Modélisation de systèmes

Introduction à la Vérification - SIN6U22

Jean-Marc Talbot

Université d'Aix-Marseille

2021-22

Vérification

Objectifs : s'assurer de la fiabilité des systèmes informatiques (logiciels, systèmes contrôlés informatiquement, systèmes électroniques numériques, ...)

... en s'appuyant sur des outils formels (mathématiques) : méthodes formelles ($\neq$ tests empiriques)

Vérification *automatique* : utiliser (autant que possible) des outils informatiques pour réaliser la vérification

Vérification

Objectifs : s'assurer de la fiabilité des systèmes informatiques (logiciels, systèmes contrôlés informatiquement, systèmes électroniques numériques, ...)

... en s'appuyant sur des outils formels (mathématiques) : méthodes formelles ($\neq$ tests empiriques)

Vérification *automatique* : utiliser (autant que possible) des outils informatiques pour réaliser la vérification

Gamme de méthodes : plus ou moins "efficace", plus ou moins automatique

- typage
- analyse statique
- model-checking
- preuve de programmes

Modélisation des systèmes informatiques

Dans ce cours, nous allons nous intéresser au *model-checking*

Vérifier *automatiquement* des propriétés sur un système (ou une abstraction d'un système) informatique représenté par un système de transitions.

Système de transitions :

- des états
- des transitions entre les états déclenchés par des évènements, signaux d'entrée, et pouvant produire des actions/commandes ou des signaux de sortie

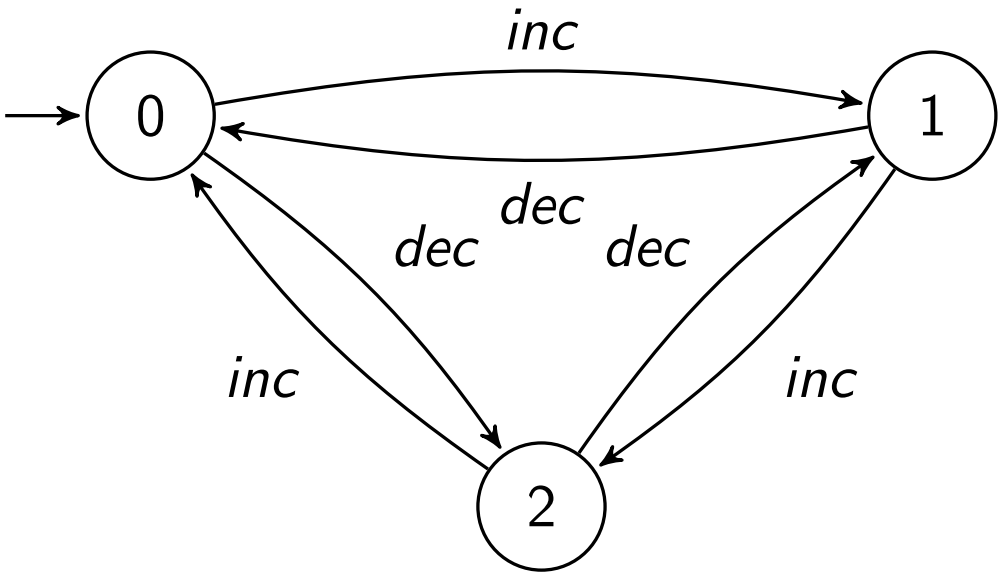
Par exemple, Machine de Mealy

... mais pas nécessairement déterminisme (notamment du fait de l'abstraction).

Pour commencer, le nombre d'états sera fini

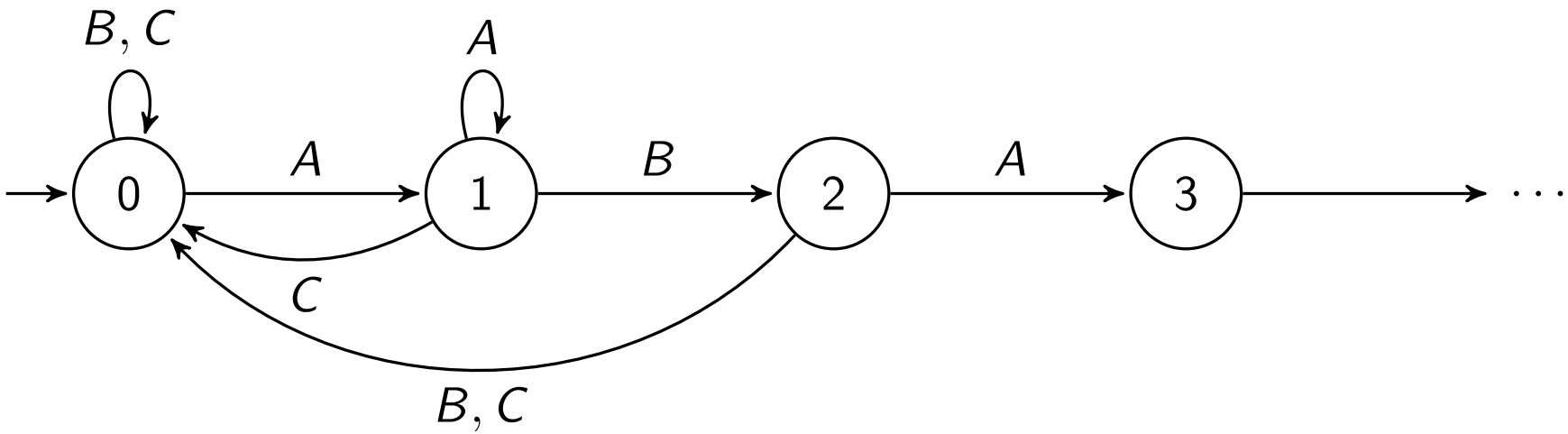
Systèmes d'états finis : un exemple

Un compteur modulo 3



Un autre exemple : le digicode

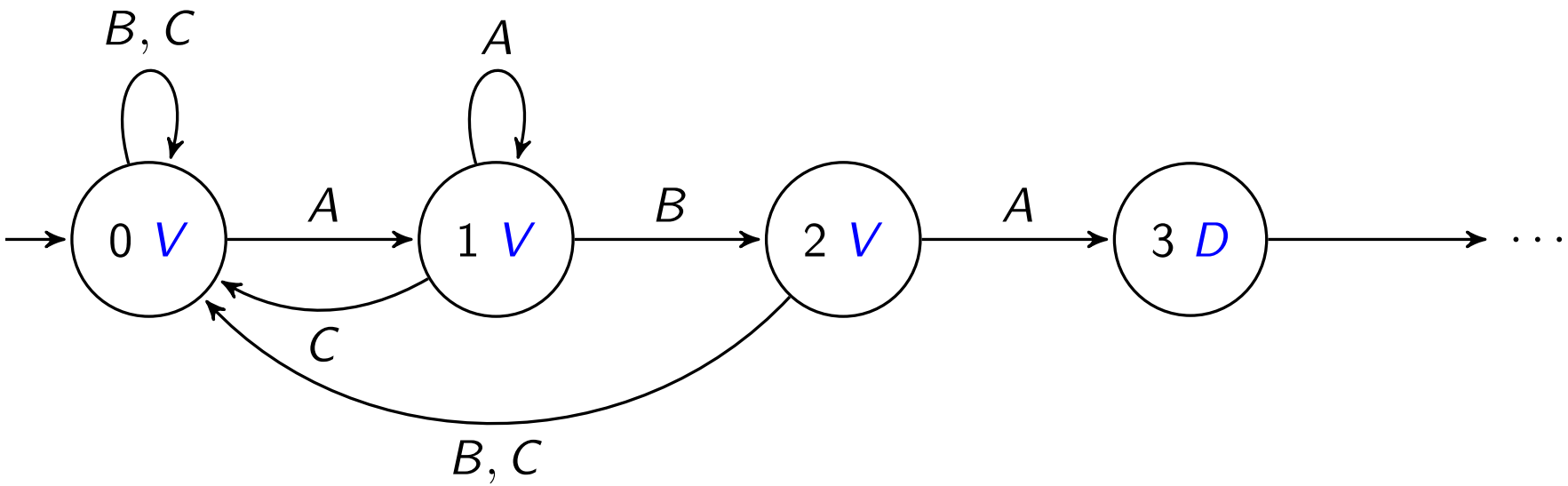
Clavier à 3 touches : A, B, C



La porte est verrouillée dans les états 0,1 et 2 et déverrouillée dans l'état 3.

