Modélisation et analyse des systèmes hétérogènes (DIS-MASTIC-05)

Bases

J. Christian Attiogbé

Février 2023 - Collège doctoral Pays de la Loire

Nantes Université

Plan de la suite (adaptation selon contraintes)

Présentation du cours

- Hétérogénéité des systèmes : la question, des éléments de caractérisation
- Abstractions nécessaires et leur formalisation avec des objets manipulables
- Ouverture sur l'interdisciplinarité, pendant les études/constructions/travaux

Chapitre 2 : **Abstractions - modélisations** (des systèmes hétérogènes)

☞ Construire les modèles appropriés aux composants du système

Introduction

- Modèles empiriques, modèles semi-formels

  versus
- Modèles formels (basés sur des outils mathématiques)

En fonction de ou des objectifs de la modélisation,
☞ **modèle formel des composants** :
(préoccupation : analyse par vérification de prop., prédiction fiable, ...)

Une interconnexion de composants hétérogènes

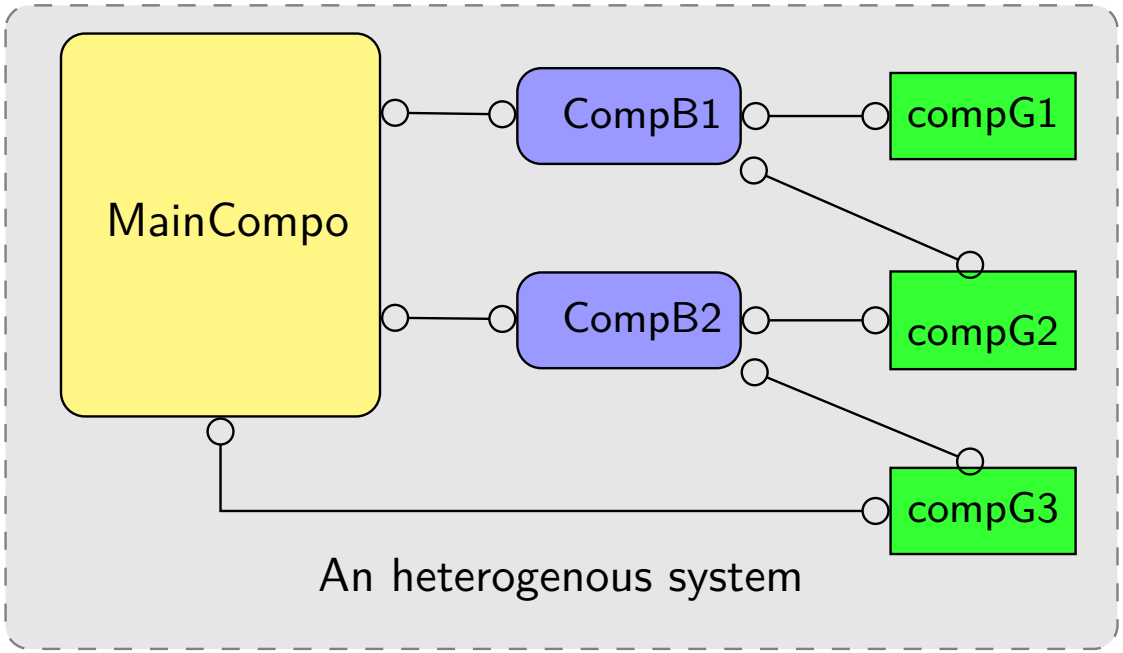


Figure: Une vision d'un assemblage de composants hétérogènes

Composants de différentes natures, qui interagissent

Une interconnexion de composants hétérogènes

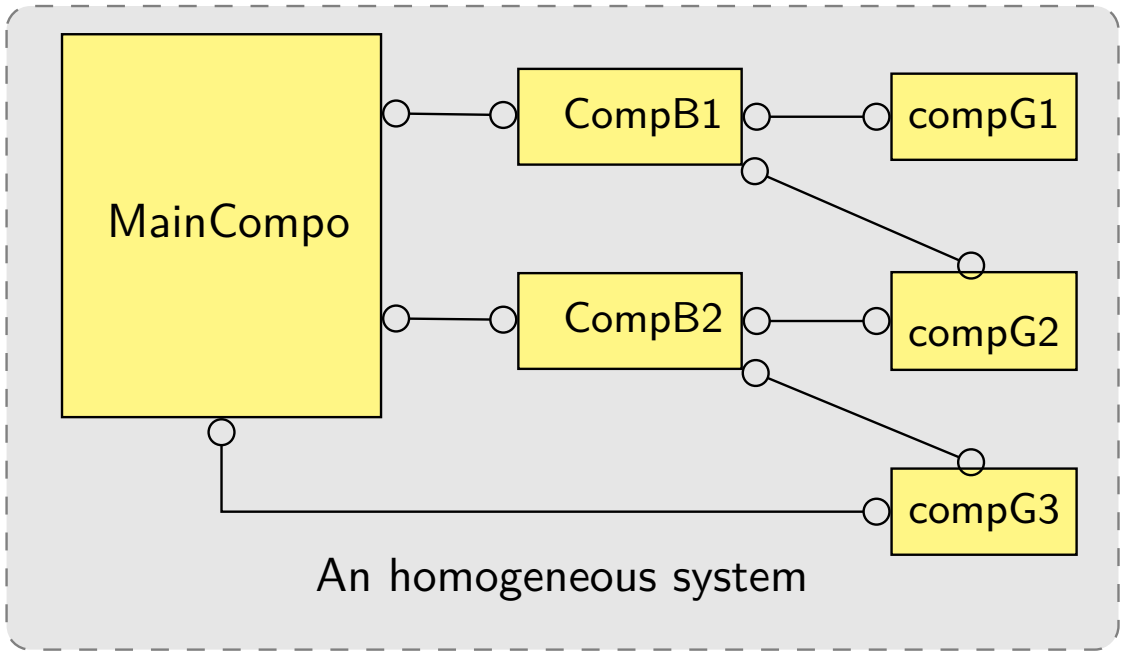


Figure: Une vision d'un assemblage de composants homogènes

Difficultés : hétérogénéité sémantique

- Modèles [**sémantiques**] différents (pour des composants)
- comment les **composer** ? les **analyser** ?

- Des familles de modèles avec des sémantiques proches
- Orchestration / plongement de sémantiques / interfaçage
- ...

A l'issue de la caractérisation d'un système

quels composants ? quels modèles ? quelles interactions ?

- Modèles mathématiques (numériques, équations différentielles, espaces vect., $\cdots$)
- Modèles physiques (lois de comportement, de réactions, ...)
- Modèles chimiques (interactions)
- Modèles biologiques (modèles d'évolutions)
- **Modèles (formels) informatiques**
- $\ldots$

Un focus préalable sur la variété des modèles informatiques

Abstractions nécessaires et leur formalisation

Modèles informatiques-mathématiques (vs mathématiques, physiques, chimiques, biologiques, ...) pour la dynamique dans les systèmes

modèles formels en Informatique

- Modèle à états : automates et extensions
- Modèles stochastiques, modèles probabilistes
- Modèle temporisés
- Réseaux de Petri
- Algèbres de processus
- Modèles à événements discrets
- Modèle logiques / algébriques / équationnels
- ...

Abstraction - système

Abstraction d'un système hétérogène

Un système hétérogène est un ensemble de composants (reliés et) en interaction = une interconnexion de composants : c'est **un graphe**.

Mais il **y a des interactions** entre les composants du graphe.

Graphe

Un système hétérogène est, structurellement, un graphe connexe, dont les nœuds sont des composants et les **arcs, les relations support des interactions** entre les composants.

