LOG8470: Vérification de la fiabilité et sécurité
Travail Pratique 2: Model Checking iSpin

Présenté à:
Oswald Pichot

Par:
Kenny Nguyen (1794914)
Yujia Ding (1801923)
Ka Hin Kevin Chan (1802812)

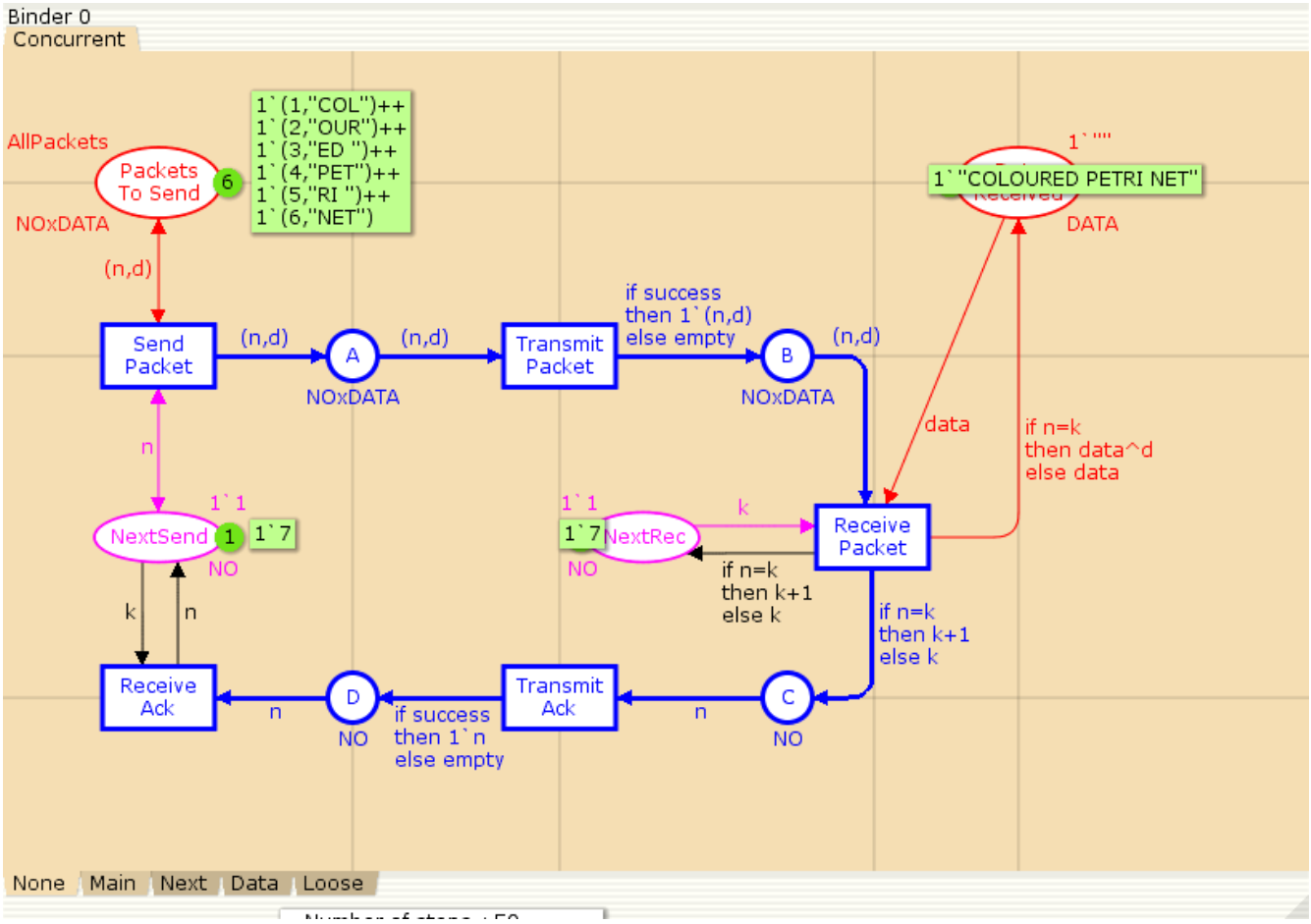
École Polytechnique de Montréal
Département de génie Logiciel
Le 22 novembre 2018

Exercice 1.

