Université catholique de louvain-la-neuve

INGI2143 - Concurrent Systems

Assignment 2 : Verification Phase

Jos Zigabe

Scott Ivinza Mbe

23 novembre 2015

1 Introduction

Dans le cadre du cours de *Concurrent Systems : models and analysis*, il nous a été demandé pour la seconde phase du projet de faire une vérification de la technique de *token-passing* avec le *protocole d'élection du leader* en *FSP*(Finite State Process) et *FLTL*(Fluent Linear Temporal Logic), modélisé lors de la première phase. Etant donné que nous avions commis quelques erreurs de modélisations lors de la première phase, comme de ne pa avoir créé un processus pour chaque composant (NODE et CHANNEL). Il a fallu que nous effectuons quelques modifications sur notre token-passing protocol avant d'entamer la seconde phase. Ainsi, en plus des fichiers (.lst) de la phase précedente, nous avons également ajouté six autres fichiers concernant cette seconde phase. Ces fichiers sont les suivants :

ABSTRACT.lst définit une séquence de modèles minimaux en FSP forçant `ELECTIONRING` à se minimiser jusqu'à ce que le dernier modèle soit équivalent à `ELECTIONSERVICE`.

OPENCLOSE-NOMINALRING.lts définit une propriété `OPENCLOSE` sur `NOMINALRING` qui capture les séquences correctes des événements *open* et *close*.

OPENCLOSE-ELECTIONRING.lts est similaire à `OPENCLOSE-NOMINALRING` sauf qu'ici la propriété `OPENCLOSE` va agir sur `ELECTIONRING` et on y retrouve également une déclaration de progression afin de vérifier que chaque noeud peut acceder indéfiniment à la ressource avec `ELECTIONRING`.

FLTL.lts définitout d'abord une propriété FLTL `ALLFAIR` équivalente à la déclaration de progession du fichier `OPENCLOSE-ELECTIONRING`. Ce fichier définit ensuite une propriété FLTL `MUTEX` qui stipule que deux nœuds ne peuvent pas accéder à la ressource en même temps. Enfin, il définit une propriété FLTL `NODUPCLAIM` qui stipule qu'aucun nœud i peut envoyer une deuxième demande (i) avant qu'il ne reçoit en retour la première demande (i).

CRASHRING.lts définit un modèle qui est l'extention de `ELECTIONRING` tels que les noeuds peuvent crasher. On y retrouve également les propriètés FLTL `NOCRASH` et `FAIR` qui stipulent qu'un noeud i ne peut jamais crasher et peut indéfiniment acceder à la ressource, en plus des autres propriété du fichier `FLTL.lts`.

MUTEX-ELECTIONRING.lts définit une propriété FLTL `MUTEX` sur `ELECTIONRING`.

2 Minimal FSP models : ABSTRACT

Voici les alphabets de modèles `ELECTIONRING` et `ELECTIONRINGSERVICE` :

election : {{rcv_claim, rcv_token}[0..2][0..2], s[0..2].{close, open}, {snd_claim, snd_token}[0..2][0..2]}

service : s[0..2].{close, open}

Abstract	Alphabet	Minimized states	Time (ms)
0	e0 = election	5652	716
1	e1 = e0/{snd_token[0][1]}	5061	610
2	e2 = e1/{snd_token[1][2]}	4740	526
3	e3 = e2/{snd_token[2][0]}	4540	349
4	e4 = e3/{rcv_token[0][1]}	4369	315
5	e5 = e4/{rcv_token[1][2]}	4278	317
6	e6 = e5/{rcv_token[2][0]}	4077	273
7	e7 = e6{snd_claim[0][0]}	3868	255
8	e8 = e7/{snd_claim[0][1]}	3828	257
9	e9 = e8/{snd_claim[0][2]}	3520	211
10	e10 = e9/{snd_claim[1][0]}	3324	189
11	e11 = e10/{snd_claim[1][1]}	3324	192
12	e12 = e11/{snd_claim[1][2]}	3303	203
13	e13 = e12/ {snd_claim[2][0]}	3293	203
14	e14 = e13/{snd_claim[2][1]}	3293	194
15	e15 = e14/{snd_claim[2][2]}	3293	198
16	e16 = e15/{rcv_claim[0][0]}	3017	193
17	e17 = e16/{rcv_claim[0][1]}	2800	311
18	e18 = e17/{rcv_claim[0][2}	2296	277
19	e19 = e18/{rcv_claim[1][0]}	1693	532
20	e20 = e19/{rcv_claim[1][1]}	1693	519
21	e21 = e20/{rcv_claim[1][2]}	1234	189
22	e22 = e21/{rcv_claim[2][0]}	740	79
23	e23 = e22/{rcv_claim[2][1]}	740	81
24	e24 = e23/{rcv_claim[2][2}	740	75
ELECTIONSERVICE	service	/	/

