# Vérification formelle par model-checking

## Introduction, principes et utilisation du logiciel UPPAAL

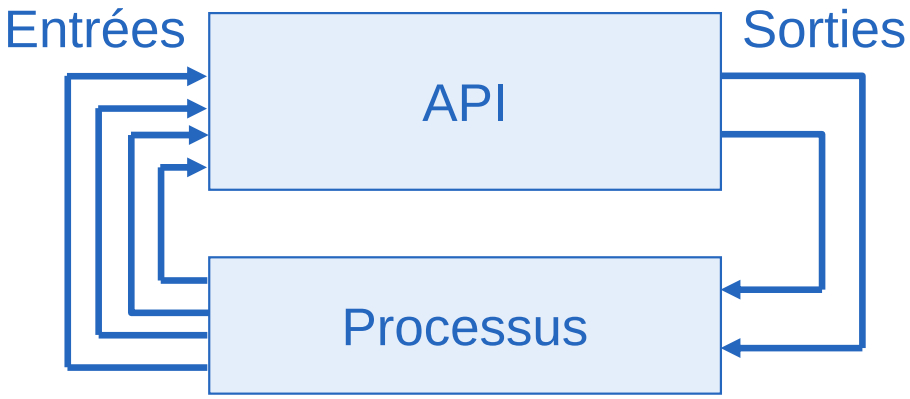
Steve LIMAL doctorant

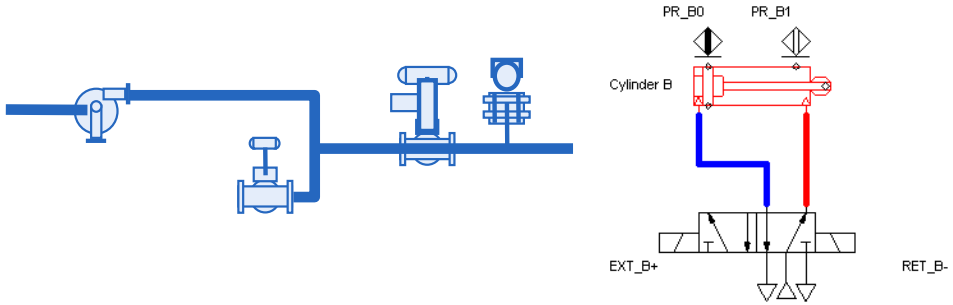
# Planning de la journée

- **Problématique**
- **Principe du model-checking**
  - définitions
  - principe général
  - Modèle de comportement
    - Difficulté de génération
    - application à UPPAAL
  - Modèle de propriété
    - la logique CTL
    - exemple de propriété
- **Traitements de cas**
- **Évaluation**

# Les systèmes automatisés

- **Objectif :**
  - contrôle automatique d'un processus.
- **Processus :**
  - constituants mécaniques, électriques, pneumatiques, électroniques, ...
  - actionneurs.
  - capteurs ou détecteurs.
- **Commande :**
  - Automate Programmable Industriel (API): contrôleur industriel monotache cyclique.
- **Enjeu :**
  - l'état du processus n'est connu de la commande qu'à partir d'informations discrètes (entrée).



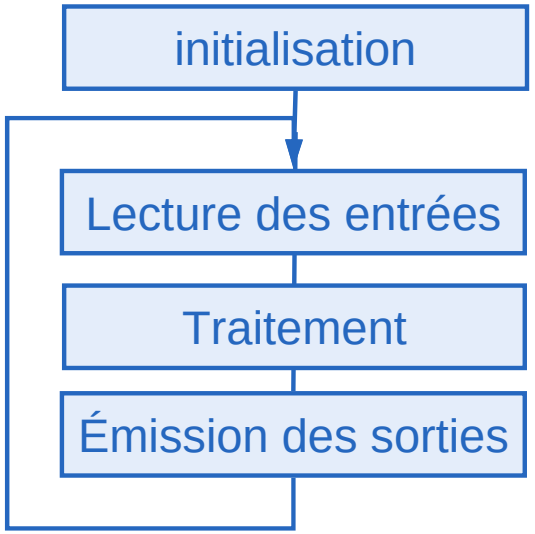



# Les API

- **Caractéristiques**
  - Robuste
  - Stable temporellement
  - Rapide

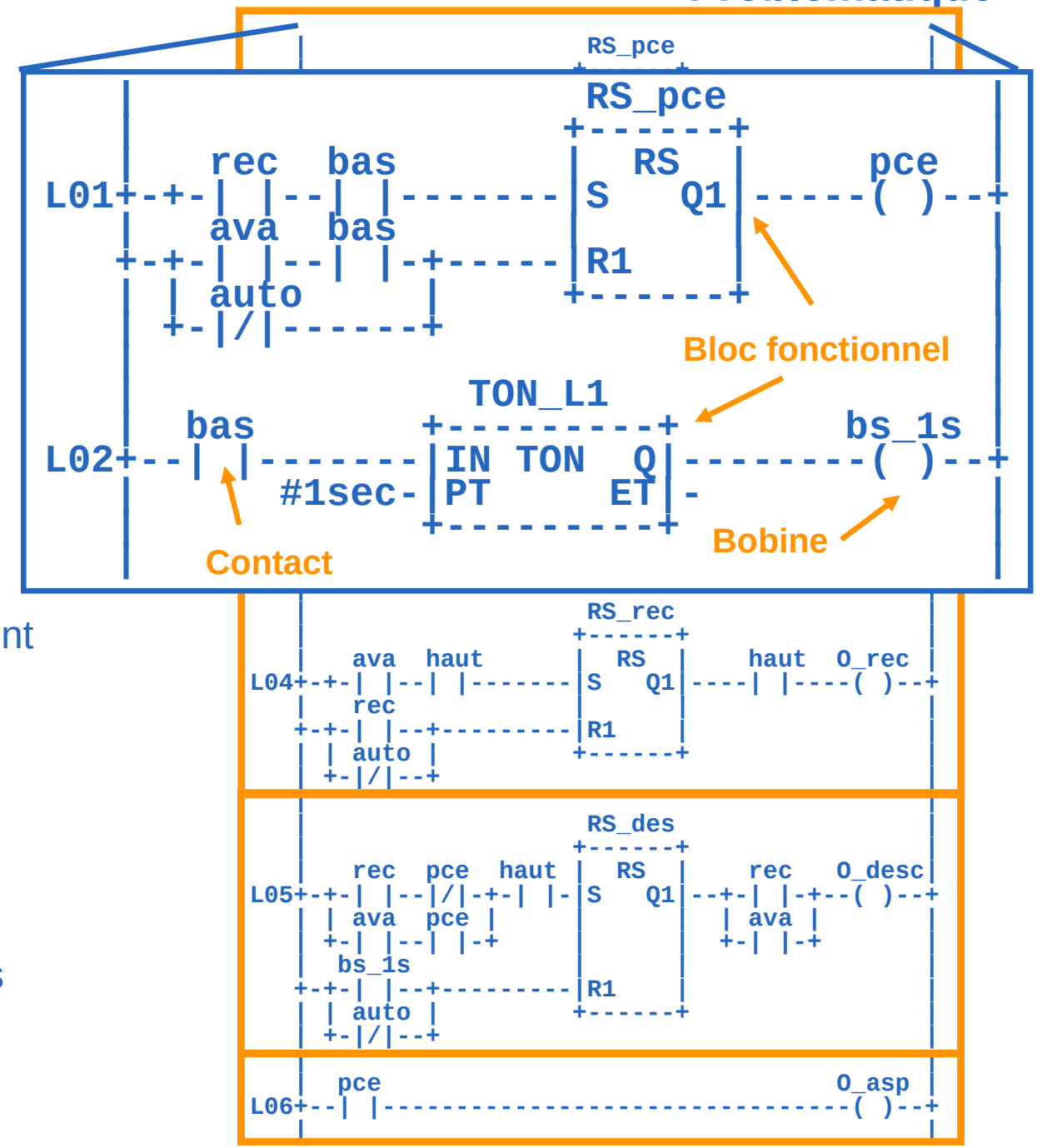
- **Fonctionnement :**
  **Mono – tache cyclique**
  - collecte des informations du processus (entrées)
  - traite les données reçues
  - donne des ordres au processus (sorties)

- **Programmation :**
  **5 langages normalisés (IEC 61131-3):**
  - textuels : ST, IL
  - graphiques : SFC, LADDER, FBD



# Ladder Diagram

- **Caractéristique**
  - langage graphique normalisé (Norme IEC 61131-3 )
- **Constitution**
  - réseaux exécutés séquentiellement
    - contacts
    - bobines
    - blocs fonctionnels
      - mémoires
      - temporisations
      - détecteurs d'événements



# Besoins

- **Comment prouver que le contrôle du processus :**
  - est exempt de bugs ?
  - est conforme aux attentes du cahier des charges ?
- **Scénario de test sur le réel ?**
  - très proche de l'utilisation nominale ;
  - vue très limitée des possibilités du contrôle ;
  - nécessite du matériel spécifique.
- **Simulation par modèle ?**
  - long ;
  - n'atteint pas tous les cas d'utilisations possibles.
- **Vérification formelle**
  - prouve que le programme de commande est conforme ;
  - taux de couverture de 100% du comportement logiciel.
  - MAIS vérification de **modèles de programme** et non de programmes.

# Vérification formelle du comportement d'un système

## Comportement Désiré (CD)

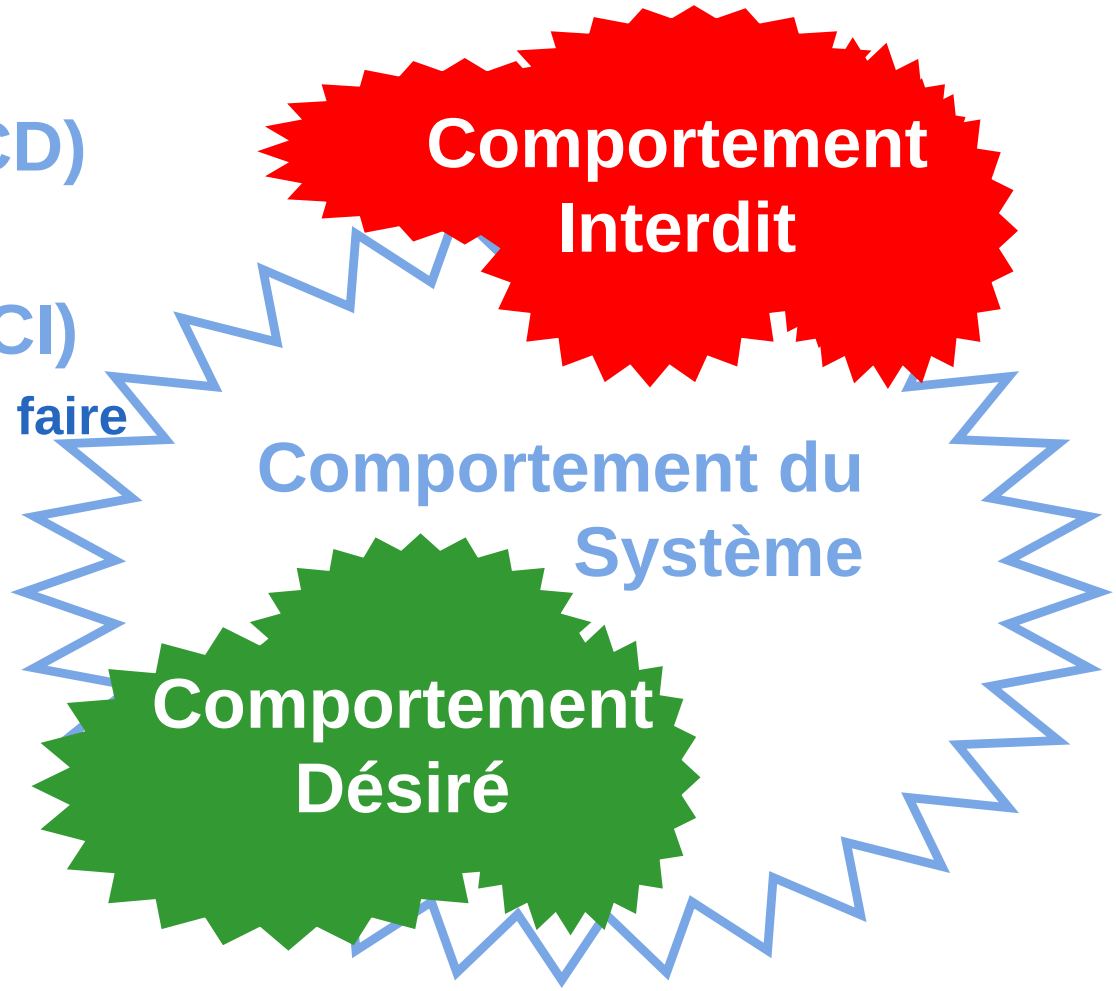
- Ce que le système **doit faire**

## Comportement Interdit (CI)

- Ce que le système **ne doit pas faire**

## Comportement du système (CS)

- Ce que fait le système




## Vérification formelle :

- Prouver que

$$CD \subset CS$$

$$CI \cap CS = \varnothing$$

# Intégration dans développement de programme de commande

Cahier des charges

Conception du logiciel de commande

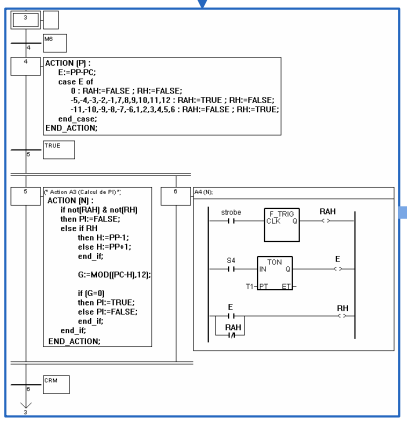
Comportement Interdit

Comportement Désiré

Propriétés à vérifier

Vérification formelle

Propriétés vérifiées ou non




Contrôleur

processus

Comportement du Système

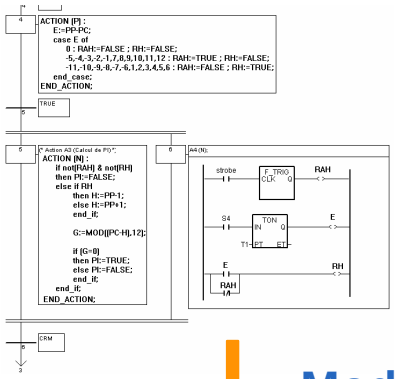
# Définitions

- **« Model ...**
  - **connaître** ou **caractériser** tous les **états** d'évolution du programme de contrôle **nécessaire à la vérification**.
  - Ex : le graphe des situations accessibles d'un GRAFCET ou d'un réseau de Pétri.
- **... - Checking »**
  - **vérifier** des **propriétés** sur ces états ou la relation entre ces états.

