Correção de Programas

Juliana Carpes Imperial[1]

[1]Departamento de Informática – Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio)
Rua Marquês de São Vicente 225 – 22.453-900 – Rio de Janeiro – RJ – Brazil

{juliana}@inf.puc-rio.br

1. Introdução

Este texto descreve um corretor formal de programas utilizando o cálculo de Hoare [1] na linguagem Prolog [4]. Ele é interativo com o usuário, perguntando, além das pré e pós-condições do trecho de programa a ser corrigido, quais são os invariantes para os *loops*, já que determinar o melhor invariante não é decidível.

Este programa foi desenvolvido para auxiliar os alunos de graduação e de pós-graduação das cadeira de Lógica e Especificação e similares. Com o mesmo, podem testar exemplos vistos na teoria e ver se o programa gera uma solução igual ou parecida com a prova que fizeram de forma que os permita verificar a corretude da mesma. Ainda, podem usar o corretor de programas para ver se um dado trecho de código satisfaz um par de pré e pós-condições.

Por fim, este projeto também tem como finalidade ajudar no entendimento do funcionamento de um programa que analisa um código e determina que sentenças devem ser demonstradas para que o mesmo satisfaça determinadas propriedades. Isso é extremamente útil para *Proof-Carrying Code* (PCC) [3], a qual é uma técnica que pode ser utilizada para a execução segura de código não-confiável. Em uma instância típica de PCC, quem vai executar o código estabelece um conjunto de regras de segurança que garantem o comportamento seguro de programas, e o produtor do código cria uma prova formal de segurança que garante a aderência às regras de segurança ao código não-confiável. Então, o primeiro é capaz de usar um validador de provas simples e rápido para verificar, com exatidão, que a prova é válida e, conseqüentemente, que o código não-confiável é seguro para ser executado. Em PCC, quem vai executar o código entra com as pré e pós-condições necessárias para a correção formal de programas utilizando o método citado. Já os invariantes de *loops* são dados pelo programador, porque é necessário ler o código para descobri-los.

2. Características do Programa

1. O trecho de código a ser corrigido deve estar escrito na linguagem utilizada pelo corretor de programas;
2. Para cada *loop* do trecho de código a ser corrigido, um invariante é pedido ao usuário;
3. O programa dá como resposta uma única prova de correção de programa, onde há um conjunto de sentenças em lógica clássica que devem ser demonstradas para que a prova seja correta;
4. Para trechos de código que possuem a pós-condição derivada de sua pré-condição, a prova possui todas as sentenças de seu conjunto demonstráveis;

5. Para trechos de código que não possuem a pós-condição derivada de sua pré-condição, a prova resultante não tem todas as sentenças de seu conjunto demonstráveis;
6. A saída da provas é num arquivo, num formato que "facilite" o entendimento das provas pelo usuário;
7. Se a linguagem do trecho de código for modificada, é fácil alterar o programa (seja nas análises léxica e sintática do arquivo que contém o código que será corrigido ou seja na hora de aplicar as regras do cálculo de Hoare por haver estruturas de controle diferentes).