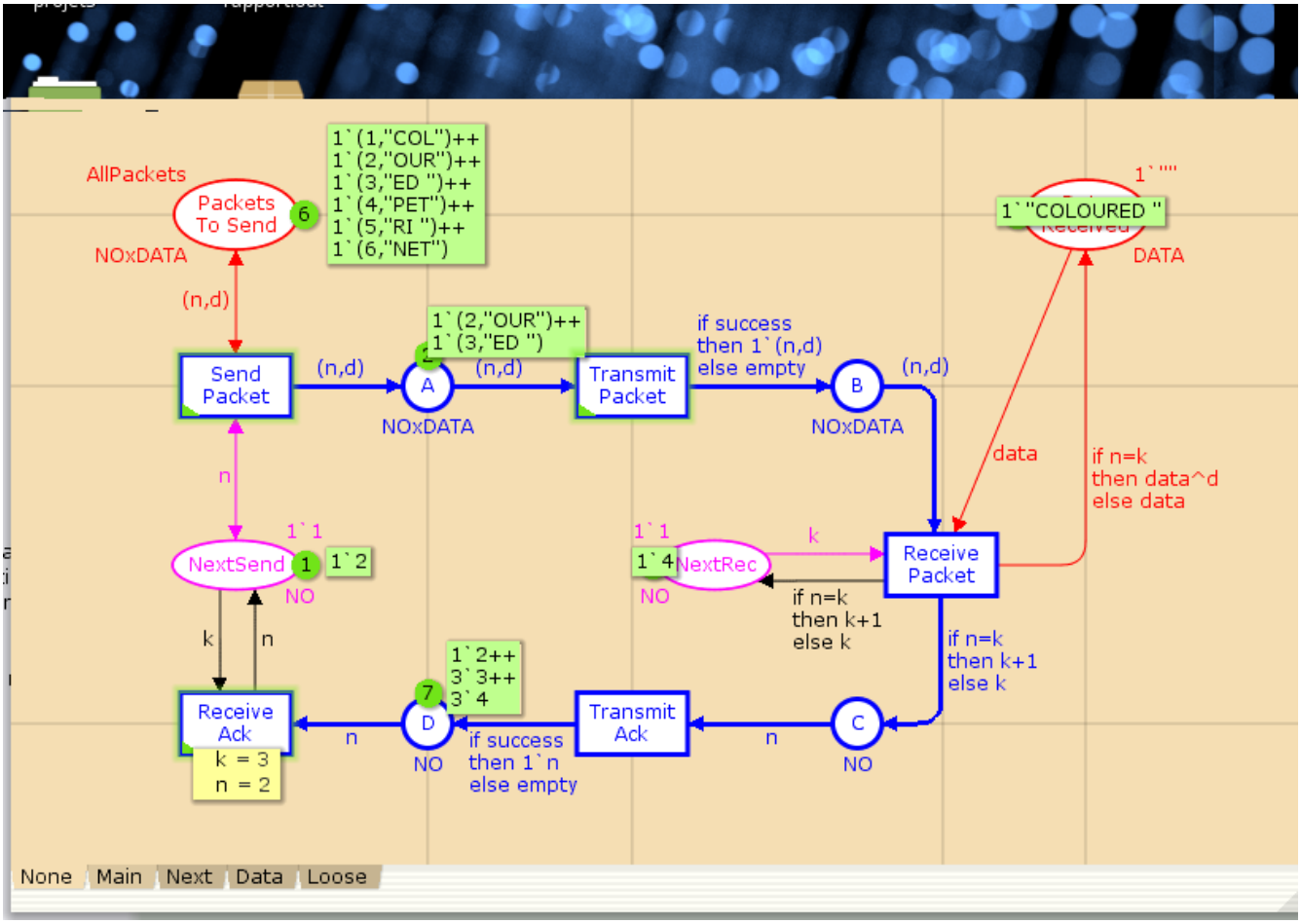
a.

Le protocole se termine avec l'état final « COLOURED PETRI NET ». Le nombre de steps diffère à chaque simulation complète, puisque la simulation est automatique. Dans tous les cas, l'état final est unique, car le k de la place « NextSend » se rend à 7, et il n'y a pas de paquet NO=7 dans la place « Packets To Send », alors le réseau est bloquant.

