Génie Logiciel
modélisation et vérification comportementales de systèmes

Tarek Melliti - tarek.melliti@ibisc.fr
Pascal Poizat - pascal.poizat@lri.fr

version du 28 février 2012

À quoi çà sert ?

LA question éternelle !

on pourrait répondre :

- c'est dans votre cursus
- il y a un examen en fin d'année

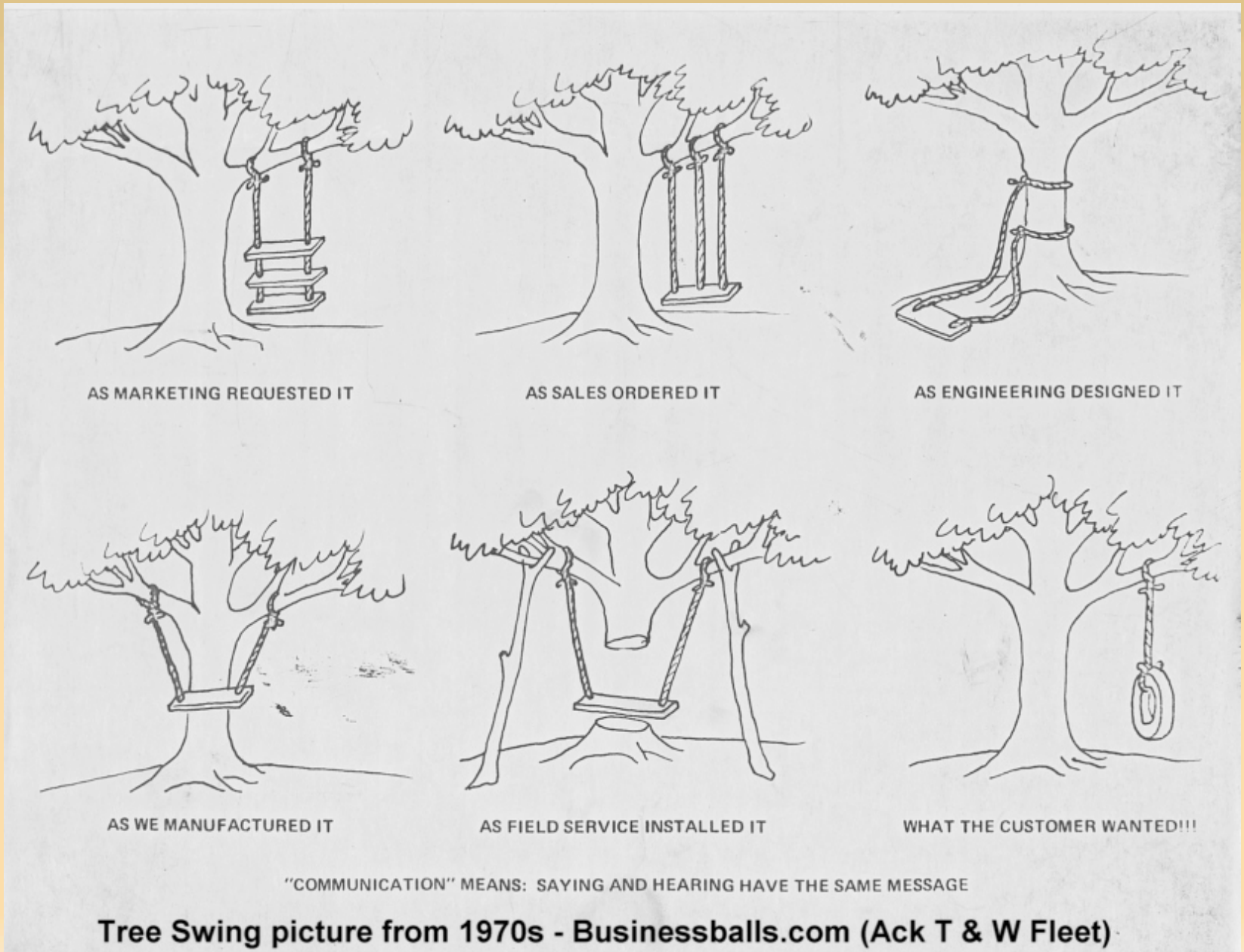
... mais il y a des arguments plus intéressants

À quoi çà sert ?

- L'ingénierie désigne l'ensemble des fonctions allant de la conception et des études, y compris la formalisation des besoins des utilisateurs, à la responsabilité de la construction et au contrôle des équipements d'une installation technique ou industrielle. Il est aussi souvent utilisé dans un sens étendu à d'autres domaines
- Le terme « ingénierie » est un terme introduit de manière récente dans la langue française où il se substitue parfois au terme « génie » désignant l'art de l'ingénieur.
- Le génie logiciel (en anglais : *software engineering*) désigne l'ensemble des méthodes, des techniques et outils concourant à la production d'un logiciel, au-delà de la seule activité de programmation.

source : wikipedia, janvier 2009

À quoi çà sert ?



Tree Swing picture from 1970s - Businessballs.com (Ack T & W Fleet)

À quoi çà sert ?

- abandon logiciels : 2/6 (de grande taille)
- retard >50% durée prévue livraison logiciels : 75%
- coût bugs & erreurs des logiciels
  59.5 Mds $ / an, économie EU (estimation '02)
- (quelques) catastrophes :
  - Therac 25, '85-'87 : 6 patients irradiés, 2 morts
  - Syst. Bagages Aéroport Denver, '95 : 16mois, 3.2 Mds$
  - Ariane 5 - vol 88/501, '96: 40s de vol, destr., 850 M$
  - Mars Climate Orbiter & Mars Polar Lander, '99 : destr.

À quoi çà sert ?

Pour conclure :

- pour un pont il y a :
  1. celui qui le conçoit
  2. celui qui le fabrique
  3. celui qui le vend (parfois avant 1.)

À quoi çà sert ?

Pour conclure :

- pour un pont il y a :
  1. celui qui le conçoit
  2. celui qui le fabrique
  3. celui qui le vend (parfois avant 1.)
- pour un logiciel il y a :
  1. celui qui le conçoit
  2. celui qui le fabrique
  3. celui qui le vend (parfois avant 1.)

Prologue

Les systèmes informatiques aujourd'hui:

- de plus en plus importants
  - omniprésence
  - informatique diffuse
- de plus en plus complexes
  - banalisation des réseaux, répartition/distribution
  - préoccupations multiples (données, interaction, ...)
- exigences
  - plus vite, mieux, moins cher

Prologue

Que peut-faire l'ingénieur logiciel ?

→ méthodes de conception

- systémique vs. analytique
- ascendante vs. descendante
  - composants → système désiré
  - système désiré → sous-systèmes nécessaires
- méthodes semi-formelles (Merise, UML, ...) vs. méthodes formelles

Système

- Système =
  ensemble de sous-systèmes (ou composants) agencés d'une certaine manière
- $S = f(A[S_i], I)$
  - A[ ] : agencement
    a après b, a ou b, a concurremment à b, ...
  - I : interaction
    - mémoire partagée, événements, messages, ...
    - émetteur/receveur, multicast, groupes, ...
    - mode synchrone, mode asynchrone
    - signal pur ou information transmise

Processus

- un système est donc décomposable en sous-systèmes plus simples

→ c'est quoi un sous-système trivial/de base ?

- programme, logiciel, application, ...
- généralement : unité de donnée/de traitement
- pour nous : ensemble structuré d'instructions opérant sur des données

→ un processus

Les 2 dimensions d'un système

- Aspect statique : les données
  - notion de type (simple / structuré)
  - expressions, évaluation des expressions
- Aspect dynamique : le comportement
  - actions sur les données (ens. fini)
  - actions d'interaction (ens. fini)
  - structuration
  - parfois dimension temporisée

Exemple de système (simple)

un chronomètre

- un compteur
  - données : durée + valeur finale (ici 0)
  - comportement :
    - actions décrémenter et beep
    - chaque seconde qui passe décrémente le compteur jusqu'au beep à zéro
- une horloge
  - données : aucune
  - comportement : un tic à chaque seconde

Exemple de système (connu)

le diner des philosophes

- deux types de sous-systèmes
  philosophes et fourchettes
  agencement : table

- philosophes
  - données : état
  - actions :
    avoir faim
    prendre / rendre f. gauche
    prendre / rendre f. droite
    réfléchir
  - différents comportements

- four
  - données : état
  - actions :
    être prise / rendue
  - comportement :
    prise rendue prise ...

Exemple de système (réaliste)

pont à une voie reliant deux routes à double voie

- trois types de sous-systèmes
  voitures g → d, voitures d → g, le pont
  agencement : [...]

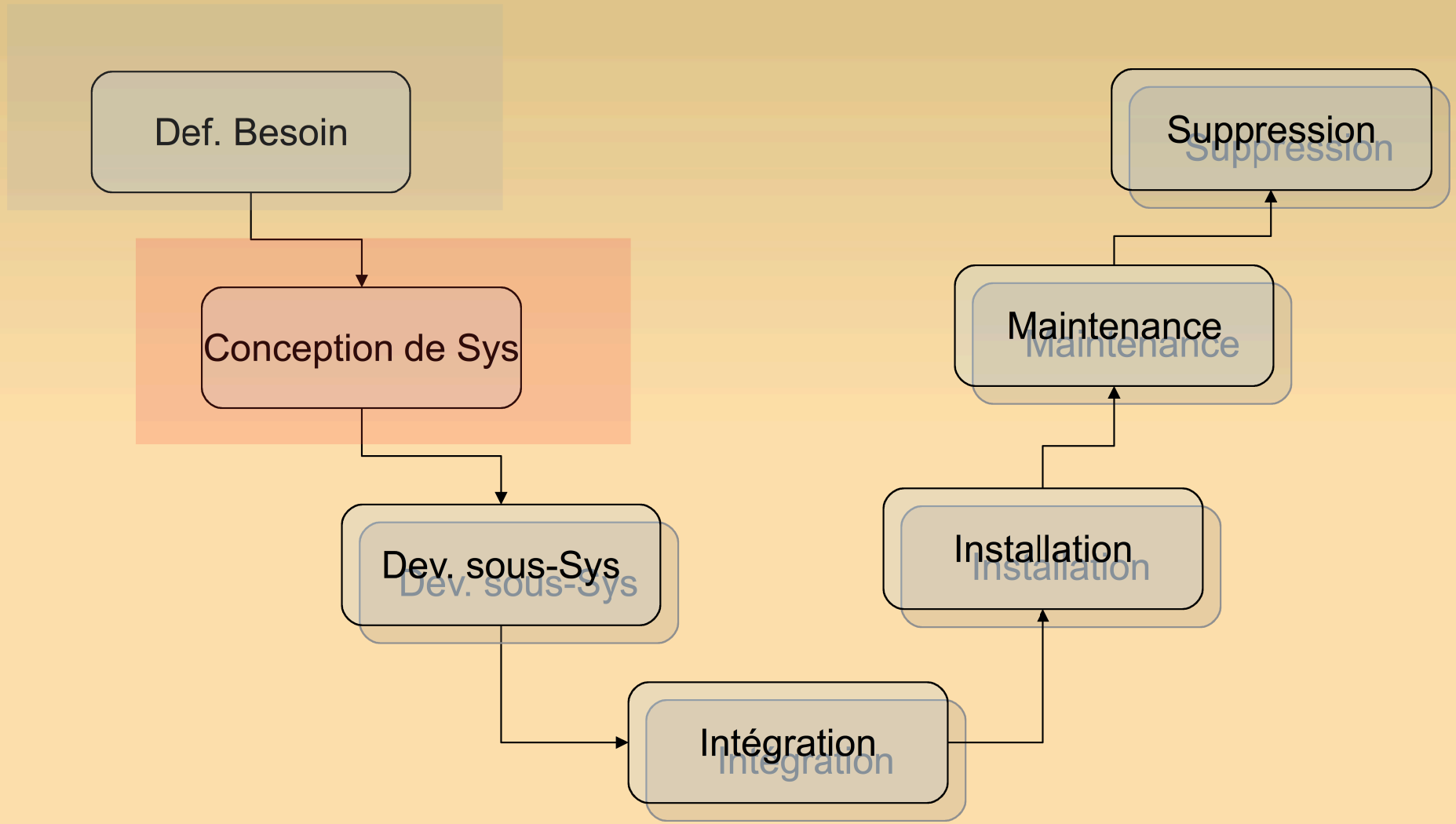
- voitures
  - données : état
    attente/sur pont/sortie
  - actions :
    entrer pont
    sortie pont
  - comportement : [...]
- pont

  - données : état
    libre/occupé(*)
  - actions :
    entrée voiture(*)
    sortie voiture(*)
  - comportement : [...]

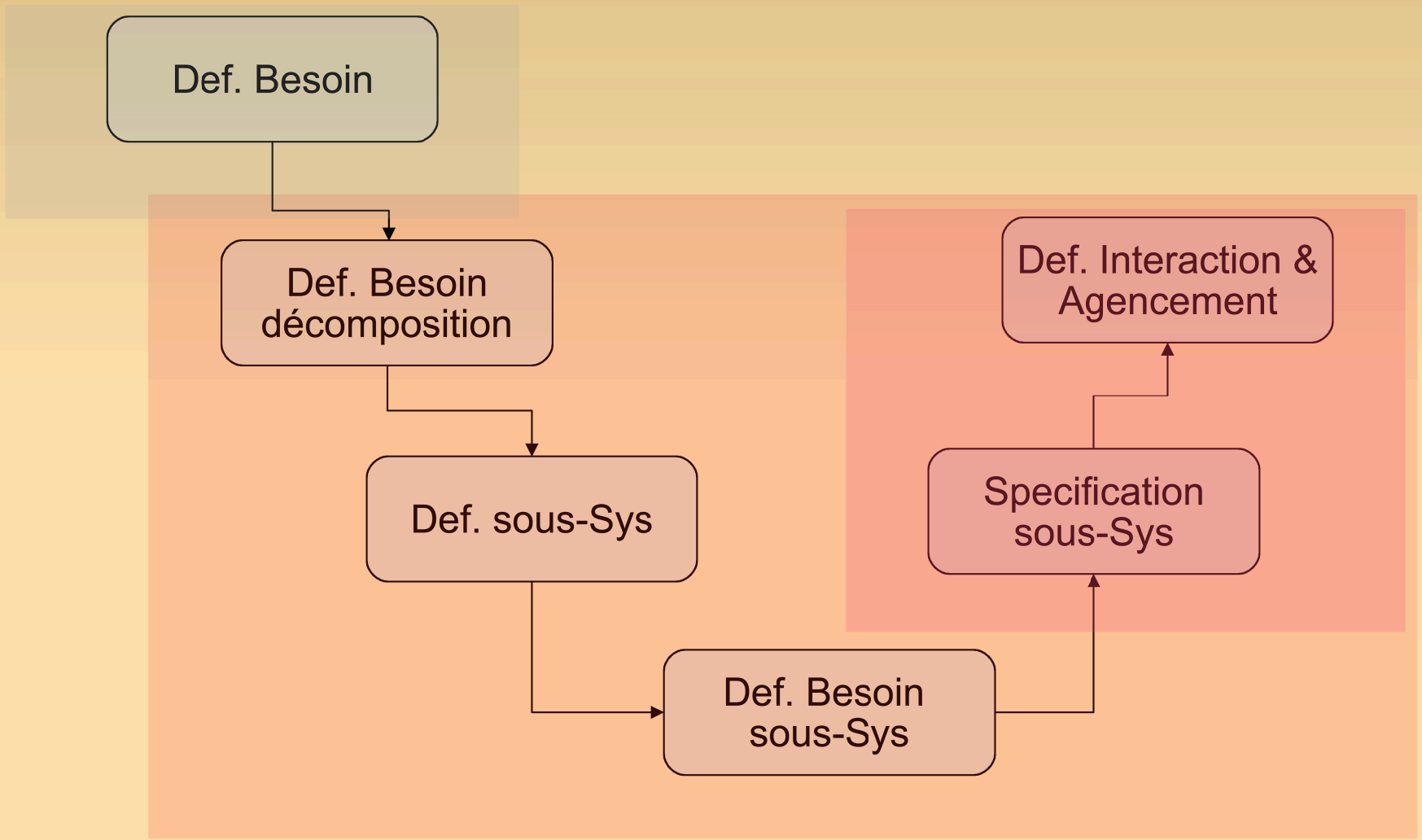
Vers une démarche d'Ingé. Logicielle

- supposez que vous soyiez face à ce(s) problèmes
- si vous êtes (un peu) rigoureux ...
  → méthode de conception qque (UML, ...)
  → démarche/gestion de projet (RUP, ...)
  ... mais cela ne suffit pas !
- avant de passer à (plus) de rigueur, quelques rappels

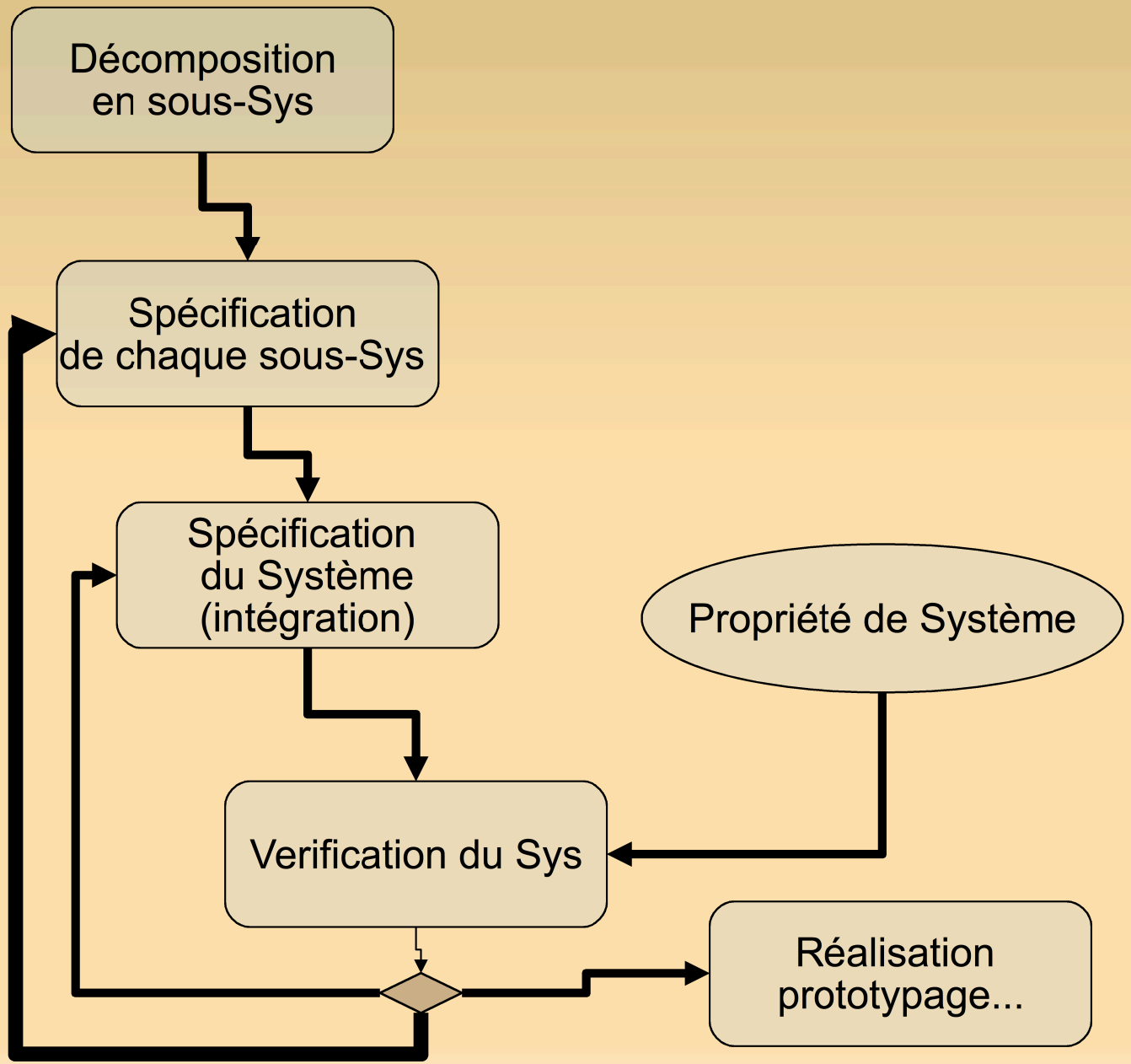
Cycle en V


Def. Besoin
Conception de Sys
Dev. sous-Sys
Intégration
Installation
Maintenance
Suppression

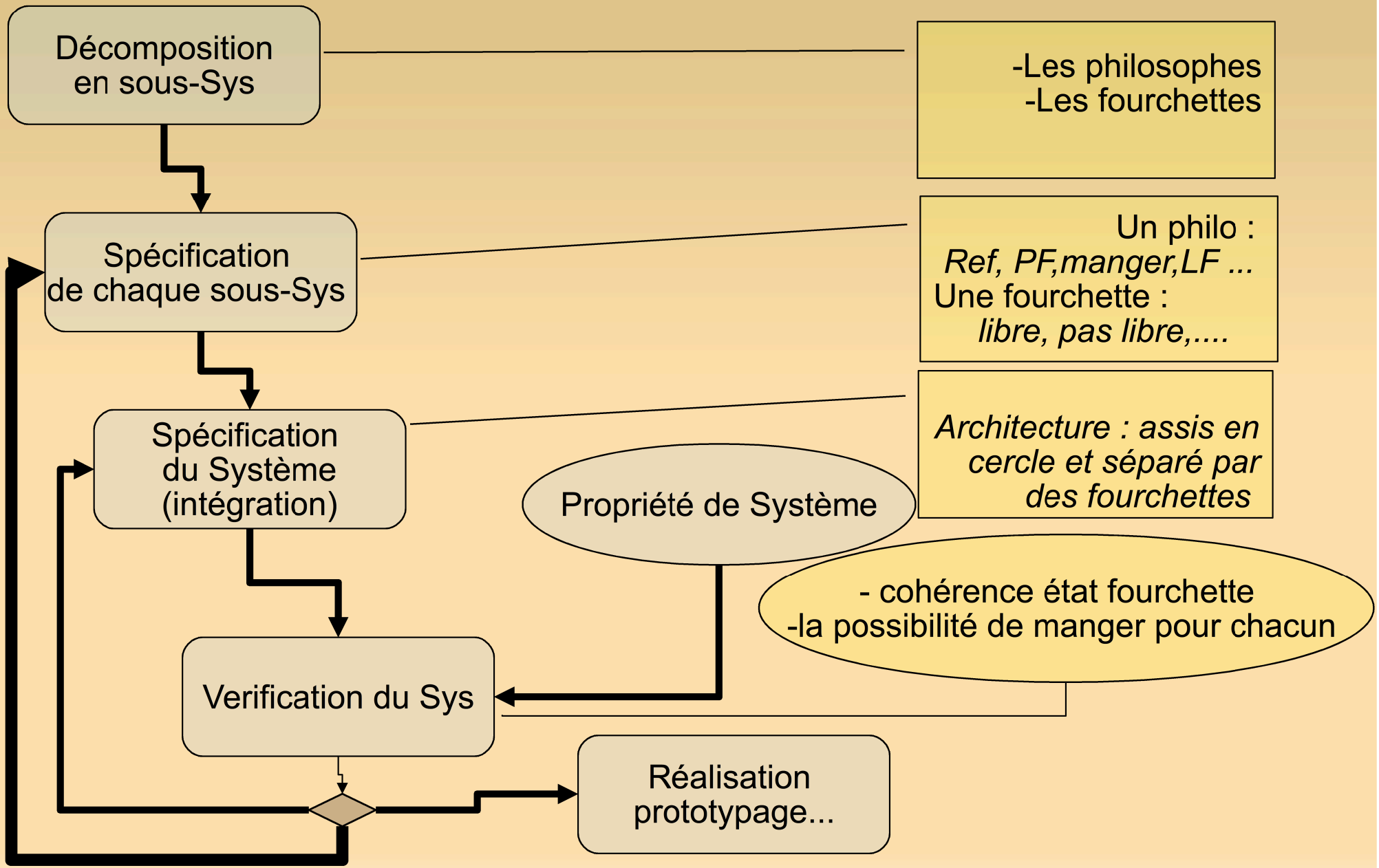
Cycle en V / conception



Cycle en V / à retenir ici



Cycle en V / à retenir ici



Spécification : comment ?

