SE206: Modélisation, génération de code et vérification

Introduction au model-checking avec UPPAAL

Etienne Borde,
inspiré d'un cours de Paul Petterson, MDH
etienne.borde@telecom-paristech.fr

Objectifs du cours

- Connaître les principes des techniques de model-checking
- Savoir modéliser un système embarqué avec des machines à états temporisé et synchronisées
- Savoir spécifier des propriétés logiques (simples) à vérifier
- Comprendre les mécanismes de base de l'évaluation de ces propriétés logiques

Plan du cours

1. Introduction et généralités
2. Présentation des automates temporisés
3. La logique CTL
4. Quelques mots sur l'outil UPPAAL

Introduction et généralités

Les principes du model-checking

P valide sur M équivaut à M ne satisfait pas non P

→ Le model-checker cherche un cas qui abouti à non P

- Types de propriété d'intérêt:
  - ⌘ Absence de deadlocks?
  - ⌘ Un état d'erreur est inaccessible?
  - ⌘ Le temps séparant un ensemble d'action est-il inférieur à une certaine valeur?
  - ⌘ La valeur de la variable x est-elle toujours comprise entre 12 et 24?
  - ⌘ …
- Il s'agit d'une vérification **exhaustive** (contrairement au tests)
- Mais sur un modèle (contrairement au tests)
- Le model-checking est une activité complémentaire au test qui peut être mené en avance de phase (avant l'implémentation)

Dans ce cours, on utilisera UPPAAL

- Outil de model checking très populaire (bien documenté, robuste, stable, permet l'usage de fonctions « C », automates paramétrables …)

- Basé sur la notion d'automates temporisés synchronisés

- Les propriétés à vérifier sont formalisés en logique CTL (*Computation Tree Logic)*

- *Beaucoup d'autres outils et formalismes existent: automate probabilistes, réseaux de Petri (temporisé, probabilistes)…*

Présentation des automates temporisés

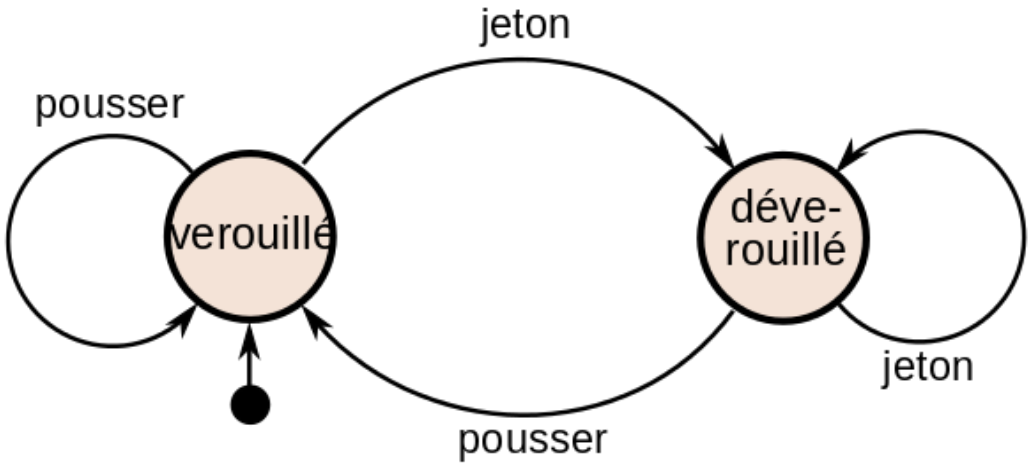
Automates temporisés…

- Un automate temporisé est la conjonction de 3 constructions
  - ⌘ Un automate à états finis
  - ⌘ Des contraintes d'horloge
  - ⌘ Des réinitialisations d'horloges

- Présentons ces trois constructions (voir slides suivants)

Automate à états finis

- C'est un simple graphe orienté, étiqueté <S, L, T>
  - S = Ensemble de nœuds (états) avec un état initial s_0
  - L = Ensemble d'étiquettes (simples identifiants).
  - T = Ensemble d'arcs (transitions). Chaque transition est définie par un tuple <s, l, s'> (source, étiquette, cible)
- Représentation (exemple du tourniquet wikipedia©)



- Objectif: représenter des séquences, potentiellement infinies, d'actions
- Problème pour les systèmes embarqués: la prise en compte du temps. i.e. pas déverrouiller le portique plus de 30 secondes

Les horloges dans les automates temporisés

- Une horloge est une représentation de l'écoulement du temps
- La valuation des horloges est initalement à 0
- Chaque horloge évolue linéairement avec une pente de 1…

Contraintes d'horloge, définition

- Formellement (et en anglais ;))

For set C of clocks with $x, y \in C$, the set of *clock constraints* over C, $\Psi(C)$, is defined by

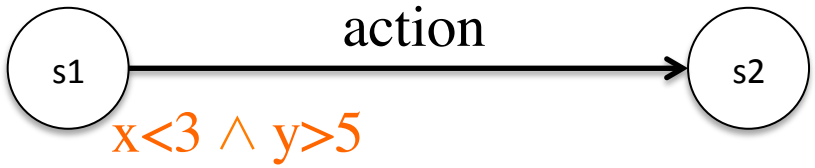
$$\alpha ::= x \prec c \mid x - y \prec c \mid \neg\alpha \mid (\alpha \wedge \alpha)$$

where $c \in \mathbb{N}$ and $\prec \in \{<, \leqslant\}$.

- Moins formellement, il s'agit de contraintes logiques avec des opérateurs plutôt simples ($\wedge$, <,≤…) sur des valeurs entières des horloges

Contraintes d'horloge sur les transition (gardes)

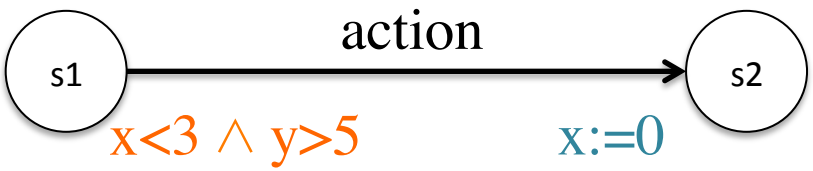
- Représentation, en considérant deux horloges x et y:



- Il s'agit d'une garde: la transition de s1 vers s2 n'est franchissable que si la valuation de x est strictement inférieure à 3 et la valuation de y est strictement supérieure à 5.

Réinitialisation d'horloge

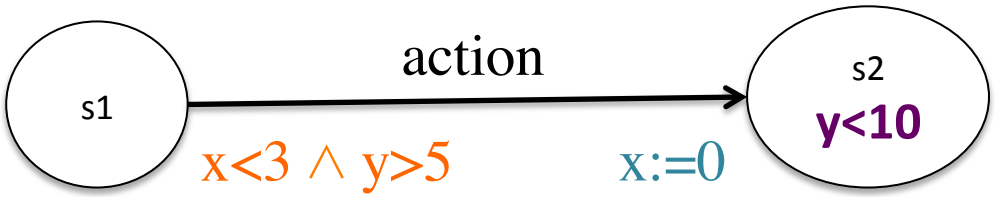
- Il s'agit d'une simple action associée à la transition



- Lorsque la transition est franchie, l'horloge x est réinitialisée (sa valuation devient égale à 0 en atteignant l'état s2)

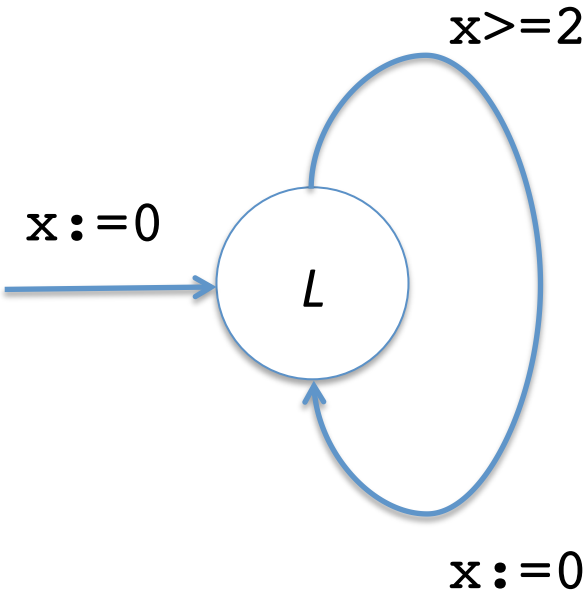
Contraintes d'horloge sur les états (invariants d'états)

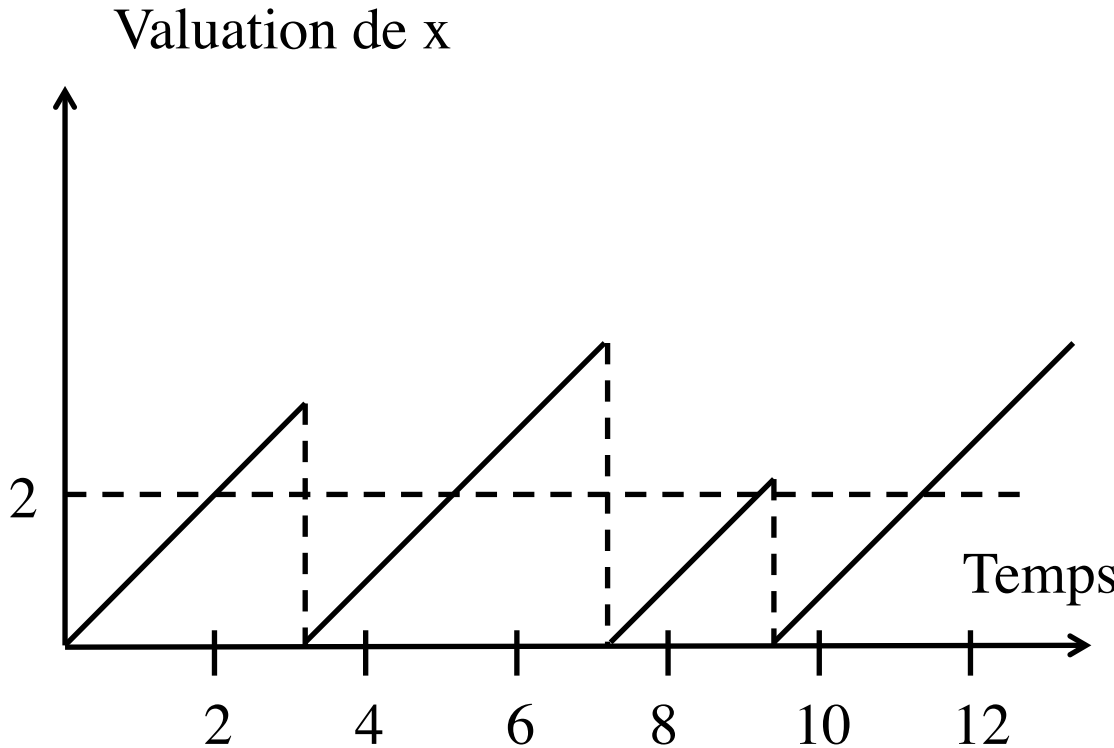
- Représentation:



