Université Paris 7
U.F.R d'informatique

DESS Logiciels Fondamentaux
Année 2002-2003

Méthodes formelles pour la vérification

PROJET

Algorithme de Franklin

1 Description du travail demande

On s'intéresse à la vérification d'un algorithme d'élection de leader dans un anneau. Vous aurez pour cela à utiliser Spin dans un premier temps, puis SMV. Le protocole permet aux processus de communiquer par des canaux (structure qui est intégrée dans Spin et pas dans SMV). Nous vous demanderons donc d'étudier comme un problème supplémentaire la modélisation de canaux bornés dans le cas général, puis d'utiliser cette modélisation lors de votre vérification avec SMV.

Le travail pourra être fait en binômes ou seul. Il donnera lieu à une soutenance, et vous rendrez un rapport.

N'hésitez pas à comparer les deux outils, augmentez le nombre de processus, les domaines des identités etc ...

2 L'algorithme de Franklin

Au départ les processus d'un anneau sont tous considérés comme actifs ou passifs, sachant qu'il y en a au moins un d'actif. Ils ont des identités distinctes décrites par un entier, et une position dans l'anneau. Peu à peu ils seront désactivés pour que finalement il n'en reste qu'un seul, le leader. Les processus désactivés continueront à relayer les messages.

L'idée de l'algorithme est la suivante : chaque processus actif compare localement son identité à celle du premier processus actif qui le précède dans l'anneau ainsi qu'à celle du premier processus actif qui le suit dans l'anneau. S'il est le minimum local, il reste actif ; sinon, il devient passif. L'algorithme termine quand un processus détecte qu'il est le leader, et que les autres recoivent un signal d'arret.

```
VAR
    etat:{actif, passif};
    resultat:{inconnu, gagne, perdu};
```

```
BEGIN
resultat:=inconnu;
REPEAT
    SI (etat=actif) ALORS
        BEGIN
        envoi_devant(<jet,mon_id>,id_suiv);
        recoit_derriere(msg_der);
        envoi_derriere(<jet,mon_id>,id_pred);
        recoit_devant(msg_dev);
        SI ( msg_der=<jet,p> ET msg_dev=<jet,q> ) ALORS
            BEGIN
            SI p=mon_id ALORS resultat:=chef;
            SI ( p<mon_id OU q<mon_id ) ALORS etat:=passif;
            END
        END
    SINON
        BEGIN
        SI (resultat=inconnu) ALORS resultat:=perdu;
        SI recoit_devant(msg_dev) ALORS envoi_derriere(msg_dev);
        SI recoit_derriere(msg_der) ALORS envoi_devant(msg_der);
        END
UNTIL ( resultat=chef OU msg_der=<stop> )
SI (resultat=chef) ALORS
    BEGIN
    envoi_devant(<stop>,id_suiv);
    recoit_derriere(<stop>);
    END
END
```

La communication se fait de manière asynchrone et bidirectionnelle sur des canaux de longueur 1 plaçés entre les process. Pour chaque processus celui qui se trouve “devant” lui est son successeur et celui qui se trouve “derrière” est son prédécesseur.

3 Modélisation - Spin

En respectant l'atomicité, décrire le protocole à l'aide d'un automate, puis en Spin pour un anneau de 5 processus dans lequel il y a au moins un processus actif et au moins un processus passif.

Validez votre modèle, qui doit satisfaire entre autre les propriétés suivantes :

1. un processus n'écrit jamais sur un canal quand ce dernier est plein ;
2. un processus ne lit jamais de message sur un canal vide ;
3. si un processus lit un message sur un canal, ce dernier se vide ;
4. si un processus écrit un message sur un canal, ce dernier est correctement rempli ;
5. si un processus envoie un message, son successeur finit toujours par le recevoir.
6. un processus passif perd toujours l'élection ;
7. un processus actif qui reçoit un jeton marqué avec une identité inférieure à la sienne devient passif ;
8. du moment qu'il est passif, un processus n'envoie pas de jeton marqué avec son identité ;
9. un processus actif ne passe dans l'état `résultat=chef` que s'il reçoit son propre jeton.
10. chaque processus n'envoie qu'une seule fois un signal $<$ **stop** $>$; (vous devrez peut être introduire un compteur)
11. chaque processus ne reçoit qu'une seule fois $<$ **stop** $>$
12. l'algorithme termine ;
13. - sûreté- deux processus ne sont jamais élu chef en même temps ;
14. - vivacité - on finit toujours par élire un chef ;
15. - monotonie - une fois le chef élu, l'identité de ce dernier ne change pas ;
16. - minimalité du chef - l'élection désigne le processus d'identité minimale parmi ceux qui sont actifs à l'initialisation de l'algorithme.

4 Modélisation des canaux en SMV

Un canal peut être vu comme un processus qui communique avec l'extérieur par l'intermédiaire d'un certain nombre de variables qui sont publiques pour les processus qui sont à ses extrémités. En interne il possède une mémoire finie qu'il gère avec une certaine logique (Fifo par exemple).

Nous nous intéressons à des modèles de canaux dont la taille est inférieure à (disons)4. Les modules utilisateurs du canal interagissent avec lui grâce aux variables publiques : ready (vraie si le canal peut être utilisé), nempty (vraie si le canal n'est pas vide), nfull (vraie si le canal n'est pas plein), get (un sémaphore pour la lecture d'un message) – put (idem pour la dépose un message), valget (variable contenant le message à lire), valput (variable recevant le message à écrire).

Avec cette interface, on lit un message dans le canal et on le range dans la variable x par une transition *si ready et nempty alors get := 1 ; x := valget*. Et pour écrire le message m dans le canal : *si ready et nfull alors put := 1 ; valput := m*

Modélisez un canal, et utilisez le dans un protocole élémentaire pour vérifier sa correction. (Donner un nombre de spécifications significatif)

5 Franklin en SMV

Effectuer la vérification de l'algorithme de Franklin à l'aide de SMV.