- [tentative] une première forme de spécification (que vous connaissez) serait de programmer
- avantages :
  - possibilité d'utiliser un langage de haut niveau
  - correspond à la finalité
- inconvénients (pour syst. complexes/réalistes) :
  - la simulation n'est pas suffisante
  - vérification difficile (voire impossible)
  - coûteux
    (temps, risque détection erreur plus tardive)

Spécification : vers le formel

besoin d'un processus de spécification formelle

- spécification ?
  - représenter/modéliser
    le système et les propriétés désirées
- formelle ?
  - clair, non ambigu, ”mathématique”
- pourquoi ?
  - vérification formelle des propriétés p/r au système
  - automatisation, outils

De multiples modèles formels

- état-transition, graphes bipartites
  - réseaux de Petri (L3, facultatif)
  - automates communicants (partie 1 de ce cours)
- processus
  - nombreuses algèbres de processus (CCS, CSP, FSP, LOTOS, mCRL2, …) vous en saurez plus en M2R !
  - mCRL2 comme ”langage” (partie 2 de ce cours)
- propriétés logiques (partie 3 de ce cours)
- formalisation d'un langage semi-formel
  - généraliste : UML, dédié : SDL, ...

Méthodes formelles en pratique

- méthode formelle =
  notation/modèle formel
  + sémantique bien définie
  + outillage et/ou outillage
- indispensable pour
  les systèmes critiques, la certification, ...
- (quelques) succès et industriels concernés :
  - Orly Val, Métro ligne 14 [méthode B]
  - U. Oxford et IBM CICS [Z] / qualité + red. coût 9%
  - NASA, Airbus, IBM, Microsoft, FT R&D, ...

Propriétés

- Sûreté
  quelque chose de mal n'arrive jamais
  - absence d'interblocage (*deadlock*)
  - concurrence d'accès / erreur de section critique
  - *philosophes non bloqués*
  - *pas deux voitures sur le pont en même temps*
- Vivacité
  quelque chose de bien finit par arriver
  - absence de famine, équité
  - *qque soit sa couleur, une voiture finit par passer*
  - *toute requête reçue finit par être satisfaite*

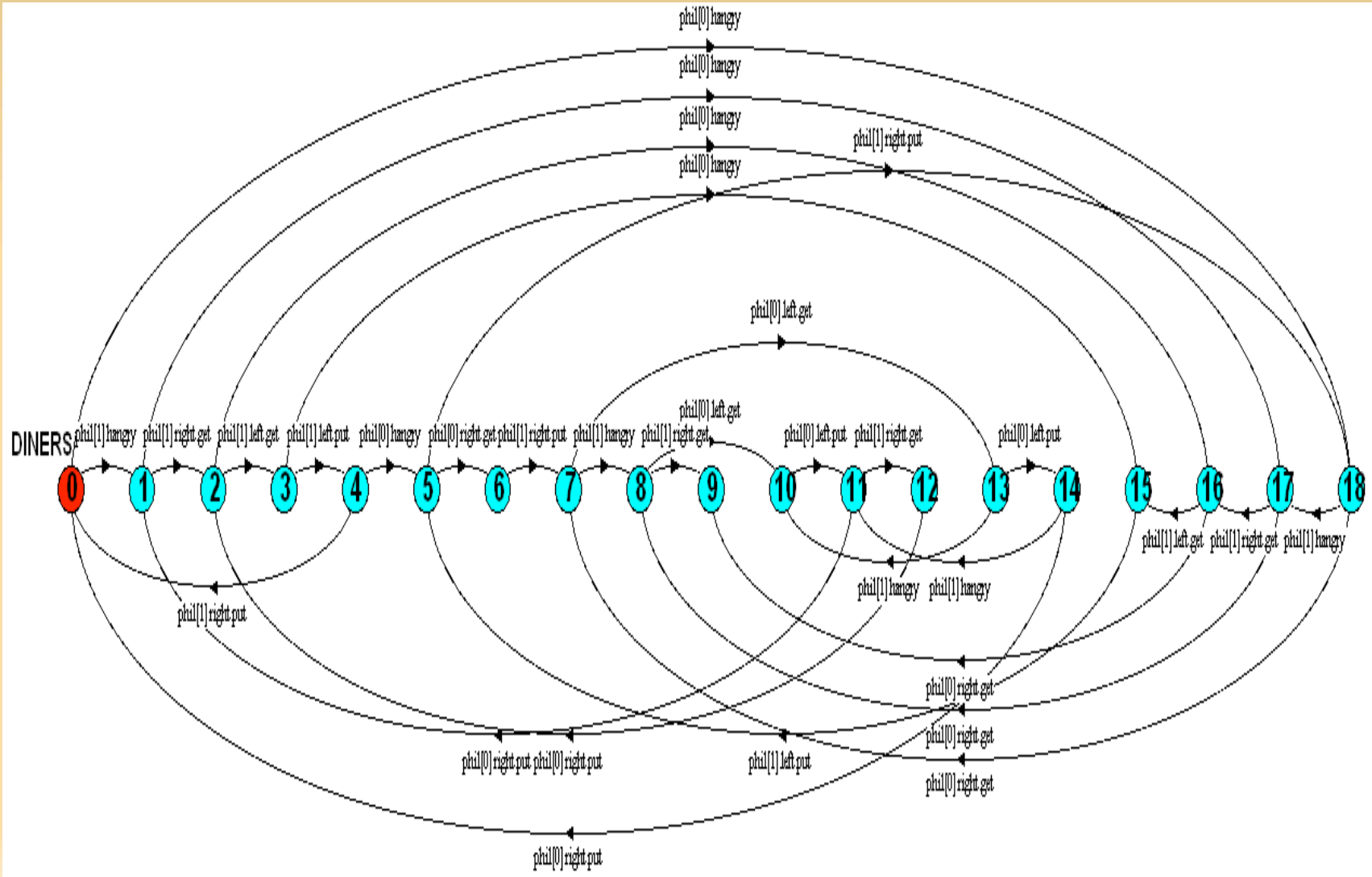
Pourquoi le besoin de formel ?

On pourraît imaginer se passer de méthodes formelles ...

- sur du code :
  - plusieurs choses sont mixées (intérêt ou non ?)
  - simulation/test à la main (non exhaustif sans formel)
  - (très) compliqué si distribution, interaction, etc.
- avec ou sur un modèle formel :
  - on peut tester la conformité mais aussi prouver
  - effort actuel de formalisation des langages (preuve mécanisée, vérif. de modèle)

Besoin d'automatisation

Philosophes, n=2 [19 états], n=5 [1743 états]



(Des) sources du problème

Prog. séquentiel vs. programme concurrent

- retour sur S = f(A[S_i],I)
- exemple : réservation d'un voyage
  - étapes en parallèle : billet avion, hotel, taxi
  - pourquoi ? : efficacité, réduction coût final, ...

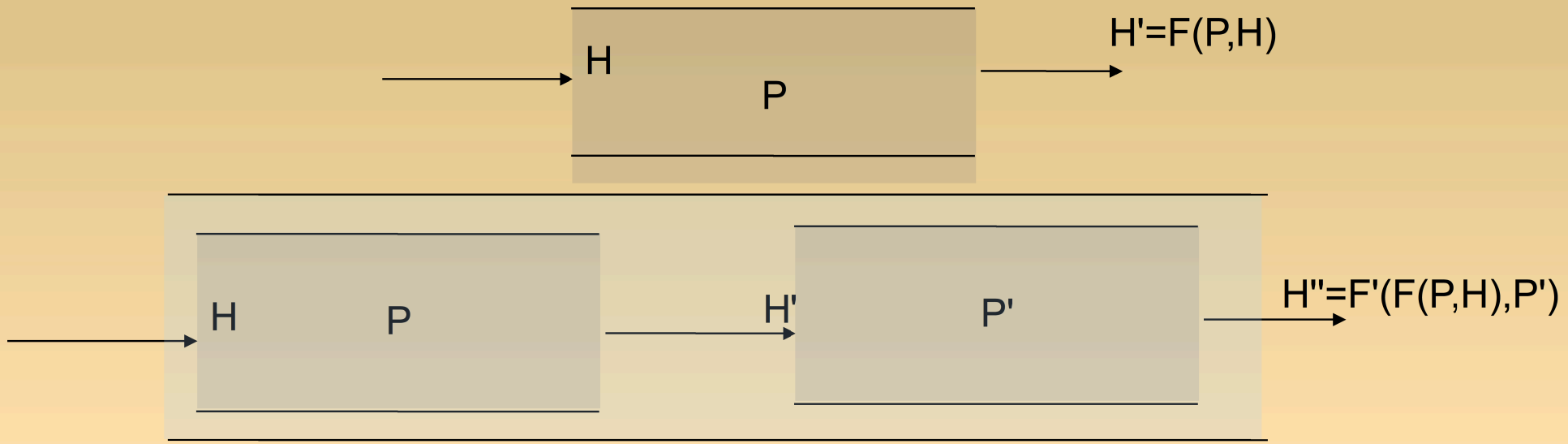
Prog. manipulant des données (état / interaction)

- payer → un événement possible
- payer(piece) → 8 év. possibles
  (2€,1€,50c,20c,10c,5c,2c,1c)

Processus séquentiel

- une configuration de données (H)
  + un comportement (P)
- P déterministe (mais + tard : abstraction)
- P = ens. de suite d'actions, chacune représentant une exécution du processus
- les actions de P agissent sur H
- H n'est modifié que par des actions de P

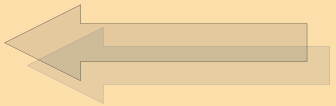
Processus séquentiel



- paire d'observables (H,H') ~ entrées / sorties
- P1 equiv. P2 si mêmes paires d'observables
  - si on insère P2 à la place de P1 dans P3, alors les observables de P3 ne changent pas
- les processus séquentiels peuvent être vus comme atomiques

Processus concurrent

- différents cas !
  - une horloge / un processeur
    → pseudo-parallélisme
  - une horloge / n processeurs
    → parallélisme synchrone
  - n horloges / n processeurs
    → parallélisme asynchrone

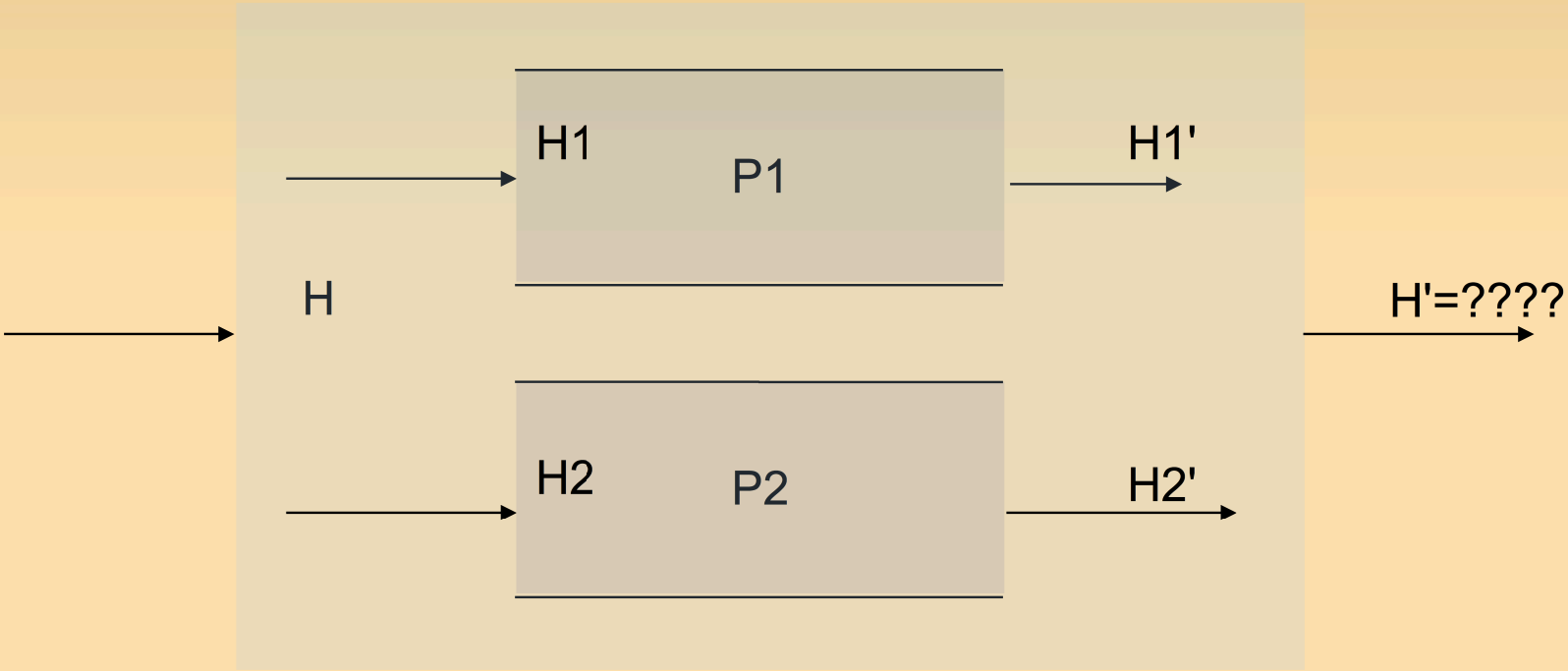


de + en + le cas :

- nv architectures
  - réseaux
- (Web) services

- on parle aussi de modèle de concurrence synchrone/asynchrone
- concurrence : autonomie des processus et indépendance du contrôle entre eux

Processus concurrent



H'=? ... ça depend ...

- si P1 et P2 indépendants, alors H'= H1 ∪ H2'
  - H1 ∩ H2 = ∅
  - aucune interaction/communication entre P1 et P2
- si P et P' interagissent alors ... ça dépend !
  - H', H1' et H2' sont fonction de H1, H2, P et P'
  - on va voir dans la suite comment obtenir
    le comportement d'un processus concurrent
    à partir de ceux des sous-processus le composant

Le point et les objectifs

- nécessité du génie logiciel
  et de modèles formels
- notions de système et sous-systèmes,
  notions de processus et d'interactions
  processus séquentiels et/ou concurrents
- expression et vérification de propriétés
- ici :
  - processus
    → principalement leur comportement
  - modèles finis à événements discrets

Plan du cours

- Partie 1 – modèle comportemental de système
  - Système de Transitions Etiquetées (STE / *LTS*)
  - sémantique STE de la concurrence synch./asynch.
  - STE communicants
  - autour de la comparaison de STE
- Partie 2 – ”langages” de processus
- Partie 3 – propriétés

Plan du cours

- Partie 1 – modèle comportemental de système
- Partie 2 – ”langages” de processus
  - limites des modèles états-transitions
  - présentation des algèbres de processus
  - pratiquer avec les outils **mCRL2**
- Partie 3 – propriétés

Plan du cours

- Partie 1 – modèle comportemental de système
- Partie 2 – ”langages” de processus
- Partie 3 – propriétés
  - description logique des propriétés
  - pratiquer avec les outils **mCRL2**

Partie 1 – Modèle comportemental les bases

- Partie 1 – modèle comportemental de système
  - Système de Transitions Etiquetées (STE/ LTS)
  - sémantique STE de la concurrence synch./asynch.
  - STE communicants
  - autour de la comparaison de STE
- Partie 2 – ”langages” de processus
- Partie 3 – propriétés

Principe

- le comportement est une suite d'états stables dans lequel peut se trouver le système
- l 'évolution (changement ou non d'état) est régie par des événements représentant :
  - des actions
  - des interactions
- on distingue :
  état(s) initial(aux), état(s) final(aux)

Dualité

il existe une dualité :

- déf. système comme ensemble d'évolutions possibles de ses états (abstraction actions)
- déf. système comme ensemble de suite d'actions possibles (abstraction état)
- exemple du pont
  - éviter que deux voitures soient sur-pont (états)
  - éviter deux actions entrer-pont de suite (actions)

Automates

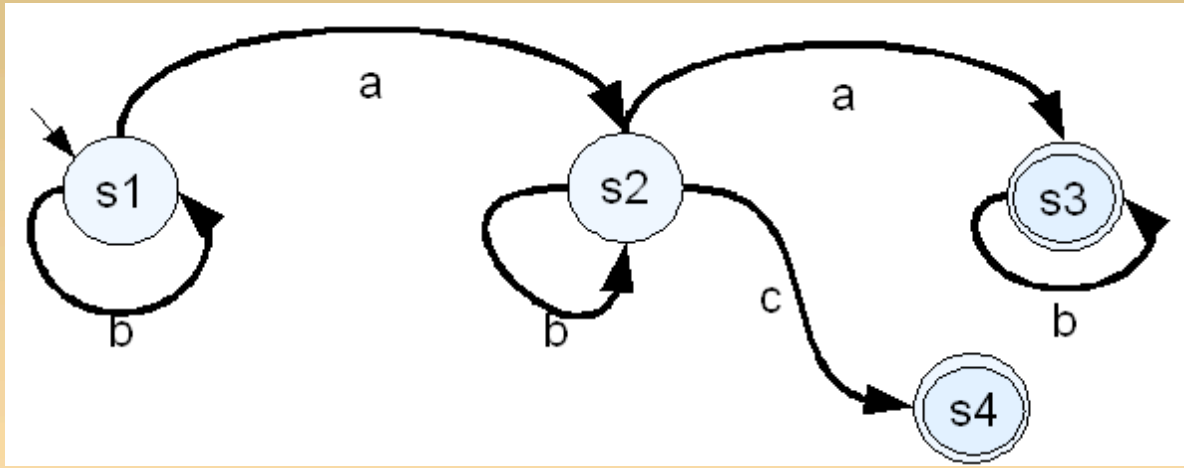
- modèle bien connu
  - fondements théoriques
  - étudiés en compilation, théorie des langages, ...
- modèle graphique / textuel
- permettent de représenter les systèmes
  il suffit d'avoir un point de vue comportemental

Rappel

Un automate est un quintuplet <A,S,T,I,F>

- A: alphabet (ensemble fini symboles du langage)
- S : ensemble d'états
- $T \subseteq S \times A \times S$: relation de transition
  on trouve aussi $T : S \times A \rightarrow \wp(S)$
- $I \subseteq S$: ensemble des états initiaux (non vide)
- $F \subseteq S$: ensemble des état finaux

Exemple



A={a,b,c}

S={s_1,s_2,s_3,s_4}

T={(s_1,b,s_1), (s_1,a,s_2), (s_2,b,s_2), (s_2,a,s_3),
(s_2,c,s_4), (s_3,b,s_3)}

I={s_1}

F={s_3, s_4}

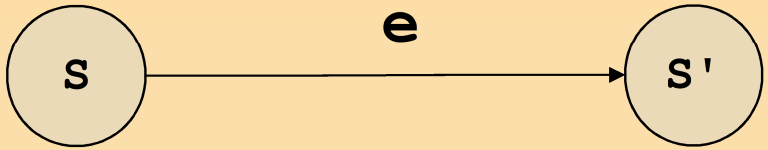
Propriétés

- finitude : S ensemble fini
- déterminisme : $T : S \times A \rightarrow S$ et $|I|=1$
- équivalence :
  - un automate définit un langage (accepté/reconnu)
  - deux automates sont équivalents s'ils acceptent les même langages
  - de même on peut définir l'inclusion

Sémantique

un automate n'est pas qu'un dessin !

- on peut l'interpréter en termes d'évolution d'un état en fonction d'événements (dans T)



- si le système est en s, et que e survient alors il passe en s' / on parle de transition
  - Hyp. : les transitions sont instantanées et atomiques
  - s'il s'agit d'actions (et non d'interactions), alors on peut représenter le début et/ou la fin par un événement

Automates et modélisation de syst.