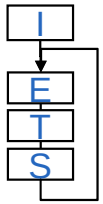
- **État**
  - Valeur des variables à un instant donné.

# Principe

Programme de contrôle

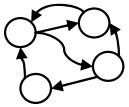


Moniteur




**Modélisation**

**Modèle de comportement (Automate à états)**



**Propriétés**

- il est toujours vrais que ...
- pas de blocage.
- tel événement suit toujours celui-ci.

**Formalisation**

AG(APB→AF ~horn)

AG(~d1→AF ~lig)

**Logique temporelle (LTL, CTL, …)**

**MODEL CHECKER**

**Propriétés vérifiées ou diagnostic en cas d'échec**

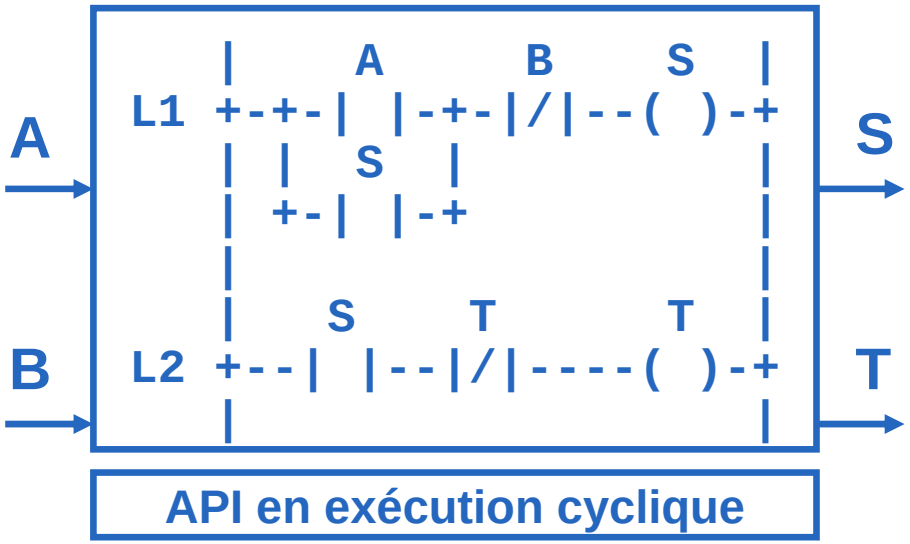
# Modélisation du comportement de l'exécution du programme LD par un API

- **Etat de l'automate**
  - Etat d'avancement du cycle de l'API
    - Lecture des entrées
    - Exécution du programme utilisateur
      - Exécution de L1
      - Exécution de L2
    - Ecriture des sorties

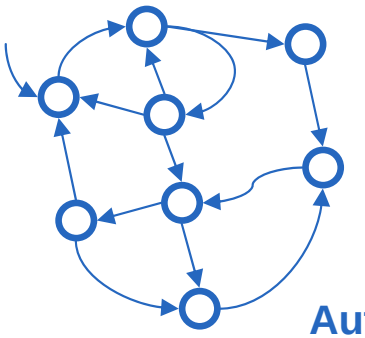
  **4 possibilités**

  - Valeur des variables présentes dans le programme
    - A, B, S, T ⟵ **16 combinaisons**

**Taille maximale de l'automate : 64 états**



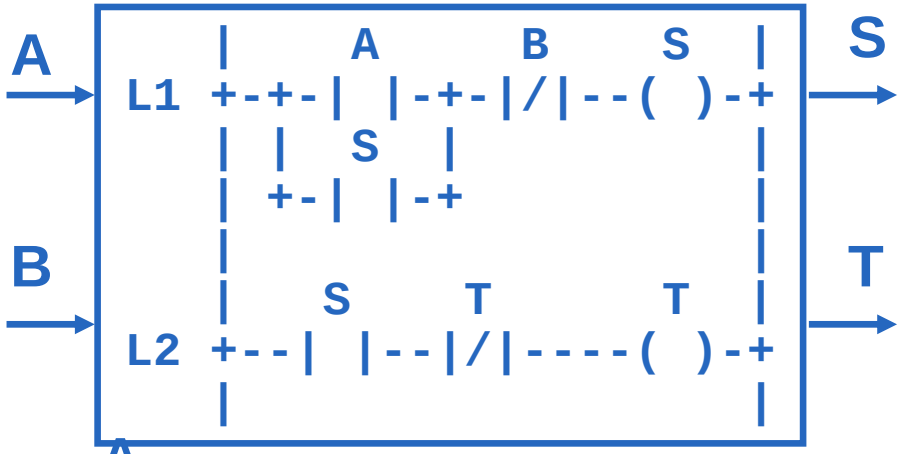

**Formalisation**

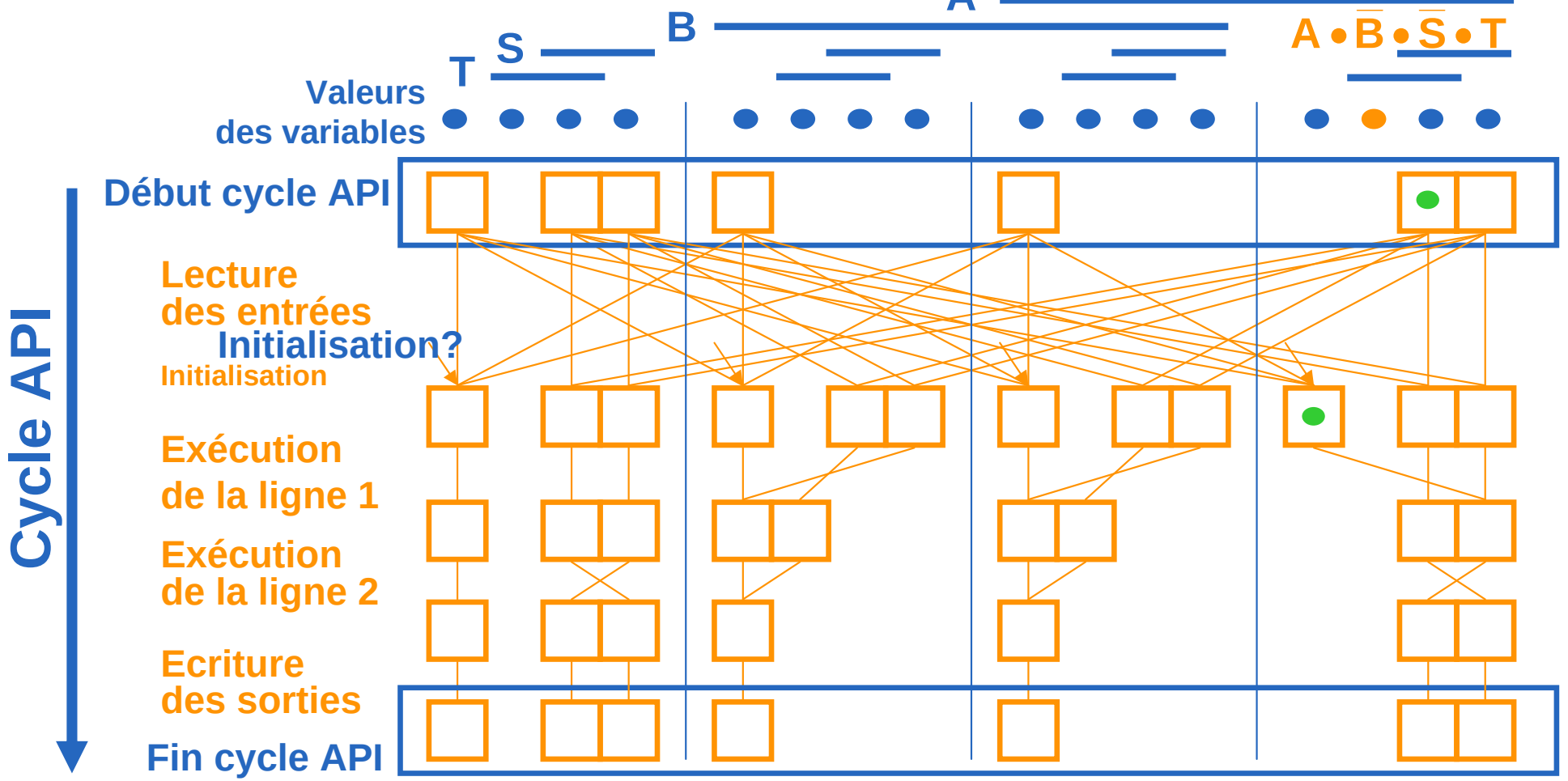


**Automate à états finis**

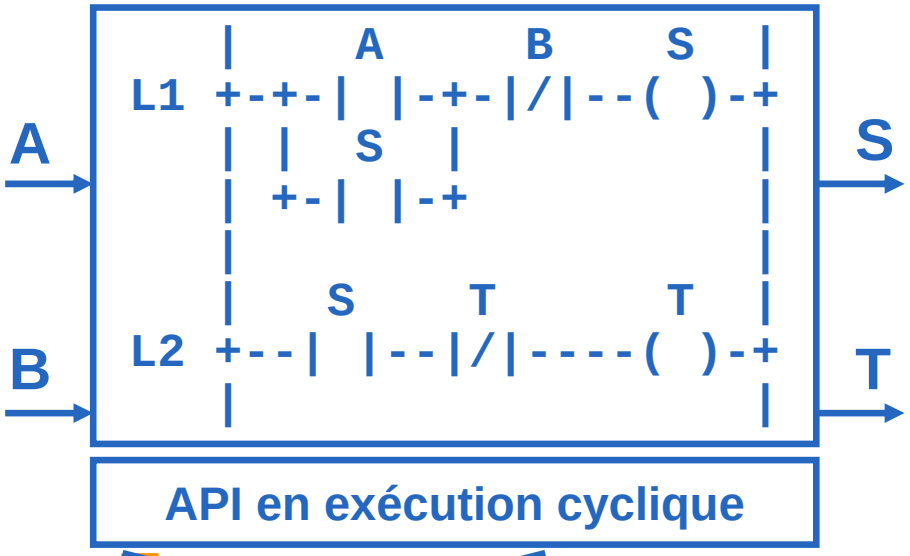
# Génération de l'automate à états finis

35 états, 56 transitions

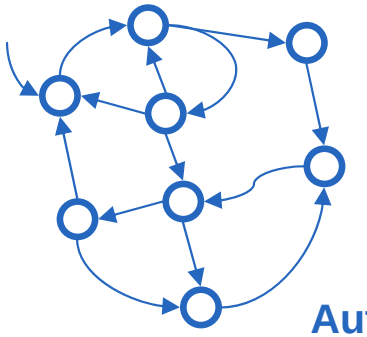


# Codage dans un outil de Model-Checking




Formalisation



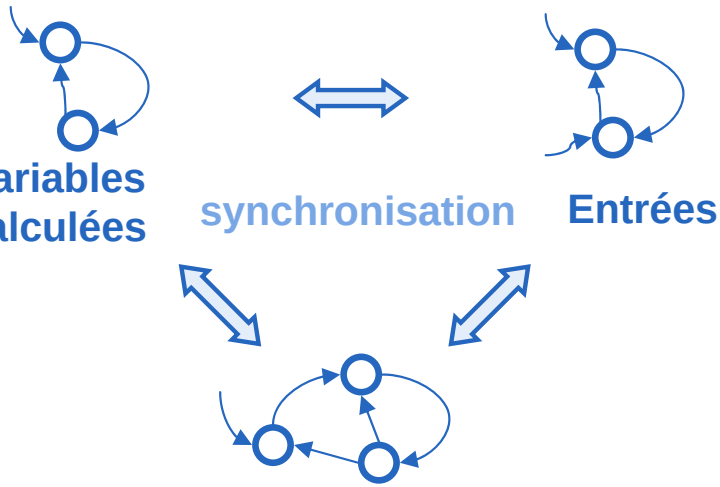
Passage direct difficile à formaliser et peu opérationnel

Automate à états finis

Identification des éléments clés

- Moniteur d'exécution
- Variables du programme
  - Entrées
  - Variables calculées