-Perte : La perte de paquets est simulée dans le circuit avec la variable succès à « false » dans la transition « Transmit Packet ».
-Duplication : La duplication de paquets est gérée par la comparaison n = k à la transition ''Receive Packet''. En effet, si le numéro de la syllabe (n) est égal au compteur de paquet reçus (k de « NextRec »), la syllabe est concaténée à « data » de la place « Data Received », sinon la syllabe n'est pas ajoutée.
-Réarrangement : Le réarrangement est manipulé par le compteur k de la place « NextSend ». Lorsqu'un Ack réussit, le compteur k est incrémenté de 1, ce qui permet de générer la syllabe d'indice n suivant. Autrement, la même syllabe est toujours produite, puisque le k demeure constant. Ainsi, par exemple, il est impossible d'envoyer NET avant que RI ne soit Ack.

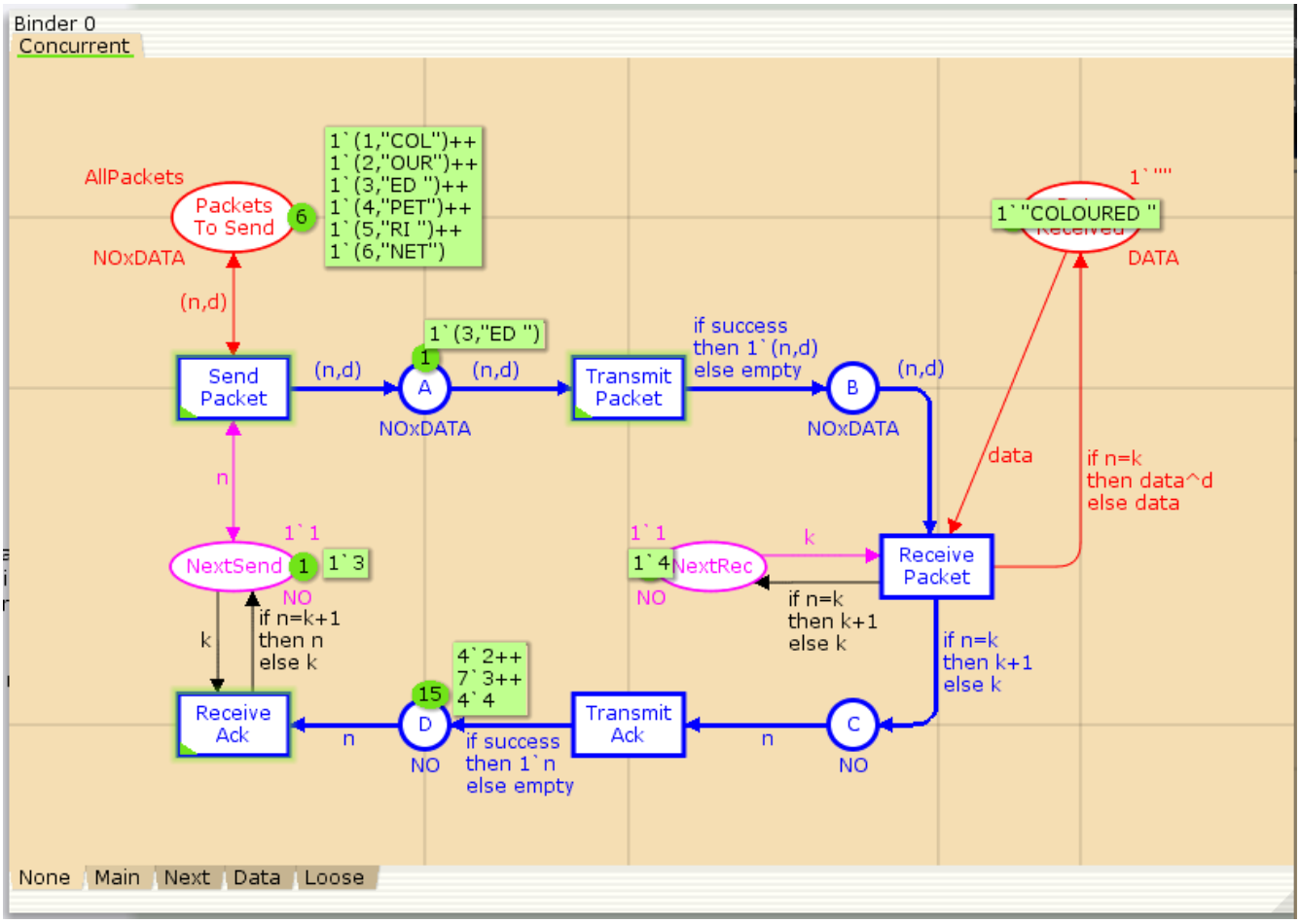


b.



