Correction TD 1

Exo 1 :

Exo 1.1 :

Les éléments identifiés sont : arc (flèche sur le dessin), place (cercle), jeton (points noirs), transition (rectangle), nombre de jetons (par exemple 3 dans la place P3), Voir Annexe pour les définitions de chaque élément.

Exo 1.2 :

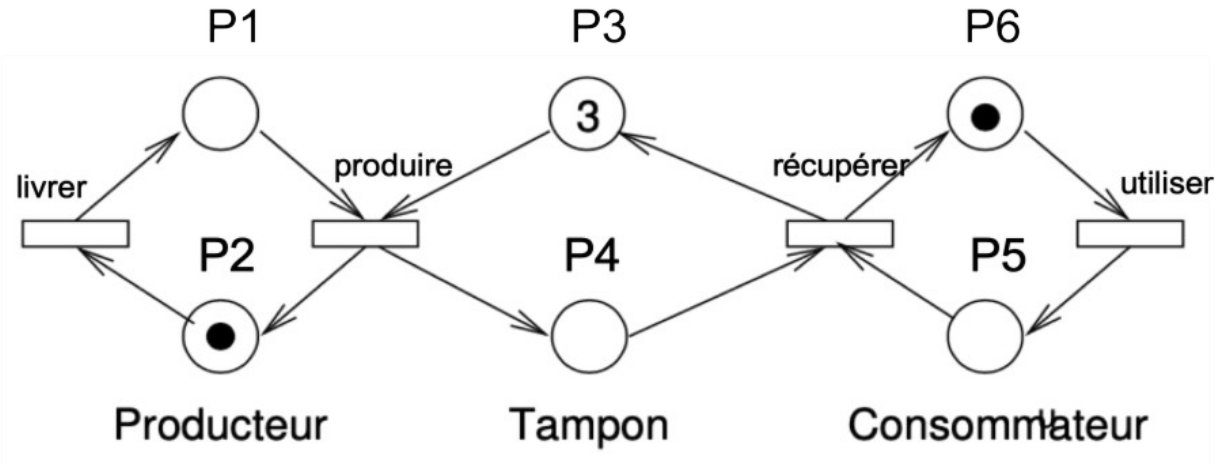
Ici, pour évoluer le réseau de pétri (animation en utilisant l'outil tina qui sera vu en TP ou si non l'outil en ligne APO[1]), on exécute à la main.

Une transition est exécutable si et seulement si il existe un nombre suffisant des jetons disponibles tel que requis par les arcs entrants de la transition.

L'exécution d'une transition implique la consommation (sauf pour l'arc de lecture) des jetons requis et en résulte une production d'un nombre de jetons dans les places cibles tel que indiqué sur les arcs de sortie.

Remarque : cette évolution a été déroulée pendant le TD avec l'outil APO.

Exo 1.3 :
Tampon (la place P4) est une place non bornée, demo faite en TD. Voici la modification de nitre exemple pour borner P4 à 3 jetons.



Cette modification va obligatoirement ralentir le producteur (la transition produire est maintenant conditionnée par les jetons venons de P1 mais aussi de P3) et donc P4 est devenue borné.

Exo 1.4 :

1 https://apo.adrian-jagusch.de

Un graphe de marquage est un graphe où :

