Le regioni critiche condizionali

Un costrutto per la programmazione concorrente è dato anche dalle **regioni critiche**, o **regioni critiche condizionali**, che sono implementate secondo uno schema come il seguente:

```
/*--------------------------------------------------------------------
 * v di tipo opportuno in base alla natura del problema
 *-------------------------------------------------------------------*/
{
   tipo v;    . . .
   region v when (condizione) istruzione;
   . . .
}
```

La variabile *v*, di tipo opportuno a seconda della natura del problema, è condivisa da diversi processi; inoltre è accessibile solo tramite le istruzioni del costrutto **region**. Queste istruzioni sono sempre eseguite in mutua esclusione.
Con le *regioni critiche* si possono risolvere in modo semplice i problemi che riguardano la sola *mutua esclusione*: basta infatti inserire in un costrutto **region** il codice da eseguire in mutua esclusione.
Per esempio il problema dell'incremento concorrente di *posti*, discusso nei precedenti paragrafi, può essere risolto nel seguente modo:

```
/*--------------------------------------------------------------------
 * Mutua esclusione con le regioni critiche condizionali
 *-------------------------------------------------------------------*/
class  RegioniCriticheMutuaEsclusione
{
   static int posti = 0;

/*--------------------------------------------------------------------
 * Ogni processo che deve incrementare posti esegue
 *-------------------------------------------------------------------*/
   void P()
   {
      . . .
      region posti when (true) posti++;
      . . .
   }
```

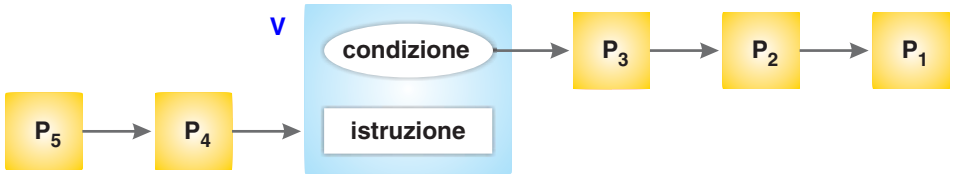
posti: variabile condivisa dai processi
Le istruzioni del costrutto **region** sono sempre eseguite in mutua esclusione

I processi che vogliono aggiornare *posti* hanno la garanzia che l'incremento sarà eseguito in mutua esclusione.
Esaminiamo in dettaglio il comportamento del costrutto:

```
region v when (condizione) istruzione;
```

In esso *v* è una variabile, di tipo opportuno, condivisa tra i processi concorrenti e *condizione* è un'espressione logica. Ottenuto l'accesso alla regione, come prima cosa viene valutata *condizione*. Se *condizione* è vera, allora *istruzione* viene eseguita. Se invece *condizione* è falsa, il processo che ha avuto accesso alla regione critica la lascia ed è inserito in una coda di processi in attesa che *condizione* diventi vera. A quel punto il processo è inserito nella coda dei processi in attesa di eseguire *istruzione*.
In una *regione critica* ci possono essere quindi due code di processi in attesa: quelli che attendono di accedere alla regione critica e quelli che attendono che *condizione* diventi vera.

Nell'esempio in figura è descritta una regione critica *v* con i processi **P4** e **P5** in attesa di accedere a *v* e i processi **P1**, **P2**, **P3** in attesa che *condizione* diventi vera. Quando questo avviene, essi sono inseriti nella stessa coda di **P4** e **P5**.



La sincronizzazione tra processi si risolve sfruttando la possibilità di subordinare l'esecuzione di *istruzione* al valore di verità di *condizione*, secondo lo schema del seguente codice, dove **P1** è un processo che si vuole sincronizzare su di un evento rilevato da **P0**.

```
/*-------------------------------------------------------------------
 * Sincronizzazione con le regioni critiche condizionali
 *-------------------------------------------------------------------*/
class  RegioniCriticheSincronizzazione
{
   static boolean evento = false;
   . . .
/*-------------------------------------------------------------------
 * P0 è il processo che rileva l'evento
 *-------------------------------------------------------------------*/
   void P0()
   {
      . . .
      RilevaEvento;
      region evento when (true) evento = true;
      . . .
   }
/*-------------------------------------------------------------------
 * P1 è il processo che si sincronizza sull'evento
 *-------------------------------------------------------------------*/
   void P1()
   {
      . . .
      region evento when (evento) EseguiAzione;
      . . .
   }
```

evento: variabile condivisa dai processi
P0: il processo che rileva l'evento
P1: il processo che si sincronizza sull'evento