Un autre exemple : le digicode

Clavier à 3 touches : A, B, C



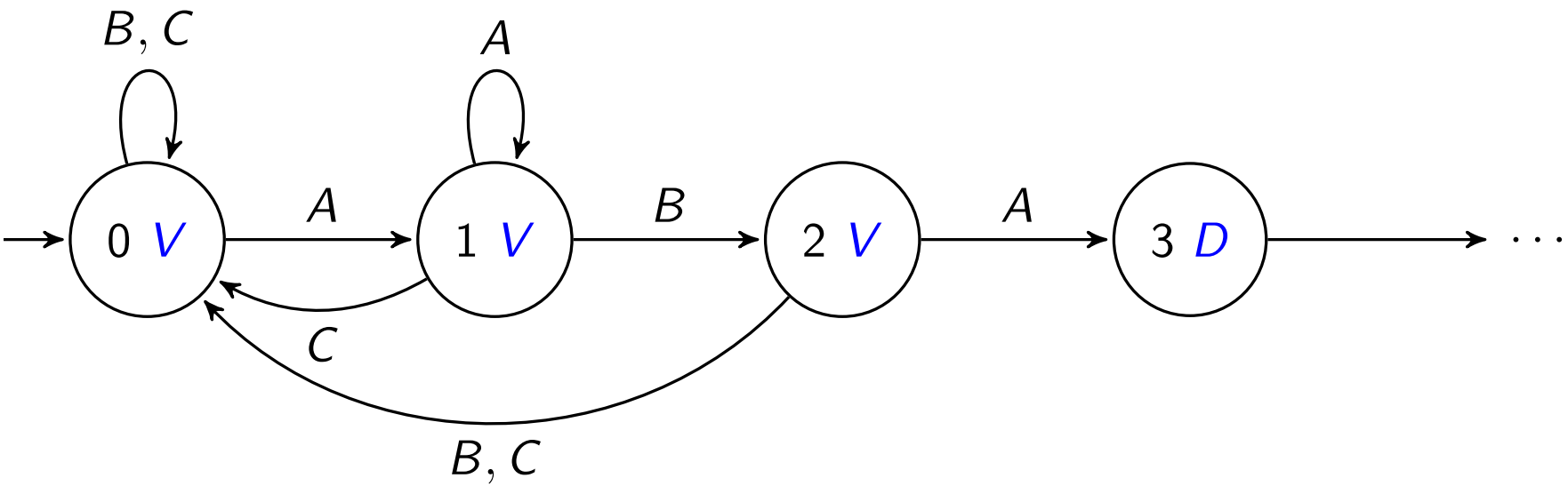
Propriétés associées aux états

V : verrouillé

D : déverrouillé

Un autre exemple : le digicode

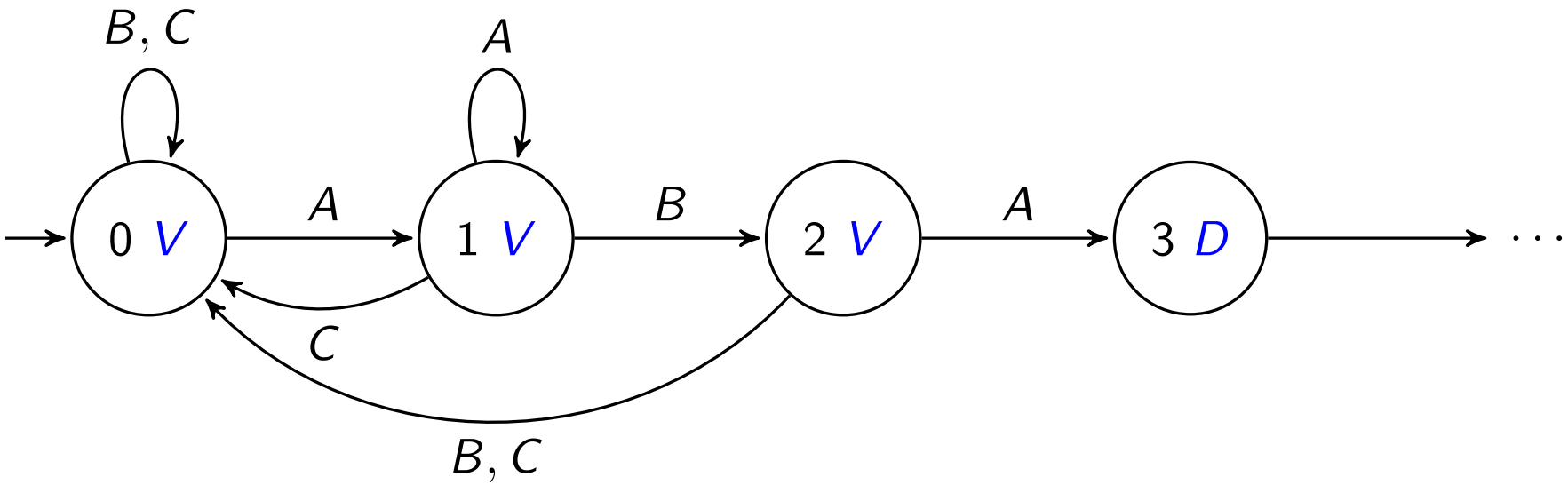
Clavier à 3 touches : A, B, C



Propriétés associées aux états
V : verrouillé
D : déverrouillé

On peut considérer des propriétés supplémentaires

Le digicode : exécution du système

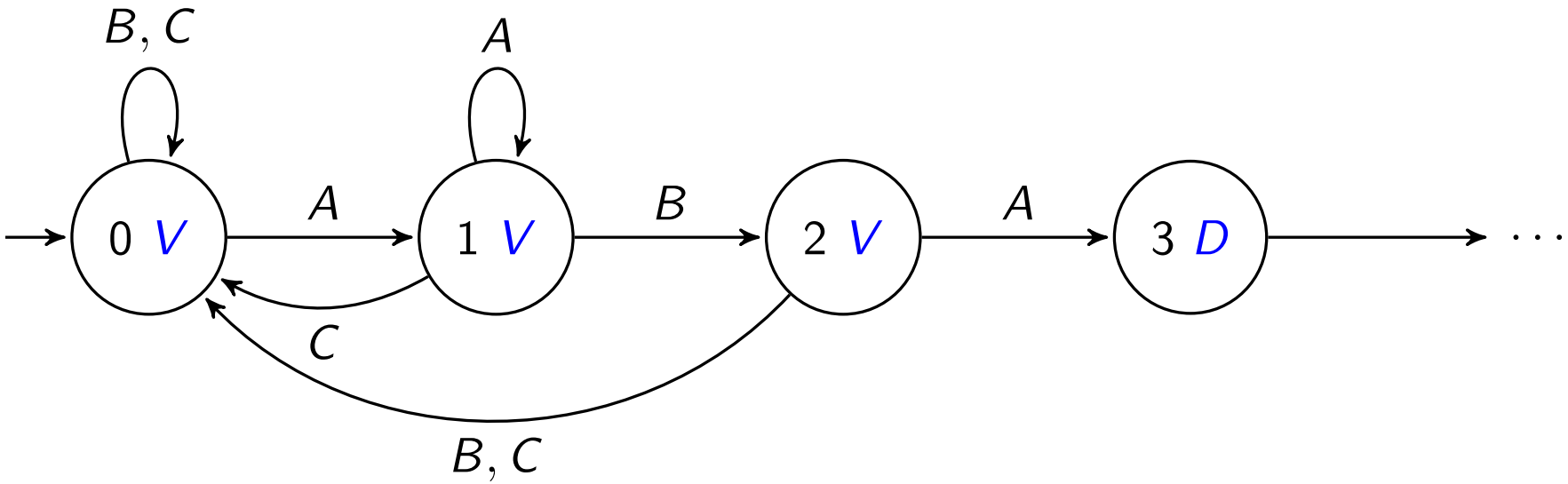


Exécutions possibles du système :

- 0010
- 01123
- 001201123

Exécutions : suite d'états ou suite de transitions

Le digicode : exécution du système



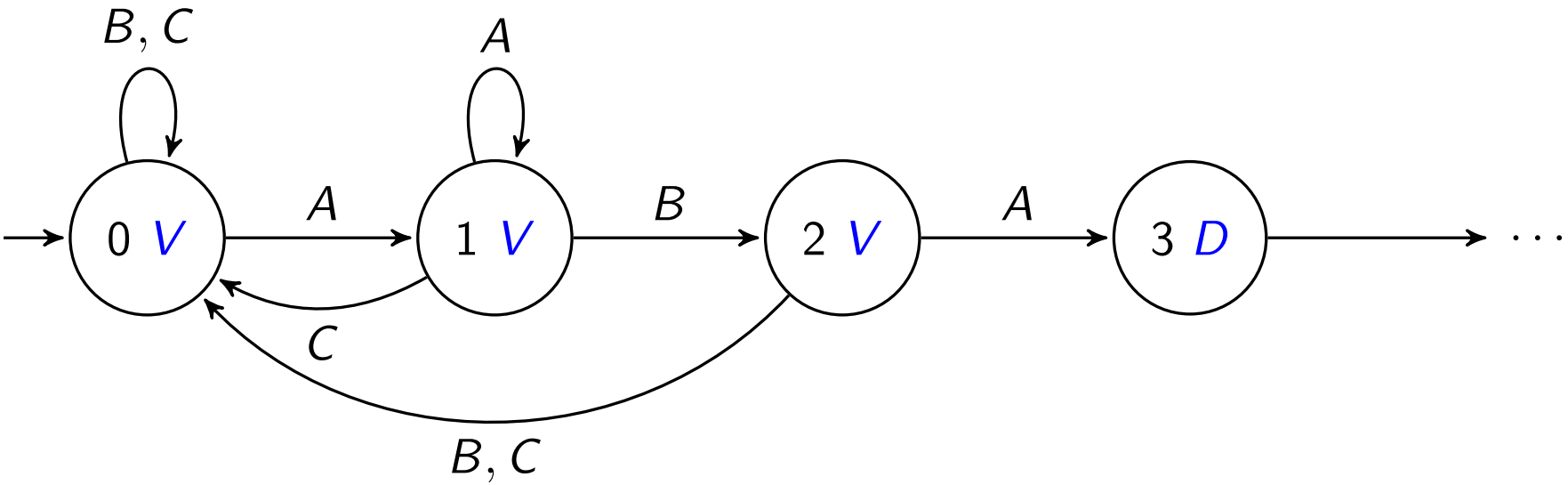
Exécutions possibles du système :

- 0010
- 01123
- 001201123

Exécutions : suite d'états ou suite de transitions

Pour les transitions ?