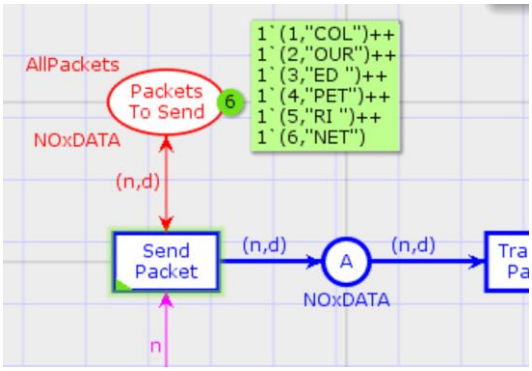
Pour décrémenter le compteur de l'accusé de réception (k de « NextSend »), nous avons utilisé le simulateur interactif pour créer l'état ci-dessus. Nous avons généré plusieurs jetons Ack de « OUR » (de valeur 3 dans la place D) que nous avons accumulés dans la place D. Ensuite, nous avons traversé la place « NextSend » une fois pour incrémenter k de 1 à 2 (SendPacket passe de COL à OUR), ce qui nous permet, de la même façon, de générer des jetons Ack de « ED » que nous accumulons dans la place D. Nous traversons une deuxième fois la place NextSend et k passe de 2 à 3. Nous accumulons des Ack de ED dans la place D (n =4). Nous traversons la place NextSend avec un jeton n = 2 (Ack de COL) et le compteur k passe de 3 à 2.

Le modèle original assigne la valeur de n à k lorsqu'on traverse la place NextSend. De cette façon, il est possible de modifier la valeur de k pour n'importe quelle valeur de n présente sur la place D. C'est l'arc n, allant de Receive Ack à NextSend qui cause ceci. Nous avons modifié cet arc pour que la valeur de k s'incrémente de 1 (then n), uniquement lorsque l'Ack de la syllabe actuelle est passée, c'est-à-dire lorsque n=k+1 (if n=k+1). Dans le cas échéant, nous voulons que k reste le même (else k)



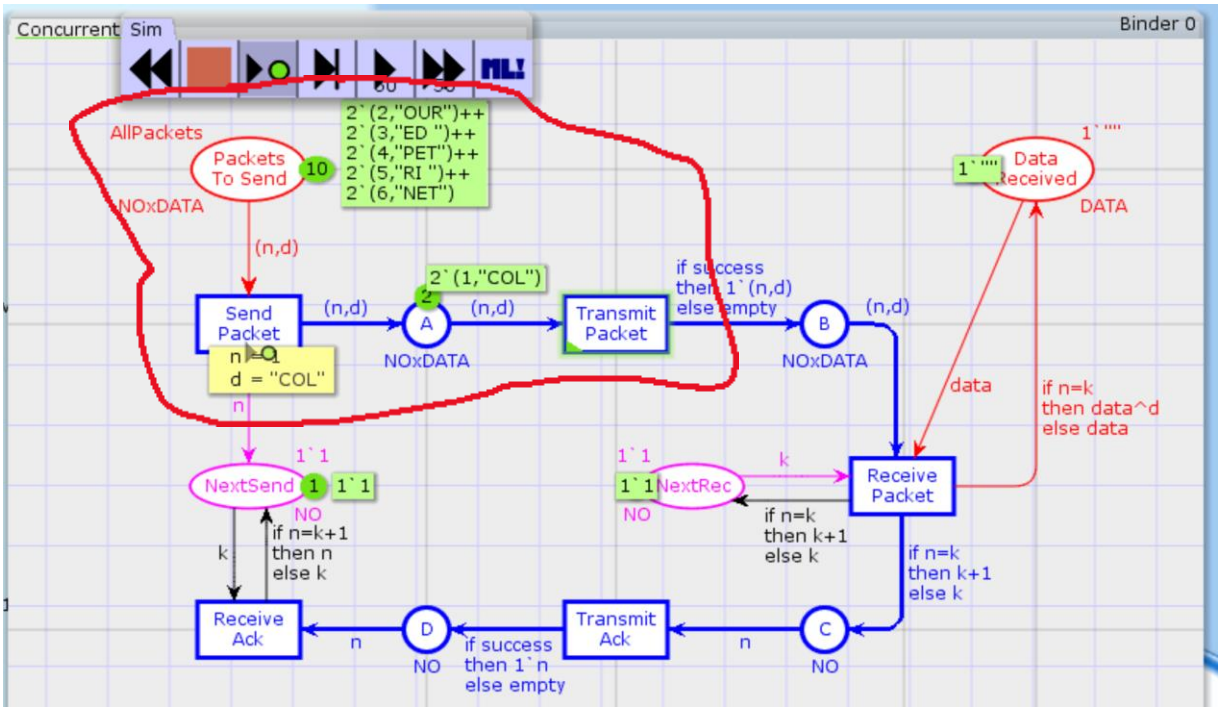
C.

