Universidade de Brasília

Departamento de Ciência da Computação

Relatório do Projeto de LC1

Autores:

Gabriel Bessa 16/0120811

Vitor Dullens 16/0148260

Giovanni Guidini 16/0122660

Diogo Pontes 16/0117992

Disciplina:

Lógica Computacional 1

Turma:

B

Professor:

Flavio Moura

Brasília
17 de junho de 2018

1 Introdução

Este trabalho trata da prova de correção da primeira fase do algoritmo Ford-Johnson aplicando dedução da lógica de predicados usando o assistente de provas PVS com as bibliotecas da nasa [1].

O PVS é um assistente de provas da NASA que consiste em uma linguagem de especificação, que é usada para especifiar teorias, conjecturas, lemas, funções, procedimentos, predicados e expressões lógicas; e uma linguagem de prova, que é responsável por aplicar regras para provar as propriedades. O PVS é uma ferramenta de prova muito poderosa, porém, nesse projeto, ficaremos restritos à Lógica de Primeira Ordem (LPO).

Para provar o algoritmo de Ford-Johnson, primeiro precisamos compreendê-lo. Trata-se de um algoritmo de ordenação que busca fazer um número de comprações próximo de um limite teórico mínimo conhecido. Por exemplo, teoricamente, ordenar 13 elementos requer 33 comparações (log(n)), porém está demonstrado que ordenar esses elementos necessita de 34 comparações. O algoritmo Ford-Johnson realiza essa tarefa em 34 comparações. Apesar disso, sabe-se que esse algoritmo não é ótimo, pois ele apresenta uma eficiência menor que outros algoritmos para um número infinito de tamanhos de entrada. As estruturas de dados específicas usadas pelo algoritmo atrapalham sua implementação, por isso ele não é muito utilizado em situações práticas, mas ainda é de interesse teórico pelo baixo número de comparações que necessita.

Primeiramente explicamos o funcionamento da fase 1 do algoritmo Ford Johnson na seção 1.1. Em seguida, na seção 2 comentamos um pouco mais sobre a formalização do algoritmo, definições importantes, e quais provas foram realizadas pro nós. As seções 2.2, 2.3 e 2.4 explicam os problemas e soluções a serem provados e descrevem a demonstração feita. Na seção 3 analisamos os resultados do projeto.

1.1 Ford Johnson - Fase 1

O algoritmo Ford Johnson utiliza uma estrutura de dados própria e bastante específica, que pode ser representada graficamente com um dígrafo acíclico. Essas estruturas inicialmente possuem apenas um número natural como "raíz"da estrutura, e são alteradas conforme comparações são feitas para "guardar"as comparações de forma conveniente, diminuindo o número de comparações necessárias para se organizar uma sequência.

As comparações são feitas 2 a 2, sempre com elementos de mesma profundidade, e a cada comparação, o elemento com a maior raíz aponta para o elemento de menor raíz, aumentando assim sua profundidade. Cada com-

paração diminui o número de elementos na sequência pela metade, de forma que no total são feitas $log_2(length(s))$ comparações até que não tenhamos mais elementos para comparar. A primeira fase do algoritmo consiste exatamente em fazer essas comparações, gerando novas sequências menores e mais profundas a cada passo, até que não existam mais comparações a serem feitas.

Explicaremos a primeira fase do Algoritmo Ford Johnson utilizando um exemplo. Seja x um conjunto de número naturais com 10 elementos {4, 2, 1, 3, 7, 5, 9, 6, 8, 10}, que ordenaremos usando o algoritmo de Ford-Johnson.

O primeiro passo é comparar cada elemento com o elemento imediatamente à sua direita, separando esses elementos em uma sequência de 5 pares ordenados, como na figura 1. Note que os maiores elementos apontam para os menores elementos.

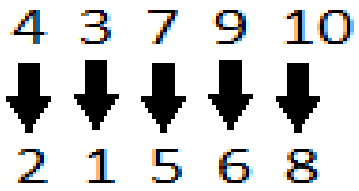

Figura 1: Pareamento inicial dos elementos

Em seguida, os elementos são novamente comparados 2 a 2, gerando uma sequência de quátruplas obtidas através da comparação dos elementos de maior valor dos pares, como na figura 2. Se houver um número ímpar de elementos, os elementos não pareados são considerados como os últimos da sequência final.

