Moniteurs

— preuve de programmes —

Preuve des programmes avec moniteurs

Composantes de la preuve :

- preuve des processus concurrents
- preuve du moniteur
- lien des preuves par la règle d'inférence pour l'appel de procédure

Preuve du moniteur basée sur l'invariant MI (à proposer).

- pour l'initialisation :

$$\{\text{true}\} \text{ initialisation } \{MI\}$$

- pour chaque procédure op ayant la pré-condition IN et la post-condition OUT :

$$\{IN \wedge MI\} \text{ corps } op \ \{OUT \wedge MI\}$$

- axiome **wait** avec L état des variables locales :

$$\{L \wedge MI\} \ \textbf{wait}(cv) \ \{L \wedge MI\}$$

- axiome **signal** (équivalent à **skip**) :

$$\{P\} \ \textbf{signal}(cv) \ \{P\}$$

Règle d'inférence pour l'appel de procédure

call op (**in** a, **inout** b, **out** c) avec x, y, z paramètres formels :

$$\frac{\begin{array}{l}\{IN\}\ \text{corps } op\ \{OUT\}\\ IN \text{ concerne que les paramètres formels passe par valeur}\\ OUT \text{ concerne que les paramètres formels passe par reference}\\ \text{b et c sont distincts}\\ PR \text{ ne concerne pas les résultats}\\ \quad\text{ou les variables changées dans } op\end{array}}{\{PR \wedge IN^{x,y}_{a,b}\}\textbf{call } op\ (in\ a, inout\ b, out\ c)\{PR \wedge OUT^{y,z}_{b,c}\}}$$

Exemple :

```
procedure swap(inout int x,y) {
  { x = X && y = Y }
  int t;
  t = x; x = y; y = t
  { x = Y && y = X }
}
...
{ a=1 && b=2 && c=3 }
call swap(a,b)
{ a=2 && b=1 && c=3 }
```


Règle d'inférence pour la synchronisation par moniteurs

Soient :

- un programme concurrent comportant $S_1, \ldots, S_n$ processus et $M_1, \ldots, M_m$ moniteurs,
- tous les processus interagissent uniquement par les moniteurs,
- toutes les variables partagées sont contrôlées par des moniteurs.

Alors la règle d'inférence suivante donne la sémantique du programme concurrent :

$$\frac{\begin{array}{l}\{P_i\}\ S_i\ \{Q_i\}\quad 1 \leq i \leq n\\ \text{aucune variable libre en } P_i, Q_i \text{ est changée en } S_j, j \neq i\\ \text{toutes les variables en } MI_k \text{ sont locales à } M_k\end{array}}{\begin{array}{l}\{MI_1 \wedge \ldots \wedge MI_m \wedge P_1 \wedge \ldots \wedge P_n\}\\ \textbf{parbegin } S_1 \mid\mid \ldots \mid\mid S_n \textbf{ parend}\\ \{MI_1 \wedge \ldots \wedge MI_m \wedge Q_1 \wedge \ldots \wedge Q_n\}\end{array}}$$

Preuve d'exclusion mutuelle avec moniteurs

Soient :

- S_1 et S_2 deux instructions dans des processus différents d'un programme concurrent.
- Ni S_1, ni S_2 sont dans un moniteur.
- $pre(S_1)$ et $pre(S_2)$ sont des pré-conditions dans une preuve du programme.
- $MI_1 \wedge ... \wedge MI_m$ sont les invariants des moniteurs qui apparaissent dans le programme et dans la preuve.
- $P_1 \wedge ... \wedge P_n$ est satisfait à l'état initial du programme.

Alors S_1 et S_2 sont en exclusion mutuelle si :

$$pre(S_1) \wedge pre(S_2) \wedge MI_1 \wedge ... \wedge MI_m = \text{false}$$

Preuve d'absence de blocage avec moniteurs

Soient :

- Un programme concurrent ayant n processus parallèles et m moniteurs.
- Les procédures des moniteurs ne font pas d'appel récursif vers les procédures d'autres moniteurs.
- $MI_1 \wedge ... \wedge MI_m$ sont les invariants des moniteurs qui apparaissent dans le programme et dans la preuve.
- $P_1 \wedge ... \wedge P_n$ est satisfait à l'état initial du programme.
- D_1 est le prédicat qui caractérise l'état où tout processus est fini ou bloqué.
- D_2 est le prédicat qui caractérise l'état où au moins un processus est bloqué.

