Verifica formale di programmi con la logica di Hoare

A.Gargantini
Linguaggi di Programmazione per la sicurezza,
Crema 2005

Scopo

- provare che i programmi sono corretti (sicuri)
  - diverso dal testing (qui proviamo veramente)
  - lo faremo a mano anche se ci sono strumenti
  - analisi statica: non richiede esecuzione del programma

materiale

- Questi lucidi
- Altro materiale su web
  - file AxiomaticSemantics.pdf
  - Capitolo 10, sezione 2 del Finkel: finkel10.pdf
- Capitolo 7 del Peled che tovate in biblioteca

While programs

- I nostri programmi saranno composti dalle seguenti istruzioni
  - Assignments y := t
  - Composition S1; S2
  - If-then-else if e then S1 else S2 fi
  - While while e do S od
- Variabili intere

Cosa vuol dire provare che un programma è corretto

Proveremo

$$\{P\}S\{Q\}$$

- Se P è vero prima di eseguire il programma S, allora dopo varrà Q
  - P: precondizione
  - Q: postcondizione
  - S: programma o frammento di programma
- P e Q vanno pensate e scritte

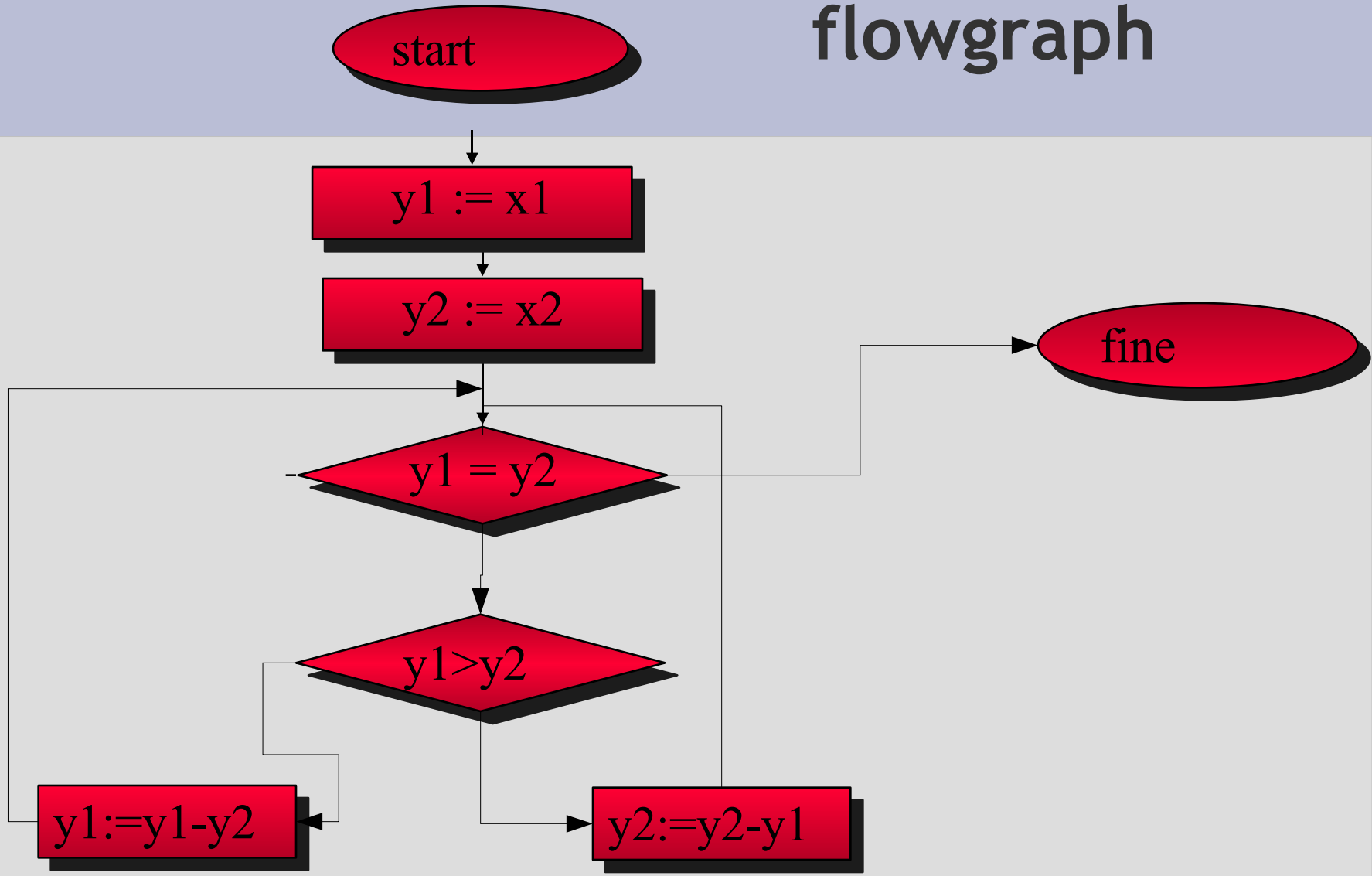
Asserzioni

- P e Q sono asserzioni
- In genere metteremo le asserzioni tra {}
- Scritte in logica
  - Con l'uso di variabili che saranno anche nel programma
  - /\ per AND
    - Esempio { x>0 /\ y = 3}
  - \/ per OR
  - ¬ per NOT