Modélisation du comportement des éléments clés




Construction de l'automate à états

Outil spécialisé

# UPPAAL

## Caractéristiques :

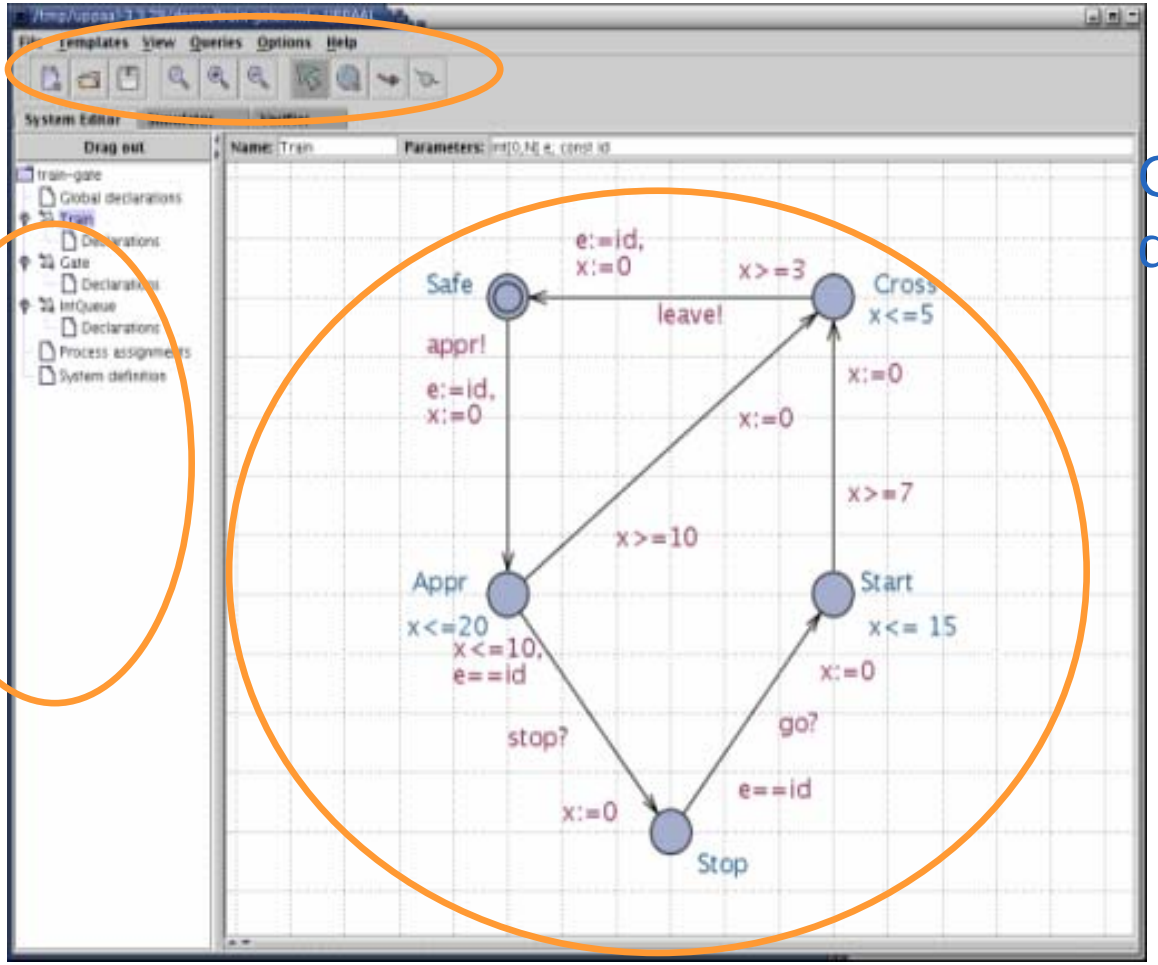
- outil pour **modéliser, simuler et vérifier** des systèmes en temps réel.
- modélise les systèmes comme une collection d'automates :
  - non déterministes;
  - temporisés par des horloges à valeurs réelles;
  - communiquant par des canaux de messages ou des variables partagées.

## UPPAAL se compose de trois parties:

- un éditeur graphique
- un simulateur
- un model-checker

# Editeur graphique

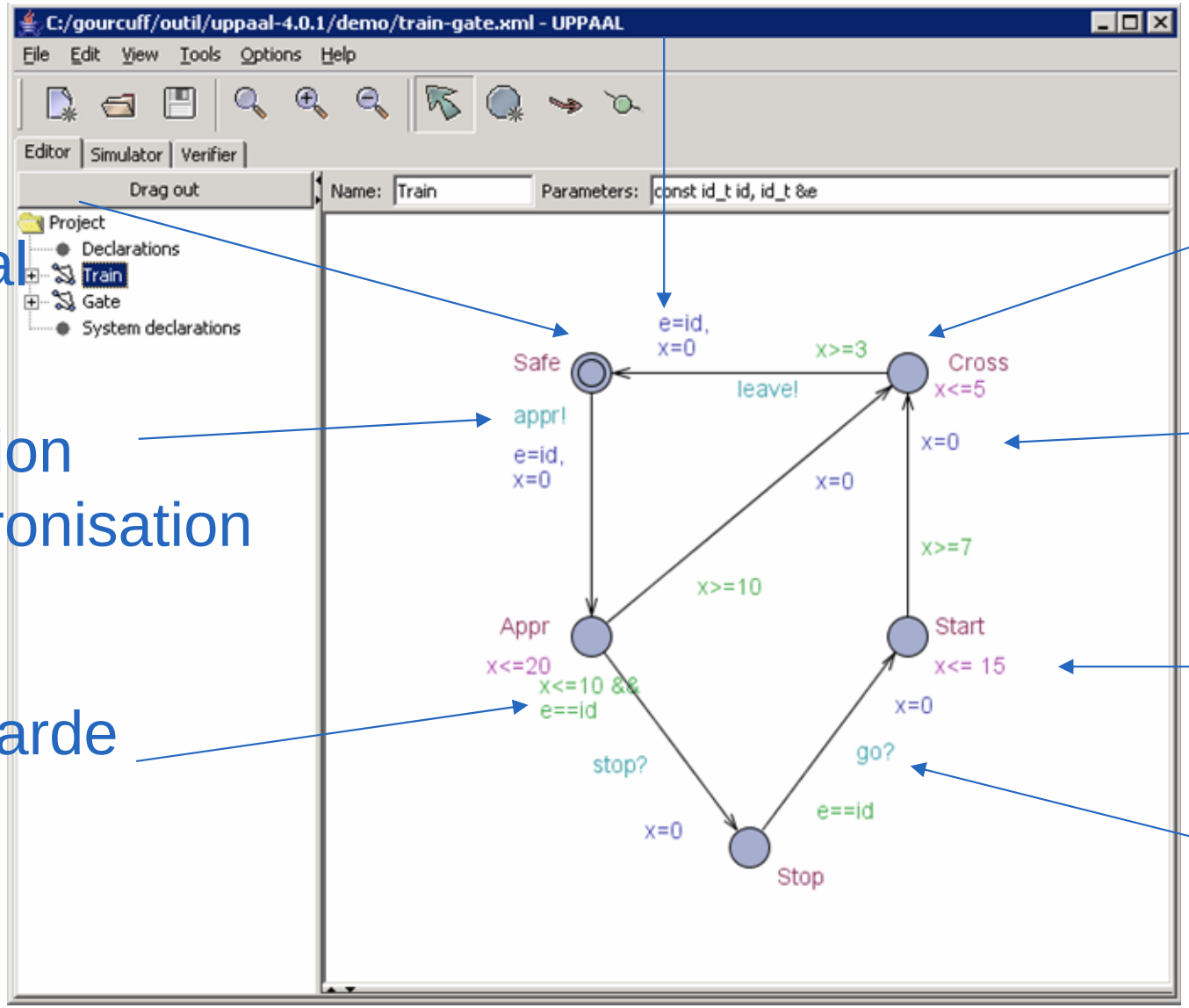
Outils




Arborescence des automates
Configuration
Déclaration de variables

Construction d'automate

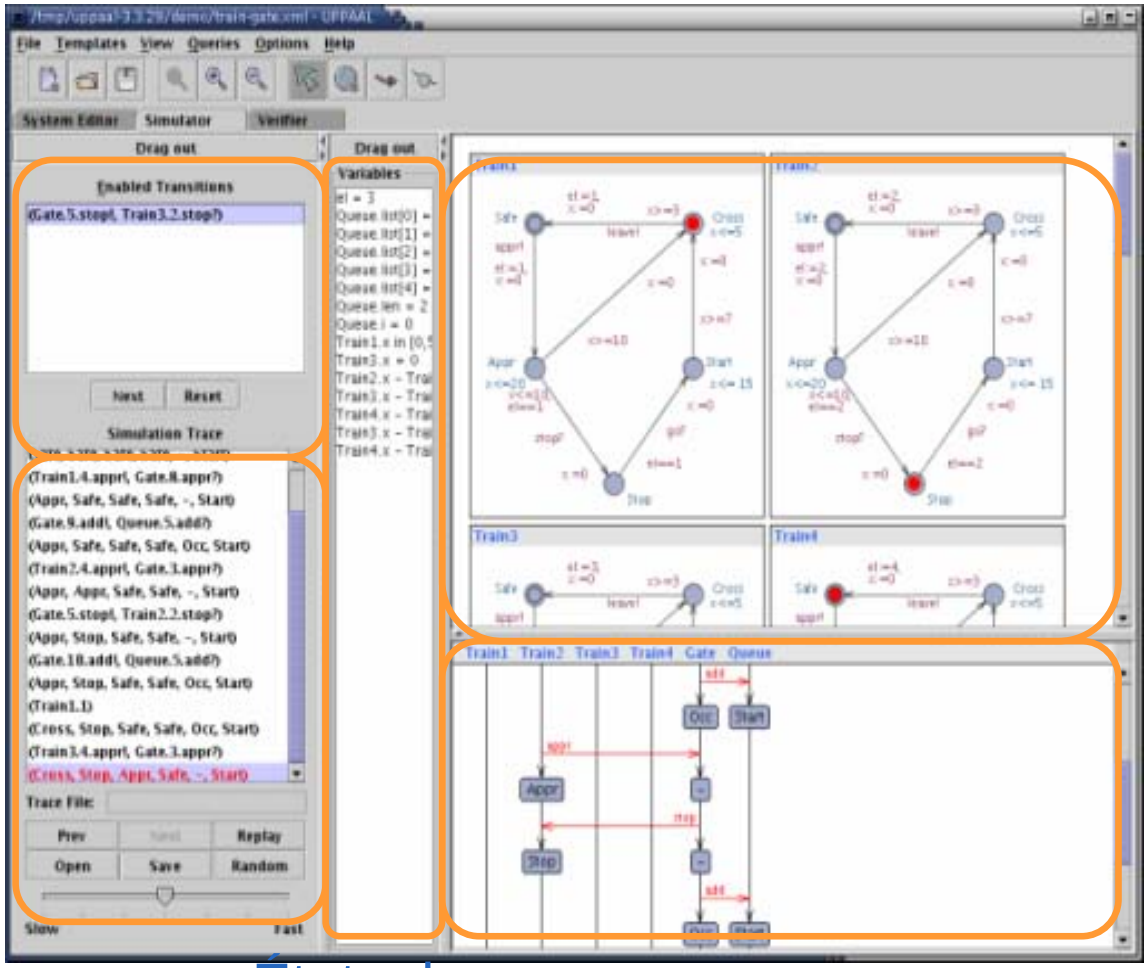
# Editeur graphique



# Simulateur

Choix d'évolution

Historique des évolutions choisies




États des variables

États des automates communicants

Évolution des automates et synchronisations

# Les états

- **À tout moment un et un seul état est actif dans un automate.**
- **3 états spéciaux :**
  - Initial 
    état unique et obligatoire, activé à l'initialisation.
  - Committed 
    état qui, une fois actif, doit être désactivé au prochain pas d'évolution.
  - Urgent 
    état qui, une fois actif, doit être désactivé dans un temps nul.
- **Invariant :** 

  - condition associable à un état.
  - si l'état est actif, la condition doit être respectée.

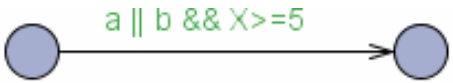
# Transition

- **4 éléments configurables :**
  - Sélection (select)

    Valeur non déterministe d'une variable dans un set donné: permet de regrouper la définition de plusieurs transition entre deux états.

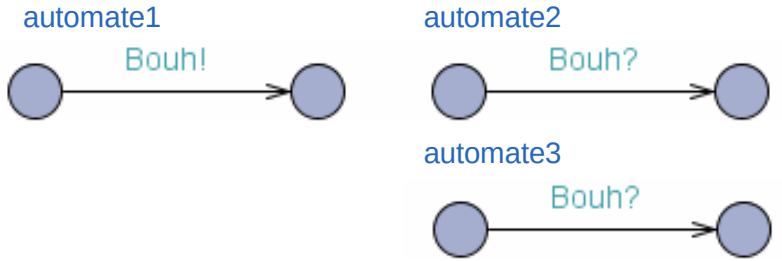
  - Garde (guard)

    condition de franchissement : si fausse, la transition ne peut être franchie.




  - Synchronisation (synchronization)

    élément de communication entre automate : permet le franchissement simultané de transitions entre automates.