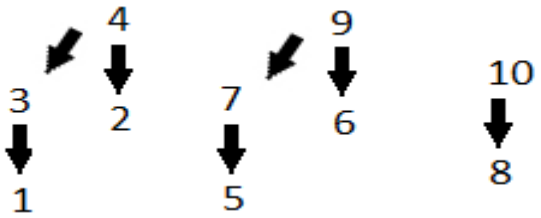

Figura 2: Sequência de quátruplas e dupla não pareada

O processo de compração das estruturas ordenadas é feito até que sobre apenas uma sequência unitária com uma única estrutura ordenada. No nosso exemplo, temos no final uma lista com uma óctupla e uma dupla, como mostra a figura 3.

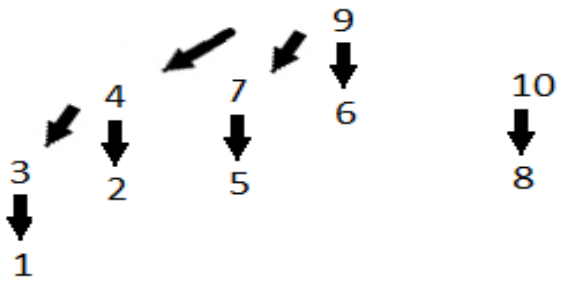

Figura 3: Lista final com uma óctupla e uma dupla

Esse é o resultado obtido ao final da primeira parte do algoritmo Ford-Johnson. Nesse projeto, queremos mostrar que a nossa especificação da primeira fase do algoritmo está correta, produzindo a estrutura desejada para uma entrada com n números naturais. O restante do algoritmo não será abordado nesse trabalho.

2 Métodos

Neste trabalho fizemos a correção da primeira fase do algoritmo de ordenação Ford Johnson, conforme definido na seção 1. A teoria que define formalmente este algoritmo foi disponibilizada para nós em arquivos do PVS [2] pelo professor. Desses arquivos o de maior interesse era `ford_johnson.pvs`, que contém o tipo `fjnode`, a estrutura usada pelo algoritmo e descrita na seção 1, e contém também a teoria `fj` que formaliza a primeira fase do algoritmo Ford Johnson, como descrito na seção 1. As funções e lemas desta formalização já estavam provados, restando a nós provar as conjecturas, separadas em questões 01, 02 e 03, respectivamente, `comparePeservesElements`, `seqfjBottPreservesElements` e `strucutural_correctionFP`. Destas, apenas as últimas duas foram consideradas obrigatórias, sendo a primeira deixada como opcional devido à complexidade da prova, que fugia aos objetivos do trabalho.

Algumas das funções e lemas definidos no arquivo `ford_johnson.pvs` serão importantes para as provas seguintes e merecem ser comentadas. Faremos isso na subseção 2.1. As subseções seguintes descrevem com mais detalhes cada uma das conjecturas e a solução encontrada. Subseções 2.2 e 2.3 tratam das conjecturas obrigatórias, cujas provas estão completas, e a subseção 2.4 trata na conjectura opcional, para a qual não conseguimos fechar a prova.

2.1 Definições da Formalização

Algumas das definições importantes na formalização da primeira fase do algoritmo Ford Johnson. Todos os itens sem especificação de tipo (entre parêntesis) são funções. Os itens abaixo formalizam a descrição da primeira fase do algoritmo como descrito na seção 1. As definições mais pontuais foram incluídas para facilitar o entendimento do leitor sobre as provas feitas para cada conjuntura, uma vez que todas as definições abaixo são importantes nessas provas.

- **fjnode** (Tipo de Dado)
  Define a estrutura de dados utilizada no algoritmo. É constituído por um valor "val" - representa o maior valor da estrutura, uma lista de fjnodes "smallerones" - as outras estruturas que já foram comparadas à atual, e um valor booleano "a?" que não é importante nesta primeira fase.

- **permutation**
  Existem duas funções permutation que definem uma permutação entre listas ou sequências de fjnodes. Duas listas (ou sequências) de fjnodes são permutação uma da outra se os elementos delas são os mesmos. Isso é feito contando o número de vezes que cada elemento aparece na estrutura da lista (ou sequência).

