

Sviluppo di Software Sicuro - S³

Condizioni di verifica - III

Corso di Laurea Magistrale in
Sicurezza Informatica: Infrastrutture e Applicazioni
Università di Pisa – Polo di La Spezia
C. Montangero
Anno accademico 2009/10

Sommario

- Chiamata di procedure
- Funzioni
- Accept
- Quantificatori

S³ 2009/10 – Condizioni di verifica – Parte III

CHIAMATA DI PROCEDURE

S3: VC- C.Montangero - Copyright 2010

3

Il problema

- Come si calcola la wp di una chiamata?
 - Come una serie di assegnamenti ai parametri out:

$$X=v_0 \Rightarrow wp[D(X)]X=v_0^*4$$

$$wp[D(X)]X=v_0^*4 \equiv v=v_0^*4$$
- Dove v è una costante, sempre diversa
 - denota il valore *assegnato*
- Caratterizzata dalla postcondizione della procedura
 - i valori assegnati soddisfano la postcondizione
 - ad esempio: $D^{\wedge} \equiv Y=Y^{\sim}*2$ (Y è il parametro)
- L'ipotesi su v dovuta alla postcondizione è
 - $D^{\wedge}[Y/v, Y^{\sim}/X] \equiv v = X*2$

S3: VC - C.Montangero - Copyright 2010

4

Una chiamata non basta

- Quindi:

$$\begin{aligned} & \text{wp}[D(X)] \ X=v0*4 \\ \equiv & \\ & v=v0*4 \text{ and } v = X*2 \\ \equiv & \text{-- all'inizio } v0 = X \\ & X*4 = X*2 \equiv X=0, \text{ insufficiente} \end{aligned}$$
- come prevedibile... ma due sì:

$$\begin{aligned} & \text{wp}[D(X)](v=v0*4 \text{ and } v = X*2) \\ \equiv & \\ & v=v0*4 \text{ and } v= v1*2 \text{ and } v1=X*2 \\ \equiv & \text{-- all'inizio } v0 = X \\ & v=X*4 \text{ and } v= v1*2 \text{ and } v1=X*2 \\ \equiv & \\ & v=X*4 \text{ and } v= X*2*2 \\ \equiv & \\ & X*4 = X*2*2 \equiv \text{true} \end{aligned}$$

S3: VC - C.Montangero - Copyright 2010

5

Run time check sugli argomenti

- Oltre al controllo sulla rappresentabilità del valore,
- quello sul rispetto della preconditione: $\wedge D[\text{Par/Arg}]$
- Esempio: divisione

```

procedure div(Ddo, Dre: Natural; Q, R: out Natural);
--# pre Dre<>0;
--# post Ddo = Q*Dre + R and R < Dre;

```

S3: SPARK - C.Montangero - Copyright 2010

6

Esercizio: $\text{wp}[\text{div}(X,Y,Z,W)]W=0$

```
procedure div(Ddo, Dre: Natural; Q, R: out Natural);
--# pre Dre<>0;
--# post Ddo = Q*Dre + R;
```

```
wp[div(X,Y,Z,W)]W=0
≡
(W=0)[Q/q1,R/w1] and div^[Ddo/X,Dre/Y,Q/q1,R/w1]
≡
w1=0 and X= q1*Y + w1
cioè
X = q1*Y    (some q1 => ...)
```

- Considerando anche la preconditione:

```
^div[Ddo/X,Dre/Y,Q/Z,R/W] ≡ Dre<>0[Dre/Y] ≡ Y<>0
```

S³ 2009/10 – Condizioni di verifica – Parte III

FUNZIONI

Specifica delle funzioni

```
function_constraints ::= [ --# pre predicate ]
                     [ --# return_expression ]
return_expression ::= return annotation_expression
                  | return simple_name => predicate
```

- Il secondo serve per definire il valore
- La prima forma per relazione semplice
 - esprimibili come espressione d'annotazione
- Esempio:

return X+1 ; -- dove X è il parametro

S3: VC - C.Montangero - Copyright 2010

9

Specifica delle funzioni (2)