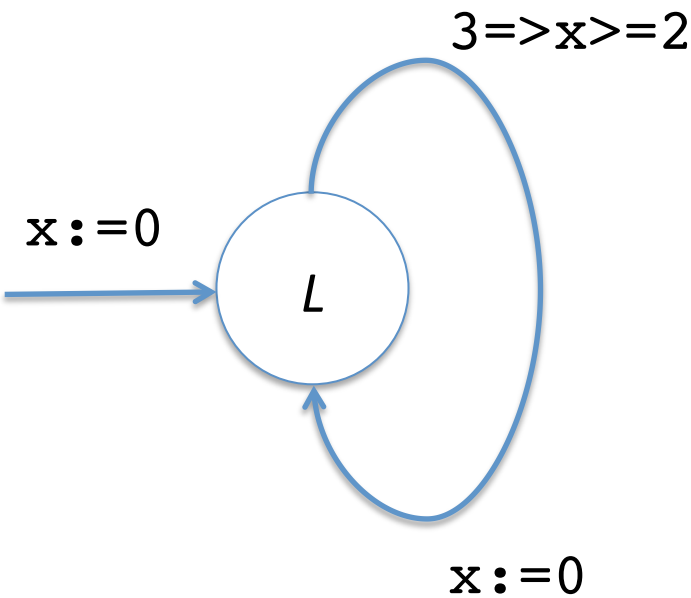
- Il s'agit d'un invariant: le système peut être dans l'état s2 si et seulement si la contrainte d'horloge est respectée (y<10).

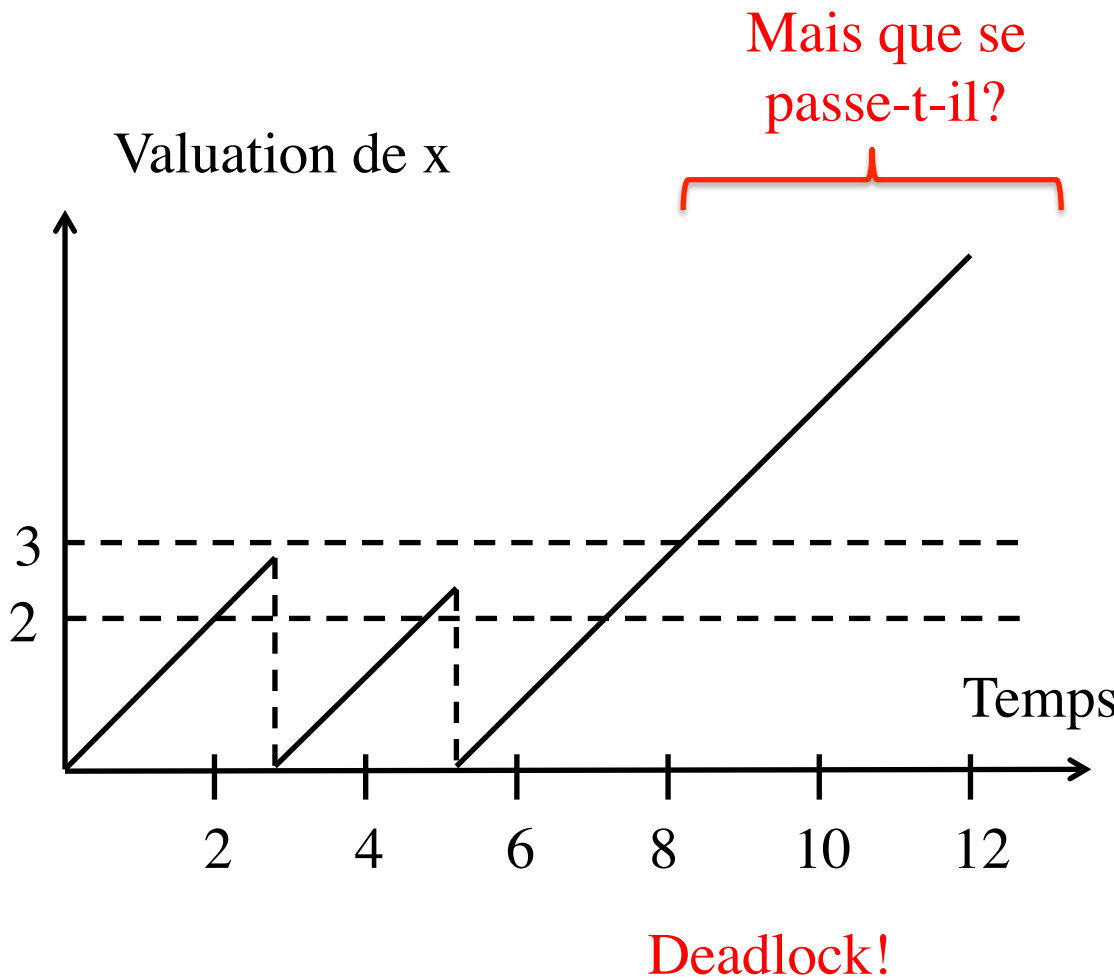
Quelques exemples simples



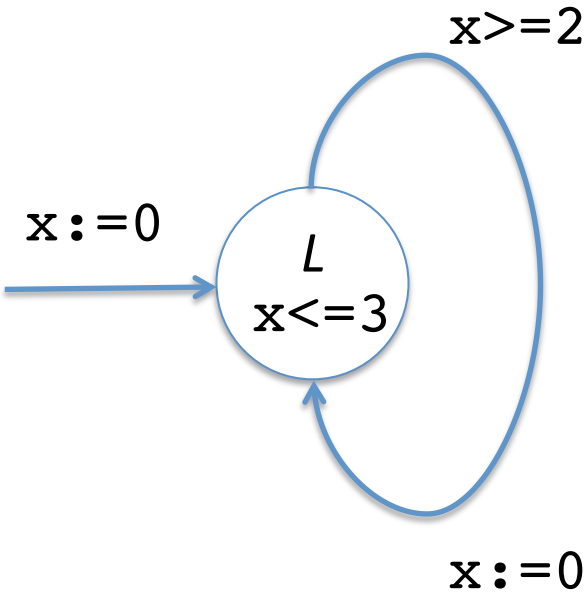


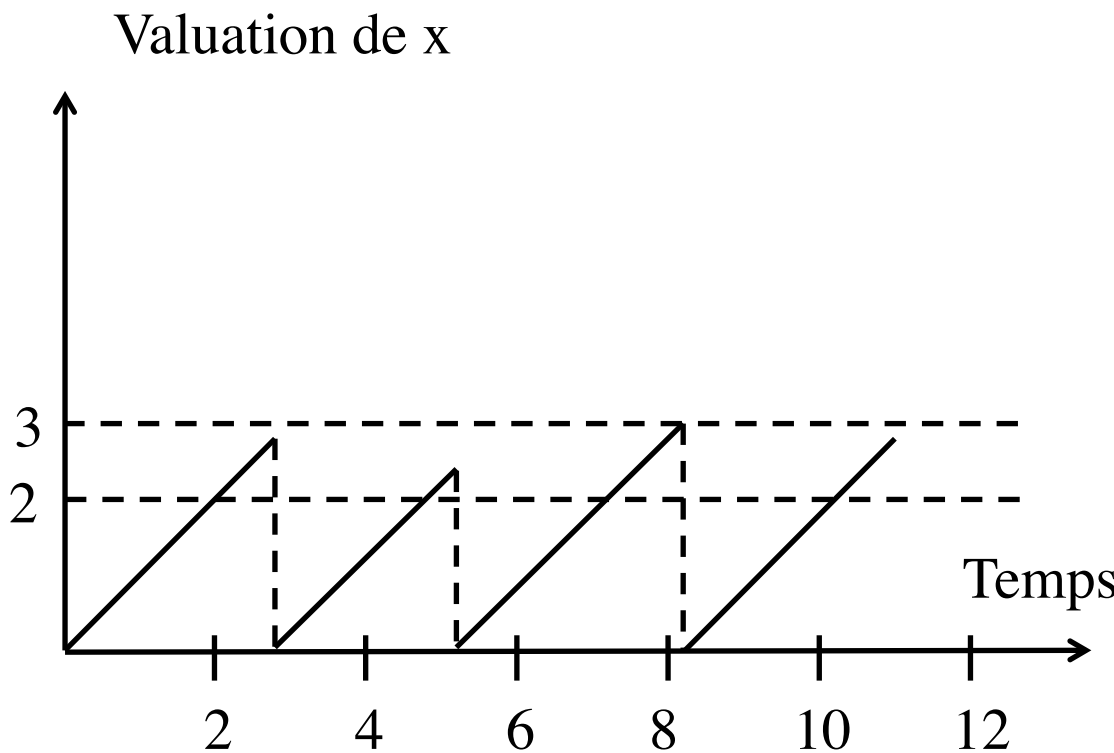
Quelques exemples simples



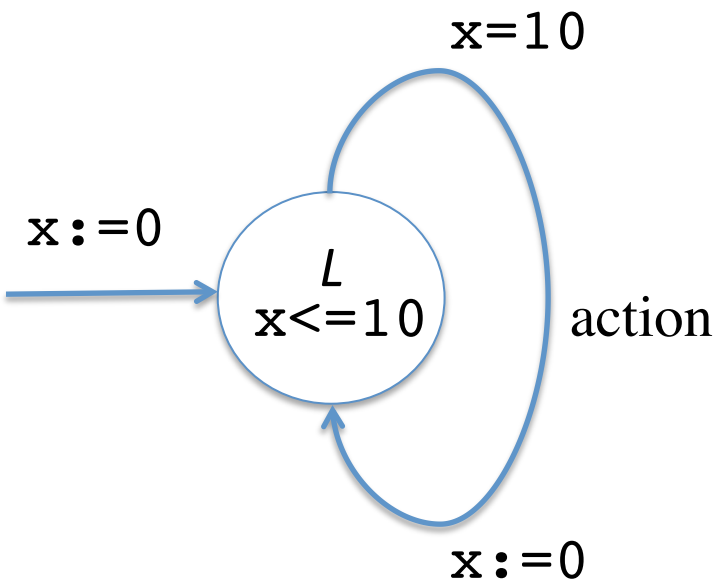


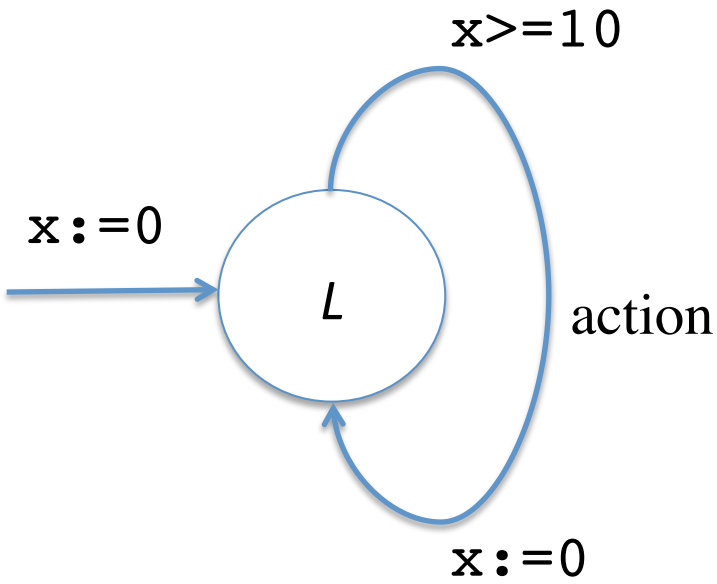
Usage des invariants: assurer la progression de l'automate





