

# LRC TME7

Yuxiang ZHANG  
Kenan ALSAFADI

Novembre 2025

## 1 Prise en main

### Exercice 1

Nous avons ouvert le réseau `16000.ndr` à l'aide de la commande :

```
bin/nd nets/16000.ndr
```

1. Après observation du réseau et de son marquage initial, on constate que les places **p1** et **p3** portent respectivement les marquages 6K et 2K. Les poids des arcs indiquent comment les jetons circulent entre les trois places : **p1**, **p2** et **p3**. L'évolution semble former un cycle : **t1** amène les jetons vers **p2**, **t2** vers **p3** et **t3** de nouveau vers **p1**. Le réseau paraît donc susceptible de fonctionner de manière cyclique.
2. Avec le simulateur pas-à-pas (*Tools* → *Stepper simulator*), nous avons simulé le comportement du réseau. Les transitions franchissables apparaissent en rouge et peuvent être tirées en cliquant à côté de leur label. Le bouton *Rand* permet une simulation aléatoire continue, et la vitesse peut être réglée dans le menu *Mode* → *Firing duration*. La simulation confirme le comportement cyclique anticipé : **t1**, **t2** puis **t3** se tirent successivement.
3. Pour faciliter la simulation, nous avons réduit les marquages 6K et 2K à 6 et 2, et remplacé les poids d'arc 6K, 3K et 2K par 6, 3 et 2. L'édition d'une place ou d'un arc s'effectue par un clic droit sur l'élément concerné. Cette simplification ne change pas la dynamique du réseau mais permet une simulation plus lisible.

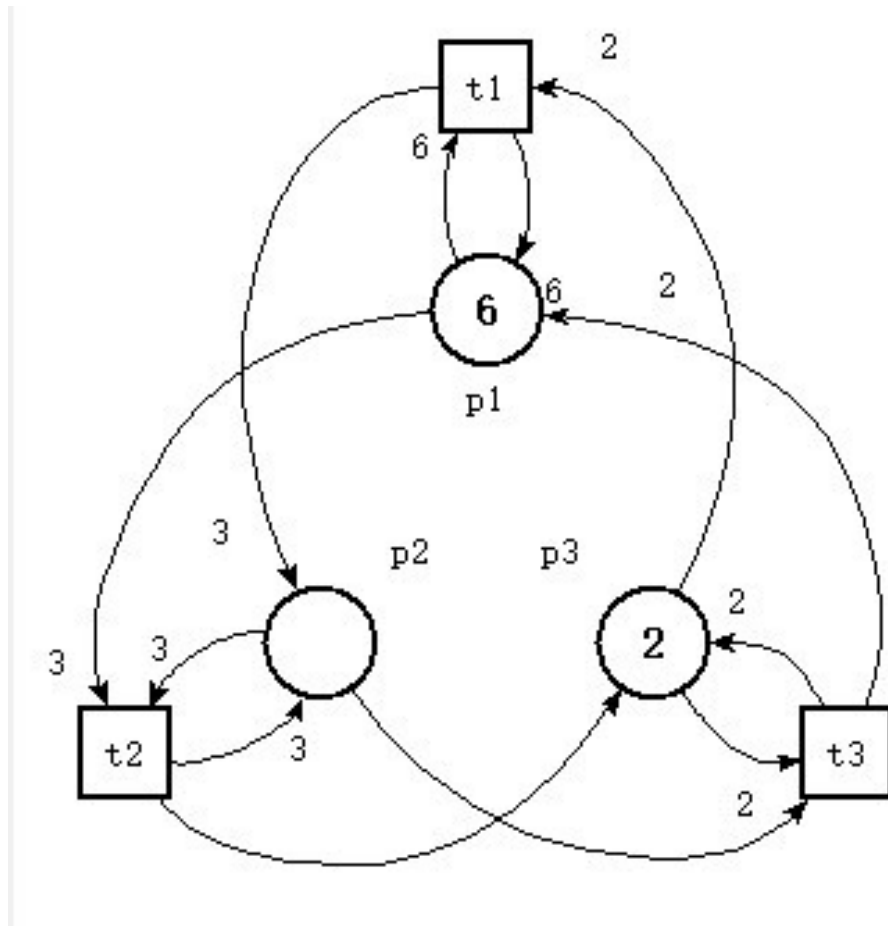


FIGURE 1 – Réseau de Petri simplifié pour faciliter la simulation.

4. À l'observation, le réseau semble :

- **borné** : les marquages restent limités et les jetons circulent en cycle,
- **vivant** : toutes les transitions semblent franchissables à répétition,
- **réversible** : on semble pouvoir toujours revenir au marquage initial.

Ces intuitions seront vérifiées dans la question suivante.

5. À l'aide de l'analyse d'accessibilité (*Tools* → *State space analysis*, sortie *verbose* avec analyse de vivacité), nous obtenons les résultats suivants :

```

bounded
6 marking(s), 6 transition(s)
live
reversible
0 dead marking(s), 6 live marking(s)
0 dead transition(s), 3 live transition(s)

```

L'analyse confirme donc que le réseau est :

- **borné**,

- **vivant**,
- **réversible**.

Le graphe possède une seule composante fortement connexe contenant tous les marquages :

**STRONG CONNECTED COMPONENTS:**

0 : 5 4 3 2 1 0

Cette unique SCC inclut le marquage initial et toutes les transitions, ce qui explique la vivacité et la réversibilité du réseau.

- Enfin, pour générer et afficher le graphe des marquages accessibles, nous avons choisi la sortie **kts (.aut)** dans l'analyse d'accessibilité, puis fait un clic droit pour *Open in nd*. Le menu *Edit* → *Draw* permet de produire une version graphique du graphe, en particulier en choisissant l'outil de visualisation *dot*. Le graphe obtenu confirme que les six marquages accessibles forment un cycle unique.

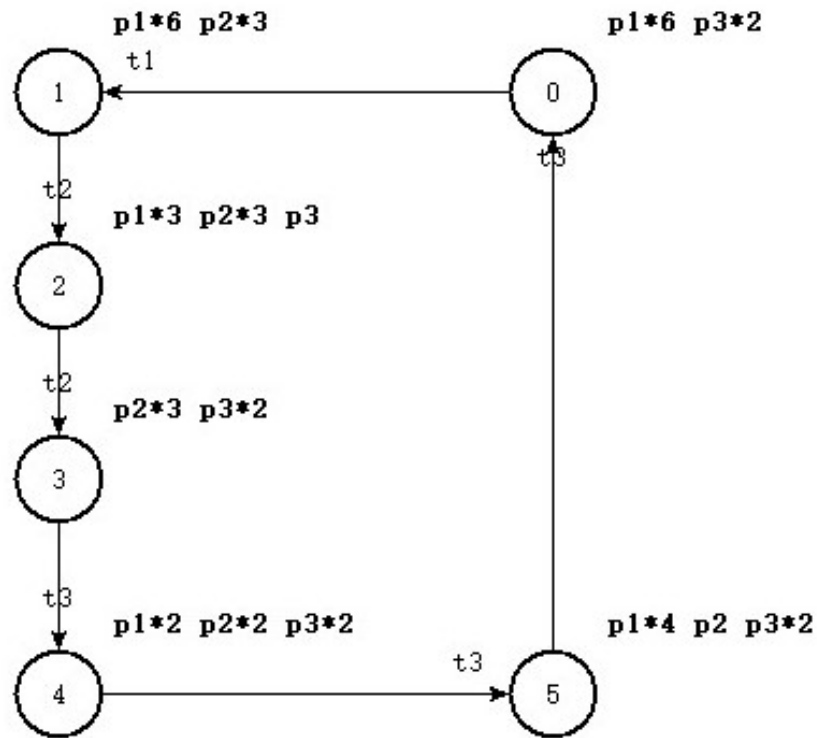


FIGURE 2 – Le graphe des marquages accessibles.