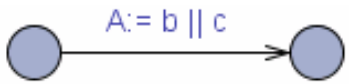
| Normale (1 émetteur, 1 récepteur) | Broadcast (1 émetteur, n récepteur(s)) $n \in \mathbb{N}$ |
|---|---|
|  |  |





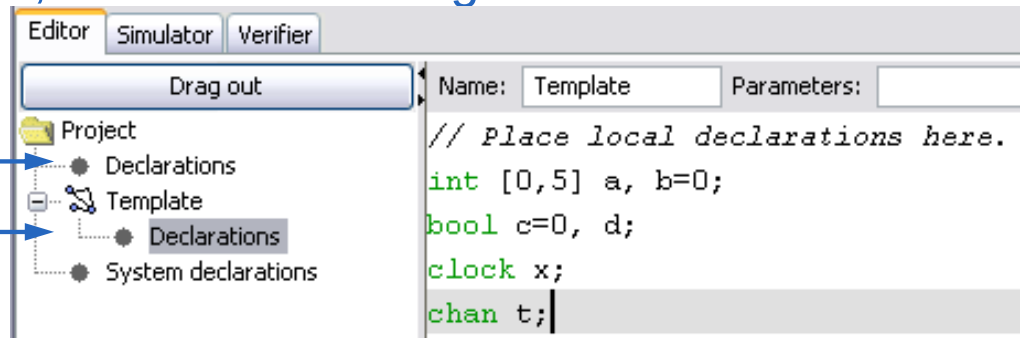
  - Affectation (update et select)

    modification de la valeur de variables par calcul, avec une formule (update)



# Les variables

- **Quatre types élémentaires:**
  - int : variable entière, par défaut dans la plage [-32768, 32767] et initialisée à 0. Plage et initialisation configurable.
  - bool : variable booléenne, initialisée par défaut à 0. Initialisation configurable.
  - clock : variable réelle dédiée au temps. Test et affectation uniquement avec des valeurs entières.
  - chan : canal de communication, de type ‘normal’ par défaut. 3 autres types possibles : ‘urgent’, ‘broadcast’ et ‘urgent broadcast’.
- **Déclaration :**
  - Globale
  - Locale



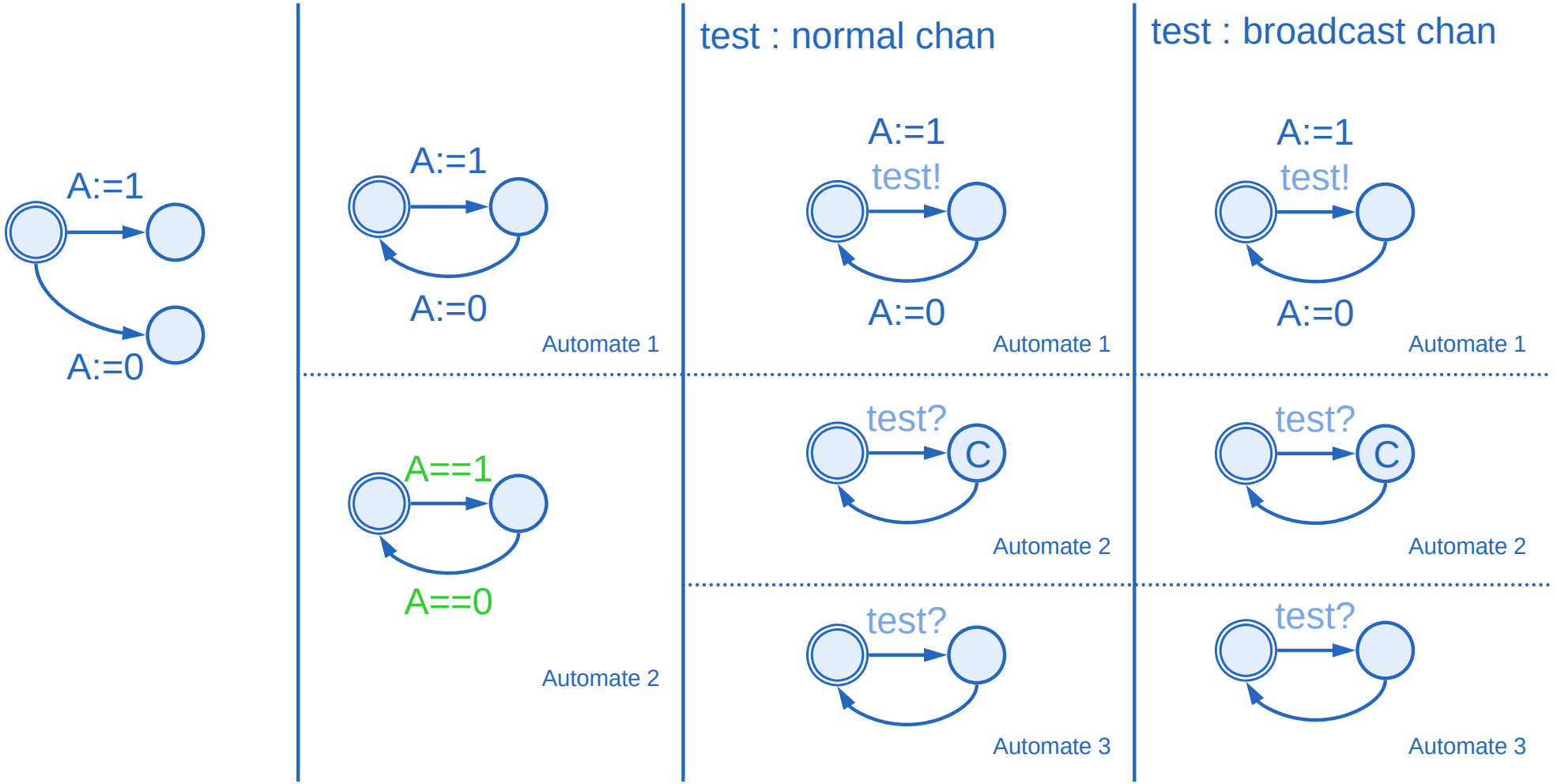

  - Implicite : Pour chaque état Ei de chaque automate Ai, une variable de type ‘bool’ représentant son activité est créée : Ai.Ei

# Règles d'évolution

- **Sans synchronisation :**
  - Si une transition est franchissable, elle peut être franchie **mais sans obligation**.
  - Une seule transition peut être franchie à chaque pas d'évolution **parmi tous les automates.**
- **Avec synchronisation :**
  - Une synchronisation possible de plusieurs automates (i.e. les transitions correspondantes sont franchissables sans la synchronisation) est :
    - immédiate ;
    - prioritaire sur d'autres transitions normales.
  - Une synchronisation **normale** non possible est bloquante pour l'émetteur et le récepteur.
  - Une synchronisation **broadcast** non possible est bloquante seulement pour les récepteurs.
- **Respect des invariants :**
  - Les invariants de chaque état actif doivent toujours être respectés.
  - Si un invariant ne peut être respecté, **tous** les automates sont alors bloqués (deadlock).

# Exemple d'évolution et première utilisation de UPPAAL

- Réaliser des modèles suivants et simuler leur évolution :



# Exercices d'utilisation

- ## Construire et simuler:

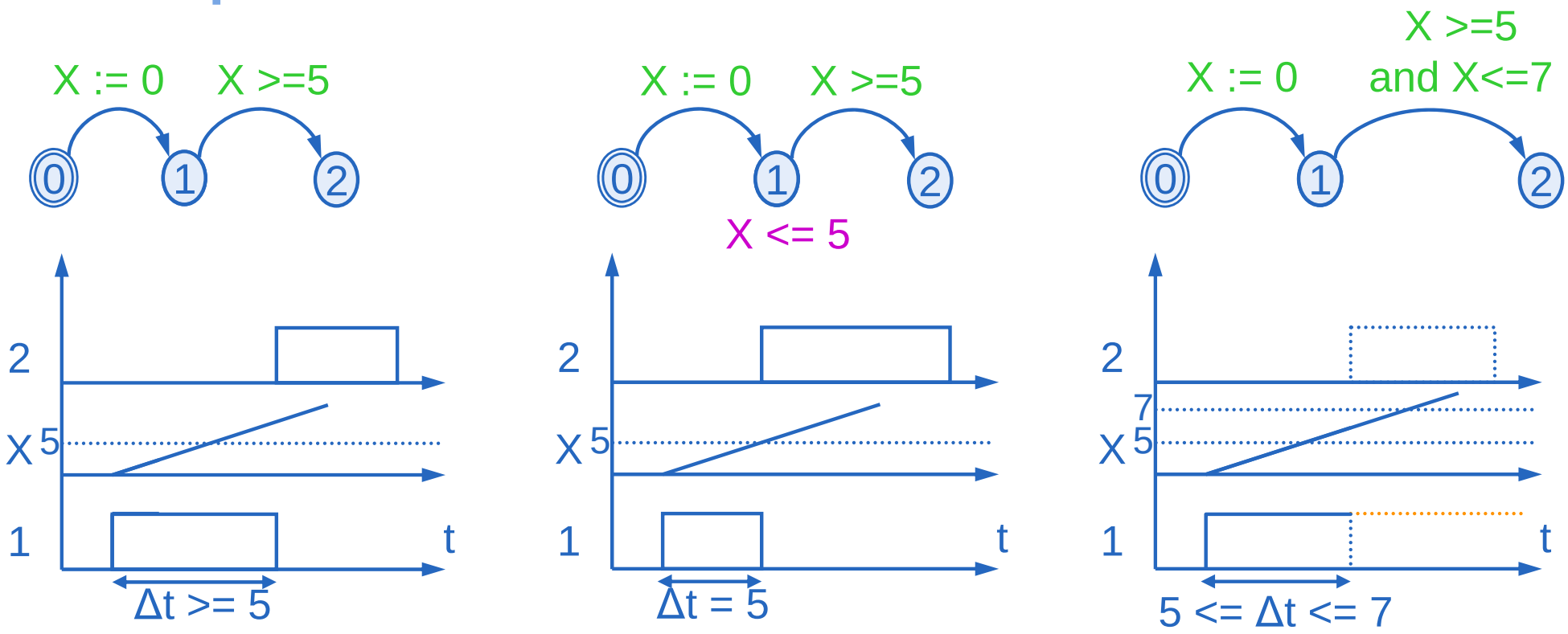
1. Un automate modifiant alternativement la valeur de la variable booléenne A entre 0 et 1.
2. Un automate comptant jusqu'à 20 avec une variable B puis mettant la variable A à 1.
3. Un automate attribuant une valeur entre 0 et 5 à la variable entière A comprise entre 0 et 10.
4. Un automate aillant le même comportement que trois automates du (1), à comparer le comportement avec 3 automates.
5. Deux automates jouant au ping-pong avec les messages « ping » et « pong ».
6. Trois automates manipulant une variable booléenne chacun, attribuant la valeur 0 ou 1, de façon non déterministe, simultanément au message *top!* d'un quatrième automate.

# Le temps

## Décompte du temps avec les horloges à valeur réelle.

- Écoulement du temps simultané sur chaque horloge
- Tests des horloges dans les gardes ou les invariants par rapport à un entier.

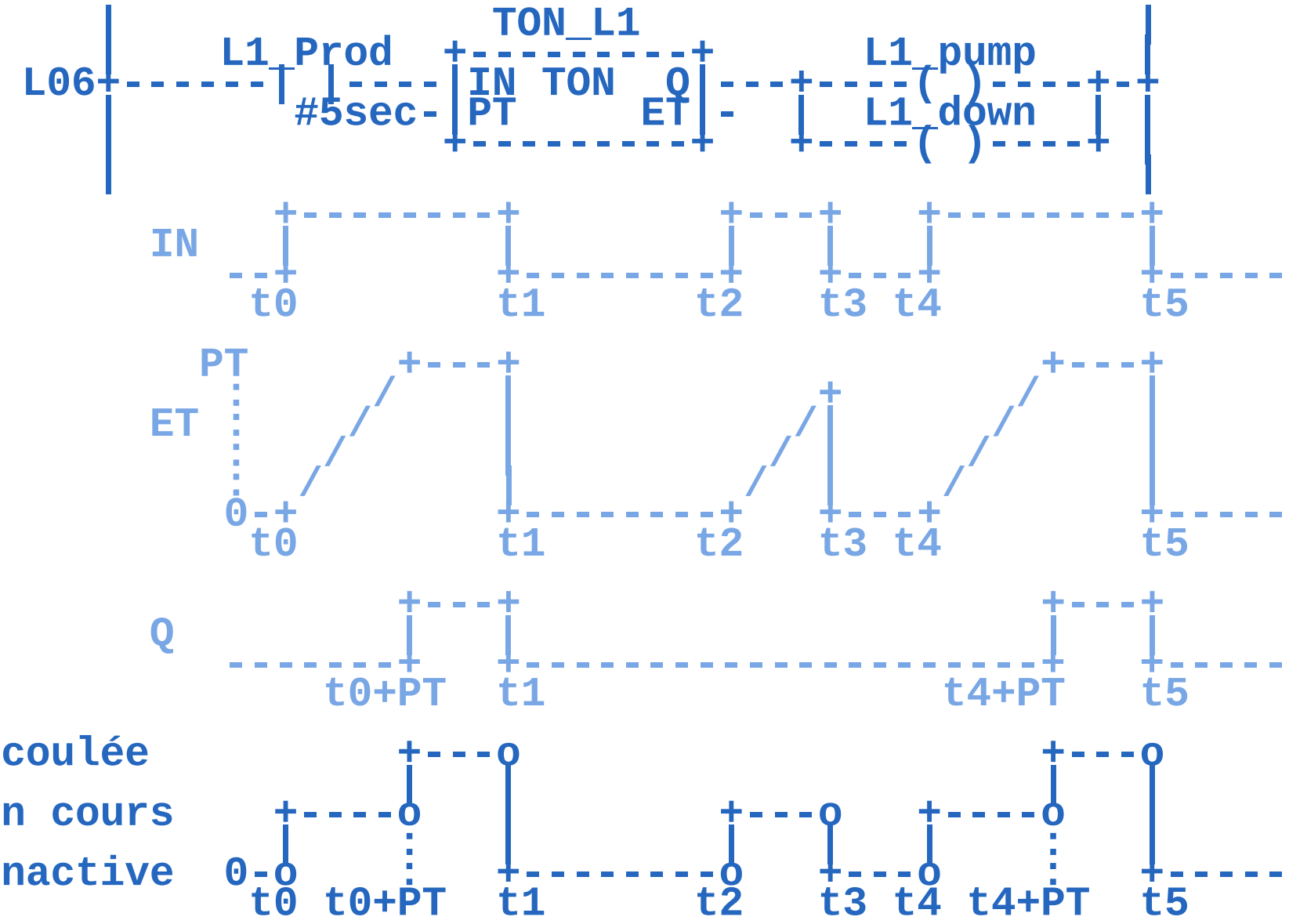
## Exemples :



# Exercices sur les horloges

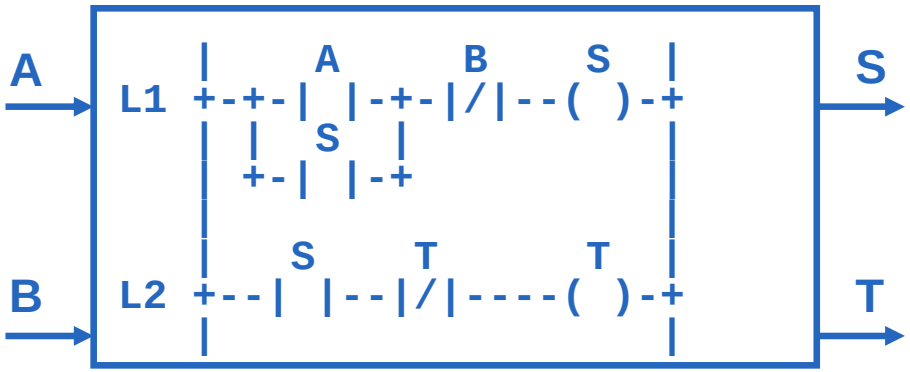
- **Construire et simuler :**
  1. Un automate attendant un temps supérieur à 20 unités de temps puis mettant la variable A à 1.
  2. Un automate modifiant alternativement la valeur de la variable booléenne A entre 0 et 1, toutes les 10 unités de temps.
  3. Un automate représentant le fonctionnement d'une TON.
     L'entrée de la TON sera modélisée par un automate indépendant.

