

## La boîte à outils Tina

La boîte à outils Tina<sup>1</sup> propose plusieurs outils dont *NetDraw* et le *model-checker selt*. Tina est installée dans le répertoire `/mnt/n7fs/ens/tp_pantel/tina`. Les exécutables sont dans le sous-répertoire `bin`. Sa documentation est disponible dans `/mnt/n7fs/ens/tp_pantel/tina/doc`.

### Exercice 1 : NetDraw (nd)

L'outil *nd* (*Net Draw*) permet de visualiser un réseau de Petri et de le simuler. Nous allons prendre comme exemple `saisons.net` (voir listing 1).

**1.1.** Lancer l'outil *nd* avec le fichier `saisons.net`. La description textuelle du réseau s'affiche dans le fenêtre principale.

**1.2.** Pour visualiser graphiquement le réseau, faire *edit / draw* et choisir soit *neato*, soit *circo*. La représentation du réseau est obtenue en utilisant des algorithmes de placement automatique.

**1.3.** Pour animer le réseau, faire *tools / stepper simulator*. Les transitions franchissables sont marquées en rouge. Pour les déclencher, il suffit de cliquer dessus (pas toujours très facile, plutôt sur le bord inférieur ou supérieur).

Il est possible de revenir en arrière dans la simulation ou repartir vers l'avant en utilisant les boutons de contrôle. Il existe aussi un mode aléatoire.

### Exercice 2 : Analyse par *model-checking* avec *selt*

Les propriétés que l'on souhaite vérifier sur un réseau de Petri peuvent être exprimées en LTL (Logique Temporelle Linéaire). Le fichier `saisons.ltl` définit deux propriétés :

```
[ ] <> Ete; # Toujours il y aura un été
- <> Ete;   # Il n'y aura pas d'été
```

Pour vérifier ces propriétés, il suffit de taper :

```
tina saisons.net saisons.ktz
selt -p -S saisons.scn saisons.ktz -prelude saisons.ltl
```

À l'écran s'affiche les résultats (donnés au listing 2). La première propriété est vraie. La seconde est fausse et *selt* exhibe un contre-exemple. Faire `quit` pour quitter *selt*. Dans le fichier `saisons.scn` on a la séquence de transitions qui correspond au contre-exemple.

---

1. <http://www.laas.fr/tina/>

#### Listing 1 – Le réseau de Petri `saisons.net`

```
1 pl Hiver (1)
2 tr h2p Hiver -> Printemps
3 tr p2e Printemps -> Ete
4 tr e2a Ete -> Automne
5 tr a2h Automne -> Hiver
```

## Listing 2 – Les résultats de selt

```
1 cregut@abakus> selt -p -S saisons.scn saisons.ktz -prelude saisons.ltl
2 Selt version 3.4.4 -- 01/05/16 -- LAAS/CNRS
3 ktz loaded, 4 states, 4 transitions
4 0.000s
5
6 - source saisons.ltl;
7 TRUE
8 FALSE
9   state 0: Hiver
10  -h2p->
11   state 1: Printemps
12  -p2e->
13   state 2: Ete
14  -e2a->
15   state 4: Automne
16   [accepting all]
17 0.000s
```

La boîte à outils Tina<sup>2</sup> propose également des outils de vérification d'autres propriétés des réseaux de Petri que la vérification des propriétés comportementales (model checking avec `selt`).

**Exercice 3 : Détection des interblocages**

Charger le fichier fourni `interblocage.ndr`.

1. Exécuter la commande `nd interblocage.ndr`
2. Lancer le simulateur pas à pas (onglet *tools / stepper*)
3. Simuler le fonctionnement du système et trouver la situation d'interblocage.
4. Quitter le simulateur (onglet *file / return to editor*)
5. Construire le graphe des marquages (onglet *tools / reachability : marking graph – output verbose*)
6. Lire le fichier `interblocage-resultat.txt`. Les annotations sont indiquées par `##`
7. Visualiser le graphe des marquages (onglet *tools / reachability : marking graph – output lts .aut*<sup>3</sup>). Vérifier ce graphe des marquages. Le stepper pourra être utilisé sur le réseau initial.
8. Ouvrir l'automate résultant sous TINA (clic droit – *open file in nd*)
9. Le dessiner (onglet *edit : option draw*). Plusieurs heuristiques de placement sont proposées (*neato*, *draw*, *circo*, ...). Les essayer. Il faut d'abord repasser en forme textuelle (onglet *edit / option textify*).

---

2. <http://www.laas.fr/tina/>

3. aut comme automaton.

**Définition :** Le **graphe de couverture** d'un réseau de Petri (Coverability graph) est un marquage qui recouvre tous les autres marquages (un marquage maximal). Il peut être calculé symboliquement. Il contient des places avec un nombre infini de jetons si le réseau n'est pas borné.

#### Exercice 4 : Analyse des propriétés des réseaux

Charger le fichier fourni `planteur.net`.

