Universidade de Brasília
Instituto de Ciências Exatas

Departamento de Ciência da Computação

117366 - Lógica Computacional 1

Relatório sobre **prova de corretude do algoritmo Radix Sort**

Bruno Esteves - 170100863
Carolina Estrella Machado - 180074792
Raphael Paula Leite Müller - 170113477

18 de novembro de 2019

1 Introdução

As regras dedutivas são implementadas em diversos sistemas computacionais como provas de teoremas e assistentes de provas. Hoje em dia, um dos principais desafios enfrentados pelo programadores é relacionado com o desenvolvimento de ferramentas capazes de auxiliar engenheiros e cientistas a verificar matematicamente softwares e hardware. Isso pode ser feito em diversas ferramentas computacionais, sendo o Prototype Verification System (PVS) escolhido nesse caso.

O trabalho a seguir tem como objetivo mostrar a corretude do algoritmo de ordenação **Radix Sort** utilizando o assistente de provas **PVS**. Para demonstrar a corretude do algoritmo, será explicado no desenvolvimento as três questões do projeto, que foram resolvidas utilizando o cálculo de sequentes de Gentzen.

2 Contextualização do Problema

Entre os diversos algoritmos de ordenação utilizados na Ciência da Computação está o **Radix Sort**, que ordena A [1 . . . n] pelo i-ésimo dígito dos números em A por meio de um algoritmo estável. Nesse projeto utilizamos o algoritmo **MergeSort** como algoritmo auxiliar assumido estável de forma geral. O MergeSort é um algortimo de ordenação recursivo, que ordena as duas metades do vetor utilizando o método de dividir e conquistar.

A primeira parte da prova se concentra em provar a corretude do MergeSort, que será importante para continuidade das seguintes provas. Sendo assim, dada uma lista de N elementos, o MergeSort deve retornar uma permutação dessa lista, ou seja, demonstrar que preserva o conteúdo da lista sendo ordenada.

A segunda parte da prova consiste em provar que, ao utilizar MergeSort como subrotina, a função Radix Sort também produz uma permutação da lista de entrada.

Por fim, a terceira parte da prova conjectura que o Radix Sort é de fato um algoritmo de ordenação dada uma lista de entrada. Para isso, é preciso utilizar as propriedades do MergeSort, que em sua definição é um algoritmo estável, além de respeitar a ordem lexicográfica construídas das pré-ordens $<<$ e $<=:$.

3 Explicação das soluções

3.1 Questão 1

A primeira questão consiste em provar que a saída do algoritmo MergeSort produz uma permutação da lista de entrada. Para tal, a estratégia utilizada foi de indução no tamanho da lista de entrada.

Após a especificação da indução como estratégia de prova, o próximo passo consistiu na expansão da definição de MergeSort, gerando o sequente descrito na figura 1.

Tal definição pode ser analisada em duas etapas: A primeira consiste em provar a conjectura descrita para qualquer lista unitária, enquanto a segunda visa provar tal definição para uma lista com mais de um elemento.

A primeira etapa pode ser facilmente provada ao se expandir a definição de *permutations*, que caracteriza uma permutação como uma igualdade entre

```
merge_sort_is_permutation :

[-1]  FORALL (y: list[T]):
        length(y) < length(x) IMPLIES permutations(merge_sort(y), y)
  |-------
{1}   permutations(IF length(x) <= 1 THEN x
                   ELSE merge(merge_sort(prefix(x, floor(length(x) / 2))),
                              merge_sort(suffix(x, floor(length(x) / 2))))
                   ENDIF,
                   x)
```

Figura 1: Sequente gerado pela expansão da definiçao de merge sort

as ocorrências dos elementos de duas listas. Assim, o MergeSort de uma lista unitária é trivialmente verdadeiro, visto que este elemento seria mantido após a aplicação desse algoritmo.

A segunda etapa, por sua vez, depende da aplicação de lemas para provar a conjectura estabelecida. O primeiro deles é o lema *merge_occurrence*, que diz que o *merge* de duas listas é equivalente à concatenação das mesmas.

Em seguida, foram utilizados os lemas *length_prefix* e *length_suffix*. A aplicação desses lemas garantirá, posteriormente, que a soma dos tamanhos do prefixo e do sufixo de uma lista l serão equivalentes ao próprio tamanho de l. Isso é fundamental para a prova, visto que o MergeSort consiste em divisões subsequentes de uma lista.

Por fim, foi aplicado o lema *occurrences_of_app*, que afirma que as ocorrências dos elementos de um prefixo e um sufixo serão preservadas ao concatenar ambos.

Em suma, a combinação das definições proporcionadas pelos lemas acima sustenta que a concatenação de duas listas produzirá uma permutação em relação às entradas. Visto que a fase de conquista do MergeSort consiste no *merge*, ou concatenação, de diversas listas ordenadas, conclui-se então que este algoritmo retorna uma permutação da entrada.

3.2 Questão 2

```
% Radixsort mege-sorts a list of elements of type T by the total preorders
% <= and <<
 radixsort(l : list[T]) : list[T] = merge_sort[T,<<](merge_sort[T,<=](l))
```

Figura 2: Definição Radix Sort

Na segunda questão nos deparamos com a definição do Radix Sort, onde temos um MergeSort de um MergeSort anterior. Sabemos que MergeSort foi definido para qualquer tipo T de uma lista, ou seja, pode-se aplicar um MergeSort de um MergeSort sem contradizer sua definição.

Como na questão anterior conseguimos provar a conjectura *merge_is_permutation*, podemos garantir que um Merge de outro Merge também será uma permutação da própria lista original.

Através da expansão de RadixSort encontramos um formato que pudemos aplicar o lema anterior *merge_is_permutation*, assim seguindo a prova através de expansões e instanciações chegamos à uma igualdade que, após instanciada é tida como verdadeira.

3.3 Questão 3

Apesar de não termos apresentado a prova dessa questão, sabemos que sua prova é realizada a partir da função *is sorted?*, que checa se o Merge Sort retorna realmente uma lista ordenada.

4 Conclusão

Com esse projeto tivemos nossa primeira experiência com assistentes de prova. Pudemos entender a importância de uma prova da corretude de algoritmos e os problemas que podem ser evitados com isso. Com a assistência do PVS conseguimos provar duas das três conjecturas propostas através de demonstrações e aplicações dos lemas e definições que nos foram disponibilizados.

5 Referências

M. Ayala-Rincon and F. L. C. de Moura. Applied Logic for Computer Scientists - Computational Deduction and Formal Proofs. Undergraduate Topics in Computer Science. Springer, 2017.