# Temporisation TON

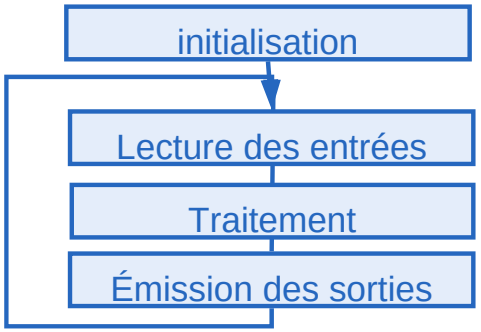


# Application

- **Modéliser le système suivant...**




**API en exécution cyclique**




Temps de cycle : 1 unité de temps

Processus modélisé par des entrées indéterministes.

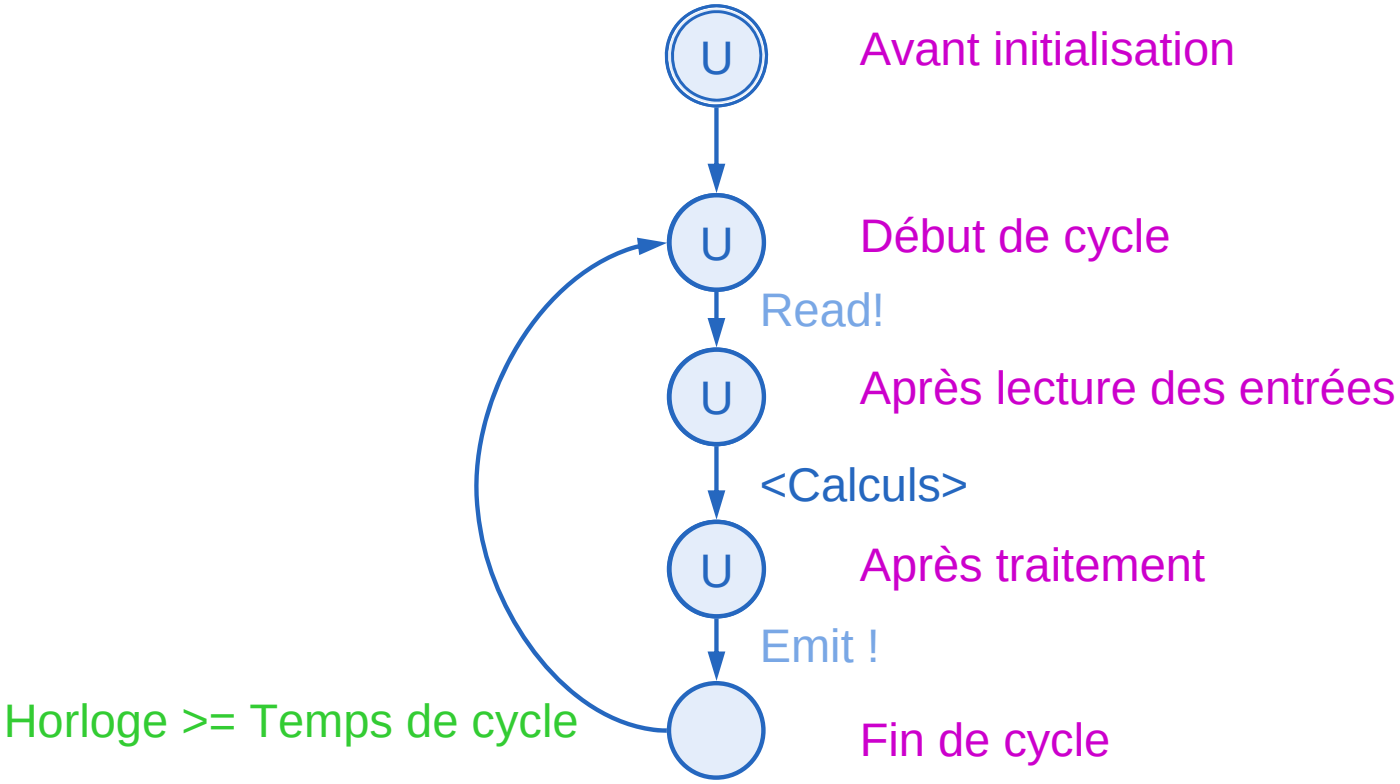
- **... en vue de la vérification de la propriété suivante...**

Lorsque l'entrée B est présente, la sortie T doit être absente.

- **... avec l'hypothèse suivante.**

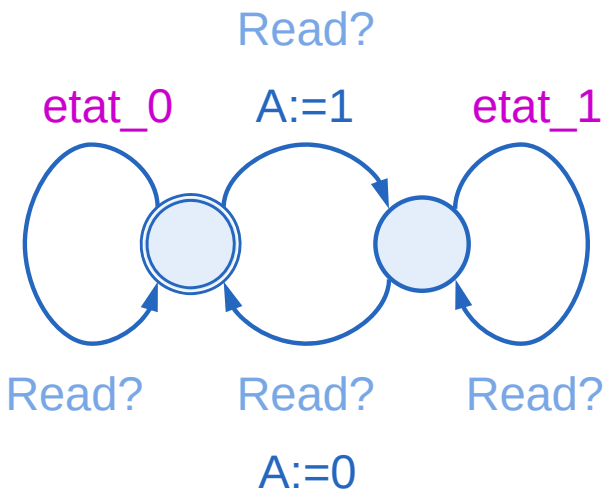
Le temps de cycle s'écoule seulement entre l'émission des sorties et la lecture des entrées du cycle suivant.

# Le moniteur d'exécution

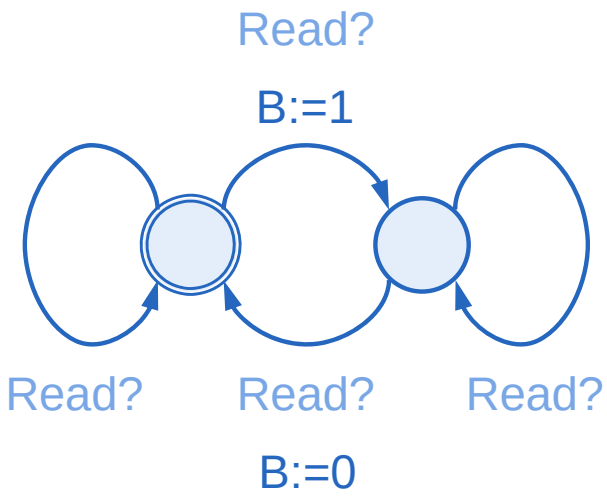


# Variable d'entrée

Variable A

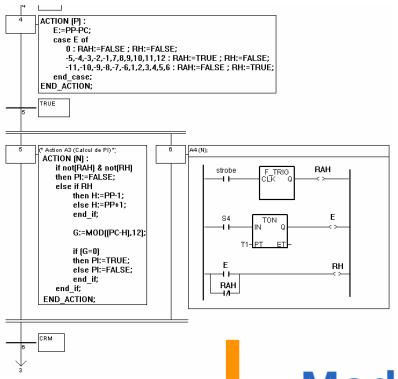



Variable B

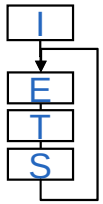


# Principe

Programme de contrôle

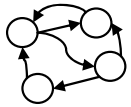


Moniteur




**Modélisation**

**Modèle de comportement (Automate à états)**



**Propriétés**

- il est toujours vrais que ...
- pas de blocage.
- tel événement suit toujours celui-ci.

**Formalisation**

AG(APB→AF ~horn)

AG(~d1→AF ~lig)

**Logique temporelle (LTL, CTL, …)**

**MODEL CHECKER**

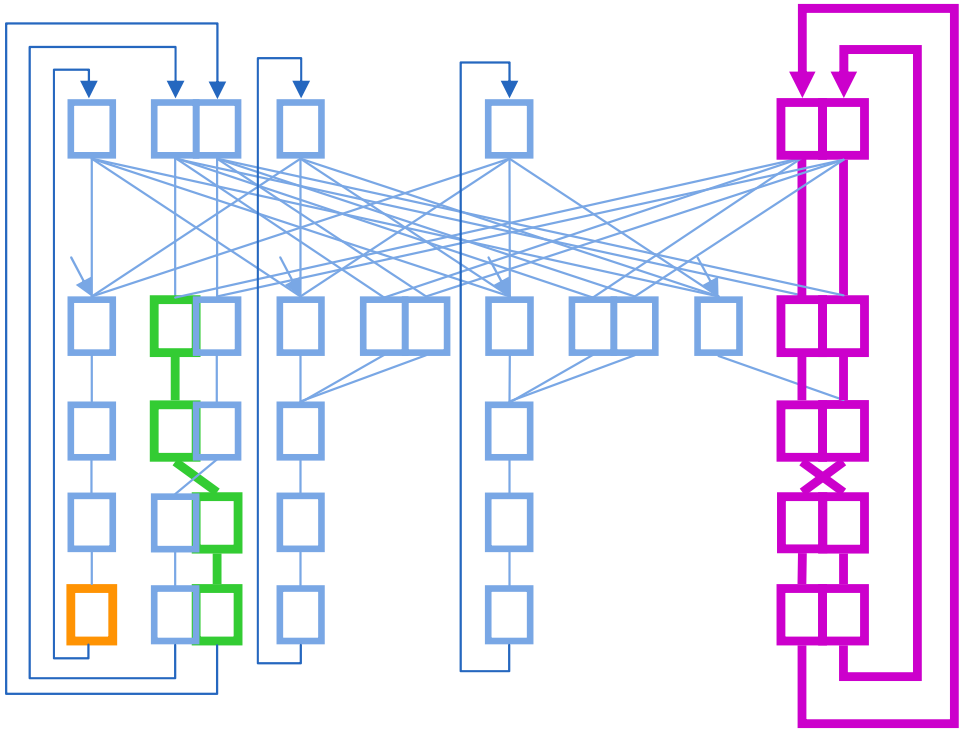
**Propriétés vérifiées ou diagnostic en cas d'échec**

# Vérification de propriétés

- **Recherche dans l'automate**
  - d'**états**
  - de **chemins**
  - de **circuits**

  ayant des caractéristiques données.

- **Expression des propriétés**
  - Logique temporelle
  - Automate ayant un rôle d'observateur



# La logique CTL : une logique temporelle

CTL : Logique temporelle utilisée pour énoncer formellement des propriétés.

## Logique temporelle

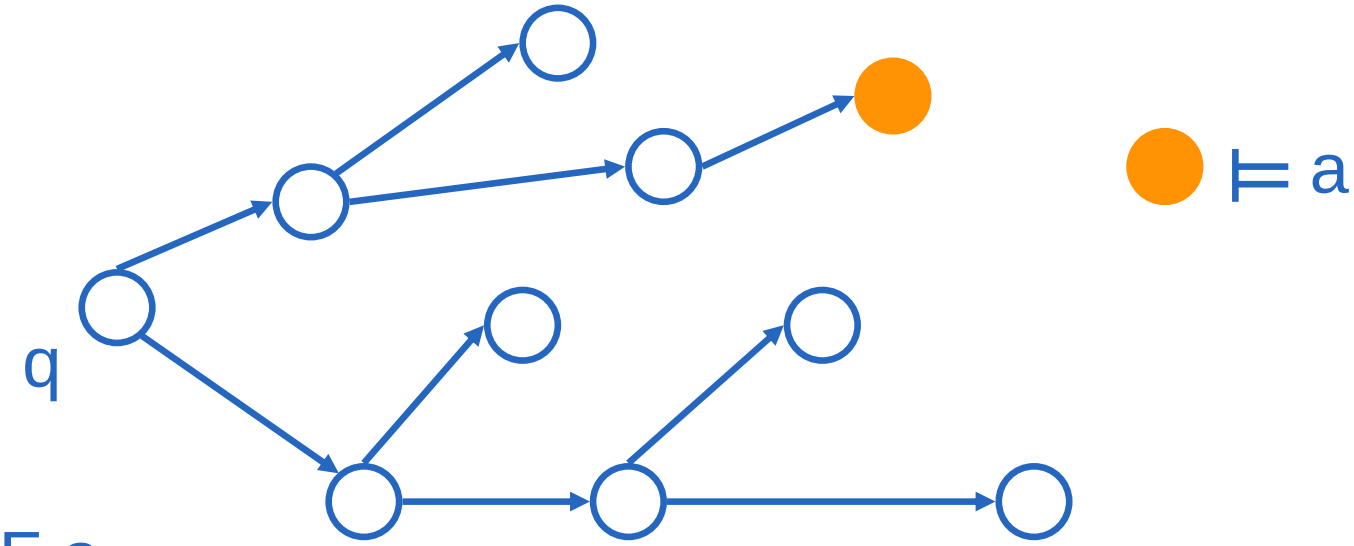
- Logique spécialisée dans les énoncés et raisonnements faisant intervenir la notion d'ordonnancement dans le temps.

## Formule CTL

- des propositions atomiques : a, b, …
- des opérateurs booléens : $\wedge$ $\vee$ $\neg$ $\rightarrow$
- des opérateurs temporels : EF, AF, AG, EG, E p U q, A p U q, EX, AX

# La logique CTL, l'opérateur EF (1)

EF a : " Il existe un chemin le long duquel
E
il existe un état vérifiant a "
F




⊨ a

q ⊨ EF a

# La logique CTL, l'opérateur EF (2)
# Analogie : Le métro parisien.


Chaque station est un état.