3. Funcionamento do Programa

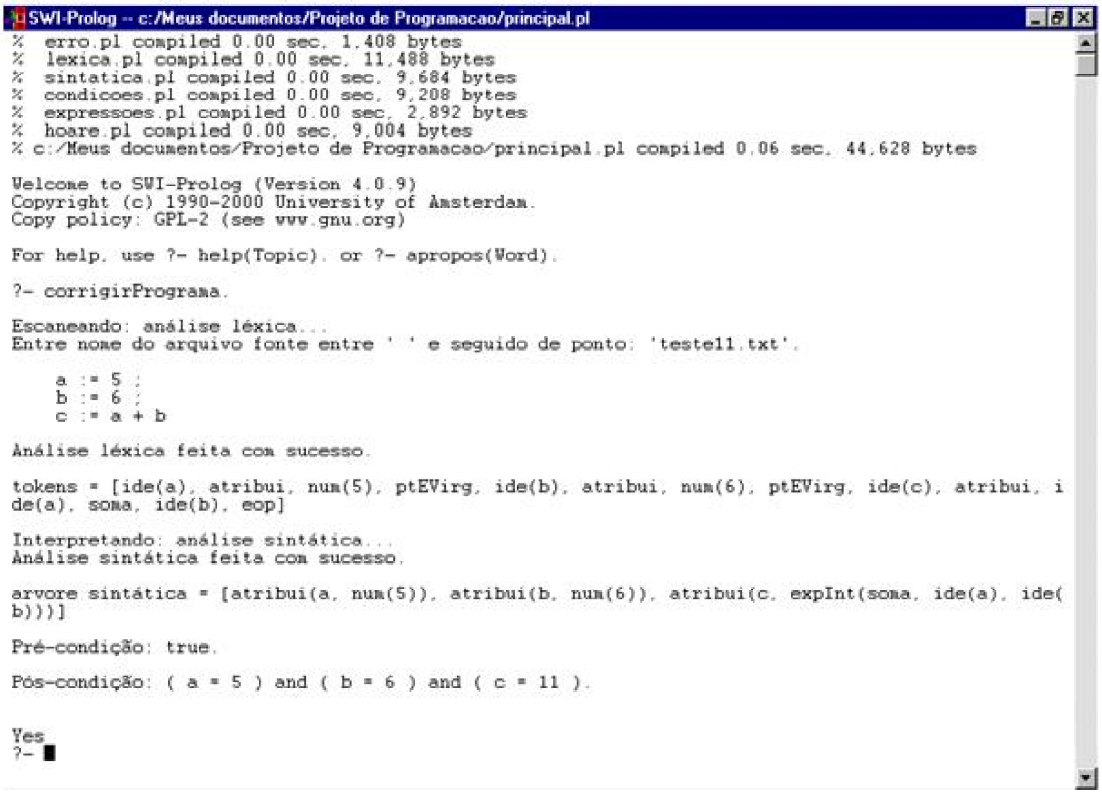
1. Como o programa foi feito em Prolog, uma linguagem interpretada, ele precisa ser carregado pelo interpretador da linguagem para que o mesmo possa começar;
2. Feito isso, o usuário deve chamar o predicado do programa principal, escrevendo o nome do mesmo no prompt do interpretador;
3. O usuário deve digitar o nome do arquivo com o trecho de código a ser corrigido.
4. É feita a análise léxica do trecho de código de acordo com uma determinada linguagem. Se algum erro léxico for encontrado, o caractere inválido será exibido ao usuário e o programa será abortado;
5. Se a análise léxica tiver sido feita com sucesso, isso é avisado ao usuário e é iniciada a análise sintática da lista de *tokens* resultante da análise anterior. Se algum erro sintático for encontrado, serão exibidos ao usuário os *tokens* reconhecidos com sucesso e a parte que não foi lida, onde provavelmente está o erro (geralmente no início da lista de *tokens* não-reconhecidos). Após isso, o programa é abortado. Se o trecho de código estiver sintaticamente correto, o usuário é avisado deste fato;
6. Se nenhum dos erros ditos acima tiver acontecido, o programa vai pedir a pré e a pós-condição para o trecho de código a ser corrigido. Se o formato de alguma delas estiver incorreto, a condição é pedida novamente;
7. Para cada *loop* (laço *while*) do trecho de código a ser corrigido, será pedido um invariante. Novamente, se o formato de algum deles estiver incorreto, o invariante sendo lido é pedido novamente;
8. As regras do cálculo de Hoare são aplicadas ao trecho de código a ser corrigido e é geradas uma prova de correção para o mesmo utilizando as estratégias descritas em [2], que permitem que uma única prova seja encontrada de forma que não seja necessário o uso de *backtracking*, o que tornaria o processo ineficiente;
9. Para a prova gerada, há um conjunto de sentenças que devem ser demonstradas para que a pós-condição seja válida a partir da pré-condição depois da execução do trecho de código.
10. A prova resultante e seu conjunto de sentenças são colocados em um arquivo HTML chamado saida.html em formato de árvore, disposta na horizontal, para facilitar o entendimento do usuário. Afinal, mesmo para pequenos programas, as provas costumam ficar grandes.
11. Por fim, o programa é terminado, podendo o interpretador Prolog ser usado novamente.

4. Interface com o Usuário

A interface do programa é a própria interface do SWI-Prolog, se for este o interpretador Prolog sendo usado. No caso do SWI-Prolog para Windows, onde o programa foi desenvolvido e testado, foi a própria janela Windows do SWI-Prolog a interface do programa, a qual é toda textual.

Nesta interface ocorre quase toda a interação com o usuário. É nela que são exibidas mensagens de erro ao usuário outras informações, e onde o usuário entra as informações que o programa precisa para executar. As únicas exceções são o trecho de código que o usuário passa ao programa e a prova e seu conjunto de sentenças, as quais devem ser demonstradas para que a sua prova esteja correta, dados como saída, o que é feito via arquivo.

Eis aqui um exemplo da execução completa do programa na janela do SWI-Prolog para Windows quando não há erros de qualquer espécie, logo após o programa ter sido carregado na memória do interpretador da linguagem:



Referências

[1] Hoare, C. A. R. An Axiomatic Basis for Computer Programming. Communications of the ACM. p 576 - 580 (1969).

[2] Imperial, J. C., Técnicas Para o Uso de Cálculo de Hoare em Proof-Carrying Code. Dissertação de Mestrado, Departamento de Informática, PUC-Rio, (2002).

[3] Necula, G. C. Proof-Carrying Code. In Symposium on Principles of Programming Languages (POPL 97), Paris, França, Proceedings. p 106 - 119 (1997).

[4] Wielemaker, J. SWI-Prolog 3.4 Reference Manual (2000).