Lógica Computacional 117366
Descrição do Projeto
Formalização de um Algoritmo para Ordenação por Inserção Binária

11 de outubro de 2017

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

Observação: Os laboratórios do LINF têm instalado o *software* necessário para o desenvolvimento do projeto (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 sobre o tipo abstrato de dados `finseq` como especificado no assistente de demonstração PVS.

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

2 Descrição do Projeto

Com base nas *teorias* `sorting_seq` e `seq_extras` especificadas na linguagem do assistente de demonstração PVS (`pvs.csl.sri.com`, executável em plataformas Unix/Linux) e disponíveis na página da disciplina e na biblioteca `nasalib`, respectivamente, os alunos deverão formalizar propriedades de uma especificaçõao para ordenação por inserção binária sobre sequências finitas. O arquivo com as questões é denominado `binsertionsort`.

2.1 Busca em sequências de naturais

O objetivo é demonstrar, formalmente, que a especificação de ordenação por inserção binária abaixo é correta:

```
binsertionsort(v) : {w : finite_sequence[nat] | length(w) = length(v) } =
IF length(v) <= 1 THEN v
ELSE binsertionsort_aux(v)(1)
ENDIF
```

Nessa especificação, a função `binsertionsort` recebe uma sequência finita como argumento e retorna uma sequência finita de igual comprimento. O mecanismo de inserção binária utilizado é especificado via a função recursiva `binsertionsort_aux` abaixo:

```
binsertionsort_aux((v | length(v)>=2))(j : below[length(v)]) :
RECURSIVE {w : finite_sequence[nat] | length(w) = length(v) } =
IF j = 0 THEN v
ELSE LET l = bin_search(v)(v(j))(0, j - 1),
         w = insert?(v(j), delete(v, j), l) IN
```

```
     IF j = length(v) - 1 THEN w
     ELSE binsertionsort_aux(w)(j+1)
     ENDIF
ENDIF
MEASURE length(v) - j
```

A função `binsertionsort_aux` recebe como argumentos uma sequência finita `v` com pelo menos 2 elementos e um índice `j` e retorna a versão ordenada desta sequência. Recursivamente, a função reconstrói a sequência `v` dada como argumento, inserindo cada elemento de `v` a partir do segundo, i.e., `j`= 1, na posição correta, computada como `bin_search(v)(v(j))(0,j)`. A função `bin_search` retorna, em geral, a posição correta de um natural `k` para a subsequência de `v` entre as posições `i` e `j`, como a seguir:

```
bin_search(v)(k)(i : below[length(v)], (j :below[length(v)] | i <=j)) :
RECURSIVE upto[j + 1] =
IF i=j THEN
   IF k <= v(i)  THEN i ELSE i + 1 ENDIF
ELSE LET l = floor((i+j)/2) IN
     IF k = v(l) THEN l
     ELSIF  k < v(l) THEN IF i=l THEN i ELSE bin_search(v)(k)(i, l-1) ENDIF
     ELSE bin_search(v)(k)(l+1, j)
     ENDIF
ENDIF
MEASURE j - i
```

3 Questões

Concretamente para a função `binsertionsort` deve ser formalizado o seguinte lema:

```
binsertionsort_works : LEMMA
FORALL (v):
       sorted(binsertionsort(v)) AND permutations(v,binsertionsort(v))
```

A solução deste problema será dividida em 4 questões auxiliares que abordam propriedades referentes ao mecanismo de inserção binária utilizado no problema de ordenação.

Questão 01 *As posições retornadas pela função* `bin_search` *estão dentro do intervalo* `(i,j+1)`*, para os parâmetros* `i` *e* `j` *dados como argumento.*

```
b_search_bound: LEMMA FORALL(v)(k)(i : below[length(v)],
(j :below[length(v)] | i <=j)) :
 bin_search(v)(k)(i,j) >= i AND bin_search(v)(k)(i,j) <= j+1
```

Questão 02 *A função* `binsertionsort_aux` *gera uma sequência que é uma permutação da sequência de entrada.*

```
bs_aux_perm: LEMMA  FORALL((v | length(v)>=2), (j : below[length(v)])) :
 permutations(v, binsertionsort_aux(v)(j))
```

Questão 03a *A função* `binsertionsort` *gera uma sequência que é uma permutação da sequência de entrada.*

```
binsertion_permutation : LEMMA FORALL (v): permutations(v, binsertionsort(v))
```

Questão 03b *A função* `binsertionsort` *gera uma sequência ordenada.*

```
binsertion_sorts : LEMMA  FORALL (v): sorted(binsertionsort(v))
```

4 Etapas do desenvolvimento do projeto

Os alunos deverão definir grupos de trabalho limitados a **quatro** membros até o dia 11 de outubro. As aulas serão realizadas no LINF a partir do dia 11 de outubro para a turma B. A turma A continuará com aulas no LINF somente nas quartas-feiras.

O projeto será dividido em duas etapas como segue:

- Verificação das Formalizações. Os grupos deverão ter prontas as suas formalizações na linguagem do assistente de demonstração PVS e enviar via e-mail para o professor os arquivos de especificação e de provas desenvolvidos (`binsertionsort.pvs` e `binsertionsort.prf`) até o dia **12.11.2017**. Na semana de **13-15.11.2017**, durante os dias de aula, realizar-se-á a verificação do trabalho para a qual os grupos deverão, em acordo com o monitor estagiário de docência e professor, determinar um horário (de 30 minutos) no qual todos membros do grupo deverão comparecer.
  Avaliação (peso 6.0):
  - Um dos membros, selecionado por sorteio, explicará os detalhes da formalização em máximo 10 minutos.
  - Os quatro membros do grupo poderão complementar a explicação inicial em máximo 10 minutos.
  - A formalização será testada nos seguintes 10 minutos.
- Entrega do Relatório Final.
  Avaliação (peso 4.0): Cada grupo de trabalho devera entregar um Relatório Final inédito, editado em LaTeX, limitado a oito páginas (12 pts, A4, espaçamento simples) do projeto até o dia **22.11.2017** 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

[ARdM17] M. Ayala-Rincón and F.L.C. de Moura. *Applied Logic for Computer Scientists - computational deduction and formal proofs*. UTiCS, Springer, 2017.

[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.