

```

// *****
// SPECIFICA DEI SEMAFORI
// *****
semaphore s = i; // i >= 0

void P(semaphore s):
    region s << when (vals > 0) vals--; >>

void V(semaphore s):
    region s << vals++; >>

// *****
// PROPRIETA' DEI SEMAFORI
// *****
// Costanti utilizzate
int I;          // Valore >= 0 con cui il semaforo è inizializzato
int nv,        // Numero di volte che l'operazione V è stata eseguita
int ni;        // Numero di volte che l'operazione P è stata INIZIATA
int np;        // Numero di volte che l'operazione P è stata COMPLETATA
int bloccati;  // Numero di processi sospesi sul semaforo

// #1: Relazione tra valore di semaforo e chiamate alle operazioni
val = I + nv - np;

// #2: Relazione di invarianza (derivata da #1 e da val >= 0)
np <= I + nv;

// #3: Relazioni derivate
bloccati = ni - np;
val > 0 => bloccati == 0;
bloccati > 0 => val == 0;

// *****
// Semafori di mutua esclusione
// *****
semaphore mutex = 1;

void op1()
{
    P(mutex);
    << corpo della funzione >>;
    V(mutex);
}
void op2()
{
    P(mutex);
    << corpo della funzione >>;
    V(mutex);
}

// *****
// PROPRIETA' MUTEX
// *****
// Proprietà 1: C'è al massimo un processo nella sezione critica
Th: 0 <= Nsez <= 1;

Hp: I = 1; // Semaforo inizializzato a 1
Tutti hanno come prologo P(mutex)

```

Tutti hanno come epilogo V(mutex)

```
Dim: Nsez = np - nv;           // Direttamente dalle ipotesi
      val = I + nv - np >= 0;  // Proprietà semafori
      1 + nv - np >= 0;
      1 >= np - nv;
      1 >= Nsez;

      // Inoltre il protocollo prevede sempre una P e una V
      np >= nv;
      np - nv >= 0;
      Nsez >= 0;

      // Unendo le due...
      0 <= Nsez <= 1;         // C.V.D.
```

// Proprietà 2: Assenza di deadlock

```
Th: Non esiste uno stato in cui: val = 0 && Nsez == 0; // Condizione di deadlock
```

```
Hp: I = 1; // Semaforo inizializzato a 1
Tutti hanno come prologo P(mutex)
Tutti hanno come epilogo V(mutex)
```

```
Dim: // Per assurdo, suppongo vera la condizione di deadlock
      val = I + nv - np;           // Proprietà semafori
      0 = 1 + nv - np;           // Da prima condizione di deadlock
      Nsez == 0 => np - nv == 0 => np = nv; // Proprietà 1
      0 = 1 + 0;                 // ASSURDO!
```

// Proprietà 2: Se il mutex è libero, non ci sono processi in attesa e il prossimo potrà entrare

```
Th: Nsez == 0 => bloccati == 0 && val > 0;
```

```
Hp: I = 1; // Semaforo inizializzato a 1
Tutti hanno come prologo P(mutex)
Tutti hanno come epilogo V(mutex)
```

```
Dim: Nsez == 0 => np = nv;
      // Supponiamo per assurdo che val == 0
      val = I + nv - np;
      0 = 1 + 0; // ASSURDO!
```

```
// *****
// Semafori evento
// *****
// Esempio: op1 esegue solo dopo op2
semaphore event = 0;
```

```
void op1()
{
    P(event);
    << corpo della funzione >>;
}
void op2()
{
    << corpo della funzione >>;
    V(event);
}
```

```

// *****
// Semafori binari composti
// *****
// Esempio: buffer di dimensione 1 condiviso
semaphore vuoto = 1;
semaphore pieno = 0;

void invio(T dato)
{
    P(vuoto);
    << corpo della funzione >>;
    V(pieno);
}

T ricezione()
{
    P(pieno);
    << corpo della funzione >>;
    V(vuoto);
}

// *****
// Semafori condizione
// *****
// op1() : region R << when(C) S1; >> - Modello con attesa circolare
semaphore mutex = 1;
semaphore sem = 0;
int csem = 0;

void op1()
{
    P(mutex);
    while (!C)
    {
        csem++;
        V(mutex);
        P(sem);
        P(mutex);
    }
    S1;
    V(mutex);
}

void op2()
{
    P(mutex);
    S2;
    if (csem > 0 && C)
    {
        csem--;
        V(sem);
    }
    V(mutex);
}

// op1() : region R << when(C) S1; >> - Modello con passaggio di testimone
// Più efficiente ma risveglia solo un processo per volta

```

```
semaphore mutex = 1;
semaphore sem = 0;
int csem = 0;
```

```
void op1()
{
    P(mutex);
    if (!C)
    {
        csem++;
        V(mutex);
        P(sem);
        csem--;
    }
    S1;
    V(mutex);
}
```

```
void op2()
{
    P(mutex);
    S2;
    if (csem > 0 && C)
        V(sem);
    else
        V(mutex);
}
```