SE206 - TP UPPAAL : introduction au model checking

II. Algorithme de Fischer

Question 6

Cf. `fischer-tpt_P2.xml`

Question 7

Avant correction : delay = 3

La propriété vérifiant que deux processus distincts ne peuvent pas être tous les deux dans la même section n'est pas vérifiée.

Le problème réside dans le fait que lorsque P1 entre en section critique (L4toLk), P2 passe en état L2 juste après puisque has_cs prend la valeur 2. Ainsi, quelques tours d'horloge plus tard P2 entre en section critique puisqu'aucun test ne l'en a empêché. On a donc P1 et P2 tous deux en section critique.

Pour corriger le problème, il suffit d'empêcher P1 d'entrer en section critique si P2 est sur le point de passer à l'état L2 et donc de mettre à jour has*cs. On augmente donc le delay de 1 pour permettre à P1 d'arriver un tour plus tard à l'état L3. Ainsi, P2 mettra à jour has*cs à 2 avant que P1 n'entre en section critique. P1 reviendra donc à l'état initial via L3goto puisque has_cs!=1 (il vient d'être mis à jour par P2).

Question 8

Il faut que le delay dépasse le temps d'exécution d'une instruction. Une fois que l'on paramètre delay de cette façon, la propriété est vérifiée.

III. Problème : preuve d'équité

Pour modéliser while(1), on ajoute une transition entre Lkplus et L0 en passant par L*wait. (cf.* `fischer-tpt_P3.xml`)

Propriété d'équité de l'algorithme de synchronisation

Cette propriété n'est pas vérifiée puisqu'un processus peut reprendre la main après être sorti de sa section critique, ne laissant pas la main à l'autre processus. Un autre processus pourrait alors ne jamais avoir la main, ce qui est une preuve de la non équité de l'algorithme. (Cf. Propriété d'équité du `fischer-tpt_P3.q`)

Ajout d'une borne sur le temps passé en section critique

Si le processus P1 contient une boucle while(1) et qu'il est dans sa section critique alors le processus P2 ne pourra peut-être jamais prendre la main. Il faut donc rajouter une borne sur le temps passé en section critiquer pour éviter cela.

Ajout d'une contrainte de temps pour le processus sorti de section critique et retournant à L0

Il faut rajouter un état d'attente pour le processus qui vient de sortir de la section critique pour laisser le temps à l'autre processus de reprendre la main. Cette contrainte sur le retour à l'état L0 du processus venant de sortir de sa section critique est nécessaire pour que chaque processus prenne la main l'un après l'autre équitablement.

Faisons une étude pour déterminer le temps pendant lequel le processus sortant de sa section critique doit attendre avant de repasser à l'état L0 Dans le pire des cas, P2 se trouve en L0 et P1 se trouve en section critique. P2 reste au maximum 4 cycles en L1 (exec<C) puis au maximum 4 cycles en L2 (exec<C) puis 4 cycles de waiting avant d'arriver à L3, puis au maximum 4 cycles en L3 (exec<C).

Ces modifications ne fonctionnent toujours pas avec un delay_2 valant 16. Pour résoudre le problème de la trace de simulation infinie en temps nul de L0 à L0goto et de L0goto à L0 révélée par le contre-exemple, il faut forcer à ce que l'exécution de L0 à L0goto à L0 se fasse en une unité de temps par exemple, d'où le exec>=1.

S'il y a plus de 2 processus, par exemple 3, le processus P3 pourrait ne jamais prendre la main puisque notre modélisation fonctionne en donnant alternativement la main à P1 et à P2. Le nombre de processus en exclusion mutuelle est donc ici limité. Pour corriger cela, il faudrait ajouter d'autres compteurs pour donner la main quand il le faut aux processus qui n'ont encore jamais eu la main.