## 2 Syntaxe

### Exercice 2 — Graphe des marquages accessibles et vivacité

Cet exercice utilise Tina pour analyser le réseau de Petri de l'exercice 2 du TD précédent. Le réseau considéré est rappelé ci-dessous.

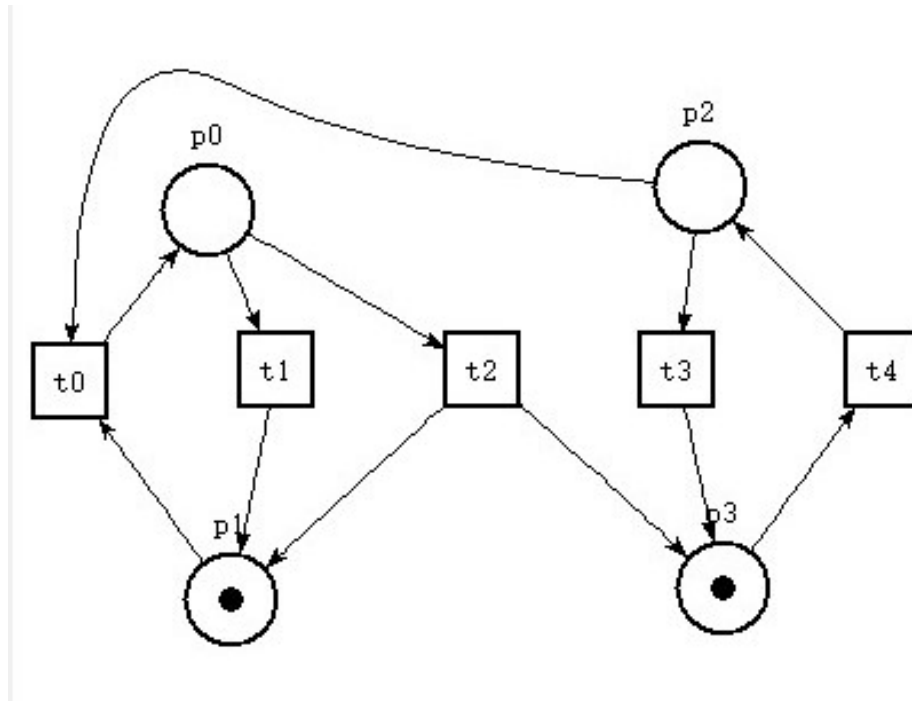


FIGURE 3 – Réseau de Petri de l'exercice 2 avec son marquage initial.

1. Tracer le réseau et indiquer le marquage initial. Le fichier `.ndr` correspondant a été sauvegardé.
2. À l'aide du simulateur pas-à-pas, nous avons tiré la séquence :

$$(t_4 \ t_3 \ t_4 \ t_3 \ t_4 \ t_0 \ t_2 \ t_4 \ t_0 \ t_2 \ t_4 \ t_3 \ t_4 \ t_0 \ t_1)$$

L'historique a été sauvegardé dans un fichier `.scn`.

3. À partir de l'analyseur d'accessibilité de Tina, nous avons généré le graphe des marquages accessibles en utilisant l'option *Sortie automate*. Pour cela, nous avons ouvert l'outil **State space analyser**, puis choisi le format d'export `kts` (`.aut`) dans le menu *Output*. Le fichier `.aut` obtenu a ensuite été ouvert dans l'éditeur `nd` grâce à l'option *Open file in nd*. Dans l'éditeur, nous avons utilisé la commande *Edit* → *Draw*, puis sélectionné le moteur de rendu *dot* afin de produire une représentation graphique du graphe des marquages accessibles. Le diagramme a ensuite été réarrangé manuellement afin d'obtenir une présentation lisible du graphe.

Le fichier `.aut` généré par Tina contient la description du graphe sous la forme d'un automate (DES) :

```

des(0,9,4)
(0,"S. 'p1' S. 'p3'",0)
(0,"E. 't4'",1)
(1,"S. 'p1' S. 'p2'",1)
(1,"E. 't0'",2)
(1,"E. 't3'",0)
(2,"S. 'p0'",2)
(2,"E. 't1'",3)
(2,"E. 't2'",0)
(3,"S. 'p1'",3)

```

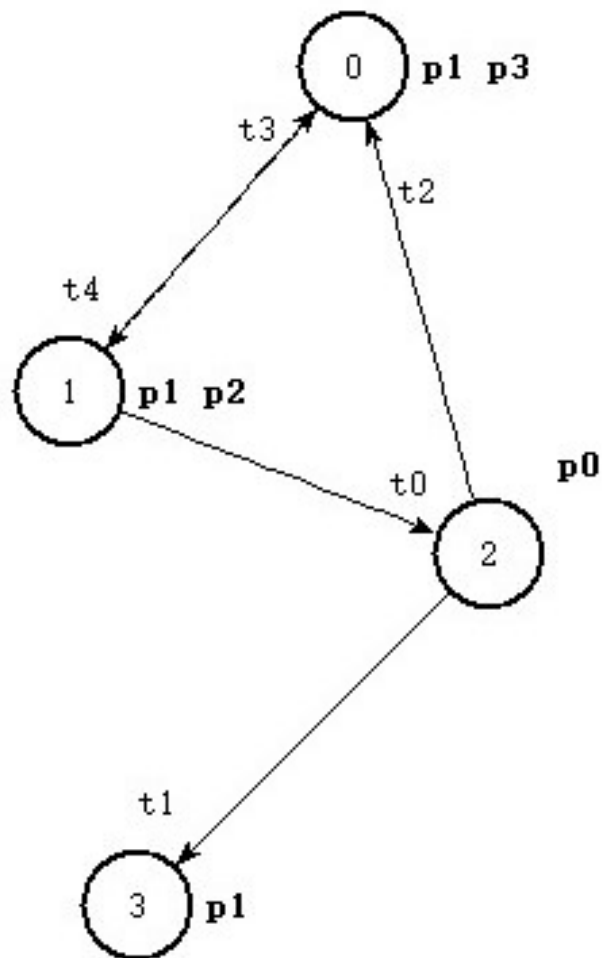


FIGURE 4 – Graphe des marquages accessibles généré par Tina.

4. L'analyse d'accessibilité textuelle donne :

- le réseau est **borné** (4 marquages),
- il n'est **pas vivant**,
- il n'est **pas réversible**,

— il possède un marquage puits : 3 : p1.

5. Pour déterminer si le graphe est **quasi-vivant** dans Tina, il suffit d'examiner l'analyse des composantes fortement connexes (SCC). Un réseau est quasi-vivant s'il existe une composante fortement connexe *terminale* dans laquelle toutes les transitions restent vivantes, c'est-à-dire qu'elles peuvent encore être tirées dans cet ensemble d'états. D'après les résultats fournis par Tina, la seule SCC terminale est la composante 0, qui ne contient que le marquage 3. Ce marquage est un puits et aucune transition n'y est encore possible. Par conséquent, aucune transition n'est vivante dans la composante terminale, et le réseau n'est donc **pas quasi-vivant**.

### Exercice 3 — Propriétés des transitions, marquages et réseaux

On considère le réseau de Petri suivant comportant 4 places ( $p_0, p_1, p_2, p_3$ ) et 5 transitions ( $t_0, t_1, t_2, t_3, t_4$ ).

