

# Sviluppo di Software Sicuro - S<sup>3</sup>

## Condizioni di verifica

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

## Sommario

- Parte I
  - Concetti introduttivi
    - precondizioni più deboli
  - Comandi elementari
  - Comandi composti: sequenza e condizionale
- Parte II
  - Cicli
  - Generazione di condizioni di verifica in Examiner

S<sup>3</sup> 2009/10 – Condizioni di verifica – Parte I

## INTRODUZIONE

S3: VC- C.Montangero - Copyright 2010

3

## Il problema

- Data la specifica di P, verificare che il suo corpo rispetta il contratto:
 

```

procedure P(X: in out Integer) ;
  --# derives X from *;
  --#pre  X<Integer'Last;
  --#post X=X~+2;
      
```
- Assunta la precondizione, se
  - eseguendo il comando  $X:=X+1$  si termina,
  - si termina in uno stato che soddisfa la postcondizione

```

procedure P(X: in out Integer)
is begin
  X:=X+1;
end P;
      
```
- Correttezza *parziale*
- Correttezza *totale*
  - no eccezioni (trattate)
  - terminazione dei cicli (a parte)

S3: VC- C.Montangero - Copyright 2010

4

## L'approccio

- Robert Floyd: annotazioni ai flow-chart (1967)
- Sir Tony Hoare: assiomi per semplice linguaggio (1969)
- Edsger Dijkstra: calcolo delle precondizioni (1976)
- R. W. Floyd. "Assigning meanings to programs." Proceedings of the American Mathematical Society Symposia on Applied Mathematics. Vol. 19, pp. 19–31. 1967.
- C. A. R. Hoare. "An axiomatic basis for computer programming". *Communications of the ACM*, 12(10):576–580, 583 October 1969.
- E. W. Dijkstra. "A discipline of programming." Prentice-Hall. 1976.

S3: SPARK - C.Montangero - Copyright 2010

5

## Assiomi per l'evoluzione dello stato

- Hoare caratterizza ciascun comando in base alle modifiche che opera sullo stato
- Formato  $[P] S [Q]$ 
  - se lo stato iniziale soddisfa  $P$ , lo stato finale soddisfa  $Q$
- Esempi di assiomi (e regole d'inferenza)
  - $[P] \text{ null}; [P]$
  - sequenza  $S S'$ 

$$\frac{[P] S [Q] \quad [Q] S' [R]}{[P] S S' [R]}$$

S3: VC- C.Montangero - Copyright 2010

6

## Il problema con Hoare

- La prima e l'ultima asserzione sono facili:
  - pre- e post-condizione
  - indicandole con  $\wedge P$  e  $P^\wedge$  per la procedura  $P$  con corpo  $B$ , il problema diventa
- In generale  $B$  è una sequenza
  - Bisogna inventarsi le proposizioni intermedie
- Dijkstra: *calcoliamo*  $WP$  da  $P^\wedge$  e  $B$  tale che

$$\frac{\wedge P \Rightarrow WP}{[\wedge P] B [P^\wedge]}$$

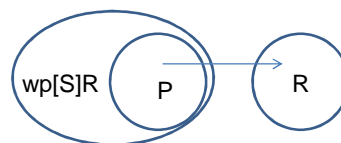
S3: VC - C.Montangero - Copyright 2010

7

## WP: weakest precondition

- Dato un comando  $S$  e un'asserzione  $R$ ,
- $wp[S]R$
- è la più debole condizione che garantisce che
  - se  $S$  termina,
  - si arriva in uno stato che soddisfa  $R$
- caratterizza il più ampio insieme di stati da cui...