Graphes

Modélisation à l'aide des graphes

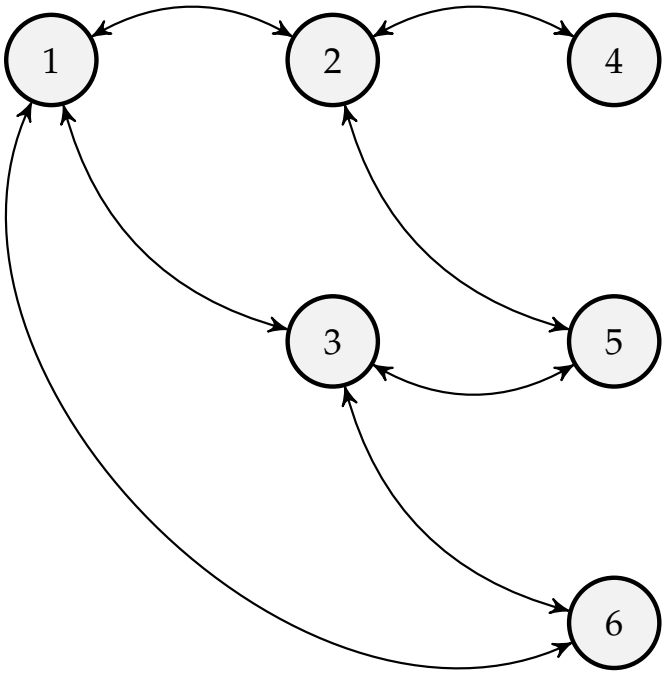
Graphe

Soit N un ensemble fini de nœuds (ou sommets).
Un graphe G est une relation sur N, telle que $G \subseteq N \times N$.
ou bien $G = \{(n, m) \in N \times N\}$

- Les nœuds sont nommés et peuvent avoir des attributs spécifiques
- Les arcs peuvent être pondérés, orientés
- Les graphes sont de différentes natures : orienté, non orienté, centralisé, cyclique, ...

Exemples de graphe

Un graphe



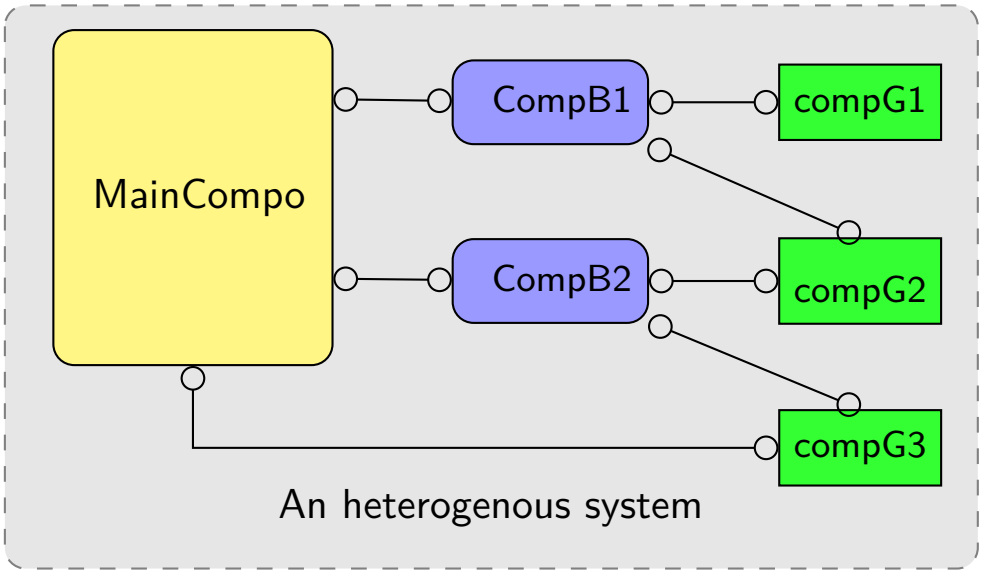
Caractéristiques

- Topologie d'un système
- Très générique
- Nœuds et les arcs peuvent être particularisés
- Arcs orientés ou non
- Accessibilité/distance/etc
- Plusieurs algorithmes existent

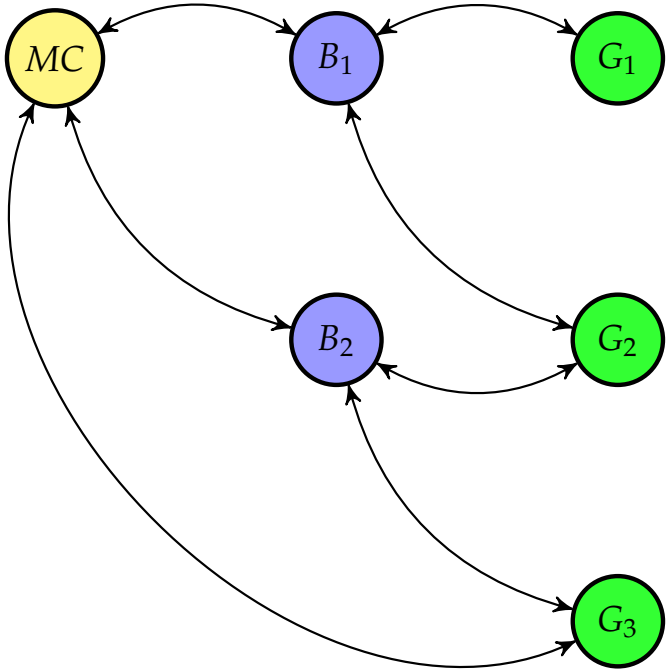
☞ le graphe est une structure statique ! quid des interactions ?

Exemples de graphe

Interconnexion de composants



Graphe



☞ graphe statique ! quid des comportements ? des interactions ?

Des abstractions

Abstractions pour les comportements

Automates

Automates à états

Automate à états

Un automate à états est défini par un 5-uplet : $\langle S, A, \delta, S_0, S_f \rangle$

- Ensemble d'**états** fini: $S = \{S_0, \cdots, \cdots\}$
- Un alphabet d'**actions** (ou **étiquettes** ou **symboles**) : A
- Une relation de **transition** δ définie sur $(S \times A)$ et S comme suit : $\delta : (S \times A) \leftrightarrow S$
  on note $s_1 \xrightarrow{\alpha} s_1' \in \delta$ pour $((s_1, \alpha),\ s_2) \in \delta$
- Un **état initial** S_0 (c'est à dire $S \in S_0$)
- Un ou des **états finaux** S_f (c'est à dire $S_f \subseteq S$)

Un automate défini le comportement (la dynamique) d'un système ou d'un composant.

Automate

Etat

Chaque état représente l'image à un instant donné, de tout le composant/système.
A un instant donné le composant/système est dans un seul état.

Automate fini déterministe (FDA)

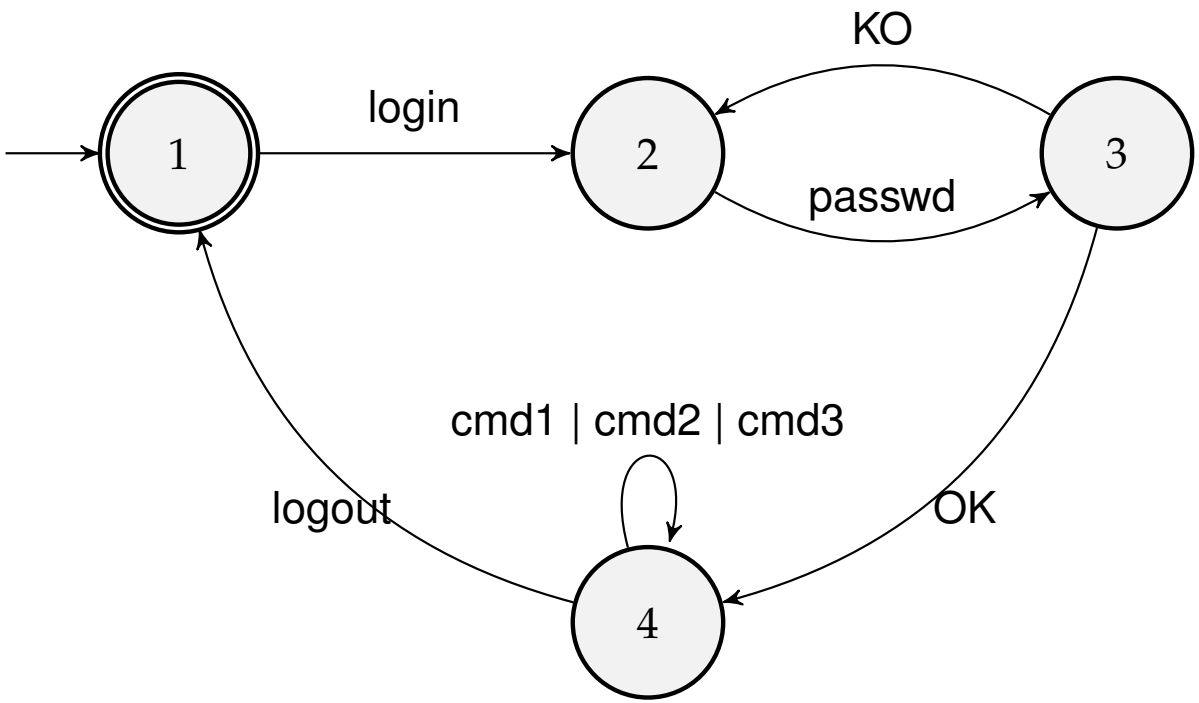
L'automate est **déterministe** lorsque il y a **un seul état initial** et la relation de transition δ est une **fonction** (et non une relation).