#### 1. Construction du réseau de base r\_base.ndr

- Réseau tracé dans Tina et sauvegardé sous r\_base.ndr
- Marquage initial :  $M_0 : p_0 = 0, p_1 = 0, p_2 = 0, p_3 = 0$

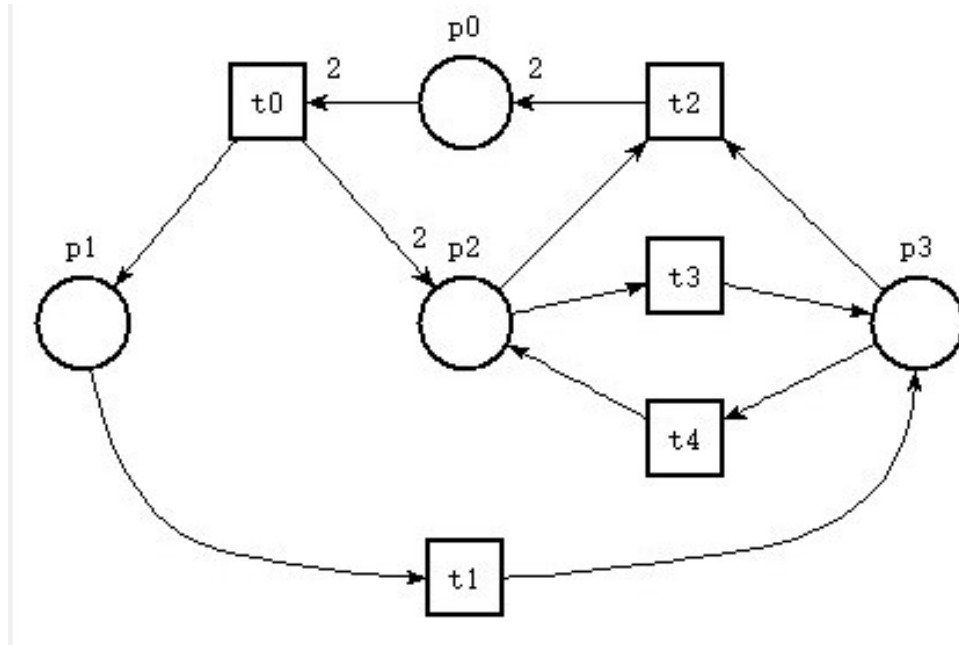


FIGURE 5 – Réseau de Petri initial

#### 2\*. Modifications par changement du marquage initial

##### 2.1 Réseau non borné r\_b0.ndr

- Marquage initial :  $M_0 : p_0 = 2, p_1 = 0, p_2 = 0, p_3 = 0$
- Analyse Tina :

```

REACHABILITY ANALYSIS -----
net unbounded
unbounded places include: p2
MARKINGS:
0 : p0*2
1 : p1 p2*2
2 : p2*2 p3
3 : p1 p2 p3
4 : p0*2 p2

```

- Conclusion : Réseau non borné, la place  $p_2$  peut croître indéfiniment

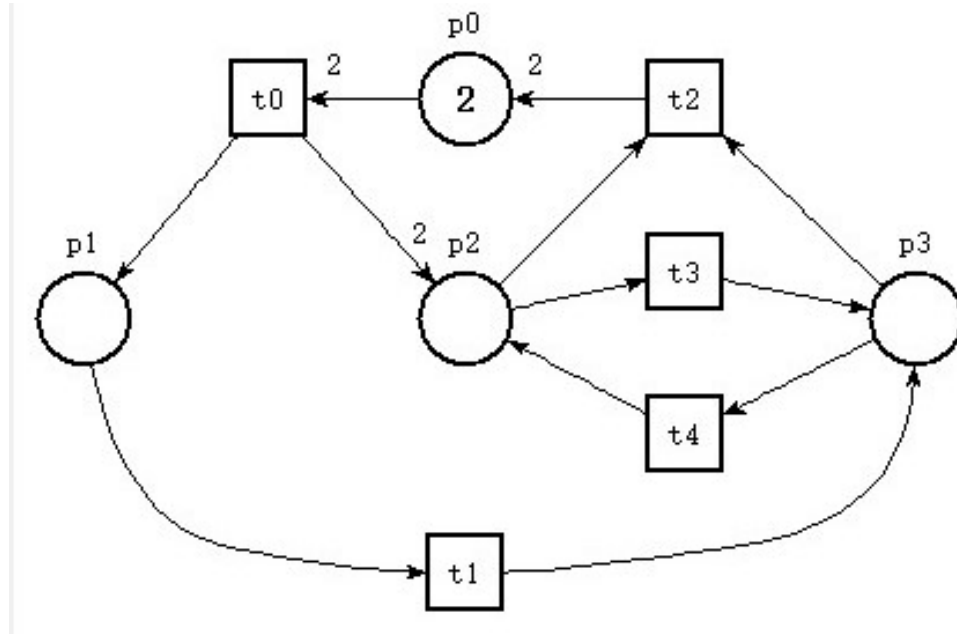


FIGURE 6 – Réseau non borné

## 2.2 Réseau borné avec blocage r\_b1s0.ndr

- Marquage initial :  $M_0 : p_0 = 1, p_1 = 0, p_2 = 0, p_3 = 0$
- Analyse Tina :

```

REACHABILITY ANALYSIS -----
bounded
MARKINGS:
0 : p0
REACHABILITY GRAPH:
0 ->
LIVENESS ANALYSIS -----

```

```

not live
reversible
1 dead marking(s), 1 live marking(s)
5 dead transition(s), 0 live transition(s)
dead marking(s): 0
dead transition(s): t4 t3 t2 t1 t0

```

— Conclusion : Réseau borné mais toutes les transitions sont bloquées

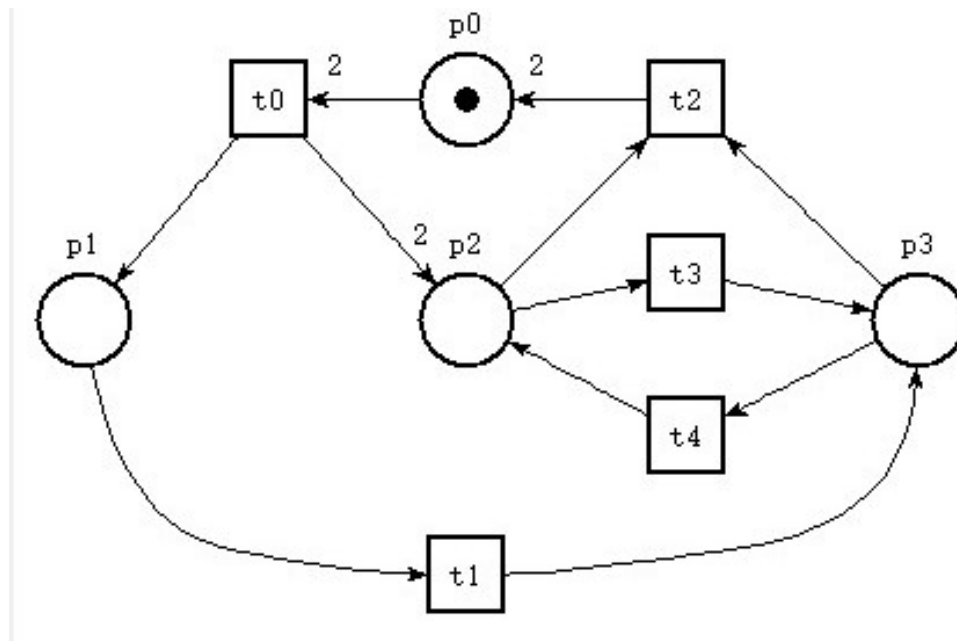


FIGURE 7 – Réseau borné avec blocage

### 2.3 Réseau borné, sans blocage, réversible, ni vivant ni quasi-vivant r\_b1s1r1.ndr

— Marquage initial :  $M_0 : p_0 = 0, p_1 = 0, p_2 = 1, p_3 = 0$   
— Analyse Tina :

```

REACHABILITY ANALYSIS -----
bounded
MARKINGS:
0 : p2
1 : p3
REACHABILITY GRAPH:
0 -> t3/1
1 -> t4/0
LIVENESS ANALYSIS -----
not live
reversible
0 dead marking(s), 2 live marking(s)
3 dead transition(s), 2 live transition(s)

```

dead transition(s): t2 t1 t0

— Propriétés vérifiées :

- **Borné** (b1) : Tina indique **bounded**
- **Sans blocage** (s1) : Au moins deux transitions franchissables
- **Réversible** (r1) : Retour possible au marquage initial
- **Ni vivant ni quasi-vivant** : Transitions t0, t1, t2 jamais franchissables

