# verifica tramite Logica Temporale

# motivazioni

- la verifica mediante **asserzioni** è limitata a specifici punti di controllo nei processi, ma alcune proprietà di correttezza non sono associabili a specifici punti di controllo; esempi:

**in *ogni* stato di *ogni* computazione:**

- **assenza di starvation**:
  se *un* processo chiede di entrare nella sezione critica, ***in almeno uno degli stati successivi*** deve poterci riuscire
- **assenza di deadlock (1)**:
  se *alcuni* processi chiedono di entrare nella sezione critica, ***in almeno uno degli stati successivi*** *almeno uno di loro* deve poterci riuscire
- **assenza di deadlock (2)**:
  se non può essere eseguita alcuna istruzione, allora il counter di ogni processo deve essere arrivato alla fine
- **mutua esclusione**:
  la variabile fantasma deve assumere quel valore (es: `critica <=1`)

# motivazioni

- la verifica mediante **asserzioni** è limitata a specifici punti di controllo nei processi, ma alcune proprietà di correttezza non sono associabili a specifici punti di controllo; esempi:

**in *ogni* stato di *ogni* computazione:**

- **agganciamento** (*latching*):
  (esempio) se un processo abortisce, da quello stato in poi le variabili che caratterizzano il suo stato devono rimanere settate a zero
- **infinitamente spesso**:
  (esempio) se la computazione è infinita, deve essere sempre possibile che quella variabile riassuma quel valore
- **buffer overflow**:
  `0 <= indice del vettore <= N-1`
- **rispetto delle quantità**:
  il numero dei token circolanti nel sistema distribuito deve essere fisso (es: `#TOKEN = 1`)

# sintassi della Logica Temporale

- basata sulla logica proposizionale

| Operator | Math | SPIN |
|---|---|---|
| not | $\neg$ | ! |
| and | $\wedge$ | && |
| or | $\vee$ | \|\| |
| implies | $\rightarrow$ | -> |
| equivalent | $\leftrightarrow$ | <-> |

- con questi operatori modali

| Operator | Math | SPIN |
|---|---|---|
| always | $\Box$ | [] |
| eventually | $\Diamond$ | <> |
| until | $\mathcal{U}$ | U |

# safety

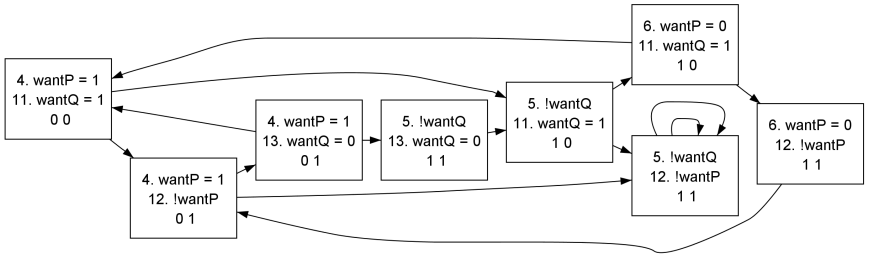
- **`[]` `qualcosa di buono`**
- **`[]!` `qualcosa di cattivo`**
- **`qualcosa di`** deve essere espresso in forma di logica proposizionale o in forma di logica (modale) temporale (LTL)
- procedimento:

1) esprimere formalmente il **`qualcosa di`**
2) aggiungere la ***negazione*** del **`[]` `qualcosa di`** al comando per generare il verificatore
3) compilare il verificatore con l'opzione **`-DSAFETY`** per ottimizzare in funzione della verifica della safety
4) lanciare il verificatore ottimizzato

# es. spin_23: mutua esclusione (modo 1)

```
bool wantP = false, wantQ = false;
byte critica = 0;
#define esclusione (critica <= 1)

active proctype P() {
  do :: wantP = true;
        !wantQ;
        critica++;
        critica--;
        wantP = false
  od
}

active proctype Q() {
  do :: wantQ = true;
        !wantP;
        critica++;
        critica--;
        wantQ = false
  od
}
```

```
1  bool wantP = false, wantQ = false;
2
3  active proctype P() {
4   do :: wantP = true;
5         !wantQ;
6         wantP = false
7   od
8  }
9
10 active proctype Q() {
11  do :: wantQ = true;
12        !wantP;
13        wantQ = false
14   od
15 }
```




```
$ spin -a -f '![]esclusione' 24.pml
$ gcc -DSAFETY -o pan pan.c
$ ./pan
```

minuscola

# es. spin_24: mutua esclusione (modo 2)

```
bool csP, csQ;
bool wantP = false, wantQ = false;

active proctype P() {
  do ::  wantP = true;
         !wantQ;
         csP = true;
         csP = false;
         wantP = false
  od
}

active proctype Q() {
  do ::  wantQ = true;
         !wantP;
         csQ = true;
         csQ = false;
         wantQ = false
  od
}
```

salvare la negazione della formula LTL in un file

24.prp

```
1   ![]!(csP && csQ)
```

includere quel file nella generazione del verificatore con **-F**

```
$ spin -a -F 24.prp 24.pml
$ gcc -DSAFETY -o pan pan.c
$ ./pan
```