Automate fini non déterministe (NFA)

L'automate est **non-déterministe** lorsque : il a plusieurs **états initiaux** ou la relation de transition δ est bien une **relation** (et non une fonction) - c'est à dire, partant d'un état et d'un symbole, il y a plusieurs états suivants.

Automate : représentation graphique

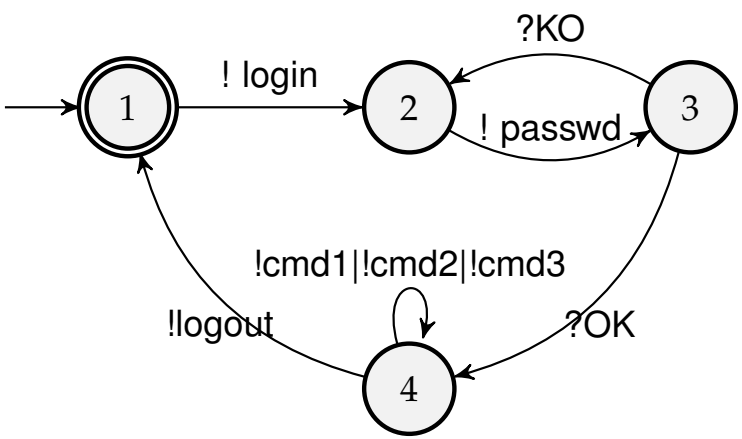
Exemple: Modèle à états d'un **processus client** qui se connecte à un processus serveur (pour une interaction/coopération à la réalisation d'une tâche).



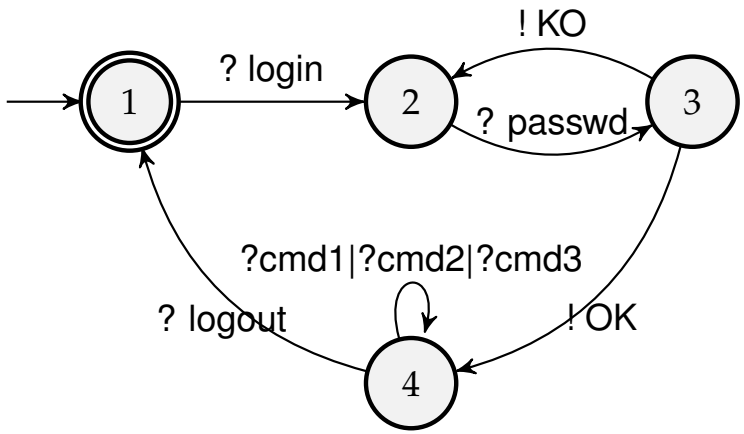
Composition d'automates : introduction

Exemple: Modèle à états de l'interaction entre un **processus client** qui se connecte au un processus serveur (pour la réalisation d'une tâche).

Client.



Serveur.

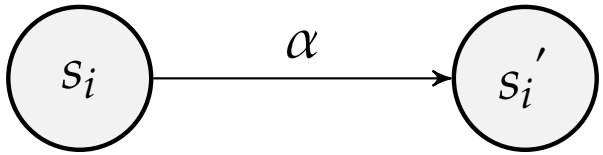


Produit synchrone

Synchronisation sur les actions (alphabet partagé)
Ici, canaux de communication implicites

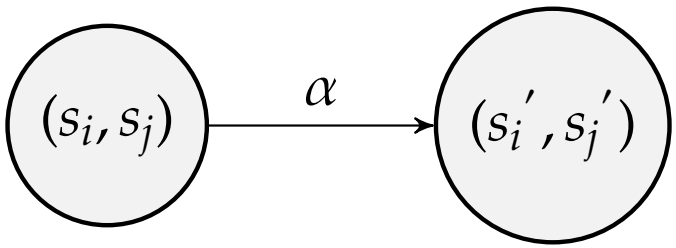
Produit d'Automates

Soient $A_1 = \langle S_1, A, \delta_1, S_{1_0}, S_{1_f} \rangle$ et $A_2 = \langle S_2, A, \delta_2, S_{2_0}, S_{2_f} \rangle$
avec $(s_i \xrightarrow{\alpha}_1 s_i^{'})$ pour $((s_i, \alpha), s_i^{'}) \in \delta_1$



Produit synchrone de A_1 et A_2

$A_1 \oplus A_2 = \langle S_1 \times S_2,\ A,\ \delta,\ S_{1_0} \times S_{2_0},\ S_{1_f} \times S_{2_f} \rangle$
avec $\delta = \{\ ((s_1, s_2) \xrightarrow{\alpha} (s_{1'}, s_2^{'}))\ \mid\ (s_1 \xrightarrow{\alpha}_1 s_1^{'}) \wedge (s_2 \xrightarrow{\alpha}_2 s_2^{'})\}$



Systèmes de transition étiquetés (*Labelled Transition Systems*)

Systèmes de transition (*LTS*) : abstraction

Definition (système de transition)

Un système de transition étiqueté par L est un tuple $\langle S, L, \rightarrow \rangle$,
où S est un ensemble d'états, L un ensemble d'actions, et
$\rightarrow \subseteq ((S \times L) \times S)$
on note : $(s \xrightarrow{a} s')$ pour $(s, a, s') \in \rightarrow$

L dénote des actions, des conditions , des événements, etc en fonction de ce qu'on veut modéliser.
Nombre d'états potentiellement infini
nombre de transition potentiellement infini.
Pas d'état initial, ni d'état terminal.

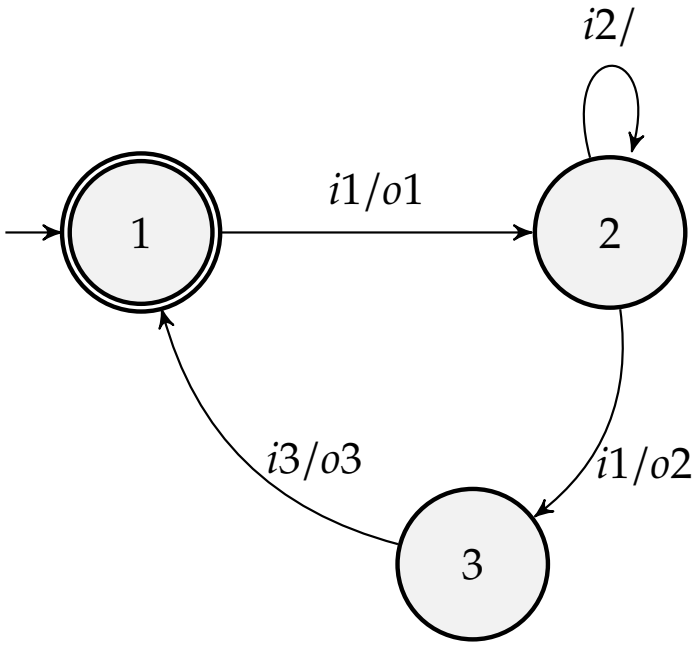
Un automate à états est un système de transition (mais pas l'inverse).