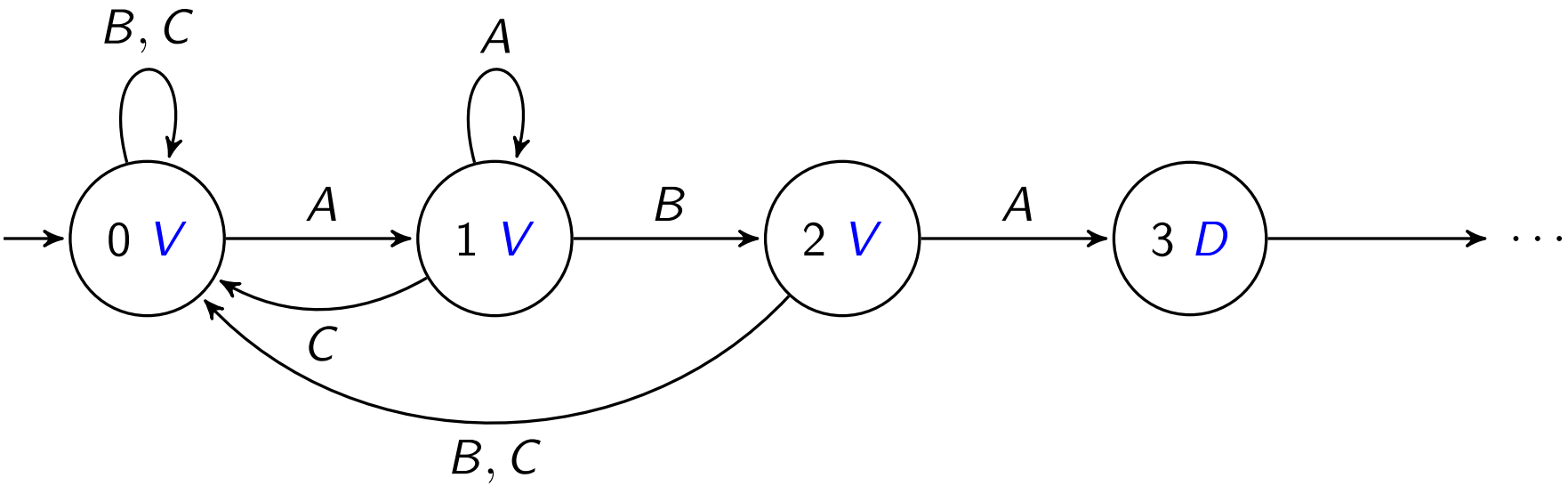
Le digicode : correction



Correction vis-à-vis d'une spécification :

"La porte se déverrouillé si et seulement si les 3 dernières lettres tapées sont A, B et A."

Le digicode : correction

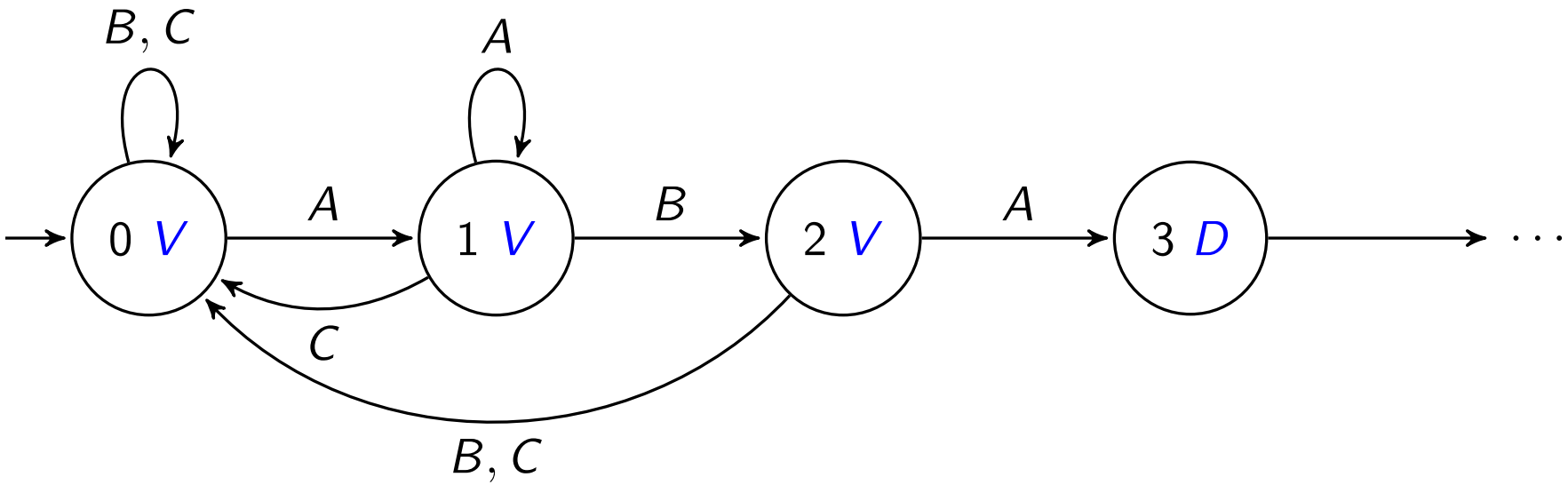


Correction vis-à-vis d'une spécification :

"La porte se déverrouillé si et seulement si les 3 dernières lettres tapées sont A, B et A."

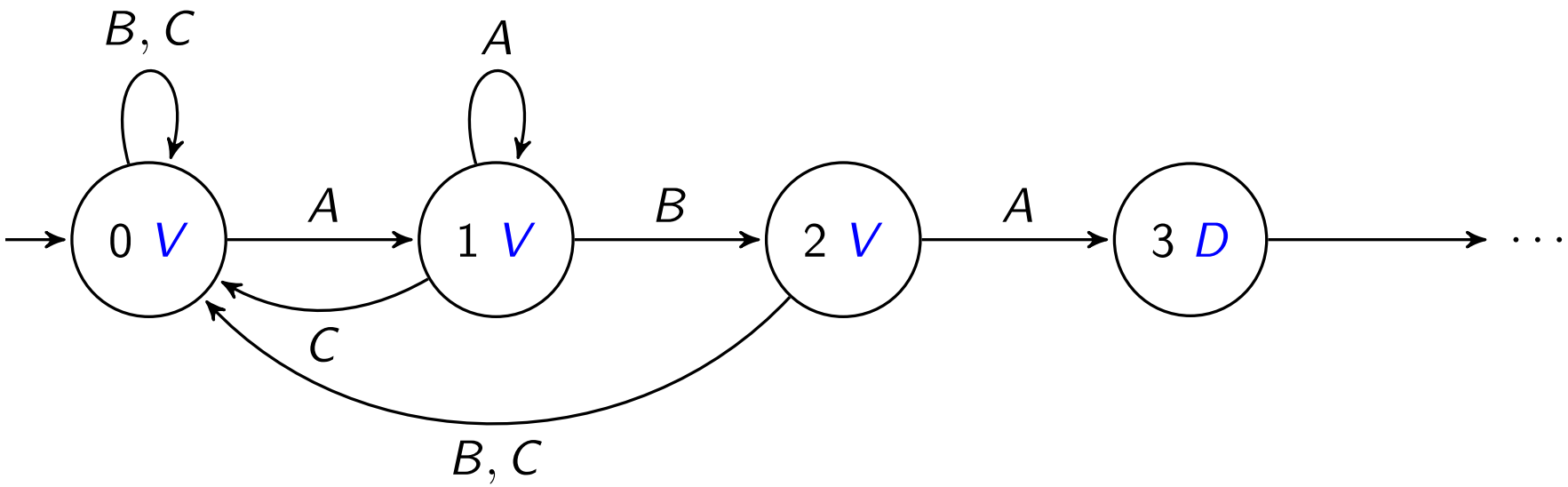
Il faut s'assurer que le système (ou sa modélisation) **vérifie** cette spécification.