- **permutation_equiv** (Lema)
  Define algumas propriedades das permutações de sequências de fjnodes, à saber as propriedades reflexiva, comutativa e transitiva.

- **compare2to2**
  Define a operação de comparar dois elementos fjnode com a mesma "profundidade" e criar um novo elemente fjnode com "profundidade" maior que as dos elementos anteriores, de modo que o maior dos elementos originais se torna o "val" do novo elemento e o outro elemento se torna a cabeça da lista "smallerones"do novo elemento.

- **seqfjBottleneck**
  Define o processo de comparar elementos fjnode dois a dois até que nenhuma outra comparação possa ser feita, ou seja, esta função transforma uma sequência de naturais em fjnodes tão profundos quanto possível através da aplicação da função `compare2to2` um número determinado de vezes.

- **plain_finseqfj**
  Define uma sequência de fjnodes com profundidade 1, ou seja, que só tenham o elemento da raíz e ainda não foram comparados à nenhum outro elemento.

- **nstructER?**
  Define que o tamanho de uma sequência 'plain' de fjnodes e sua respectiva sequência obtida após a execução da função `seqfjBottleneck` possuem as propriedades esperadas, i.e são permutações, cada uma com o tamanho esperado e cada elemento com a profundidade esperada.

2.2 Questão 02 - seqfjBottPreservesElements

Esta conjectura formaliza que qualquer sequência s de fjnodes é permutação da sequência s' obtida ao aplicar a função `seqfjBottleneck` em s, isto é, comparar todos os elementos da sequência dois a dois criando elementos cada vez mais profundos até que nenhuma nova comparação possa ser feita não altera os elementos da sequência.

A prova é feita por indução no tamanho da sequência s. Basicamente vamos provar que, para qualquer sequência de fjnodes s_0, aplicar a função `seqfjBottleneck` exatamente uma vez ,i.e. ignorando a recursividade da função, não altera a sequência.

Para isso, separamos a prova em 2 casos, de acordo com a definição da função `seqfjBottleneck`: um caso em que o tamanho da sequência s_0 é maior que 1, portanto ainda existem comparações que podem ser feitas, e um segundo caso em que não existem mais comparações a serem feitas. O segundo caso é trivial, pois consiste em provar que uma sequência é permutação dela mesma.

O primeiro caso provamos utilizando a hipótese de indução (1), a conjectura `comparePeservesElements` (2)(veja seção 2.4), e a propriedade transitiva do lema `permutation_equiv`(3) (veja seção 2.1), através de instanciações convenientes, como mostrado abaixo. Tendo os antecedentes podemos concluir o consequente da implicação, que significa exatamente aplicar a função uma em qualquer sequência e o resultado ser uma nova sequência que é permutação da primeira.

$$permutation(compare2to2(s_0), seqfjBottleneck(compare2to2(s_0))) \tag{1}$$

$$permutation(s_0, compare2to2(s_0)) \tag{2}$$

$$\begin{aligned} & permutation(compare2to2(s_0), seqfjBottleneck(compare2to2(s_0))) \\ & \quad \wedge permutation(s_0, compare2to2(s_0)) \\ & \quad \rightarrow permutation(s_0, seqfjBottleneck(compare2to2(s_0)) \end{aligned} \tag{3}$$

Podemos instanciar os lemas utilizados com a sequência $compare2to2(s_0)$, pois ela atende à restrição que o tamanho dela deve ser menor que o tamanho de s_0. Sabemos por hipótese que a sequência s_0 possui tamanho maior do que 1, pois ela está sendo analisada no primeiro caso da prova, logo para provar que o tamanho da sequência $compare2to2(s_0)$ é menor que de s_0 basta aplicar a definição de `compare2to2`.

Isso conclui a prova da conjectura, pois mostra que para qualquer sequência podemos aplicar a função `seqfjBottleneck` uma vez, e que a condição de parada também não altera a sequência. Fazendo isso começando com a sequência inicial e aplicando sucessivamente nas sequências geradas a cada passo, mostramos que a as comparações sucessivas do algoritmo não altera a sequência.