Le temps total cumulé pour minimiser les abstracts avec 3 noeud est de 63.154 ms alors que l'on ne peut pas minimiser `ELECTIONSERVICE(3)` car la minimisation n'est pas faite après plus de 30 minutes.

3 Properties

3.1 Picture of the LTS of OPENCLOSE

On peut voir sur l'image que l'on rentre dans l'état *ERROR* une fois que l'on viole la propriété, c'est-à-dire lorsque plusieurs noeuds accedent à la ressource.

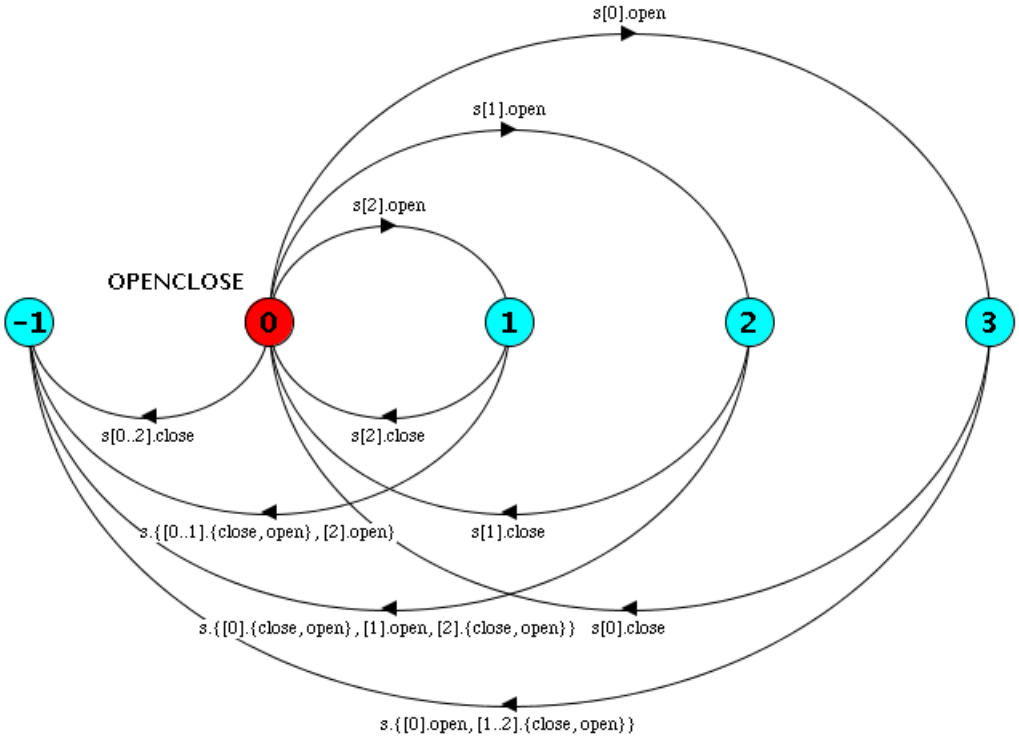

FIGURE 1 – Propriété OPENCLOSE

3.2 Violation trace for OPENCLOSE on ELECTIONRING

Voici la trace de violation de la propriété OPENCLOSE :

- snd_claim.0.1
- rcv_claim.0.1
- snd_claim.0.1
- snd_claim.0.2
- rcv_claim.0.1
- rcv_claim.0.2
- snd_claim.0.0
- rcv_claim.0.0
- snd_claim.0.2
- snd_token.0.1
- rcv_token.0.1
- snd_claim.0.1
- s.1.open
- rcv_claim.0.2