# es. spin_24: mutua esclusione (modo 3)

```
bool csP, csQ;
bool wantP = false, wantQ = false;

active proctype P() {
  do ::  wantP = true;
         !wantQ;
         csP = true;
         csP = false;
         wantP = false
  od
}

active proctype Q() {
  do ::  wantQ = true;
         !wantP;
         csQ = true;
         csQ = false;
         wantQ = false
  od
}
```

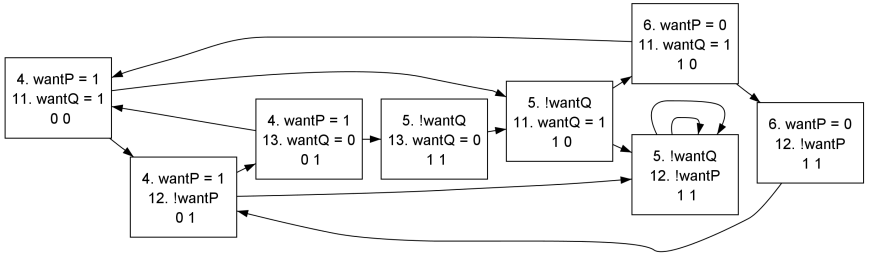
tradurre la formula LTL in “never claim” e sistemarla in un file

```
$ spin -a -f '![]!(csP && csQ)' > 24.ltl
$ spin -a -N 24.ltl 24.pml
$ gcc -DSAFETY -o pan pan.c
$ ./pan
```

# es. spin_23b: senza variabili fantasma

```
bool wantP = false, wantQ = false;

#define esclusione !(P@cs && Q@cs)

active proctype P() {
  do :: wantP = true;
        !wantQ;


cs:     wantP = false
  od
}

active proctype Q() {
  do :: wantQ = true;
        !wantP;


cs:     wantQ = false
  od
}
```

```
1  bool wantP = false, wantQ = false;
2
3  active proctype P() {
4   do :: wantP = true;
5         !wantQ;
6         wantP = false
7   od
8  }
9
10 active proctype Q() {
11  do :: wantQ = true;
12        !wantP;
13        wantQ = false
14   od
15 }
```




```
$ spin -a -f '![]esclusione' 23b.pml
$ gcc -DSAFETY -o pan pan.c
$ ./pan
```

# liveness

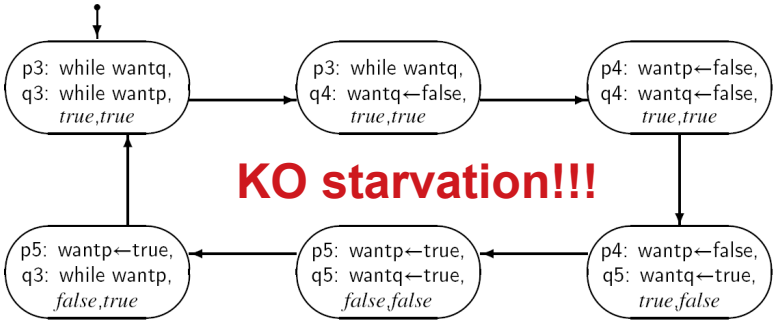
- **<>** **qualcosa di buono**
- **!<>** **qualcosa di cattivo**
- **qualcosa di** deve essere espresso in forma di logica proposizionale o in forma di logica (modale) temporale (LTL)
- procedimento:

1) esprimere formalmente il **qualcosa di**
2) aggiungere la ***negazione*** del **<>** **qualcosa di** al comando per generare il verificatore
3) compilare il verificatore
4) lanciare il verificatore con l'opzione **-a**