Actions périodiques et sporadiques



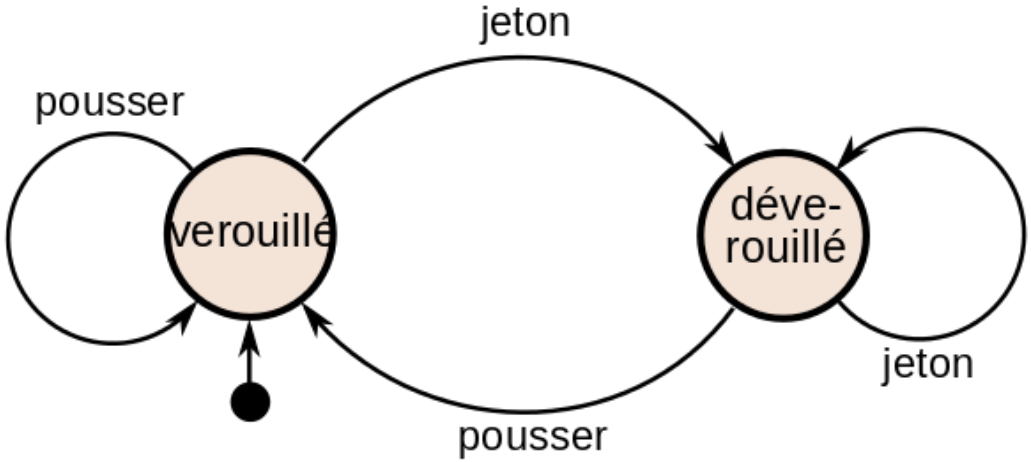


<u>Actions périodiques:</u> la transition étiquetée « action » est franchie toutes les 10 unités de temps

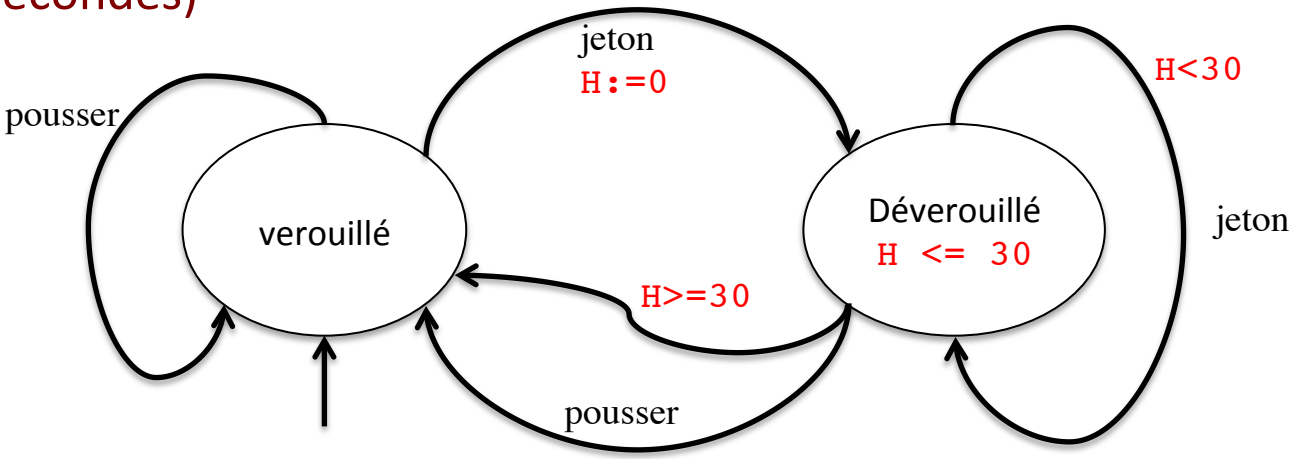
<u>Actions sporadique:</u> la transition étiquetée « action » est franchie avec au moins 10 unités de temps entre deux franchissements

Automate à états finis

- Retour sur l'exemple du tourniquet (wikipedia©)



- Exigence temporelle: pas laisser le portique déverrouillé plus de 30 unités de temps (e.g. secondes)



La logique CTL

Spécification des propriétés d'intérêt

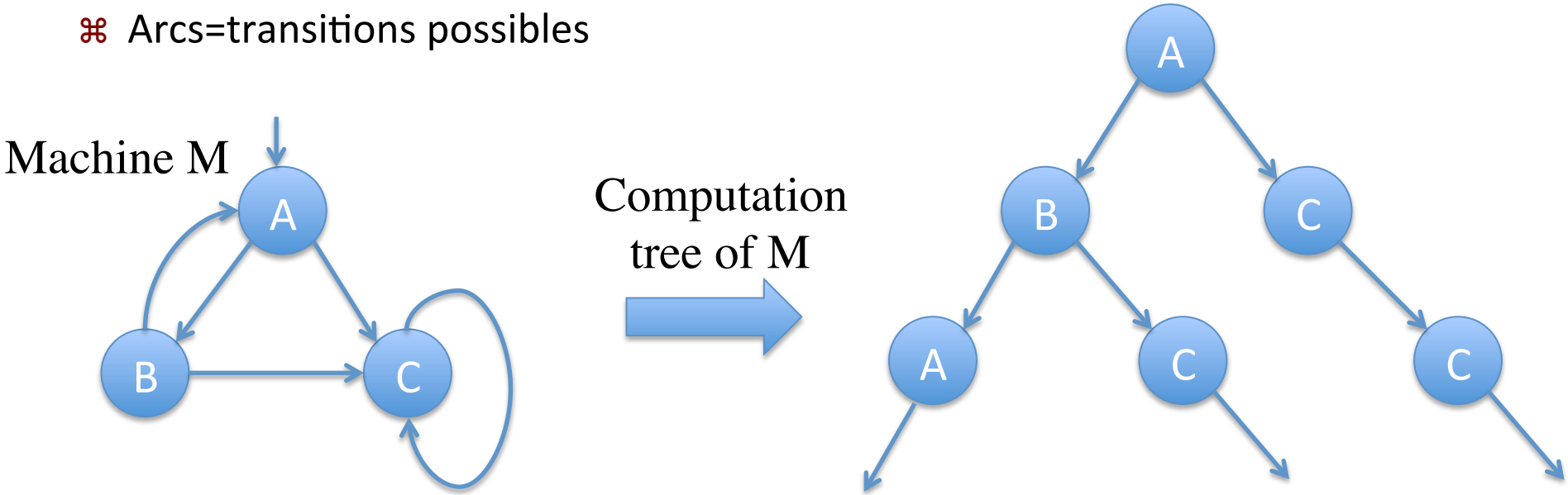
- Petit rappel (principes du model checking):

- Modèles considérés: automates temporisés présentés jusque là (on verra une syntaxe dédiée lors de la présentation de l'outil UPPAAL)

- Expression de propriétés considérées: CTL (Computation Tree Logic)

CTL, intuition…

- Computation tree:
  - ⌘ Noeud de l'arbre=état+valuation **des horloges et variables**
  - ⌘ Arcs=transitions possibles



- CTL: formules logiques sur les chemin de l'arbre « Computation Tree », via les opérateurs (liste non-exhaustive)
  - ⌘ E: il existe un chemin (E → Exists)
  - ⌘ A: pour tous les chemin (A → All)
  - ⌘ G: tous les état du chemin (G → Globally)
  - ⌘ F: un état du chemin (F → Finally)

CTL, syntaxe (sous-ensemble)

- Une propriété est définie en utilisant la syntaxe suivante:

  `Φ ::= p | not Φ | Φ ∨ Φ | Φ => Φ | AG Φ | AF Φ | EG Φ | EF Φ`

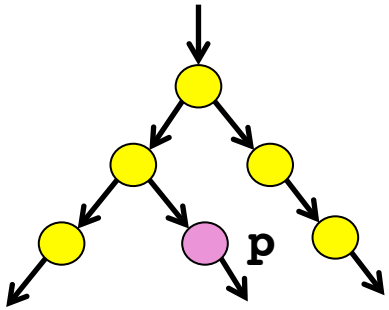
  - ou p est une proposition logique atomique (on détaillera pour UPPAAL à la fin du cours)

- En résumé, il s'agit d'une combinaison
  - de la logique propositionnelle, c.à.d. basée sur des variables logiques (du type « l'automate est dans l'état S », « la valeur de l'horloge H<30 »), des opérateurs simples : *not, or, and, imply,* et la constante *FALSE*.
  - de combinaisons d'opérateurs sur le « computation tree » (*AG, AF, EG, EF*)

Exemples de propriétés CTL, s'appuyant sur une propriété p

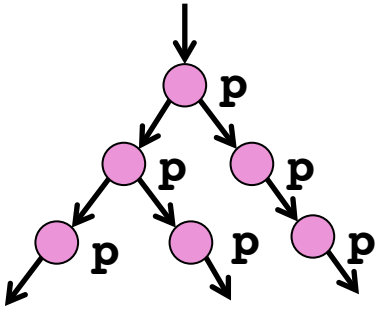
- Etat atteignable (*reachability*)

EF p



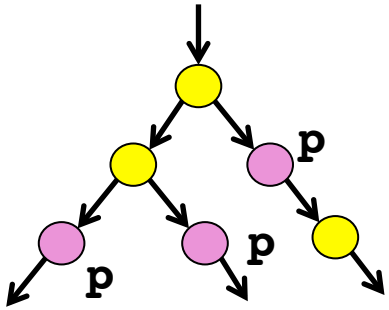
- Propriété toujours vraie (*invariant*)