État initial : **Bagneux**.

Comportement : **Chemins possibles.**

Propriété à vérifier :

EF cine_présent

⇒Il existe un chemin le long duquel il existe un état vérifiant « cine_présent »

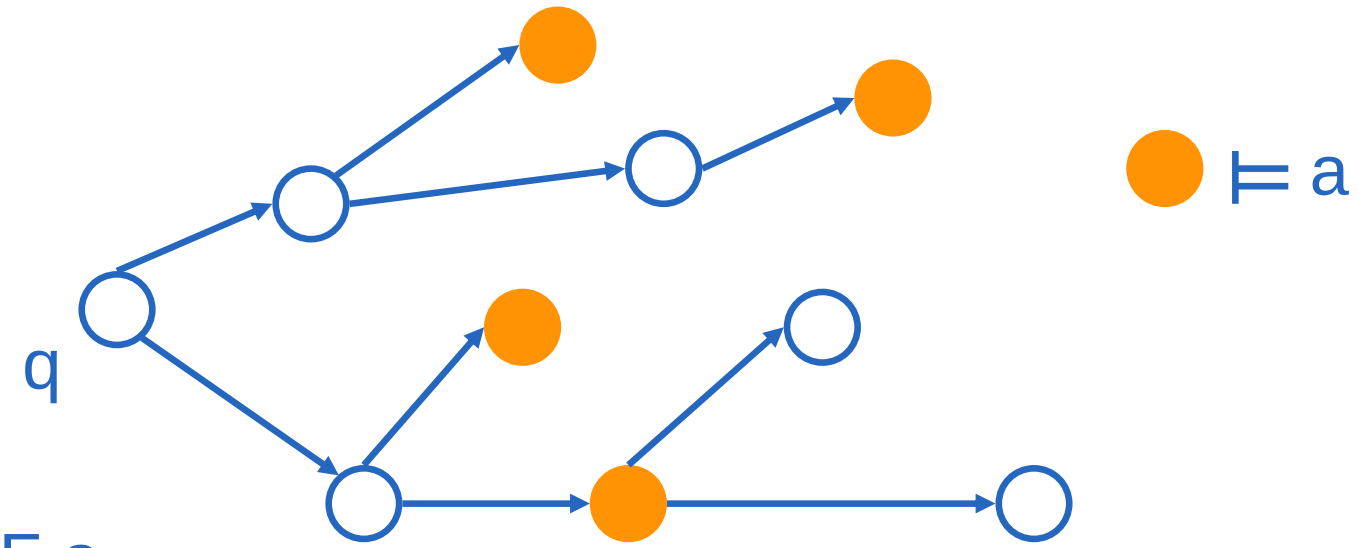
⇒Existe t'il un chemin depuis Bagneux qui me mène a un ciné présent prés de la station ?

Réponse : **OUI**.

# La logique CTL, l'opérateur AF

AF a : " Pour tout chemin, il existe un état vérifiant a "

A F

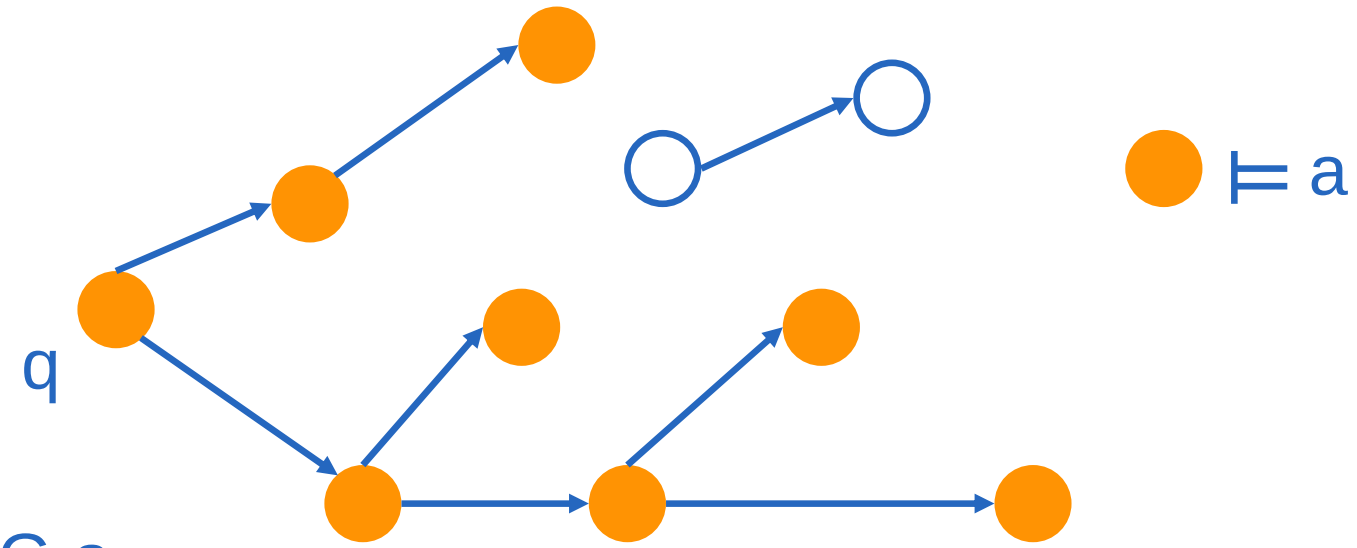



q ⊨ AF a

# La logique CTL, l'opérateur AG

AG a : " le long de tous les chemins, tous les états vérifient a "

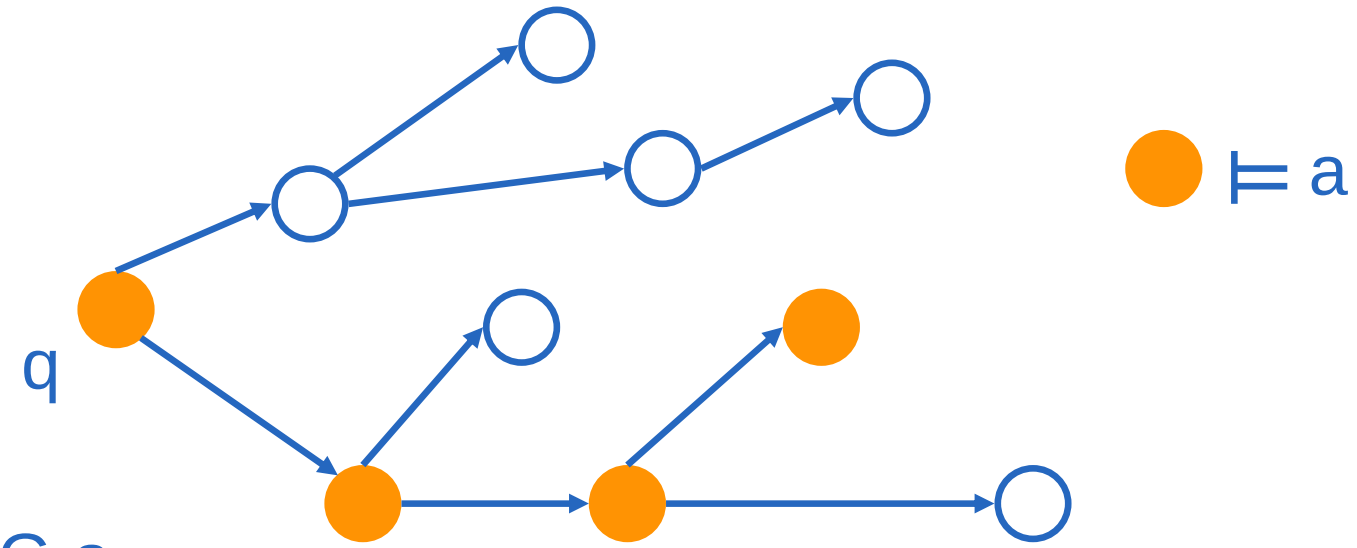
A G



# La logique CTL, l'opérateur EG

EG a : " Il existe un chemin le long duquel tous les états vérifient a "

E G



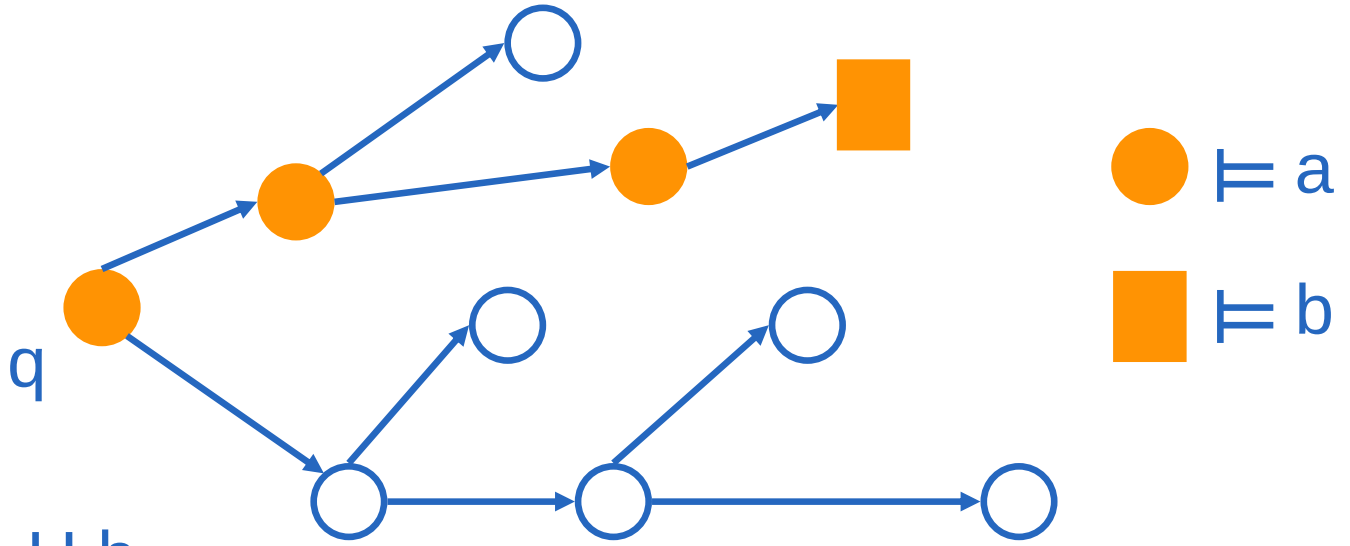

q ⊨ EG a

# La logique CTL, l'opérateur E a U b

E a U b : " Il existe un chemin où a est vrai jusqu'à ce que b soit vrai"

E

a U b

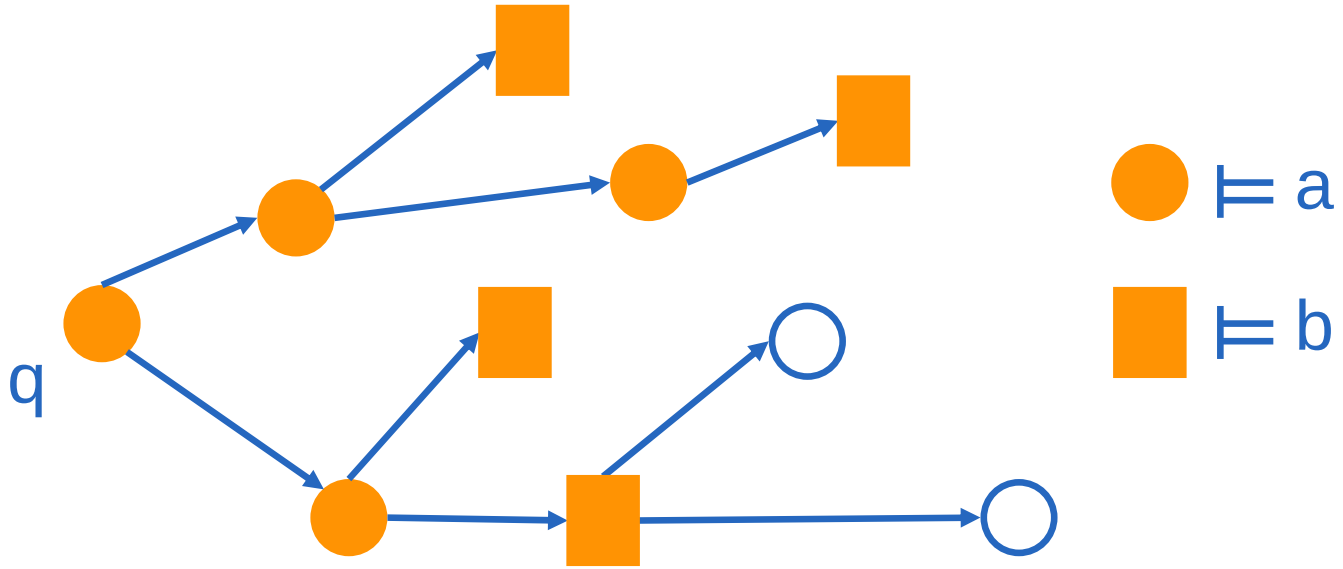



q ⊨ E a U b

# La logique CTL, l'opérateur A a U b

A a U b : " Sur tous les chemins, a est vrai jusqu'à ce que b soit vrai"