Alors le programme est sans blocage si :

$$D_1 \wedge D_2 \wedge MI_1 \wedge ... \wedge MI_m = \text{false}$$

Exemple : Lecteurs et écrivains

Solution avec moniteur pour le contrôle de l'accès

```
process Reader(int i) {
  do :: true ->

        RW_Control.req_read();

        /* lecture donnee */
        RW_Control.rel_read();
  od
}

process Writer(int j) {
  do :: true ->

        RW_Control.req_write();

        /* ecriture donnee */
        RW_Control.rel_write();
  od
}
```



```
moniteur RW_Control {
  /* RW : (nr==0 || nw==0) && nw<=1 */
  int nr = 0, nw=0;
  cond okread;  /* signale nw==0 */
  cond okwrite; /* signale nr==0 && nw==0 */
  procedure req_read() {
    do :: nw>0 -> wait(okread) :: else -> break od;
    nr=nr+1
  }
  procedure rel_read() {
    nr=nr-1;
    if :: nr==0 -> signal(okwrite) :: else -> skip fi
  }
  procedure req_write() {
    do :: nw>0 || nr>0 -> wait(okwrite)
                :: else -> break
    od;
    nw=nw+1
  }
  procedure rel_write() {
    nw=nw-1;
    signal(okwrite); signal_all(okread)
  }
}
```

Preuve du programme

Utilise des variables auxiliares pour enregistrer l'état d'exécution de chaque processus :

- read[i] est 1 si Reader(i) lit, 0 sinon
- write[j] est 1 si Writer(j) écrit, 0 sinon

et qui seront passées en paramètre et écrites par les procédures du moniteur.

Lien entre les variables auxiliaires et les variables du moniteur :

$$
\begin{aligned}
VA: \quad & (nr == read[0] + ... + read[N-1]) \;\; \wedge \\
& (nw == write[0] + ... + write[M-1])
\end{aligned}
$$


```
int read[N] = ([N] 0), write[M] = ([M] 0);
process Reader(int i) {
  do :: true ->
        /* read[i] == 0 */
        RW_Control.req_read(read[i]);
        /* read[i] == 1 */
        /* lecture donnee */
        RW_Control.rel_read(read[i]);
  od
}

process Writer(int j) {
  do :: true ->
        /* write[j] == 0 */
        RW_Control.req_write(write[j]);
        /* write[j] == 1 */
        /* ecriture donnee */
        RW_Control.rel_write(write[j]);
  od
}
```

```
moniteur RW_Control {
  /* RW && VA */
  int nr = 0, nw=0;
  cond okread;   /* signale nw==0 */
  cond okwrite; /* signale nr==0 && nw==0 */
  procedure req_read(inout int r)
  { /* RW && VA && r==0 */
    do :: nw>0 -> wait(okread) :: else -> break od;
    nr=nr+1; r=1
  } /* RW && VA && r==1 */
  procedure rel_read(inout int r)
  { /* RW && VA && r==1 */
    nr=nr-1; r=0;
    if :: nr==0 -> signal(okwrite) :: else -> skip fi
  } /* RW && VA && r==0 */
  procedure req_write(inout int w)
  { /* RW && VA && w==0 */
    do :: nw>0 || nr>0 -> wait(okwrite)
                :: else -> break
    od;
    nw=nw+1; w=1
  } /* RW && VA && w==1 */
  procedure rel_write(inout int w)
  { /* RW && VA && w==1 */
    nw=nw-1; w=0;
    signal(okwrite); signal_all(okread)
  } /* RW && VA && w==0 */
}
```


Preuve d'exclusion mutuelle

- entre lecteur et écrivain :

$$(read[i] = 1 \wedge write[j] = 1 \wedge RW \wedge VA) = faux$$

- entre écrivains :

$$(write[j] = 1 \wedge write[k] = 1 \wedge j \neq k \wedge RW \wedge VA) = fa$$

Preuve d'absence d'interblocage

Etat d'interblocage : tous les processus (Reader et Writer) sont bloqués :

- Reader[i] peut être bloqué que sur wait(okread), dont la pré-condition est :

$$D_2 : RW \;\wedge\; VA \;\wedge\; read[i] = 0 \;\wedge\; nw > 0$$

- Writer[j] peut être bloqué que sur wait(okwrite), dont la pré-condition est :

$$D_2' : RW \wedge VA \wedge write[j] = 0 \wedge (nw > 0 \vee nr > 0)$$

Mais :

$$(D_2 \wedge D_2' \wedge RW \wedge VA) = faux$$