Dans le modèle original, on s'aperçoit que la transition « Send Packet » consomme puis remet un jeton de la place « Packets To Send ». La place contient 1 jeton de chaque type.



Pour borner le nombre de jetons retransmis, il faut limiter le nombre de jetons générés par la source. De plus, l'arc rouge (n,d) ne doit pas remettre de jeton dans « Packets To Send ». Il faut retirer l'arc rouge partant de « Send Packet » vers « Packets To Send ». Ceci étant fait, le nombre de jetons diminue de 1 à tirage de la transition. Lorsque le nombre de jetons atteint 0, on aura borné le nombre de jetons retransmis à k (ici nous bornons le nombre de jetons transmis à 2).

Dans la capture ci-dessous, nous montrons l'effet de cette modification. En utilisant la simulation pas à pas, on clique tire deux fois « Send Packet » pour transmettre deux jetons de COL. Puisque le n de l'arc mauve est resté à 1, on essaye de tirer un autre jeton COL. Cependant, on a retransmis le nombre maximal de COL, tous les jetons sont épuisés. Il ne reste que 10 jetons, deux pour chaque syllabe restante. La transition « Send Packet » n'apparaît pas en vert, ce qui veut dire que la retransmission est impossible.



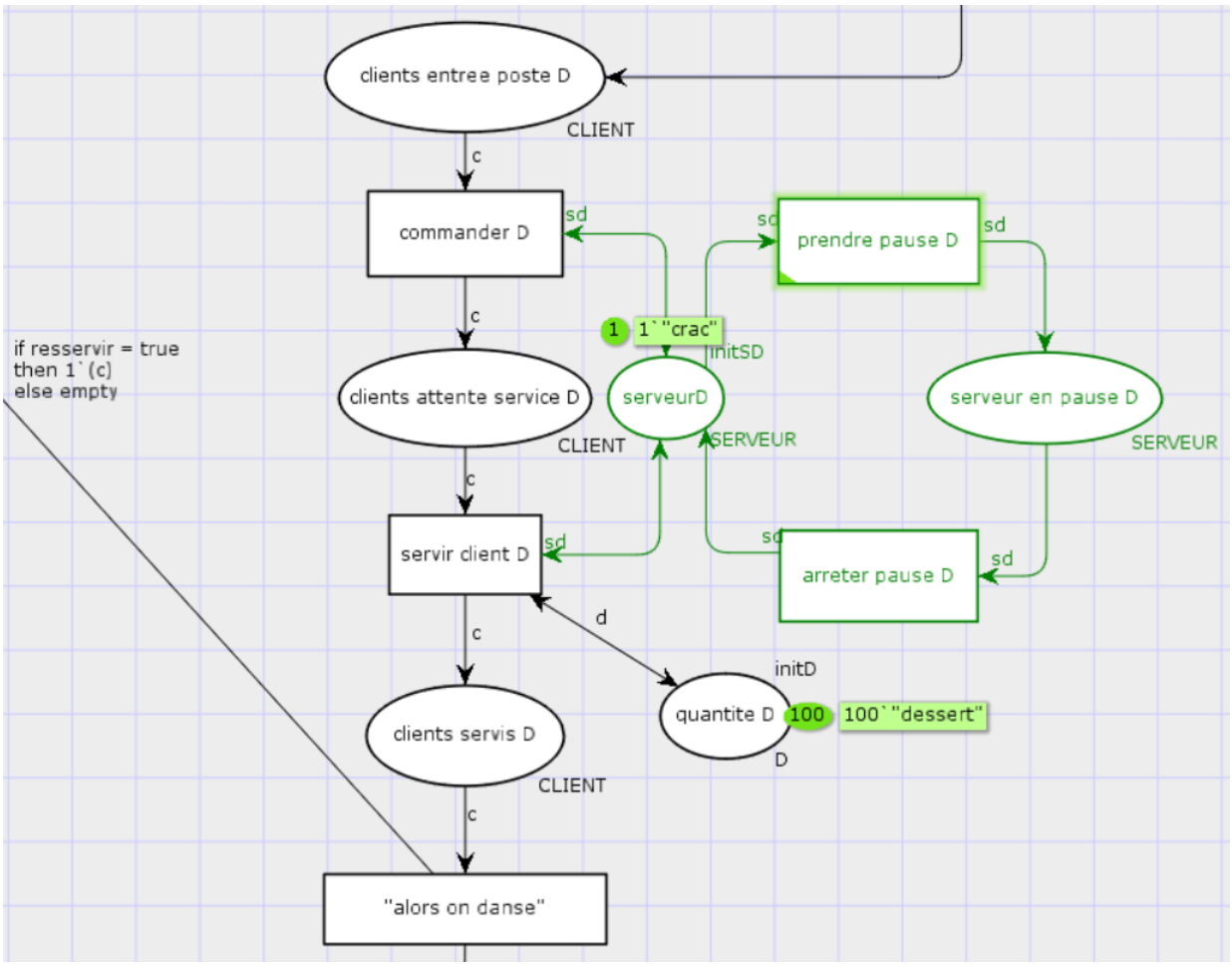
Exercice 2.

