

DIM0436

17. Frama-C

20140930

# Sumário

1 Frama-C

2 Mais sobre WP

1 Frama-C

2 Mais sobre WP

# Um quadro de cooperação entre análises

- Frama-C é um conjunto de ferramentas para analisar software escrito em C
- Resultados calculados por uma análise podem ser reusados numa outra
- Open-source
- Análises estáticas corretas
- <http://frama-c.com>

## Usos

- Compreensão de código-fonte
- Prova formal para software crítico

# Value

- Análise por interpretação abstrata de programas sequenciais
- Cálculo dos domínios de variação das variáveis do programa
- Avisos de erros no tempo de execução
- Análise semântica

```
1  int main(void)
2  { int i ;
3    int *p ;
4    int *tmp ;
5
6    {p = T;
7      i = 0;
8      while (i < 5) {S += i;
9        {{tmp = p;
10         p ++;}}
11
12         *tmp = S;}
13         i ++;}}
14
15     return (S);}
16
17 }
18
```

Informations	Messages
--------------	----------

Function: main  
Statement: 8  
Variable S has type "int".  
It is a global variable.  
It is referenced and its address is not taken.  
Before statement:  
 $S \in \{0; 1; 3; 6; \}$   
At next statement:  
 $S \in \{0; 1; 3; 6; 10; \}$

# WP / Jessie

- Cálculo de pré-condição mais fraca
- WP é uma versão mais integrada de Jessie
- Uso da plataforma Why para descarregar obrigações de prova

The screenshot shows the gWhy verification conditions viewer interface. The window title is "gWhy: a verification conditions viewer". The interface is divided into two main sections: a table of proof obligations on the left and a code editor on the right.

Proof obligations	Alt-Ergo 0.9	Statistics
Function max Default behavior	✓	20/20
Function max Normal behavior 'empty'	✓	2/2
1. postcondition	●	
2. postcondition	●	
Function max Normal behavior 'not_empty'	✓	8/8
1. postcondition	●	
2. postcondition	●	
3. postcondition	●	
4. postcondition	●	
5. postcondition	●	
6. postcondition	●	
7. postcondition	●	
8. postcondition	●	
Function max Safety	✓	12/12
1. pointer dereferencing	●	
2. pointer dereferencing	●	

```
max_ensures_default_po_1
a: int_P pointer
n_0: int32
int_P_a_2_alloc_table: int_P alloc_table
H1: is_valid_int_range(a, n_0, int_P_a_2_alloc_table)

not (integer_of_int32(n_0) = 0 and 0 < integer_of_int32
(n_0))

...
(check disjoint_behaviors : (! ((n == 0) && (0 <
n)))));
(check complete_behaviors : ((0 < n) || (n == 0)));
```

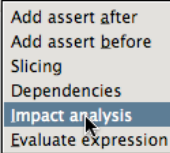
# Metrics

- Cálculo de métricas de software
  - ▶ LOC
  - ▶ Complexidade ciclométrica
  - ▶ Complexidade de Halstead
- Análise sintática
- Cálculo de penetração da análise Value

# Impact

- Cálculo das instruções impactadas por uma atribuição
- Baseado no plugin Value

```
1  int S = 0;
2  int T[5];
3  int main(void)
4  { int i;
5    int *p;
6    int *tmp;
7
8    {p = T;
9      i = 0;
10     while (i < 5) {S += i;
11       {{tmp = p;
12         p++;}}
13
14         *tmp = S;}}
15     i++;}
16
17     return (S);}
18
19 }
20
```





# Slicing

- Cálculo de um novo programa a partir de um sub-conjunto de instruções do programa analisado.
- As instruções são selecionadas de acordo com um critério de *slicing*
- O novo programa fica **compilável**
- Baseado no plugin Value

```
void f() {return;}  
void g() {return;}  
  
void main(void)  
{  
    f();  
    g();  
    return;  
}
```

→  
-slice-calls f

```
void f_slice_1(void)  
{  
    return;  
}  
  
void main(void)  
{  
    f_slice_1();  
    return;  
}
```

# Outras análises

## Aorai

- Análise de propriedades temporais do programa
- Baseado no WP

## Mthread

- Análise estática de código **concorrente**
- Baseado no plugin Value

## Regras de codificação

- Indentação
- Zonas de declaração das variáveis
- Nomes de variáveis
- Regras MISRA

1 Frama-C

2 Mais sobre WP

# Vetores

```
/*@ requires n > 0;  
requires |valid(p + (0..n-1));  
assigns |nothing;  
ensures |forall int i; 0 <= i <= n - 1 ==> |result >= p[i];  
ensures |exists int e; 0 <= e <= n - 1 && |result == p[e];  
*/  
int max_seq(int* p, int n);
```

# Predicados

## Suponha

```
typedef struct _list { int element; struct _list* next; } list;
```

## Acessibilidade

```
/*@  
inductive reachable{L} (list* root, list* node) {  
  case root_reachable{L}:  
    \forall list* root; reachable(root,root);  
  case next_reachable{L}:  
    \forall list* root, *node;  
    \valid(root) ==> reachable(root->next, node) ==>  
    reachable(root,node);  
}  
*/
```

## Finitude

```
/*@ predicate finite{L}(list* root) = reachable(root, \null); */
```

# Funções

## Tamanho

```
/*@ axiomatic Length {  
  logic integer length{L}(list* l);  
  
  axiom length_nil{L}: length(\null) == 0;  
  
  axiom length_cons{L}:  
    \forall list* l, integer n;  
      finite(l) && \valid(l) ==>  
        length(l) == length(l->next) + 1;  
}  
*/
```

# max\_list

## Especificação

```
/*@  
  requires |valid(root);  
  
  assigns |nothing;  
  terminates finite(root);  
  
  ensures |forall list* l;  
    |valid(l) && reachable(root,l) ==> |result >= l->element;  
  ensures |exists list* l;  
    |valid(l) && reachable(root,l) && |result == l->element;  
*/  
int max_list(list* root);
```

# Código *ghost*

```
int max_seq(int* p, int n) {
    int res = *p;
    //@ ghost int e = 0;
    /* loop invariant \forall integer j;
       0 <= j < i ==> res >= \at(p[j],Pre);
       loop invariant
       \valid(\at(p,Pre)+e) \&\& \at(p,Pre)[e] == res;
       loop invariant 0<=i<=n;
       loop invariant p==\at(p,Pre)+i;
       loop invariant 0<=e<n;
    */
    for(int i = 0; i < n; i++) {
        if (res < *p) {
            res = *p;
            //@ ghost e = i;
        }
        p++;
    }
    return res;
}
```

- Código *ghost* não deve mudar o comportamento original do código C
- Prestem atenção nas interferências entre anotações e código



# Resumo

① Frama-C

② Mais sobre WP

## Referências

<http://frama-c.com>

Perguntas ?



<http://dimap.ufrn.br/~richard/dim0436>