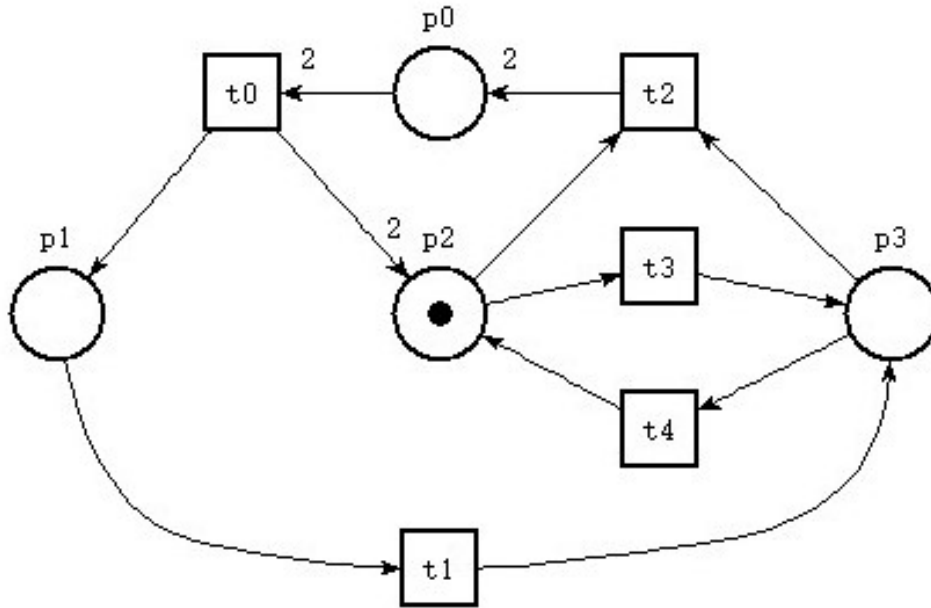


FIGURE 8 – Réseau borné, sans blocage, réversible, ni vivant ni quasi-vivant

## 2.4 Réseau borné, sans blocage, non réversible, ni vivant ni quasi-vivant r\_b1s1r0.ndr

- Marquage initial :  $M_0 : p_0 = 0, p_1 = 1, p_2 = 0, p_3 = 0$
- Analyse Tina :

```

REACHABILITY ANALYSIS -----
bounded
MARKINGS:
0 : p1
1 : p3
2 : p2
REACHABILITY GRAPH:
0 -> t1/1
1 -> t4/2
2 -> t3/1
LIVENESS ANALYSIS -----
not live
not reversible

```

0 dead marking(s), 2 live marking(s)  
 2 dead transition(s), 2 live transition(s)  
 dead transition(s): t2 t0

— Propriétés vérifiées :

- **Borné** (b1) : Tina indique bounded
- **Sans blocage** (s1) : Transitions t1, t3, t4 franchissables
- **Non réversible** (r0) : Impossible de retourner au marquage initial
- **Ni vivant ni quasi-vivant** : Transitions t0, t2 jamais franchissables

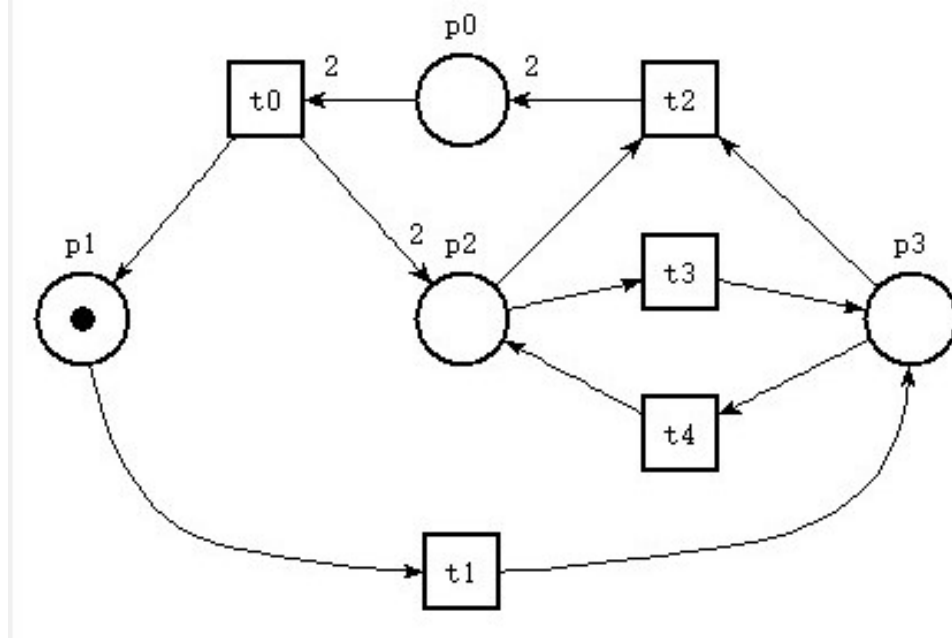


FIGURE 9 – Réseau borné, sans blocage, non réversible, ni vivant ni quasi-vivant

### 3\*. Modifications avec changement des poids des arcs de t0

#### 3.1 Réseau borné, quasi vivant avec blocage r\_b1s0r0q1v0.ndr

— Configuration :

- Marquage initial :  $M_0 : p_0 = 3, p_1 = 0, p_2 = 0, p_3 = 0$
- Transition modifiée :  $t_0 : p_0^3 \rightarrow p_1 + p_2$

— Analyse Tina :

```

REACHABILITY ANALYSIS -----
bounded
LIVENESS ANALYSIS -----
not live
not reversible
1 dead marking(s), 1 live marking(s)
0 dead transition(s), 0 live transition(s)
dead marking(s): 4

```

— Propriétés vérifiées :

- **Borné** (b1) : bounded
- **Quasi-vivant** (q1) : Toutes transitions franchissables au moins une fois
- **Avec blocage** (s0) : Marquage mort présent
- **Non réversible** (r0) : not reversible
- **Non vivant** (v0) : not live

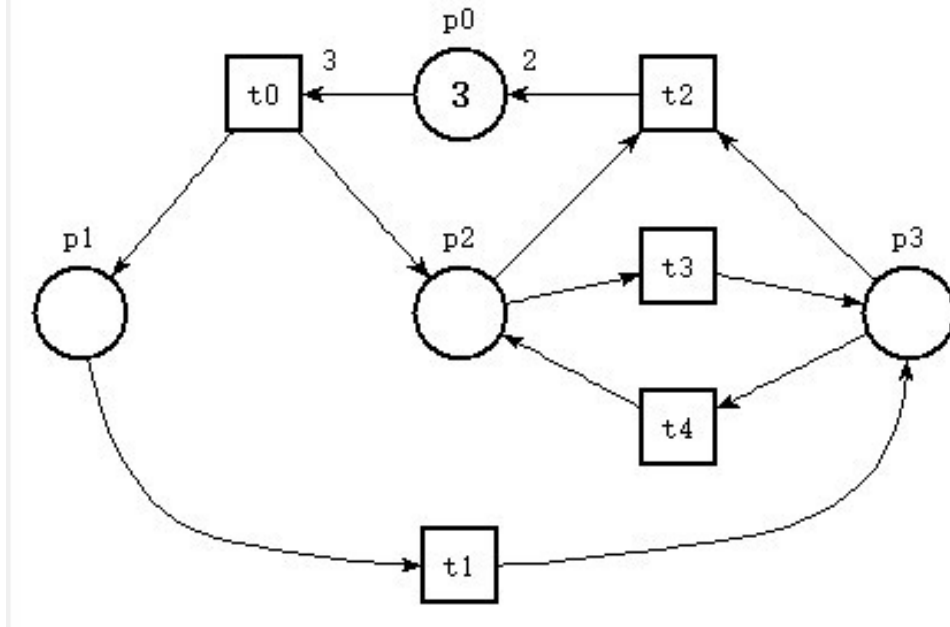


FIGURE 10 – Réseau borné, quasi vivant avec blocage

### 3.2 Réseau borné, quasi vivant, sans blocage, non réversible, non vivant r\_b1s1r0q1v0.ndr

— Configuration :