AG p



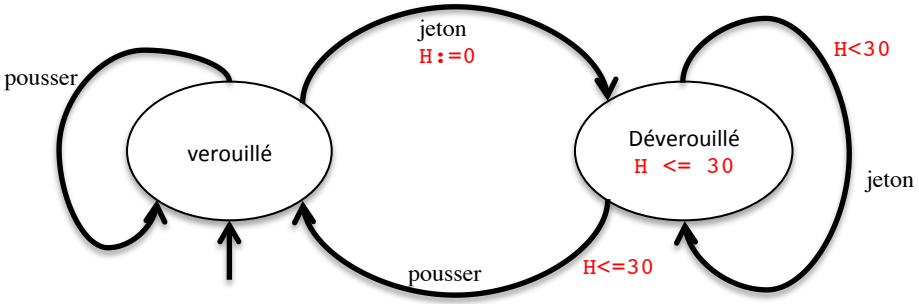
- Propriété inévitable

AF p

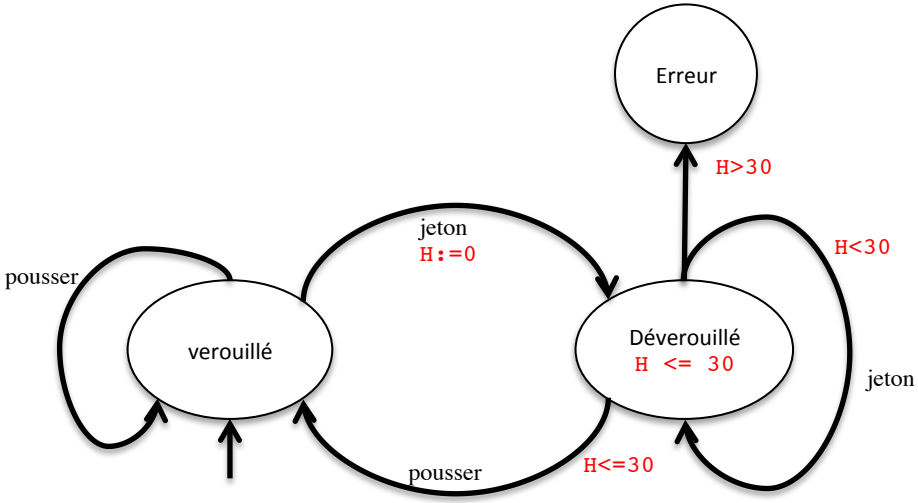


Note sur la notion d'observateur

- Il est souvent plus simple (mais pas toujours possible) de compléter un automate avec des états d'observation (ou observateurs) pour vérifier une propriété



- Exigence temporelle: pas laisser le portique déverrouillé plus de 30 secondes



Etat Erreur atteignable?
→ Comment le spécifier?

EF Erreur

Résultat attendu: false

Point sur la vérification

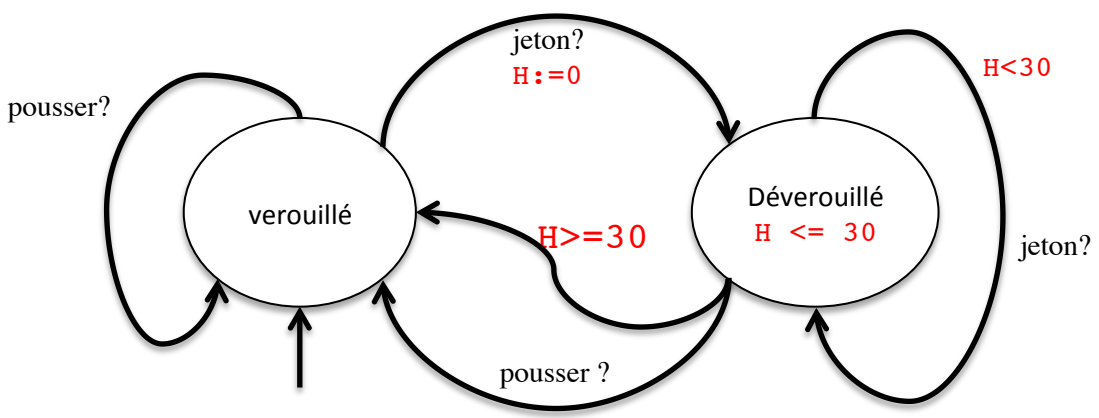
- On peut vérifier des propriétés de
  - ⌘ Accessibilité d'une configuration d'états, d'un ensemble d'actions
  - ⌘ Sûreté "Quelque chose de mauvais n'arrive jamais"
  - ⌘ Vivacité "Quelque chose de bon finit par arriver"
- Comment?
  - ⌘ Propriétés exprimées en CTL (A|E, F|G)
  - ⌘ Et/ou via des observateurs

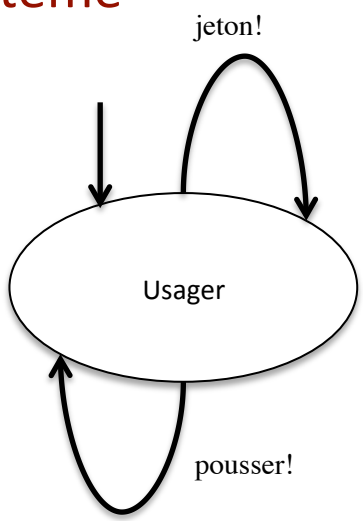
- Model-checking: modèle formel (mathématiquement bien définit) + langage d'expression de propriétés + algo/outils de vérification dédiés.

Quelques mots sur l'outil UPPAAL et les produits d'automates

Les labels et action sous UPPAAL

- Associées aux transitions, UPPAAL permet de réaliser des actions assez classiques: incréments de **variables**, réinitialisation d'horloge, appel de fonctions (syntaxe proche du C)
- Les labels associés aux transitions sont représentés via des *channels*, en mode émission ou réception;
  - ⌘ Notez que les deux sont « bloquants »: une transition avec une émission ne peut être franchie **QUE** si une transition avec une réception est franchie en même temps (communication dite « par poignée de main »).
  - ⌘ Seul intérêt de l'identification du mode: le *broadcast channel*.
- Type d'usage: modélisation de l'environnement du système





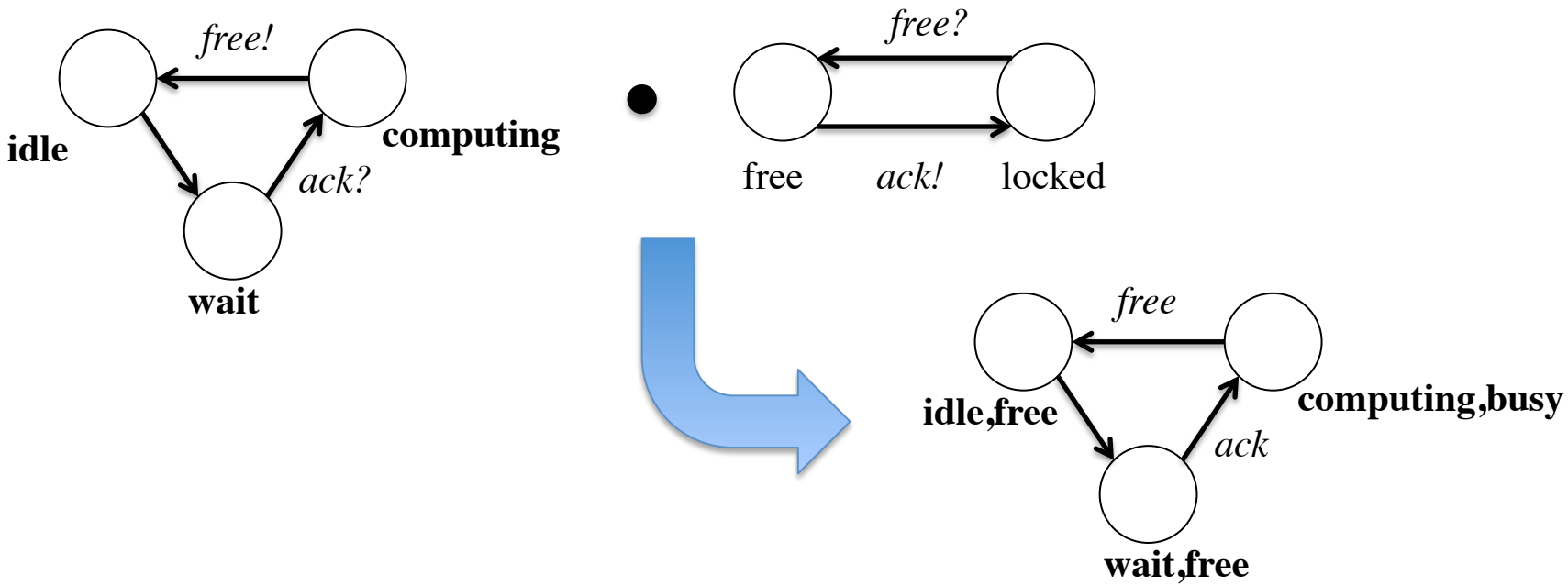
Produit synchronisé d'automates

- Les *channels* sont les labels dans la définition suivante du produit synchronisé d'automates.
- Définition
  - Soient M_1 et M_2 deux automates avec $S_1 \cap S_2 = \emptyset$
  - Le produit synchronisé $M (<S, L, T>) = M_1 \bullet M_2$ est définit par:
    - $L = L_1 \cup L_2$
    - $(s_{01}, s_{02}) \in S \subset S_1 \times S_2$
    - S et T sont les + petits ensembles vérifiant :

Si $(s_1,s_2) \in S$ et $(s_1,l,s_1') \in T1$ avec $l \notin L_1 \cap L_2$	Alors $(s'_1,s_2) \in S$ et $((s_1,s_2),l,(s'_1,s_2)) \in T$
Si $(s_1,s_2) \in S$ et $(s_2,l,s_2') \in T2$ avec $l \notin L_1 \cap L_2$	Alors $(s_1,s'_2) \in S$ et $((s_1,s_2),l,(s_1,s_2')) \in T$
$(s_1,s_2) \in S$ et $(s_2,l,s_2') \in T_2$ et $(s_1,l,s_1') \in T_1$ avec $l \in L_1 \cap L_2$	Alors $(s'_1,s'_2) \in S$ et $((s_1,s_2),l,(s_1',s_2')) \in T$

Exemple de produit synchronisé

- On ne reprend pas ici l'exemple du tourniquet, trop simple
- Prenons celui du verrou (simplifié)



Note: le ? Ou ! N'est utile que pour un channel « broadcast » (qui change un peu la définition formelle du produit synchronisé mais pas le principe)