- généralement un état initial et n finaux
- parfois pas d'états finaux (cas de syst. réactifs)
- généralement déterministes (mais abstraction)
- problématiques différentes: composition, (bi-)simulation, propriétés temporelles, ...

on les retrouve donc sous le nom de Systèmes de Transitions Etiquetées (STE / *LTS*)

STE

Système de transition étiqueté sur un alphabet A

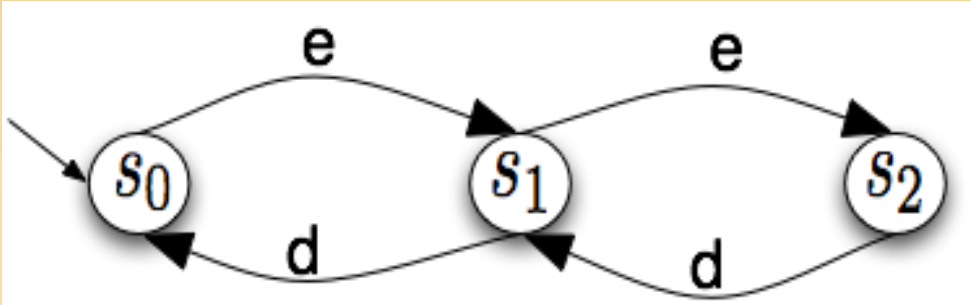
(S,T,A,s_0,F)

- S : états,
- T: $S \times A \times S$ les transitions étiquetés par A
- $s_0 \in S$: état initial
- $F \subseteq S$: états finaux

Exemple

Pile de taille 2

deux actions : empiler (e) et dépiler (d)



$S = \{s_0, s_1, s_2\}$

$T = \{(s_0, e, s_1), (s_1, d, s_0), (s_1, e, s_2), (s_2, d, s_1)\}$

$s_0 \in S$: état initial

$A = \{e, d\}$

Extensions

- plusieurs extensions ont été proposées
- certains concernant les systèmes, d'autres non
- extensions des transitions :
  - gardes, actions, fonctions de sortie
- extensions des états :
  - actions, fonctions d'entrée/de sortie, automates

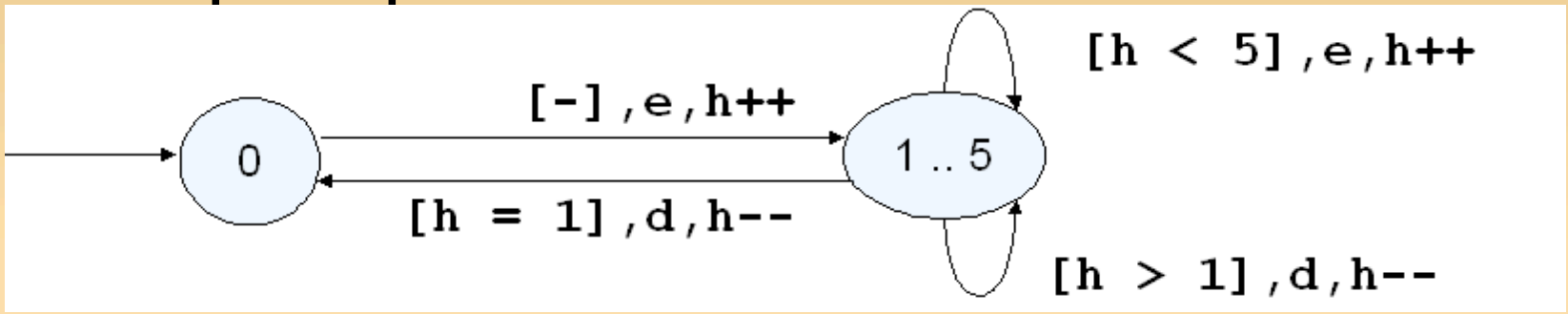
STE gardés

$(A,S,s_0,F,T,V,B,I,v0)$

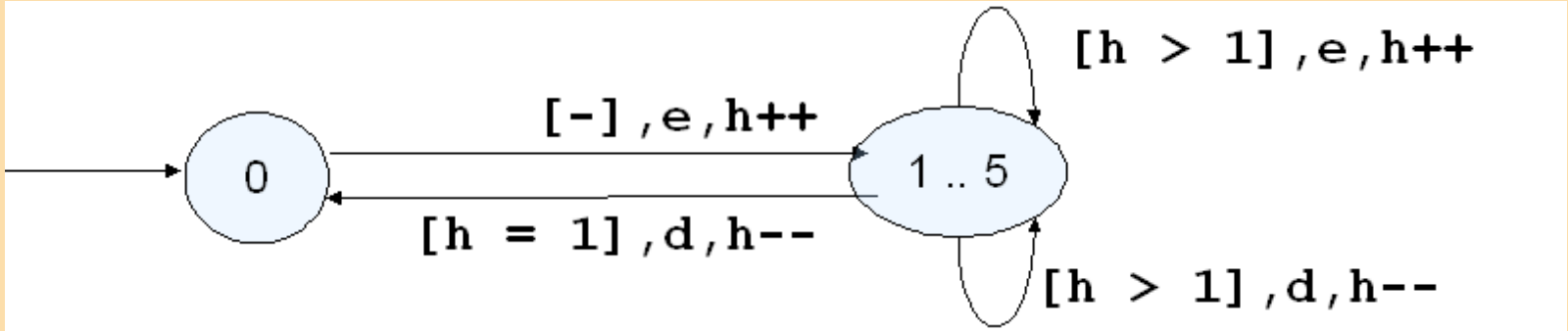
- V : ensemble de variables x_i prenant une valeur dans le domaine D_i
- B : ens. de formules booléennes sur V et D_i
- I : ens. de formules de la forme xi=Exp avec $type(x_i)=D_i=type(Exp)$
- v_0 : ens. de valeurs initiales pour V ($\{(v_i,d_i)\}$)
- $T \subseteq S \times B \times A \times \iota \times S$ avec $\iota \in 2^I$

STE gardés

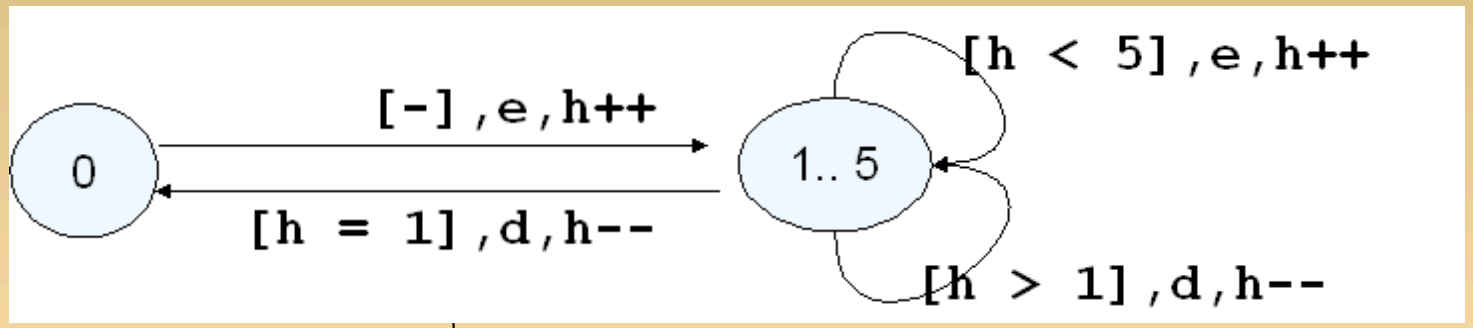
- facilité d'écriture (pouvoir d'expression)
  exemple : pile de taille 5

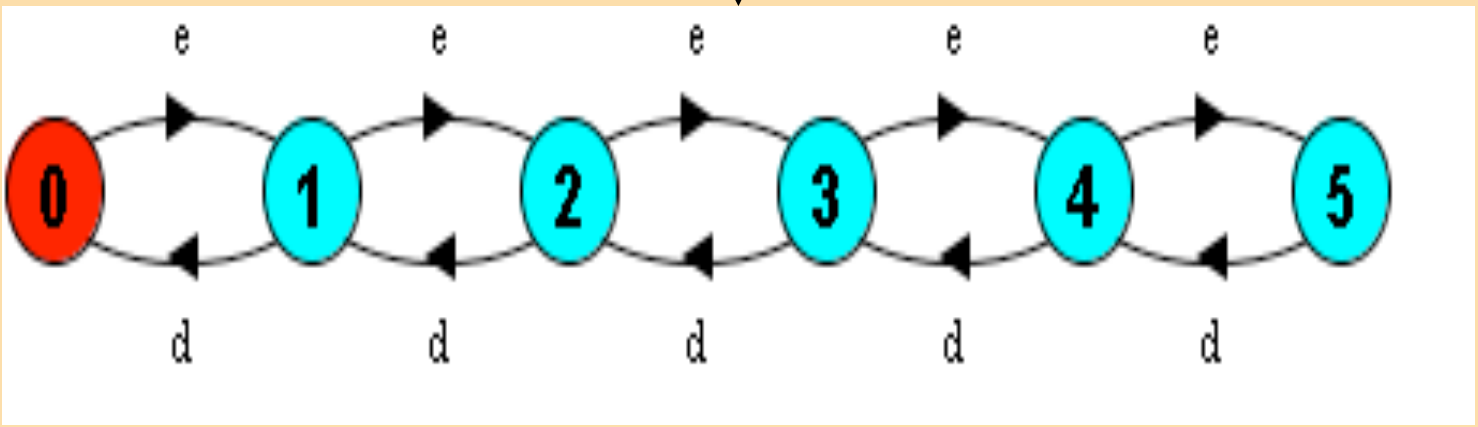


- augmentation de l'expressivité
  exemple : pile de taille infinie



Sémantique STE gardés





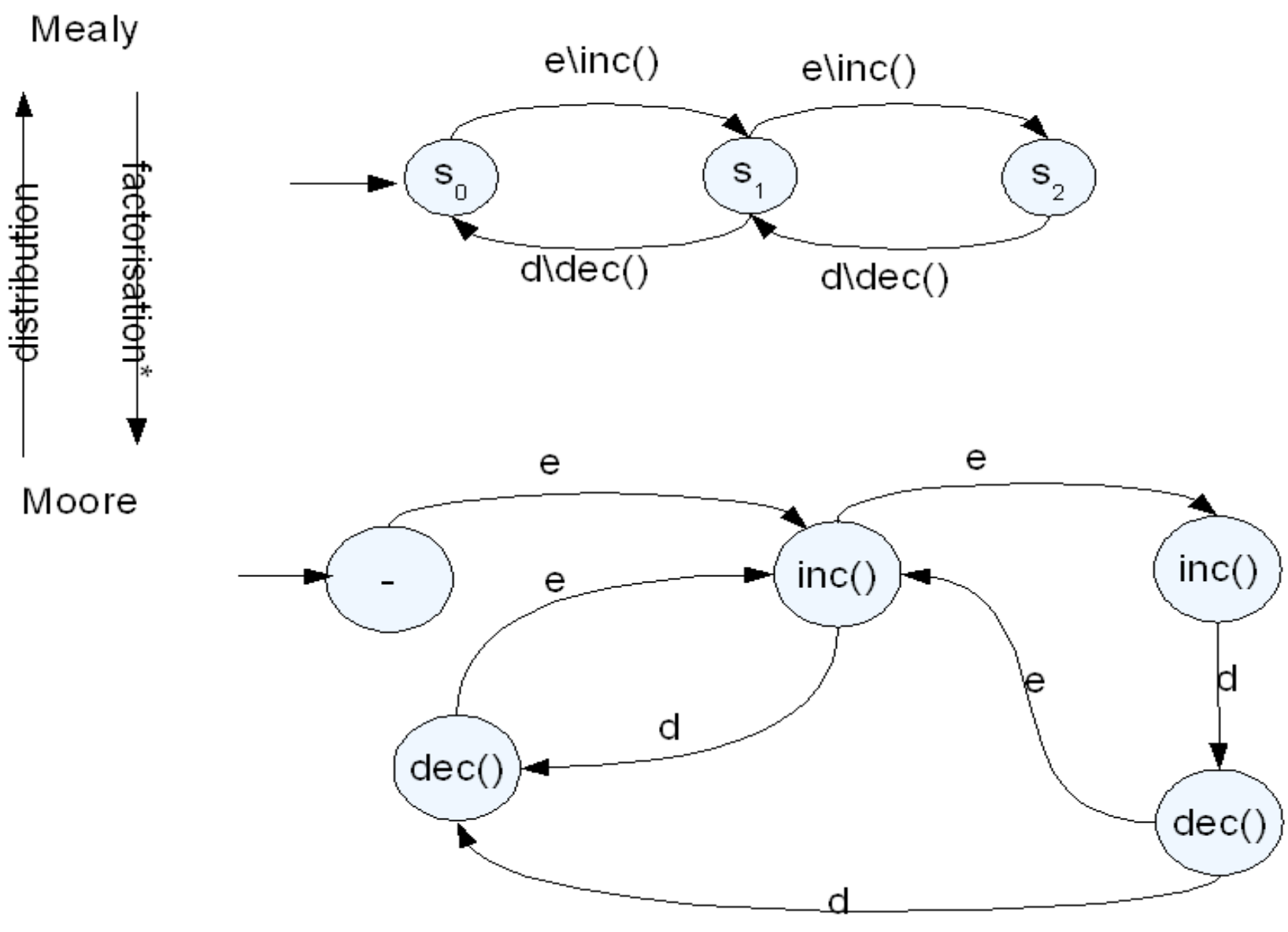
Machines de Mealy/Moore

deux autres extensions : ajout de fonctions

- machine de Mealy:
  - ajout sur les transitions / f. peuvent durer
  - événements non observable / f. observables
  - attention états implicites entre 2 états stables
- machine de Moore (dual):
  - ajout dans les états / f. peuvent durer
  - attention si événement arrive alors que f. s'exécute
- on retrouve tout ça (et +) dans UML

Exemple

pile de taille 2



Retour sur les STE ...

pour pouvoir travailler sur la structuration

on rajoute :

- une notion d'action interne : τ
  - action non observable
  - permet d'abstraire un calcul interne
    ou de représenter un choix interne (indét. interne)
- une action vide : ϵ
  - permet de donner un sémantique à la concurrence
  - $\forall s \in S . (s,\epsilon,s) \in T$

Partie 1 – Modèle comportemental la structuration

- Partie 1 – modèle comportemental de système
  - Système de Transitions Etiquetées (STE / *LTS*)
  - sémantique STE de la concurrence synch./asynch.
  - STE communicants
  - autour de la comparaison de STE
- Partie 2 – ”langages” de processus
- Partie 3 – propriétés

Le point et les objectifs

- le comportement d'un système simple (séquentiel) peut être modélisé par un STE
- notions d'états stables, d'événements (actions,interactions) et de transitions
- différentes extensions permettent de gagner en expressivité
- comment donner une sémantique à un système composé de sous-systèmes (modèle global) ? c-à-d, comment connaître le STE d'un système $S = f(A[S_i],I)$?

Passage à l'échelle

- démarche ascendante
  composer pour réutiliser
  1. définir (ou réutiliser) les sous-systèmes
  2. construire le système
  3. vérifier l'assemblage
- démarche descendante
  (le bon vieux) diviser pour régner
  1. définition gros grain
  2. raffiner
     - pas abordé ici, le raffinement marche aussi en ascendant (on verra ça plus loin)

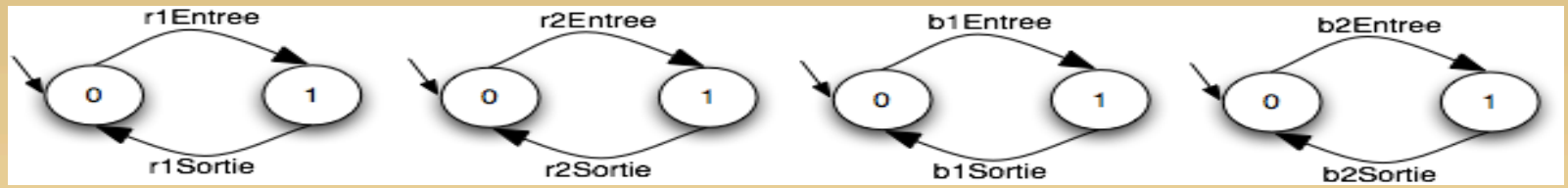
Exemple

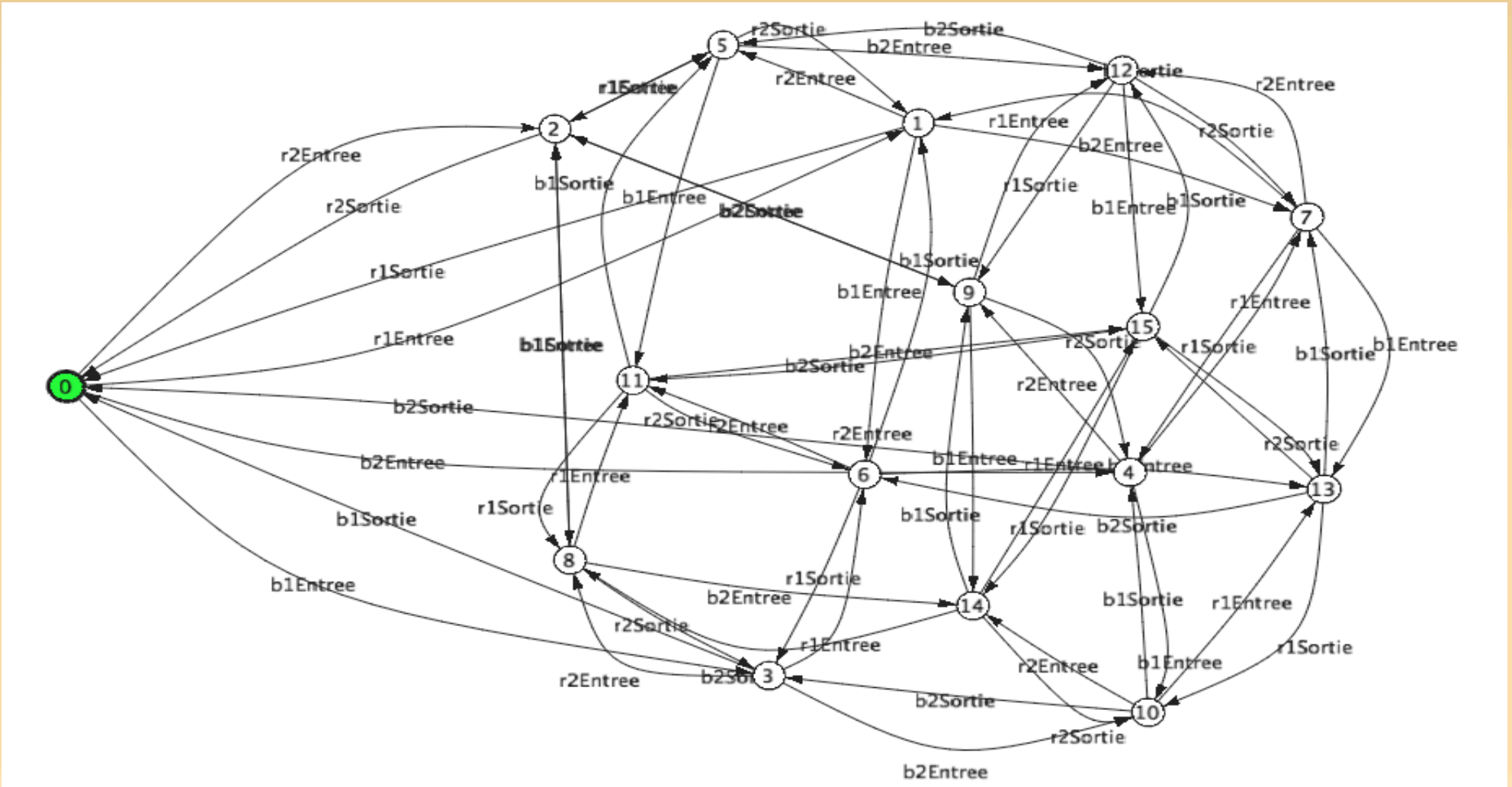
le pont en démarche ascendante

liste des sous-systèmes :

- voitures (ici 2 de chaque couleur)
- pont (1)
- ... et autre chose qu'on découvrira dans la suite

Exemple


r1Entree
0
1
r1Sortie
r2Entree
0
1
r2Sortie
b1Entree
0
1
b1Sortie
b2Entree
0
1
b2Sortie


0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
r1Entree
r1Sortie
r2Entree
r2Sortie
b1Entree
b1Sortie
b2Entree
b2Sortie

Vers un modèle de la composition

un préliminaire important :

- pour construire (puis analyser/vérifier) le modèle global correspondant à la composition d'un système
  il faut d'abord définir la relation entre ses composantes

trois possibilités :

- séquentiel, alternatif et parallèle

pour le pont :

- indépendance entre les voitures
- indépendance entre le pont et les voitures
- → ils sont ”en parallèle”

Composition séquentielle de STE

La composition séquentielle de deux STE

- $L_1 = (A_1,S_1,s_{01},F_1,T_1)$
- $L_2 = (A_2,S_2,s_{02},F_2,T_2)$

est le STE $L=L_1;L_2=(A,S,s_0,F,T)$ tel que :

- $A = A_1 \cup A_2$,
- $S = (S_1 \cup S_2)$, $s_0=s_{01}$, $F=F_2$,
- $T = T_1 \cup T_2$
  $\cup \{(s,a,s') \mid (s_{02},a,s') \in T_2 \text{ AND } s \in F_1\}$

Composition alternative de STE

La composition alternative de deux STE

- $L_1 = (A_1,S_1,s_{01},F_1,T_1)$
- $L_2 = (A_2,S_2,s_{02},F_2,T_2)$

est le STE $L=L_1+L_2=(A,S,s_0,F,T)$ tel que :

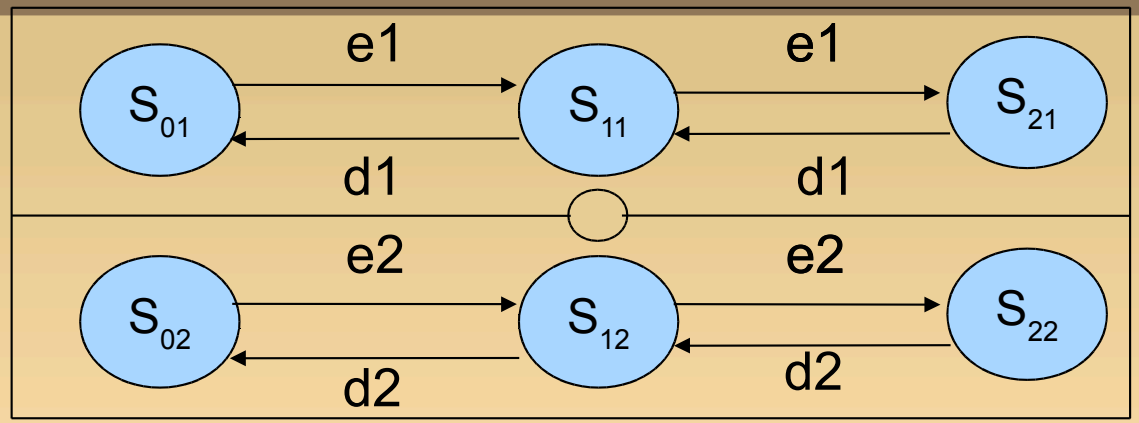
- $A = A_1 \cup A_2$, $F=F_1 \cup F_2$,
- $S = \{s_0\}\cup(S_1\cup S_2) \setminus \{s_{0i} \mid \neg\exists s\in S_i\ (s,a,s_{0i}) \in T_i\}_{i \in [1..2]}$
- $T = T_1 \cup T_2$
  $\cup \{(s_0,a,s') \mid (s_{0i},a,s') \in T_i\}_{i \in [1..2]}$

Composition parallèle de STE

- les deux types de composition précédents sont relativement simples
- l'espace d'état est linéaire p/r aux STE de départ
- dans le cas de la composition parallèle c'est plus compliqué
  - plusieurs possibilités de concurrence :
    1 horloge/1 proc, 1 horloge/n proc, n horloge/n proc
  - l'espace d'état est exponentiel p/r aux STE de départ