  - -> per implica : A -> B : A implica B, cioè A è più forte di B, es: x =2 -> x >0

Esemio: Greatest common divisor GCD

{x1>0 /\ x2>0}
y1:=x1;
y2:=x2;
while ¬(y1=y2) do
    if y1>y2 then y1:=y1-y2
            else y2:=y2-y1 fi
od
{y1=gcd(x1,x2)}

flowgraph



Perchè funziona?

Suppose that y1,y2 are both positive integers.

- If y1>y2 then gcd(y1,y2)=gcd(y1-y2,y2)
  - Esempio: gcd(10,4) = gcd(6,4)
- If y2>y1 then gcd(y1,y2)=gcd(y1,y2-y1)
- If y1=y2 then gcd(y1,y2)=y1=y2

Come fare a dimostrare {P}S{Q}

- Divideremo la dimostrazioni in piccoli passi
- ad ogni passo applichiamo una regola (assioma) che ci permette di ridurre la dimostrazione ad un caso più semplice

Assignment axiom

{p [y->t] } y:=t {p}

p[y->t]: sostituisci y in p con t

: al posto di y ci metto t

Esempi:

{u+1>3} x:=u+1 {x>3}

{y+y<z} x:=y {x+y<z}

{y+5=10} y:=y+5 {y=10}

{2*(y+5)>20} y:=2*(y+5) {y>20}

{6>10} x:=6 {x > 10}

Giustificazione: scrivi p con y' invece di y e aggiungi il congiunto (/\) y'=t. Ora elimina y' sostituendolo con t.

Esempio

- voglio trovare la precondizione di

a := b +10 {a>0}

- p[a->b+10]= {b+10 > 0}= {b > -10}
- ho provato che

{b > -10} a := b +10 {a>0}

Esercizio

- x = y/2 -1 {x<10}
-
- quale è la precondizione

in avanti

- così andiamo all'indietro (da post calcoliamo la pre) e se volessiomo andare in avanti?

