em modelar `regexes` como `Algebraic Data Types`, os pesos foram adicionados à estrutura via `semirings`. O segundo ato implementa o algoritmo de Glushkov para `matchings` mais eficientes e adiciona a funcionalidade de computar o maior matching à esquerda. O último ato adiciona `lazyness` na versão do ato anterior. Ao final do artigo original tem-se uma implementação elegante em Haskell.

3. Property-Based Testing

Testes são uma das maneiras mais básicas se garantir a robustez de um software, sendo assim a qualidade do software pode estar atrelada a quão bem escritos são seus testes.

Uma vantagem de se construir programas com funções puras é que testá-las se torna algo menos laborioso, visto que não é mais necessário se preocupar com o estado antes e depois de sua execução.

[Claessen and Hughes 2011] descrevem que o custo para escrita de testes motiva esforços para automatizá-los. O resultado de seu trabalho conjunto foi o QuickCheck, uma biblioteca em Haskell para geração de testes aleatórios a partir de propriedades.

Em ferramentas como QuickCheck, um programador deve definir propriedades que precisam ser satisfeitas (usando combinadores disponibilizados pela biblioteca). Esses combinadores são utilizados para gerar a distribuição dos dados, criando automaticamente casos de testes, quando um contra-exemplo à propriedade é encontrado, o QuickCheck tenta minimizá-lo de forma a encontrar um contra-exemplo mínimo, i.e. a caso mais simples que falsifica uma propriedade.

4. Implementação parcial em COQ

Diferentemente da implementação em Haskell, o sistema de tipos usado em COQ força que provemos certas propriedades formalmente. Por exemplo, ao utilizar um `semiring`, temos que garantir formalmente (via uma prova) que essa estrutura obedece todas as propriedades algébricas.

A biblioteca utilizada para testes baseados em propriedades é a QuickChick, que implementa um protocolo similar ao QuickCheck original em Haskell. Dessa forma, a responsabilidade do programador é transferida da elaboração de casos de testes para a criação de bons geradores e especificação de boas propriedades, utilizando as primitivas já fornecidas pela biblioteca.

Para a geraração de `strings`, é necessária a criação de geradores de caracteres `ascii`. Estruturas mais complexas e recursivas (como árvores) é necessário garantir que o gerador termine, sendo comum passar a profundidade da árvore como argumento. Os `regexes` definidos na implementação de [Fischer et al. 2010] são basicamente árvores.

5. Conclusão

O projeto focou em ser uma exploração básica das ferramentas e metodologias utilizadas para justificar a corretude matemática da especificação de programas. Apenas o primeiro ato do algoritmo foi implementado, uma formalização completa do problema poderia demorar e ir além de um trabalho de conclusão de curso. Contudo, a utilização de ferramentas como COQ, conceitos da programação funcional e metodologias como testes baseados em propriedades podem servir como fundamento para trabalhos mais complexos.

Referências

Claessen, K. and Hughes, J. (2011). Quickcheck: a lightweight tool for random testing of haskell programs. *Acm sigplan notices*, 46(4):53–64.

Fischer, S., Huch, F., and Wilke, T. (2010). A play on regular expressions: functional pearl. In *ACM Sigplan Notices*, volume 45, pages 357–368. ACM.

Team, T. C. D. (2020). The coq proof assistant, version 8.11.0.