Le digicode : autres propriétés



- A : la dernière lettre tapée est A
- B : la dernière lettre tapée est B
- C : la dernière lettre tapée est C
- prec1 : l'état précédent est nécessairement 1
- prec2 : l'état précédent est nécessairement 2

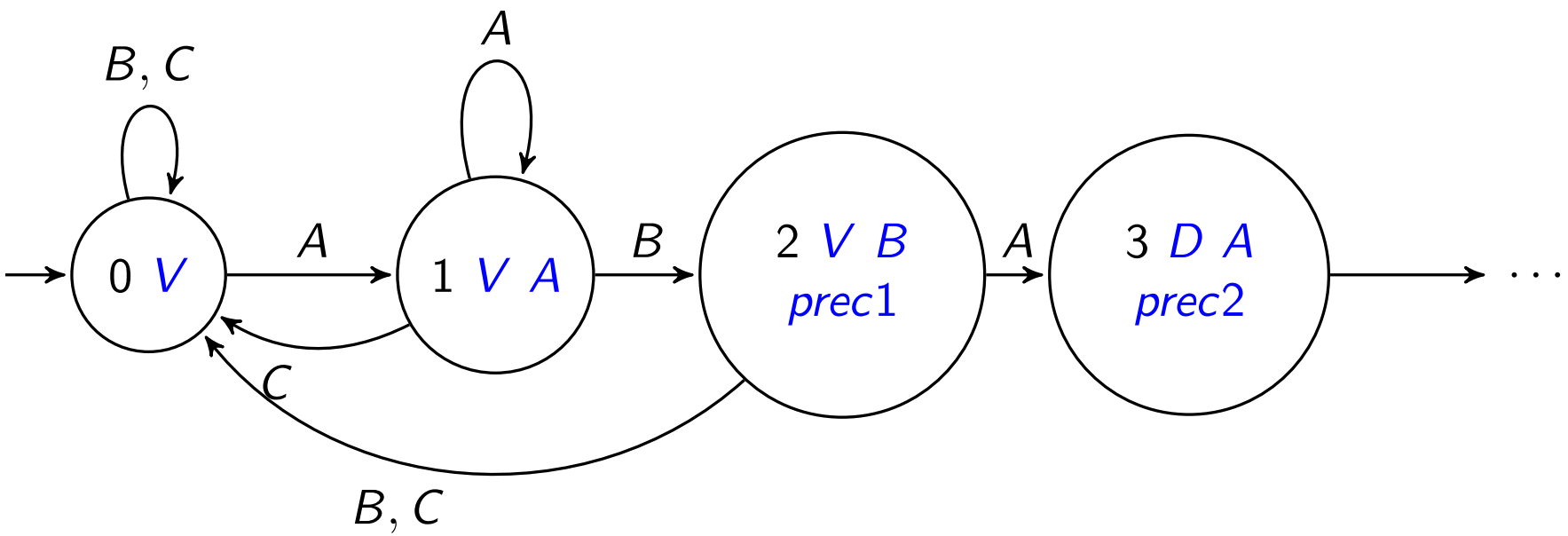
Le digicode : autres propriétés



- A : la dernière lettre tapée est A
- B : la dernière lettre tapée est B
- C : la dernière lettre tapée est C
- prec1 : l'état précédent est nécessairement 1
- prec2 : l'état précédent est nécessairement 2

Quels états vérifient ces propriétés ?

Le digicode : autres propriétés



- A : la dernière lettre tapée est A
- B : la dernière lettre tapée est B
- C : la dernière lettre tapée est C
- prec1 : l'état précédent est nécessairement 1
- prec2 : l'état précédent est nécessairement 2

Le digicode : correction

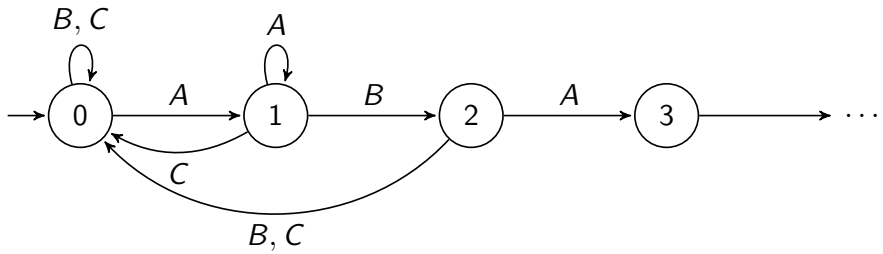
- la porte est fermée dans 0,1 et 2
- la porte est ouverte en 3

- nous sommes dans l'état 3 implique que la dernière lettre tapée est un A et que l'état précédent est l'état 2.
- nous sommes dans l'état 2 implique que la dernière lettre tapée est un B et que l'état précédent est l'état 1.
- pour être dans l'état 1, il faut avoir taper un A.

Donc ABA ouvre bien la porte.

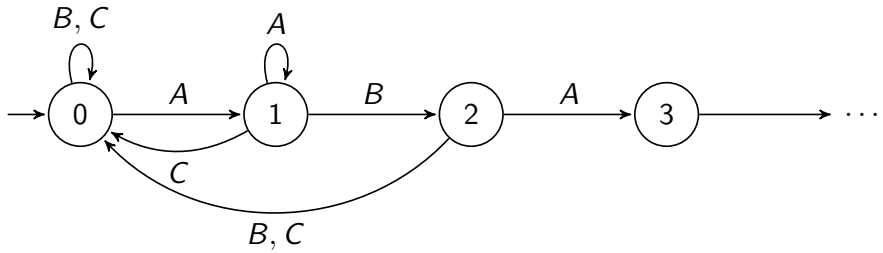
Sans lire ABA, on n'atteint pas l'état 3.

Le digicode : arbre d'exécution



Arbre infini mais régulier

Le digicode : arbre d'exécution



Arbre infini mais régulier

A quelle branche correspond l'exécution 1121 ?

Automate avec propriétés

Definition

Soit *Prop* un ensemble fini de propositions (propriétés atomiques).
Un automate avec propriétés est un quintuplet $\mathcal{A} = (Q, E, T, q_0, \ell)$ tel que

- Q est un ensemble fini d'états
- E est un ensemble fini d'étiquettes
- $T \subseteq Q \times E \times Q$ est l'ensemble des transitions (relation)
- $q_0 \in Q$ est l'état initial de l'automate
- $\ell : E \mapsto \wp(Prop)$ associe à chaque état l'ensemble des propriétés vraies dans cet état.

- Remarque 1 : souvent, il ne considérera pas les étiquettes car on peut se ramener au cas où $T \subseteq Q \times Q$
- Remarque 2 : il n'y a pas d'états finals (mais certains états peuvent avoir une propriété similaire).

Automate avec propriétés : le digicode

En se concentrant à la serrure

- $Q = \{0, 1, 2, 3\}$
- $E = \{A, B, C\}$
- $T = \left\{ \begin{array}{l} (0, A, 1), (0, B, 0), (0, C, 0), \\ (1, A, 1), (1, B, 2), (1, C, 0), \\ (2, A, 3), (2, B, 0), (2, C, 0) \end{array} \right\}$
- $q_0 = 0$
- $\ell : \left\{ \begin{array}{l} 0 \mapsto \{V\} \\ 1 \mapsto \{V, A\} \\ 2 \mapsto \{V, B, prec1\} \\ 3 \mapsto \{D, A, prec2\} \end{array} \right.$

Automate avec propriétés : chemin

Definition

Un *chemin* d'un automate $\mathcal{A}$ est une suite finie ou infinie de transitions (q_i, e_i, q'_i) de $\mathcal{A}$ s'enchainant ($q'_i = q_{i+1}$ pour tout i).

On note un tel chemin $q_1 \xrightarrow{e_1} q_2 \xrightarrow{e_2} q_3 \xrightarrow{e_3} \dots$ (Par exemple, pour le digicode, $2 \xrightarrow{B} 0 \xrightarrow{A} 1 \xrightarrow{A} 2 \xrightarrow{A} 2$).

Definition

La longueur du chemin σ (noté $|\sigma|$) est le nombre de transitions qu'il contient, éventuellement infini : $|\sigma| \in \mathbb{N} \cup \{\omega\}$ (ω pour l'infini).

Pour tout $1 \leq i < |\sigma|$ on note $\sigma(i)$ le $i$ème état de σ, soit q_i, l'état atteint après i transitions.