{p} y:=t {?}

come postcondizione scirvo p e il congiunto y=t e sostituisco y con y' sia in p che in t e poi elimino y' (y': old y)

{y>5} y:=2*(y+5) {?}

{p} y:=t {p[y->y'] /\ t[y->y']=y)}

Q: y’>5 /\ y=2*(y’+5) = y>20

2. Composition rule

regola di composizione

$$\frac{\{p\}\ S1\ \{r\},\ \ \{r\}\ S2\ \{q\}}{\{p\}\ S1;S2\ \{q\}}$$

diamo le regole in questa forma

$$\frac{A1,\ \ldots,\ An}{A}$$

vuol dire:

(all'indietro) se vuoi dimostrare A dimostra A1, ..An

(in avanti) se hai dimostrato A1 .. An allora A è vera

Esempio

Per esempio se ho le istruzioni

x:=x+1; y:=y+2

q è x=y, e p è x+1=y+2

posso applicare la regola, infatti

A1. {x+1=y+2} x:=x+1 {x=y+2} per regola asseg.

A2. {x=y+2} y:=y+2 {x=y} per regola assegnamento

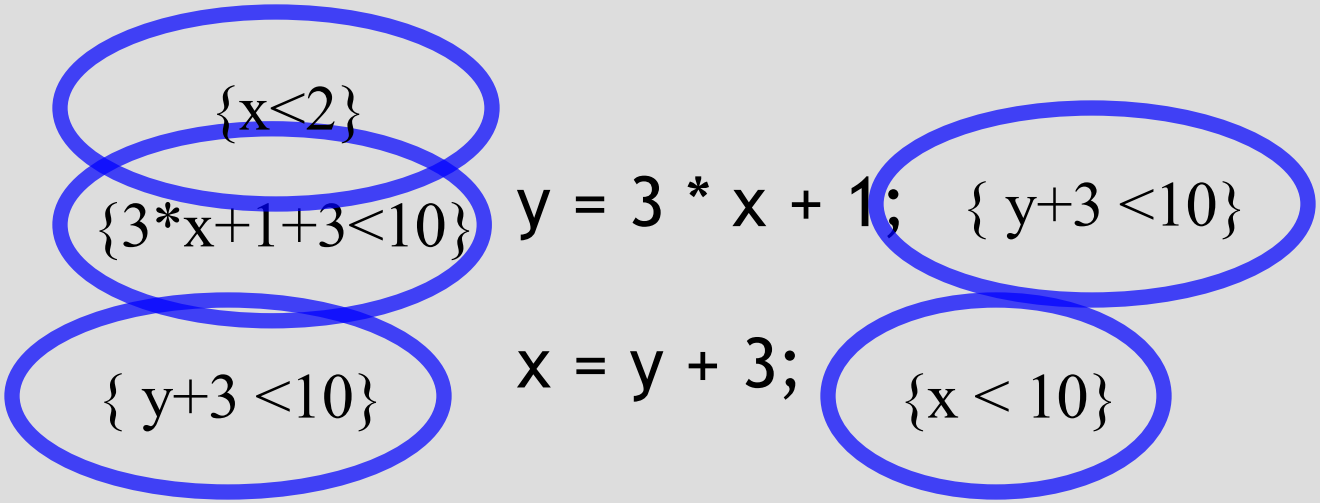
la conseguenza è A:

{x+1=y+2} x:=x+1; y:=y+2 {x=y}

Altro esempio

y = 3 * x + 1;
x = y + 3;
{x < 10}

quale è la precondizione?



More examples

$$\frac{\{p\}\ S1\ \{r\},\ \ \{r\}\ S2\ \{q\}}{\{p\}\ S1;S2\ \{q\}}$$

$\{x1>0/\backslash x2>0\}\ y1:=x1$
$\quad \{gcd(x1,x2)=gcd(y1,x2)/\backslash y1>0/\backslash x2>0\}$

$\{gcd(y1,x2)=gcd(y1,x2)/\backslash y1>0/\backslash x2>0\}\ y2:=x2$
$\quad \{gcd(x1,x2)=gcd(y1,y2)/\backslash y1>0/\backslash y2>0\}$

$\{x1>0/\backslash x2>0\}\ y1:=x1\ ;\ y2:=x2$
$\quad \{gcd(x1,x2)=gcd(y1,y2)/\backslash y1>0/\backslash y2>0\}$

