Universidade de Brasília
Instituto de Ciências Exatas

Departamento de Ciência da Computação

117366 - Lógica Computacional 1

Prova de Corretude do algoritmo Radix Sort utilizando o assistente de provas PVS

Fernanda Macedo de Sousa - 17/0010058
Rafael Gonçalves de Paulo - 17/0043959
Vinícius Costa e Silva - 15/0052138

17 de novembro de 2019

1 Introdução e Especificação do Problema

O seguinte trabalho tem como objetivo mostrar a corretude do algoritmo de ordenação **Radix Sort** utilizando o assistente de provas PVS (*PVS Specification and Verification System*). A corretude do algoritmo será demonstrada passo a passo através de três questões principais, utilizando o cálculo de sequentes de Gentzen. A prova de corretude possibilitará aos alunos da matéria de **Lógica Computacional 1** um contato maior com o assistente de provas PVS, assim como maior familiariedade com o cálculo de sequentes.

O **Radix Sort** é um dos muitos algoritmos de ordenação comuns na Ciência da Computação. Temos o problema de ordenar vários elementos inteiros, e sabemos que todos esses inteiros podem ser representados com apenas n digitos. O Radix Sort realiza essa tarefa ordenando os elementos pelo i-ésimo dígito utilizando um algoritmo *estável*. Nesse caso, um algoritmo

de ordenação é estável se preserva a ordem relativa de elementos com valores idênticos. A característica de *estabilidade* é crucial para o funcionamento do Radix Sort.

Na implementação do algoritmo Radix Sort utilizada nessa prova, o algoritmo de ordenação **MergeSort** é utilizado como sub-rotina do Radix Sort. O MergeSort pode ser utilizado pois possui a característica de estabilidade que é desejada nessa aplicação. Logo, provar a corretude do MergeSort acaba se tornando uma parte importante da prova como um todo.

A primeira parte da prova, então, se baseia em provar a seguinte conjectura: dada uma lista de elementos de entrada, o MergeSort deve retornar uma permutação dessa lista. Ou seja, provar que o MergeSort preserva os dados.

A segunda parte da prova é uma consequência direta da primeira. Sabendo-se que o Radix Sort utiliza o MergeSort como subrotina, quer-se saber que dada uma lista de entrada, o Radix Sort também retorna uma permutação dos elementos dessa lista. Ou seja, mais uma vez verifica-se se o algoritmo preserva os dados.

Por último, a terceira parte da prova consiste em mostrar que o Radix Sort de fato realiza a ordenação dos elementos da lista de entrada. Para essa prova, são utilizadas propriedades conhecidos do algoritmo MergeSort, uma delas envolvendo o conceito de *estabilidade* discutido acima.

Entendida a especificação do problema, foi utilizado o assistente de provas PVS para auxiliar no desenvolvimento das provas das três partes do projeto. Para tal, foi utilizado material fornecido pelo professor, tais como um conjunto de lemas já provados que poderiam ser utilizados nas provas finais, assim com as conjecturas já formadas. Desse modo, o grupo pode se concentrar em aplicar os conceitos de cálculo de sequentes de Gentzen para realização das provas.

2 Explicação das Soluções

Nessa seção serão explicadas as formalizações utilizadas dentro do PVS para solução das três questões propostas no projeto, cujo objetivo final é demonstrar a corretude do algoritmo de ordenação Radix Sort. Ao longo da explicação das soluções serão apresentados alguns dos comandos de prova utilizados dentro do PVS, os quais correspondem diretamente com operações do cálculo de Gentzen.

2.1 Questão 1 - MergeSort retorna permutação da lista de entrada

A primeira parte da prova consiste em demonstrar que o algoritmo de ordenação MergeSort retorna uma permutação da lista de entrada. Para tal, será necessário realizar uma indução no comprimento da lista e aplicação de algumas propriedades da função *occurrences*, cuja definição foi disponibilizada junto do material de apoio para o projeto.

Fazendo-se a indução no comprimento da lista, a ideia seguinte foi expandir a definição do *mergesort* dentro do sequente. Tal expansão nos levou a considerar dois casos diferentes: o caso em que o comprimento da lista fosse menor que 1 e o caso em que isso não fosse verdade. Utilizando-se os comandos *lift-if* e *prop* do PVS, foi possível separar a árvore de prova em dois ramos diferentes para tratar cada um desses casos. O primeiro caso revelou-se trivial, sendo resolvido com uma expansão da definição de *permutations*, enquanto que todo o restante da prova foi definido no segundo ramo da árvore.

Para tratar esse segundo ramo da árvore, foi necessário incluir um lema: *mergeocurrence*, que já havia sido provado antes. Esse lema mostra que as ocorrências de um elemento x dentro da união (merge) de duas listas é igual a soma do número de ocorrências em cada uma das listas individualmente. Também foi necessário incluir e provar o lema *lengthprefix*, o qual mostra que dado uma lista l e um inteiro n menor que o comprimento da lista, o comprimento do prefixo dessa lista (*length(l, n)*) é igual a n. Foi provado e instanciado um lema semelhante chamado *lengthsuffix*, que mostra o mesmo resultado para o sufixo de uma lista.

Com esses lemas, foi possível manipular os sequentes de modo que a resposta já parecia bastante próxima, bastando então tratar das ocorrências dos elementos dentro da lista. Para tal, mais um lema seria necessário. Esse último lema a ser incluso na prova demonstrava que o número de ocorrências de um elemento x dentro de uma lista l era igual ao número de ocorrências desse elemento no prefixo de l mais o número de ocorrências no sufixo de l. Instanciando esse lema na prova foi possível utiliza a função *occurrences* para terminar a prova da primeira questão, demonstrando finalmente que o MergeSort de fato retorna uma permutação dos elementos da lista de entrada.