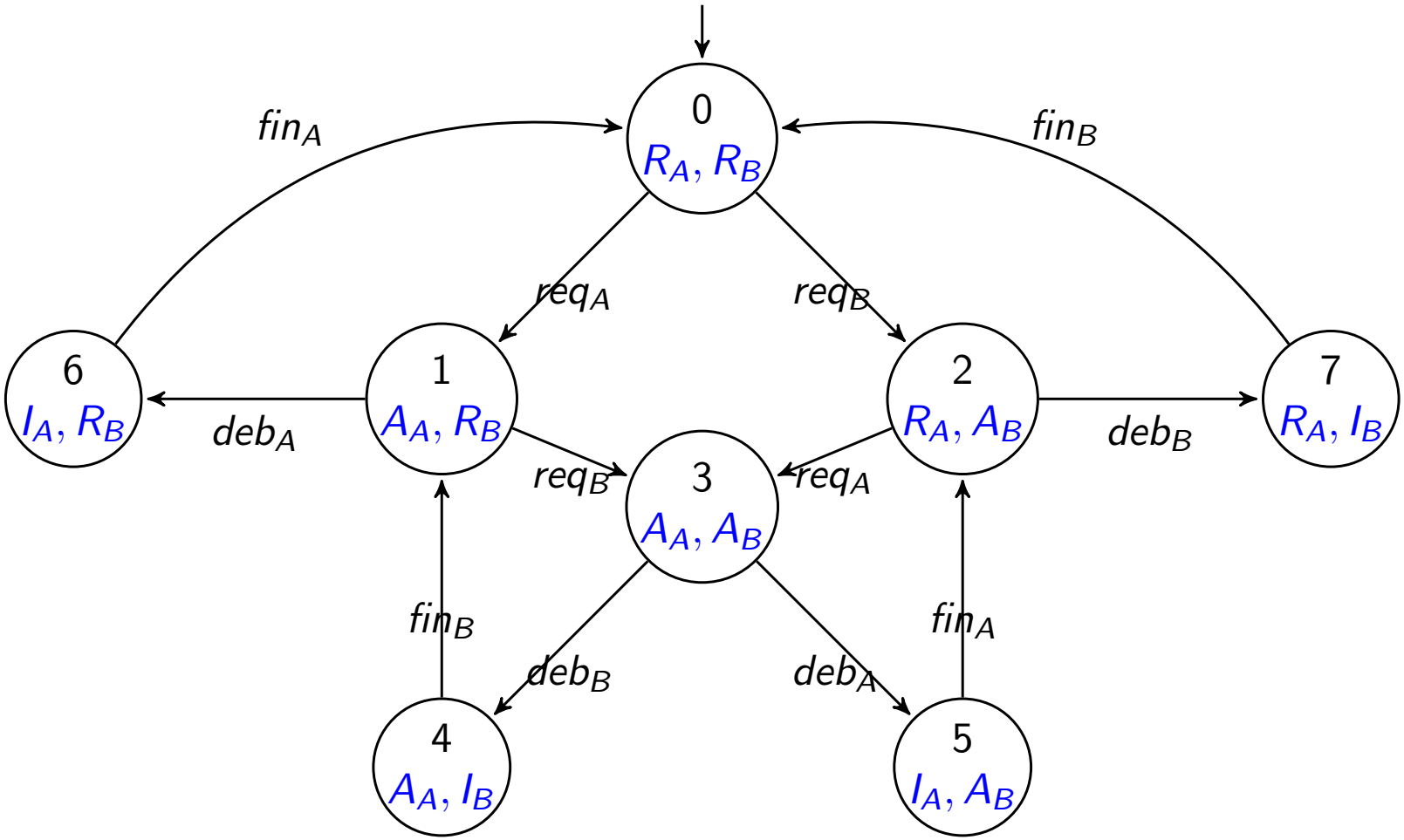
Automate avec propriétés : exécution

Definition

Une *exécution partielle* de $\mathcal{A}$ est une chemin partant de l'état initial q_0.
Par exemple, $0 \xrightarrow{A} 1 \xrightarrow{A} 1 \xrightarrow{B} 2$ est une exécution partielle du digicode.

Une *exécution complète* est une exécution (partielle) maximale : il s'agit soit d'une exécution infinie, soit d'une exécution finie qui ne peut être prolongée (aucune transition ne part du dernier état de cette exécution). Dans ce dernier cas, il s'agit d'une exécution considérée comme terminée ou avec un blocage.

Modélisation d'un gestionnaire d'impression



R : Repos A : Attente I : Imprime

Modélisation d'un gestionnaire d'impression

- I_A : l'imprimante traite la requête de A
- I_B : l'imprimante traite la requête de B
- R_A : A n'a pas fait de requête (il est au repos)
- R_B : B n'a pas fait de requête (il est au repos)
- A_A : A a fait une requête non encore traité (il est en attente du traitement)
- A_B : B a fait une requête non encore traité (il est en attente du traitement)

Modélisation d'un gestionnaire d'impression

- I_A : l'imprimante traite la requête de A
- I_B : l'imprimante traite la requête de B
- R_A : A n'a pas fait de requête (il est au repos)
- R_B : B n'a pas fait de requête (il est au repos)
- A_A : A a fait une requête non encore traité (il est en attente du traitement)
- A_B : B a fait une requête non encore traité (il est en attente du traitement)

Est-ce que toute requête est à terme satisfaite ?

Automates avec variables

Afin de modéliser de manière plus concise les systèmes réels, on considère la possibilité d'ajouter des variables d'états aux automates. Celui-ci pourra tester leur valeur et les modifier.

On retrouve la dualité contrôle et données : les états+transitions forment le contrôle tandis que les variables représentent les données

Automates avec variables : idée

Comme dans n'importe quel programme, on doit

- pouvoir tester les variables et modifier ainsi le flot de contrôle
- pouvoir modifier les variables

Automates avec variables : idée

Comme dans n'importe quel programme, on doit

- pouvoir tester les variables et modifier ainsi le flot de contrôle
- pouvoir modifier les variables

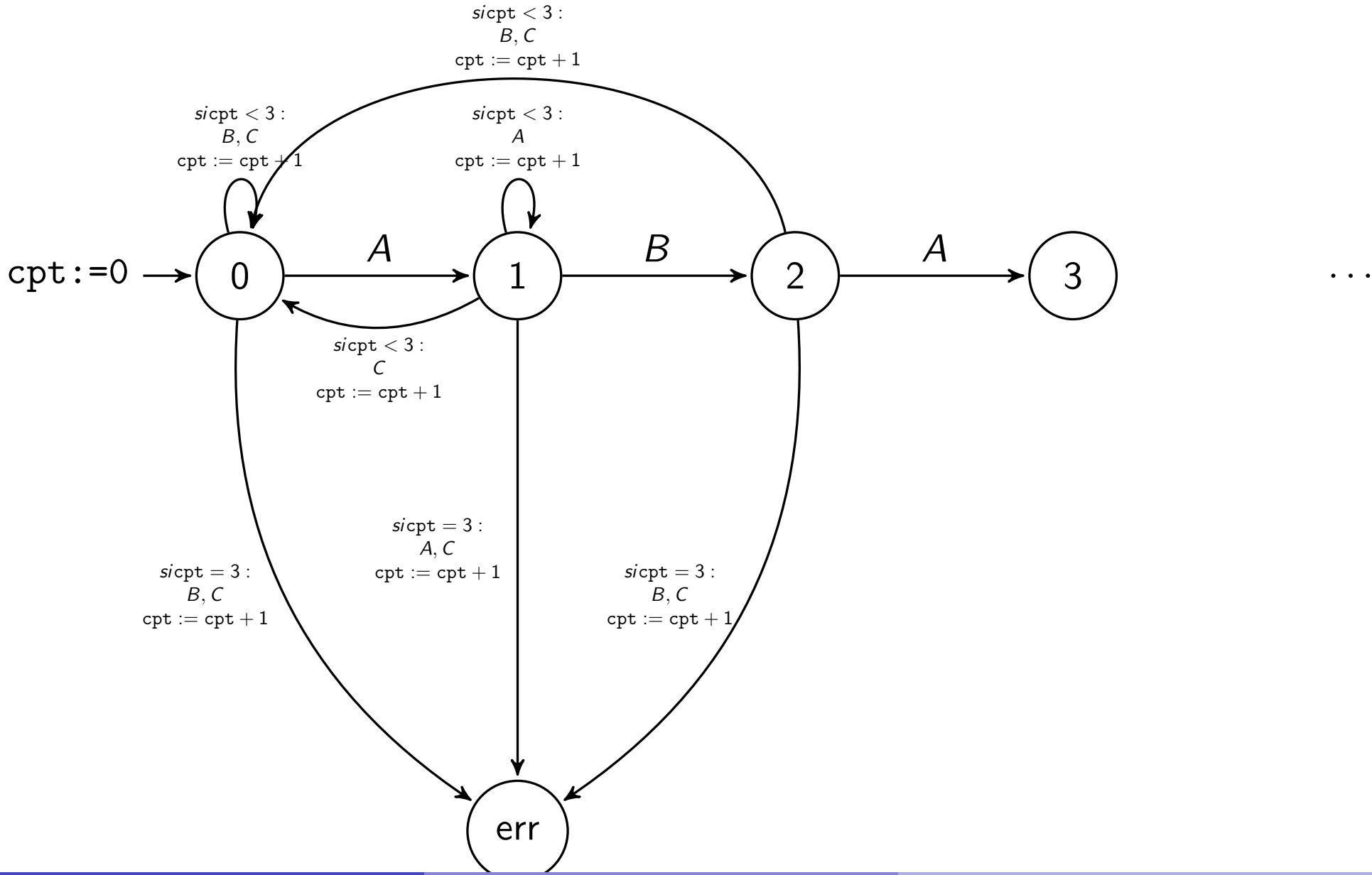
- Gardes : chaque transition est gardée par une condition qui teste les valeurs des variables; la transition est franchissable que si la garde est vraie.
- Affectations : le franchissement d'une transition (outre changer l'état (de contrôle)) modifie la valeur d'une ou plusieurs variables.

Automates avec variables : un exemple

On reconsidère le digicode avec une borne sur le nombre de tentatives (en fait, sur le nombre de touches tapées).