A a U b




q ⊨ A a U b

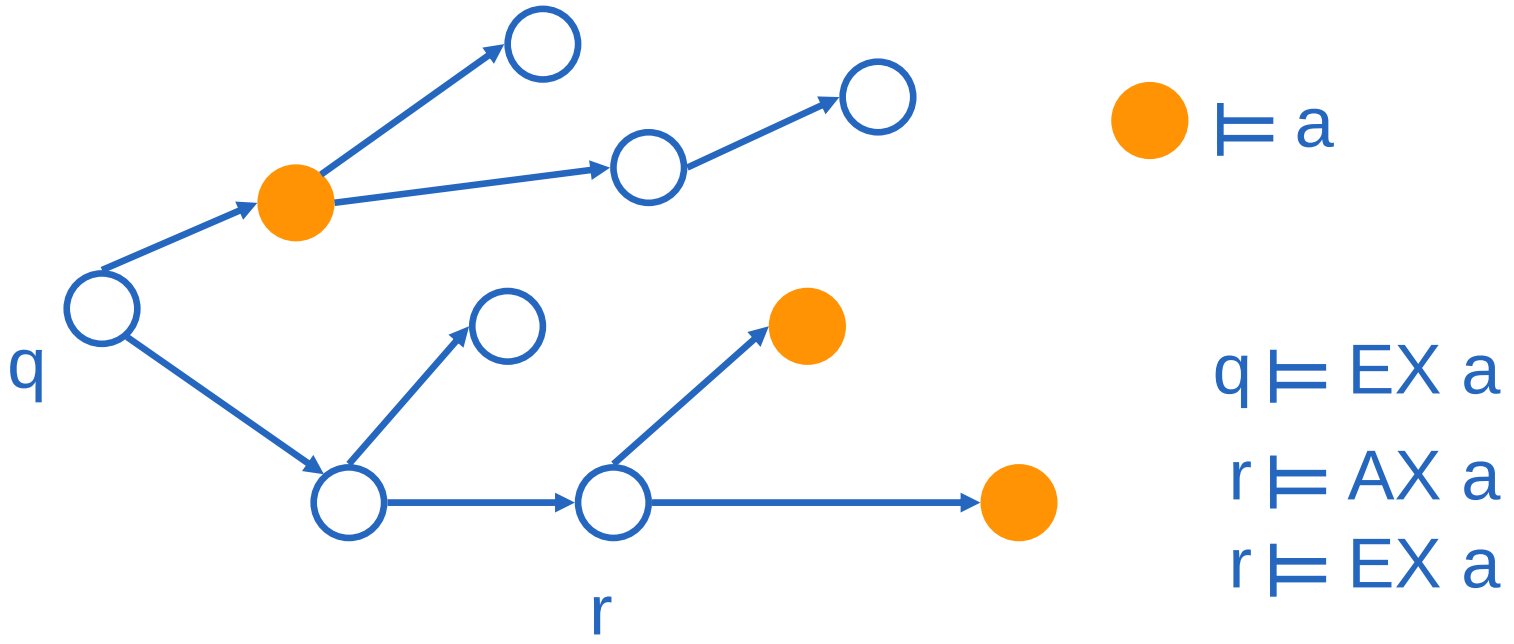
# La logique CTL, les opérateurs EX et AX

EX a : " Il existe un chemin le long duquel le prochain état vérifie a "

E X

AX a : " Tous les prochains états vérifient a "

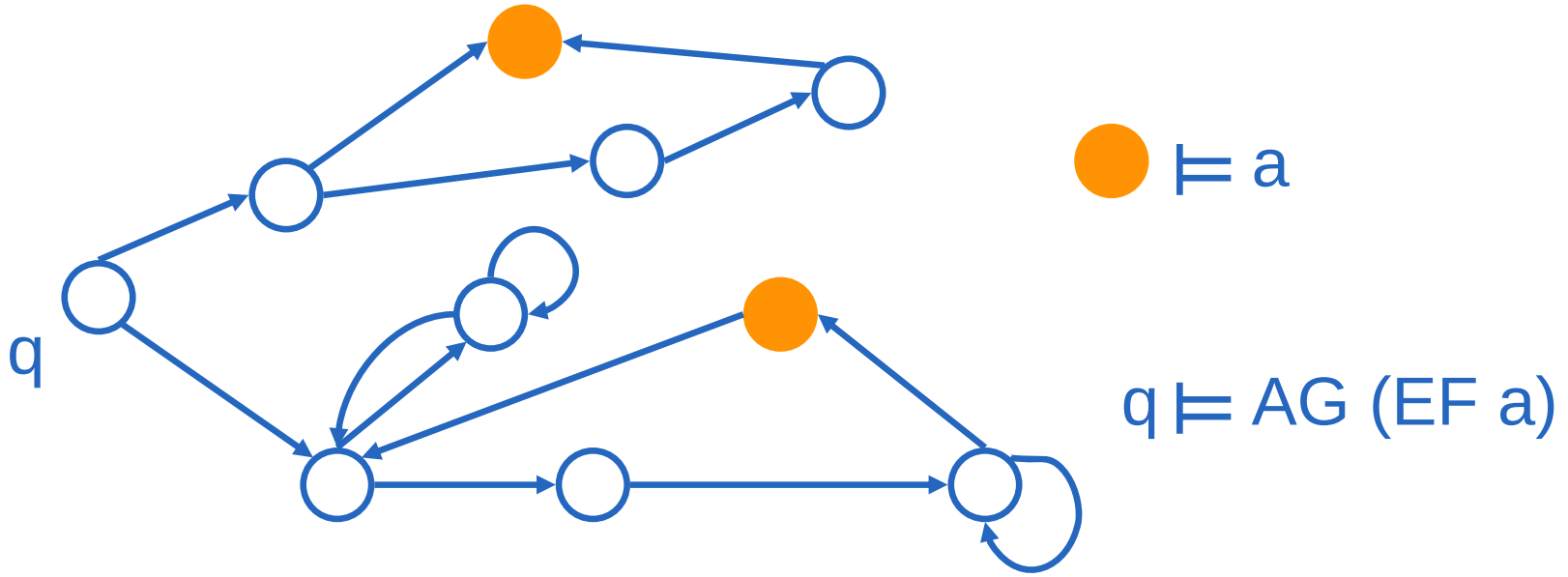
A X



# La logique CTL, emboîtement d'opérateurs

AG EF a : "Depuis tout état accessible, il est possible d'atteindre un état vérifiant a"

AG EF



# Egalités remarquables

$$AG\,a \equiv \neg EF \neg a$$

$$AF\,a \equiv \neg EG \neg a$$

$$EF\,a \equiv \neg AG \neg a$$

$$EG\,a \equiv \neg AF \neg a$$

Il est toujours vrais que la propriété ‘a’ est vérifiée

≡

Il n’est pas possible d’atteindre un état où la propriété ‘a’ n’est pas vérifiée

# Correspondance CTL - UPPAAL

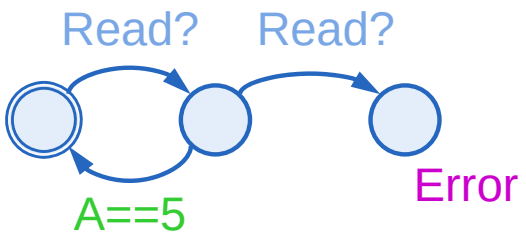
| CTL | UPPAAL |
|---|---|
| A | A |
| E | E |
| G | [ ] |
| F | <> |
| AG (p $\Rightarrow$ AF q) | p --> q |
| $\wedge$ $\vee$ $\neg$ $\Rightarrow$ | && \|\| ! imply<br>and or not imply |

# Vérification à l'aide d'observateurs

- **Dans le cas de propriétés complexes :**
  - traitant de simultanéité,
  - ou découlant de plusieurs séquentialités,
  - ou dépendantes du temps physique (horloge),
  - ou difficile à exprimer.

⇒ **Utilisation d'observateurs :**

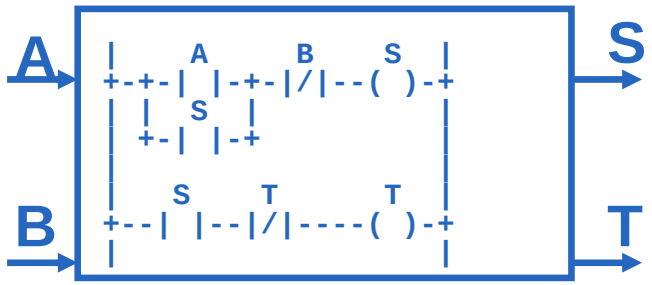
- Modification du modèle par :
  - Ajout d'un automate supplémentaire (automate observateur) représentant la propriété à vérifier.




Automate « obs »

  - Synchronisation de l'observateur avec le modèle original.
- Vérification avec une propriété CTL utilisant l'état de l'observateur.
  - AG (! obs.Error)

# Exemple de propriétés (1)

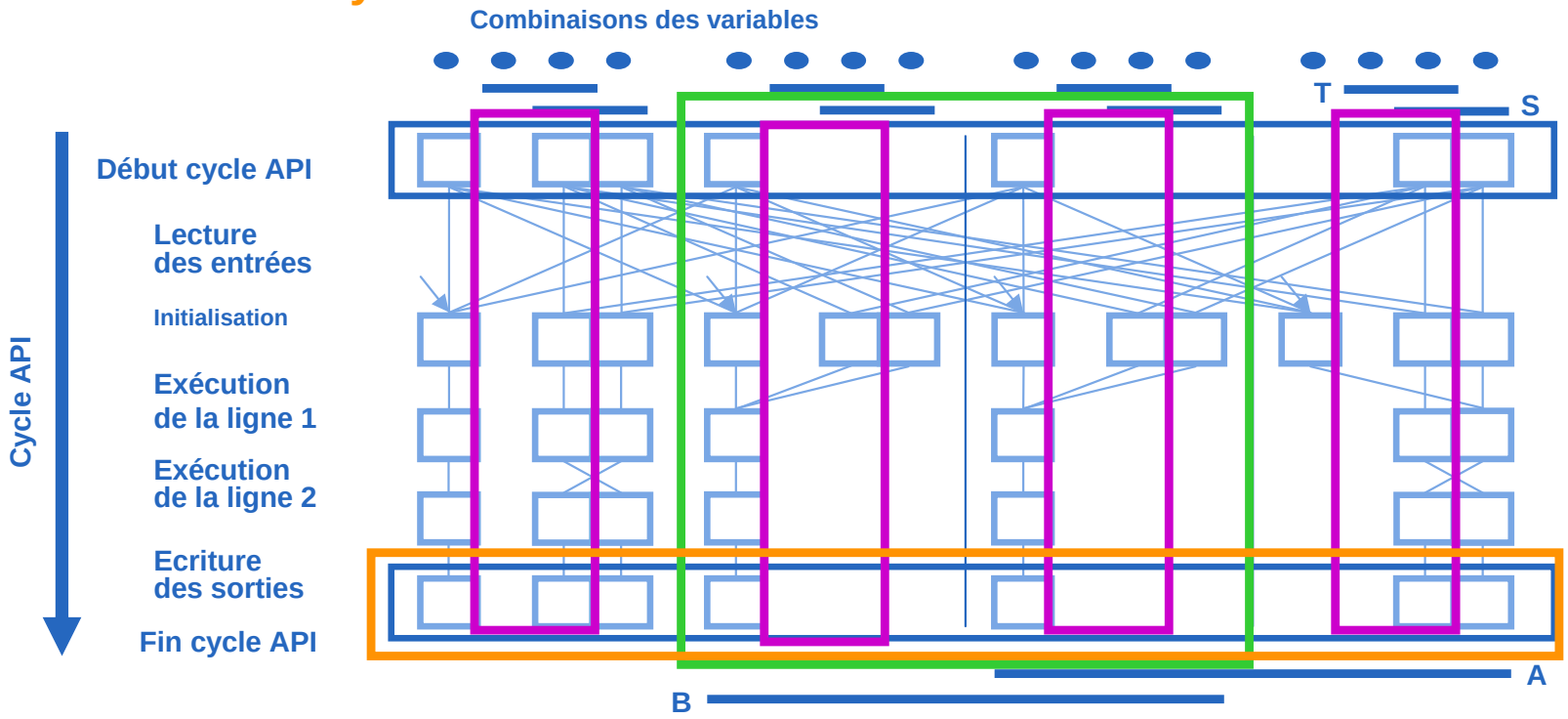



**P1** **Lorsque l'entrée B est présente, la sortie T doit être absente.**

**Existe-t-il un état de l'automate pour lequel :**

- **B est présent**
- **T est présent**
- **lorsque l'API est en fin de cycle**

**Non, P1 est vérifiée.**



# Exemple de propriétés (2)

- **Exercice :**

  **P1 « Lorsque l'entrée B est présente, la sortie T doit être absente »**

  - Enoncer la propriété P1 dans le langage CTL.
  - Vérifier la propriété P1 sur le modèle UPPAAL.