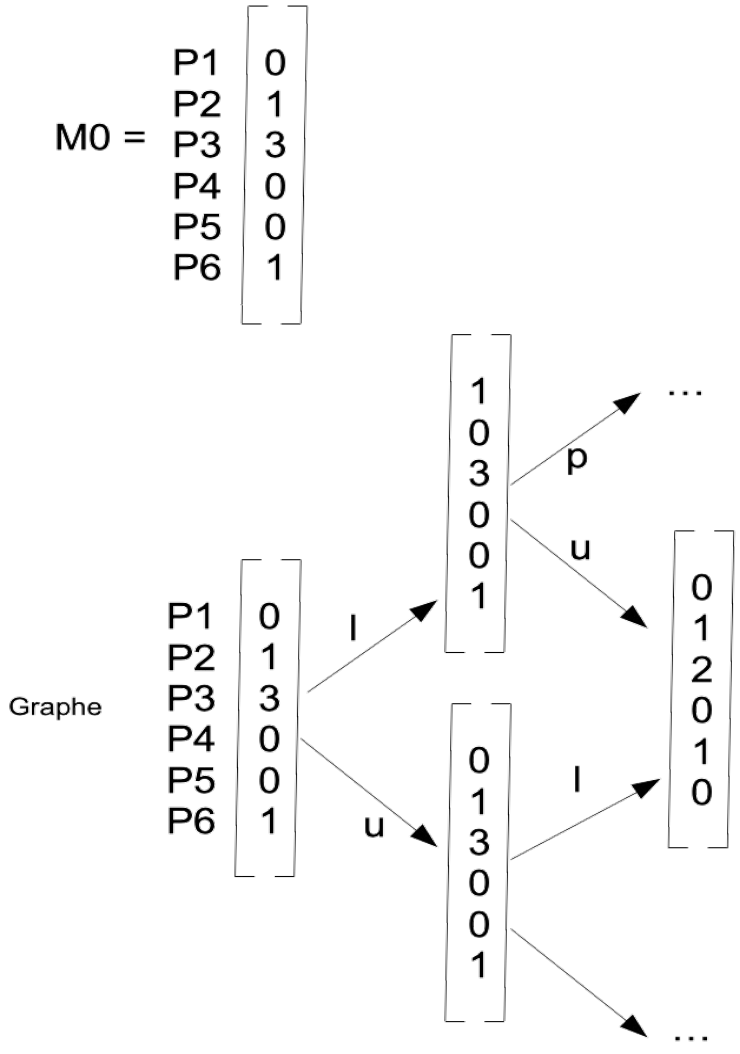
- les sommets (états) sont les marquages atteints[2] et les arcs,
- les transitions franchissables permettent de passer d'un marquage à un autre.
- les marquages identiques sont « fusionnés ». C'est à dire on forme un cycle (ou une boucle) à chaque fois on retrouve un marquage déjà identifié.

Voici le début de ce graphe avec M0 comme marquage initial. Un graphe complet pourrait être construit avec l'outil APO. Dans ce graphe, j'ai simplifié la notation des transitions :

- p pour produire
- l pour livrer
- u pour utiliser

L'algorithme suivi pour obtenir ce graphe est le suivant :

- M0 est un vecteur où Pi représente une place et le vecteur mentionne le nombre de jetons existant dans cette place.
- À partir de chaque marquage on dessine toutes les transitions ``franchissables". En cas de plusieurs transitions possibles, on alterne toutes les transitions. On parle ici d'un entrelacement.
- À chaque fois qu'on retrouve un marquage déjà visité on boucle vers celui existant plutôt qu'en rajouter un sur le graphe.



2 Il faut revoir l'appendice de votre sujet pour la définition d'un marquage.

Exo 2 :

2.1.1

Voici les scénarios :

1) A1 --f2s--> A2 la contrainte f2s signifie que A2 ne peut commencer que si A1 a terminé. La solution sera donc :

   Commencer A1
   Terminer A1
   Commencer A2
   Terminer A2

2) A1 --s2f--> A2

Commencer A1 Commencer A2 Terminer A1 Terminer A2	Commencer A2 Commencer A1 Terminer A2 Terminer A1	Commencer A2 Commencer A1 Terminer A1 Terminer A2	Commencer A1 Commencer A2 Terminer A2 Terminer A1	Commencer A1 Terminer A1 Commence A2 Terminer A2

3) A1 --s2f--> A2 et A0 --f2f--> A2 ci-dessous quelques scénarios possibles :

Commencer A0 Terminer A0 Commencer A1 Commencer A2 Terminer A1 Terminer A2	Commencer A0 Terminer A0 Commencer A1 Terminer A1 Commencer A2 Terminer A2	Commencer A1 Terminer A1 Commencer A0 Terminer A0 Commencer A2 Terminer A2	Commencer A1 Commencer A0 Terminer A0 Commencer A2 Terminer A2 Terminer A1

Vous pouvez continuer à proposer les autres scénarios valides c'est à dire en respectant les contraintes (s2f c'est l'étiquette start to finish, f2f est l'étiquette de finish to finish).

