# Algèbres de réseaux de Petri

Master 2 CNS/SA
Examen de première session
8 novembre 2022

Durée totale : 2h

**Avertissement.** Lisez *attentivement* le sujet. Les calculatrices, les documents, et les téléphones portables sont interdits. Vous devez *expliquer et justifier toutes vos réponses*. Si le sujet vous semble comporter des erreurs ou imprécisions, détaillez à l'écrit et répondez de votre mieux. *Il ne sera répondu à aucune question concernant le sujet.* Cet énoncé comporte 4 pages.

**Exercice 1 (Dîner de confrères)**
Le philosophe JCVD a invité trois de ses confrères à un dîner. Il a préparé un modèle ABCD selon le principe du dîner des philosophes. La sémantique réseaux de Petri de ce modèle est donnée à la figure 1 page 2 mais JCVD a oublié d'y noter quelques détails.

```
const count = 4
buffer forks : int = range(count)

net philo (left) :
  ([forks-(left), forks-(r) if r == (left+1) % count]
   ; [forks+(left)]
   ; [forks+((left+1) % count)])
  * [False]

philo(0)
| philo(1)
| philo(2)
| philo(3)
```

*Questions:*

**1** • Indiquez les places d'entrée, les places internes, et les places de sorties sur le réseau de Petri. (*3 points*)

**2** • Indiquez les gardes des transitions. (*2 points*)

**3** • Ajoutez les annotations manquantes sur quatre des arcs reliés à la place centrale. (*2 points*)

**4** • Expliquez pourquoi ce modèle peut ou non avoir des blocages ? (Des états à partir desquels plus aucune transition ne peut être tirée.) (*3 points*)

**Exercice 2 (Le petit oiseau va sortir)**
Le but de l'exercice est de modéliser en ABCD une cabine de photo automatique (photomaton) avec les fonctionnalités suivantes :

— le client choisit d'abord le type de cliché désiré : photos d'identités (5€) ou portrait (7€) ;
— le client doit ensuite payer le prix correspondant au type de cliché choisi ;
— lorsque le client est prêt, il appuie sur un bouton pour prendre la photo ;
— l'automate prend la photo, l'affiche, et propose de la valider ou de la refaire ;
— on peut refaire la photo deux fois au maximum, la troisième prise sera donc forcément validée ;
— la photo validée est imprimée et la monnaie rendue.