Les scénarii 1 et 2 sont des réseaux de Petri à deux couleurs.

Scénario 1.
a. Le réseau est vivace puisque toutes les transitions sont tirables entre deux marquages accessibles. Ceci est rendu possible par la forme du réseau : ses transitions prennent un jeton d'une seule place, puis en mettent un dans une seule place. Cette propriété contribue également à le rendre non-bloquant. Le réseau est quasi-vivace puisqu'il est vivace. Il est n'est pas borné puisque la seule condition pour se resservir est de le vouloir (resservir = true).

b. Du point de vue des clients, le protocole se termine lorsque tous les clients sont dans la place des clients en train de célébrer. En effet, cette place est finale et un client qui s'y rend ne peux plus en sortir. Le booléen « resservir » détermine si le client retourne à l'entrée du poste des plats principaux. Du point de vue des serveurs, le protocole ne finit pas, puisqu'un serveur qui part en pause finit toujours par en revenir, et vice-versa.

c. Non, il n'est pas possible d'arriver à l'état final sans consommer de produit. À chaque poste, il est obligatoire de tirer une place de service. Cela consomme un produit, même si, pour les fins de l'exercice, on remet le produit pour simuler une quantité infinie de produits.



Scénario 2
Remarque : voir c.
a. Le réseau est vivace. Comme le scénario 1, ses transitions prennent un jeton d'une seule place, puis en mettent un dans une seule place. Le réseau est bloquant puisqu'éventuellement, il n'y aura plus assez de plats, et on ne pourra pas plus tirer la transition bloquante. Le réseau est16- borné par le nombre, 16 étant le nombre de jetons de produits, qui ne se renouvellent pas.

b. Comme le premier scénario, le protocole se termine si tous les clients célèbrent. Il ne se termine pas pour les serveurs, car ils entrent et sortent de pause indéfiniment.

