7
Conclusões

Este trabalho mostrou como gerar provas de correção de programas de maneira eficiente utilizando o cálculo de Hoare para linguagens de programação imperativas. As heurísticas pesquisadas, embora possam gerar provas com asserções e sentenças maiores do que o necessário, fazem a prova de modo automático, a menos da demonstração das sentenças se não há vetores na linguagem.

O texto, portanto, mostrou uma abordagem para se construir provas de verificação de programas que, em particular, pode ser aplicada a *proof-carrying code* para a verificação de propriedades de segurança. Porém, esta técnica se aplica a vários tipos de aplicações além de PCC que necessitem garantir propriedades de um determinado trecho de código.

Obviamente, para se implementar a técnica de PCC por completo, seria necessário também gerar a demonstração das sentenças (o que necessita de um provador de teoremas), definir a melhor maneira para enviar a prova ao consumidor de código, e implementar um validador de provas, tanto para a prova de correção quanto para a demonstração das sentenças. Este trabalho foi centrado na primeira etapa da técnica de *proof-carrying* code por essa ser a mais difícil.

O segundo maior problema é a transmissão das provas para quem vai executar o programa. Somando a prova de correção e a prova das sentenças, o volume de dados pode ser muito extenso. Portanto, faz-se extremamente necessário não deixar que as sentenças e as asserções fiquem gigantescas para que a técnica de PCC possa ser viável.

O controle desse tamanho também é importante para a fase de verificação. Ela é simples e automática, mas pode levar muito tempo se o que tiver que ser verificado for de grande extensão. E para o consumidor de código, por mais que essa fase só seja realizada uma única vez, não é de seu desejo que ela seja muito demorada.

Por fim, tendo-se sentenças que não sejam desnecessariamente grandes, ainda assim muitas vezes as mesmas compartilham informações. Portanto, parece interessante que partes delas sejam substituídas por “macros” que

contenham as informações repetidas. Mais ainda, como possuem informações iguais, muito provavelmente possuirão partes das provas semelhantes ou mesmo iguais. Nesse caso, o uso de lemas pode ajudar a diminuir o tamanho total das provas das sentenças. Isso sem contar com a existência de sentenças que, se demonstráveis, tornam outras também válidas. Portanto, é útil prestar atenção neste fato para diminuir o tamanho das demonstrações e das próprias sentenças no momento em que se estiver criando as demonstrações necessárias.

7.1
Decisões de Projeto

O corretor de programas foi feito em Prolog, utilizando o SWI-Prolog [9]. Primeiramente, porque ele possui facilidades para lidar com as análises léxica e sintática de um programa. Afinal, além do *backtracking*, que é útil para encontrar os *tokens*, analisá-los e fazer a geração da árvore dos mesmos, o SWI-Prolog possui mecanismos embutidos que simplificam a implementação da geração dessa árvore, que são as regras de DCG (*Definite Clause Grammar*). Alternativamente, poderiam ter sido utilizadas ferramentas como o Lex e o Yacc para as análises léxica e sintática, respectivamente [44, 47].

Além disso, o *backtracking* também é útil na hora de se provar a correção de um programa e montar a prova desejada, já que facilita a escolha das regras apropriadas, sem que o usuário tenha que implementar tantos testes para saber qual a regra correta, como aconteceria se a linguagem utilizada fosse uma linguagem comum como C.

Mais ainda, em Prolog não é necessário lidar com ponteiros na hora de se utilizar árvores e listas (o que é extremamente utilizado neste programa) devido ao suporte que a linguagem dá para essas estruturas de dados.

Por fim, é possível se argumentar o porquê do não uso dos sistemas λProlog [10, 37] ou Maude [48] para a implementação do corretor de programas, já que eles possuem facilidades para a especificação de lógicas e sistemas de inferência. Deve-se considerar ainda que, com o primeiro, não seria necessário tratar “manualmente” a substituição de variáveis.