Automates de Mealy

Machine à états de Mealy : abstraction

Un automate de Mealy est défini par un n-uplet
$\langle S, A = In \cup Out, \delta, \delta_o, S_0, S_f \rangle$

- Ensemble d'états : S
- Ensemble d'entrées : In (événements d'entrées, conditions)
- Ensemble de sorties : Out (actions de sortie, traitements)
- Un état initial : S_0
- **Deux relations(fonctions) :**
  - transition $\delta : S \times In \rightarrow S$
  - **sortie** $\delta_o : S \times In \rightarrow Out$
- Etat final S_f



Automate de Mealy

- Notation des transitions : $S_s \xrightarrow{\textbf{i/o}} S_t$
- si l' **entrée** i est reçue alors que le système est dans l'état Ss, la **sortie** o **est produite** et le nouvel état du système est St
- i est aussi appelé le déclencheur (*trigger*).

La **notation des transitions est étendue** de la façon suivante : les transitions entre états ont la forme evt [garde] / actions*.

evt est l'entrée reçue/lue.
[garde] : garde est une condition sur l'état du système après l'événement evt.
action* est une suite de 0 ou plusieurs actions.

Automates de Mealy : abstraction

Une forme condensée d'une machine à états. Utilisé pour représenter les comportements de programme (logiciel).

On peut spécifier pour le programme les étiquettes de chaque ligne/instruction : les *location*.

Abstraction pour tenir compte de l'explosion des états. Plusieurs variables + plusieurs transitions → faramineux !
On peut abstraire les états et considérer plusieurs locations à l'intérieur d'un même état.

Graphe d'un programme : abstraction

On construit alors un graphe de comportement du programme où les nœuds sont les locations et les arêtes sont transitions gardées.

Le graphe d'un programme sur un ens. Var de variables typées.

Graphe d'un programme

C'est un n-uple $\langle Loc, Act, Effect, \hookrightarrow, Loc_0, g_0 \rangle$:

- Loc est un ens de locations, Act un alphabet d'actions,
- $Effect : Act \times Eval(Var) \to Eval(Var)$ une fonction d'effet
- $\hookrightarrow \subseteq Loc \times Cond(Var) \times Act \times Loc$, la relation de transition gardée,
- $Loc_0 \subseteq Loc$ est un ensemble de locations initiales,
- $g_0 \in Cond(Var)$ est la condition/garde initiale.

on note : $l \xhookrightarrow{g\,:\,\alpha} l'$ pour $(l, g, \alpha, l') \in \hookrightarrow$

Automates temporisés (*Timed Automata*)

Automates temporisés : introduction

Temps dans la modélisation (et analyse) : temps passé dans les états, durée des transitions.

Temps : continu ou discret ? (souvent discret dans les approches de modélisation ; simple)

Les systèmes de transition, comme abstraction, sont limitées (abstraites, insuffisantes) pour modéliser des systèmes asynchrones, avec des contraintes de temps.
Différentes échelles de temps, ...

Une abeille (processus) s'approche d'une fleur, se pose sur la fleur, reste ou pas pendant une durée, repart, ...

Automates temporisés : introduction

Modélisation des systèmes :

On peut utiliser plusieurs/des horloges (*clock*) ;
elles peuvent progressent de la même façon (synchronisées, raz);
ainsi, plusieurs variables d'horloge dans les modèles, avec des contraintes.

La valeur (réelle, $\mathbb{R}$) d'une horloge dénote le temps écoulée depuis sa dernière *reset* ; on ne peut que lire et raz une horloge.

Les transitions sont conditionnées par des contraintes de temps (exprimées avec les variables d'horloge et des constantes entières : $x > c,\ \ y = c,\ \ x > c \wedge y = c, \cdots$).

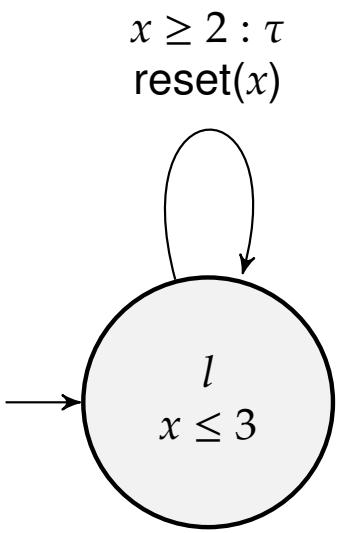
Automates temporisés (*Timed Automata*) : intuition

On se base sur le graphe d'un programme, augmenté par les contraintes de temps.
Une arête (du graphe) dans un automate TA est étiqueté par : une garde, une action, et un ensemble d'horloge (à réinitialiser).

Locations: étiquettes et nœuds.
Une location est assortie d'un invariant qui contraint la durée maxi à passer dans la location ; la location doit être quittée après.

Un automate temporisé est un graphe de programme assorti d'un ensemble fini d'horloges C. Les arêtes de l à l' sont étiquetées par (g, α, D): une garde (contraintes de temps), une action et un ens. d'horloges à réinitialiser.

Automates temporisés (Timed Automata) : exemple



L'horloge x croit jusqu'à 2 **ou 3**, puis est remis à 0, ainsi de suite

L'état du TA est déterminé par la location et les valeurs des horloges !

Le programme/processus reste dans l, car une évolution (transition) interne τ est effectuée.

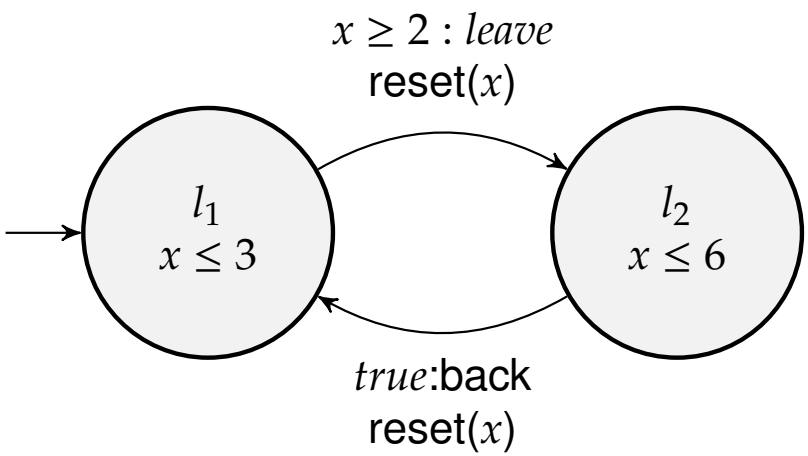
Automates temporisés (Timed Automata) : définition

Definition (Automate temporisé)

C'est un n-uplet $\langle Loc, Act, C, \hookrightarrow, Loc_0, Inv, AP, L\rangle$:

- Loc est un ens de locations, Act un alphabet d'actions,
- C un en. fini d'horloges, avec $CC(C)$ l'ensemble de contraintes sur C, $ACC(C)$ les contraintes atomiques sur C,
- $\hookrightarrow \subseteq Loc \times CC(C) \times Act \times 2^C \times Loc$, la relation de transition gardée,
- $Loc_0 \subseteq Loc$ est un ensemble de locations initiales,
- $Inv : Loc \to CC(C)$ une fonction d'affectation d'invariant
- AP un ens fini de propositions atomiques,
- $L : Loc \to 2^{AP}$ une fonction d'étiquettage des locations par AP.

Automates temporisés (Timed Automata) : exemple



Un processus qui reste un moment dans l_1, peut quitter après 2 unités, reste un moment(6 unités) dans l_2 puis revient à l.

Automates temporisés : sémantique

Un automate temporisé est un système de transition potentiellement infini, où
- les états sont des **paires : (location, valuation des horloges)** ; avec
- les **transitions** (étiquetées, **contraintes** ou non) entre ces états.

Pour chaque horloge x, $valeur(x) \in \mathbb{R}_+$
Les étiquettes peuvent être : soit le temps écoulé pour une/les horloges, soit l'action contrainte de transition dans le TA.