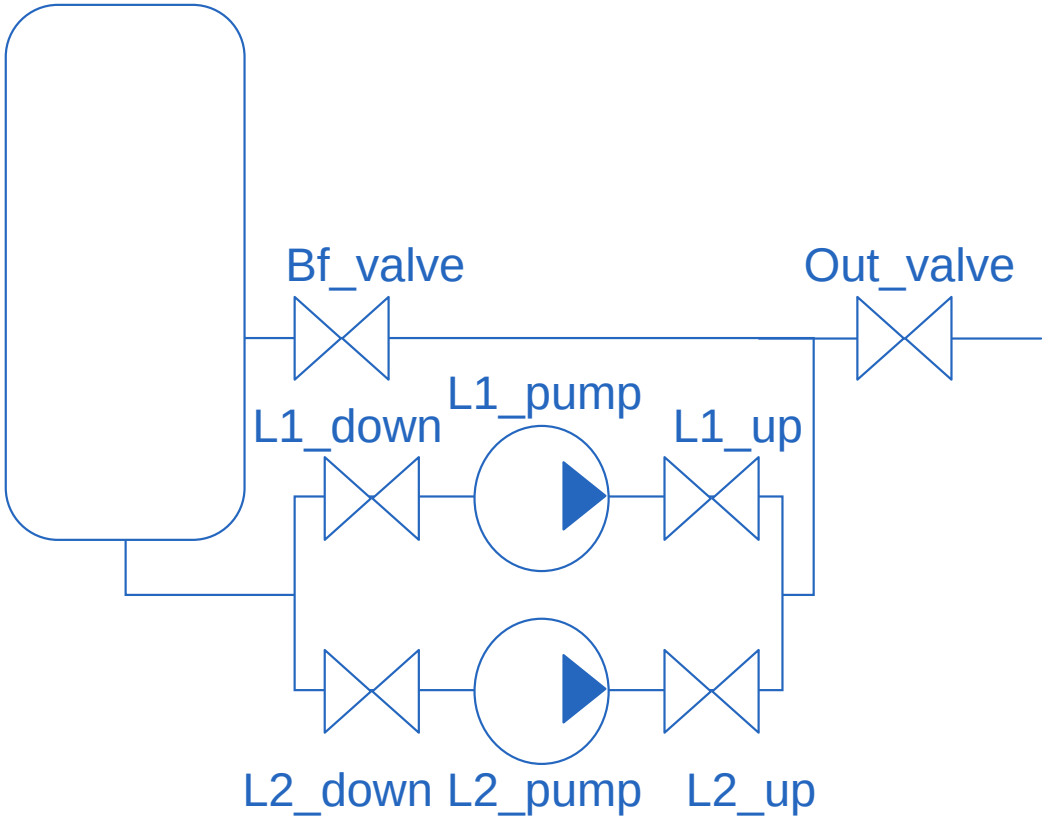
# Gestion d'un système de distribution d'eau

## Constitution du système

- un réservoir
- deux circuits de pompage (L1,L2) (une pompe et deux vannes)
- une vanne de distribution générale
- une vanne de refoulement

## Règles de fonctionnement

- Les deux pompes ne doivent jamais fonctionner simultanément.
- En cas de panne de la pompe principale, la pompe de secours doit assurer la distribution.
- Permutation des circuits toutes les 24h



# Propriétés à vérifier pour ce système (1)

## Propriété n°1

- Les 2 pompes ne doivent jamais fonctionner en même temps.

## Propriété n°2

- Une panne générale provoque l'arrêt des pompes et la fermeture des vannes.

## Propriété n°3

- Une panne d'une pompe provoque l'arrêt de cette pompe et la fermeture des vannes associées.

# Propriétés à vérifier pour ce système (2)

## Propriété n°4

- En cas de panne de la pompe principale, la pompe de secours doit assurer la distribution.

## Propriété n°5

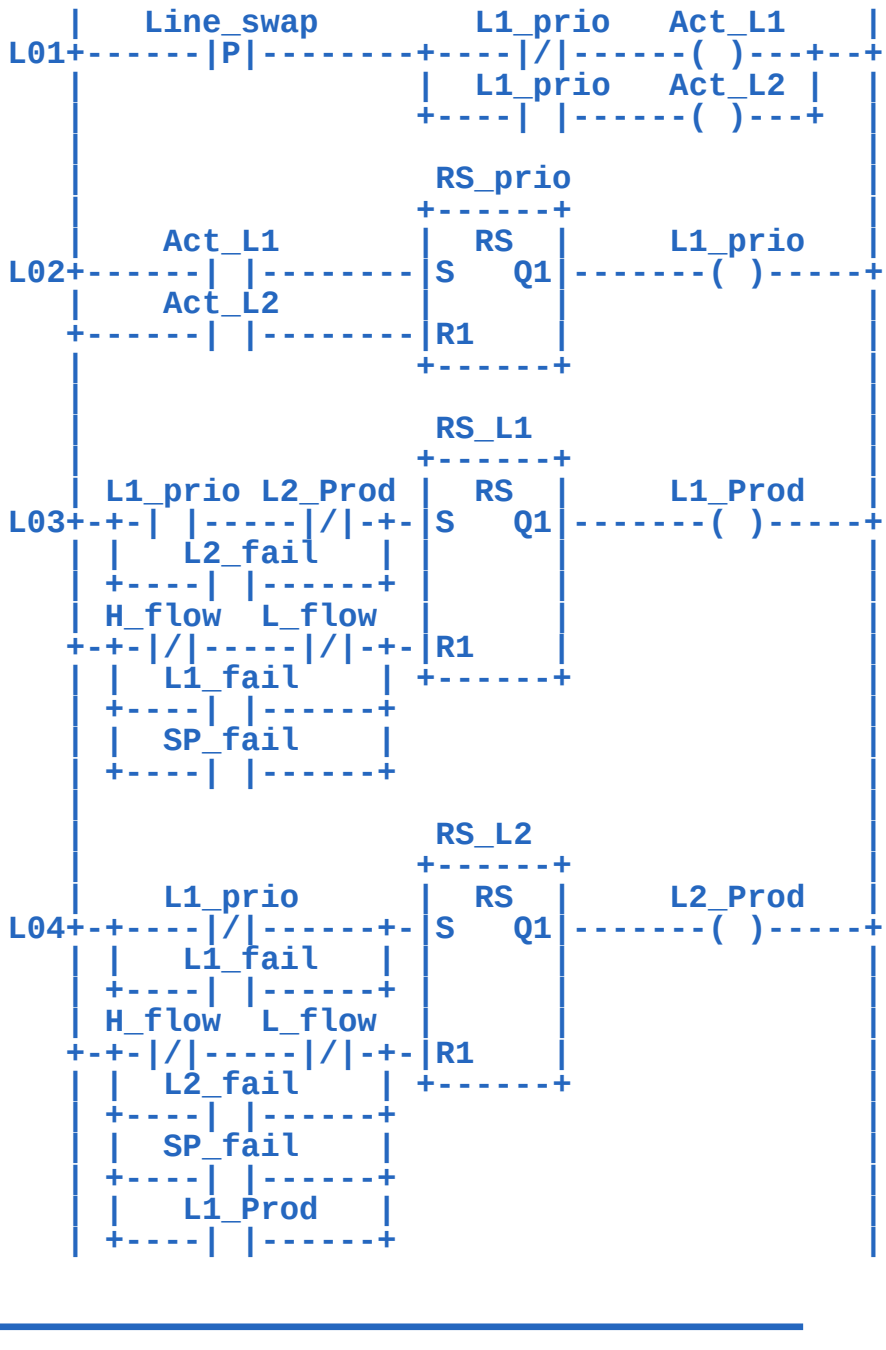
- Dans le cas d'une demande de distribution "bas_debit" seule, la vanne refoulement doit être ouverte.

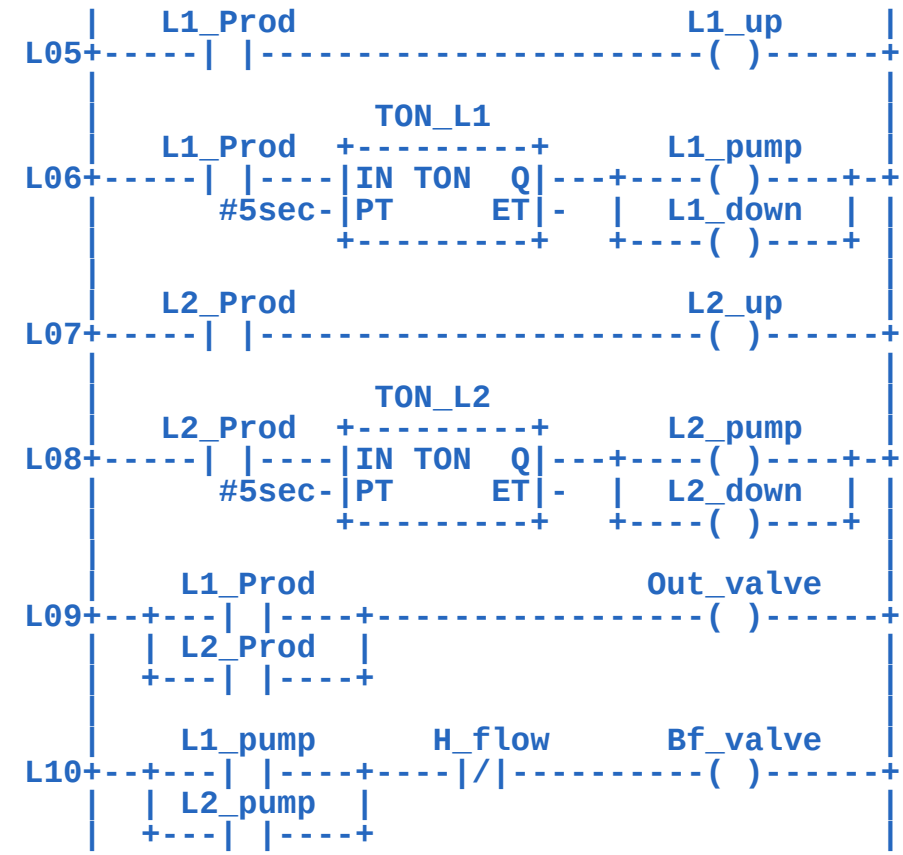
## Propriété n°6

- La mise en route d'une pompe ne peut se faire qu'après avoir ouvert la vanne en amont pendant 5s.

# Programme de commande

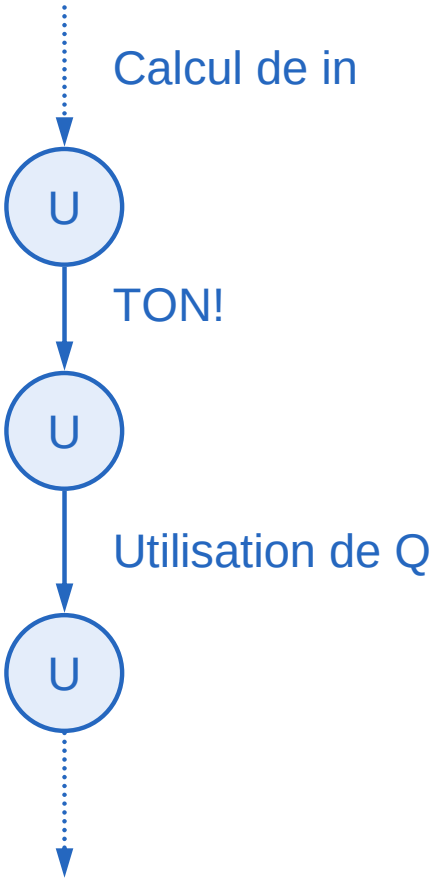
## Exemple

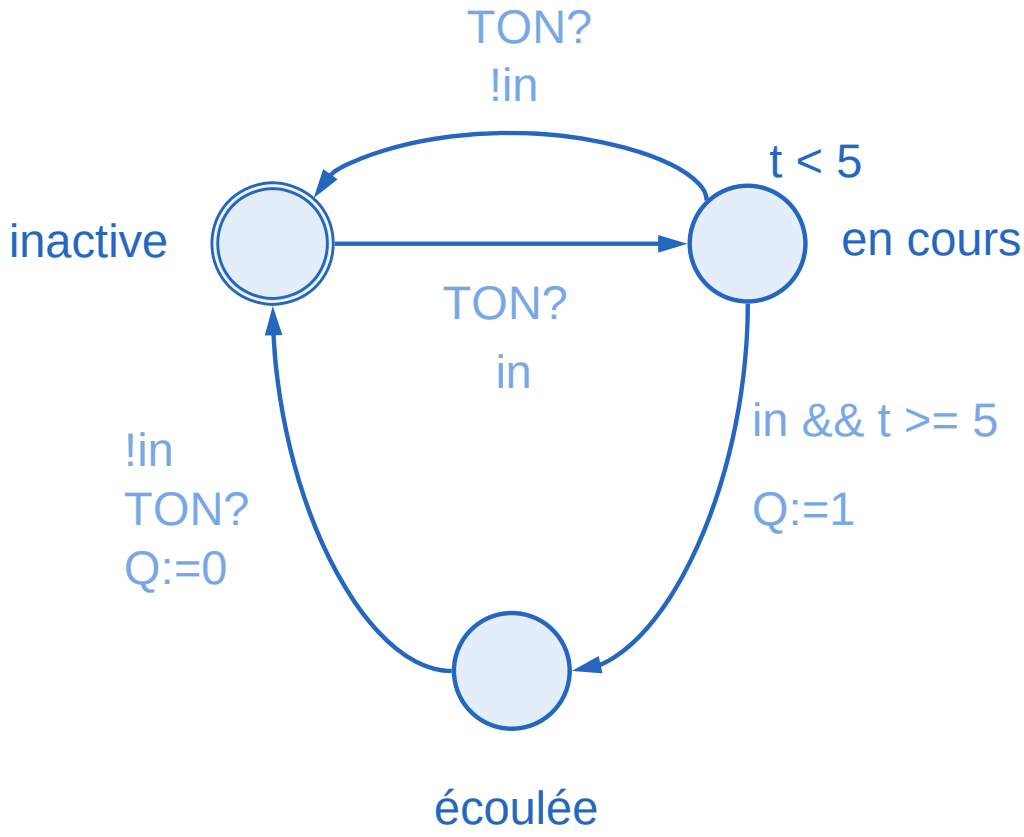



```
VAR_INPUT
  H_flow, L_flow, L1_fail : BOOL;
  L2_fail, Line_swap, SP_fail : BOOL;
END_VAR
VAR_OUTPUT
  Bf_valve, L1_down :   BOOL;
  L1_pump, L1_up, L2_down : BOOL;
  L2_pump, L2_up, Out_valve : BOOL;
END_VAR
VAR
  Act_L1, Act_L2, L1_prio : BOOL;
  L1_Prod, L2_Prod : BOOL;
  RS_prio, RS_L1, RS_L2 : RS;
  TON_L1, TON_L2 : TON
END_VAR
```

# Temporisation TON



# Traitement de cas

- **Vérifier, sur le modèle de contrôle de distribution d'eau donné, toutes les propriétés énoncées en langage naturel :**
  - Choix du type de vérification (propriété CTL seule ou avec un automate observateur),
  - Expression de la propriété CTL  et de l'observateur le cas échéant.
  - Expression de la propriété en langage UPPAAL,
  - Association avec les variables du modèle,
  - Vérification de la propriété formelle sur le modèle avec UPPAAL,
  - Conclusion.