- La seconda per relazioni più complesse
 - caratterizza il risultato con un predicato

```
return M => (X > Y -> M = X and
            Y >= X -> M = Y);
```

 - *Si legge: restituisci M tale che se X>Y allora M=X, altrimenti M=Y*
- La prima è un'abbreviazione per:


```
return Dummy => ( Dummy = X+1 )
```

S3: VC - C.Montangero - Copyright 2010

10

Verifica di una funzione

```

function Max(X,Y: in Integer) return Integer
--# return D => (X>Y -> D=X) and (X<=Y -> D=Y);
is
  Dummy
  M : Integer ;
begin
  if (X>Y)
  then
    M := X;
  else
    M := Y;
  end if;
  return M;
end Max;

```

Diagram labels:

- Dummy**: points to the return statement `return M;`
- pseudoPost**: points to the postcondition `--# return D => (X>Y -> D=X) and (X<=Y -> D=Y);`
- body**: points to the `if` block
- Rexp**: points to the expression `M` in the return statement

- Viene risolta come

$$^{\wedge}\text{Max} \rightarrow \text{wp}[\text{body}]\text{pseudoPost}[\text{Dummy}/\text{Rexp}]$$
 – posizione fissa di return...

S3: VC - C.Montangero - Copyright 2010

11

Condizioni di verifica

- Una tipica, da Max:

```

function_max_3.
...
H6:  x > y .
...
->
C1:  (x > y) -> (x = x) .
C2:  (x <= y) -> (x = y) .
...

```

- Ridotta a true da Simplifier.

S3: VC - C.Montangero - Copyright 2010

12

Chiamata di funzioni

- Data

```
function Max(X,Y: in Integer) return Integer
--# return M => (X>Y -> M=X) and (X<=Y -> M=Y);
```

- come descrivo il risultato di `R:=Max(F1,F2);` in

```
procedure Call(F1,F2: in Integer; R : out Integer)
--# derives R from F1,
--#                F2;
--# post R =      ?      ;
```

- Usando la *funzione di prova*

```
function max(x,y : integer) return integer
– implicitamente dichiarata con Max
– notare il gioco Maiusc/Minusc
```

S3: VC - C.Montangero - Copyright 2010

13

Funzioni di prova

```
procedure Call(F1,F2: in Integer; R : out Integer)
--# derives R from F1,
--#                F2;
--# post R = max(F1,F2);
is
begin
  R := Max(F1,F2);
end Call;
```

- `wp[R:=Max(F1,F2)]R=max(F1,F2)`
- `Max(F1,F2) = max(F1,F2)`
- `max(F1,F2) = max(F1,F2)`
- *true ... purchè Max sia corretta!*

S3: VC - C.Montangero - Copyright 2010

14

Documentazione

```
-- function max(x,y : integer) return integer;  
-- Def: ( x>=y -> max(x,y) = x ) and ( x<=y -> max(x,y) = y ) ;  
-- Th (di correttezza): Max(x,y) = max(x,y);  
-- prova: max.siv 28-APR-2010, 21:54:01
```

- Le definizioni e il teorema di correttezza
– nel file .rul per il Proof Checker...
- Codice in funzioni\max.adb

S3: VC - C.Montangero - Copyright 2010

15

S³ 2009/10 – Condizioni di verifica – Parte III

ANNOTAZIONE ACCEPT

S3: VC- C.Montangero - Copyright 2010

16

--# accept

- In una funzione che calcola il resto:

```

Z := 0;
T := Ddo;
loop
  --# assert Ddo = Z*Dre + T;
  exit when (T < Dre);
  T := T - Dre;
  Z := Z+1;
end loop;

```

- Z serve solo come ausilio, non viene usata
- Examiner genera un errore di Flow:

Locations			
Examiner (2 items)			
resto.adb (2 items)			
45:7	Flow Error	10	- Ineffective statement
51:10	Flow Error	10	- Ineffective statement.