A partir de $\langle Loc, Act, C, \hookrightarrow, Loc_0, Inv, AP, L \rangle$ on définit
la sémantique par le système de transition $\langle S, Act', \rightarrow, I, AP', L \rangle$
avec $S = \cdots$, $Act' = \cdots, \cdots$

Automates probabilistes

Automates probabilistes

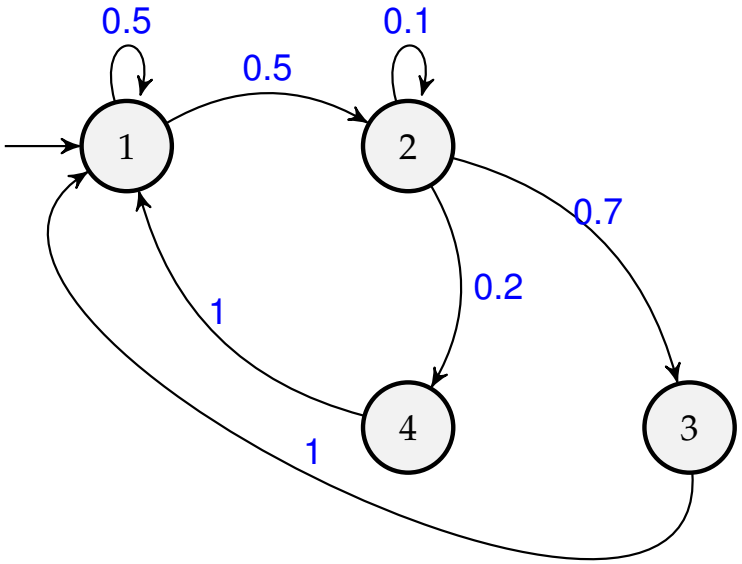
Chaînes de Markov.

Les chaînes de Markov sont des systèmes de transition avec **des distributions de probabilités pour atteindre les états** successeurs. Au lieu d'un choix non-déterministe des transitions, à partir d'un état, on choisit l'état suivant en fonction de la probabilité de la transition pour l'atteindre.

- La transition vers un état ne dépend que de l'état courant ; l'historique n'a aucun effet sur les transitions (*memoryless*).
- Les transitions sont modélisées par une matrice de transition, contenant les probabilités d'aller d'un état à un autre.

Automates probabilistes : chaîne de Markov

Châine de Markov



et sa matrice de transition

S \ S	1	2	3	4
1	0.5	0.5	0	0
2	0	0.1	0.7	0.2
3	1	0	0	0
4	1	0	0	0

Automates : modélisation, algorithmes, extensions,...

Travaux de Henzinger et *Al.*

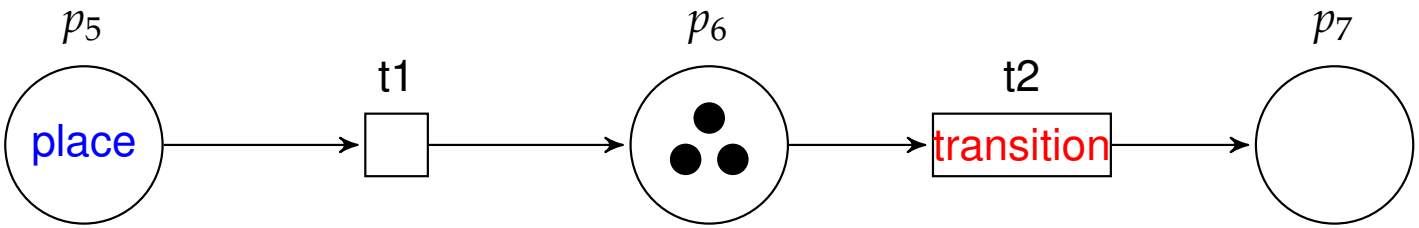
- ☞ *Principles of Model checking*, Christel Baier, J.P. Katoen, K. G. Larsen, 2008, MIT Press
- *The theory of hybrid automata*
  T.A. Henzinger.
  logic in computer science (1996)
- *Discrete abstractions of hybrid systems*
- Interface automata
  Luca de Alfaro;Thomas A. Henzinger.
  foundations of software engineering (2001)
- *Handbook of Model Checking*
  `https://readthedocs.web.cern.ch/download/attachments/206440519/HandbookofModelCheckingbyEdmundM.Clarke%2CThomasA.Henzinger%2CHelmutVeith%2CRoderickBloem(eds.).pdf`

Graphe versus Automates à états

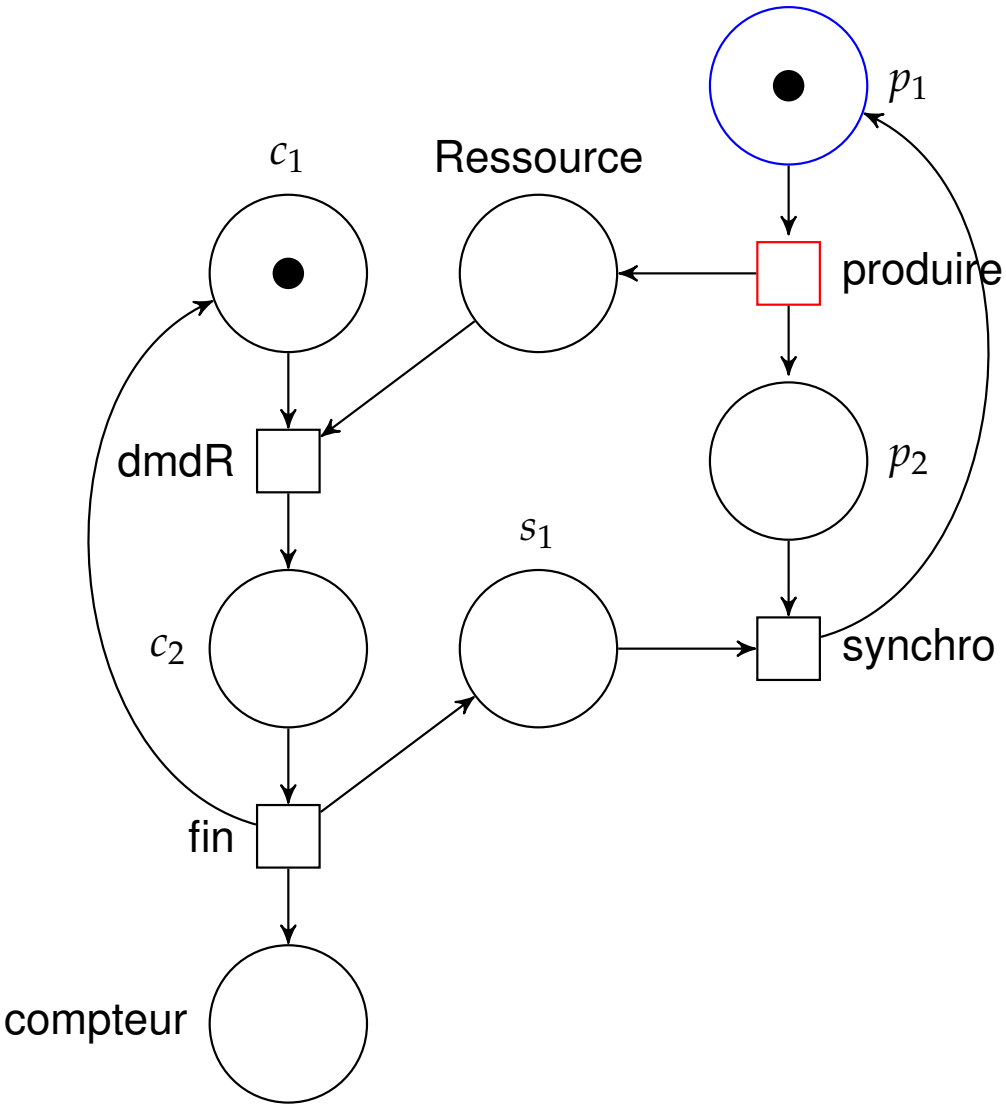
Graphe	Automate
Décrit une structure Statique pas d'évolution mémorise un état Nœuds	Décrit un comportement Dynamique des états successifs (évolution) tout l'état change déterministe/non états | état initial | états finaux
Exemples	
Réseaux d'ordinateurs Composants d'un avion	 Exécution d'un programme Comportement du train d'atterissage

Réseaux de Petri (RdP) - ÍPetri Nets

Exemples de réseaux de Petri



Exemples de réseaux de Petri



Place
Transition
Ensemble de places
Ensemble de transitions
Jetons
Marquage
Etat
Vivacité
Interblocage

Marquage : $(c_1, c_2, cpt, R, s_1, p_1, p_2)$
$(1, 0, 0, 0, 0, 1, 0)$

Définition formelle d'un RdP et notations

Un réseau de Petri (R) est un quadruplet $(P,\ T,\ Pre,\ Post)$ où :

- P : un ensemble fini de places (avec $|P| = m$, le cardinal de P),
- T : un ensemble fini de transitions, disjoint de P, ($|T| = n$)
- $Pre : P \times T \rightarrow N$ une application d'incidence avant (une matrice) : les places entrant dans une transition
- $Post : P \times T \rightarrow N$ une application d'incidence arrière (une matrice) : les places sortant d'une transition

Notations étendues : $Pre(*, t)$, $Pre(t)$, $Post(*, t)$, $Post(t)$

Il reste le marquage (où sont les jetons) !