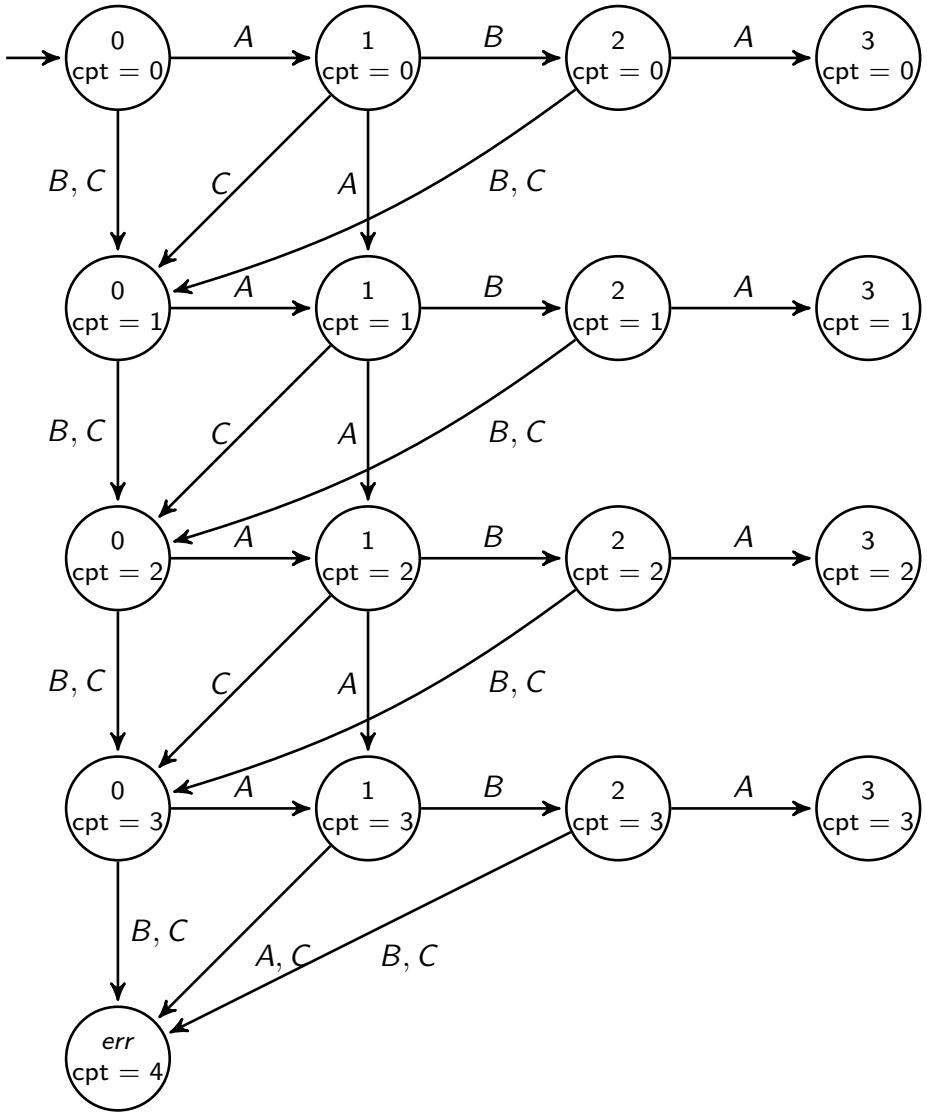
On utilise une variable `cpt` pour compter le nombre de touches tapées, cette variable est initialisée à 0.

Automates avec variables : un exemple



Dépliage des automates avec variables

Les gardes et les affectations disparaissent, les valeurs des variables apparaissent dans les états.



Dépliage des automates avec variables

Pour que l'automate reste fini, il faut que :

- les variables soient en nombre fini
- les variables ne puissent prendre qu'un nombre fini de valeurs différentes.

Définition modulaire de systèmes

Pour modéliser des systèmes réels, on va les définir de manière modulaire comme la combinaison de plusieurs sous-systèmes.

Les sous-systèmes communiquent/interagissent entre eux pour se synchroniser ou échanger des données

Produit cartésien d'automates

Pour rappel, il est possible de faire le *produit cartésien* de plusieurs automates fonctionnant sur le **même alphabet**, pour faire l'intersection des langages par exemple.

Produit cartésien d'automates

Pour rappel, il est possible de faire le *produit cartésien* de plusieurs automates fonctionnant sur le **même alphabet**, pour faire l'intersection des langages par exemple.

Definition

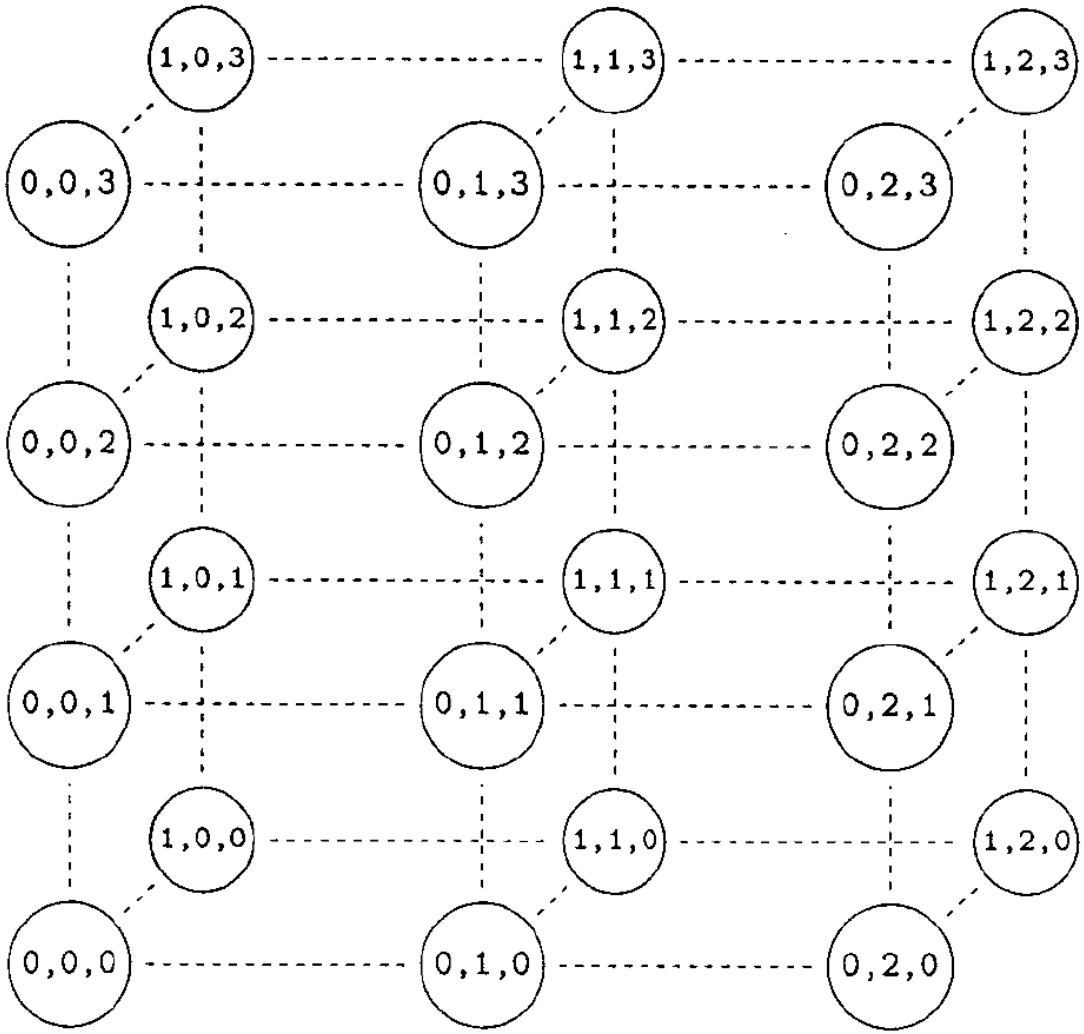
Soit $(\mathcal{A}_i)_{1 \leq i \leq n}$ une famille de n automates, $\mathcal{A}_i = (Q_i, E_i, T_i, q_{0,i}, \ell_i)$
Le produit cartésien $\mathcal{A}_1 \times \ldots \times \mathcal{A}_n$ est l'automate (Q, E, T, q_0, ℓ) avec :

- $Q = Q_1 \times \ldots \times Q_n$
- $E = (E_1 \cup \{\_\}) \times \ldots \times (E_n \cup \{\_\})$
- $T \subseteq$
$$\left\{ (q_1, .., q_n), (e_1, .., e_n), (q'_1, .., q'_n) \mid \forall 1 \leq i \leq n, \begin{array}{l} \text{soit } (q_i, e_i, q'_i) \in T_i, \\ \text{soit } e_i = \_ \text{ et } q_i = q'_i \end{array} \right\}$$
- $q_0 = q_{0,1} \times \ldots \times q_{0,n}$
- $\ell((q_1, \ldots, q_n)) = \bigcup_{1 \leq i \leq n} \ell_i(q_i)$