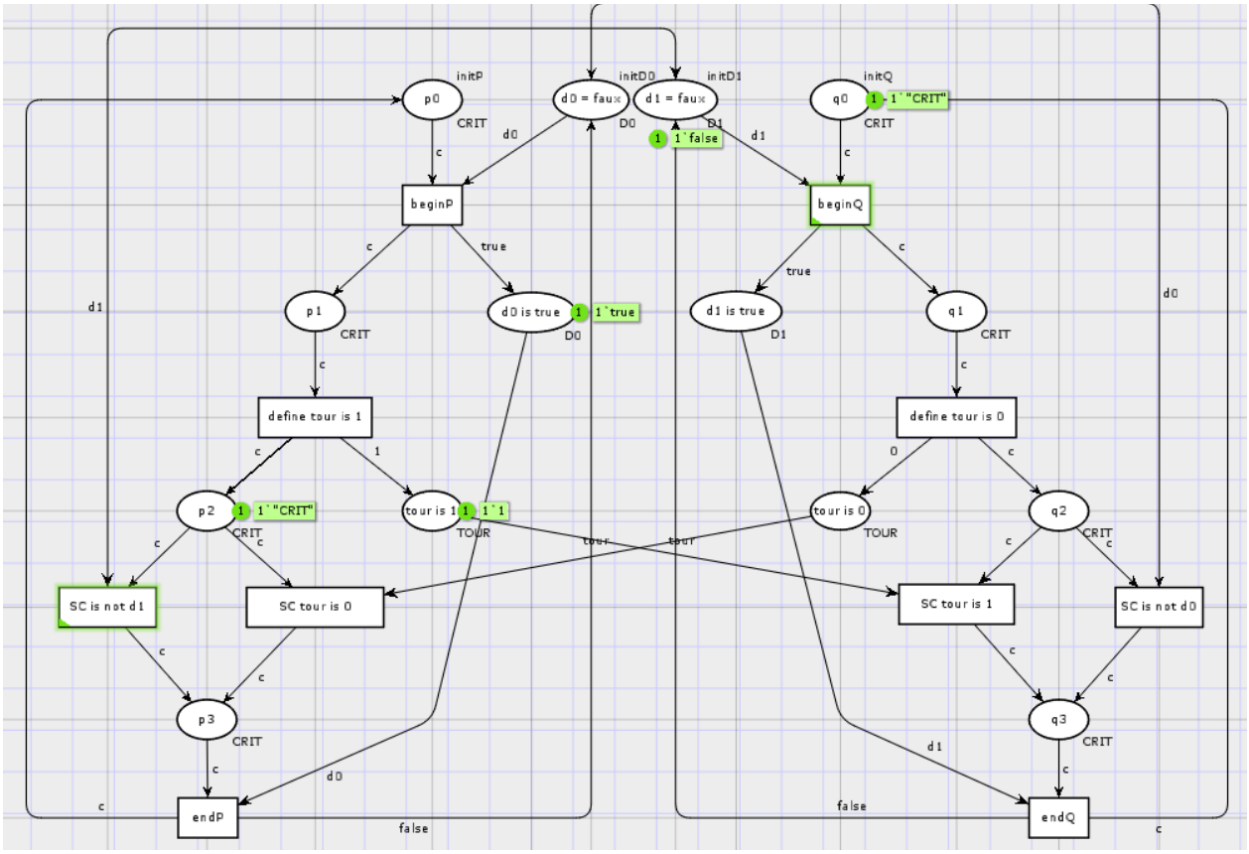
c. Il serait possible d'arriver dans un tel état. Or, nous n'avons pas eu le temps de récréer cet état par manque de temps. Il aurait fallu que les arcs transportent le menu. Il faudrait que les transitions servir offrent la possibilité de ne pas prendre de plat (avec un if content true, then menu, else empty). Ensuite, on peut créer l'état en choisissant rien à chaque transition.

Exercice 3.