*Questions:*

**1** • Quels sont les états du système ? (*2 points*)

**2** • Quels sont les événements du système ? (*2 points*)

**3** • Donnez une conditions de sûreté. (*1 point*)

**4** • Donnez une condition de vivacité. (*1 point*)

**5** • Proposez une modélisation en ABCD de ce système en expliquant votre démarche. (*4 points*)

**Répondez directement sur cette page et rendez la avec votre copie.**

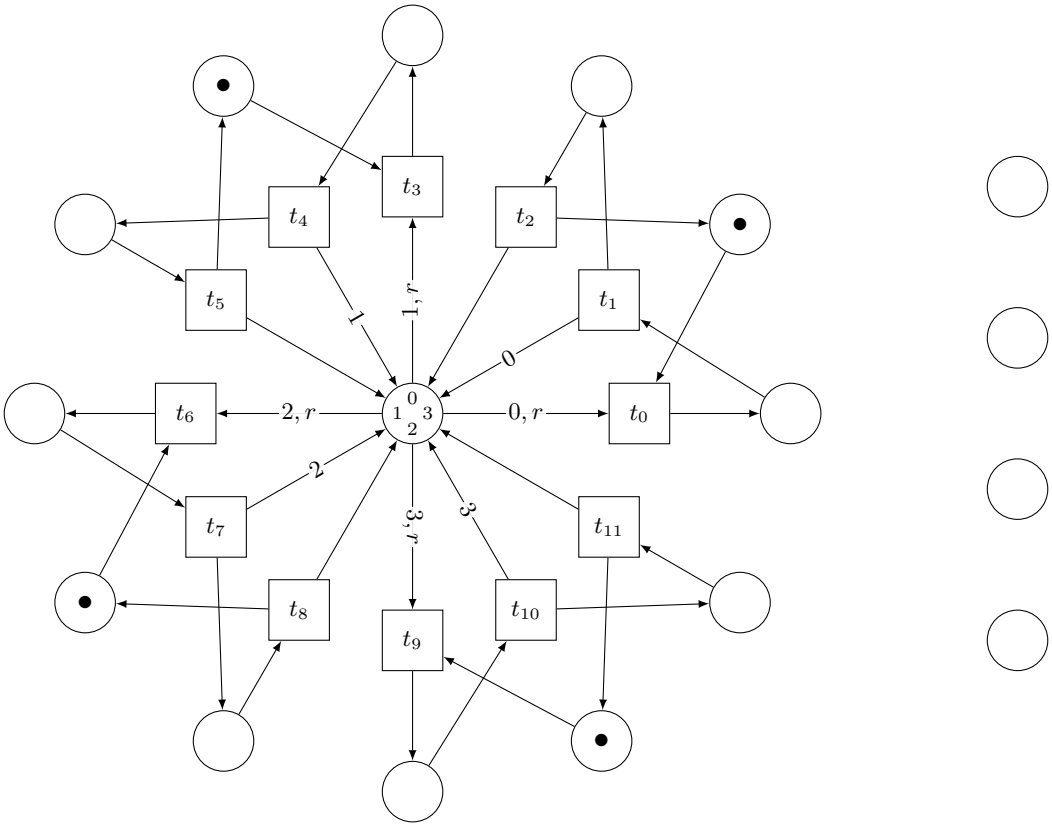



**Figure 1.** Sémantique réseau de Petri du modèle de l'exercice 1, le marquage indiqué est le marquage initial.

**Gardes des transitions :**

— $t_0$ : .....

— $t_1$ : .....

— $t_2$ : .....

— $t_3$ : .....

— $t_4$ : .....

— $t_5$ : .....

— $t_6$ : .....

— $t_7$ : .....

— $t_8$ : .....

— $t_9$ : .....

— $t_{10}$ : .....

— $t_{11}$ : .....

# ABCD cheatsheet

```
$ abcd [option]...  spec.abcd
```

`--pnml=FILE` save net as PNML (SNAKES' variant)

`--dot=FILE` `--neato=FILE` `--towpi=FILE`
`--circo=FILE` `--fdp=FILE` draw net using a GraphViz engine

`--load=PLUGIN` load a plugin before to build net (may be repeated)

`--simul` start interactive simulator

```
# a comment
# another comment
declarations
process
```

**buffer declarations:**
`buffer name: type = ()`
▷ empty buffer
`buffer name: type = val, ...`
▷ buffer with initial content

**type expressions:**
any Python type or class
▷ eg, `int`, `str`, `object`, ..., or user-defined classes
`enum(val, val, ...)`
▷ enumerated type
`type * type`
▷ cross-product of types
`type | type`
▷ union of types
`type & type`
▷ intersection of types

**types definition:**⋆
`typedef name: type`

**constants:**⋆
`const name = expr`

**symbols:**⋆
`symbol name, ...`
▷ define fresh unique values

**Python imports:**⋆
▷ just use regular Python imports

**sub-processes**

⋆ global declarations only

**control flow:**
`process ; process`
▷ sequential composition
`process + process`
▷ non-deterministic choice (use opposite guards to make it deterministic)
`process * process`
▷ iterate left-hand-side and exit with right-hand-side
`process | process`
▷ parallel composition

(no priorities ⇒ use parentheses)

**sub-process instances:**
`name(args)`
▷ substitute `args` in net `name`
scope its local buffers
insert the net

**sub-process declarations:**
`net name (params):`
indented block
declarations
process

**sub-process parameters:**
a comma-separated list of:
`name`
▷ a value is expected
`name: buffer`
▷ a buffer name is expected