Come usare gli assiomi per provare la correttezza

- nel caso voglio provare {P}S{Q}, con S assegnamento
  - ho già Q, calcolo P e vedo se è quello specificato
  - ho già P, calcolo Q e controllo che sia quello assegnato
- problemi:
    - se P non è esattamente quello trovato
- esempio: prova

** {x>0} x := x+1 {x>0}

Se applico gli assiomi trovo

o P = {x>-1} o Q = {x>1}

intuitivamente ** è corretto

3 Consequence rules

- in questi casi posso applicare queste due regole
- rafforzare una precondizione

$$\frac{r \rightarrow p, \{p\}\ S\ \{q\}}{\{r\}\ S\ \{q\}}$$

- indebolire una postcondizione

$$\frac{\{p\}\ S\ \{q\},\ q \rightarrow r}{\{p\}\ S\ \{r\}}$$

Esempio

$\{x>0\}\ x := x+1\ \{x>0\}$

4 If-then-else rule

$$\frac{\{p \wedge e\}\ S1\ \{q\},\ \{p \wedge \neg e\}\ S2\ \{q\}}{\{p\}\ \text{if } e \text{ then } S1 \text{ else } S2 \text{ fi } \{q\}}$$

prova:

{y>1} if (x > 0) then y := y - 1;

else y := y + 1; {y > 0}

cioè (p = y>1):

{p/\ x >0 } y:=y-1 {y>0}, {p/\x<=0} y:=y-1 {y>0} [assi.]

{y>-1} y:=y-1 {y>0}, {y>1} y:=y-1 {y>0}

ok, {p/\ x >0 } -> {y>-1} e {p/\x<=0} -> {y>1}

If-then-else rule

$$\frac{\{p/\backslash e\}\ S1\ \{q\},\ \{p/\backslash \neg e\}\ S2\ \{q\}}{\{p\}\ \text{if } e \text{ then } S1 \text{ else } S2 \text{ fi } \{q\}}$$

For example:

p is gcd(y1,y2) = gcd(x1,x2)
/\ y1>0 /\ y2>0 /\ ¬(y1=y2)

e is y1>y2

S1 is y1:=y1-y2

S2 is y2:=y2-y1

q is gcd(y1,y2)=gcd(x1,x2)/\y1>0/\y2>0

While rule

$$\frac{\{p \wedge e\}\ S\ \{p\}}{\{p\}\ \text{while } e \text{ do } S \text{ od } \{p \wedge \neg e\}}$$

p e q

{x>=0} while x!=0 do x:=x-1 od {x=0}

1 {p/\¬e} -> {q} : {x>=0 /\ ¬ x!=0} = {x=0} ok

2 {p/\e}S {p} : {x>=0 /\ x!=0} x:=x-1 {x>=0}

{x > 0} x:=x-1 {x>=0}

[assignamento x->x-1, pre : x-1>=0]

{x-1>=0}-> {x>0} ok

invarianti

per i cicli while devo trovare {p} tale che {p/\e} S {p}

{p} : INVARIANTE

trovare un invariante può essere difficile. Quando si deve dimostrare,

{P} while B do S end {Q}

cerco I tale che

1 {I /\ not B} -> {Q}:
l'invariante all'uscita del ciclo implica la postcondizione
2 {I /\ B } S {I} I è effettivamente invariante

Esempio: divisione intera

```
{x>=0 /\ y>=0}
a:=0;
b:=x;
while b>=y do
   b:=b-y;
   a:=a+1
od.
{x=a*y+b /\ 0<=b /\ b<y}
```