- Marquage initial :  $M_0 : p_0 = 3, p_1 = 0, p_2 = 0, p_3 = 0$
- Transition modifiée :  $t_0 : p_0^3 \rightarrow p_1^2 + p_2$

— Analyse Tina :

```

REACHABILITY ANALYSIS -----
bounded
LIVENESS ANALYSIS -----
not live
not reversible
0 dead marking(s), 2 live marking(s)
0 dead transition(s), 2 live transition(s)

```

— Propriétés vérifiées :

- **Borné** (b1) : bounded
- **Sans blocage** (s1) : Aucun marquage mort

- **Quasi-vivant** (q1) : Toutes transitions franchissables
- **Non réversible** (r0) : not reversible
- **Non vivant** (v0) : not live

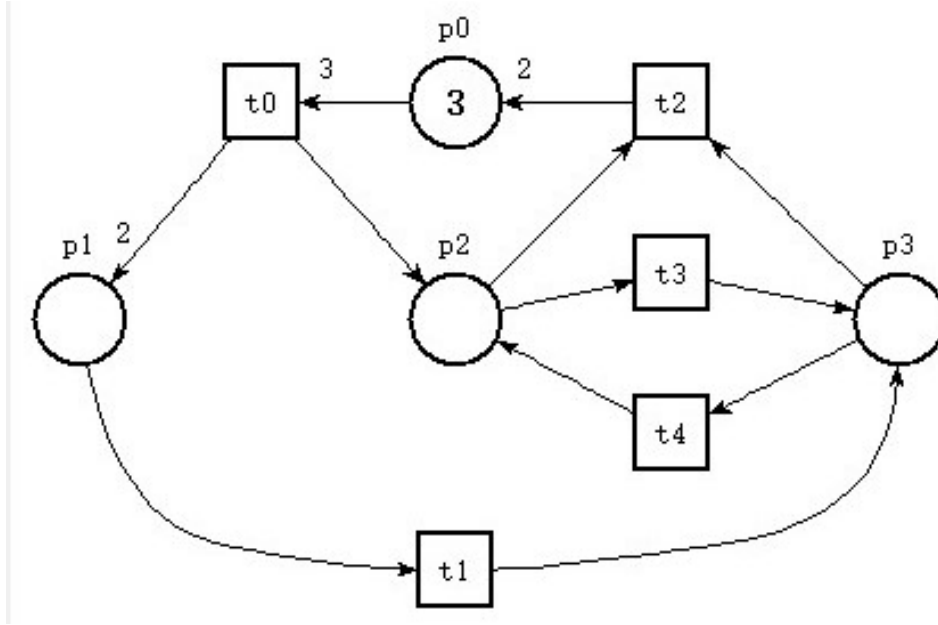


FIGURE 11 – Réseau borné, quasi vivant, sans blocage, non réversible, non vivant

### 3.3 Impossibilité du réseau borné, quasi vivant, sans blocage, réversible mais non vivant Preuve d'impossibilité :

#### — Définitions :

- **Quasi vivant** : Toute transition franchissable au moins une fois depuis  $M_0$
- **Réversible** : Retour à  $M_0$  possible depuis tout marquage accessible
- **Sans blocage** : Au moins une transition franchissable dans tout marquage
- **Vivant** : Toute transition franchissable depuis tout marquage accessible

#### — Implication logique :

$$\text{Réversible} \wedge \text{Quasi vivant} \Rightarrow \text{Vivant}$$

#### — Preuve :

1. Soit  $t$  une transition quelconque et  $M$  un marquage accessible
2. Par réversibilité :  $\exists \sigma_1$  tel que  $M \xrightarrow{\sigma_1} M_0$
3. Par quasi-vivacité :  $\exists \sigma_2$  tel que  $M_0 \xrightarrow{\sigma_2} M'$  avec  $t \in \sigma_2$
4. Donc :  $M \xrightarrow{\sigma_1 \sigma_2} M'$  avec  $t$  franchissable
5. Ainsi  $t$  est vivante

- **Conclusion** : La combinaison b1s1r1q1v0 est mathématiquement impossible

### 3.4 Réseau borné, vivant et réversible r\_all1.ndr

— Configuration :

— Marquage initial :  $M_0 : p_0 = 2, p_1 = 0, p_2 = 0, p_3 = 0$

— Arc modifié : Poids de sortie de t0 vers p2 réduit

— Analyse Tina :

```
REACHABILITY ANALYSIS -----
bounded
LIVENESS ANALYSIS -----
live
reversible
0 dead marking(s), 6 live marking(s)
0 dead transition(s), 5 live transition(s)
```

— Propriétés vérifiées :

— **Borné** (b1) : bounded

— **Vivant** (v1) : live

— **Réversible** (r1) : reversible

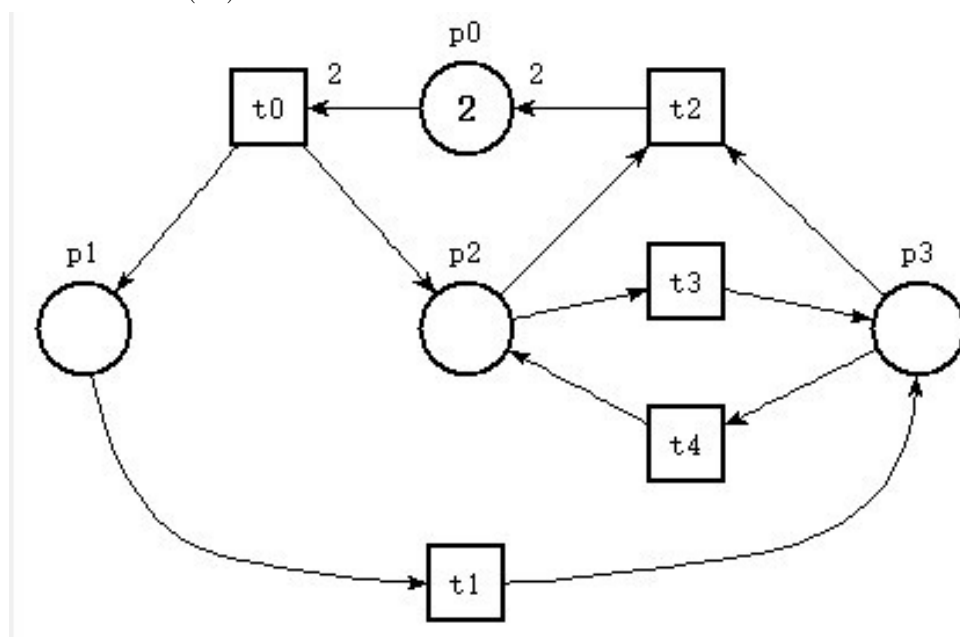


FIGURE 12 – Réseau borné, vivant et réversible

## 3 Modélisation

### Exercice 4 – Variantes des modèles de TD

**1\*. Modélisation du processus de montage** Nous avons modélisé le processus de fabrication d'une voiture en bois à l'aide d'un réseau de Petri composé de cinq transitions `init`,

fabriquer\_roues, sculpter, peindre et assembler, ainsi que de sept places représentant l'état d'avancement des différentes pièces (besoin c, besoin de roues, roue prête, c sculptée, carrosserie prête, commande, voiture prête).

La transition `init` consomme la commande initiale et produit les besoins de fabrication ; la transition `assembler` consomme la carrosserie sculptée et peinte ainsi que les quatre roues, et produit une `voiture prête`. Aucune transition ne régénère la place `commande`, ce qui signifie qu'une seule voiture est assemblée.