Produit libre (1H/1P)

exemple
bi-pile
de taille 2

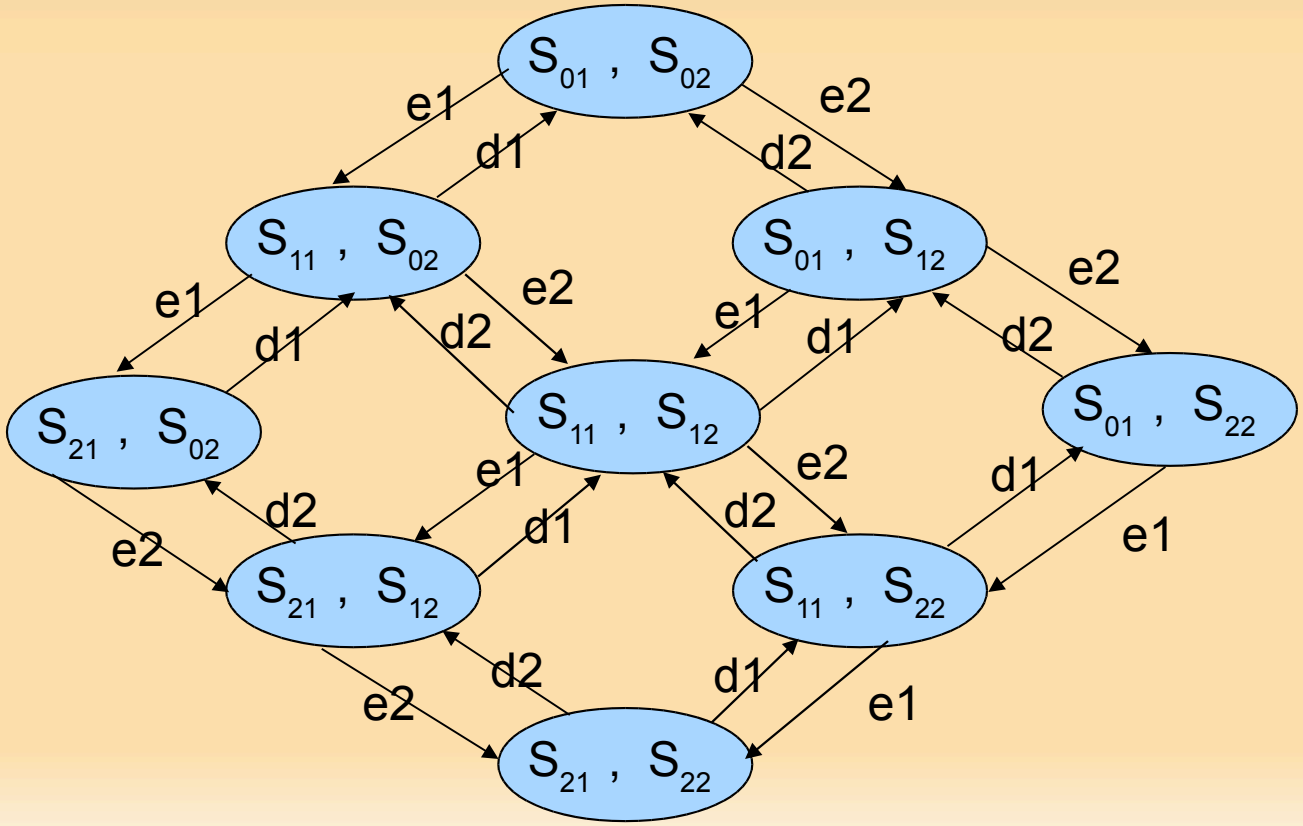


États = produit cartésien des états

$$\{S_{01}, S_{11}, S_{21}\} \times \{S_{02}, S_{12}, S_{22}\}$$

Alphabet = union disjonctive des Alphabets

$$\{e1,d1\} \cup \{e2,d2\}$$



Produit libre (1H/1P)

Le produit libre de deux STE

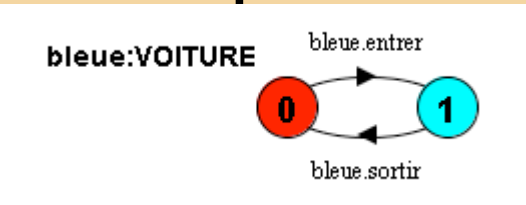
- $L_1 = (A_1,S_1,s_{01},F_1,T_1)$
- $L_2 = (A_2,S_2,s_{02},F_2,T_2)$

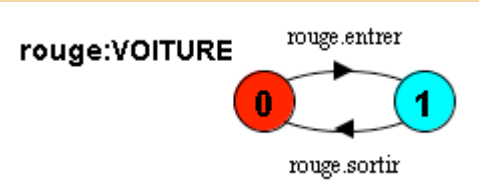
est le STE $L=L_1|||L_2=(A,S,s_0,F,T)$ tel que :

- $A = A_1 \cup A_2$,
- $S=S_1 \times S_2$, $s_0=(s_{01},s_{02})$, $F=F_1 \times F_2$,
- $T = \{((s_1,s_2),a,(s_1',s_2')) \mid$
  $((s_1,a,s_1') \in T_1 \wedge s_2'=s_2)$
  $\vee ((s_2,a,s_2') \in T_2 \wedge s_1'=s_1)\}$

Exemple

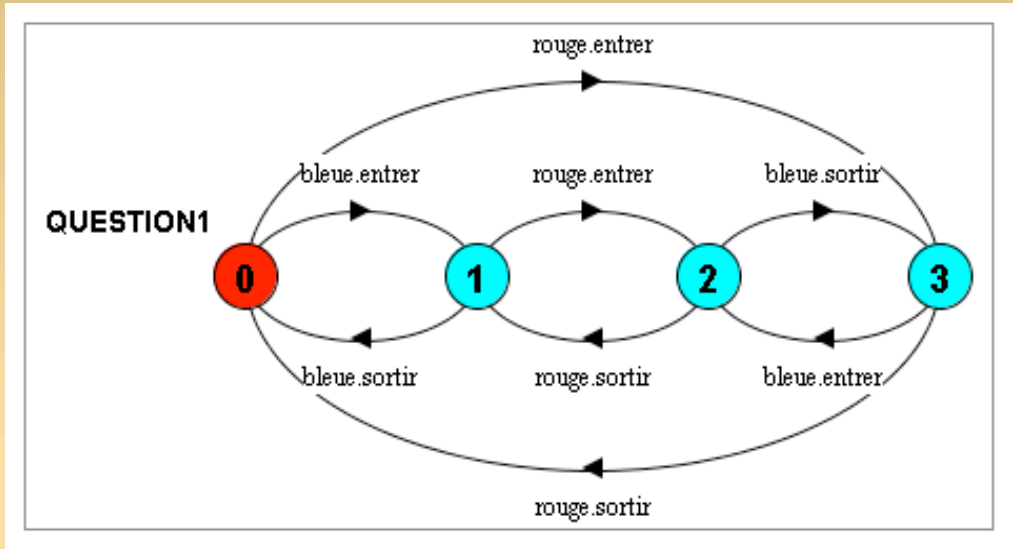
- soit le système constitué de 2 voitures (une rouge, une bleue), dont le comportement est défini par les STE suivants

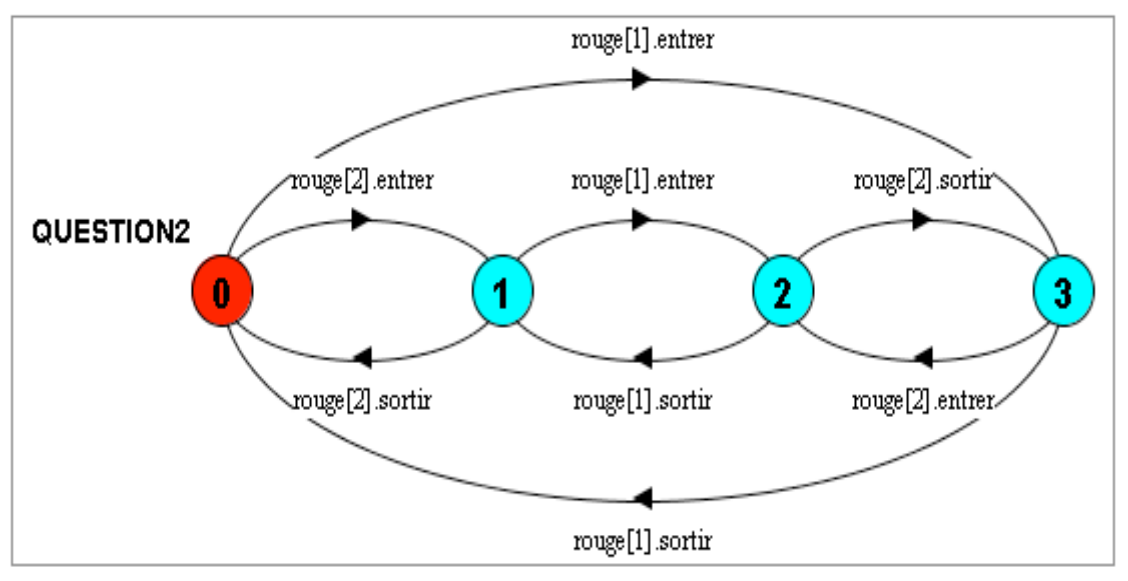




  - STE du système ? remarques ?
- soit le système constitué de 2 voitures de même couleur
  - STE du système ? remarques ?

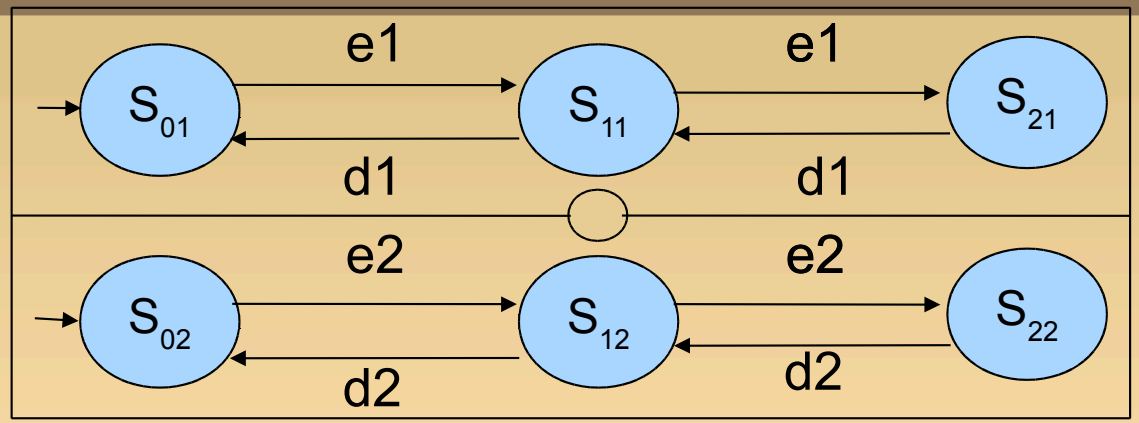
Solutions


QUESTION1
rouge.entrer
bleue.entrer
rouge.entrer
bleue.sortir
0
1
2
3
bleue.sortir
rouge.sortir
bleue.entrer
rouge.sortir


QUESTION2
rouge[1].entrer
rouge[2].entrer
rouge[1].entrer
rouge[2].sortir
0
1
2
3
rouge[2].sortir
rouge[1].sortir
rouge[2].entrer
rouge[1].sortir

Produit cartesien (1H/nP)

exemple
bi-pile
de taille 2

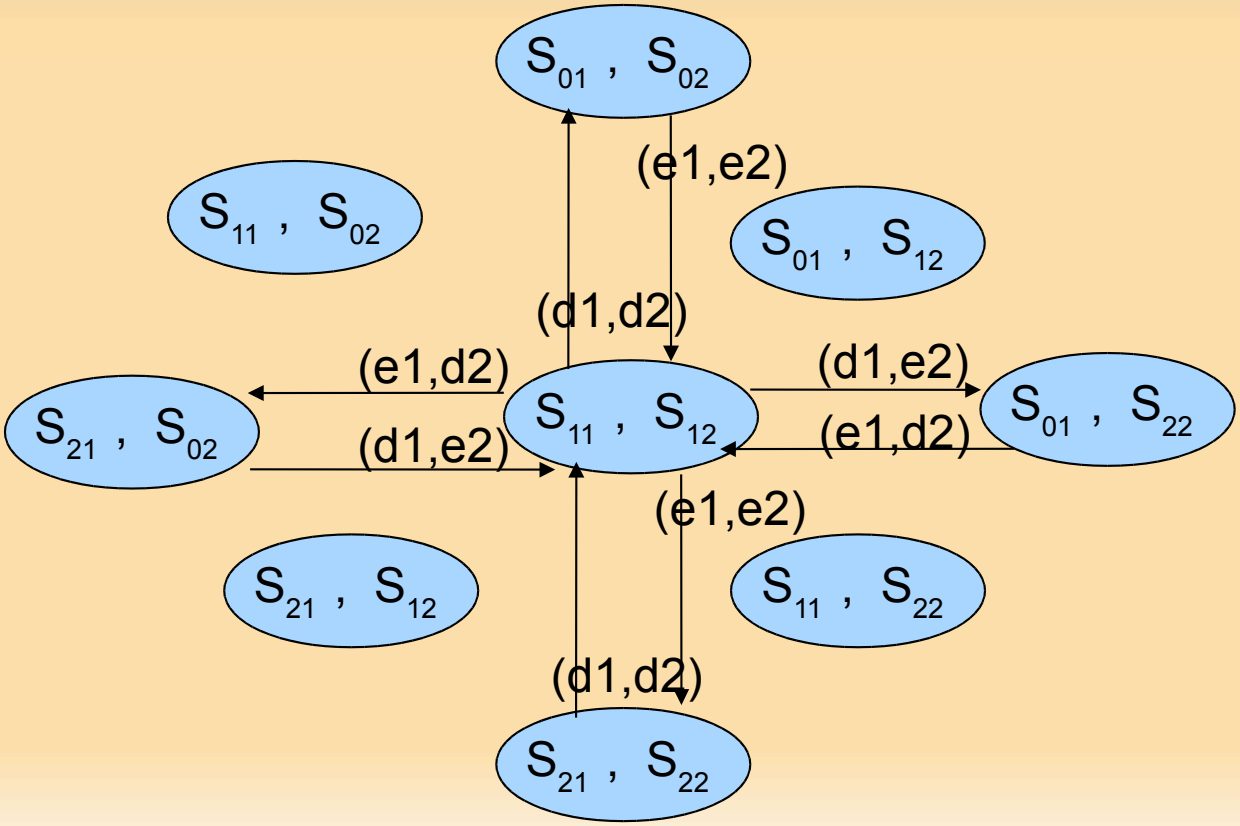


États ⊆ produit cartésien des états

$$\{S_{01}, S_{11}, S_{21}\} \times \{S_{02}, S_{12}, S_{22}\}$$

Alphabet ⊆ produit cartésien des alphabets

{e1,d1} × {e2,d2}



Produit cartésien (1H/nP)

Le produit cartésien de deux STE

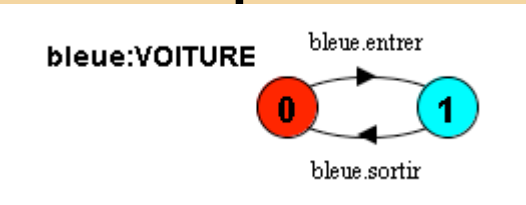
- $L_1 = (A_1,S_1,s_{01},F_1,T_1)$
- $L_2 = (A_2,S_2,s_{02},F_2,T_2)$

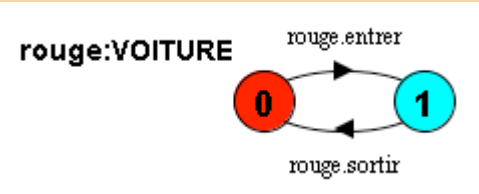
est le STE $L=L_1 \times L_2=(A,S,s_0,F,T)$ tel que :

- $A \subseteq A_1 \times A_2$,
- $S \subseteq S_1 \times S_2$, $s_0=(s_{01},s_{02})$, $F=(F_1 \times F_2) \cap S$,
- $T = \{((s_1,s_2),(a_1,a_2),(s_1',s_2')) \mid$
  $((s_1,a_1,s_1') \in T_1)$
  $\wedge ((s_2,a_2,s_2') \in T_2)\}$

Exemple

- soit le système constitué de 2 voitures (une rouge, une bleue), dont le comportement est défini par les STE suivants

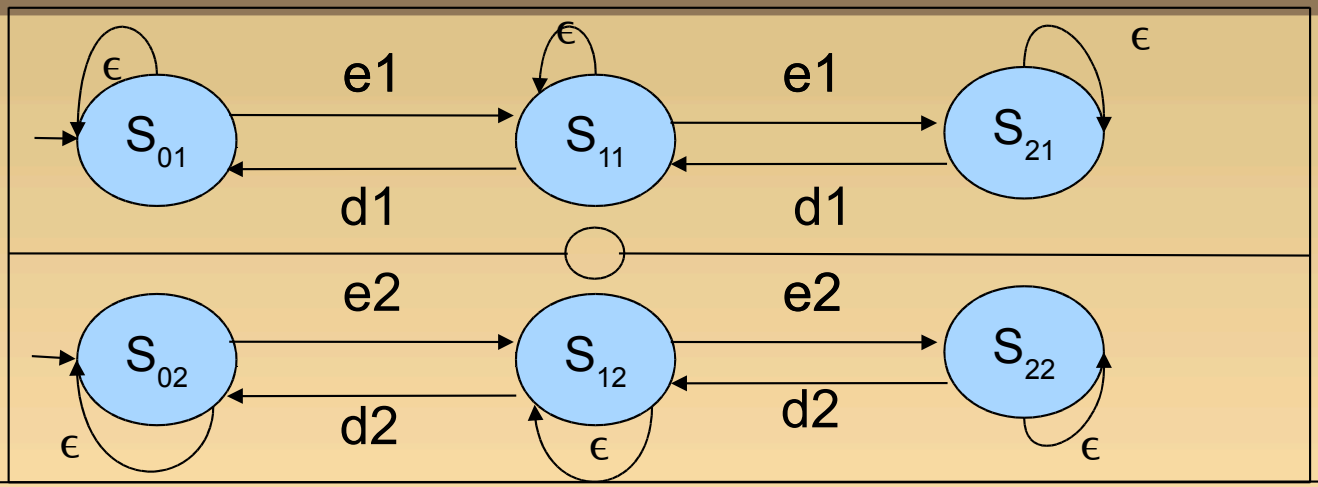




  - STE du système ? remarques ?
- soit le système constitué de 2 voitures de même couleur
  - STE du système ? remarques ?

Produit avec ϵ (nH/nP)

exemple
bi-pile
de taille 2

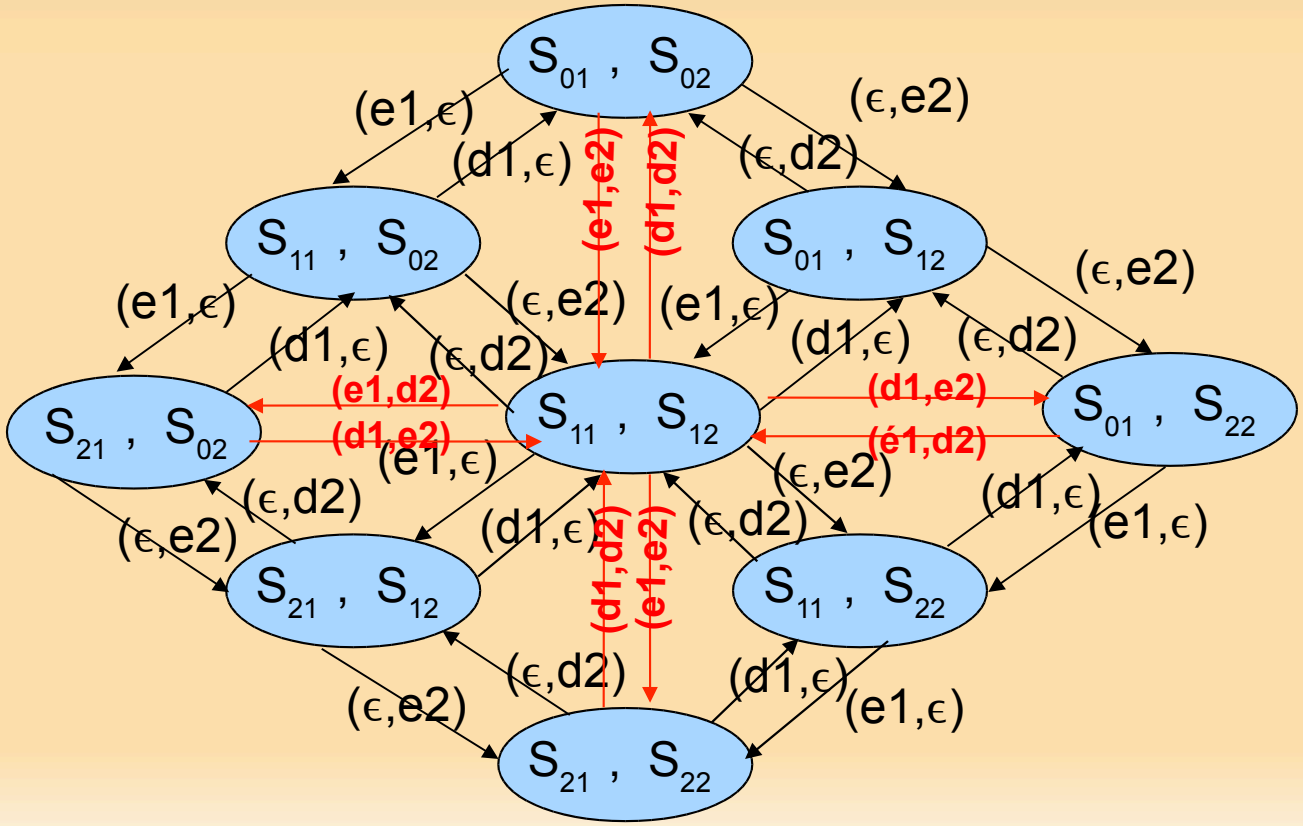


<u>États = produit cartésien des états</u>

$$\{S_{01}, S_{11}, S_{21}\} \times \{S_{02}, S_{12}, S_{22}\}$$

Alphabet = produit cartésien des alphabets

$\{e1,d1,\epsilon\} \times \{e2,d2,\epsilon\}$



Produit avec ϵ (nH/nP)