Invariante

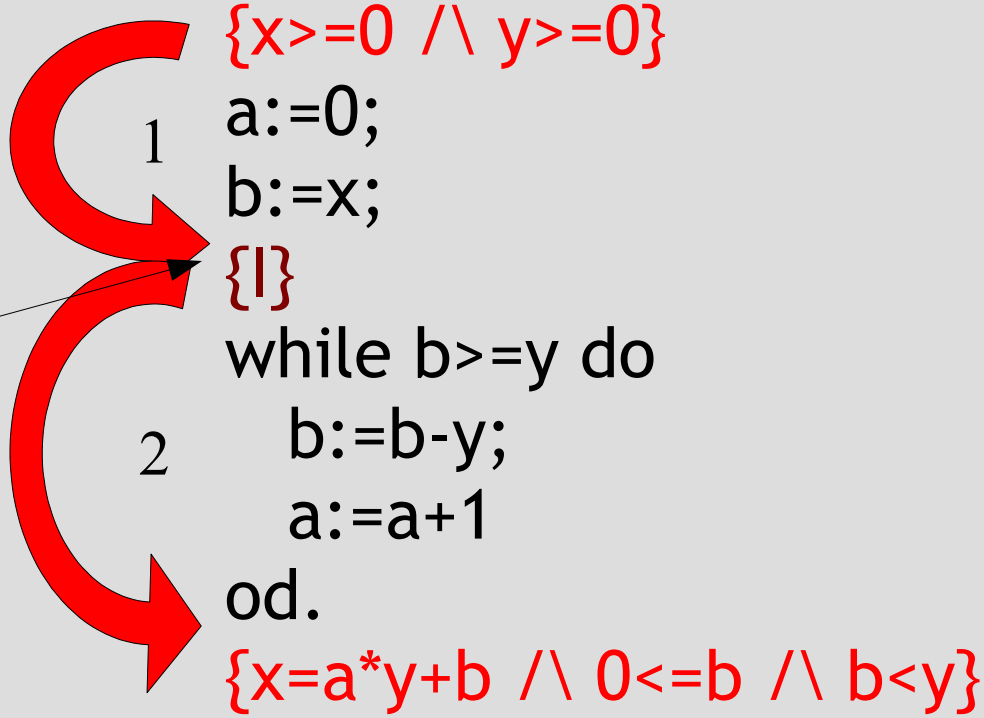
x=a*y+b /\ b>=0

calcola a = x/y e b il resto (x mod y)

Dimostrazione

divido in due
la dimostrazione

prendo P del ciclo
while esattamente
uguale a I
(cond 3 su I è OK)



1 {x>=0 /\ y>=0} a:=0; b:=x; {x=a*y+b /\ b>=0}

Dimostrazione parte 1

$\{p[y->t]\}\ y:=t\ \{p\}$

$$\frac{\{p\}S1\{r\},\ \{r\}\ S2\{q\}}{\{p\}\ S1;S2\ \{q\}}$$

(1) {x=a*y+x/\x>=0} b:=x {x=a*y+b/\b>=0} (Assignment)

(2) {x=0*y+x/\x>=0} a:=0 {x=a*y+x/\x>=0} (Assignment)

(3){x=0*y+x /\x>=0} a:=0;b:=x{x=a*y+b/\x>=0}
(Composition (2), (1))

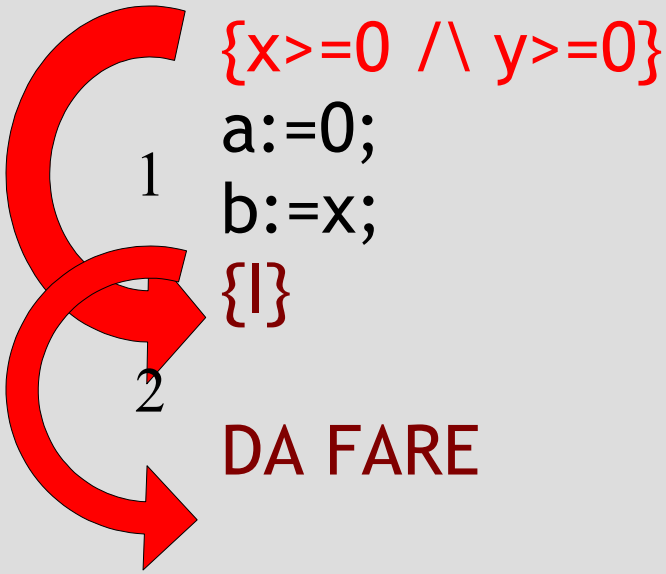
Proof (cont.)