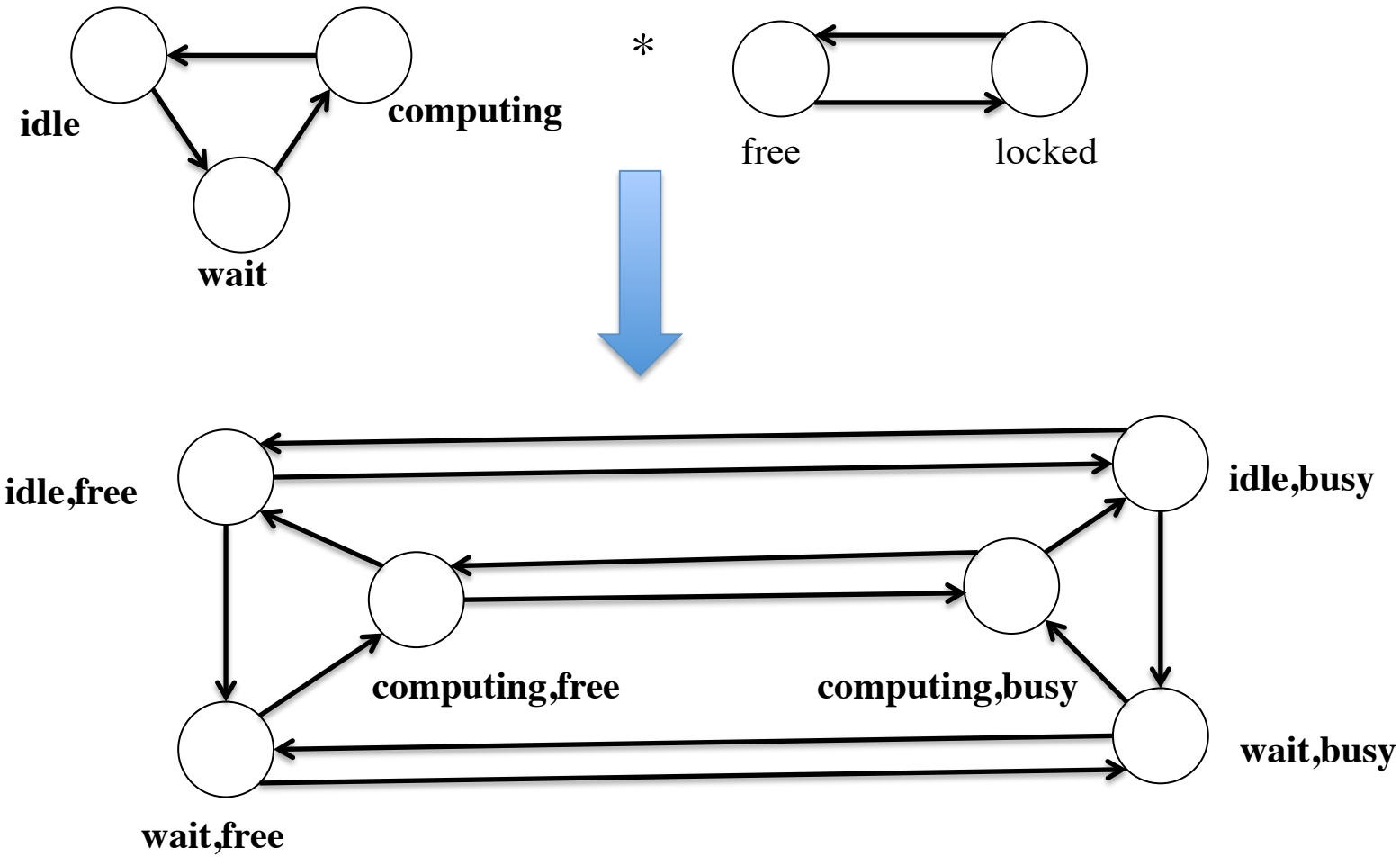
Produit libre d'automates

- Pour les transition sans channels, UPPAAL réalise un produit libre d'automates
- Définition
  - Soient M_1 et M_2 deux automates avec $S_1 \cap S_2 = \emptyset$
  - Le produit libre M (<S, L, T>) = $M_1 * M_2$ est définit par:
    - $L = L_1 \cup L_2$
    - $(s_{01}, s_{02}) \in S \subset S_1 \times S_2$
    - S et T sont les + petits ensembles vérifiant :

Si $(s_1,s_2) \in S$ et $(s_1,l,s_1') \in T_1$	Alors $(s'_1,s_2) \in S$ et $((s_1,s_2),l,(s'_1,s_2)) \in T$
Si $(s_1,s_2) \in S$ et $(s_2,l,s_2') \in T_2$	Alors $(s_1,s'_2) \in S$ et $((s_1,s_2),l,(s_1,s'_2)) \in T$

Exemple de produit libre

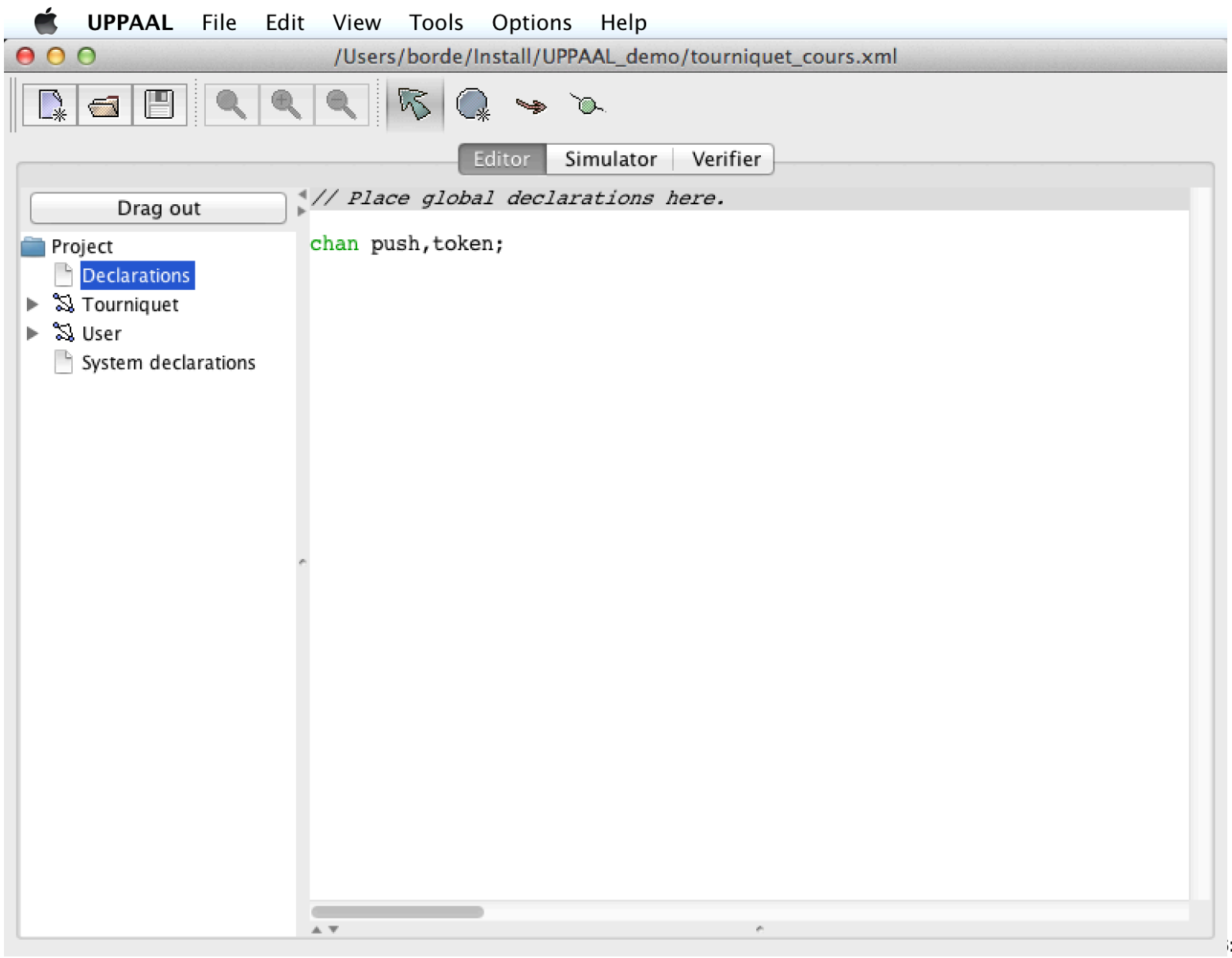
- Toujours avec l'exemple du verrou (simplifié)