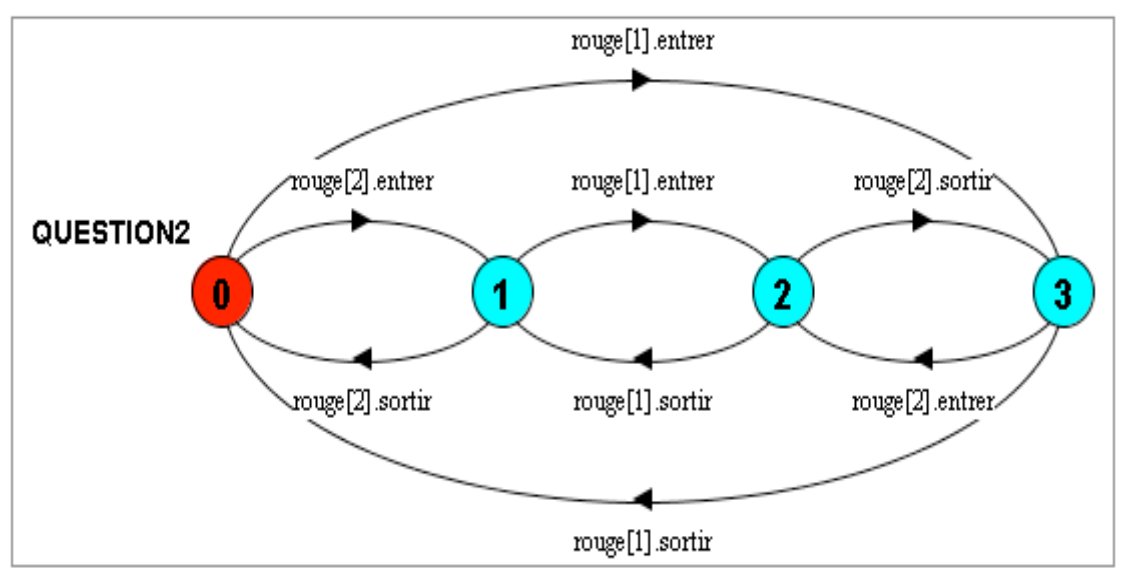
Le produit ϵ de deux STE

- $L_1 = (A_1,S_1,s_{01},F_1,T_1)$, tel que $\epsilon \in A_1$
- $L_2 = (A_2,S_2,s_{02},F_2,T_2)$, tel que $\epsilon \in A2$

est le STE $L=L_1 \times L_2$

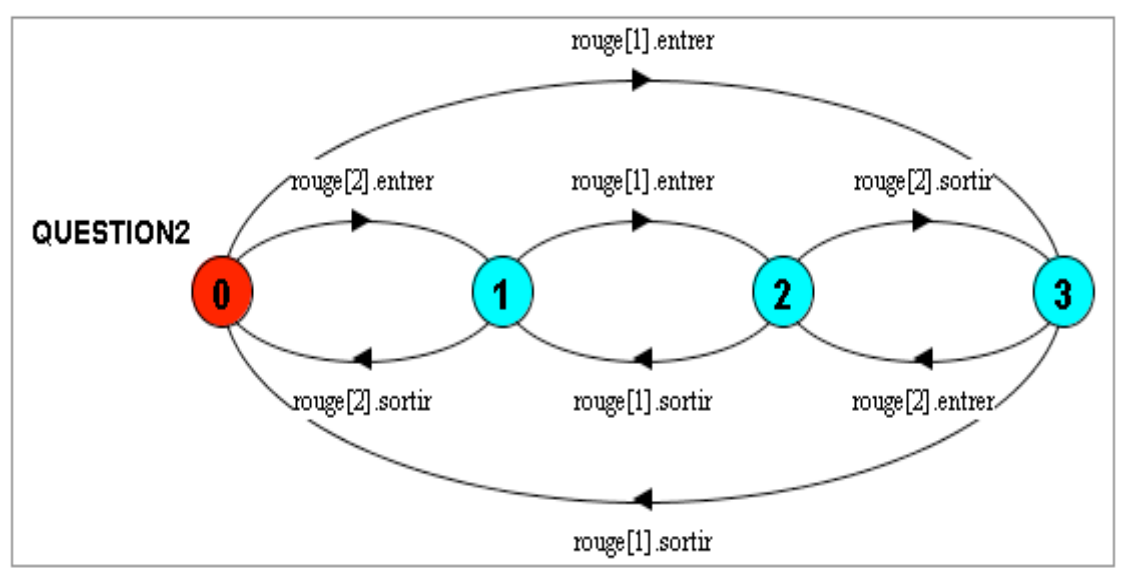
On va garder ça en tête ...
on va voir qu'avec un peu plus
on va pouvoir tout faire avec un seul produit

Exemple



- une voiture peut en doubler une autre sur un pont a voie unique
- deux voitures qui se présentent au pont peuvent rentrer dans n'importe quel ordre indépendemment de leur ordre d'arrivée

Exemple

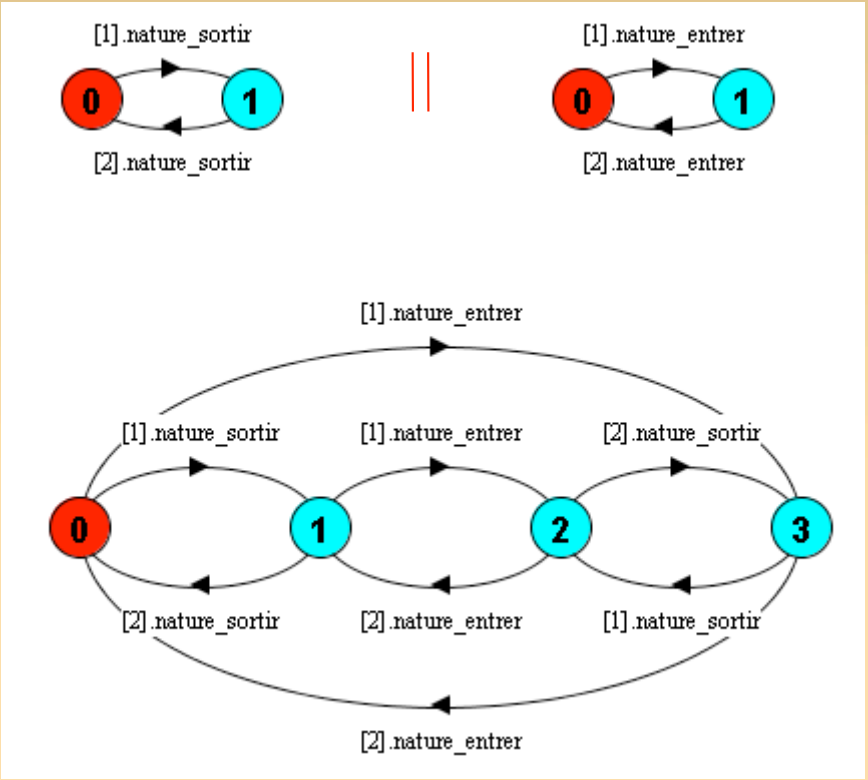


- le modèle n'est pas complètement faux ...
- il manque juste quelque chose : lois d'interaction
  - *si a arrive avant b alors a entre avant b sur le pont*
  - *si a arrive après b sur le pont alors a sort apres b*

Exemple

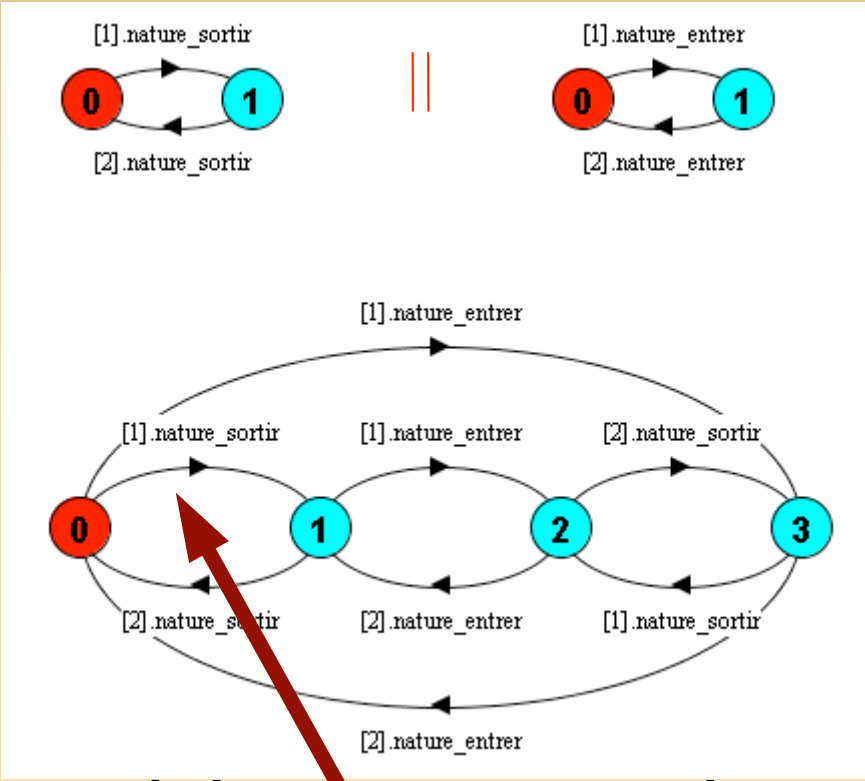
loi_sortir

loi_entrer



Exemple

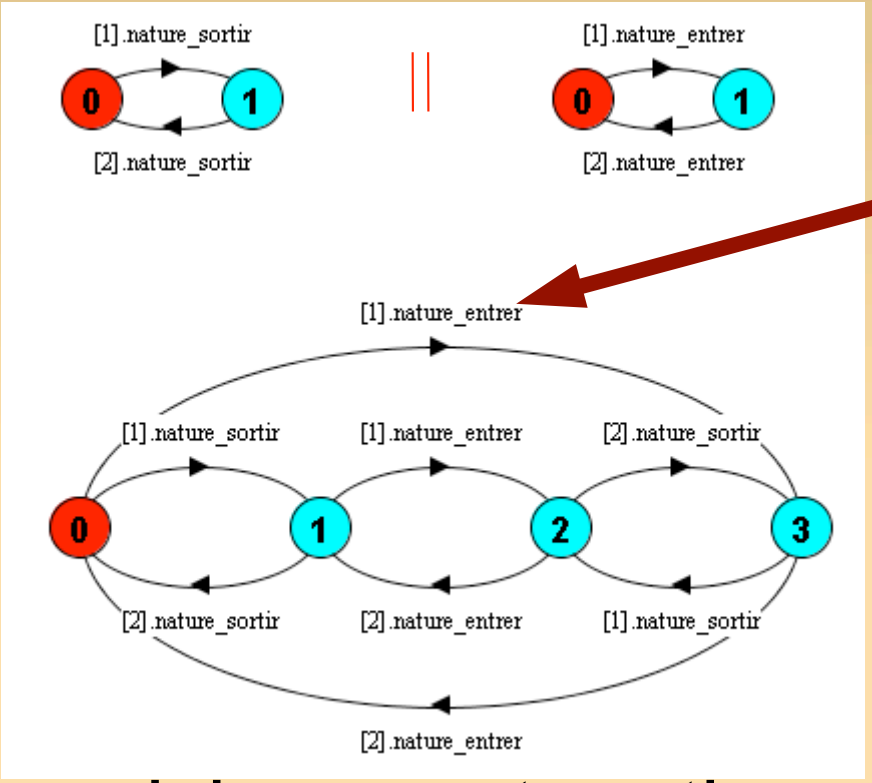
loi_sortir || loi_entrer



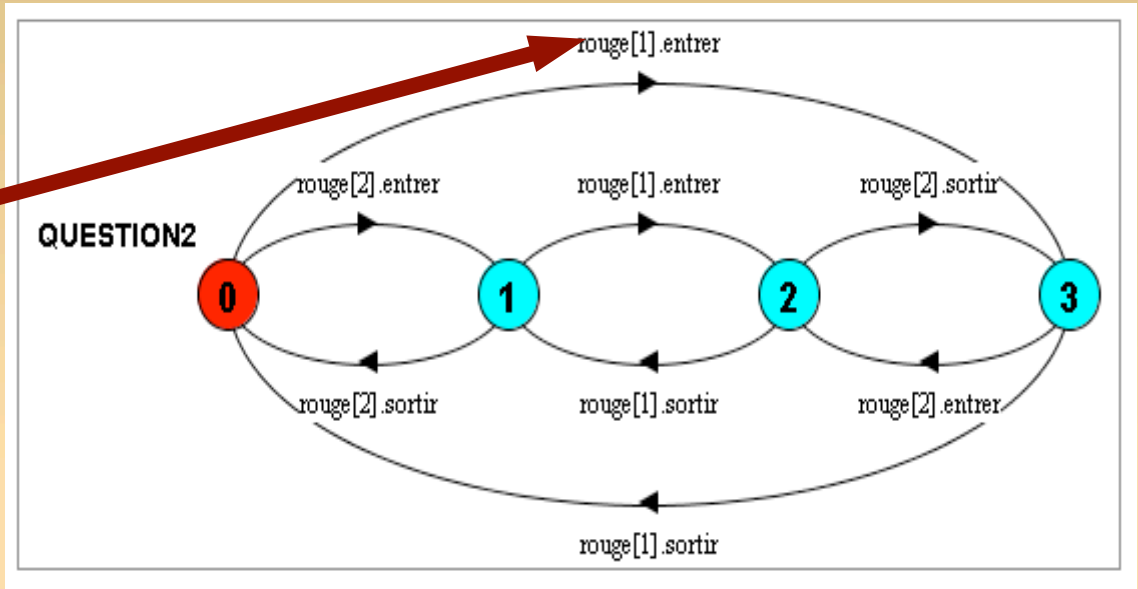
- ici on peut sortir avant d'entrer

Exemple

loi_sortir



loi_entrer



- ici on peut sortir avant d'entrer
- MAIS c'est loi interaction, pas modèle voitures
- interaction $|^{++}|$ voitures : ok (il faut synchroniser)

Produit synchronisé

Le produit synchronisé de deux STE
$L_1 = (A_1,S_1,s_{01},F_1,T_1)$, $L_2 = (A_2,S_2,s_{02},F_2,T_2)$
par rapport à un ensemble de vecteurs
$V = \{(a_1,a_2) \mid a_1 \in A_1 \cup \{\epsilon\} \wedge a_2 \in A_2 \cup \{\epsilon\}\}$

est le STE $L=L_1|V|L_2=(A,S,s_0,F,T)$ tel que :

- $A = V$, $S=S_1 \times S_2$, $s_0=(s_{01},s_{02})$, $F=F_1 \times F_2$,
- $T = \{((s_1,s_2),(a_1,a_2),(s_1',s_2')) \mid (a_1,a_2) \in V$
  $\wedge ((s_1,a_1,s_1') \in T_1)$
  $\wedge ((s_2,a_2,s_2') \in T_2)\}$

Relation avec les autres produits

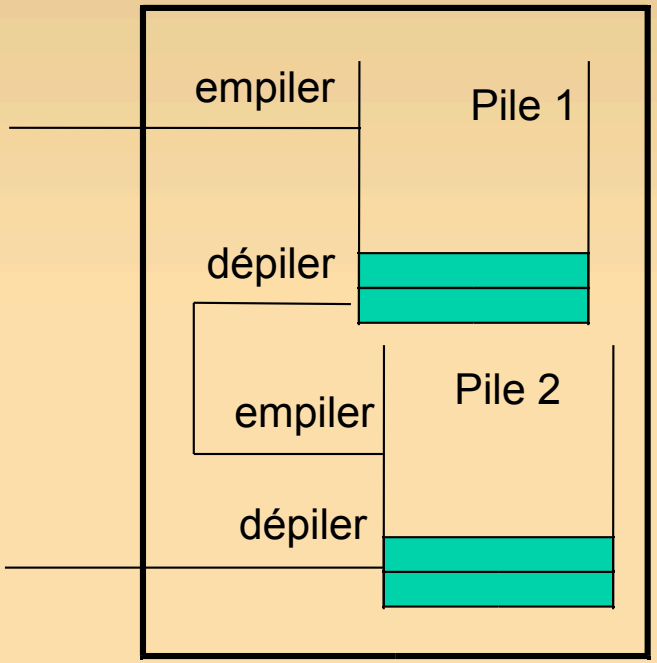
- $L_1 ||| L_2 = L_1 \,|V|\, L_2$
  avec $V=(A_1 \times \{\epsilon\}) \cup (\{\epsilon\} \times A_2)$

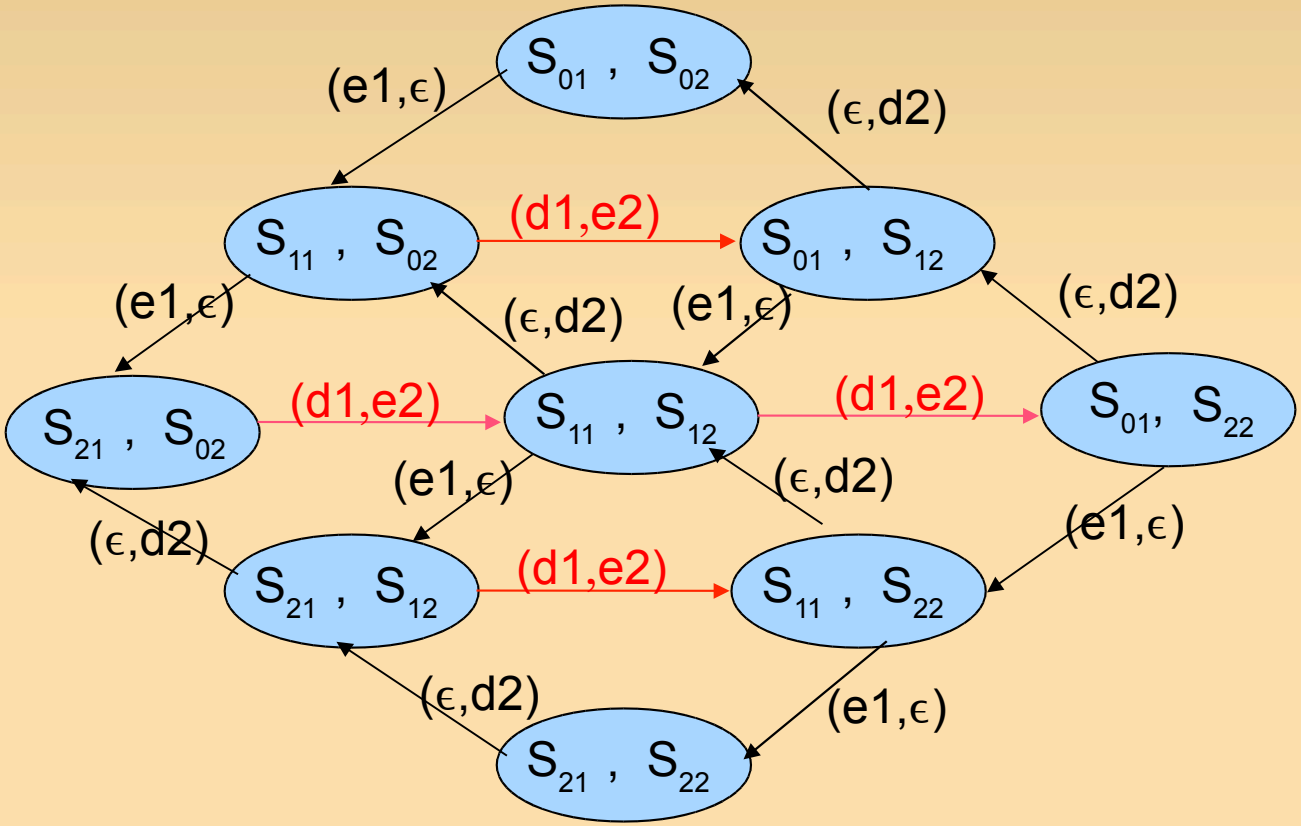
- $L_1 \times L_2 = L_1 \,|V|\, L_2$
  avec $V=A_1 \times A_2$

- un petit nouveau: $L_1 \,||\, L_2 = L_1 \,|V|\, L_2$
  avec $V=\{(a,a) \mid a \in (A_1 \cap A_2)\}$

Exemple

pile de taille 4 avec deux piles de taille 2





V = {(e1,ϵ),
(d1,e2),
(ϵ,d2)}

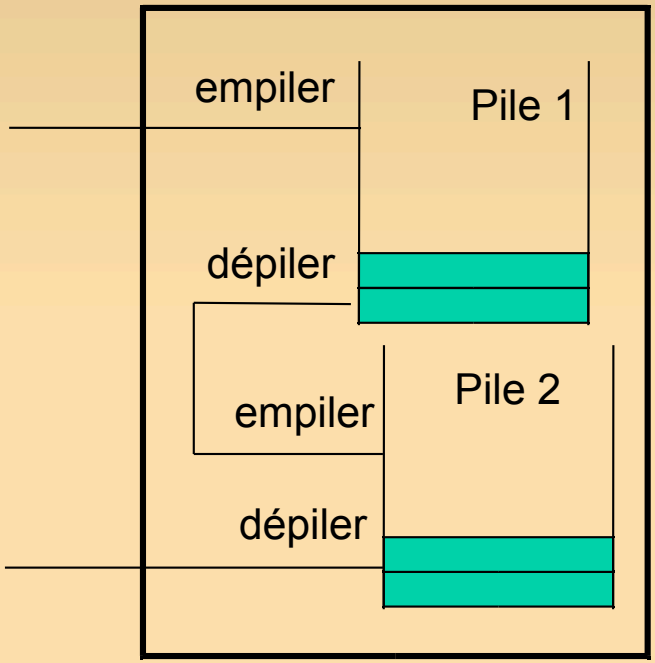
Produit synchronisé (avec matrice)

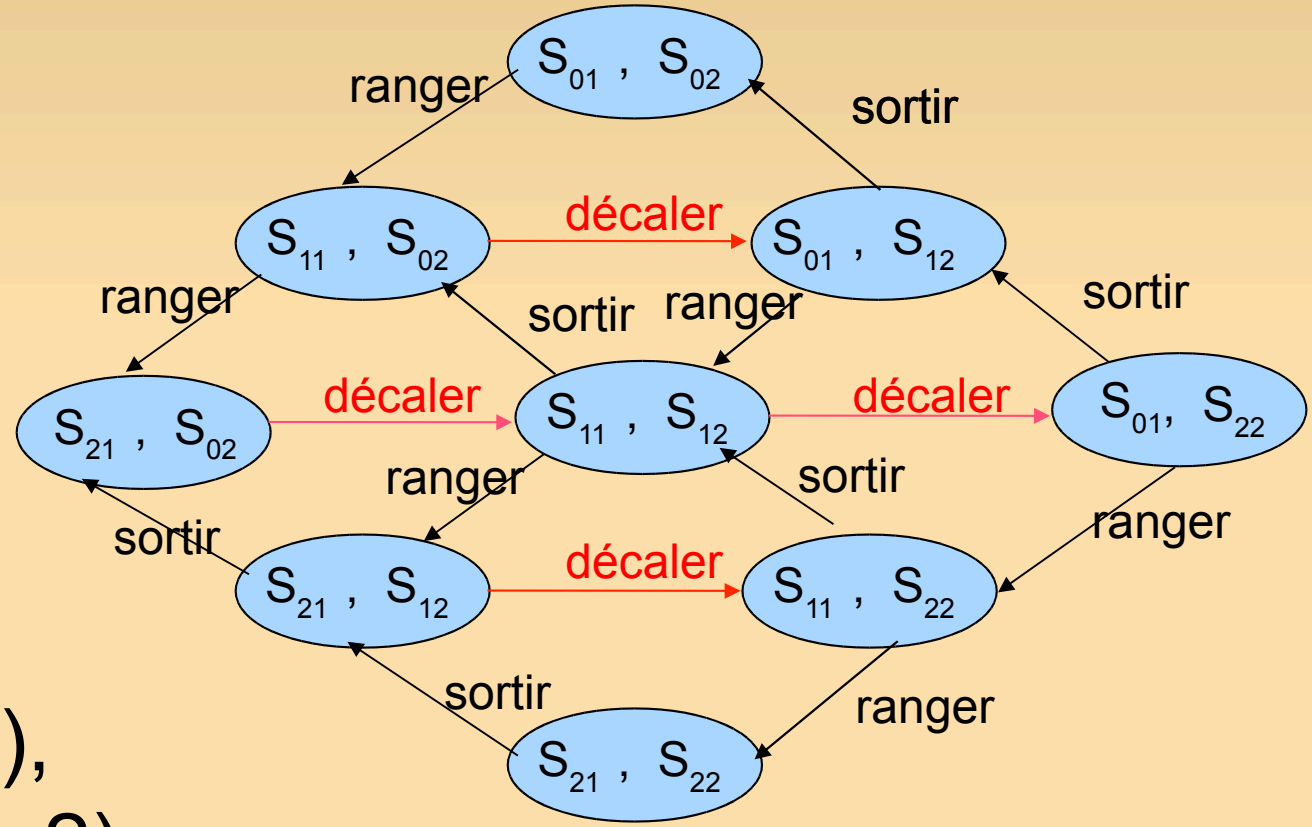
- on peut associer un nouveau label aux synchronisations