1. Exécuter la commande `nd planteur.net`. Le réseau apparaît mais sous forme textuelle.
2. Lancer le simulateur pas à pas (onglet *tools / stepper*).
3. Dessiner le réseau (onglet *edit* : option *draw*). L'esthétique peut être améliorée !
4. Lancer le simulateur pas à pas (onglet *tools / stepper*)
5. Simuler le fonctionnement du système. Se persuader qu'il est non borné.
6. Quitter le simulateur (onglet *file / return to editor*)
7. Construire le graphe de couverture du réseau (onglet *tools / reachability : coverability graph – output verbose*).
8. Regarder le fichier résultat. Les places non bornées apparaissent avec un poids =  $w$ .
9. Dessiner le graphe de couverture (onglet *tools / reachability : coverability graph – output lts .aut*). ...

#### Exercice 5 : Création de réseau de Petri

Saisir le réseau de la Figure 1 en utilisant la forme textuelle. On utilisera `nd` pour le visualiser.

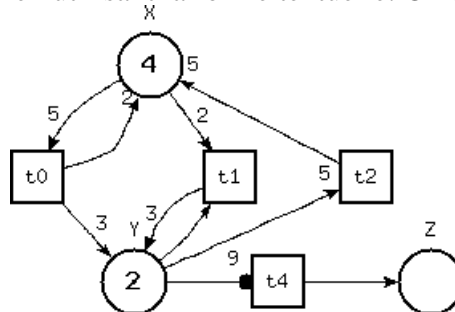


FIGURE 1 – Un exemple de réseau de Petri

#### Exercice 6 : Allocations de Ressource avec un client et deux gestionnaires de ressource

Un client ( $C_1$ ) a besoin pour travailler de posséder de manière mutuellement exclusive deux ressources ( $R_1, R_2$ ) qui sont distantes et gérées individuellement par deux gestionnaires ( $G_1, G_2$ ).

La communication entre client et gestionnaire est régie par les messages suivants :

**Req** - demande d'accès à la ressource,

**Ack** - autorisation d'accès à la ressource,

**Lib** - libération de la ressource

**Client  $C_1$  :**  $C_1$  demande séquentiellement les deux ressources. Plus précisément, il demande d'abord la ressource  $R_1$ , et lorsqu'il l'a obtenue, il demande la ressource  $R_2$ . Il entre en section critique lorsqu'il possède les deux ressources. Il libère ensuite en même temps les deux ressources et retourne à son état initial.

**Gestionnaire  $G_j$  :** À la réception d'un message Req envoyé par le client  $C_i$ ,  $G_j$  alloue exclusivement la ressource à  $C_i$  (i.e. envoi du message Ack à  $C_i$  si la ressource est libre, sinon la requête reste en attente. À la réception du message Lib, la ressource est libérée.

**6.1.** Modéliser ce système par un réseau de Petri : en vous inspirant du réseau associé au client  $C_1$  contenu dans le fichier `cl1.net` qui représente son comportement, modéliser le comportement du gestionnaire 1 (resp 2) dans un fichier `gest1.net` (resp. `gest2.net`) Le système global peut être obtenu en composant les 3 fichiers `.net` en un seul fichier en utilisant le format `tpn` (voir exemple dans `composition.tpn`). Le chargement dans les outils d'un fichier `.tpn` réalise l'assemblage des fichiers `.net` indiqués. En cas d'erreur de compilation, utiliser la commande `tina -p composition.tpn` pour voir les erreurs.

**6.2.** Simuler le système global résultant de la composition à l'aide du Stepper.

**6.3.** Construire son graphe des marquages.

**6.4.** S'assurer qu'il est exempt de blocage.

**6.5.** S'assurer qu'il est réinitialisable (ici une seule composante fortement connexe contenant tous les marquages).

### Exercice 7 : Allocations de Ressource avec deux clients et deux gestionnaires de ressource

On ajoute au système précédent un second client ( $C_2$ ) dont le comportement est décrit ci-dessous :

**Client  $C_2$  :**  $C_2$  demande séquentiellement les deux ressources. Plus précisément, il demande d'abord la ressource  $R_2$ , et lorsqu'il l'a obtenue, il demande la ressource  $R_1$ . Il entre en section critique lorsqu'il possède les deux ressources. Il libère ensuite en même temps les deux ressources et retourne à son état initial.

**7.1.** Modéliser de façon compositionnelle ce système.

**7.2.** Simuler ce système à l'aide du Stepper.

**7.3.** À l'aide du stepper, trouver des scénarios démontrant 1) la possibilité d'interblocage et 2) la possibilité de famine (i.e. un client désirant travailler travaillera-t-il nécessairement ?).

**7.4.** On s'intéresse à étudier l'accès en exclusion mutuelle à chaque ressource.

**7.4.1.** Exprimer cette propriété en utilisant les places et les marquages de votre modèle.

**7.4.2.** La simulation apporte-t-elle une garantie ?

**7.4.3.** Comment s'en assurer en analysant le graphe des marquages ?

**7.4.4.** Pour s'en assurer de façon automatique :

- Construire un réseau observateur comportant une seule transition qui sera tirable si et seulement si cette propriété d'exclusion est violée.
- Composer cet observateur avec le reste du système et s'assurer que cette transition est « morte ».
- Prendre soin à définir un observateur non « intrusif » (i.e. qui ne modifie pas le comportement du système observé).