(4) x>=0/\y>=0 -> x=0*y+x/\x>=0 (Logic)

(5) {x>=0 /\ y>=0} a:=0; b:=x

{x=a*y+b/\b>=0/\b<y} (Consequence 3 e 4)

ok quindi



parte 2 : ciclo while

- I ha le proprietà degli invarianti?

1 {I /\ not B} -> {Q}:
l'invariante all'uscita del ciclo implica la postcondizione
2 {I /\ B } S {I} I è effettivamente invariante

3 P -> I : I è valido all'inizio del ciclo OK

Dimostro proprietà 1
{I /\ not B} -> {x=a*y+b /\ 0<=b /\ b<y}
{x = a*y+b /\ b>=0 /\ not b>=y} -> {x=a*y+b /\ 0<=b /\ b<y}

ok perchè not b >= y -> b<y

manca 2: {I /\ B } S {I}
{I /\ B } b:=b-y; a:=a+1{I}

Proof (cont.)

$\{p[t/y]\}\ y:=t\ \{p\}$

$$\frac{\{p\}S1\{r\},\ \{r\}\ S2\{q\}}{\{p\}\ S1;S2\ \{q\}}$$

(6){x=(a+1)*y+b/\b>=0} a:=a+1{x=a*y+b/\b>=0}
(Assignment)

(7){x=(a+1)*y+b-y/\b-y>=0} b:=b-y{x=(a+1)*y+b/\b>=0}
(Assignment)

(8){x=(a+1)*y+b-y/\b-y>=0} b:=b-y;a:=a+1{x=a*y+b/\b>=0}
(Composition (6), (7))

Proof (cont.)

(9) x=a*y+b/\b>=0/\b>=y ->

x=(a+1)*y+b-y /\ b-y >= 0 (Logic)

(10) {x=a*y+b/\b>=0/\b>=y}

b:=b-y; a:=a+1 {x=a*y+b/\b>=0}

(Consequence (8), (9))

(9) {x=a*y+b/\b>=0}while b>=y do b:=b-y;

a:=a+1 od {x=a*y+b/\b>=0/\b<y}

(while (10))

Come “intuire” l'invariante

- L'invariante deve implicare all'uscita la postcondizione
  - Domandati perchè alla fine la post condizione vale?
- Prova a percorrere il ciclo con qualche caso di test, con un “giro” del ciclo, con due e così via
-

Esercizi

anche in questo caso divido la dimostrazione in due parti

```
{y >= 0}
i = 0;
{I}
quad = 0;
while ( i != y) {
    quad = quad + y;
    i = i + 1;
}
{quad = y^2}
```

questo I è assegnato

```
I = q = i * y
```

nella seconda parte devo dimostrare che
1) I and not B -> Q
cioè q = i *y AND not (i!=y)
q = i *y AND i = y -> q = y*y

OK
2) {I and B}S {I}
{q = i*y and i!=y} q = q+y; i = i+1{...}

da fare

Altri esempi

- Esempio 1:somma

```
{n>0}
count = 0;
sum = 0;
while count < n  do
    count = count + 1;
    sum = sum + count;
end
{sum = 1 +2 + ... + n}
```

count cresce fino a diventare n, e sum accumula la somma da 1 a count:

I: sum = 1 + ... + count

- Esempio 2: assegna

```
{x>=0}
Y:= 0;
while (y < x) {
    y:= y+1;
}
{y=x}
```

Esempio 3

```
int old_x = x;
int dop = x;
while (x != 0) {
dop := dop +1 ; x := x-
  1;
}
{dop = 2 *old_x}
```

gcd -skip

```
while ¬(y1=y2) do
    if y1>y2 then y1:=y1-y2
             else y2:=y2-y1 fi
```

p is {gcd(y1,y2) = gcd(x1,x2) /\ y1>0 /\ y2>0}

e is (y1=y2)

S is if y1>y2 then y1:=y1-y2 else y2:=y2-y1 fi

Combining program

{x1>0 /\ x2>0}

y1:=x1; y2:=x1;

{gcd(x1,x2)=gcd(y1,y2)/\y1>0/\y2>0}

while S do

if e then S1 else S2 fi od

{gcd(x1,x2)=gcd(y1,y2)/\y1>0/\y2>0}

Combine the above using concatenation rule!

Use of first consequence rule

Want to prove

{x1>0/\x2>0} y1:=x1
{gcd(x1,x2)=gcd(y1,x2)/\y1>0/\x2>0}

By assignment rule:

{gcd(x1,x2)=gcd(x1,x2)/\x1>0/\x2>0} y1:=x1
{gcd(x1,x2)=gcd(y1,x2)/\y1>0/\x2>0}

x1>0/\x2>0 gcd(x1,x2)=gcd(x1,x2)/\x1>0/\x2>0

Not completely finished

{x1>0/\x2>0}

y1:=x1; y2:=x1;

while ~(y1=y2) do

if e then S1 else S2 fi od

{gcd(x1,x2)=gcd(y1,y2)/\y1>0/\y2>0/\y1=y2}

But we wanted to prove:

{x1>0/\x1>0} Prog {y1=gcd(x1,x2)}

Use of secend consequence rule

{x1>0/\x2>0} Prog
{gcd(x1,x2)=gcd(y1,y2)/\y1>0/\y2>0/\y1=y2}
And the implication
{gcd(x1,x2)=gcd(y1,y2)/\y1>0/\y2>0/\y1=y2}
    ->y1=gcd(x1,x2)
Thus,
{x1>0/\x2>0} Prog {y1=gcd(x1,x2)

Annotating a while program fine skip

{x1>0/\x2>0}
y1:=x1;
{gcd(x1,x2)=gcd(y1,x2)
 /\y1>0/\x2>0}
y2:=x2;
{gcd(x1,x2)=gcd(y1,y2)
 /\y1>0/\y2>0}

while ¬(y1=y2) do
{gcd(x1,x2)=gcd(y1,y2)/\
   y1>0/\y2>0/\¬(y1=y2)}
 if y1>y2 then y1:=y1-y2
          else y2:=y2-y1 fi
od
{y1=gcd(x1,x2)}

Soundness - Consistenza

la logica di Hoare è consisstente (sound) nel senso che ogni cosa che può essere provata è corretta (cioè è vera)

Dim.: ogni assioma conserva la varità dalle asserzioni (consistente).

Completezza

Un sistema di dimostrazioni è detto completo se ogni asserzioni vera può essere dimostrata

- la logica proposizionale è completa
- Per l'aritemtica invece non c'è sistema di regole che sia completo (Godel).

- puoi leggere “Gödel, Escher, Bach: An Eternal Golden Braid”, in italiano: Godel, Escher, Bach un’Eterna Ghirlanda Brillante di Douglas R. Hofstadter
  http://en.wikipedia.org/wiki/G%C3%B6del,_Escher,_Bach

E la logica di Hoare?

Due problemi:

1) incompletezza del calcolo aritemitico

2) probleam della terminazione:

{p} S {false} significa che S non termina a partire dalle precondizioni p. Questo è indecidibile, cioè le regole non riescono a provarlo sempre:

il sistema però è relativamente completo

Esercizi ....