	Automate A	Automate B
ranger	**e1**	**ϵ**
décaler	**d1**	**e2**
sortir	**ϵ**	**d2**

Exemple

pile de taille 4 avec deux piles de taille 2





V = {ranger:(e1,ϵ),
décaler:(d1,e2),
sortir:(ϵ,d2)}

Produit synchronisé (avec matrice)

- on peut associer un nouveau label aux synchronisations
- permet la compositionalité
  - STE produits/STE simples : même type étiquettes
  - utiliser un STE produit dans un produit
    → sous-systèmes composites

MACHINE	A	B
X	1	2
Y	3	ϵ
Z	ϵ	4

	AA	MACHINE	BB	CC
EVT1	a	X	ϵ	ϵ
EVT2	b	Y	ϵ	ϵ
EVT3	c	Z	1	ϵ

Produit synchronisé (avec matrice)

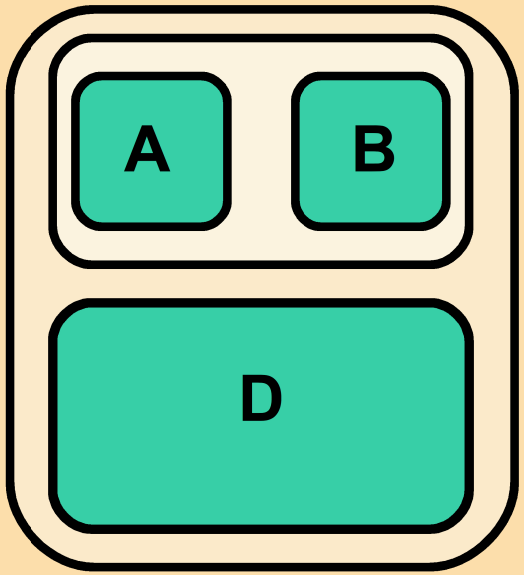
- on peut associer un nouveau label aux synchronisations
- permet la compositionalité
  - STE produits/STE simples : même type étiquettes
  - utiliser un STE produit dans un produit
    → sous-systèmes composites

MACHINE	AA	A	B	BB	CC
EVT1	a	1	2	ϵ	ϵ
EVT2	b	3	ϵ	ϵ	ϵ
EVT3	c	ϵ	4	1	ϵ

	AA	MACHINE	BB	CC
EVT1	a	X	ϵ	ϵ
EVT2	b	Y	ϵ	ϵ
EVT3	c	Z	1	ϵ

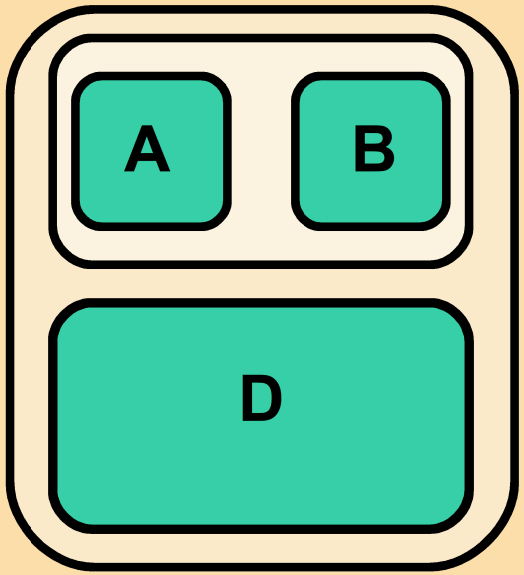
Passage à l'échelle (retour)

- démarche ascendante
  composer pour réutiliser
  1. définir (ou réutiliser) les sous-systèmes
  2. construire le système
  3. vérifier l'assemblage



Passage à l'échelle (retour)

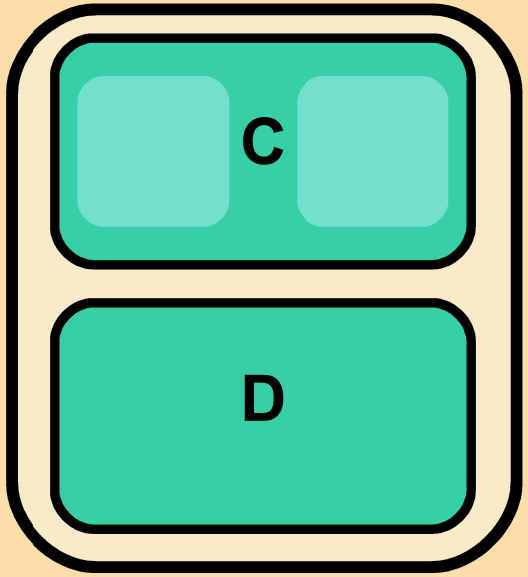
- démarche ascendante
  composer pour réutiliser
  1. définir (ou réutiliser) les sous-systèmes
  2. construire le système
  3. vérifier l'assemblage



1. A, B, $V_{A,B}$

Passage à l'échelle (retour)

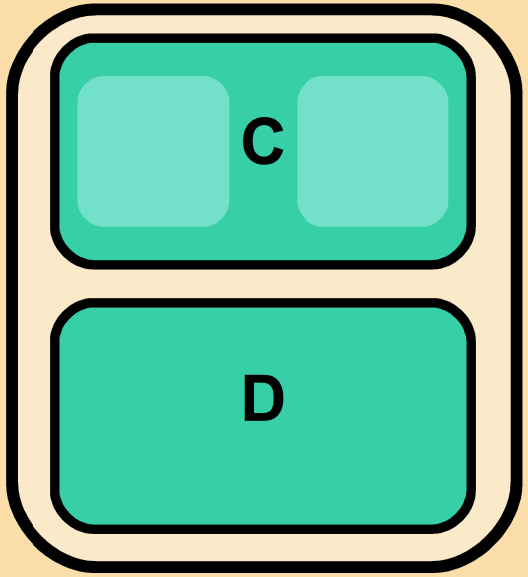
- démarche ascendante
  composer pour réutiliser
  1. définir (ou réutiliser) les sous-systèmes
  2. construire le système
  3. vérifier l'assemblage



1. A, B, $V_{A,B}$
2. C=A|$V_{A,B}$|B
3. C ok ?

Passage à l'échelle (retour)

- démarche ascendante
  composer pour réutiliser
  1. définir (ou réutiliser) les sous-systèmes
  2. construire le système
  3. vérifier l'assemblage

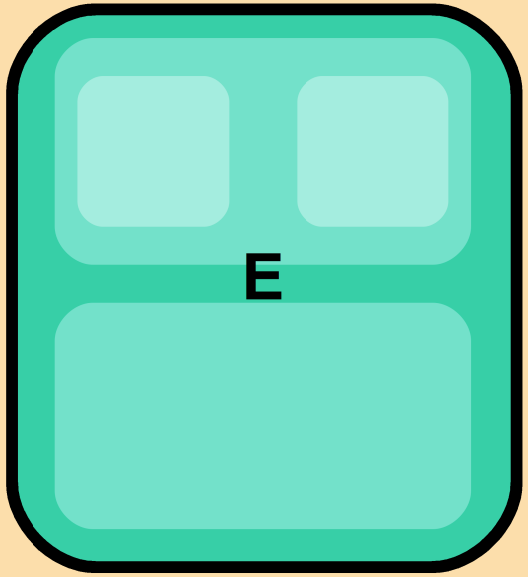


1. A, B, $V_{A,B}$
2. C=A|$V_{A,B}$|B
3. C ok ?

1. C, D, $V_{C,D}$

Passage à l'échelle (retour)

- démarche ascendante
  composer pour réutiliser
  1. définir (ou réutiliser) les sous-systèmes
  2. construire le système
  3. vérifier l'assemblage



1. A, B, $V_{A,B}$
2. C=A|$V_{A,B}$|B
3. C ok ?

1. C, D, $V_{C,D}$
2. E=C|$V_{C,D}$|D
3. E ok ?

Exemple

Algorithme de Peterson (1968)

var d0:bool := false
var d1:bool := true
var t in {0,1} := 0

loop forever // P0
1: {NCS0}
2: d0 := true
3: t:=0
4: **wait** (d1=false **or** t=1)
5: {B_CS0}
6: {E_CS0}
7: d0 := false
endloop

loop forever // P1
1: {NCS1}
2: d1 := true
3: t:=1
4: **wait** (d0=false **or** t=0)
5: {B_CS1}
6: {E_CS1}
7: d1 := false
endloop

Exemple

Algorithme de Peterson (1968)

var d0:bool := false
var d1:bool := true
var t in {0,1} := 0

loop forever // P0
1: {NCS0}
2: d0 := true
3: t:=0
4: **wait** (d1=false **or** t=1)
5: {B_CS0}
6: {E_CS0}
7: d0 := false
endloop

loop forever // P1
1: {NCS1}
2: d1 := true
3: t:=1
4: **wait** (d0=false **or** t=0)
5: {B_CS1}
6: {E_CS1}
7: d1 := false
endloop

Exemple

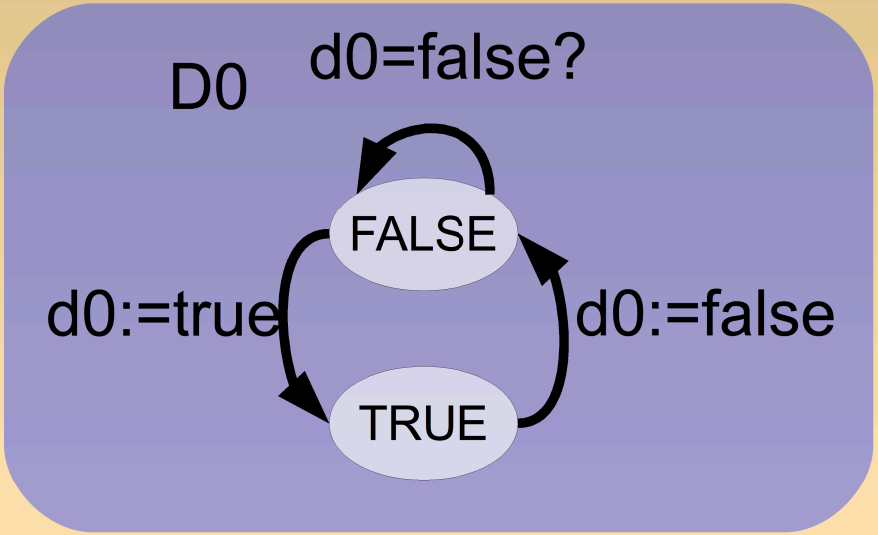
Algorithme de Peterson (1968)

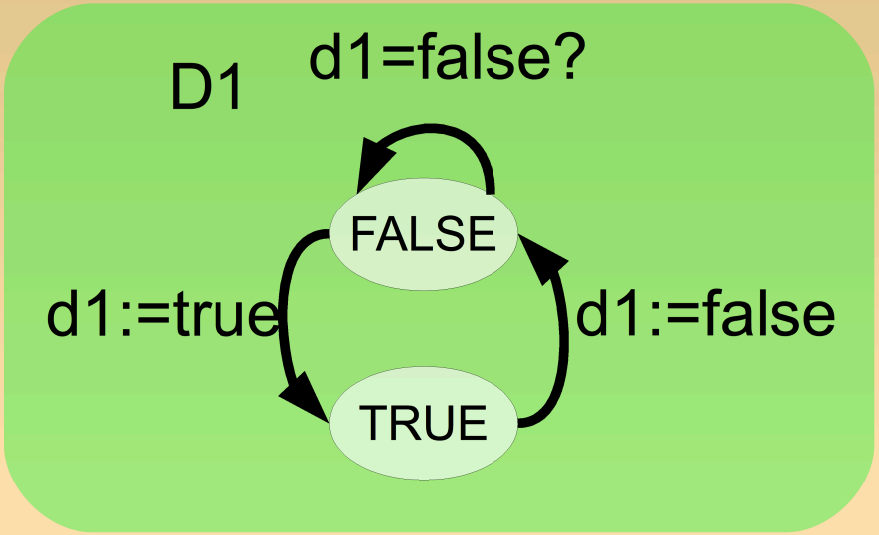
```
var d0:bool := false
var d1:bool := true
var t in {0,1} := 0
```

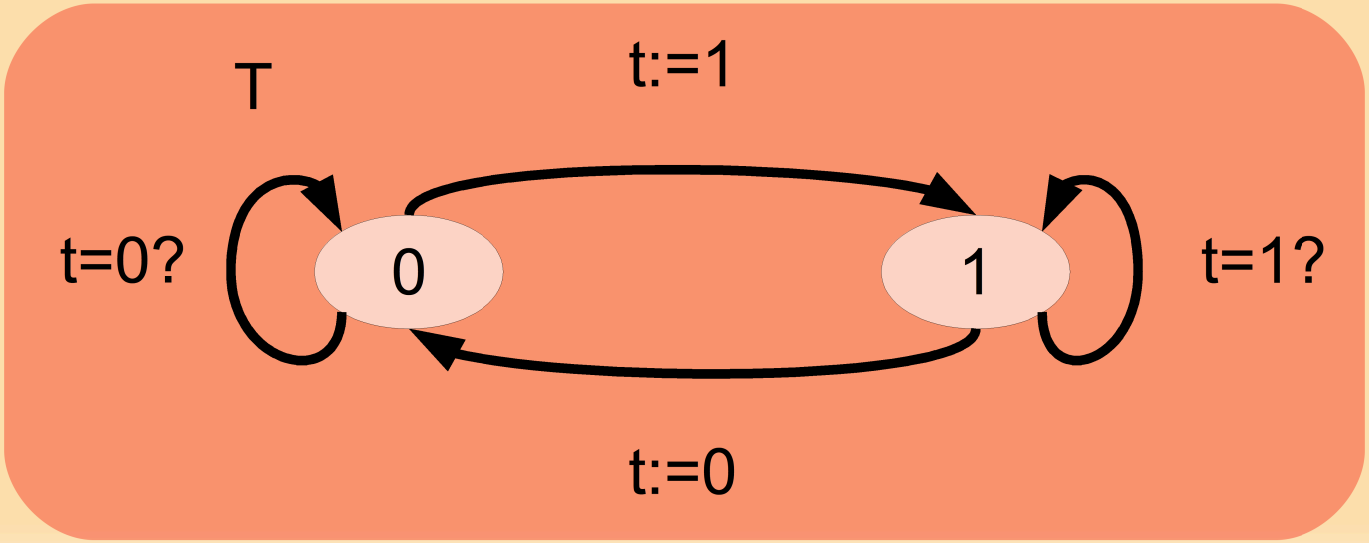
loop forever // P0
1: {NCS0}
2: d0 := true
3: t:=0
4: **wait** (d1=false **or** t=1)
5: {B_CS0}
6: {E_CS0}
7: d0 := false
endloop

loop forever // P1
1: {NCS1}
2: d1 := true
3: t:=1
4: **wait** (d0=false **or** t=0)
5: {B_CS1}
6: {E_CS1}
7: d1 := false
endloop

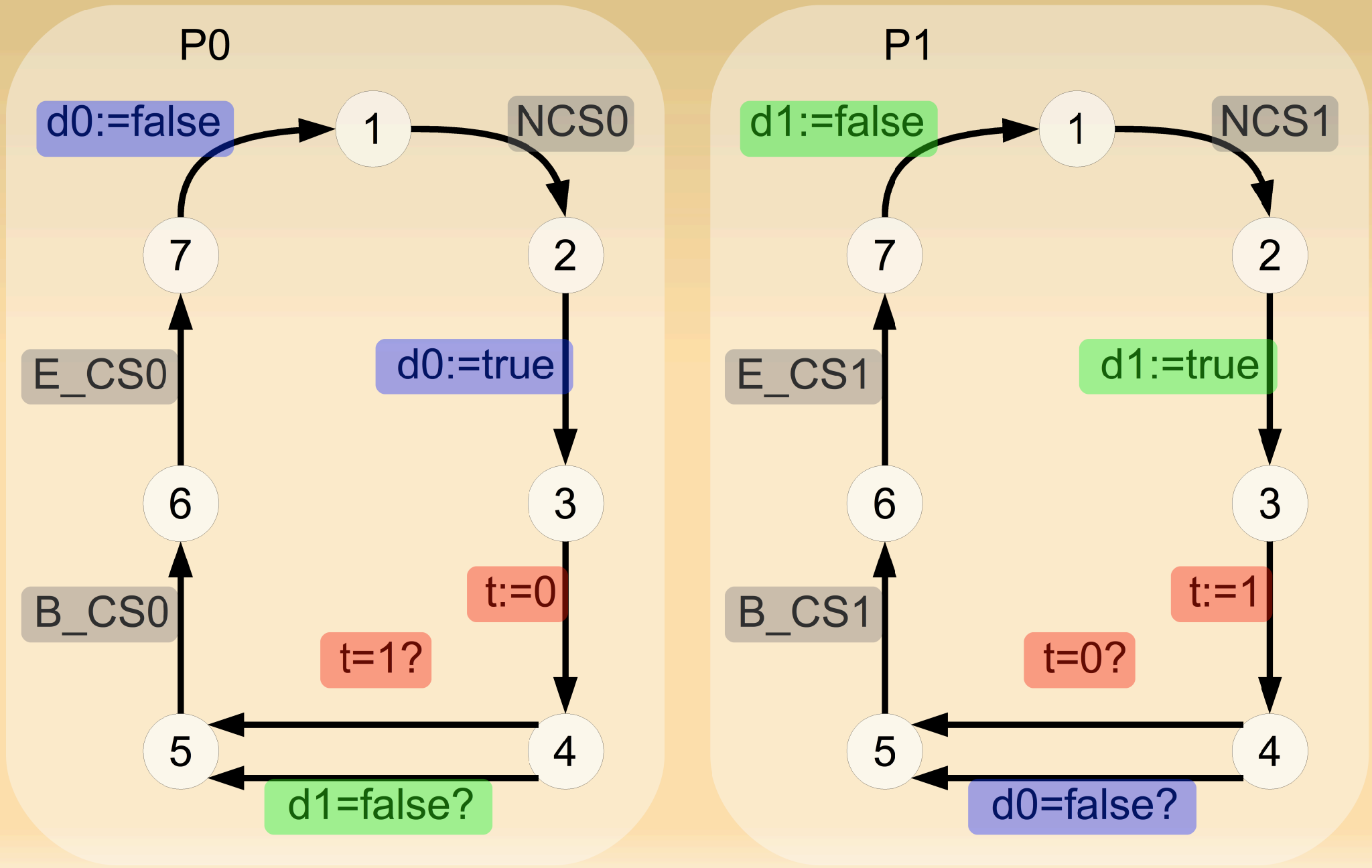
Exemple : composants



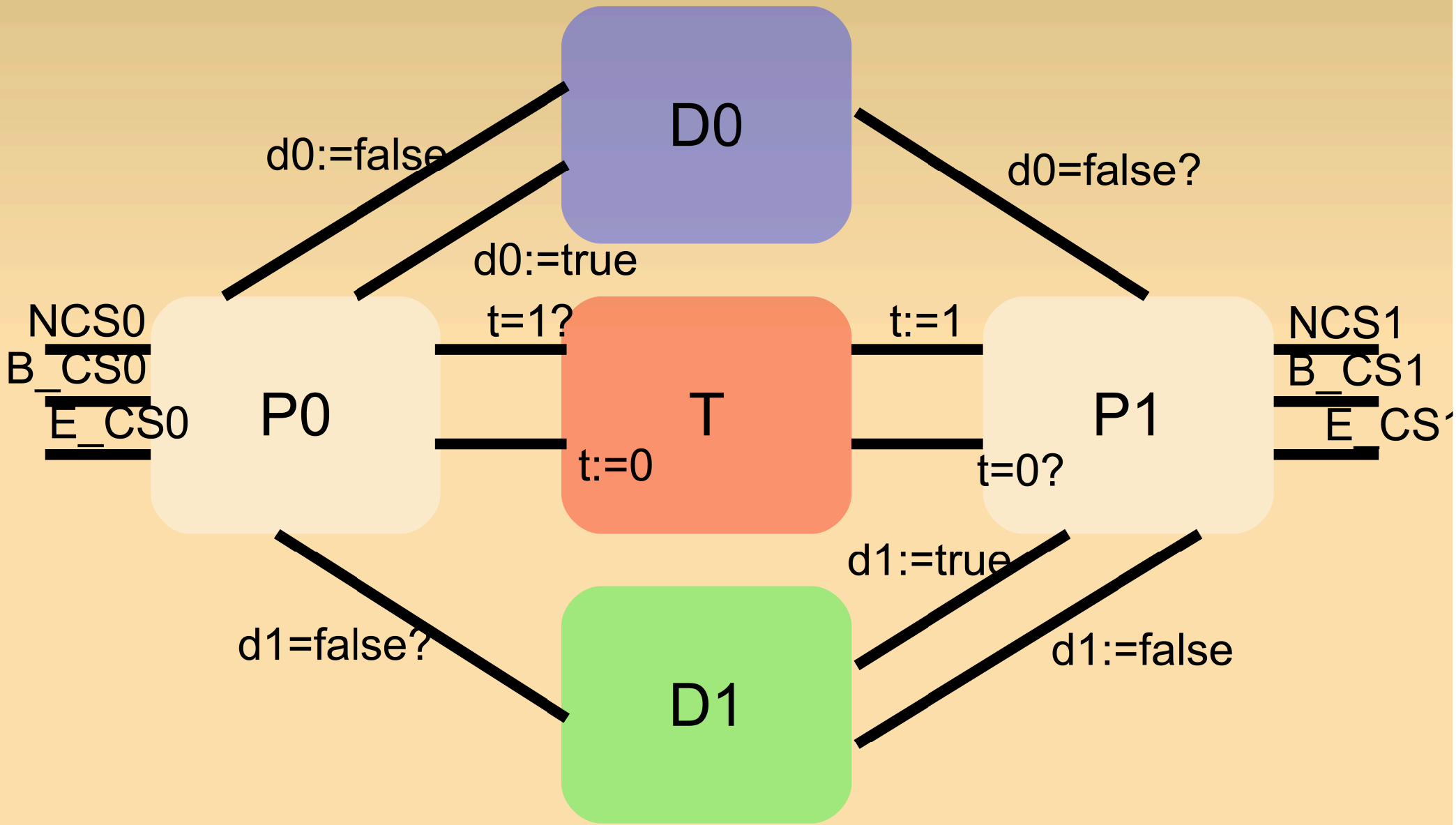




Exemple : composants

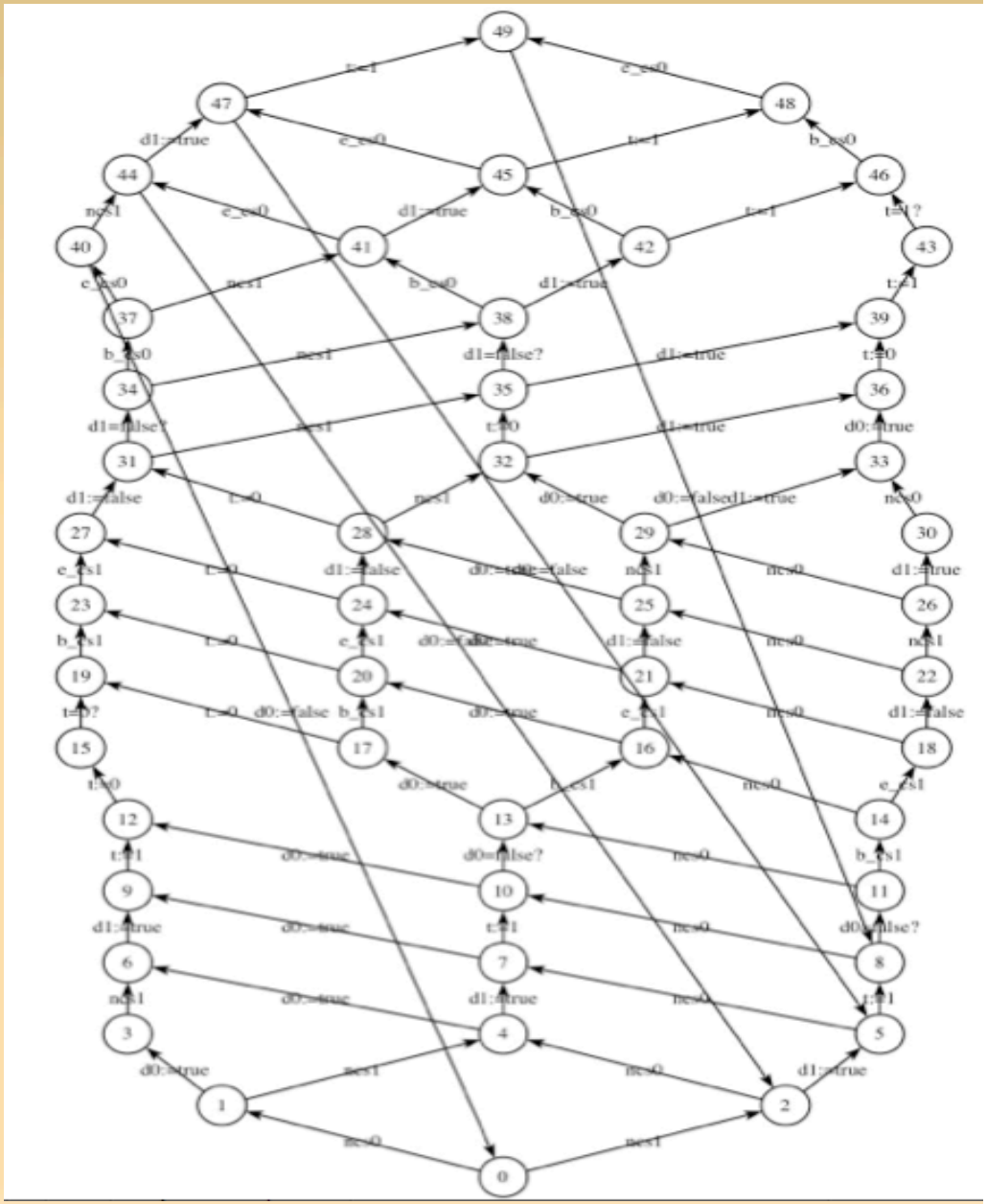


Exemple : vecteurs



Exemple : produit et vérification

- STE fini
  $|S| = 50 \leq 2 \times 2 \times 2 \times 7 \times 7$
- remarque: en présence de synchronisations $|S|$ du produit est (bcp) plus petit que $\times_i |S_i|$
- propriétés :
  - absence de blocage
  - exclusion mutuelle : pas d'état (_,_,_,6,6)
  - vivacité