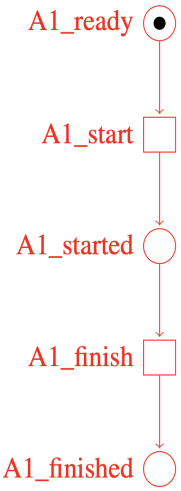
2.1.2 Pour donner un scénario non valide il suffit d'en donner un qui ne respecte pas la contrainte qui relie les activités. Par exemple pour le modèle A1 --f2s--> A2, un scénario non valide (erreur) peut être le suivant :

commencer A1
commencer A2 KO car on doit respecter f2s !!!!
terminer A1
terminer A2

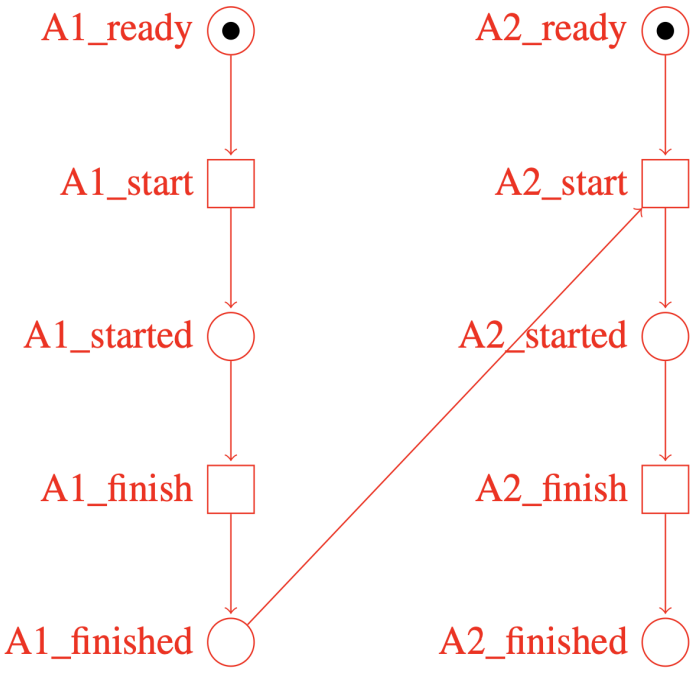
2.2 Cet exercice la transformation d'un modèle vers un autre modèle. Pour transformer le modèle SimplePDL vers un réseau de Petri, on applique les règles suivantes. Chaque activité est représentée en réseau de pétri en terme des :

- places : pour décrire les états et on aura alors :
  - place initiale Ai_ready pour exprimer que l'activité est prête à commencer
  - place intermédiaire Ai_started pour dire que l'activité a commencer
  - place final Ai_finished pour dire que l'activité a terminé
- transitions : on en a Ai_start et Ai_finish

Voilà donc la traduction pour A1 :

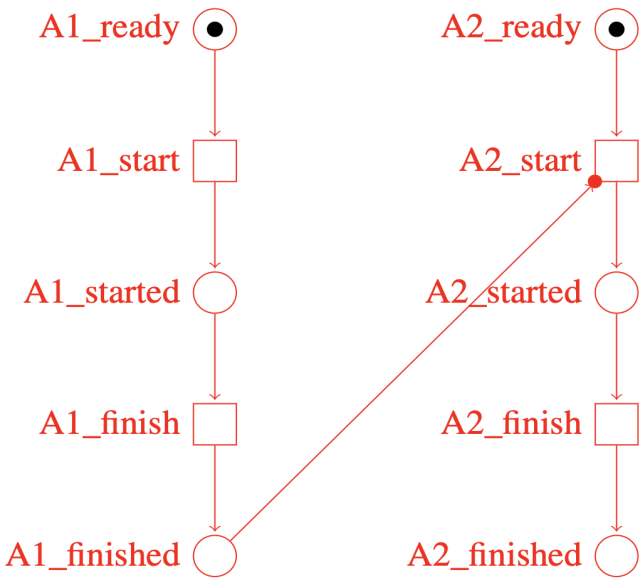


Maintenant pour donner la traduction de l'exemple 1 : « L'activité A2 ne peut commencer que quand l'activité A1 est terminée. » On rajoute un arc entre la place finished de A1 et la transition start de A2 afin de garantir la contrainte A1 – finishToStart → A2



