S3 2009/10 – Condizioni di verifica – Parte II

CICLI

Cicli

- Esempio
  - da SPARK: calcolo X*Z
  - posto di exit bloccato

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

- Perchè **`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[loop(*e*,Ass,S)]R ≡ ?
  - non basta, induce delle condizioni di verifica aggiuntive
    - rispetto a quella globale della procedura

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[loop(*e*,Ass,S)]R ≡ Ass **and** Def[e]

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

    (Ass **and** e) => R
    - R implica la wp di tutto quello che segue il ciclo
  - VC2: nel ciclo si conserva l'invariante e Def[e]

    (Ass **and not** e) => wp[S](Ass **and** Def[e])

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 ≡

P=X*Y **and** 0<=Y<=Z **and true**

VC1 (da assert alla fine)

(P=X*Y **and** 0<=Y<=Z **and** Y=Z) => P=X*Z

banalmente vera

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 **and** 0<=Y<=Z **and** Y<>Z

=>

wp[P:=P+X; Y:=Y+1;](P=X*Y **and** 0<=Y<=Z)

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.

wp[P:=P+X; Y:=Y+1;]P=X*Y ≡

P+X=X*(Y+1) ≡ P+X=X*Y+X ≡ P=X*Y

Ma: P=X*Y **and** 0<=Y<=Z **and** Y<>Z =>

P=X*Y

QVD

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<=Y<=Z è invariante

wp[P:=P+X; Y:=Y+1] 0<=Y<=Z

≡ 0<=Y+1<=Z

≡ 0<=Y<Z

Ma: P=X*Y **and** 0<=Y<=Z **and** Y<>Z

=> 0<=Y<Z QVD

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 **and** 0<=Y<=Z ?

La sequenza

P:=0; Y:=0;

Infatti

wp[P:=0; Y:=0;] P=X*Y **and** 0<=Y<=Z ≡ true

Concludendo

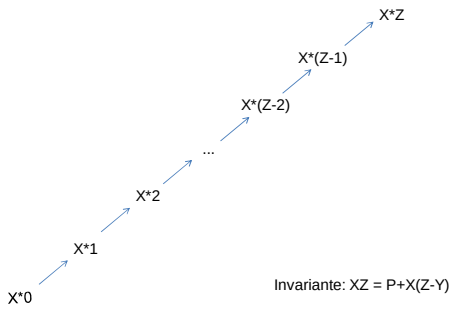
```
procedure Prod(X,Z: in Natural; P : out Natural)
--# derives P from X,Z;
--# pre: ?;
--# post: P = X*Z;
is
  Y: Integer;
begin
  P:=0; Y:=0;
  loop
  --# assert P=X*Y and 0<=Y<=Z;
  exit when Y=Z;
  P:=P+X; Y:=Y+1;
  end loop;
end Prod;
```

- Come dev'essere pre,
  per evitare eccezioni?
- X*Z <= Natural'Last

E quindi...

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

Interpretazione grafica



Esercizio

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