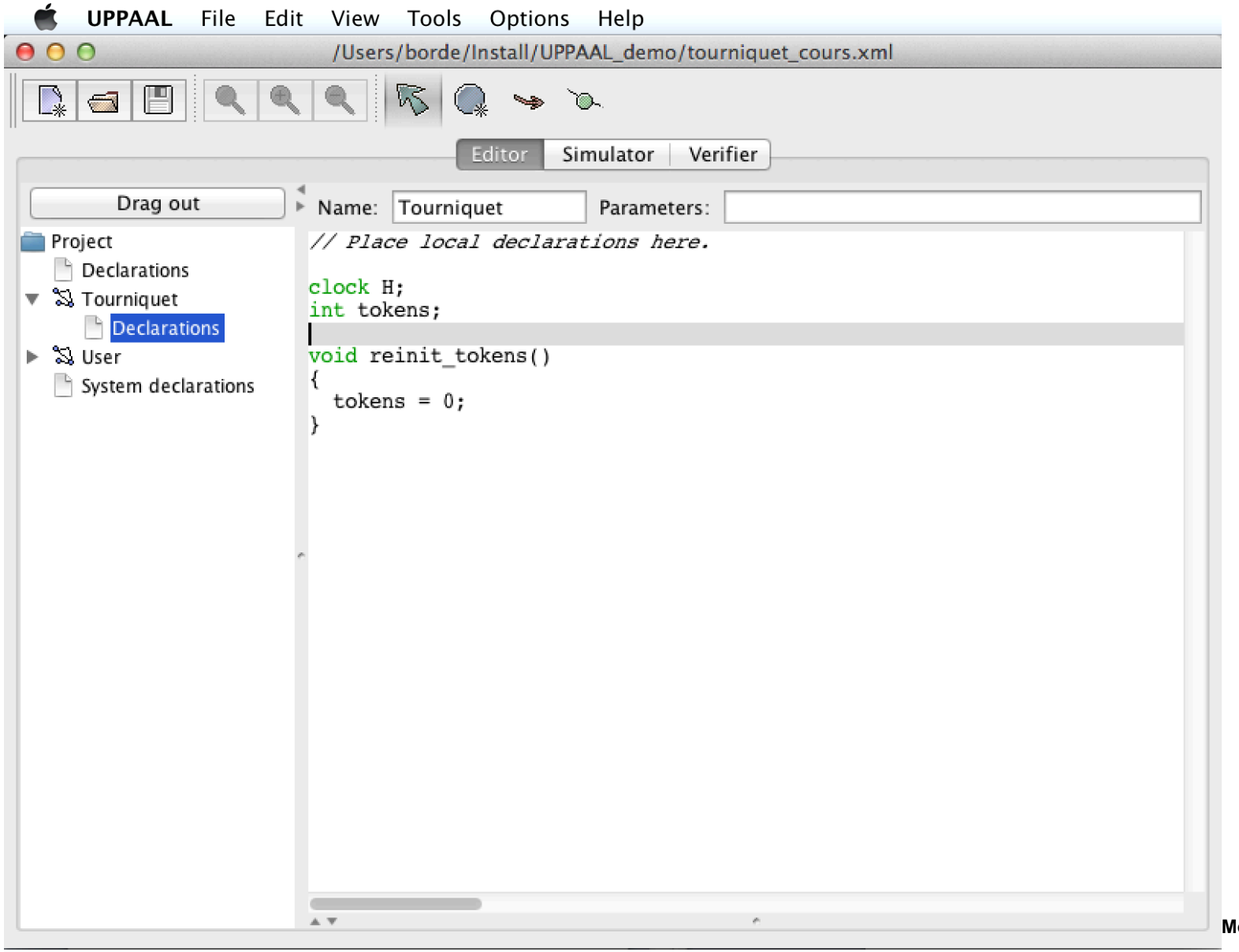
A propos de l'outil…

- Tutoriel pour prise en main: http://www.it.uu.se/research/group/darts/uppaal/small_tutorial.pdf
- L 'outil fourni un menu « Help » très complet, car il contient toute la documentation relative à l'utilisation de l'outil:
  - GUI Reference: une présentation de l'interface utilisateur (description des menus, des actions associées…)
  - Language reference: présentation des constructions supportées par le langage (à la fois la syntaxe et la sémantique):
    - Reserved keyword: décrit l'ensemble des mots-clés de la syntaxe supporté par UPPAAL
    - System Description : décrit l'instanciation du **réseau d'automates** UPPAAL. La notion d'instanciation ici est proche de celle des langages de description d'architecture (AADL: type,implementation/instance de composant) ou des langages orientés objets (Java: classe/objet)
    - Requirements : décrit le sous-ensemble de CTL supporté par UPPAAL
    - Expressions : décrit les expressions (logiques, arithmétiques) utilisables dans les états et les transitions des automates.
- L'aide donne de nombreuses et précieuses information! La suite de la présentation de l'outil est loin d'être exhaustive, se référer à l'aide pour gagner en autonomie.

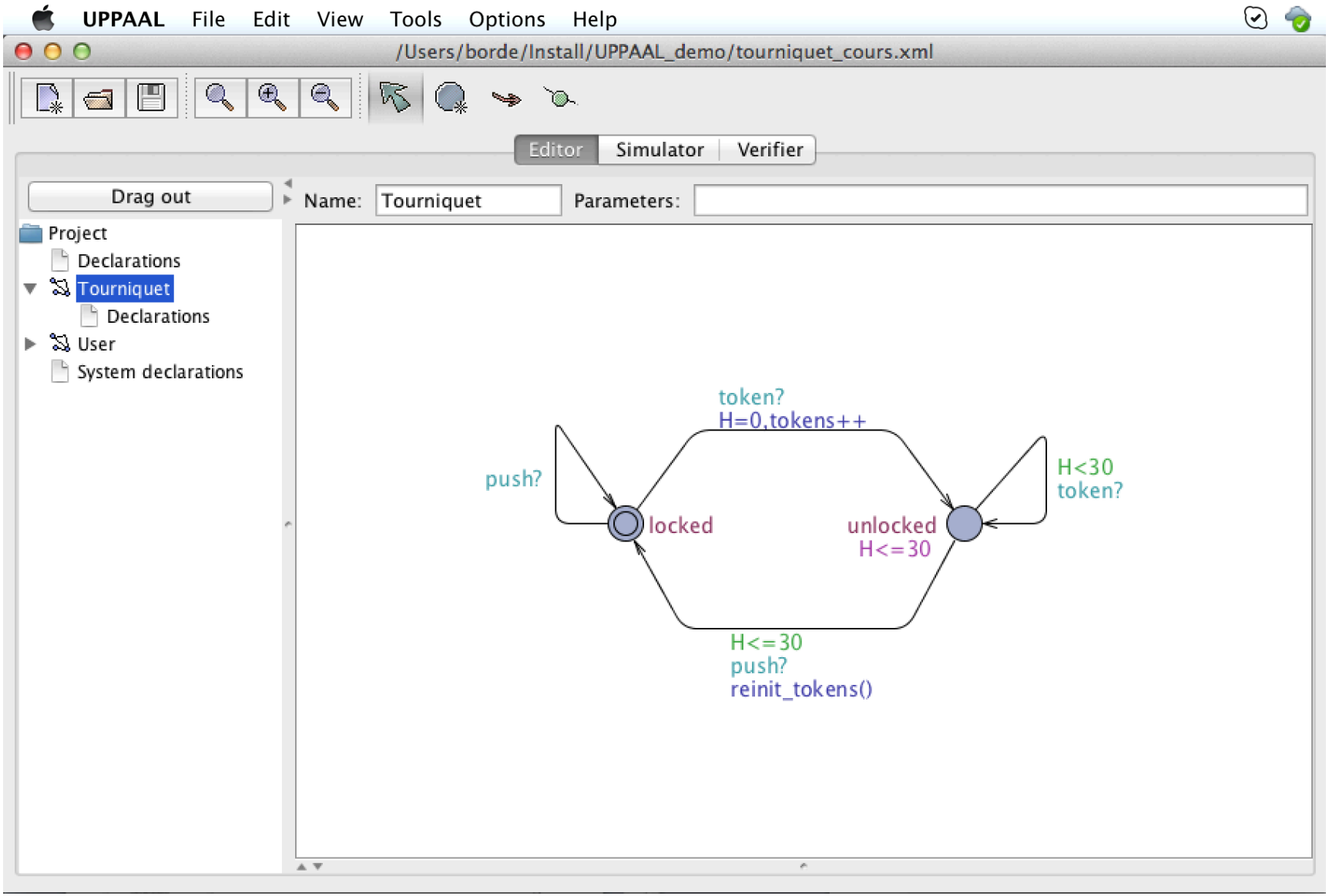
Exemple du tourniquet: déclarations globales



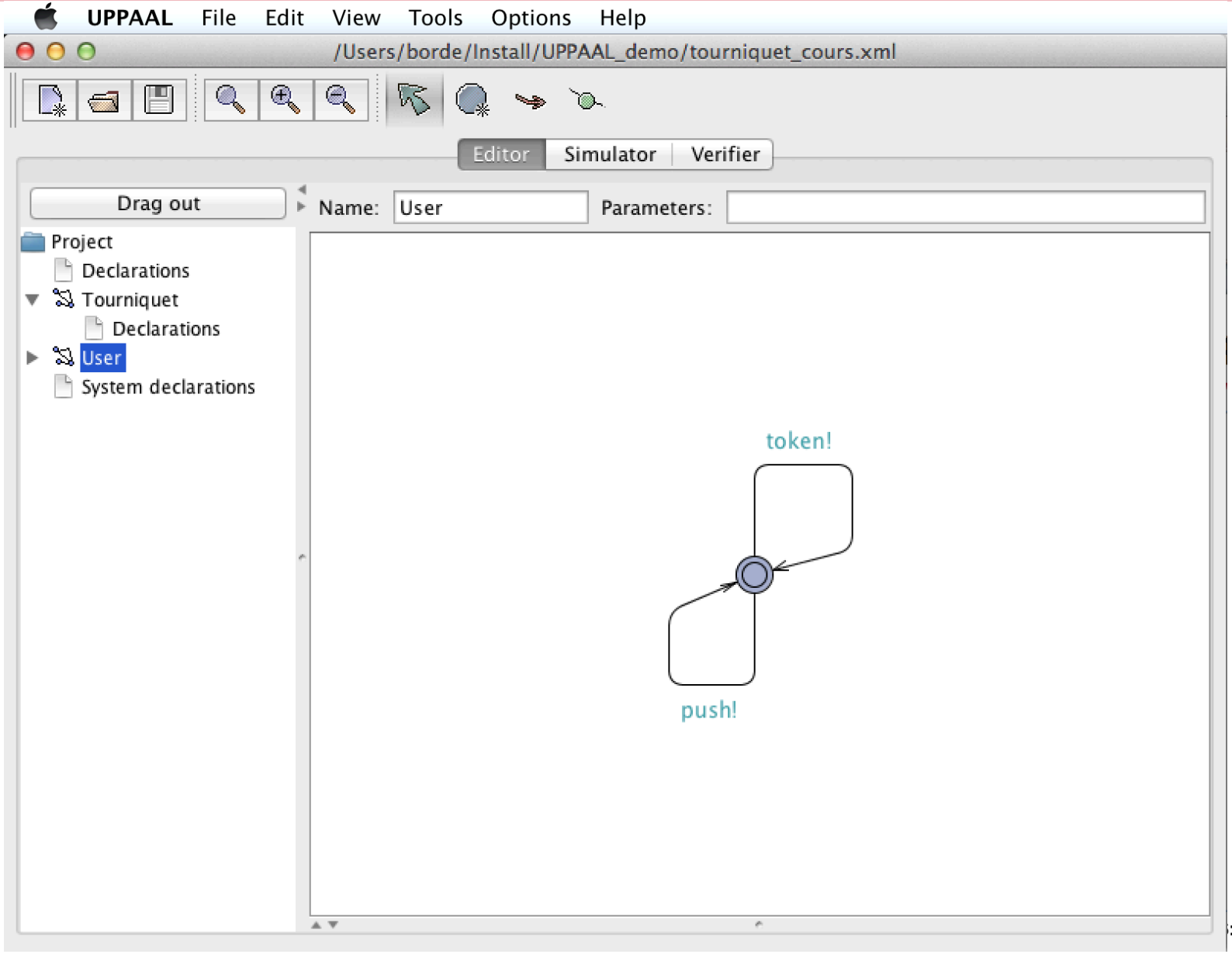
Exemple du tourniquet: déclarations locales