**atomic actions:**
`[True]`
▷ no-op non-blocking action
`[False]`
▷ always-blocking action
`[accesses]`
▷ unguarded action
`[accesses if expr]`
▷ guarded action

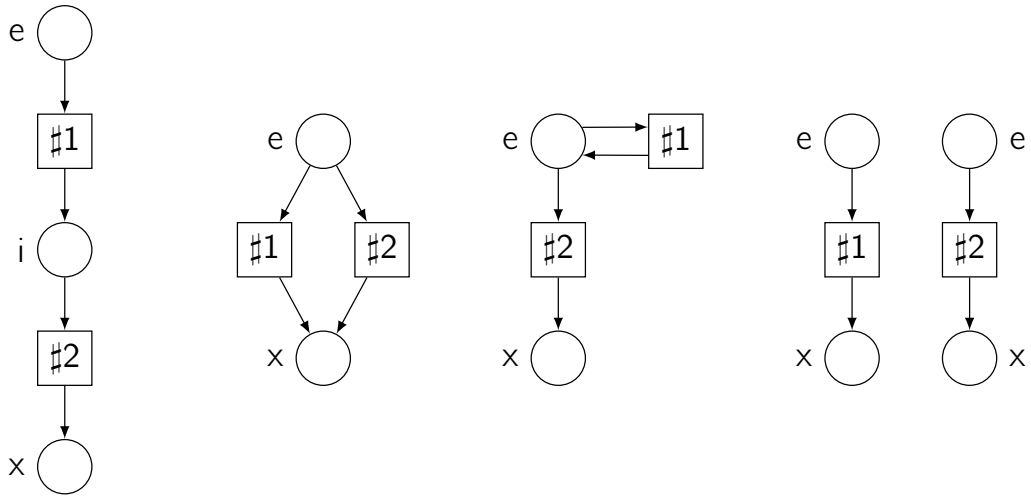
**accesses:**
`buff-(val)`
▷ consume `val` from `buff`
`buff-(var)`
▷ consume a value from `buff` and binds it to `var`
`buff+(expr)`
▷ produce a value into `buff`
`buff?(val)`
▷ test for `val` in `buff`
`buff?(var)`
▷ test for a value in `buff` and binds it to `var`
`buff>>(var)`
▷ flush `buff` into `var`
`buff<<(var)`
▷ add the values contained in `var` to `buff`
`buff<>(val=expr)`
▷ replace `val` in `buff` with `expr`
`buff<>(var=expr)`
▷ replace `var` in `buff` with `expr`

(a comma-separated list of accesses is performed atomically)

## Specifying control-flow operators

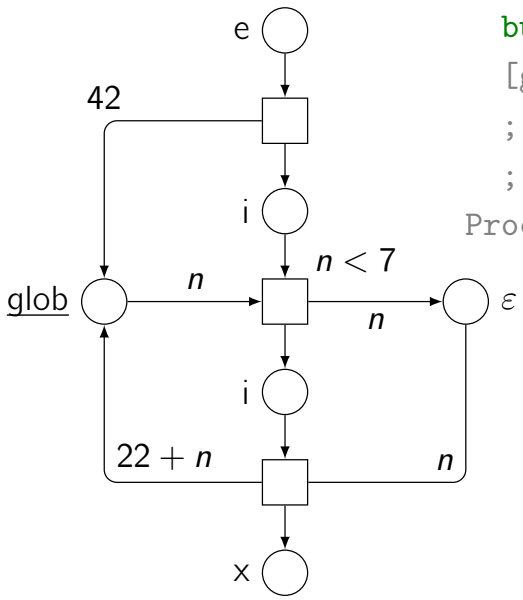
with operator nets

sequence ⨾ choice □ iteration ⊛ parallel ||






## Translating ABCD into Petri nets




```
buffer glob :  int = ()
net Process (num=22) :
  buffer loc :  int = ()
  [glob+(42)]
  ; [glob-(n), loc+(n) if n < 7]
  ; [loc?(n), glob+(22 + n)]
Process(22)
```