Réseaux de Petri : Marquage

Un réseau marqué est le couple $N = (R, \mu)$ formé de :

- un réseau R et
- une application (fonction totale) $\mu : P \rightarrow \mathbb{N}$.
  $\mu(p)$ est le marquage de la place p,
  on dit aussi le nombre de marques contenues dans p.

Jeton : indique le marquage de chaque place.

Le **marquage initial est noté** M_O

On peut limiter le marquage des places à 1 ou à un entier k.
Réseau k-borné : $\forall p.\mu(p) \leq k$ propriétés sur le modèle !

Réseaux de Petri : Marquage

Le marquage initial du réseau en exemple est :

$$M_0 = (\mu(c_1), \mu(c_2), \mu(cpt), \mu(R), \mu(s_1), \mu(p_1), \mu(p_2))$$

$$= (1, 0, 0, 0, 0, 1, 0)$$

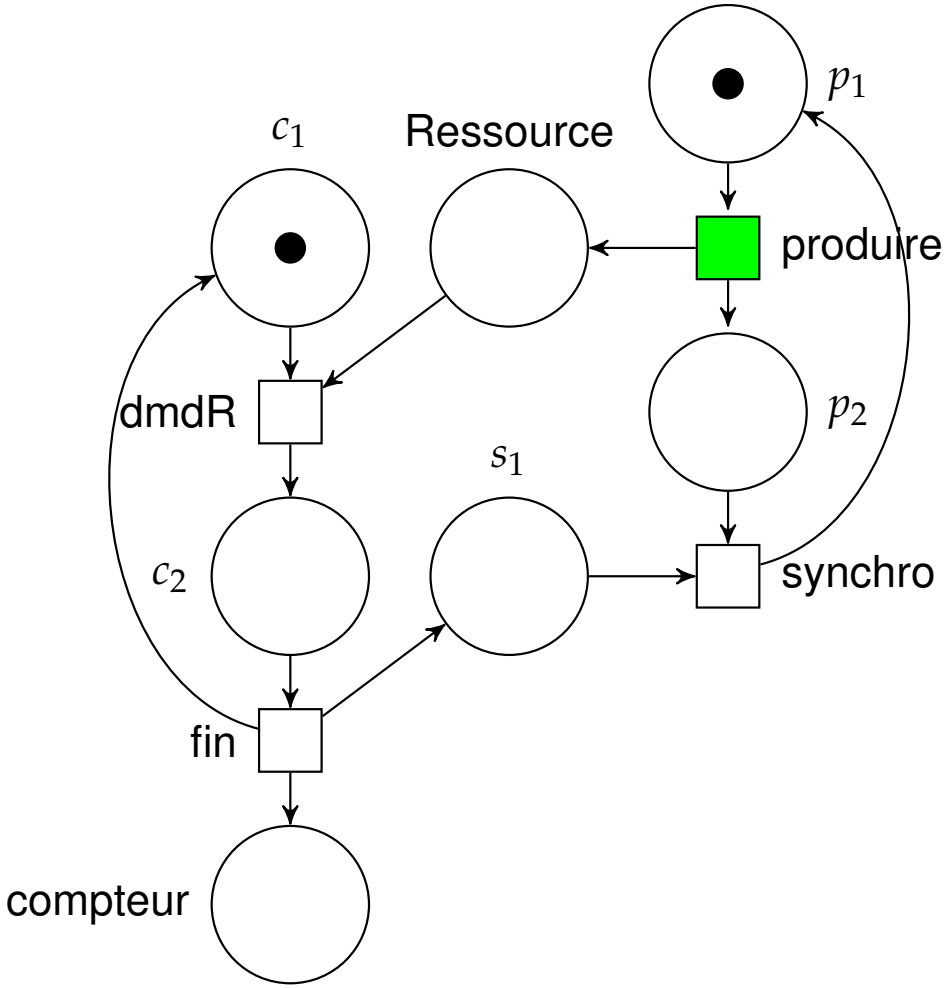
Le marquage un état du réseau, ou un état du système modélisé !

Réseaux de Petri : évolution du réseau

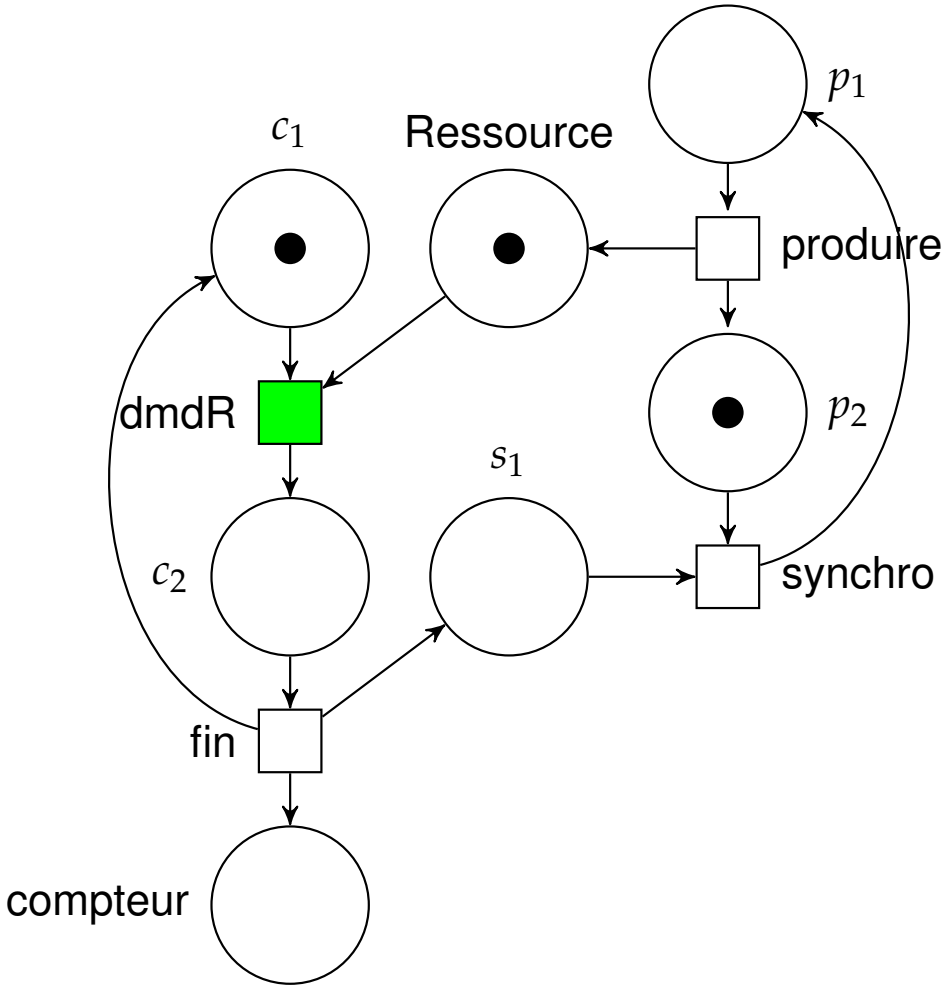
- Franchissement des transitions tirables (*enabled*, *fireable*).
- Non-déterminisme en cas de choix
- Après une transition, modification du marquage

$$= (1,0,0,0,0,1,0) \xrightarrow{produire} (1,0,0,1,0,0,1)$$

Exemples de réseaux de Petri



$(1,0,0,0,0,1,0)$



$(1,0,0,1,0,0,1)$

Réseaux de Petri : évolution du réseau

Franchissements successifs des transitions

$$(1,0,0,0,0,1,0)$$
$$\downarrow produire$$
$$(1,0,0,1,0,0,1)$$
$$\downarrow dmdR$$
$$(0,1,0,0,0,0,1)$$
$$\downarrow fin$$
$$(1,0,1,0,1,0,1)$$
$$\downarrow synchro$$
$$(1,0,1,0,0,1,0)$$
$$\downarrow produire$$
$$(1,0,1,1,0,0,1)$$
$$\ldots$$