2.3 Questão 03 - strucutural_correctionFP

Esta conjectura formaliza o fato que a estrutura obtida ao final da primeira fase do algoritmo satisfaz o predicado `nstructER?` (veja seção 2.1) em relação à sequência "plain"original (na qual nenhuma comparação foi feita ainda). Assim, vamos trabalhar com duas sequências: s_0, uma sequência de fjnodes não vazia com a propriedade `plain_finseqfj`, e $s = seqfjBottleneck(s_0)$, uma sequência de fjnodes não vazia. É importante salientar aqui que o número de vezes que a função `seqfjBottleneck` é aplicada durante esta fase é exatamente o log base 2 do tamanho da sequência original. Denotemos esse número então por n.

A prova faz uso do lema `correctionER`. Este lema define que, para sequências de fjnodes com as mesmas propriedades que s_0 e s (mas sem que s precise ser $seqfjBottleneck(s_0)$), se existir um natural $m \leq n$, tal que a sequência s obtida ao comparar a sequência original s_0 m vezes satisfaça a propriedade `nstructER?` podemos concluir exatamente o que queremos provar (que a sequência s possui a propriedade `nstructER?` após n manipulações). Ou seja, o uso desse lema reduz nossa prova a encontrar este m e mostrar que a sequência s obtida após m manipulações satisfaz `nstructER?`.

Definimos então $m = 0$ e $s = s_0$. Ou seja, precisamos mostrar que s_0 satisfaz a propriedade `nstructER?` em relação à ela mesma após 0 manipulações. Isso é feito usando o lema `plain_finseqfj_nstructER0`, que diz que para qualquer sequência de fjnodes com tamanho maior do que 0, se ela possuir a propriedade `plain_finseqfj`, então também possuirá a propriedade

`nstructER?` em relação à ela mesma após 0 manipulações, que é exatamente o que queremos mostrar.

Lembre-se que podemos usar o lema `plain_finseqfj_nstructER0` pois a sequência s_0 possui as características necessárias para usá-lo.

Resta mostrar que $m \leq n$, ou seja, $0 \leq \log_2(length(s_0))$. De fato isso é verdade sempre que a sequência for não vazia, pela definição de logaritmo. Como uma das propriedades de s_0 é ser não vazia, isso conclui a prova da conjectura.

2.4 Questão 01 - compareSeservesElements

Esta conjectura formaliza que uma sequência de fjnodes s construída comparando todos os elementos de uma sequência s_0 dois a dois como descrito na seção 1 é permutação de s_0, ou seja, comparar os elementos de uma sequência dois a dois construindo elementos mais profundos não altera a sequência.

A prova é por indução no tamanho da sequência s_0. No entanto a separação de casos é bastante complicada. A maneira como foi definida a função `permutation` não facilita uma separação de casos apropriada.

Tentamos fazer a prova aplicando a definição de `permutation` e tentando mostrar que, de fato, cada elemento aparece o mesmo número de vezes nas duas sequências. Isso pode ser feito sem tantos problemas para os casos em que as sequências não tem nenhum elemento, ou quando o primeiro elemento das sequências é o mesmo. No entanto não soubemos resolver a prova para os casos em que um elemento era encontrado em uma posição em alguma das sequências mas não estava na posição correspondente na outra sequência. Por conta disso não conseguimos finalizar a prova.

Sugerimos que uma tentativa poderia ser feita aplicando a definição da função `compare2to2` inicialmente, para uma separação de casos melhor. Não testamos este caminho para a prova, no entanto.

3 Conclusão

Neste trabalho analisamos a formalização da primeira fase do algoritmo Ford Johnson de ordenação. A formalização foi feita no assistente de provas PVS, em arquivos disponibilizados para nós.

Ficou a nosso encargo provar 2 conjunturas relacionadas às operações que o algoritmo executa em uma sequência de naturais durante a primeira fase, construindo estruturas bastante específicas, os fjnodes. Através de dedução de predicados usando a Lógica de Primeira Ordem (LPO) provamos

a correção das conjunturas propostas, mostrando as operações de comparação dos elementos não altera a sequência, e cria elementos fjnodes com as propriedades e profundidade esperadas.

Referências

[1] Langley Formal Methods Program | NASA PVS Library, `https://shemesh.larc.nasa.gov/fm/ftp/larc/PVS-library`

[2] Ford-Johnson sorting algorithm (over type nat), Nikson Fernandes and Mauricio Ayala-Rincon, May, 2018, `http://flaviomoura.mat.br/files/lc1/pjFordJohnson.tgz`