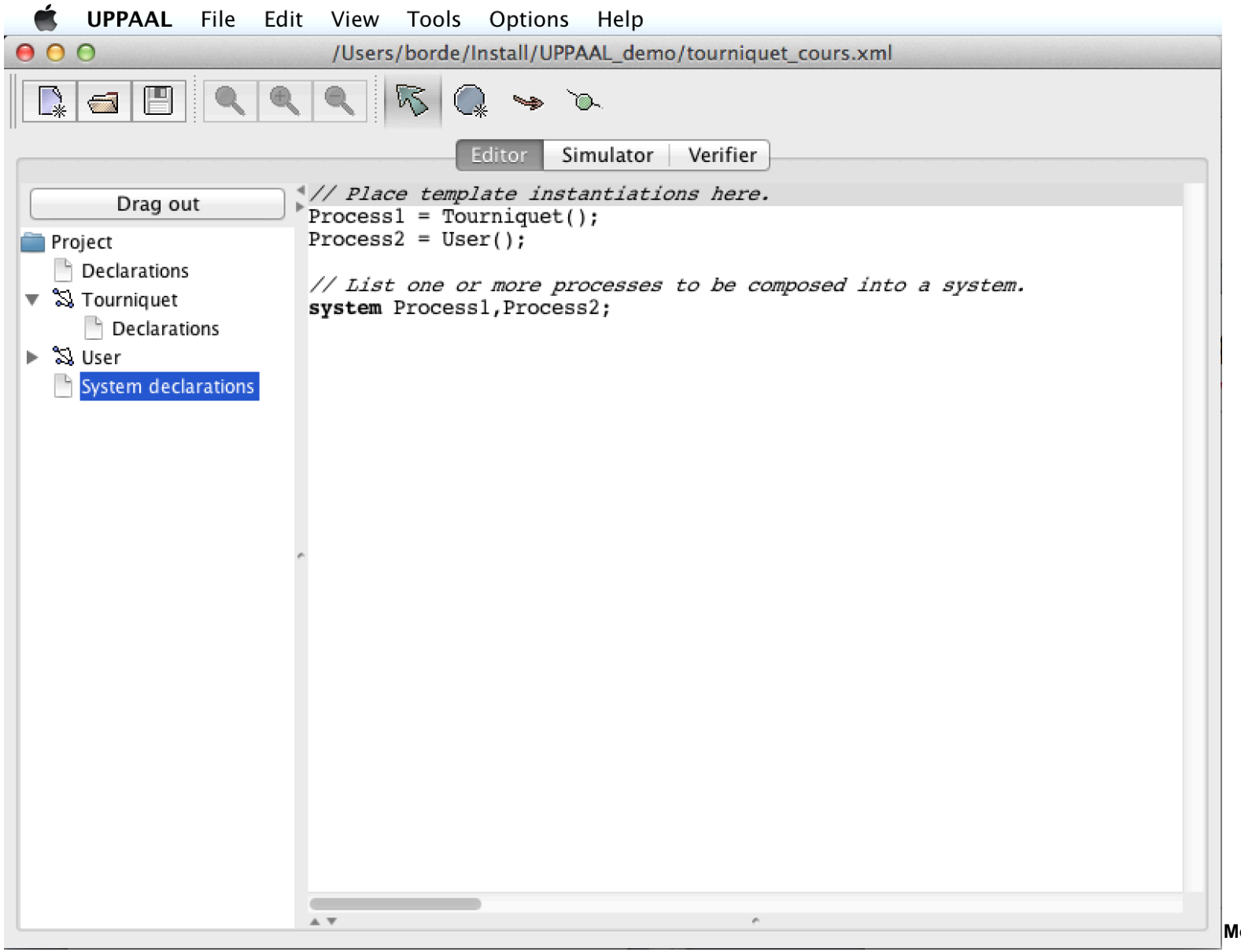
Exemple du tourniquet: automate tourniquet



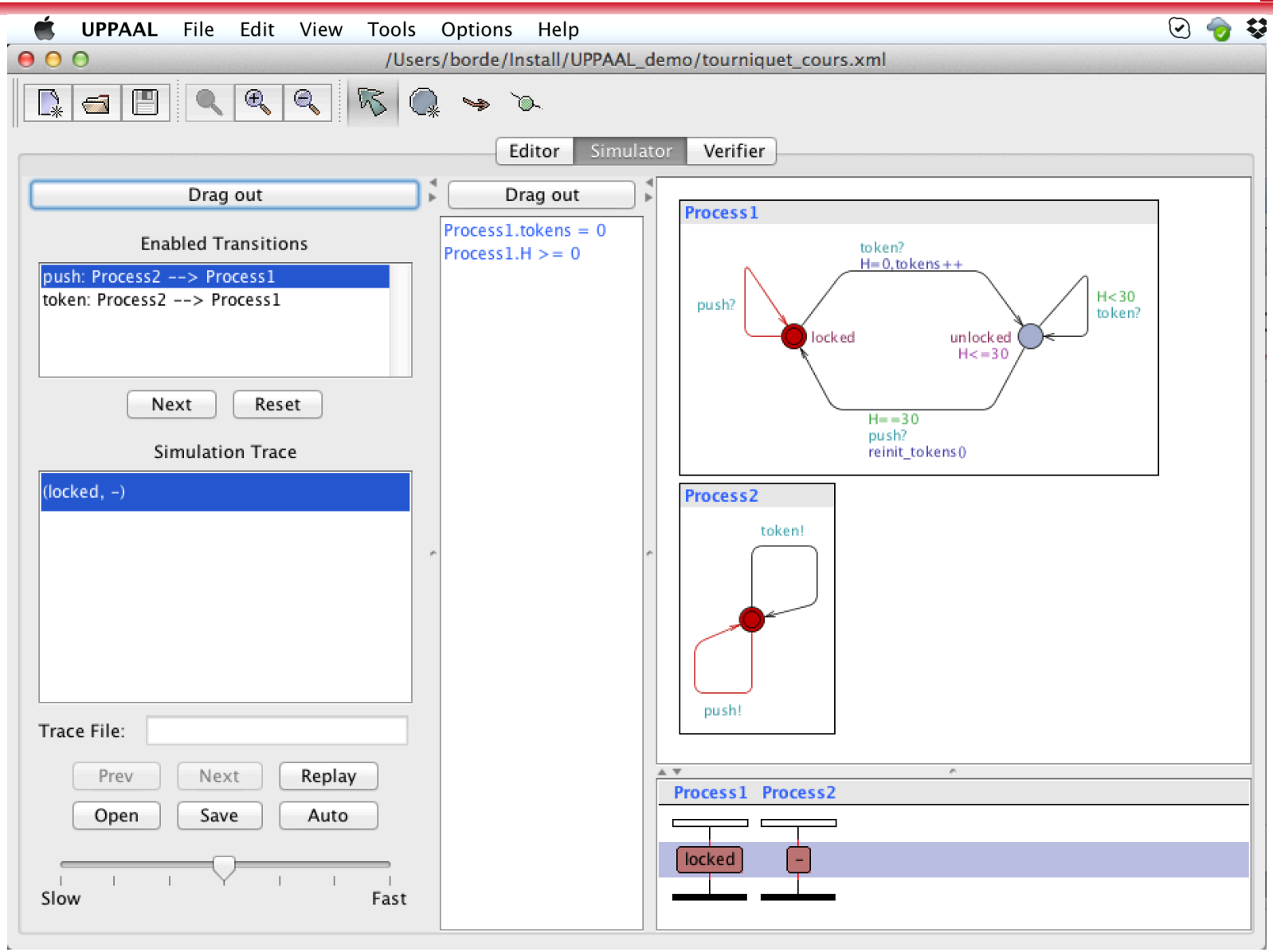
Exemple du tourniquet: automate user



Exemple du tourniquet: Instanciation du réseau d'automates



Exemple du tourniquet: simulation du réseaux d'automates



Exemple du tourniquet: vérification

UPPAAL File Edit View Tools Options Help

/Users/borde/Install/UPPAAL_demo/tourniquet_cours.xml

Editor | Simulator | Verifier

Overview

```
A[] not deadlock
A[] Process1.unlocked imply Process1.H<=30
```

Check
Insert
Remove
Comments

Query

A[] not deadlock

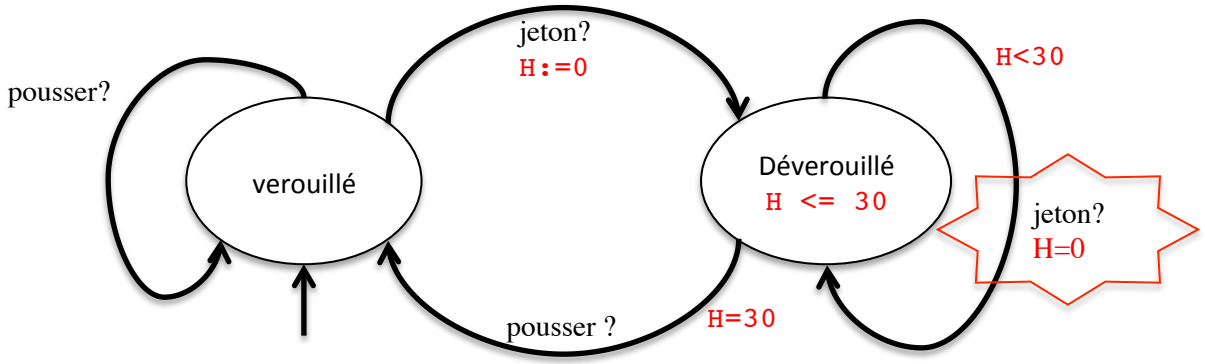
Comment

Status

A[] not deadlock
Property is satisfied.
A[] Process1.unlocked imply Process1.H<=30
Property is satisfied.