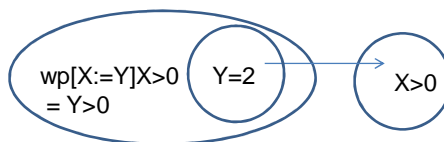
$$\frac{[P] S [R]}{P \Rightarrow wp[S]R}$$



S3: VC - C.Montangero - Copyright 2010

8

## Esempio



- Prendere per fede, per ora
- Notare le somiglianze tra  $X>0$  e  $Y>0$
- Semantica assiomatica
  - coerente con le altre
    - operativa, denotazionale

S3: VC - C.Montangero - Copyright 2010

9

S<sup>3</sup> 2009/10 – Condizioni di verifica – Parte I

## COMANDI ELEMENTARI

S3: VC- C.Montangero - Copyright 2010

10

## Un semplice linguaggio

- Sia Ass il linguaggio delle asserzioni
  - comprende le espressioni booleane del linguaggio
- $S ::= \text{skip} \mid x := e \mid \text{cond}(e, S_t, S_f) \mid \text{seq}(S, S') \mid \text{loop}(e, \text{Ass}, S)$
- $\text{wp} : S \times \text{Ass} \rightarrow \text{Ass}$
- $\text{wp}[S]R$  è la più debole condizione che assicura che se  $S$  termina, termina in uno stato che soddisfa  $R$
- tranne che per il loop, è anche totale

S3: VC - C.Montangero - Copyright 2010

11

## Comandi semplici

- $\text{wp}[\text{skip}]R \equiv R$  ( $\equiv$  denota l'equivalenza logica)
  - dato che skip non fa nulla, non possiamo che partire da uno stato che soddisfa già  $R$
- $\text{wp}[x := e]R$  - preliminari
  - Se  $A$  è un'asserzione,  $e$  un'espressione  
 $A[x/e]$  è l'asserzione ottenuta sostituendo ogni occorrenza (libera) di  $x$  in  $A$  con  $e$
  - Esempio:  $y - z > 0[z/z - 1] \equiv y - (z - 1) > 0 = y - z > -1$

S3: VC - C.Montangero - Copyright 2010

12

## Assegnamento

- $wp[x := e]R \equiv R[x/e]$ 
  - ciò che vale dopo per  $x$ , deve valer prima per  $e$
- Per renderla totale
  - $wp[x := e]R \equiv R[x/e] \textbf{ and } \text{Def}[e]$
  - $\text{Def}$  esprime condizioni sufficienti perchè  $e$  non generi eccezioni (meglio se anche necessarie)

S3: VC - C.Montangero - Copyright 2010

13

## Assegnamento: esempi

- $wp[x := 0]x \geq 0 \equiv x \geq 0[x/0] \equiv 0 \geq 0 \equiv \text{true}$
- $wp[x := 0]x > 0 \equiv x > 0[x/0] \equiv 0 > 0 \equiv \text{false}$ 
  - nel costruire un assegnamento si pensa: "come posso sfruttare i valori che ho a disposizione per dare alla variabile un valore utile dopo il comando?"
  - il primo va bene, il secondo non mi avvicina all'obiettivo
  - $\text{false}$  indica l'insieme vuoto (nessuno stato)
  - $\text{true}$  l'universo degli stati d'interesse

S3: VC - C.Montangero - Copyright 2010

14

## Assegnamento: esempi (2)

- $wp[x:=x-1]x>0 \equiv x>0[x/x-1] \equiv x-1>0 \equiv x>1$   
– se voglio decrementare x mantenendolo positivo, posso farlo, purchè x sia almeno 1
- $wp[x:=x+1]x>0 \equiv x>0[x/x+1] \equiv x \geq 0$
- Ma, gli interi sono limitati  
–  $Def[x+1] \equiv Integer'First \leq x+1 \leq Integer'Last$
- $wp[x:=x+1]x>0 \equiv x \geq 0$  **and**  $x < Integer'Last$
- NB: i tipi sono essenziali per Def