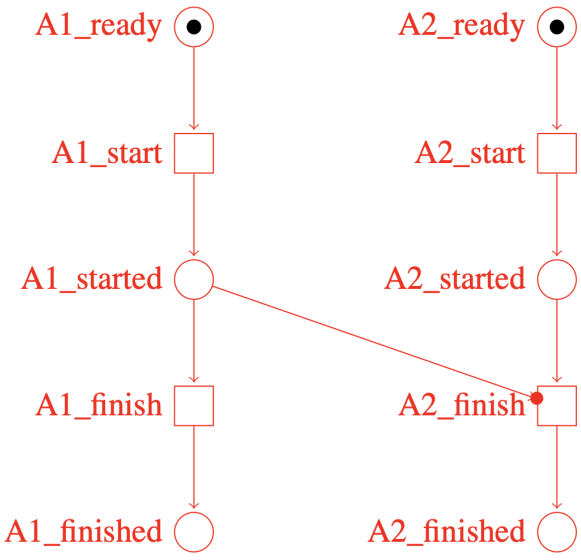
Question : Est ce que la terminaison peut être vérifiée avec la solution qu'on vient de donner ???
Réponse : Non car A1 consommé sont jeton dans sa place finale :(

Solution : l'arc entre A1_finished et A2_star doit plutôt être un arc de lecture pour garder en mémoire que A1 a terminé. Voici donc la correction :



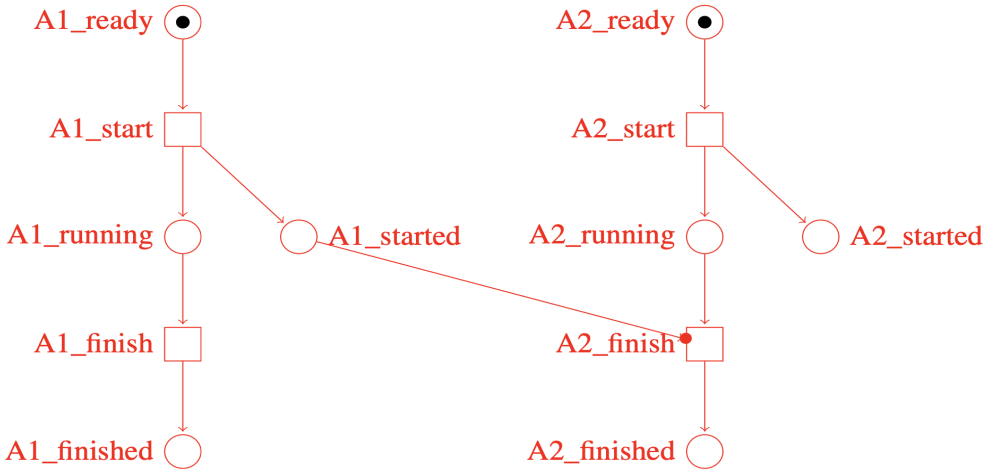
Vérifiez que les scénarios valides et pas valides déterminés dans les exercices précédents peuvent être modélisé également par le réseau de pétri.

A1 – start2finish → A2



La solution présente un problème : imaginons que A1 est dans A1_started et transite à finish, alors le jeton se déplace (se consomme de A1_started) donc dans A1_finished. En résultat, ici malheureusement A2 ne pourra jamais terminer car A_2 finish ne peut pas s'exécuter (il va manquer un jeton de A1_started) !

Astuce : pour résoudre ce problème : on introduit un état Ai_running comme suit :



Cette solution permet donc de mémoriser que chaque activité a commencé, ensuite l'autre activité peut se terminer ou pas indépendamment du reste.

Dérouler ce réseau et vérifier que les scénarios sont bien décrits par ce modèle !

3) suivez le même raisonnement pour le dernier scénario …

Exo 2.3 :

1. Le processus peut terminer. Cela revient à vérifier que les jetons arrivent tous en états « finished » !
   La logique linéaire temporelle (LTL) permet de décrire cette propriété formellement. Nous verrons en TP que l'outil tina permet de vérifier ces propriétés sur des réseaux de pétri.
   Voici comment traduire « Le processus peut terminer »en LTL :

   ```
   op finished = A1_finshed /\ A2_finished
   <> finished
   ```

2. Le processus ne peut pas terminer.
   C'est l'iverse de la propriété précédente :

   ```
   op finished = A1_finshed /\ A2_finished

   - <> finished
   ```