Para resolver essa primeira questão, a estratégia adotada pelo grupo foi desenvolver e aplicar lemas que facilitassem a prova, tornando-a mais modular. Desse modo, ao invés de realizar uma única prova muito grande, o grupo

poderia dividir seus esforços em conseguir provar elementos menores da prova completa, juntando-os todos numa única prova final que fosse mais simples de se compreender. Sob essa perspectiva, a resolução da primeira questão foi um sucesso, pois foi possível obter uma prova relativamente simples através da aplicação dos lemas, não sendo necessários passos desnecessários. Essa abordagem foi utilizada após uma primeira tentativa que não foi bem sucedida de tentar resolver todo o problema de uma só vez. O resultado foi uma prova muito maior e mais extensa, além de confusa de entender. Mesmo chegando-se ao resultado final, o grupo percebeu que uma abordagem mais modular para a prova seria melhor e facilitaria as próximas questões do projeto, e foi essa a versão final explicada acima.

2.2 Questão 2 - RadixSort retorna uma permutação dos elementos da lista de entrada

Para iniciar a prova da segunda questão, também foi utilizada uma indução no comprimento da lista de entrada, e após manipular os sequentes (através da utilização dos comandos *skolem* e *flatten*), foi expandida a definição de *radixsort* e em seguida introduzido o lema *merge sort is permutation*, que já havia sido provado anteriormente. Lembrando que o Radix Sort utiliza o MergeSort como algoritmo intermediário, já era previsto que esse lema teria de ser usado em algum momento para provar a corretude do Radix Sort.

No entanto, foi necessário instanciar esse lema duas vezes, considerando dois tipos de ordenação diferentes ($\ll e <=$).

Por último, foi necessário incluir mais um lema (também provado anteriormente). A esse lema foi dado o nome *permutation switch*. A ideia do lema é intuitiva, mas suas inclusão era necessária para possibilitar a conclusão da prova no PVS. Dadas três listas (*l1, l2, l3*), se *l1* é permutação de *l2* e *l2* é permutação de *l3*, então *l1* é permutação de *l3*. Atráves da instanciação desse lema como uma das hipóteses, ficou aparente que seria possível manipular o sequente de modo a completar a prova, e por isso foi utilizado o comando *assert* para concluir a prova e assim demonstrar que o Radix Sort de fato retorna uma permutação dos elementos da lista de entrada.

Essa prova se mostrou muito mais curta do que a primeira. De fato, ao provar que essa propriedade (preservação dos dados) é verdade para o caso do MergeSort, torna-se muito mais fácil provar o mesmo para o RadixSort. Desse modo, mostra-se também que dividir a prova em passos menores é uma

estratégia eficiente de resolução utilizando o PVS

2.3 Questão 3 - RadixSort realiza ordenação dos elementos da lista de entrada

A prova final consiste em demonstrar que o algoritmo em questão (Radix Sort) de fato ordena os elementos dada uma lista de entrada. Para realizar essa prova, será utilizada uma função chamada *is sorted?*, responsável por checar se o resultado da aplicação do Radix Sort numa lista de entrada retorna uma lista ordenada.

Diferentemente da primeira questão, nesse caso não foi utilizada uma indução no comprimento da lista. A prova foi iniciada com o comando *skolem* para exclusão do "para todo", seguido da expansão das definições de *radixsort* e *is sorted?*. Depois desses passos iniciais, o primeiro lema foi introduzido, conjecturando que o Merge Sort retorna uma lista ordenada (esse lema já havia sido provado anteriormente). Assim como na questão anterior, foi necessário instanciar esse lema utilizando duas ordenações distintas ($\ll e <=$).

Também foi necessário a inclusão de um segundo lema, o qual conjectura que o Merge Sort é conservativo.

Após a inclusão desses lemas, foi necessário realizar algumas manipulações nos sequentes antes que fosse possível enxergar um caminho para a solução completa. O que faltava era a inclusão de um último lema, *is sorted implies monotone*. Esse foi um dos pontos mais difíceis para a prova, pois o grupo não era familiarizado com o conceito de monotonicidade. Em matemática, no entanto, uma função monótona é aquela que preserva um dado ordenamento. O conceito de monotonicidade é importante no caso de algoritmos de ordenação pois implica a preservação de uma ordem. A aplicação e instanciação desse lema era a peça que faltava para concluir a realização da prova.

Com a conclusão dessa prova, o objetivo inicial (a prova de corretude do algoritmo Radix Sort) estava completa.

3 Conclusão

Através da realização desse projeto foi possível compreender como é possível utilizar as formalizações do cálculo de sequentes de Gentzen para a prova de

corretude de algoritmos. Desse modo, foi possível perceber como é possível provar que um algoritmo é correto quebrando-o em seus elementos fundamentais, e então utilizando formalizações lógicas para conseguir provar a corretude das peças individuais. Além disso, o projeto possibilitou um estudo aprofundado do assistente de provas PVS, possibilitando aos alunos um contato inicial com uma ferramenta avançada utilizada na academia para a prova de algoritmos complexos.

Por último, o projeto como um todo acaba sendo uma maneira interessante de abordar uma gama ampla de conceitos fundamentais em Ciência da Computação: do entendimento de algoritmos até a aplicação de conceitos de lógica. Através das dificuldades encontradas foi possível obter um entendimento melhor de como funciona o assistente de provas e também de como utilizar o cálculo de sequentes para a resolução de problemas mais complexos do que os vistos em sala de aula.

4 Referências

AYALA-RINCÓN, Mauricio; MOURA, Flávio L.C. Applied Logic for Computer Scientists. 1. ed. Brasília: Springer, 2017.

T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction to Algorithms , 2nd edition, MIT Press McGraw-Hill, 2001