S3: VC - C.Montangero - Copyright 2010

15

S<sup>3</sup> 2009/10 – Condizioni di verifica – Parte I

## COMANDI COMPOSTI

S3: VC- C.Montangero - Copyright 2010

16



## Condizionale

- $wp[\text{cond}(e, S_t, S_f)] R \equiv$   
 $(e \text{ and } wp[S_t]R) \text{ or } (\text{not } e \text{ and } wp[S_f]R)$ 
  - si esegue  $S_t$  solo se  $e$  vale, viceversa per  $S_f$
  - per renderla totale anche  $\text{Def}[e]$
- Esempio (trascurando  $\text{Def}$ ):  
 $wp[\text{cond}(x \geq 0, \text{skip}, x := -x)] x \geq 0$   
 $\equiv (x \geq 0 \text{ and } wp[\text{skip}]x \geq 0) \text{ or}$   
 $(x < 0 \text{ and } wp[x := -x]x \geq 0)$   
 $\equiv (x \geq 0 \text{ and } x \geq 0) \text{ or } (x < 0 \text{ and } -x \geq 0)$   
 $\equiv x \geq 0 \text{ or } x < 0$   
 $\equiv \text{true}$  (qualunque stato (valore di  $x$ ) va bene)  
 Ma:  $\text{Def}[-x] \equiv \text{Int}'F \leq -x \text{ and } -x \leq \text{Int}'L \iff -\text{Int}'L \leq x \leq \text{Int}'L$

S3: VC - C.Montangero - Copyright 2010

17

## Sequenza

- $wp[\text{seq}(S, S')]R \equiv wp[S](wp[S']R)$ 
  - se  $S$  mi porta in uno stato da cui posso arrivare in  $R$ , sono a posto
  - eleganza: composizione di comandi  $\rightarrow$  comp.  $wp$
- Esempio:  
 $wp[\text{seq}(x := x+1, x := x+1)] x \geq 0$   
 $\equiv wp[x := x+1] (wp[x := x+1]x \geq 0)$   
 $\equiv wp[x := x+1] (x+1 \geq 0 \text{ and}$   
 $\text{Integer}'\text{First} \leq x+1 \leq \text{Integer}'\text{Last})$   
 $\equiv (x+1)+1 \geq 0 \text{ and}$   
 $\text{Integer}'\text{First} \leq (x+1)+1 \leq \text{Integer}'\text{Last}$   
 $\equiv x \geq -2 \text{ and } \text{Integer}'\text{First} \leq x+2 \leq \text{Integer}'\text{Last}$   
 sapendo che  $x$  è Integer ( $\geq \text{Integer}'\text{First}$ )  
 $\equiv x \geq -2 \text{ and } x \leq \text{Integer}'\text{Last}-2$

S3: VC - C.Montangero - Copyright 2010

18

## Esempio

Sia  $MAX \equiv (M=X \text{ or } M=Y) \text{ and } M \geq X \text{ and } M \geq Y$   
 $wp[ \text{seq}(\text{cond}(X>Y, T:=X, T:=Y), M:=T) ] MAX$   
 $\equiv wp[\text{cond}(X>Y, T:=X, T:=Y)](wp[M:=T]MAX)$   
 $\equiv wp[\text{cond}(X>Y, T:=X, T:=Y)]MAX[M/T]$   
 $\equiv (X>Y \text{ and } wp[T:=X] MAX[M/T]) \text{ or }$   
 $(X \leq Y \text{ and } wp[T:=Y] MAX[M/T])$   
 $\equiv (X>Y \text{ and } MAX[M/T][T/X]) \text{ or }$   
 $(X \leq Y \text{ and } MAX[M/T][T/Y])$

S3: VC - C.Montangero - Copyright 2010

19

## Esempio (2)

$MAX[M/T]$   
 $\equiv (T=X \text{ or } T=Y) \text{ and } T \geq X \text{ and } T \geq Y$   
  
 $MAX[M/T][T/X]$   
 $\equiv (X=X \text{ or } X=Y) \text{ and } X \geq X \text{ and } X \geq Y$   
 $\equiv \text{true and true and } X \geq Y$   
  
 $X>Y \text{ and } wp[T:=X] MAX[M/T]$   
 $\equiv X>Y \text{ and } X \geq Y \equiv X>Y$

S3: VC - C.Montangero - Copyright 2010

20

## Esempio (3)

Analogamente per l'altro ramo:

