Teste Frama-C + CBMC (2013/2014)

INTRODUÇÃO

De acordo com a noção de igualdade de programas induzida pela semântica operacional, dois programas são iguais se para todos os estados iniciais a execução de ambos resulta no mesmo estado final (ou então nenhum dos dois termina).

Uma técnica utilizada para a prova de igualdade com recurso a ferramentas de verificação baseia-se na construção de um programa que corresponde à *composição* dos dois programas cuja equivalência se quer mostrar, renomeando-se todas as variáveis de *um deles*.

Considere-se o seguinte exemplo de um ciclo:

```
for (k=i ; k<=j; k++) {
  b += k;
  a *= k;
}
```

que pode ser "refactored" para o seguinte código equivalente, uma vez que não existe qualquer interacção entre as variáveis a e b:

```
for (k=i; k<=j; k++) b += k;
for (k=i; k<=j; k++) a *= k;
```

Para a utilizção da técnica de composição referida, pode-se construir o programa seguinte (a ordem pela qual se compõe os programas é irrelevante):

```
for (k=i; k<=j; k++) b += k;
for (k=i; k<=j; k++) a *= k;

for (ks=is ; ks<=js; ks++) {
  bs += ks;
  as *= ks;
}
```

Observe-se que uma vez que os espaços de nomes de ambos os programas são disjuntos, a execução sequencial dos dois programas no programa composto permite raciocinar sobre execuções independentes dos dois programas, mas permitindo relacionar os valores iniciais e finais das variáveis de ambos. Informalmente, a especificação a considerar para este programa dirá que se os valores iniciais das variáveis de ambos os programas forem iguais, então os valores finais dessas variáveis deverão também ser iguais.

Por exemplo em ACSL este código teria que ser colocado dentro de uma função cujo contrato especificaria a igualdade em causa:

```
int a, b, as, bs;
int i, j, k, is, js, ks;

/*@ requires i==is && j==js && k==ks && a==as && b==bs;
  @ ensures  i==is && j==js && k==ks && a==as && b==bs;
  @*/
void main_verif () {

  for (k=i; k<=j; k++) b += k;
  for (k=i; k<=j; k++) a *= k;

  for (ks=is ; ks<=js; ks++) {
    bs += ks;
    as *= ks;
  }
}
```

PARTE 1

Considere os dois 'snippets' de código seguintes:

```
  for (k=i ; k<=j; k++)
    b++;
```

e

```
  for (k=j ; k>=i; k--)
    b++;
  k = j+1;
```

A igualdade é neste caso trivial: os dois ciclos efectuam as mesmas operações (cuja ordem é irrelevante) por ordem inversa um do outro.

A. Escreva uma função com anotações em ACSL que lhe permita verificar que os dois programas são iguais, utilizando a técnica de composição descrita. Acrescente à função outras anotações (pré-condições adicionais, invariantes e variantes de ciclo) que possam ser necessárias para efectuar com sucesso esta prova, e apresente detalhadamente os resultados obtidos com o plugin Jessie.

B. Relativamente à verificação de segurança deste programa composto, é natural que a presença de operações aritméticas gere VCs inválidas relacionadas com a ocorrência de 'overflow'. Discuta este facto e coloque novas anotações no programa que permitam garantir a ausência de 'overflow', às custas de se limitar as gamas de valores de entradas para as variáveis em jogo.

PARTE 2

Considere agora os dois 'snippets' seguintes:

```
for (k=i ; k<=j; k++)
  b++;
for (k=i ; k<=j; k++)
  a*=2;
```

e

```
for (k=i ; k<=j; k++) {
  b++;
  a*=2;
```

Este caso é mais parecido com o descrito na introdução (as duas operações do ciclo são independentes), mas mais simples.

Repita as tarefas A e B de assessment-1 (reproduzidas em baixo).

A. Escreva uma função com anotações em ACSL que lhe permita verificar que os dois programas são iguais, utilizando a técnica de composição descrita. Acrescente à função outras anotações (pré-condições adicionais, invariantes e variantes de ciclo) que possam ser necessárias para efectuar com sucesso esta prova, e apresente detalhadamente os resultados obtidos com o plugin Jessie.

B. Relativamente à verificação de segurança deste programa composto, é natural que a presença de operações aritméticas gere VCs inválidas relacionadas com a ocorrência de 'overflow'. Discuta este facto e coloque novas anotações no programa que permitam garantir a ausência de 'overflow', às custas de se limitar as gamas de valores de entradas para as variáveis em jogo.

PARTE 3

Esta tarefa (bem como a próxima) difere das anteriores no seguinte:
- os programas passam a conter arrays;
- as técnicas de verificação a utilizar são automáticas em vez de dedutivas.