Réseaux de Petri : analyse et extensions

Analyse des modèles

- simulation
- vérification
- **outils** : Pipe, CPN, Tina, Romeo, ...

des extensions

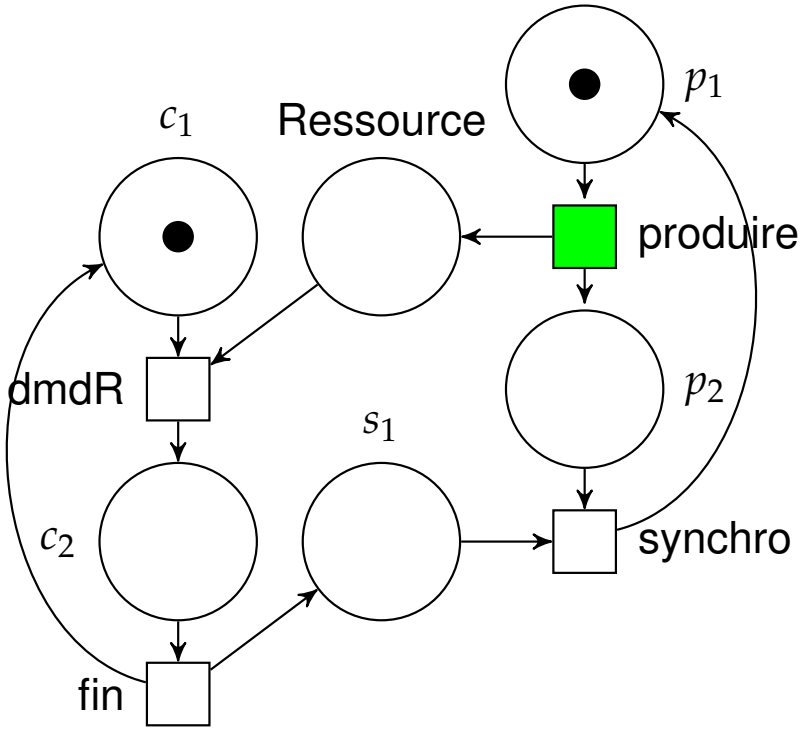
Réseau de Petri :

- colorés,
- à prédicats,
- temporisés,
- probabilistes,
- ...

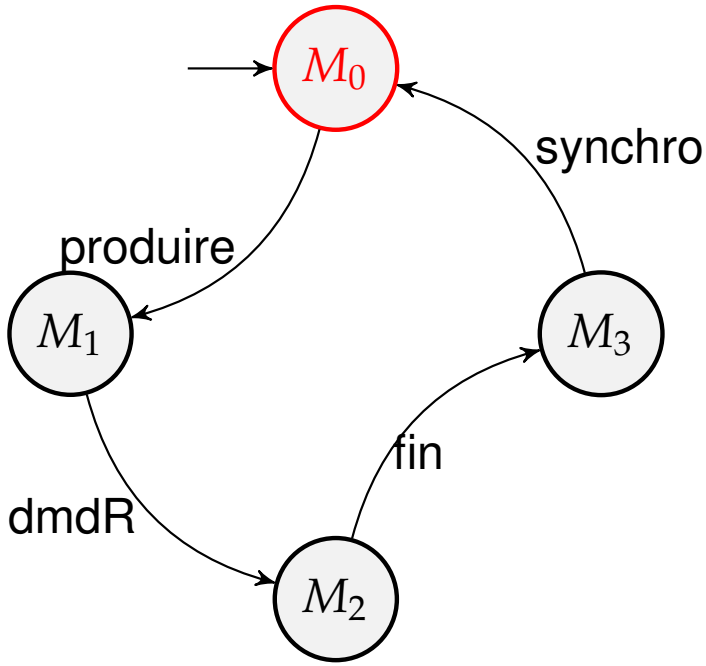
Réseaux de Petri : graphe versus automate

le comportement d'un système décrit par un RdP est un automate dont les états sont les marquages du réseau de Petri.

graphe du réseau



comportement



$M_0 = (1, 0, 0, 0, 0, 1, 0)$

Exemple de Réseaux de Petri

Système hétérogène de pollinisation : Fleurs-Abeille

Figure: Je butine, en pollinisant

Composants : Fleur mâle + Fleur femelle + Abeille

Algèbres de processus

Algèbres de processus
(historiquement : Hoare, Milner)

- Communicating Sequential processes (Hoare)
- Calculus of Communicating Systems (Milner)
- π-calculus (Milner)

De nombreuses dérivées et extensions (dont E-Lotos, PAT)

Modèles à événements discrets

Modèles à événements discrets : Event-B

De

Méthode B pour l'ingénierie logicielle

à

Event-B pour l'ingénierise système

State space

The value of a variable may be changed ⇒ it takes different *states*

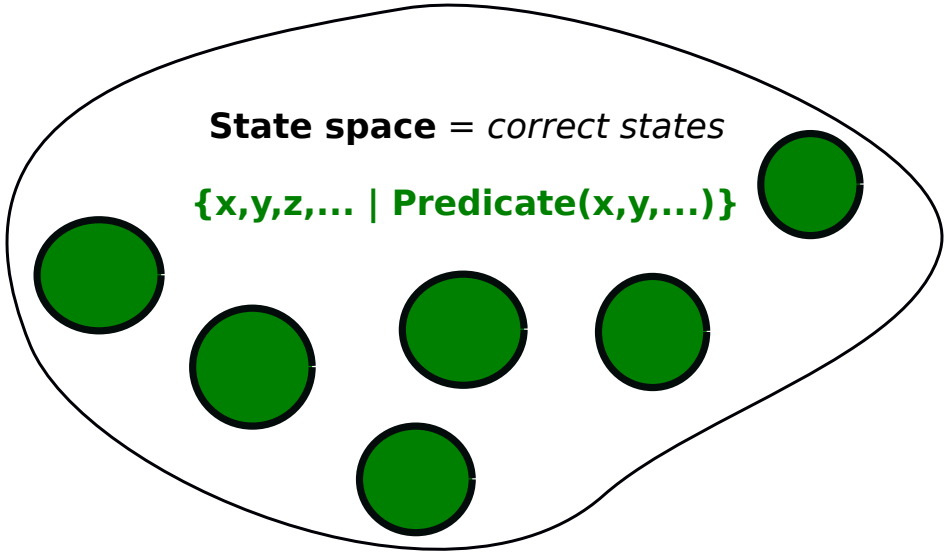


Figure: A view of a state space

A correct software: composition of operations that relate the states.

Development Approach

Formal model (or state) oriented

- Describe a state space
- Describe the operations that explore the space
- Transition system between the states

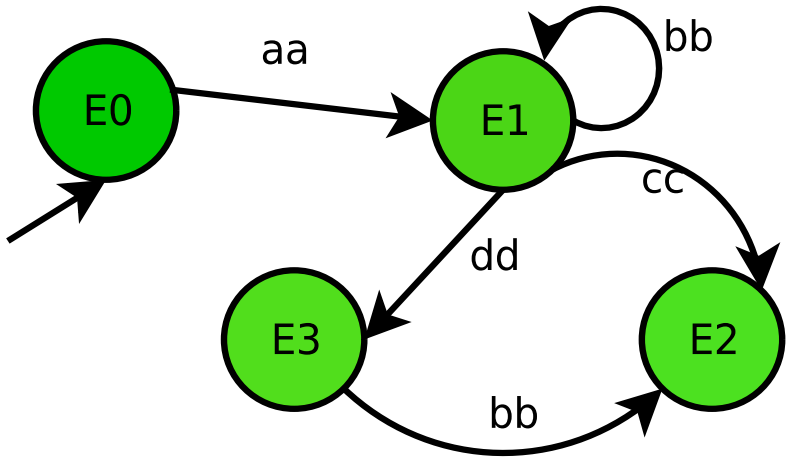


Figure: Evolution of a software system

Modèles à événements discrets : Event-B

Un **modèle Event-B** dénote une simulation mathématique du comportement d'un système asynchrone.
Le modèle est décrit par un espace d'état et un ensemble d'événements qui modélisent les transitions entre états.