$\text{MAX}[M/T][T/Y] \equiv X \leq Y$

e

$X \leq Y \text{ and wp}[T:=Y] \text{ MAX}[M/T] \equiv X \leq Y$

quindi

$\text{wp}[\text{seq}(\text{cond}(X > Y, T := X, T := Y), M := T)]$

$\equiv X > Y \text{ or } X \leq Y$

$\equiv \text{true}$  : va sempre bene, *purchè siano Integer*

S3: VC - C.Montangero - Copyright 2010

21

S<sup>3</sup> 2009/10 – Condizioni di verifica – Parte II

## CICLI

S3: VC - C.Montangero - Copyright 2010

22

## Cicli

- Esempio
  - da SPARK: calcolo  $X*Z$
  - posto di exit bloccato
- Perché **assert**?
  - in questo caso, si calcola a partire da informazioni sulla semantica del ciclo: l'*invariante*, che descrive come il risultato vien costruito per *approssimazioni successive*
    - in questo caso il prodotto per somme successive di X e Z
- $wp[\text{loop}(e, \text{Ass}, S)]R \equiv ?$ 
  - non basta, induce delle condizioni di verifica aggiuntive
    - rispetto a quella globale della procedura

```
loop
--# assert P=X*Y and 0<=Y<=Z;
exit when Y=Z;
P:=P+X; Y:=Y+1;
end loop;
```

S3: VC - C.Montangero - Copyright 2010

23

## L'approccio

```
loop
--# assert P=X*Y and 0<=Y<=Z;
exit when Y=Z;
P:=P+X; Y:=Y+1;
end loop;
```

- Un ciclo taglia la propagazione delle wp:
  - si garantisce che
    - l'invariante venga stabilito e che la guardia sia definita
      - inizializzazione del ciclo

$wp[\text{loop}(e, \text{Ass}, S)]R \equiv \text{Ass and Def}[e]$

S3: VC - C.Montangero - Copyright 2010

24

## L'approccio (2)

```

loop
--# assert P=X*Y and 0<=Y<=Z;
exit when Y=Z;
P:=P+X; Y:=Y+1;
end loop;

```

- Bisogna però garantire anche:
  - VC1: quando si esce si stabilisce R  
 $(\text{Ass and } e) \Rightarrow R$ 
    - R implica la wp di tutto quello che segue il ciclo
  - VC2: nel ciclo si conserva l'invariante e Def[e]  
 $(\text{Ass and not } e) \Rightarrow \text{wp}[S](\text{Ass and Def}[e])$

S3: VC - C.Montangero - Copyright 2010

25

## Esempio

```

loop
--# assert P=X*Y and 0<=Y<=Z;
exit when Y=Z;
P:=P+X; Y:=Y+1;
end loop;

```

LOOP

wp[LOOP]P=X\*Z  $\equiv$   
 $P=X*Y \text{ and } 0 \leq Y \leq Z \text{ and true}$

VC1 (da assert alla fine)  
 $(P=X*Y \text{ and } 0 \leq Y \leq Z \text{ and } Y=Z) \Rightarrow P=X*Z$   
 banalmente vera

S3: VC - C.Montangero - Copyright 2010

26

## Esempio (2)

```

loop
--# assert P=X*Y and 0<=Y<=Z;
exit when Y=Z;
P:=P+X; Y:=Y+1;
end loop;

```

LOOP

VC2 (da assert a assert):

$P=X*Y \text{ and } 0 \leq Y \leq Z \text{ and } Y \neq Z$

$\Rightarrow$

$\text{wp}[P:=P+X; Y:=Y+1;](P=X*Y \text{ and } 0 \leq Y \leq Z)$

S3: VC - C.Montangero - Copyright 2010

27

## Esempio (3)

```

loop
--# assert P=X*Y and 0<=Y<=Z;
exit when Y=Z;
P:=P+X; Y:=Y+1;
end loop;

```

LOOP

Prima parte.  $P=X*Y$  è invariante.