# es. spin_25: assenza starvation

```
bool wantP = false, wantQ = false;
bool csp = false;
active proctype P() {
  do
  :: wantP = true;
     do
     :: wantQ -> wantP = false; wantP = true
     :: else -> break
     od;
  csp = false;
  csp = true;
  wantP = false
od
}
active proctype Q() {
  do
  :: wantQ = true;
    do
    :: wantP -> wantQ = false; wantQ = true
    :: else -> break
    od;
    wantQ = false
  od
}
```

| boolean wantp ← false, wantq ← false | | | |
|---|---|---|---|
| **p** | | **q** | |
| | loop forever | | loop forever |
| p1: | non-critical section | q1: | non-critical section |
| p2: | wantp ← true | q2: | wantq ← true |
| p3: | while wantq | q3: | while wantp |
| p4: | wantp ← false | q4: | wantq ← false |
| p5: | wantp ← true | q5: | wantq ← true |
| p6: | critical section | q6: | critical section |
| p7: | wantp ← false | q7: | wantq ← false |




```
$ spin -a -f '!<>csp' 25.pml
$ gcc -o pan pan.c
$ ./pan -a -f
```

fairness

acceptance

# es. spin_26: terminazione e weak fairness

```
int n = 0;
bool flag = false;

active proctype p() {
  do
  ::  flag -> break;
  ::  else -> n = 1 - n;
  od
}

active proctype q() {
  flag = true
}
```

```
$ spin -a -f '!<>flag' 26.pml
$ gcc -o pan pan.c
$ ./pan -a -f
```

fairness

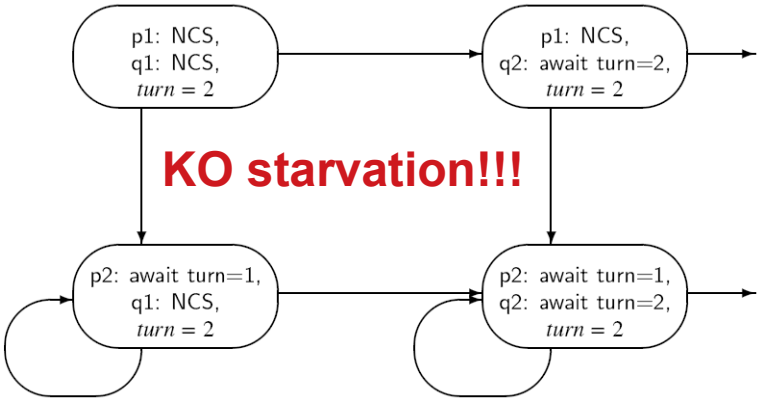
acceptance

# es. spin_27: sezione NON critica (ok ME)

```
# define esclusione !(P@cs && Q@cs)
# define spesso Q@cs

byte   turno = 1;

active proctype P() {
  do
  :: if  /* SNC può NON evolvere */
     :: true
     :: true -> false
     fi;
     turno == 1;
cs:  turno = 2
  od
}

active proctype Q() {
  do
  :: turno == 2;
cs:  turno = 1
  od
}
```

integer turn ← 1

| p | | q | |
|---|---|---|---|
| | loop forever | | loop forever |
| p1: | non-critical section | q1: | non-critical section |
| p2: | await turn = 1 | q2: | await turn = 2 |
| p3: | critical section | q3: | critical section |
| p4: | turn ← 2 | q4: | turn ← 1 |



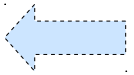

```
$ spin -a -f '![]esclusione' 27.pml
$ gcc -DSAFETY -o pan pan.c
$ ./pan
$ spin -a -f '![]<>spesso' 27.pml
$ gcc -o pan pan.c
$ ./pan -a -f
```

# es. spin_28: algoritmo di Dekker (no starvation)

```
#define spesso Q@cs
bool wantP = false, wantQ = false;
byte turno = 1;
active proctype P() {
 do
 :: if  /* SNC può NON evolvere */
       :: true
       :: true -> false
    fi;
    wantP = true;
    do
    :: wantQ ->
       if
       :: (turno == 2) -> wantP = false;
                          turno == 1;
                          wantP = true
       :: else -> skip
       fi
    :: else -> break
    od
cs: turno = 2;
    wantP = false
 od
}
active proctype Q() {
 do
 :: wantQ = true;
    do
    :: wantP ->
       if
       :: (turno == 1) -> wantQ = false;
                          turno == 2;
                          wantQ = true
       :: else -> skip
       fi
    :: else -> break
    od
cs: turno = 1;
 wantQ = false
 od
}
```

boolean wantp ← false, wantq ← false
integer turn ← 1

| p | | q | |
|---|---|---|---|
| | loop forever | | loop forever |
| p1: | non-critical section | q1: | non-critical section |
| p2: | wantp ← true | q2: | wantq ← true |
| p3: | while wantq | q3: | while wantp |
| p4: | if turn = 2 | q4: | if turn = 1 |
| p5: | wantp ← false | q5: | wantq ← false |
| p6: | await turn = 1 | q6: | await turn = 2 |
| p7: | wantp ← true | q7: | wantq ← true |
| p8: | critical section | q8: | critical section |
| p9: | turn ← 2 | q9: | turn ← 1 |
| p10: | wantp ← false | q10: | wantq ← false |



```
$ spin -a  -f  '![]<>spesso'  28.pml
$ gcc  -o pan pan.c
$ ./pan -a  -f
```