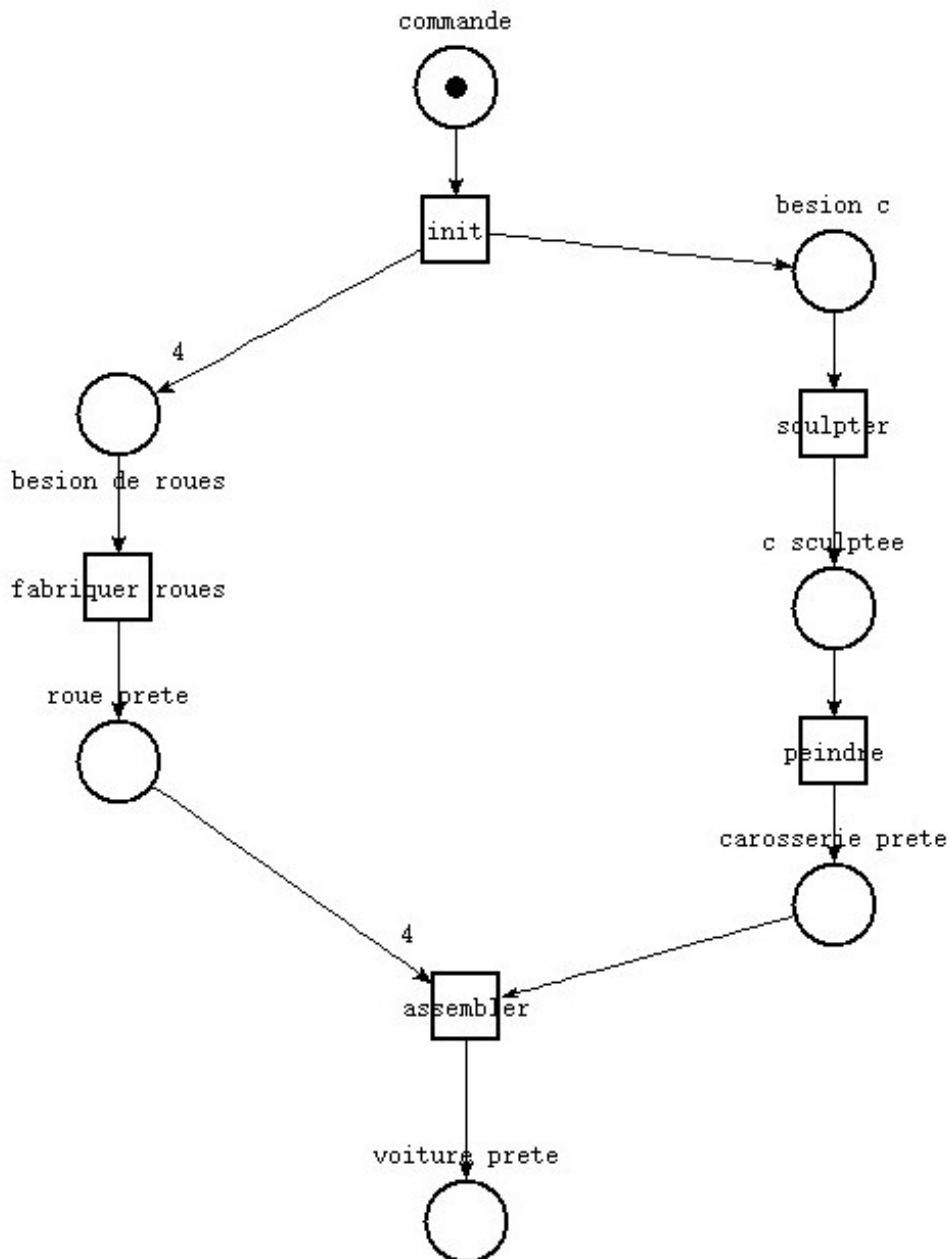


FIGURE 13 – Réseau de Petri modélisant le processus de fabrication de la voiture en bois.

L'analyse de bornitude effectuée avec TINA montre que le réseau est *borné* et possède exactement

**17 marquages accessibles.** Le graphe des marquages accessibles confirme qu'il n'existe aucune boucle productive : la transition **assembler** mène à un marquage terminal où seul un jeton **voiture\_prête** est présent.

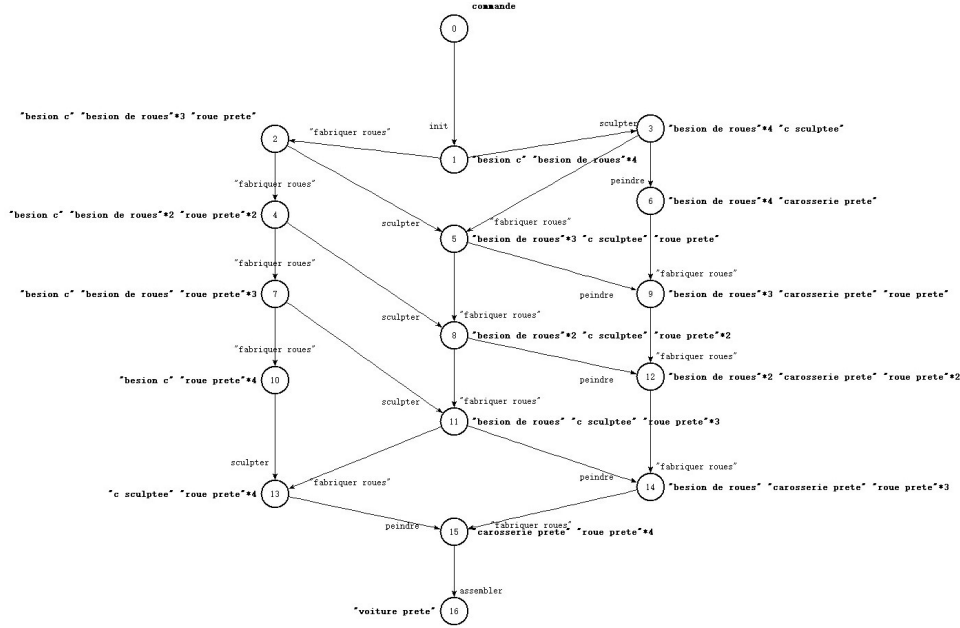


FIGURE 14 – Graphe des marquages accessibles généré à partir du fichier `.aut`.

En utilisant le simulateur pas à pas (mode *Stepper* de TINA), nous constatons que le montage d'une voiture suit une structure de choix bornée par l'ordre possible entre les activités suivantes :

- la fabrication des quatre roues (**fabriquer\_roues**) peut s'intercaler de plusieurs manières avec le branchement **sculpter** puis **peindre** ;
- la carrosserie doit être sculptée avant d'être peinte ;
- l'assemblage ne peut se produire qu'une fois toutes les pièces prêtes.

**Calcul du nombre de scénarios.** Le graphe des marquages accessibles présenté ci-dessus est un *graphe dirigé acyclique* (DAG), puisqu'aucune transition ne régénère un marquage antérieur et qu'aucune boucle productive n'est présente. Dans un tel graphe, le nombre de scénarios possibles correspond exactement au nombre de chemins distincts menant du marquage initial au marquage final.

Nous avons donc calculé ce nombre en procédant par dynamique arrière. On note  $w(m)$  le nombre de chemins allant d'un marquage  $m$  jusqu'au marquage final 16 (celui contenant uniquement un jeton dans **voiture\_prête**). On a naturellement  $w(16) = 1$ , puisqu'il n'existe aucun choix à partir de ce marquage terminal.

En parcourant le graphe de Tina de la fin vers le début, on obtient successivement :

$$\begin{aligned}
w(16) &= 1, \\
w(15) &= w(16) = 1, \\
w(13) &= w(15) = 1, \quad w(14) = w(15) = 1, \\
w(10) &= w(13) = 1, \\
w(11) &= w(13) + w(14) = 2, \\
w(12) &= w(14) = 1, \\
w(7) &= w(10) + w(11) = 3, \\
w(8) &= w(11) + w(12) = 3, \\
w(9) &= w(12) = 1, \\
w(4) &= w(7) + w(8) = 6, \\
w(5) &= w(8) + w(9) = 4, \\
w(6) &= w(9) = 1, \\
w(2) &= w(4) + w(5) = 10, \\
w(3) &= w(5) + w(6) = 5, \\
w(1) &= w(2) + w(3) = 15, \\
w(0) &= w(1) = 15.
\end{aligned}$$

On en conclut que le nombre total de scénarios possibles, c'est-à-dire le nombre de chemins distincts reliant le marquage initial au marquage final, est :

$$\boxed{15}$$

Ce résultat correspond exactement aux 15 scénarios observés dans le graphe des marquages accessibles généré par TINA.