— snd_claim.0.0
— rcv_claim.0.0
— s.0.open
— ERROR

On remarque que le noeud 0 commence par envoyer un claim au noeud 1 et une fois que le noeud le reçoit, le noeud 0 renvoit un second claim[0][1] au noeud 1. Ensuite le premier claim lui revient ce qui fait de lui le leader, envoit immédiatement le token au noeud 1 et envoye un troisième claim afin d'être de nouveau eligible. Ensuite, le noeud 1 accede à la ressource comme il est le leader, cependant le noeud 0 recoit le second claim et devient à son tour le leader et accede à la ressource ce qui viole la propriété OPENCLOSE.

3.3 Violation trace for ALLFAIR on ELECTIONRING

Voici la trace de violation de la propriété `ALLFAIR` lorsqu'on désactive l'open $fair\ choice$:

— snd_claim.0.1
— rcv_claim.0.1
— snd_claim.0.1
— snd_claim.0.2
— rcv_claim.0.1
— snd_claim.0.1
— rcv_claim.0.2
— snd_claim.0.2
— rcv_claim.0.1
— snd_claim.0.0
— rcv_claim.0.0
— snd_token.0.1
— rcv_claim.0.2
— snd_claim.0.2
— rcv_token.0.1
— snd_claim.0.1
— snd_claim.0.0
— rcv_claim.0.0
— rcv_claim.0.2
— snd_token.1.2
— rcv_claim.0.1
— snd_token.0.1
— snd_claim.0.0
— rcv_token.1.2
— snd_claim.0.2
— rcv_token.0.1
— snd_claim.0.1
— rcv_claim.0.0
— snd_token.2.0
— rcv_claim.0.2
— snd_token.1.2
— rcv_claim.0.1
— snd_token.0.1
— rcv_token.2.0
— snd_claim.0.0
— rcv_token.1.2
— snd_claim.0.2
— rcv_token.0.1
— snd_token.0.1

— rcv_claim.0.0
— snd_token.2.0
— rcv_token.2.0
— rcv_claim.0.2
— snd_token.1.2
— rcv_token.0.1
— snd_token.0.1
— snd_claim.0.0
— rcv_token.1.2
— snd_token.1.2
— rcv_token.0.1
— snd_claim.0.1
— rcv_claim.0.0
— snd_token.2.0
— rcv_token.1.2
— snd_token.1.2
— rcv_claim.0.1
— snd_token.0.1
— rcv_token.2.0
— s.0.open
— OPENCLOSE.0
— s.0.close

Cycle :

— snd_token.2.0
— rcv_token.1.2
— send_laim.0.2
— rcv_token.0.1
— snd_token.0.1
— rcv_token.2.0
— snd_token.2.0
— rcv_claim.0.2
— snd_token.1.2
— rcv_token.0.1
— snd_token.0.1
— rcv_token.2.0
— snd_claim.0.0
— rcv_token.1.2
— snd_token.1.2
— rcv_token.0.1
— snd_token.0.1
— rcv_claim.0.0
— snd_token.2.0
— rcv_token.1.2
— snd_token.1.2
— rcv_token.0.1
— snd_claim.0.1
— rcv_token.2.0
— snd_token.2.0
— rcv_token.1.2
— snd_token.1.2
— rcv_claim.0.1

— snd_token.0.1
— rcv_token.2.0
— s.0.open
— OPENCLOSE.0
— s.0.close

Après une longue séquence d'actions, on remarque que l'on rentre dans un cycle et comme le noeud n'accède jamais à la ressource dans ce cylcle étant donné que le *fluent* OPENCLOSE n'est jamais vrai, la propriété `ALLFAIR` est violée car l' équité pour `open` du noeud 1 est perdue.