Renommage et masquage

- opérations très utiles pour la structuration de systèmes (composites)
- soit un STE L = (A,S,I,F,T)
  - renommage L[f] avec f : $(A\backslash\{\tau\}) \rightarrow A$
    L[f] = (f(A),S,I,F,T[f]), où T[f] = {(s,f(a),s') | (s,a,s') ∈ T}
  - masquage α_BL avec $B \subseteq A$
    $\alpha_B L = (A\backslash B \cup \{\tau\}, S, I, F, \alpha_B T)$, où $\alpha_B T = \{(s,\alpha_B a,s') \mid (s,a,s') \in T\}$
    avec $\alpha_B a = \tau$ si $a \in B$ et $\alpha_B a = a$ sinon
    dérivé : projection $\pi_B L = \alpha_{(A\backslash B)} L$

Partie 1 – Modèle comportemental STE communicants

- Partie 1 – modèle comportemental de système
  - Système de Transitions Etiquetées (STE / *LTS*)
  - sémantique STE de la concurrence synch./asynch.
  - STE communicants
  - autour de la comparaison de STE
- Partie 2 – ”langages” de processus
- Partie 3 – propriétés

Le point et les objectifs

- le comportement d'un système simple (séquentiel) peut être modélisé par un STE
- le comportement d'un système composite (concurrent) peut être modélisé par une architecture et le produit de STE
- on va s'intéresser à des systèmes particuliers
  - ouverts, interagissant avec un environnement
  - distribués, communicants
  - exemples : services Web

Système interactif communicant

- système ouvert : au contraire d'un système clos, il peut interagir avec un environnement :
  - soit un autre système ouvert
    avec lequel il sera composé au sein d'un système composite
    soit un utilisateur
  - système conversationnel vs. système réactif
- système distribué : sous-systèmes concurrents s'exécutant sur différentes machines
- système communicant : importance des échanges entre systèmes vs. actions internes
  actions = $A^{?} \cup A^{!} \cup \{\tau\}$

Modélisation

- $A^?$: entrées, ensemble de symboles (sans τ)
  éléments notés ?a, (simplement) a ou a○
- $A^!$: ensemble associé de sorties (sans τ)
  éléments notés !a, 'a, $\overline{a}$ ou a●
- la notion d'entrée/sortie est une interprétation
  en fait il s'agit d'actions complémentaires
  (on pourrait inverser, ex: 'a entrée, a sortie)
- $Act = A^? \cup A^! \cup \{\tau\}$
- $A^! = (A^?)^c$ et $A^? = (A^!)^c$ avec $(X)^c = \{x^c \mid x \in X\}$
  $(?a)^c = !a$, $(!a)^c = ?a$, $((x)^c)^c = x$, $\tau^c = \tau$

Renommage (bis)

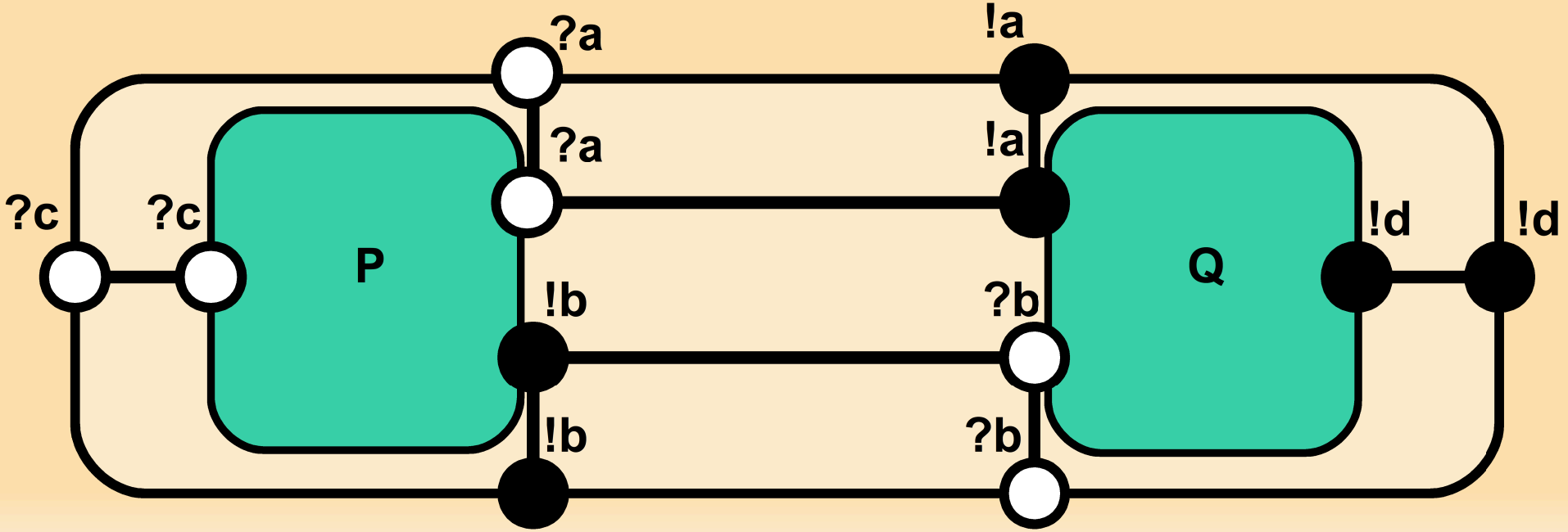
- le renommage est adapté
  soit
  $f : A^? \rightarrow A^?$ (rappel : $A^?$ ne contient pas τ)
  alors
  $f^c : Act \rightarrow Act$
  $f^c(a)=f(a)$ si $a \in A^?$
  $f^c(a)=(f(a^c))^c$ si $a \in A^!$
  $f^c(\tau)=\tau$
- exemples: (notations alternatives: $[x_i \rightarrow y_i]$ vs. $[y_i/x_i]$)
  ?e [e→empiler, d→dépiler] = ?e [?empiler/?e] = ?empiler
  !d [e→empiler, d→dépiler] = !d [?dépiler/?d] = !dépiler

Masquage (bis)

- le masquage est adapté
  $\alpha_B L$ avec $B \subseteq A$
  $\alpha_B L = (A \backslash B \cup \{\tau\}, S, I, F, \alpha_B T)$
  où $\alpha_B T = \{(s, \alpha_B a, s') \mid (s,a,s') \in T\}$
  avec $\alpha_B a = \tau$ si $a \in (B \cup B^c)$ et $\alpha_B a = a$ sinon
- exemples:
  $\alpha_{\{?dépiler\}}?empiler = \alpha_{\{dépiler\}}?empiler = ?empiler$
  $\alpha_{\{?empiler\}}!empiler = \alpha_{\{empiler\}}!empiler = \tau$
  $\alpha_{\{?empiler\}}?empiler = \alpha_{\{empiler\}}?empiler = \tau$

Sémantique syst. communicant

- soit un système S constitué de :
  - $P=(Act_P,...)$, $Act_P=A^?_P \cup A^!_P \cup \{\tau\}$ (certains evt. vides)
  - $Q=(Act_Q,...)$, $Act_Q=A^?_Q \cup A^!_Q \cup \{\tau\}$ (certains evt. vides)

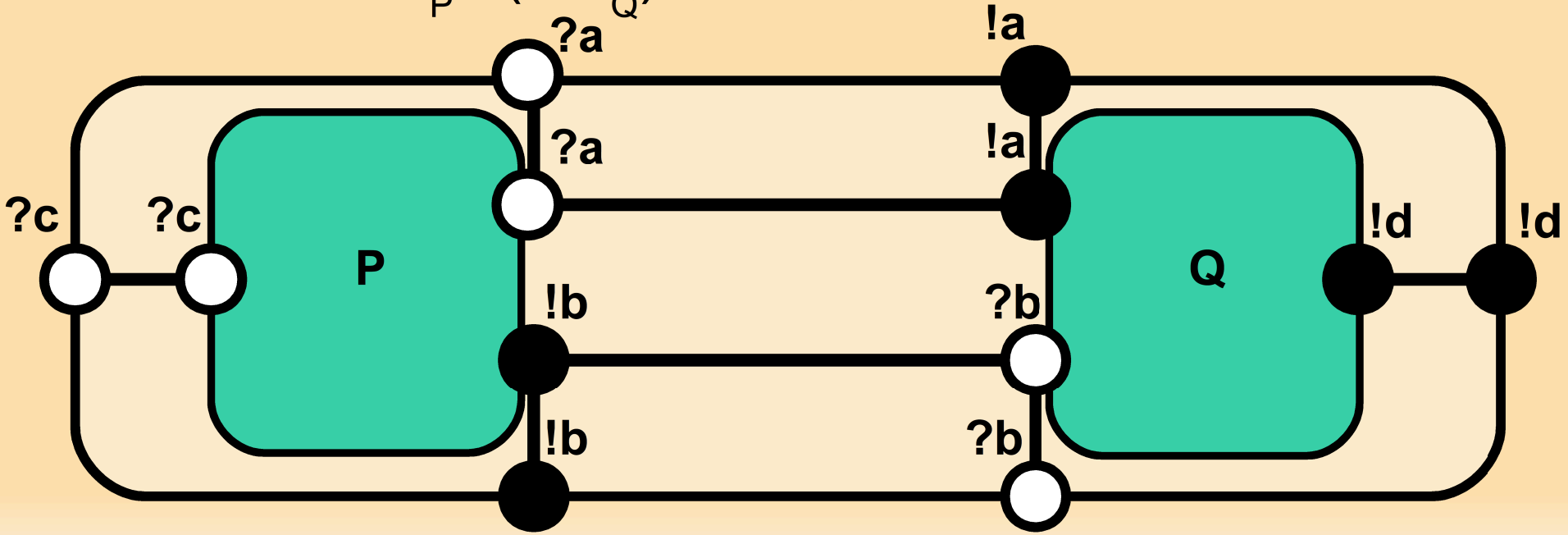


Sémantique syst. communicant

- S=P|Q=P|V|Q avec
  - ∀ a ∈ Act_P , a:<a,ϵ> ∈ V
  - ∀ a ∈ Act_Q , a:<ϵ,a> ∈ V
  - ∀ a ∈ $Act_P \cap (Act_Q)^c$, τ:<a,a^c> ∈ V

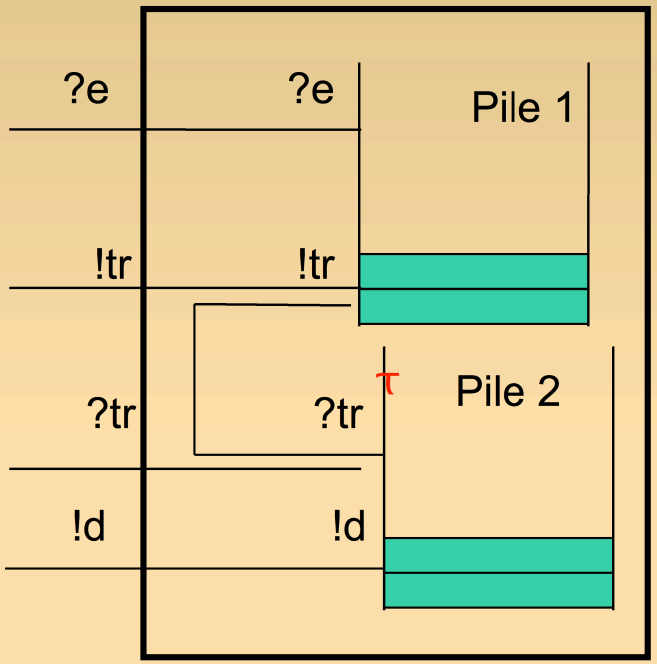
attention : P|Q != P||Q

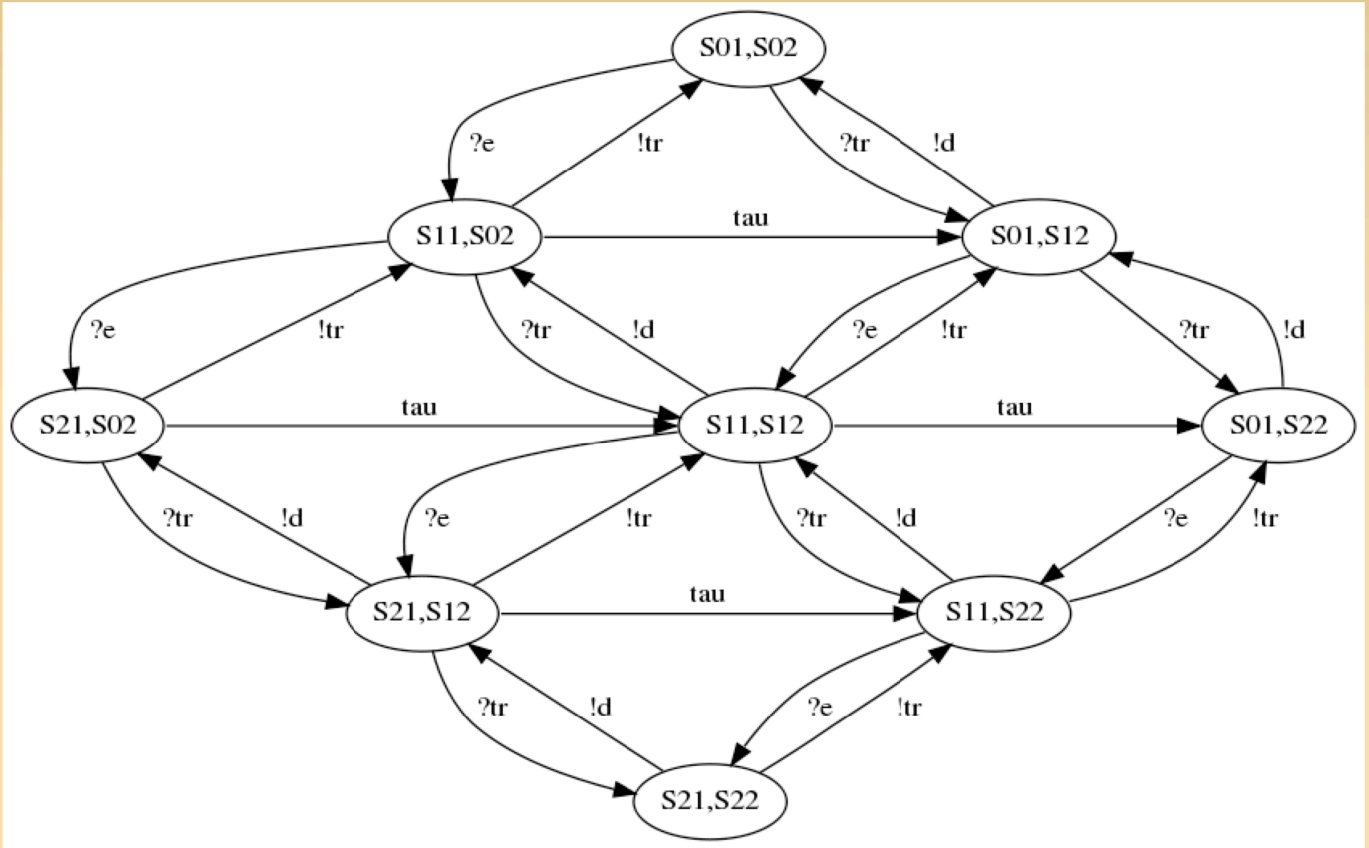
	P|Q	P||Q
système	ouvert	clos
interactions	a/a^c	a/a
		a/ϵ
		ϵ/a



Exemple

pile de taille 4 avec deux piles de taille 2



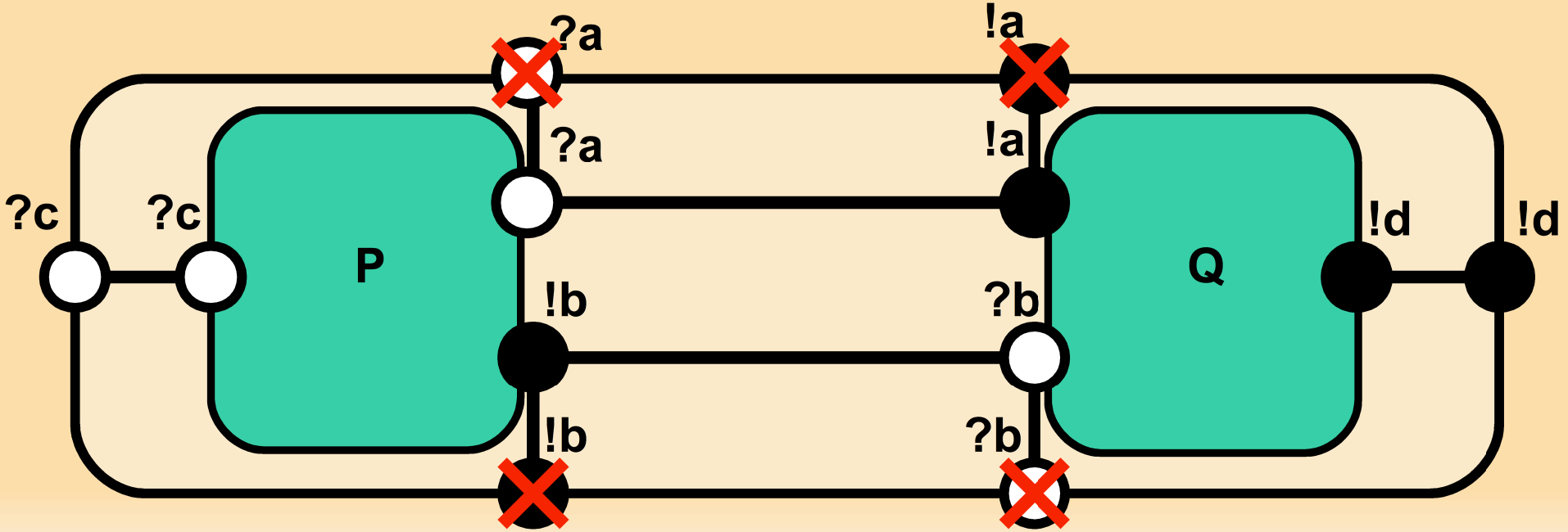


Pile1 |{?e:(?e,ϵ), !tr:(!tr,ϵ), τ:(!tr,?tr),
?tr:(ϵ,?tr), !d:(ϵ,!d)}| Pile2

Pile1 | Pile2

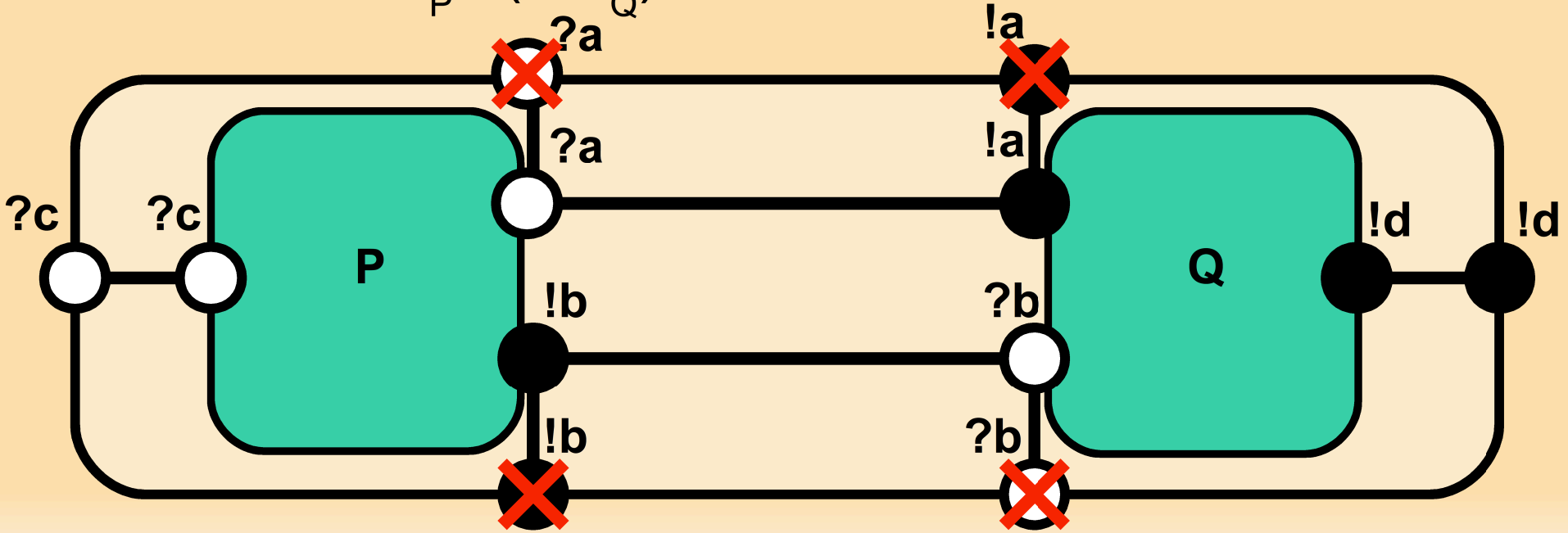
Sémantique syst. communicant

- comment faire si on veut forcer à ce que certaines communications soient internes ?