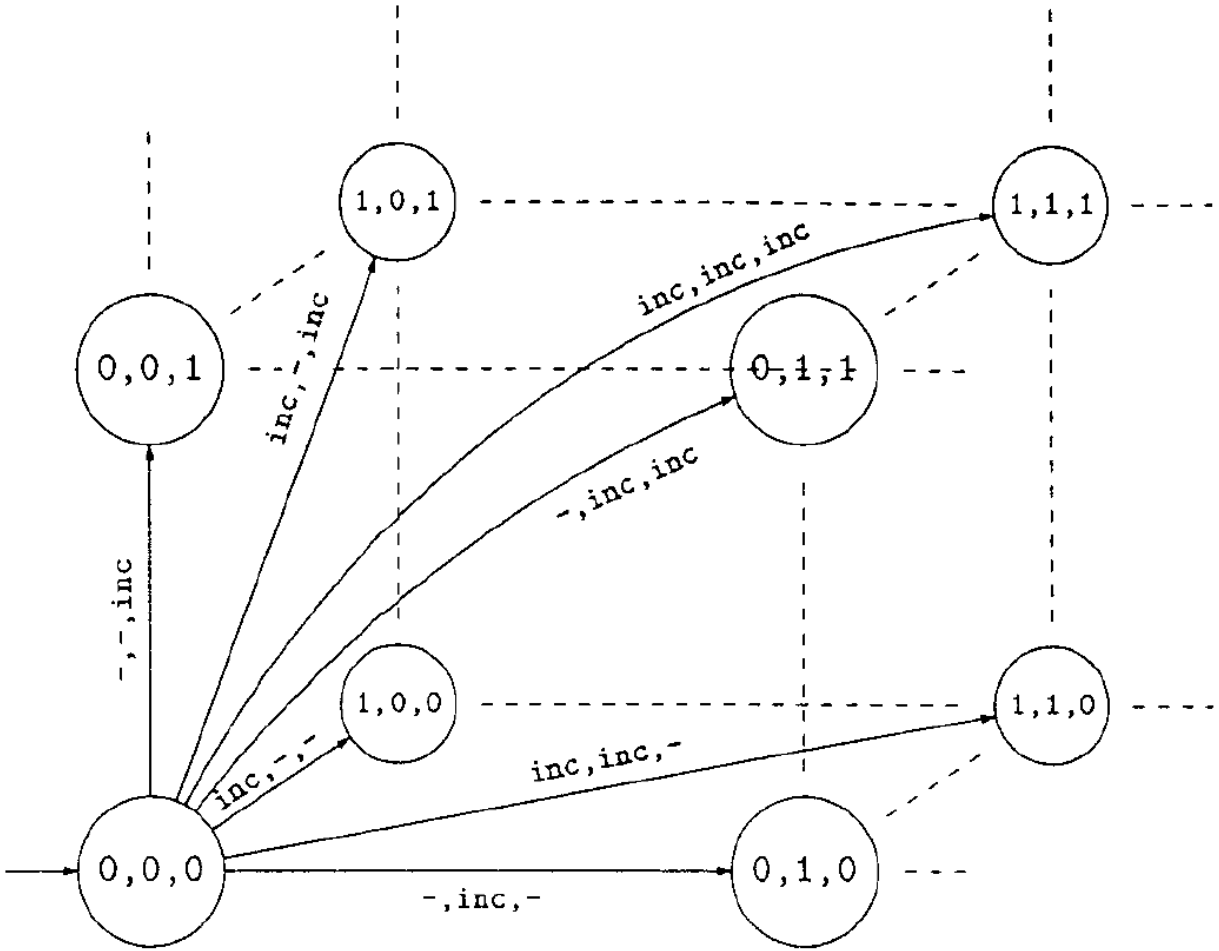
L'action _ permet à une composante de rester dans le même état tandis que les autres "avancent".

Produit cartésien d'automates : un exemple

Produit d'un compteur modulo 2, d'un compteur modulo 3 et d'un compteur modulo 4.



Produit cartésien d'automates : un exemple (II)



Produit synchronisé d'automates

Pour synchroniser les comportements des différentes composantes, on restreint les transitions autorisées : le produit synchronisé est donné par $\mathcal{A} = (Q, E, T, q_0, \ell, \mathit{Sync})$ où (Q, E, T, q_0, ℓ) est le produit cartésien et

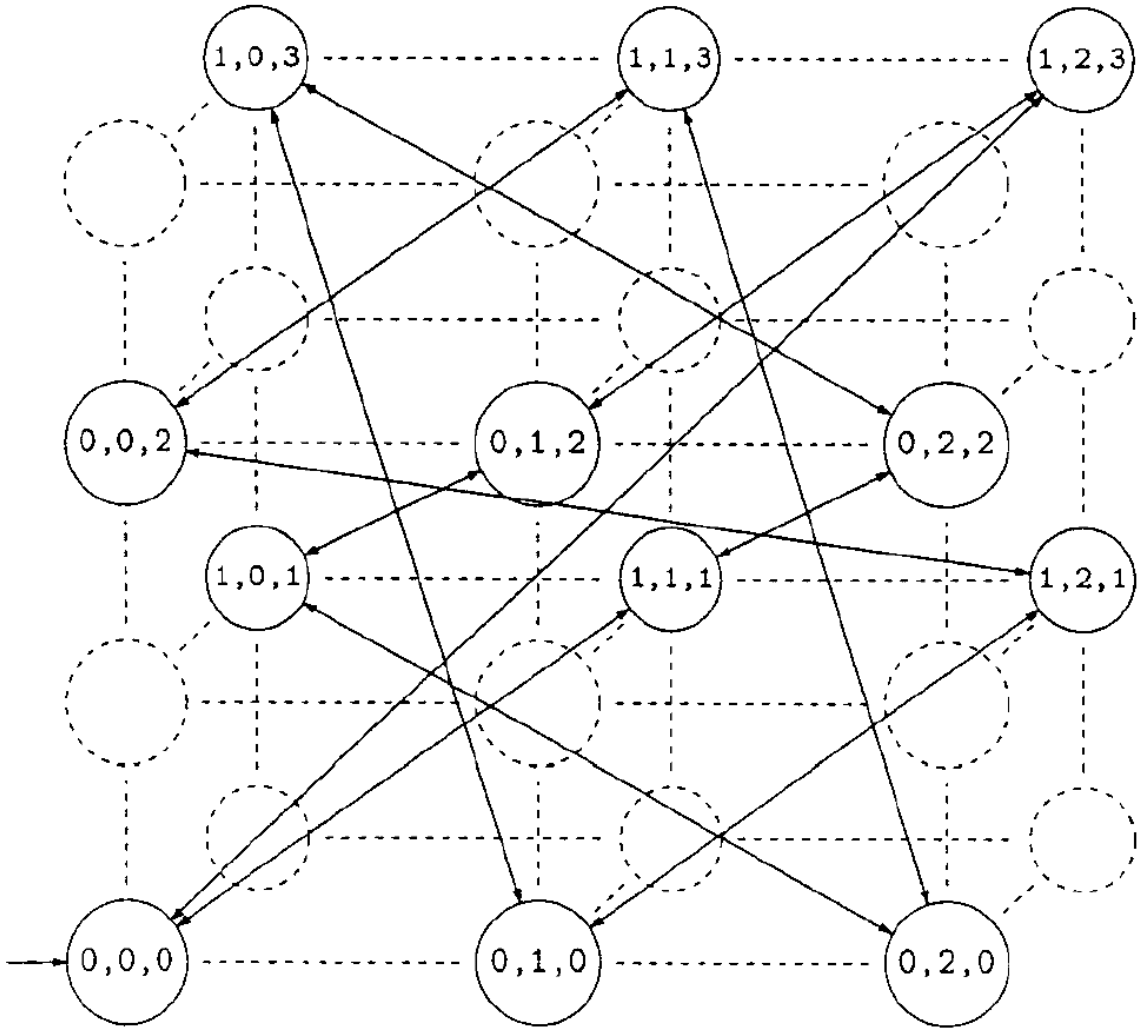
$$\mathit{Sync} \subseteq \Pi_{1 \leq i \leq n}(E_i \cup \{\_\})$$

La transition n'est possible dans l'automate produit que si l'étiquette figure dans *Sync*. Aussi, les étiquettes qui ne sont pas dans *Sync* ne figurent pas dans le produit synchronisé d'automates.

Produit synchronisé d'automates

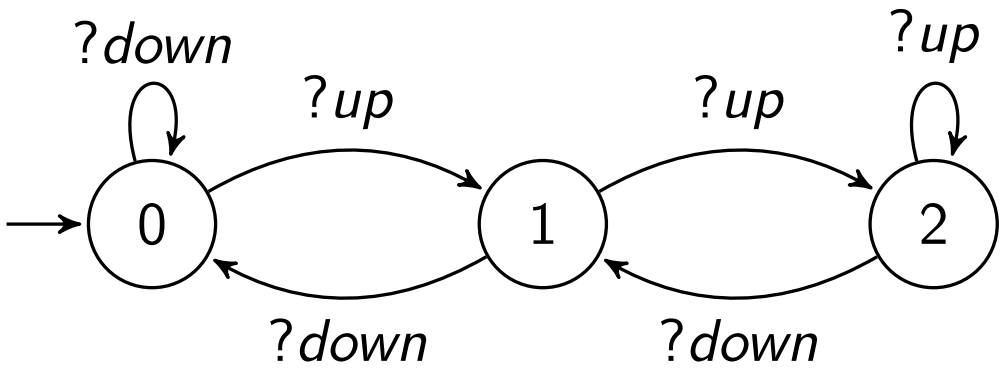
On souhaite synchroniser les trois compteurs en leur imposant la même action, ainsi

$$Sync = \{(inc, inc, inc), (dec, dec, dec)\}$$

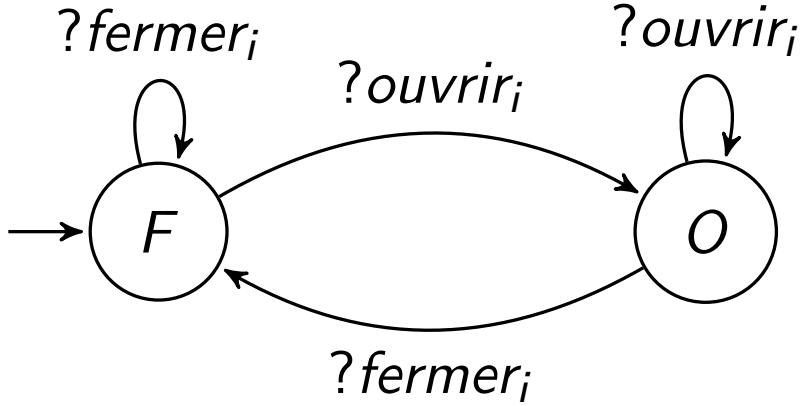


L'exemple de l'ascenseur (I)

