Interpretação abstrata
O *plugin* value de `Frama-C`

Richard Bonichon Vítor Almeida

20141122

Introdução

Esta lista propõe exercícios para a aplicação prática do ensino em interpretação abstrata. Para isto, usaremos o Frama-c com o plugin *Value Analysis*.

A documentação está disponível no site de Frama-C usando o link:

`http://frama-c.com/value.html`

Para realizar os exercícios abaixo, baixe os arquivos de código-fonte que encontram-se disponíveis no site.

1 Introdução ao *Value Analysis*

Para este exercício utilizaremos o código do listing 1.

```c
int x, y, *z, *idx, a[101], t;
float f;

int
main (int c)
{
        if (0 <= c && c <= 9) z = &x;
        else z = &y;

        if (c == 5) x = 1;
        else x = 2;

        switch (c) {
        case 0:  y = 3;   break;
        case 1:  y = 42;  break;
        case 2:  y = 36;  break;
        case 3:  y = 25;  break;
        case 4:  y = 10;  break;
        case 5:  y = 100; break;
        case 6:  y = 18;  break;
        case 7:  y = 75;  break;
        case 8:  y = 83;  break;
        case 9:  y = 98;  break;
        default: y = 2;   break;
        }

        idx = &a[y];

        if (c) f = 1.5;
        else f = 0x1p-2;

        return y + *z;
}
```

Listing 1: Exercício 1

1. Execute o seguinte comando:

```
% frama-c -val exercicio1.c
```

Observe os resultados fornecidos para as diferentes variáveis do programa

2. Execute novamente o comando com interface gráfica:

```
% frama-c-gui -val exercicio1.c
```

3. Insira a instrução `Frama_C_show_each_res(c,y,z);` após o *switch* e relance a análise de valor. O que pode ser observado?
`Frama_C_show_each_xxx(args)` requisita e mostra os valores de `args` toda vez que o analisador passa por esta instrução

4. Reexecute os comandos com a opção `-slevel <n>`, onde `<n>` é um inteiro não-negativo. Tente com diferentes valores. O resultado está mais ou menos preciso? O que mudou?

5. A opção `-val-ilevel <n>` redefine a quantidade máxima do conjunto de possíveis valores distintos para cada variável. Execute-o só e conjuntamente com `-slevel`. Qual a precisão do resultado obtido?

2 Alarmes

Para este exercício, consideraremos o programa do listing 2:

```
int
main(int c)
{
        int x, y, z[5];
        if (c) { x = c; } else { y = 4; }
        if (c) { y = x; } else { x = y; }
        z[0] = x;
        z[1] = y;
        return x / y;
}
```

Listing 2: Exercício 2

1. Execute `Frama-C` com o plugin *value analysis*. Quais os avisos mostrados?

2. Cada *warning* é um alarme de que há potencialmente um problema no código. Para cada alarme, verifique se o mesmo é verdadeiro ou não. Caso necessário, corrija o código

3. Com ajuda de `-slevel` e de eventuais asserções, remova os falsos alarmes

3 Definição de um contexto inicial

Nos exercícios anteriores, o parâmetro `c` era considerado um inteiro com qualquer valor possível (INT_MIN $\leq c \leq$ INT_MAX). Podemos restringir os valores possíveis de entrada para testar a função em um contexto mais limitado.

Para isso, consideremos o programa do listing 3:

```
int
f(int x)
{
        int S = 0;
        for (int i = 0; i <= x; i++) {
                if (S >= i) S -= i; else S += i;
        }
        return S;
}

int
main()
{
        int x = 10;
        int res = f(x);
        return res;
}
```

Listing 3: Exercício 3

1. Qual o valor de `res` no fim da execução de `main`?

2. Iremos agora saber quais os valores possíveis de `res` quando `x` possui valores entre 10 e 100 (inclusos). Para isto, utilizaremos as funções

declaradas no arquivo `builtin.h` situado no diretório de `Frama-C`. Podemos obter a localização deste diretório através do comando:

```
% frama-c -print-share-path
```

Modifique a função `main` para que ela chame a função `f` com um argumento na faixa de valores de 10 a 100 (inclusos). Adicione no comando a opção `-cpp-extra-args="-I$(frama-c -print-share-path)"` para indicar ao preprocessador as funções disponíveis no diretório do `Frama-C`.

Qual a faixa de valores possíveis para `res` após esta modificação?

3. Uma outra possibilidade de análise consiste em lançar `Frama-C` diretamente de `f` com a opção `-main f` e uma anotação ACSL restringindo os argumentos. A sua tarefa é de criar uma anotação que restrinja os valores de x para o mesmo intervalo da questão anterior e analisar a função `f` diretamente. Os possíveis valores para `res` são os mesmos da questão anterior?

4 PPCM

Consideraremos a função PPCM do listing 4:

```
int
ppcm(int x, int y)
{
        int a = x, b = y;
        while (b != 0) {
                int tmp = mod(a, b);
                a = b;
                b = tmp;
        }
        return x * y / a;
}

void
test2()
{
        int P = ppcm(49,28);
}
```

Listing 4: PPCM

1. Estude o valor encontrado por `P` após a execução de `test2`
2. Dentro de um contexto genérico, a multiplicação no fim de `ppcm` pode levar a um *overflow*. Limitemos o intervalo de `x` e `y` entre -10000 e 10000. Utilizando a função `interval` mostrada abaixo, forneça uma função `main` que chama `ppcm` dentro deste contexto

   ```
   int interval(int a, int b);
   ```

3. É possível haver uma divisão por 0 na execução de `ppcm` com qualquer parâmetro? Se for o caso, corrija a função.

5 Advinhar um nome

Para este problema consideraremos a função `devine` do listing 5.

```c
#include "builtin.h"

int
devine(int secret, int max)
{
        int tentative = max / 2, inf = 0, sup = max, nb = 1;
        while (tentative != secret) {
                nb++;
                if (tentative < secret) inf = tentative + 1;
                if (tentative > secret) sup = tentative - 1;
                tentative = (inf + sup) / 2;
        }
        return nb;
}
```

Listing 5: Devine

1. A função `devine` só poderá terminar se `secret` estiver em um intervalo particular. Insira uma anotação ACSL que delimite este intervalo.
2. Escreva uma função `main` que chama `devine` com `secret` entre 0 e 100. Experimente `max` com diferentes valores
3. Quais valores `devine` pode retornar?
4. Verifique a saída da função com `secret` definido entre `INT_MAX-5000` e `INT_MAX`.

6 Euler

O problema do listing 6 é uma implementação proposta para se achar todos os inteiros inferiores a 1000 que são múltiplos de 3 e múltiplos de 5.

```c
int total = 0;

void
add(int max, int x)
{
        total += x;
}

void
sum(int max)
{
        int i = 1;
        while ( i <= max / 3) {
                add(max, 3 * i);
                if (i % 3 != 0) add(max, 5 * i);
                i++;
        }
}

int
main()
{
        sum(1000);
        return total;
}
```

Listing 6: Euler

1. Utilize `Frama-C` para descobrir o valor de `total` no fim da função `main`
2. Escreva uma anotação ACSL que verifica se o argumento `x` em `add` pode ser adicionado à soma. Se a função não respeitar o contrato, corrija o código.
3. Responda a mesma questão para o caso de `sum` puder ser um valor qualquer entre 1 e 1000