Lógica Computacional 117366
Descrição do Projeto
Formalização de Unicidade de Respostas para Algoritmos de Ordenação

08 de Maio de 2016

Prof. Mauricio Ayala-Rincón & Prof. Flávio L. C. de Moura

O estagiário de docência Thiago Ramos (thiagomendoncaferreiraramos@yahoo.com.br) dará suporte aos alunos no desenvolvimento do projeto. Laboratórios do LINF têm instalado o *software* necessário (PVS 6.0 com as bibliotecas PVS da NASA).

1 Introdução

Algoritmos de ordenação são fundamentais em Ciência da Computação. Neste projeto considerar-se-ão algoritmos de ordenção "corretos" sobre os tipos abstratos de dados `list` e `finite_sequences` como especificado no assistente de demonstração PVS.

O objetivo do presente projeto é introduzir os mecanismos básicos de manuseio de tecnologias de verificação e formalização que utilizam técnicas dedutivas lógicas, como as estudadas na parte teórica da disciplina, para garantir que objetos computacionais são logicamente corretos.

2 Descrição do Projeto

Este projeto aborda questões apresentadas nos arquivos de especificação e prova de algoritmos de ordenação "corretos", onde por correto entendem-se algoritmos que geram uma "permutação" da entrada que está ordenada. Os arquivos estão disponíveis na página da disciplina, especificados na linguagem do assistente de demonstração PVS (pvs.csl.sri.com) executável em plataformas Unix/Linux. Os alunos deverão formalizar propriedades relacionadas com a *unicidade* das respostas computadas por algoritmos de ordenação corretos sobre listas e sobre sequências finitas de naturais.

2.1 Algoritmos de Ordenação corretos

Na biblioteca `sorting` disponível como parte da biblioteca PVS de NASA Langley Research Center, encontram-se especificados e formalizados diversos algoritmos de ordenação sobre elementos de um espaço métrico arbitrário (http://shemesh.larc.nasa.gov/fm/ftp/larc/PVS-library) como *quicksort, heapsort, maxsort, mergesort, bubblesort* etc [BvG99, CLRS09]. A correção desses algoritmos está expressa como lemas que indicam que a saída computada é uma permutação da entrada, que adicionalmente está ordenada.

Algumas funções utilizadas na especificação destes algoritmos estão especificadas para listas de naturais e sequências finitas nos arquivos `sorting.pvs` e `sorting_seq.pvs` (arquivos de provas `sorting.prf` e `sorting_seq.prf`), como a noção de `occurrences`, que especifica quantas vezes determinado natural está presente em uma lista, dada por:

```
occurrences(l:list[nat])(x:nat): RECURSIVE nat =
IF  null?(l) THEN 0
ELSIF car(l) = x THEN 1 + occurrences(cdr(l))(x)
```

```
        ELSE occurrences(cdr(l))(x)
        ENDIF
        MEASURE length(l)
```

Também temos para sequências finitas uma função semelhante, dada por:

```
occurrences(s : finite_sequence[nat])(x:nat): RECURSIVE nat =
IF  length(s) = 0 THEN 0
ELSIF
     s(0) = x THEN 1 + occurrences(s^(1,length(s)-1))(x)
ELSE
     occurrences(s^(1,length(s)-1))(x)
ENDIF
MEASURE length(s)
```

A noção de *permutação* está especificada como um predicado que confere igual número de ocorrências de cada natural, seja em listas ou em sequências finitas. A noção de *ordenado* está especificada como predicados que conferem que cada par de naturais em posições contíguas, seja da lista ou da sequência finita, preserva a relação "$\leq$". Para listas, temos os seguintes predicados:

```
permutations(l1,l2:list[nat]):bool =
         FORALL (x:nat): occurrences(l1)(x) = occurrences(l2)(x)

is_sorted?(l:list[nat]): bool =
         FORALL (k : below[l`length]) :
             0 <= k AND k <= length(l) - 2 => nth(l, k) <= nth(l, k+1)
```

3 Questões

Deverão ser provados os resultados a seguir, conforme especificação dada no arquivo `unicity.pvs`. A primeira questão do projeto consiste em demonstrar que, *em listas ordenadas, o primeiro elemento é o mínimo*, especificado como a conjectura abaixo. Para provar este resultado será necessário o uso de indução na estrutura da lista.