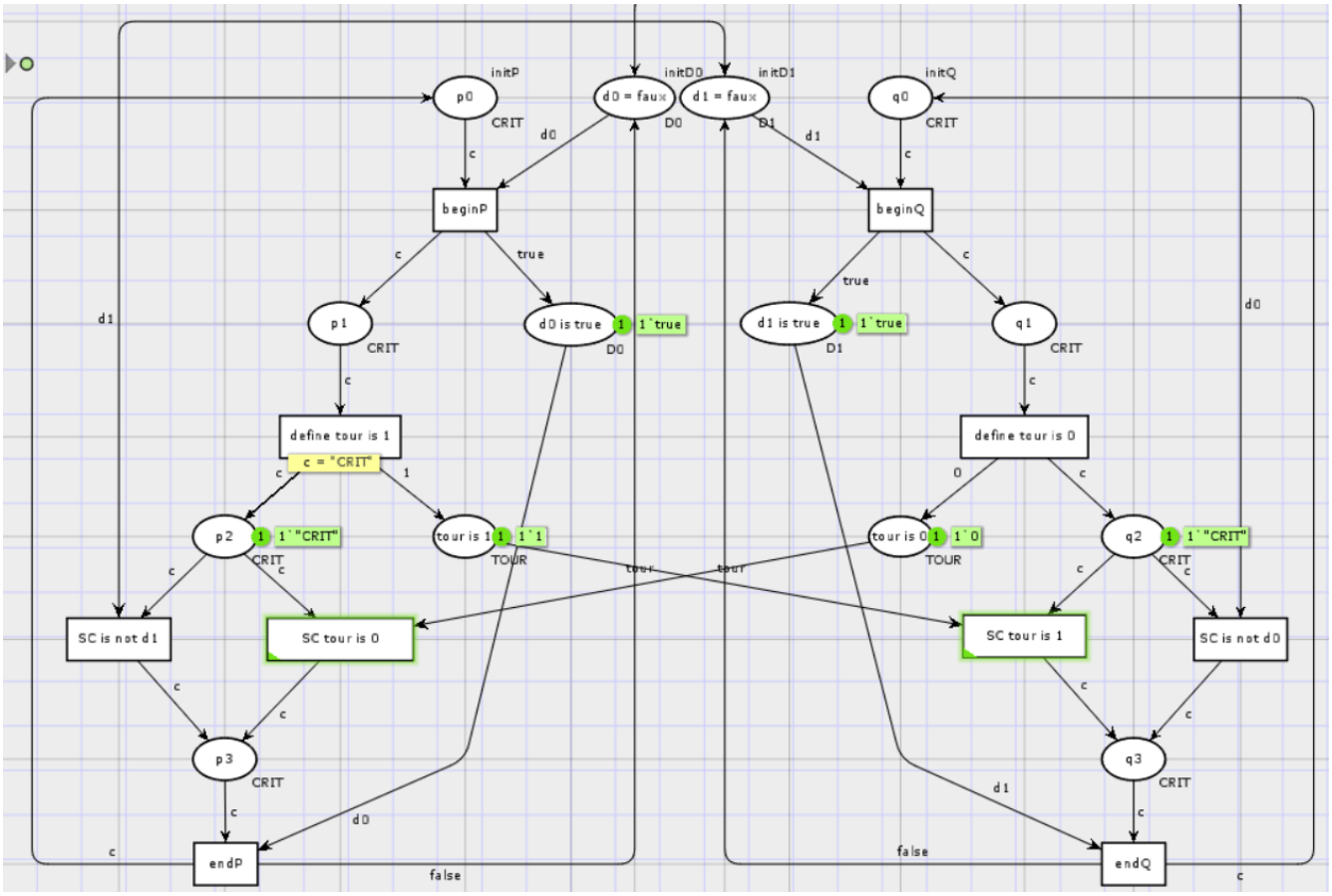
Remarques :
-La variable c est employée afin de tirer des transitions. Sa valeur n'a pas d'importance.
-Dans notre modélisation de l'algorithme, tirer une transition d'une garde de wait until équivaut à traverser la section critique.

Pour respecter l'exclusion mutuelle, il faut qu'un seul des processus soit en mesure de traverser la section critique. Chacun des processus peut accéder à la section critique de deux manières.

Pour le processus P, la première façon est de tirer beginP, define tour is 1. Cela fait en sorte qu'on atteint la garde sans que d1 ne passe à faux. Ceci nous permet de tirer SC is not d1, sans que Q n'ait accès à la section critique. On procède de la même façon avec le processus Q, en tirant les transitions équivalentes.



La deuxième façon est de tirer beginP, beginQ, define tour is 1 et define tour is 0. Théoriquement, on arrive à la section critique de P sans que Q n'ait accès à la section critique. Cependant, nous constatons que P et Q ont accès à la section critique, puisque les deux processus peuvent tirer la transition. Nous avons choisi de rester fidèles au réseau présenté en classe.



Nous aurions pu résoudre ce problème en utilisant une place unique pour tour en appliquant les changements de la capture d'écran ci-dessous, mais cela aurait nécessité de remettre un jeton dans les places p0 et d1 = faux, ce qui n'est pas cohérent avec l'algorithme de Peterson.