Ainsi, le réseau modélise correctement la variabilité d'exécution du processus : les roues peuvent être produites dans n'importe quel ordre, en parallèle de la sculpture de la carrosserie, tant que les contraintes causales (sculpture avant peinture, pièces prêtes avant assemblage) sont respectées.

**2. Production parallèle de plusieurs jouets** Pour modéliser la production simultanée de plusieurs voitures en bois, nous avons augmenté le nombre de jetons dans la place **commande** pour représenter plusieurs commandes (par exemple  $N = 3$ ).

Chaque jeton **commande** déclenche la production d'une voiture complète : quatre roues et une carrosserie, qui peuvent être fabriquées en parallèle avec les autres véhicules. La transition **assembler** consomme les pièces d'une seule voiture à la fois et produit un jeton dans **voiture\_prête**.

La simulation en mode pas à pas (*Stepper*) montre que les différentes voitures peuvent être

produites dans des ordres variables, illustrant la concurrence et le parallélisme du processus. Le graphe des marquages accessibles s'élargit en conséquence, reflétant le nombre accru de scénarios possibles.

**3. Réseaux de Petri temporisés** Nous avons étendu notre modèle précédent en un réseau de Petri *temporisé*, en ajoutant des intervalles de temps aux transitions afin de modéliser la durée nécessaire à chaque étape de fabrication d'une voiture en bois par un elfe. Les intervalles choisis sont les suivants :

- `fabriquer_roues` :  $[10,20]$  unités de temps,
- `sculpter` :  $[50,75]$  unités de temps,
- `peindre` :  $[20,30]$  unités de temps,
- `assembler` :  $[0,15]$  unités de temps.

La transition `init` reste instantanée, et la logique de consommation/production de jetons est identique au réseau non temporisé.

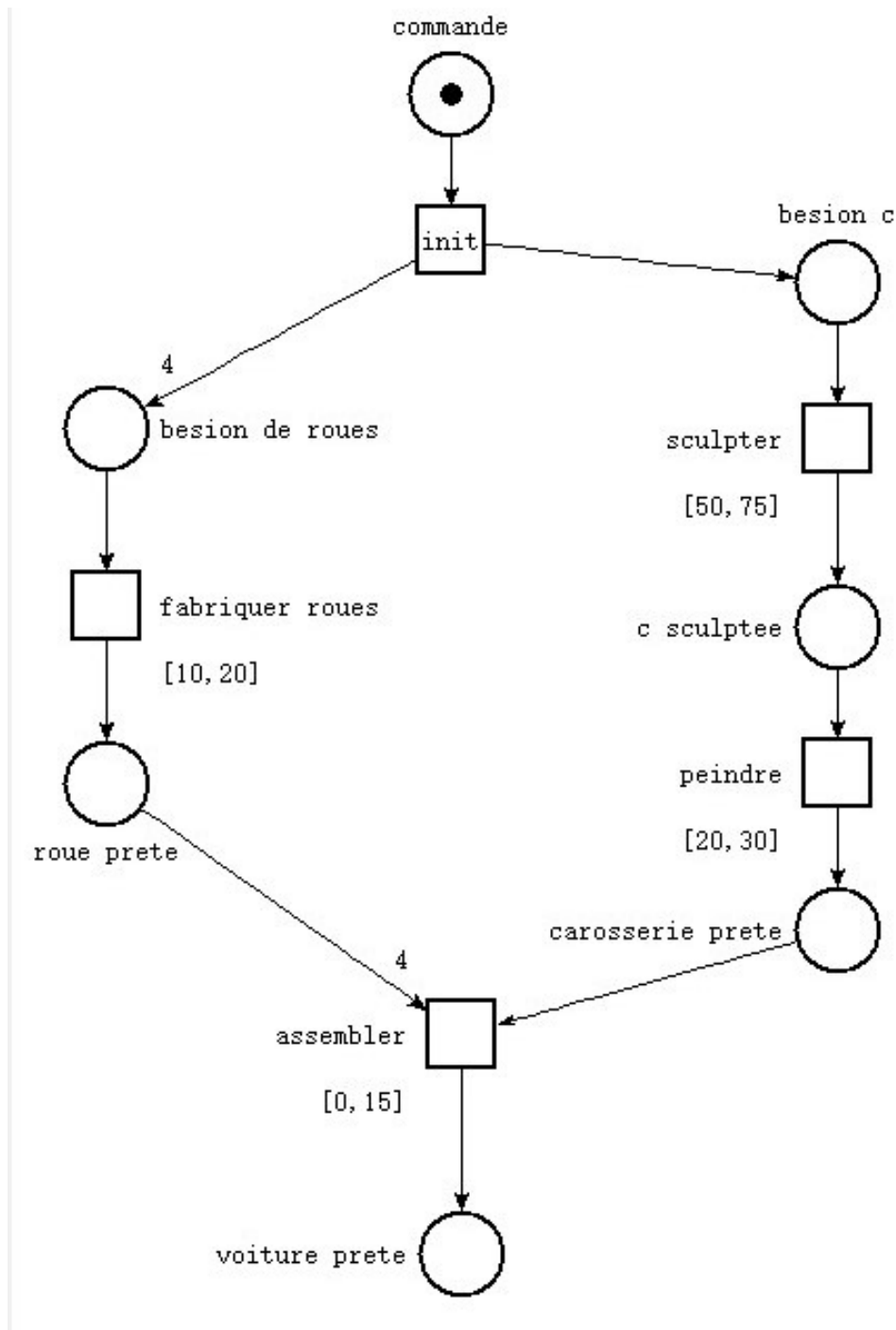


FIGURE 15 – Réseau de Petri temporisé modélisant le processus de fabrication de la voiture en bois.

L'analyse de bornitude effectuée avec TINA confirme que le réseau reste borné. Le graphe des classes d'accessibilité indique 16 classes et 19 transitions, chaque classe étant associée à un domaine temporel correspondant aux intervalles de franchissement des transitions activées.

L'utilisation du simulateur pas à pas en mode *Timed Stepper* permet de visualiser le déroulement dynamique du processus. On constate que :

- les roues peuvent être fabriquées dans un ordre variable, respectant leur intervalle  $[10,20]$  ;
- la sculpture et la peinture de la carrosserie se succèdent, chacune avec son intervalle spécifique ( $[50,75]$  pour sculpter,  $[20,30]$  pour peindre) ;
- l'assemblage ne peut se produire qu'une fois toutes les pièces prêtes, et prend entre 0 et 15 unités de temps.

Cette extension temporisée permet donc de simuler non seulement les différentes séquences d'exécution, mais aussi les variations de durée totale de production d'une voiture, reflétant plus fidèlement les contraintes réelles de temps de travail des elfes.

**4. Modélisation du salon de barbier avec capacité limitée à  $N$  clients** On considère le modèle suivant pour le salon de barbier :

- La salle d'attente peut contenir au maximum  $N$  clients.
- Le barbier ne se repose pas entre chaque client, mais se met au repos uniquement lorsque la salle est vide.
- Si le barbier est au repos, l'entrée d'un client le réveille immédiatement.
- Contrairement au cas 1, on ne représente pas un rasage en cours, c'est-à-dire qu'on considère une transition unique **rasage d'un client** sans distinguer le début et la fin.