Des **abstractions** puis des **raffinements successifs** permettent de construire progressivement et correctement les modèles et les systèmes.

Approche ingénierie système :

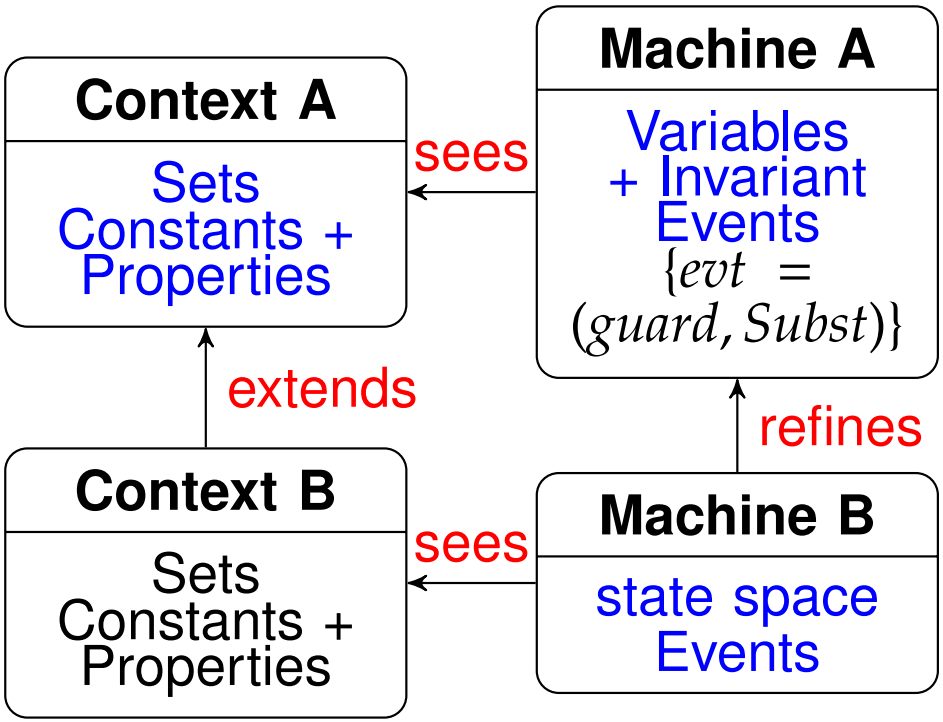
- abstraction : modèle global de tout un avec ses invariants
- raffinement : concrétisation progressive
- décomposition vers des sous-systèmes (de différentes natures ?!)

Modèle Event-B

Espace d'états : décrit par un ensemble de variables, des prédicats qui contraignent ces variables et des propriétés invariantes du système en étude.
Evénement : une substitution gardée.
Lorsque la garde est vraie, la substitution permet la transition d'états en réalisant les changements des valeurs des variables d'états.

Model MySystem
Invariant: Predicate (over state variables)
Events: $\{e_i, e_j, e_k, \cdots\}$
End

Event B model




```
MACHINE MySys
SEES MyCtx1, MyCtx2, ...
REFINES AbsModel2
VARIABLES
sv1, sv2, sv3, ...
INVARIANTS
sv1 : MyTYPE -> NAT
sv2 : ...
/* properties */
...
EVENTS
...
END
```

Exemple : Event B model

Interaction entre un composant producteur de ressources et un composant consommateur.

Example : producer / consumer

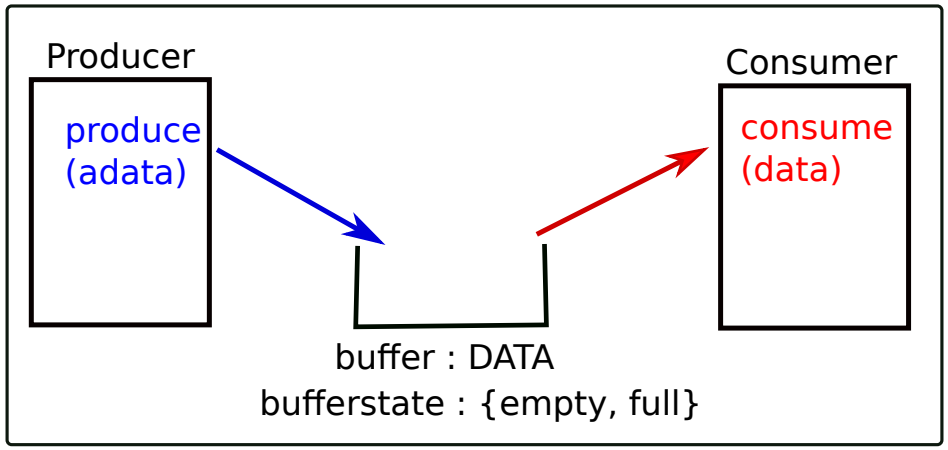


Figure: An overview of a producer-consumer

- Concurrent running of a consumer which retrieves a data from a buffer filled by a producer.
- The consumer cannot retrieve an empty buffer; the producer cannot fill in a buffer already full.

An event-driven model of the system is as follows:

Example : producer/consumer

Machine *ProdCons* /* the abstract model */
sets DATA ; STATE = {empty, full}
variables buffer, bufferstate, bufferc
invariants
bufferstate ∈ STATE ∧ buffer ∈ DATA ∧ bufferc ∈ DATA
initialization
bufferstate := empty || buffer :∈ DATA || bufferc :∈ DATA
events
produce /* if buffer empty */
any dd **where** dd ∈ DATA ∧ bufferstate = empty
then buffer := dd || bufferstate := full
end ;
consume /* if buffer is full */
select bufferstate = full
then bufferc := buffer || bufferstate := empty
end
end

Figure: A Producer-Consumer Abstract System

Example : producer/consumer

An event-B system can be decomposed and refined separately.

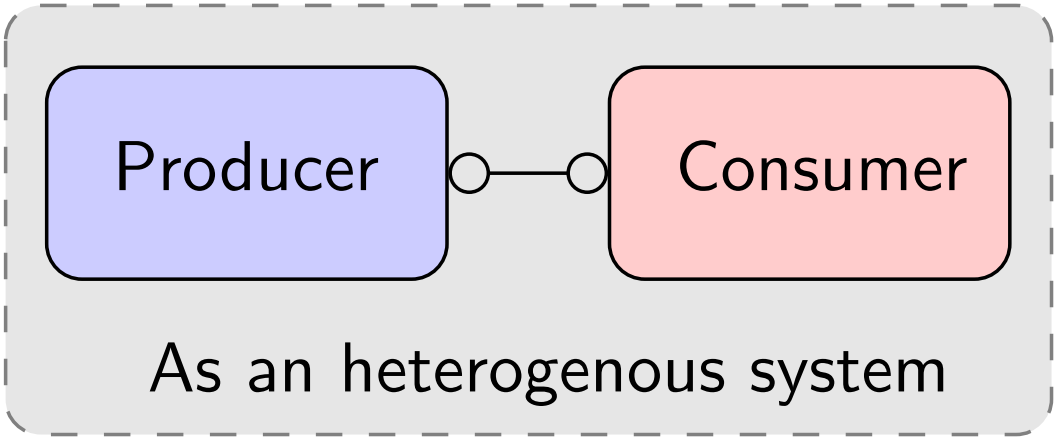


Figure: The producer-consumer decomposed into components

Conclusion

De nombreux modèles existent pour décrire les composants/systèmes selon leur caractéristiques

Reste la question des compositions et des interactions : chapitre suivant

Chapitre 3 : **Composition de modèles hétérogènes**

☞ Composer les divers modèles des composants du système