Questão 01

```
min_is_sorted : CONJECTURE  FORALL (l:list[nat], k : below[length(l)]) :
       is_sorted?(l) => null?(l) OR nth(l,0) <= nth(l,k)
```

Esse resultado é central para obter a unicidade para o caso de listas, que são permutações ordenadas, e junto com propriedades de transformações entre listas e sequências finitas e vice-versa, obtém-se também a unicidade para o caso de sequências finitas, a seguir.

```
unicity_sorted_lists :THEOREM  FORALL (l1,l2:list[nat]) :
            permutations(l1,l2) AND is_sorted?(l1) AND is_sorted?(l2) =>
                                              l1 = l2

unicity_sorted_seqs : THEOREM FORALL (s1, s2 : finite_sequence[nat]) :
            permutations(s1, s2) AND sorted(s1) AND sorted(s2) =>
                                              s1 = s2
```

Os algoritmos *heapsort* e *maxsort* trabalham sobre sequências, enquanto que *quicksort* e *bubblesort*, sobre listas. A unicidade das respostas computadas por esses algoritmos deve ser demonstrada utilizando os teoremas principais de unicidade:
Questão 2 e **Questão 3**

```
func_eqQB : CONJECTURE  FORALL (l:list[nat]) : quick_sort(l) = bubblesort(l)

func_eqMH : CONJECTURE  FORALL (s:finite_sequence[nat]) : maxsort(s) = heapsort(s)
```

Finalmente, deve ser demonstrado, utilizando também as conversões entre listas e sequências finitas que os algoritmos *quicksort* e *maxsort*, que trabalham sobre estruturas de dados diferentes, computam as mesmas respostas. **Questão 04**

```
func_eqQM : CONJECTURE FORALL (l:list[nat], s:finite_sequence[nat]) :
  list2finseq(l) = s IMPLIES quick_sort(l) = maxsort(s)
```

As três últimas questões não requerem aplicação de indução, mas de resultados previamente formalizados nos arquivos `sorting` e `sorting_seq`.

4 Etapas do desenvolvimento do projeto

Os alunos deverão definir os grupos de trabalho limitados a **três** membros até o dia 17 de Maio. Exceto pelo dia da segunda prova, 29 de Junho de 2016, as aulas serão realizadas no LINF a partir do dia 04 de Maio.

O projeto será dividido em duas etapas como segue:

- A primeira etapa do projeto é a de Verificação das Formalizações. Os grupos deverão ter prontas as suas formalizações na linguagem do assistente de demonstração PVS e enviar via e-mail ao estagiário com cópia para o professor os arquivos de especificação e de provas desenvolvidos (`unicity.pvs` e `unicity.prf`) até às 08h da manhã do dia **13.06.2016**. Na mesma semana dias **13 e 15.06.2016**, durante o horário de aula, realizar-se-á a verificação do trabalho para a qual os grupos deverão, em acordo com o estagiário e professor, determinar um horário (de 30 min) no qual todos membros do grupo deverão comparecer.
  Avaliação (peso 6.0):
  - Um dos membros, selecionado por sorteio, explicará os detalhes da formalização em, no máximo, 10 minutos.
  - Os quatro membros do grupo poderão complementar a explicação inicial em, no máximo, 5 minutos.
  - A formalização será testada nos 15 minutos seguintes.
- A segunda etapa do projeto consiste da apresentação dos resultados finais e conclusões do estudo do problema.
  Avaliação (peso 4.0): Cada grupo de trabalho deverá entregar um Relatório Final inédito, editado em LaTeX, limitado a 8 páginas (12 pts, A4, espaçamento simples) do projeto até o dia **20.06.2016** com o seguinte conteúdo:
  - Introdução e contextualização do problema.

- Explicação da soluções.
- Especificação do problema e explicação do método de solução.
- Descrição da formalização.
- Conclusões.
- Referências.

Referências

[BvG99] S. Baase and A. van Gelder. *Computer Algorithms — Introduction to Design and Analysis*. Addison-Wesley, 1999.

[CLRS09] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Electrical Engineering and Computer Science Series. MIT press, third edition, 2009.