4 CRASHRING

4.1 Picture of the LTS of a node in CRASHRING(2)

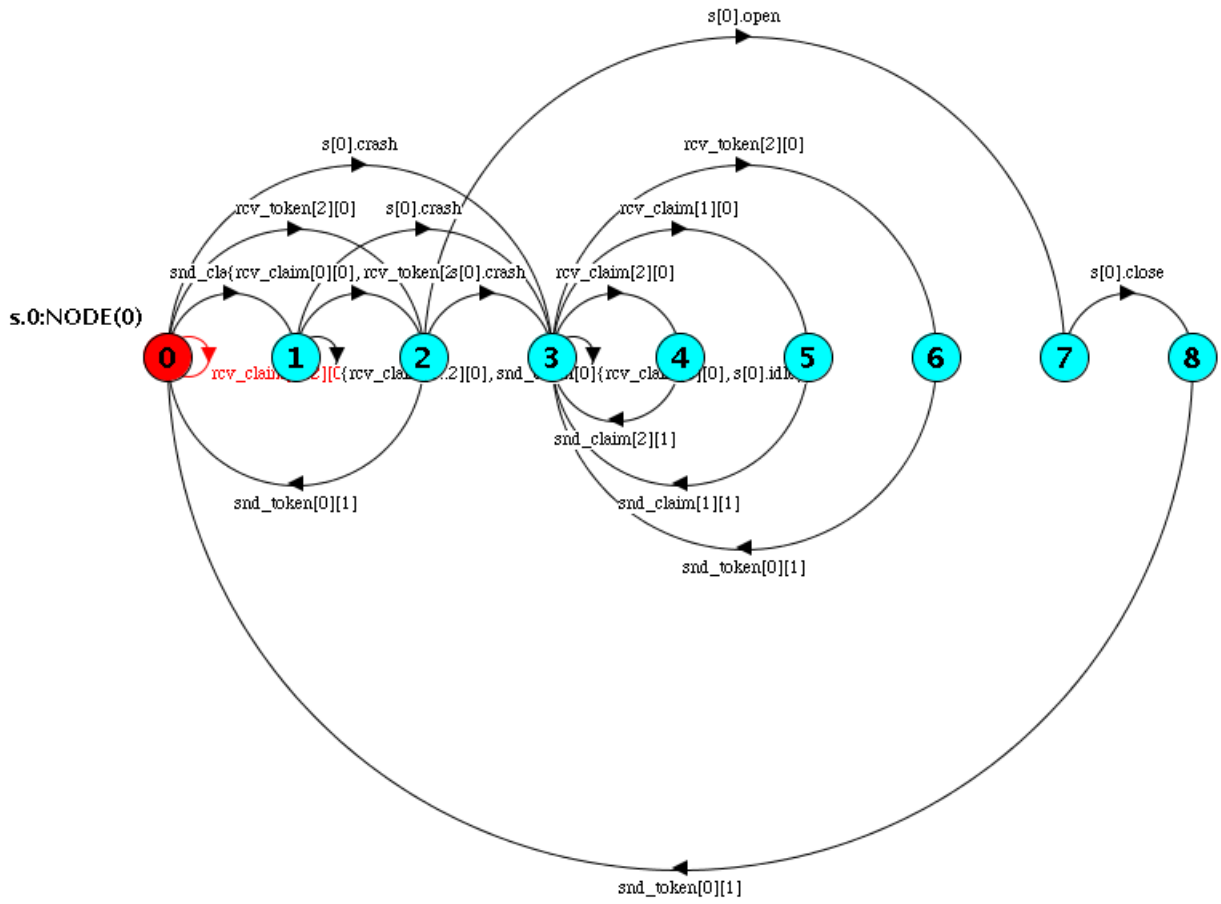

FIGURE 2 – Node in CRASHRING

Les états répresentant le crash sont 3 qui représente l'état une fois que le crash est se produit, 4 et 5 qui représentent les états intermédaires lorsque le noeud fait suivre un claim et un token.

4.2 Violation trace for ALLFAIR on CRASHRING(3)

Voici la trace de violation de la propriété `ALLFAIR` dans le modèle `CRASHRING` :

— s.0.crash
— s.1.crash
— s.2.crash

Cycle :

— s.0.idle

Lorsqu'un noeud crash il ne peut plus acceder à la ressource. Dans cette trace, on peut voir que les 3 noeuds plantent, ainsi aucun d'entre eux ne peut acceder à la ressource. La seule action possible pour ces noeud est de se mettre en pause ce qui viole la propriété `ALLFAIR`.