Le réseau de Petri est défini comme suit :

$$\begin{array}{l}
 \text{Places : } P_{\text{generation}} (1), \quad P_{\text{barbier\_dort}} (1), \quad P_{\text{barbier\_travaille}} (0), \quad P_{\text{attente}} (0) \\
 \text{Transitions : } \left\{ \begin{array}{l}
 T_{\text{arrivee}} [1, 500] : P_{\text{generation}} \rightarrow P_{\text{attente}} \quad P_{\text{generation}} \\
 T_{\text{rasage}} [40, 70] : P_{\text{attente}} \quad P_{\text{barbier\_dort}} \rightarrow P_{\text{barbier\_travaille}} \\
 T_{\text{prend\_client\_suivant}} [0, w[ : P_{\text{barbier\_travaille}} \quad P_{\text{attente}} \rightarrow P_{\text{barbier\_travaille}} \\
 T_{\text{verifie\_salle\_vide}} [0, w[ : P_{\text{barbier\_travaille}} \quad P_{\text{attente}}? - 1 \rightarrow P_{\text{barbier\_dort}}
 \end{array} \right.
 \end{array}$$

**Remarques :**

- L'intervalle  $[1, 500]$  pour  $T_{\text{arrivee}}$  représente le temps entre l'arrivée des clients.
- L'intervalle  $[40, 70]$  pour  $T_{\text{rasage}}$  correspond au temps nécessaire pour raser un client.
- La capacité de la salle est simulée via la limite de  $N$  tokens dans  $P_{\text{attente}}$ .
- L'utilisation de l'arc inhibiteur  $P_{\text{attente}}? - 1$  sur  $T_{\text{verifie\_salle\_vide}}$  garantit que le barbier ne se repose que si la salle est vide.

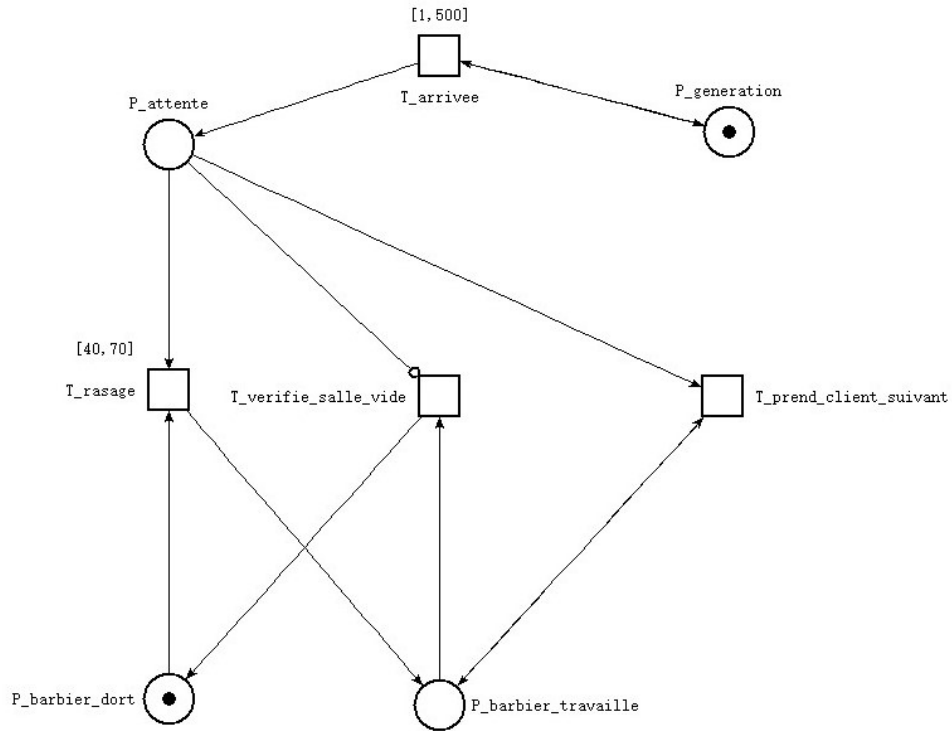


FIGURE 16 – Réseau de Petri temporisé modélisant le salon de barbier.

## Exercice 5 : Distributeur de café

On souhaite modéliser le fonctionnement d'un distributeur de café, un café valant 50 centimes. Le réseau de Petri `renduMonnaie.ndr` représente le sous-système de rendu de monnaie uniquement (la confection et la livraison du café ne sont pas modélisées).

### Spécifications du système

- La machine rend de préférence des pièces de 1 euro, puis des pièces de 50 centimes. Par exemple, pour une pièce de 2 euros, la machine rend une pièce de 1 euro et une pièce de 50 centimes si le stock le permet.
- Trois transitions extérieures : `insert50c`, `insert1e`, et `insert2e`, franchies lorsque la pièce est insérée.
- Transitions internes : `rendu1e`, `rendu50c`, `recharge` pour gérer le rendu automatique et le réapprovisionnement.
- Les arcs inhibiteurs sont autorisés.
- La machine peut recharger automatiquement le stock de pièces de 50 centimes à l'aide de la transition `recharge` qui ajoute 10 pièces.

## Réseau de Petri : renduMonnaie.ndr

```
net renduMonnaie

# Places
pl attente_piece (1)
pl montant (0)
pl stock_1e (5)
pl stock_50c (10)

# Transitions d'insertion
tr insert50c [0,w[ attente_piece -> montant
tr insert1e [0,w[ attente_piece -> montant*2
tr insert2e [0,w[ attente_piece -> montant*4

# Achat d'un café (50c)
tr acheter_cafe [0,w[ montant -> attente_piece

# Transitions de rendu
tr rendue1e [0,w[ stock_1e montant*2 -> rendu
tr rendu50c [0,w[ stock_50c montant stock_1e?-1 -> rendu

# Recharge automatique du stock de 50c
tr recharge [0,w[ stock_50c?-1 -> stock_50c*10
```

## Description des éléments

### — Places :

- `attente_piece` : la machine attend l'insertion d'une pièce
- `montant` : montant courant inséré en unités de 50 centimes
- `stock_1e` : stock de pièces de 1€
- `stock_50c` : stock de pièces de 50 centimes
- `rendu` : pièces rendues au client

### — Transitions :

- `insert50c`, `insert1e`, `insert2e` : insertion de pièces
- `acheter_cafe` : achat d'un café (consomme 50c)
- `rendue1e`, `rendu50c` : rendu des pièces, priorité 1€ puis 50c
- `recharge` : réapprovisionnement du stock de 50c

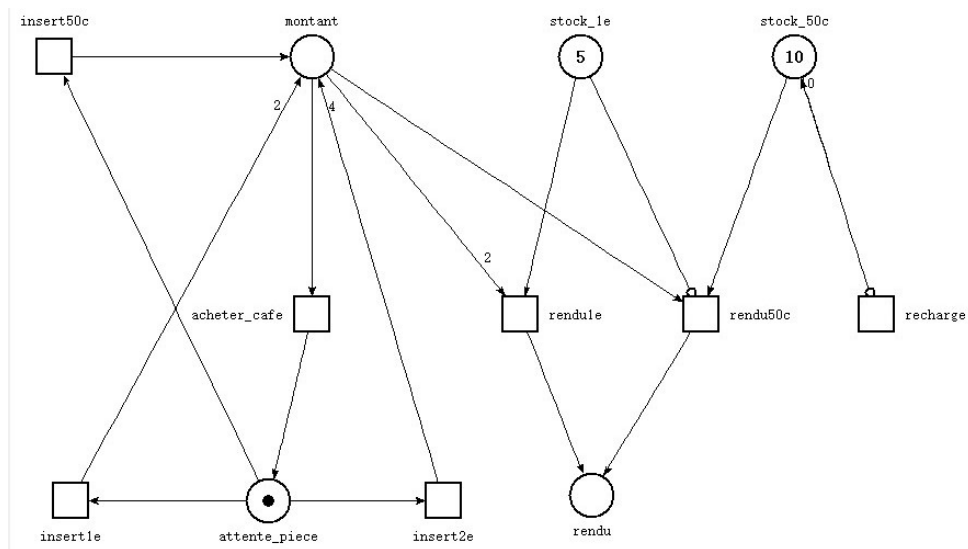


FIGURE 17 – Réseau de Petri modélisant le fonctionnement d'un distributeur de café.