La cabine :

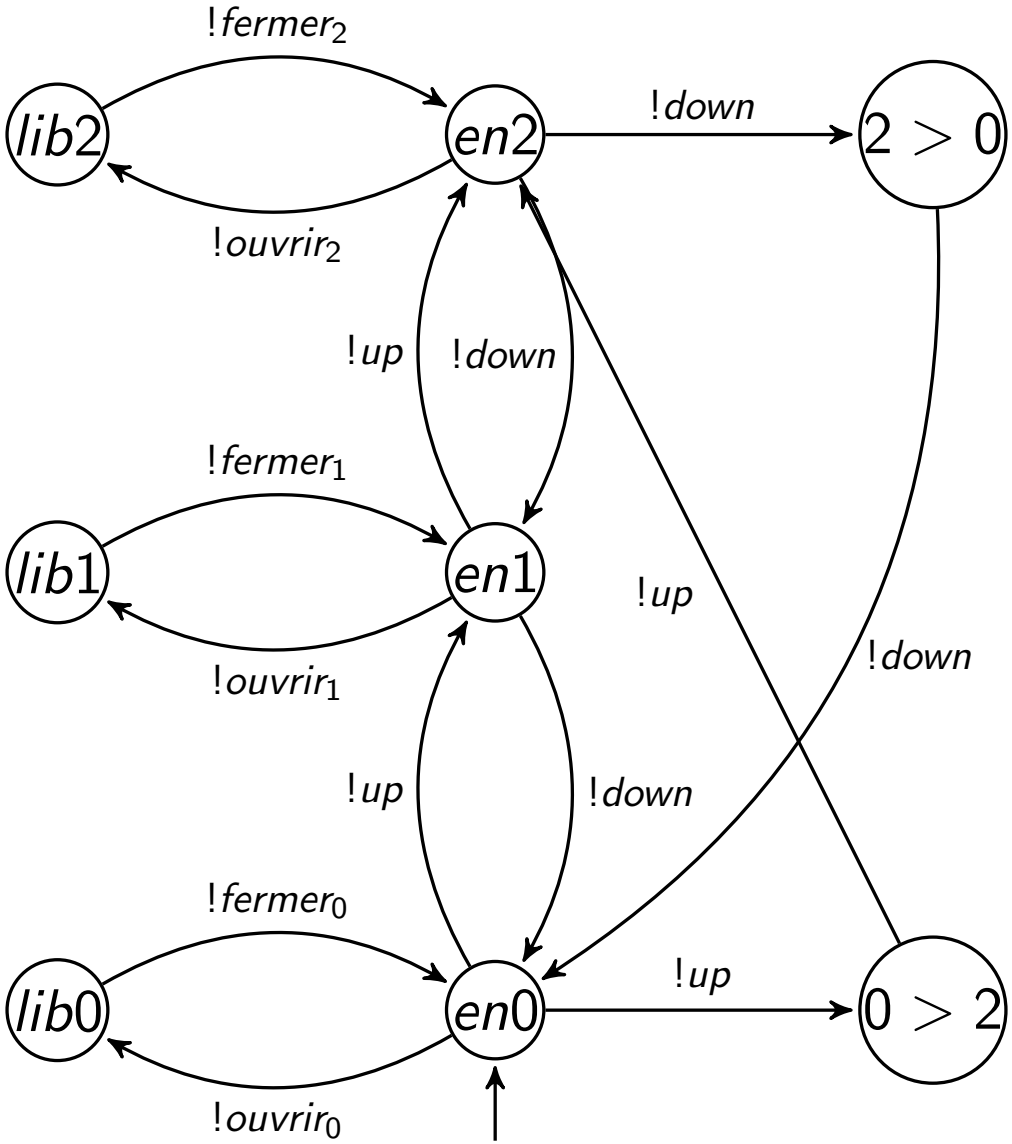


La porte numéro i :



L'exemple de l'ascenseur (II)

Le contrôleur de l'ascenseur



L'exemple de l'ascenseur (III)

La synchronisation est réalisée par

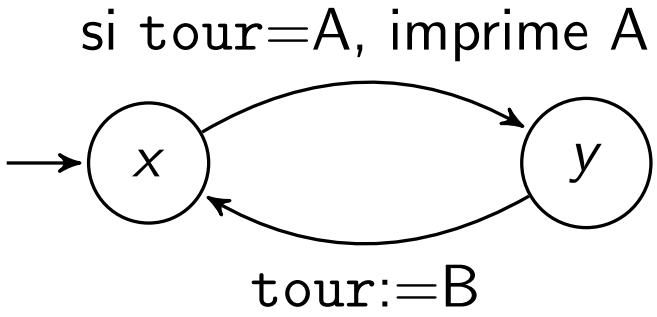
$$Sync = \begin{array}{l} \{(?ouvrir_1, \_, \_, \_, !ouvrir_1), (?fermer_1, \_, \_, \_, !fermer_1), \\ (\_, ?ouvrir_2, \_, \_, !ouvrir_2), (\_, ?fermer_2, \_, \_, !fermer_2), \\ (\_, \_, ?ouvrir_3, \_, !ouvrir_2), (\_, \_, ?fermer_1, \_, !fermer_2), \\ (\_, \_, \_, ?down, !down), (\_, \_, \_, ?up, !up)\} \end{array}$$

Communication par variable partagée

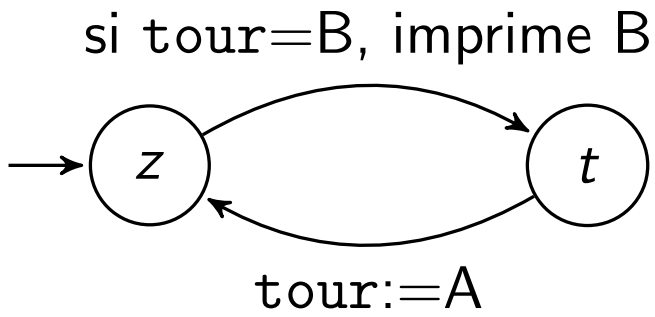
Lorsque les automates possèdent des variables alors ils peuvent échanger des données via ces variables.

La gestion d'une imprimante : une variable `tour` est partagé entre deux utilisateurs A et B

L'utilisateur A

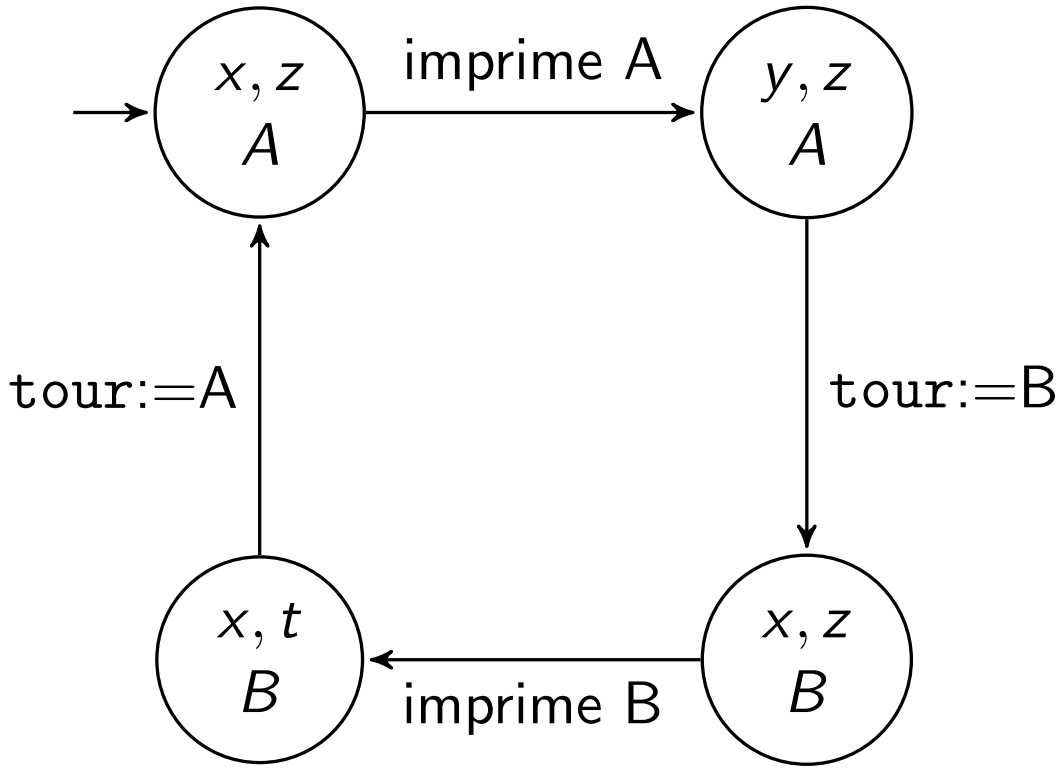


L'utilisateur B



Communication par variable partagée (II)

Produit synchronisé des deux automates et dépliage



L'utilisateur A