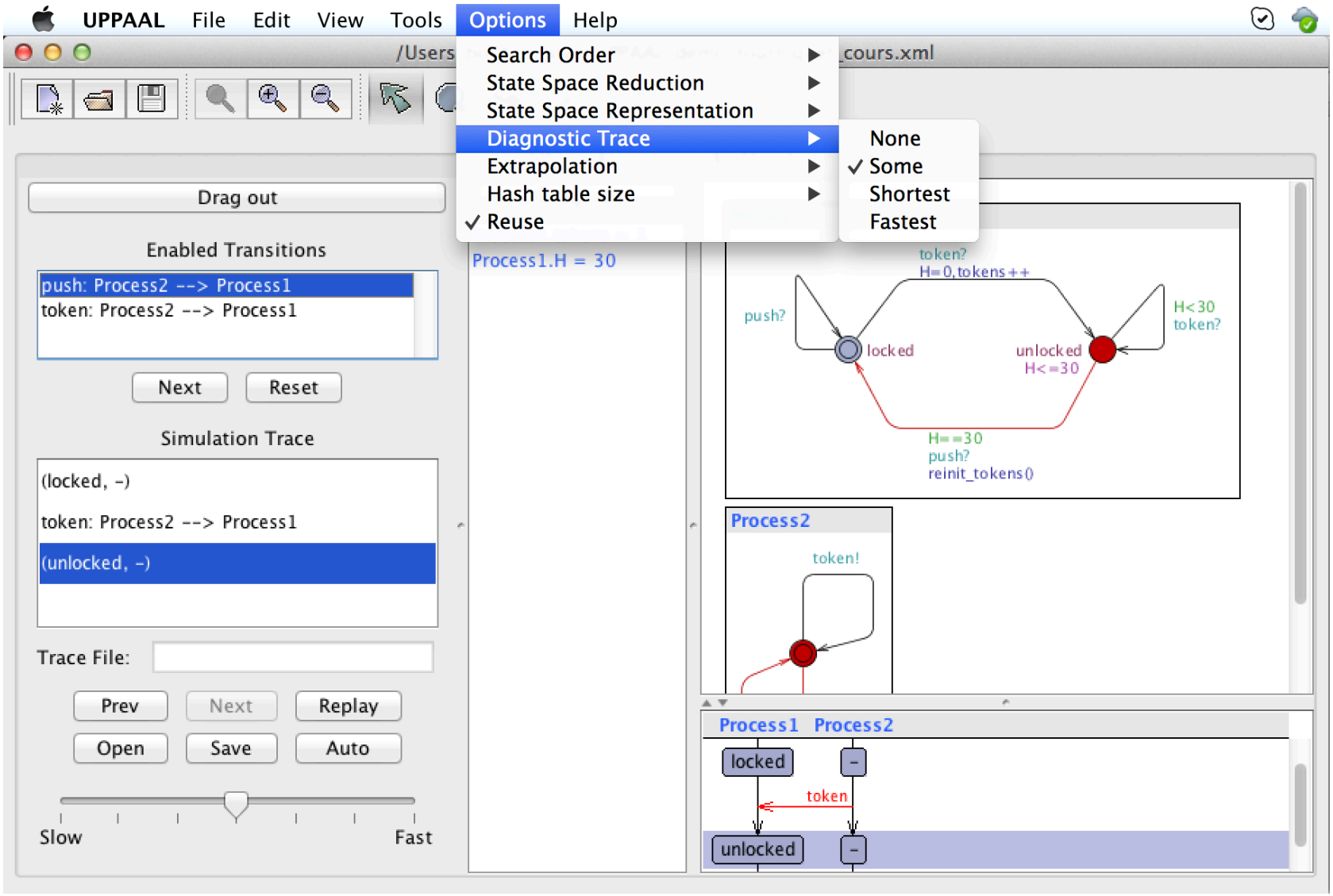
Preuve?

- Cela dépend de la bonne réinitialisation de H…
- Le MC sert à éliminer des erreurs de spécification/conception. A trouver des bugs
  - ⌘ Pb: si le modèle est faux, ce ne sera pas toujours détecté à la vérification; exemple: trouvez l'erreur!

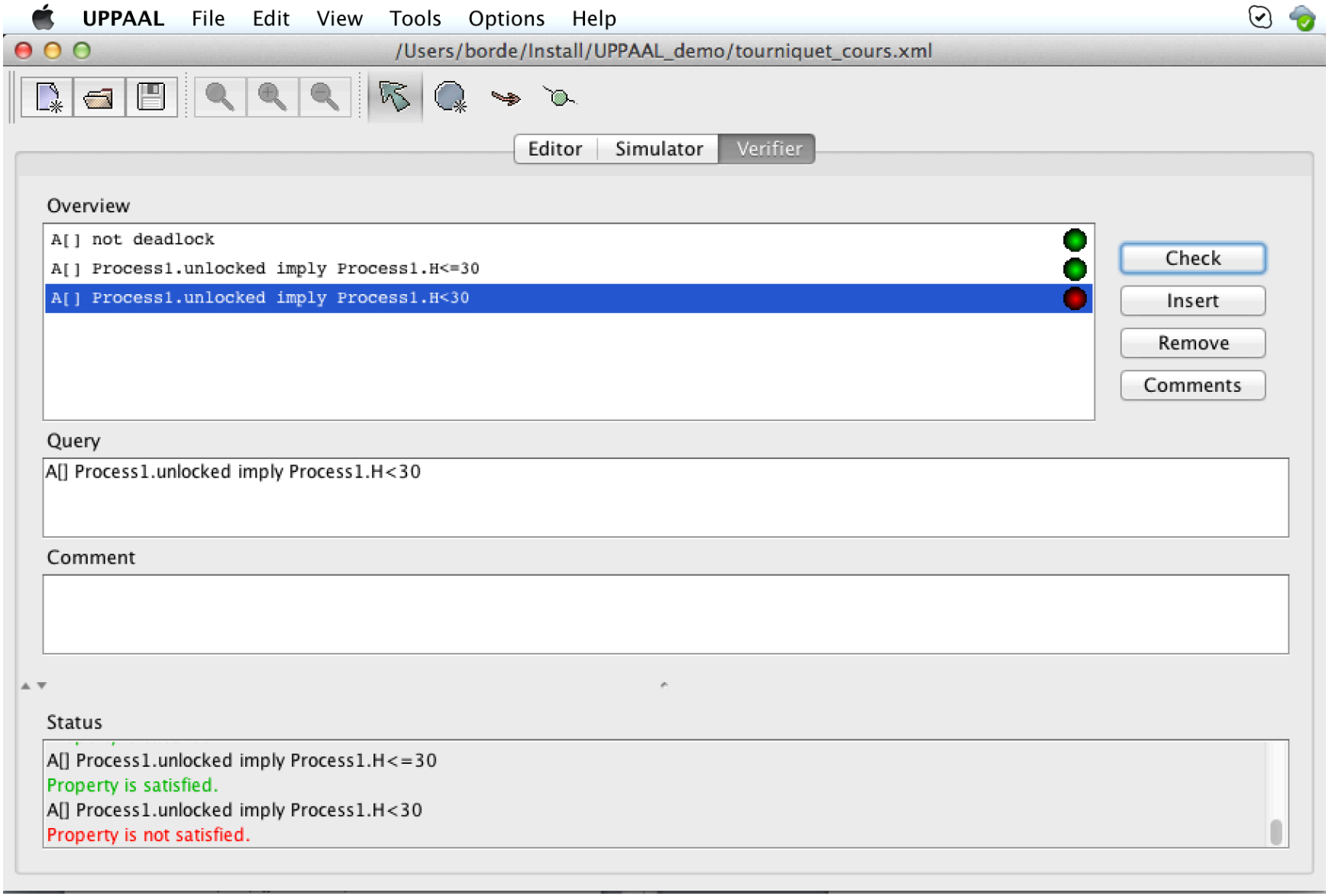


- Donc, le résultat de la vérification « Propriété vérifiée » n'est pas une preuve… Pour la preuve, il faudra en plus formuler des hypothèses à valider manuellement:
  - ⌘ Exemple: il n'y a pas de transition sortante de l'état Dévérouillé qui réinitialise l'horloge H
- En revanche, si la propriété est non satisfaite, on obtient un contre-exemple pour le diagnostique

Exemple du tourniquet: on active le diagnostique



Exemple du tourniquet: vérification fausse (illustration diagnostique)



Exemple du tourniquet: diagnostique produit

UPPAAL File Edit View Tools Options Help

/Users/borde/Install/UPPAAL_demo/tourniquet_cours.xml

Editor | Simulator | Verifier

Drag out

Enabled Transitions

push: Process2 --> Process1
token: Process2 --> Process1

Next | Reset

Simulation Trace

(locked, -)
token: Process2 --> Process1
(unlocked, -)

Trace File:

Prev | Next | Replay
Open | Save | Auto

Slow — Fast

Drag out

Process1.tokens = 1
Process1.H = 30

Process1

token?
H=0,tokens++
push?
locked
unlocked
H<=30
H<30
token?
H==30
push?
reinit_tokens()

Process2

token!

Process1 Process2

locked
token
unlocked

Sous ensemble de CTL supporté par UPPAAL

- Les opérateurs de base:
  - E – il existe un chemin ("**E**" in UPPAAL).
  - A – pour tous les chemins ("**A**" in UPPAAL).
  - G – pour tout état sur un chemin ("**[]**" in UPPAAL).
  - F – pour au moins un état sur le chemin ("**<>**" in UPPAAL).

- Syntaxe supportée:
  - A[]p, A<>p, E<>p, E[]p,
    p1 --> p2 (« p1 leads to p2 », ou A[] (p1 imply A<> p2)

Le menu Help est ton ami…

s: état

dc: contrainte logique sur les données

  - p::= a.s | dc | cc | deadlock |
    p and p | p or p | not p |
    p imply p | (p)

cc: contrainte sur les horloges

a: process uppaal

Exercice d'introduction aux réseaux d'automates

- Illustrons l'atomicité des transition et les entrelacements entre automates d'un réseau d'automates:
  - ⌘ How much can we reach? (http://people.cs.aau.dk/~bnielsen/TOV08/ESV04/exercises)

Consider the parallel execution of two, simple (and identical) sequential programs:

```
int n=1;
Process P:
do
    n := n+n
forever
```

Given that n:=n+n was atomic the possible values of n during execution would be (all) powers of 2. However, consider the situation when the update n:=n+n is carried out as the following sequence of three atomic updates: r:=n; r:=r+n; n:=r. Now the question as to which values n may achieve during execution is no longer obvious.

Model the above scenario in UPPAAL and try to figure out whether (and how) n can achieve the values 2, 3, 8, 9, and 23. Which (and why) values can n achieve?

Ce que vous ferez en TP

- Vérification d'un protocole d'exclusion mutuelle sans verrou (Algorithme de Fischer) et avec delais
  - ⌘ Traduction du code objet en automate UPPAAL
  - ⌘ Vérification de l'exclusion mutuelle
  - ⌘ Discussions sur la validité du modèle (hypothèses explicites et implicites).

- Attention, CR à rendre à la fin de la semaine du TP → noté

Conclusion à propos du model-checking

- Activité complémentaire du test, visant plutôt à détecter des erreurs en avance de phase
  - ⌘ Exhaustivité de la vérification, mais sur un modèle

- D'excellents résultats ont déjà été obtenus, mais c'est encore un sujet de recherche:
  - ⌘ Passage à l'échelle,
  - ⌘ Extension à des modèles probabilistes, temporisés et probabilistes,
  - ⌘ Conformité entre modèle et réalisation/implémentation

- Analyse de temps de réponse avec des automates temporisés?