S3: VC - C.Montangero - Copyright 2010 17

--# accept (2)

```

--# accept Flow, 10, "Z serve per calcolare T";
Z := 0;
T := Ddo;
loop
  --# assert Ddo = Z*Dre + T;
  exit when (T < Dre);
  T := T - Dre;
  Z := Z+1;
end loop;

```

tipo (flow, warn), numero, "giustificaz

Locations			
Examiner (2 items)			
resto.adb (2 items)			
45:7	Flow Error	10	- Ineffective statement
51:10	Flow Error	10	- Ineffective statement.

S3: VC - C.Montangero - Copyright 2010 18

--# accept (3)

```
--# accept Flow, 10, "Z serve per calcolare T";
Z := 0;
T := Ddo;
loop
  --# assert Ddo = Z*Dre + T;
  exit when (T < Dre);
  T := T - Dre;
44  Z := Z+1;
end loop;
```

In .lsb:

Expected messages marked with the accept annotation

Type	Msg	Lines		Reason	Match	
	No.	From	To		No.	Line
Flow	10	11	end	Z serve per calcolare T	2	18
Flow	10	44	end	Z serve per calcolare T	2	51

QUANTIFICATORI

Quantificatori

- Come si specifica una funzione che restituisce il resto della div?
- Devo dire: restituisco un R tale che,
 - per qualche Q vale $Ddo = Dre * Q + R$ and $R < Dre$
- Come si specifica una funzione che dice se tutti gli elementi di un array sono nulli?
- Devo dire: restituisco
 - per ogni i nel range di indici ammissibili, $A(i)=0$

S3: VC - C.Montangero - Copyright 2010

21

Quantificatori: sintassi

quantified_expression ::=
 kind identifier in *discrete_subtype_mark* [range range] => (predicate)
 kind ::= **for all** | **for some**

- Esempi
 - for some Q in Natural => ($Ddo = Dre * Q + R$)
 - for some M in Index => ($A(M) = Sought$)
 - for all I in Index => ($A(I) = 0$)
 - for all M in Index range 1 .. (Z - 1) => ($A(M) \neq Sought$)
- Warning:
 - non c'è vero scoping: identifier devono essere freschi

S3: VC - C.Montangero - Copyright 2010

22

La funzione Resto

```
function Resto2(Ddo, Dre: Natural) return Natural
--# pre  Dre /= 0;
--# return R =>
--#      (for some Q in Natural => ( Ddo = Q*Dre + R and R < Dre ));
```

- Chiamata:

```
procedure Call2(F1,F2: in Natural; Res : out Natural)
--# derives Res from F1, F2;
--# pre  F2 /= 0;
--# post Res = resto2(F1,F2);
is
begin
  Res := Resto2(F1,F2);
end Call2;
```

- La post usa la funzione di prova implicita

S3: VC - C.Montangero - Copyright 2010

23

La funzione resto

- La VC For path(s) from start to finish:
 procedure_call2_3.
 ...
 ->
 C1: resto2(f1, f2) = resto2(f1, f2) .

- viene semplificata, il che è lecito purché

```
-- function resto2(x,y : Natural) return Natural;
-- Def: for some qq in Natural =>
--      ( y <> 0 -> ( x = qq*y + rest(x,y) and rest(x,y) < y ) ) ;
-- Th (di correttezza): y <> 0 -> Resto2(x,y) = resto2(x,y) ;
-- prova: call2.siv 28-APR-2010, 22:59:00
```

- codice in funzioni/resto.adb

S3: VC - C.Montangero - Copyright 2010

24

Esercizio

- Progettare e realizzare un package Divisione,
 - procedura che fa la divisione di due naturali, e memorizza in uno stato globale "own" quoziente e resto
 - due funzioni che restituiscono rispettivamente quoziente e resto dell'ultima divisione effettuata.
- Come si tratta il divisore nullo?
- E la chiamata delle funzioni prima di una divisione?

S3: VC - C.Montangero - Copyright 2010

25