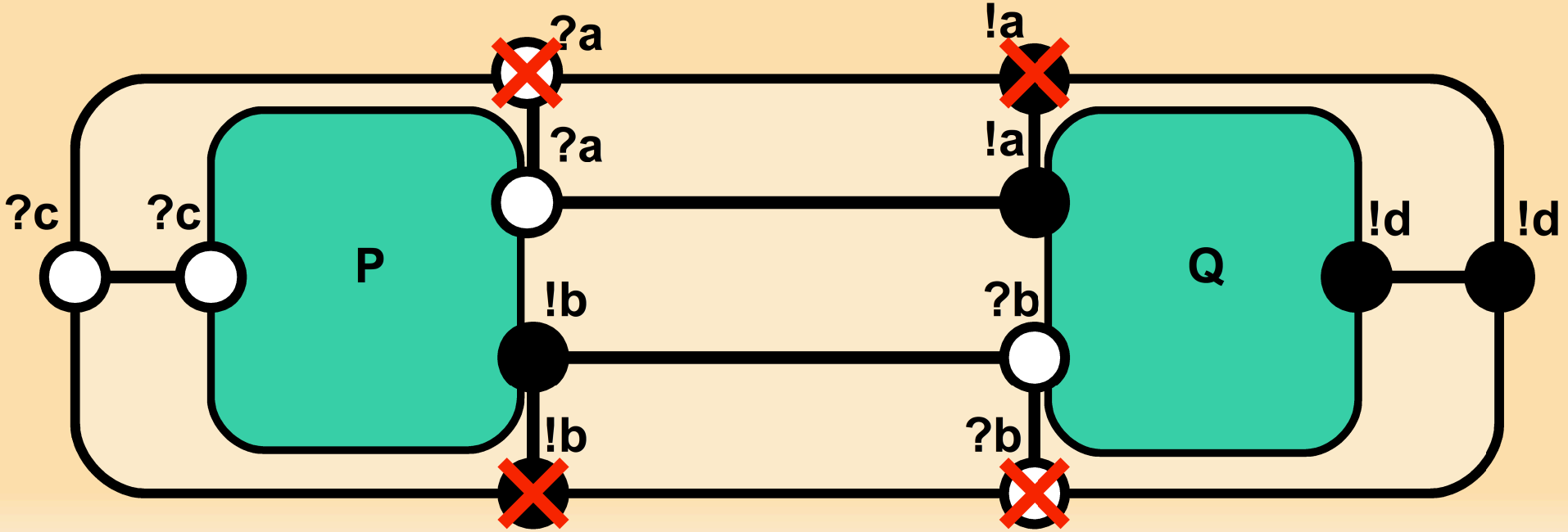
Sémantique syst. communicant

- S=P|V|Q avec
  - $\forall\ a \in\ Act_P \setminus (Act_Q)^c$, $a:<a,\epsilon> \in\ V$
  - $\forall\ a \in\ Act_Q \setminus (Act_P)^c$, $a:<\epsilon,a> \in\ V$
  - $\forall\ a \in\ Act_P \cap (Act_Q)^c$, $\tau:<a,a^c> \in V$



Sémantique syst. communicant

- S=P|V|Q avec [...]
  → lien avec P|Q ?

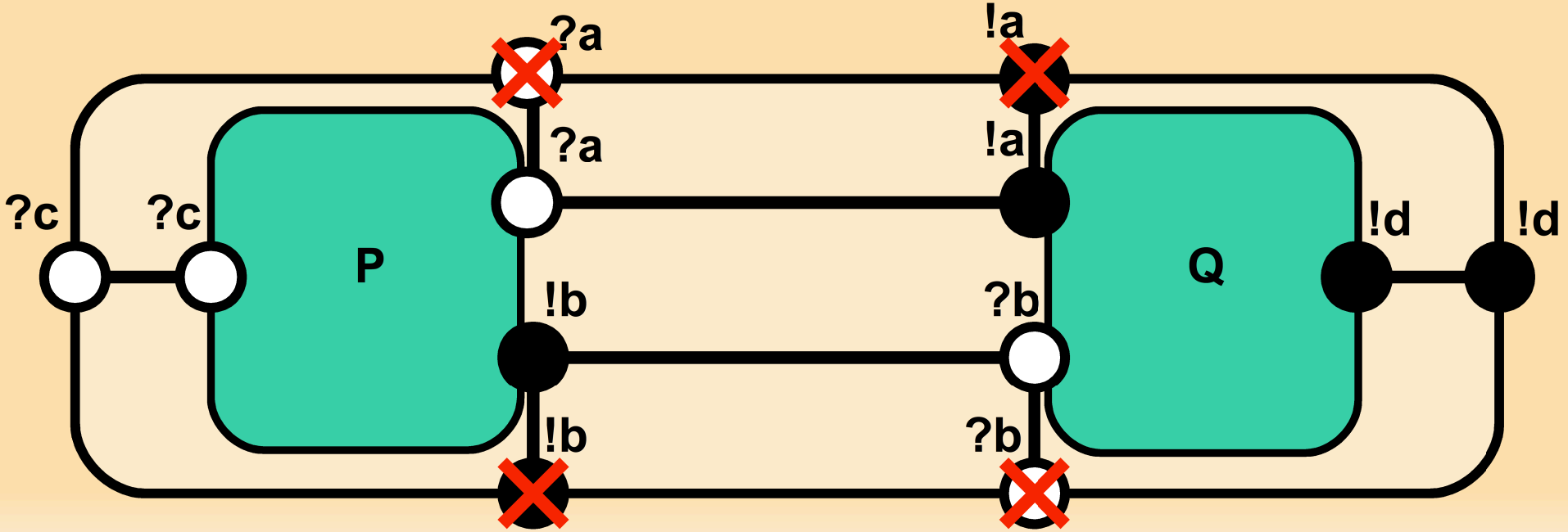


Restriction

- la restriction interdit certaines transitions
  L\B avec $B \subseteq A$
  L\B = (A\B,S,I,F,T\B)
  où $T\backslash B = \{(s,a,s') \mid (s,a,s') \in T \wedge \neg(a \in B \cup B^c)\}$
- ! c'est différent du masquage
  le masquage masque (τ) des transitions
  la restriction interdit des transitions
- on peut aussi avoir la restriction sur les STE non communicants
  $\neg(a \in B)$ au lieu de $\neg(a \in B \cup B^c)$
- dérivé : autorisation, L#B = L\{A\B}

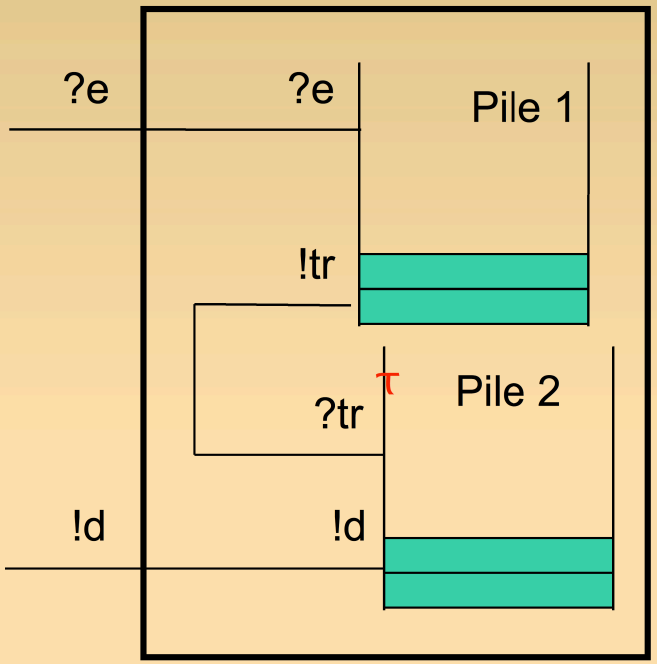
Sémantique syst. communicant

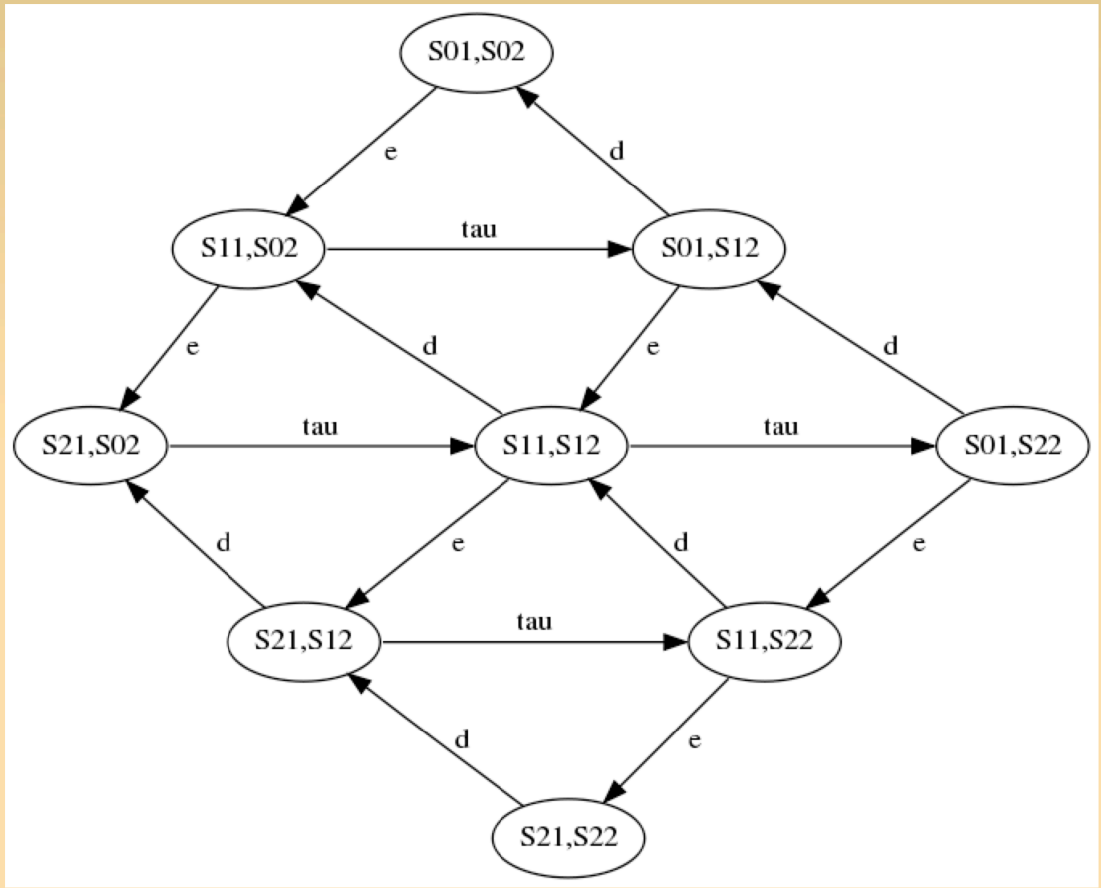
- S=P|V|Q avec [...]
  → $(P \mid Q) \setminus \{Act_P \cap (Act_Q)^c\}$
- ici : (P | Q) \ {?a,?b} ou (P | Q) \ {a,b} (notation alt.)



Exemple

pile de taille 4 avec deux piles de taille 2



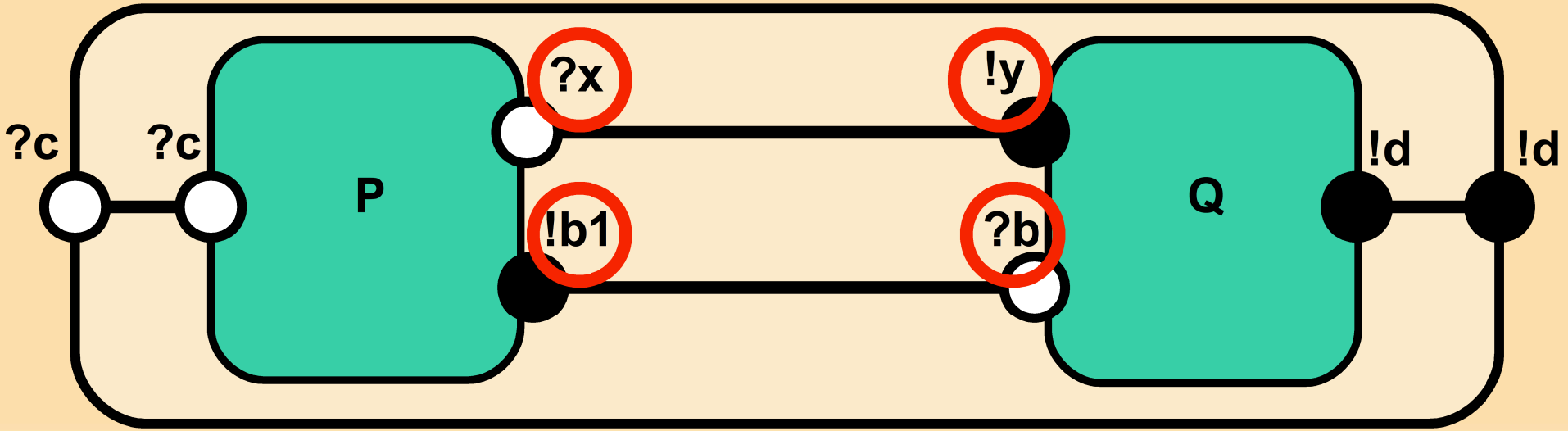


Pile1 |{?e:(?e,ϵ), τ:(!tr,?tr), !d:(ϵ,!d)}| Pile2

(Pile1 | Pile2) \ {tr}

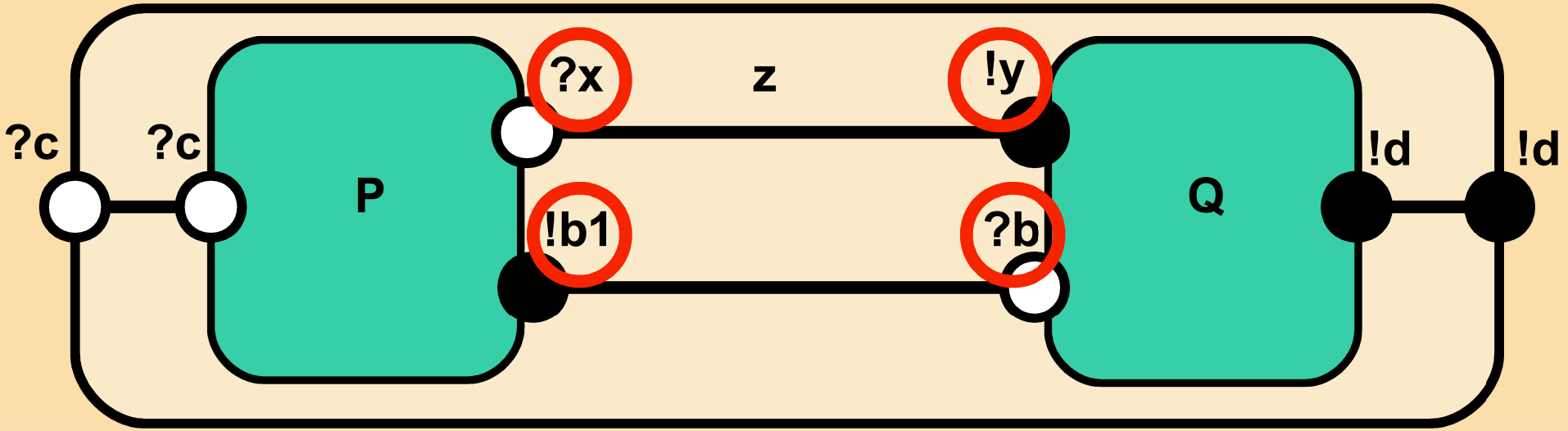
Sémantique syst. communicant

- comment faire si certaines communications font correspondre des actions de noms différents ?



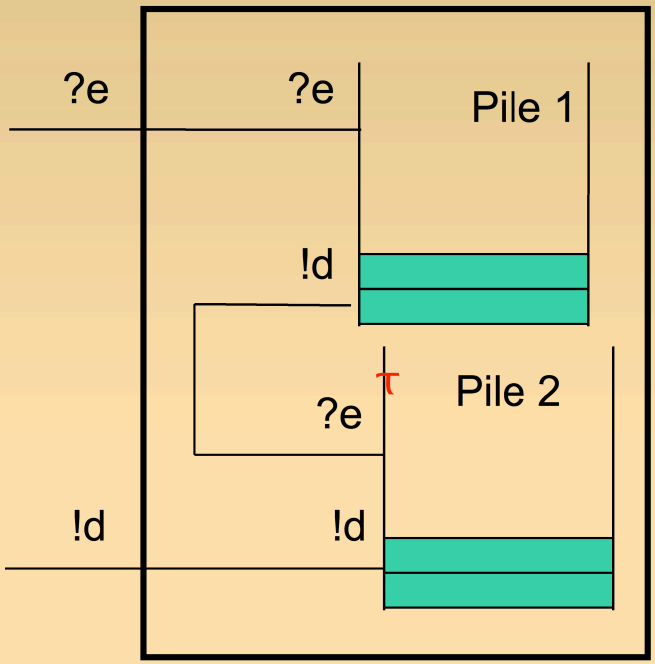
Sémantique syst. communicant

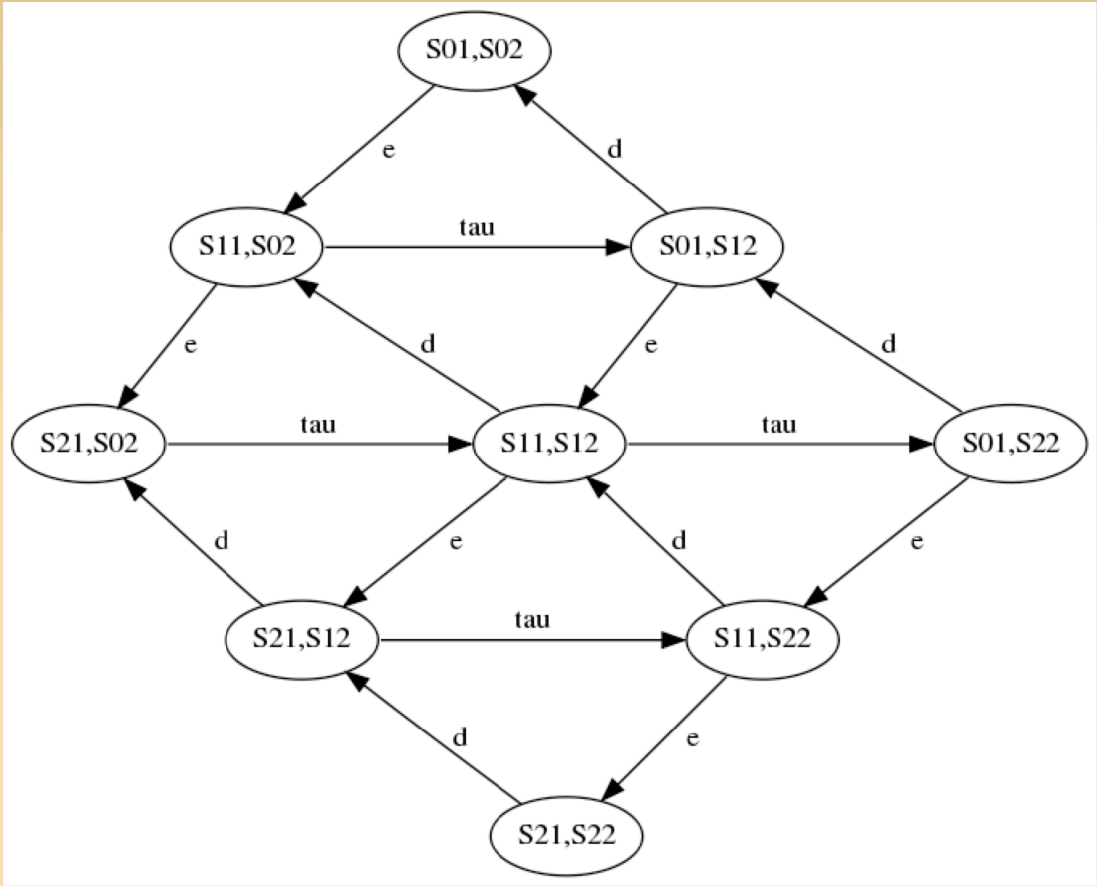
- on utilise le renommage
  - soit en choisissant l'un des noms
  - soit en en introduisant un nouveau
  - en pensant aux renommages/masquages/restrictions au-dessus
- S = (P[z/x,b/b1] | Q[z/y]) \ {z,b}



Exemple

pile de taille 4 avec deux piles de taille 2





Pile |{e:(e,ϵ), τ:(d,e), sortir:(ϵ,d)}| Pile

(Pile[tr/d] | Pile[tr/e]) \ {tr}

Partie 1 – Modèle comportemental comparaisons entre systèmes

- Partie 1 – modèle comportemental de système
  - Système de Transitions Etiquetées (STE / *LTS*)
  - sémantique STE de la concurrence synch./asynch.
  - STE communicants
  - autour de la comparaison de STE
- Partie 2 – ”langages” de processus
- Partie 3 – propriétés

Le point et les objectifs

- le comportement d'un système simple (séquentiel) peut être modélisé par un STE
- le comportement d'un système composite (concurrent) peut être modélisé par une architecture et le produit de STE
- comment savoir si:
  - un système correspond à un besoin ?
  - un système peut en remplacer un autre ?

[Intra]-STE vs. [Inter]-STE

Dans ce qui suit, nous allons nous interesser :

- [INTRA] soit aux propriétés d'un STE
  note: on reverra ça (mais autrement) dans la partie 2
- [INTER] soit à des relations entre plusieurs STE (préordre, equivalence, ...)
- les deux sont liés car on a besoin de réfléchir à [INTRA] pour travailler sur [INTER]
- des exemples seront faits au fur et à mesure au tableau