$\text{wp}[P:=P+X; Y:=Y+1;]P=X*Y \equiv$

$P+X=X*(Y+1) \equiv P+X=X*Y+X \equiv P=X*Y$

Ma:  $P=X*Y \text{ and } 0 \leq Y \leq Z \text{ and } Y \neq Z \Rightarrow$

$P=X*Y$

QVD

S3: VC - C.Montangero - Copyright 2010

28

## Esempio (4)

```

loop
--# assert P=X*Y and 0<=Y<=Z;
exit when Y=Z;
P:=P+X; Y:=Y+1;
end loop;

```

LOOP

Seconda parte:  $0 \leq Y \leq Z$  è invariante

$\text{wp}[P:=P+X; Y:=Y+1] \ 0 \leq Y \leq Z$

$\equiv 0 \leq Y+1 \leq Z$

$\equiv 0 \leq Y < Z$

Ma:  $P=X*Y \text{ and } 0 \leq Y \leq Z \text{ and } Y \neq Z$

$\Rightarrow 0 \leq Y < Z$  QVD

S3: VC - C.Montangero - Copyright 2010

29

## Esempio (5)

```

loop
--# assert P=X*Y and 0<=Y<=Z;
exit when Y=Z;
P:=P+X; Y:=Y+1;
end loop;

```

LOOP

Qual è il modo più semplice di inizializzare,  
ossia stabilire l'invariante

$P=X*Y \text{ and } 0 \leq Y \leq Z$  ?

La sequenza

$P:=0; Y:=0;$

Infatti

$\text{wp}[P:=0; Y:=0;] \ P=X*Y \text{ and } 0 \leq Y \leq Z \equiv \text{true}$

S3: VC - C.Montangero - Copyright 2010

30

## Concludendo

```

procedure Prod(X,Z: in Natural; P : out Natural)
--# derives P from X,Z;
--# pre: ?;
--# post: P = X*Z;
is
  Y: Integer;
begin
  P:=0; Y:=0;
  loop
    --# assert P=X*Y and 0<=Y<=Z;
    exit when Y=Z;
    P:=P+X; Y:=Y+1;
  end loop;
end Prod;

```

- Come dev'essere pre,  
per evitare eccezioni?
- $X*Z \leq \text{Natural'Last}$

S3: VC - C.Montangero - Copyright 2010

31

## E quindi...

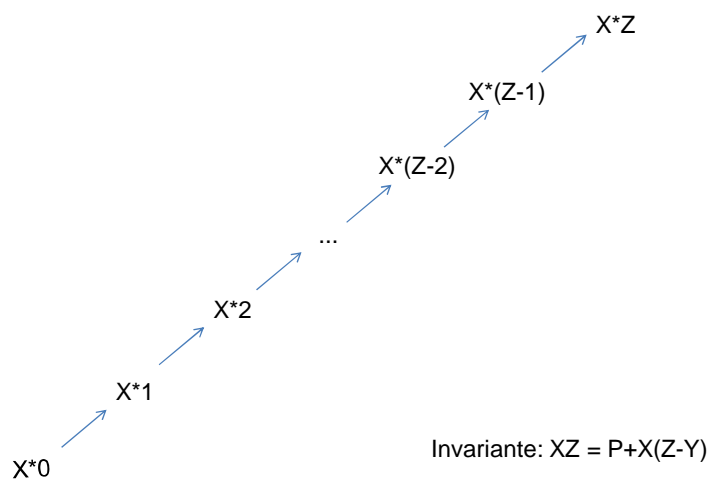
- Alla chiamata di Prod(A,B):
  - verifica dinamica
    - if  $A \leq \text{Natural'Last}$  / B  
then Prod(A,B)  
else -- errore
  - verifica statica:
    - --# check  $A*B \leq \text{Natural'Last}$
    - propago all'indietro le condizioni  
– fino alla verifica degli input
- Presuppone la disponibilità delle operazioni inverse

S3: VC - C.Montangero - Copyright 2010

32



## Interpretazione grafica



S3: VC - C.Montangero - Copyright 2010

33

## Esercizio

- Divisione per sottrazioni successive:
  - $Ddo = Dre * Q + R$  **and**  $R < Dre$
- Condizione di uscita:  $R < Dre$