No contexto contendo:

```
#define MAX ...
extern int u[MAX];
```

Considere os dois 'snippets' seguintes:

```
for (i=0; i<MAX; i++)
  u[i] = u[i]*2 + x;
```

e

```
for (i=0; i<MAX; i++)
  u[i] *= 2;
for (i=0; i<MAX; i++)
  u[i] += x;
```

A equivalência corresponde agora a uma decomposição das operações aritméticas presentes no primeiro ciclo.

O plugin de análise de valor do Frama-C não lida com asserts, mas calcula aproximações dos valores finais das variáveis, nomeadamente dos elementos dos arrays, e é possível compará-los. Para facilitar esta tarefa pode-se utilizar uma variável Booleana, incluindo na função o seguinte código depois do programa composto (k é uma variável do programa de verificação que não deve coincidir com nenhuma variável dos programas cuja equivalência se deseja provar):

```
int verif = 1;

// Programa Composto

for (k=0; k<MAX; k++) {
  verif = verif && (u[k] == us[k]);
```

Bastando depois que verif tenha um valor final de 1 para garantir que os arrays de saída são iguais.

A. Começaremos por considerar variáveis e arrays de entrada concretos (e iguais para os dois programas). Acrescente o seguinte código no início da função de verificação:

```
x = xs = 1;
for (k=0; k<MAX; k++) {
  u[k] = us[k] = k;
}
```

Aplique o plugin para mostrar que os dois programas se comportam da mesma forma com este array inicial concreto. Comente todos os resultados e construa uma tabela do tempo necessário para a análise, para valores crescentes de MAX até um tempo máximo de alguns minutos (Em Unix poderá medir este tempo com o comando "$ time -p"). Comente os resultados.

B. Naturalmente a alínea anterior não constitui uma verificação da igualdade, e poderia ser substituída por uma simples execução do programa composto. Tente agora utilizar o plugin para a prova mais genérica, retirando o ciclo de inicialização da alínea anterior. Tenha em atenção que os arrays devem ser declarados como 'extern', caso contrário os arrays serão considerados inicializados com [0].

Sugestões:
1. Efectue a inicialização dos valores do array com o valor de uma variável declarada externamente, mas sem valor atribuído.
2. Utilize uma pré-condição, semelhante à que utilizaria com um plugin de verificação dedutiva.

Comente os resultados obtidos e interprete-os, possivelmente com a ajuda do manual do plugin de análise de valor.

PARTE 4

Considere ainda os dois 'snippets' de assessment-3:

```
#define MAX ...
extern int u[MAX];

 for (i=0; i<MAX; i++)
   u[i] = u[i]*2 + x;
```

e

```
 for (i=0; i<MAX; i++)
   u[i] *= 2;
 for (i=0; i<MAX; i++)
   u[i] += x;
```

Nesta tarefa pretende-se agora utilizar a ferramenta CBMC para averificação da igualdade.

A. Comecemos novamente por considerar variáveis e arrays de entrada concretos (e iguais para os dois programas). Acrescente, tal como em assessment-3, o seguinte código no início da função de verificação:

```
 x = xs = 1;
 for (k=0; k<MAX; k++) {
   u[k] = us[k] = k;
 }
```

Acrescente também depois do programa composto o código necessário (utilizando asserts) para garantir que os valores das variáveis (incluindo os arrays, para o que deverá utilizar um ciclo) são iguais à saída.

Comente todos os resultados obtidos com o CBMC, e construa uma tabela do tempo necessário para a análise, para valores crescentes de MAX até um tempo máximo de alguns minutos (Em Unix poderá medir este tempo com o comando "$ time -p"). Comente os resultados e compare-os com os que obteve com a análise de valor do Frama-C.

B. Naturalmente a alínea anterior não constitui uma verificação da igualdade, e poderia ser substituída por uma simples execução do programa composto. Em CBMC é possível a verificação automática da igualdade.

Para isso substitua agora o código de inicialização por outro que estabeleça, utilizando assumes, a igualdade das variáveis de entrada dos dois programas (incluindo os arrays, para o que deverá utilizar um ciclo).

Muito importante: todas as variáveis e arrays deverão antes de mais ser inicializdas com a função nondet_int, que deverá declarar desta forma:

```
extern int nondet_int();
```

De outro modo, todas as variáveis e arrays serão considerados inicializados com valor 0, e novamente não teremos uma prova de igualdade. Teste a robustez da sua verificação alterando ligeiramente partes do código por forma a que a igualdade deixe de ser constatada, e relate este processo.

Comente os resultados obtidos e interprete-os, indicado o tamanho máximo dos arrays para o qual a verificação pode ser concluída com sucesso num tempo máximo de 5 minutos.

C. Finalmente, considere que MAX não é uma constante, mas sim uma variável 'extern', sem valor atribuído. Como deve proceder para verificar que a acção dos dois programas sobre as 100 primeiras posições dos arrays é a mesma?