Guião para aula laboratorial de Verificação Formal (2018/19)

Frama-C (2)

Esta aula é dedicada à verificação dedutiva de programas, utilizando o plugin WP da plataforma Frama-C. Será também dada alguma atenção à modelação lógica das funções C e ao poder expressivo da linguagem de especificação ACSL.

Deverá ter instalado o Frama-C, o plugin WP, assim como alguns SMT solvers.

O website do Frama-C disponibiliza toda a documentação, assim como alguns tutoriais.

1 Especificações lógicas

A linguagem das expressões lógicas usadas nas anotações ACSL pode ser estendida por declarações de novos tipos lógicos e novas constantes, funções lógicas e predicados. Estas declarações seguem a configuração clássica das especificações algébricas.

Para além da declaração de novos predicados e funções lógicas, é possível definir lemas, predicados indutivos, fazer definições axiomáticas e definições recursivas. É possível também definir funções e predicados *híbridos*. Isto é, funções lógicas e predicados que tanto podem receber argumentos um tipo C (puro), como um tipo lógico. Tal predicado (ou função) pode ser definido com a mesma sintaxe de antes (ou axiomatizado). No entanto, essas definições geralmente dependem de um ou mais pontos do programa, porque são dependentes do estado da memória (devido à presença de expressões que acedem à memória). Neste caso é obrigatório adicionar após o identificador declarado um conjunto de *labels*, entre chavetas.

Nos predicados (ou funções) híbridas as expressões devem então ser incluídas numa construção `\at` para se referir a uma determinada *label* (que denota um ponto no programa). No entanto, para facilitar a leitura de tais expressões lógicas, é possível omitir a *label* sempre que houver apenas uma etiqueta no contexto.

2 Exemplos para explorar

A secção *"Program Verification"* dos slides *"Dedictive Program Verification with Frama-C"*, apresentados na aula, tem uma serie de exemplos e vários desafios para resolver. Esses exemplos são acompanhados de ficheiros C disponíveis no wiki da disciplina.

Siga as instruções indicadas nos slides, analise estes ficheiros com o plugin WP, e acrescente/altere as anotações ACSL de forma a completar as provas.

3 Exercícios

1. Considere a seguinte função que ordena um array de inteiros por ordem crescente.

```
void maxSort (int *a, int size) {
  int i, j;

  for (i=size -1; i>0; i--) {
    j = maxarray(a,i+1);
    swap(a,i,j);
  }
}
```

   (a) Escreva um contrato de *safety* para esta função e prove que a função garante esse contrato.

   (b) Melhore agora o contrato que escreveu para garantir que o array produzido pela função está ordenado por ordem crescente. Escreva os invariantes de ciclo para provar o novo contrato.

   (c) Complete o contrato de forma a garantir que a função implementa um algoritmo de ordenação. Em seguida, conclua a prova.

2. A seguinte função deve contar as ocorrências de x no array a, de tamanho n.

```
int numOccur (int *a, int n, int x);
```

   (a) Declare a função lógica `count` que determina o número de ocorrências de um valor num array, e apresente uma definição axiomatica para essa função.

   (b) Escreva um contrato completo para esta função.

   (c) Defina a função e prove a sua correcção face ao contrato que escreveu.

3. A seguinte função deve inverter o array a, de tamanho n.

```
void reverse (int a[], int n);
```

   (a) Escreva um contrato completo para esta função.

   (b) Defina a função e prove a sua correcção face ao contrato que escreveu.