5 Tables and/or graphs showing the number of states, transitions, time and length of violation trace for checking MUTEX using ordinary model-checking on ELECTIONRING(N), without and with partial-order reduction, for N = 1 up to the largest model you can handle.

# of nodes	# of states	# of transitions	time[ms]	length of violation trace
2	105	193	2	13
3	1339	3434	4	17
4	15955	52093	34	21
5	223354	860008	826	25
6	3677214	16243809	31346	29

TABLE 1 – Tableau sans reduction d'ordre partiel

# of nodes	# of states	# of transitions	time[ms]	length of violation trace
2	100	174	1	13
3	1212	2626	9	17
4	13806	33428	32	21
5	178172	451911	713	25
6	2622651	6830174	21001	29

TABLE 2 – Tableau avec reduction d'ordre partiel

Commentaires :
Le premier tableau (Table 1) répresente le nombre d'états, le nombre de transitions, le temps(en milliseconde) et la longueur de violation trace sans la réduction d'ordre partiel et le second tableau(Table 2) répresente les mêmes valeurs cependant avec la reduction d'ordre partiel. De plus, nous observons que nous avons moins d'états, moins de transitions et la vérification est plus rapide avec la réduction d'ordre partiel. Ceci est du au fait que nous évitons d'explorer des éxécutions redondantes.

# of nodes	# of states	# of transitions	time[ms]	length of violation trace
2	35	70	0	35 (depth of 65)
3	108	257	0	108 (depth of 250)
4	209	564	3	198 (depth of 500)
5	199700	739093	433	186 (depth of 500)
6	2342919	9099617	6469	180 (depth of 500)

TABLE 3 – Tableau avec supertrace

6 Tables and/or graphs showing the number of states, transitions, time and length of violation trace for checking MUTEX using Supertrace (with 100 MB hashtable and search depth 500) on ELECTIONRING(N), for N = 1 up to the largest model you can handle.

Commentaires :
Lorsqu'on regarde dans la table 3, on peut voir que nous avons beaucoup moins d'états et de transitions (à l'exception des deux dernières longueurs) et c'est beaucoup plus rapide. La raison est que *supertrace* dans LTSA un bitstate hashing, il enregistre les états de manière compacte dans une hashtable.

7 Tables and/or graphs showing the number of states, transitions, time and length of violation trace for checking MUTEX using Supertrace on ELECTIONRING(5), for different values of the hashtable size (between 1 K and 100 M) and search depth (between 20 and 2000). Discuss how the length of the violation trace varies, and the values that lead to the shortest trace.

hashtable [kB]	depth	# of states	# of transitions	time[ms]	length of violation trace
1	20	1902	8226	5	No violation found
1	200	5269	19021	11	No violation found
1	1000	257	719	1	257 (depth of 704)
1	2000	257	719	1	257 (depth of 704)
100	20	1899	8223	5	No violation found
100	200	141453	483629	292	77 (depth of 200)
100	1000	354	966	4	354 (depth of 949)
100	2000	354	966	3	354 (depth of 949)
10000	20	1899	8223	5	No violation found
10000	200	102719	354239	227	82 (depth of 200)
10000	1000	354	966	3	354 (depth of 949)
10000	2000	354	966	2	354 (depth of 949)

TABLE 4 – Tableau avec différents paramètres supertrace (N = 5)

Commentaires :
Lorsqu'on regarde la table 4, on peut voir que si on limite trop la profondeur, on ne sera pas confronté à la

violation trace. Avec une profondeur moyenne, nous obtenons une violation trace et avec une grande profondeur (1000 ou 2000) nous obtenons une plus grande violation trace. Concernant la taille de la hashtable, on remarque qu'elle doit être assez grande pour trouver une violation trace cepedant avec une bonne profondeur la taille de la hashtable peut être négligée. Pour finir, on peut voir que le nombre d'états est maximal pour une valeur moyenne de la hashtable (100 kb) et de la profondeur (200) et avec ces valeurs nous obtenons une plus petite violation trace.

8 Conclusion

Tous les objectifs ayant été remplis à temps, nous estimons être satisfait de cette seconde phase malgré les soucis rencontrés lors de la phase 1. Nous espèrons donc reproduire un travail de qualité tout en remplissant tous les objectifs du dernier projet de ce cours.