Porém, como já se possui um conhecimento grande do funcionamento do Prolog e de como se implementar programas em tal linguagem, a versão inicial do corretor sem heurísticas e sem lógica de primeira ordem já estava escrito nela e como não há o uso de lógica de ordem superior, preferiu-se continuar a

implementação do mesmo em Prolog, ao invés de traduzi-lo e fazer as alterações na versão em outra linguagem. Além disso, tem-se que a implementação da substituição de variáveis não foi muito trabalhosa.

7.2
Trabalhos Futuros

Primeiramente, seria interessante fazer uma extensão à linguagem de programação utilizada, colocando-se funções, procedimentos e recursão, e implementar tais modificações no corretor de programas de modo eficiente. Desta maneira, como o corretor estaria utilizando uma linguagem de programação muito mais próxima de uma linguagem real, seria mais fácil visualizar como seria a implementação de um para uma linguagem de fato utilizada e, conseqüentemente, de um gerador de provas para PCC.

Em segundo lugar, fazer deste corretor de programas uma ferramenta executável que possa ser facilmente utilizável e que não dependa de um interpretador de Prolog. Embora ele não utilize uma linguagem real, a ferramenta pode ser extremamente útil para a correção de algoritmos, que, posteriormente, podem ser interpretados em outra linguagem. Além disso, reescrever certas partes deste programa de modo que ele fique mais manutenível e extensível.

Em terceiro lugar, é necessário estudar e implementar uma maneira de fazer a prova de correção de programas ficar mais fácil de ser lida pelo usuário. Por exemplo, na hora de apresentá-la ao usuário, abreviar partes de um programa no arquivo de saída que não estejam sendo utilizadas em um determinado passo, o que diminui o tamanho do que está escrito e facilita sua leitura. Isso é muito importante porque, mesmo para programas pequenos como uma implementação de ordenação pelo método da bolha e achar o maior e o menor elementos de um vetor geraram provas difíceis de serem lidas.

Outra pesquisa que pode ser feita é sobre o tamanho das sentenças e das asserções. Como já foi dito, se elas possuírem grande extensão, grandes serão as demonstrações das sentenças e grande será a prova de correção, o que dificultará a transmissão de dados e a verificação no lado do usuário. No caso, a idéia é criar um programa que pegue uma prova de correção já pronta e interaja com o usuário, perguntando-lhe se uma asserção que foi originada de um enfraquecimento ou um fortalecimento pode ser simplificada e qual seria a nova asserção. Obviamente, o usuário teria de entrar informações coerentes, pois se

não o fizer estragará a prova. Não se pensa em fazer isso automaticamente pois a compactação de provas é um problema muito difícil (coNP-completo), já que isso significa tentar fazer as provas terem um tamanho polinomial.

Também seria interessante pesquisar como o cálculo de Hoare com atribuições e lógica de primeira ordem pode ser traduzido diretamente para uma extensão da AKT, a qual teria que ser modificada para tratar explicitamente as atribuições e testes de primeira ordem.

Mais ainda, fazer uma comparação do cálculo de Hoare para linguagens imperativas com relação ao cálculo para código de máquina, vendo as vantagens e desvantagens de cada um para a técnica de *proof-carrying code*. Essa comparação seria, basicamente, quanto ao tamanho das provas, à facilidade de se construir as mesmas e às aplicações para cada abordagem.

Finalizando, deseja-se saber se é possível fazer a correção de um programa com uma lógica similar ao cálculo de Hoare, porém que possua as asserções em uma lógica que não seja a lógica de primeira ordem, e que gere sentenças mais fáceis de serem demonstradas. Uma outra modificação a esse método formal seria tentar modificar as regras do **if**, de modo que a correção sempre pudesse ser feita de trás para frente. Isso é interessante, pois nenhum existencial seria inserido e nem propriedades mais fortes do que as desejadas provadas, o que certamente diminuiria os tamanhos de sentenças e asserções.