La vérification formelle de programmes temporisés

Gérard Berry

Collège de France
Chaire Algorithmes, machines et langages
gerard.berry@college-de-france.fr

Cours 5, 30/03/2016

Suivi du séminaire de Kim Larsen (CISS Aalborg)

Grand merci à Patricia Bouyer (LSV Cachan)

COLLÈGE DE FRANCE
1530

Spécifier, modéliser ou programmer ?

- Spécifier, modéliser :
  - décrire formellement un ensemble de comportements autorisés et un ensemble de comportements interdits, sans dire comment les construire
  - avantages majeurs : abstractions à niveaux variés, facilitant le design, la compréhension, la vérification formelle et laissant la place aux choix d'implémentations

- Programmer :
  - fabriquer le système réellement utilisable, celui qui aura les vrais comportements
  - avantage majeur : parle du vrai système
  - difficulté : beaucoup plus difficile à vérifier à cause du grand niveau de détails

Du modèle au programme : synthèse automatique, ou nouvelle vérification

Ici, nous parlerons seulement de modélisation

Trois cas où la modélisation formelle est essentielle

- Systèmes temps-réel (ce cours)
  - modélisation d'algorithmes où le temps joue un rôle essentiel et où le respect du timing est une obligation.
- Algorithmes distribués
  - exclusion mutuelle, bases de données réparties, élection de leader, etc.
  - cf. cours du 6 avril 2016 et séminaire de M. Raynal le 6 janvier 2010
- Protocoles de sécurité
  - connexions sécurisées, vote électronique, sécurité des données et de la vie privée, etc.
  - cf. séminaire de Véronique Cortier le 25 mars 2015, séminaire de Stéphanie Delaune le 6 avril 2016 + cours de Martin Abadi en 2010-2011

Dans tous ces domaines, l'intuition et le test sont insuffisants, et la modélisation formelle des specs indispensable : si la spec est fausse, la réalisation le sera aussi !

Objectif, questions centrales et applications

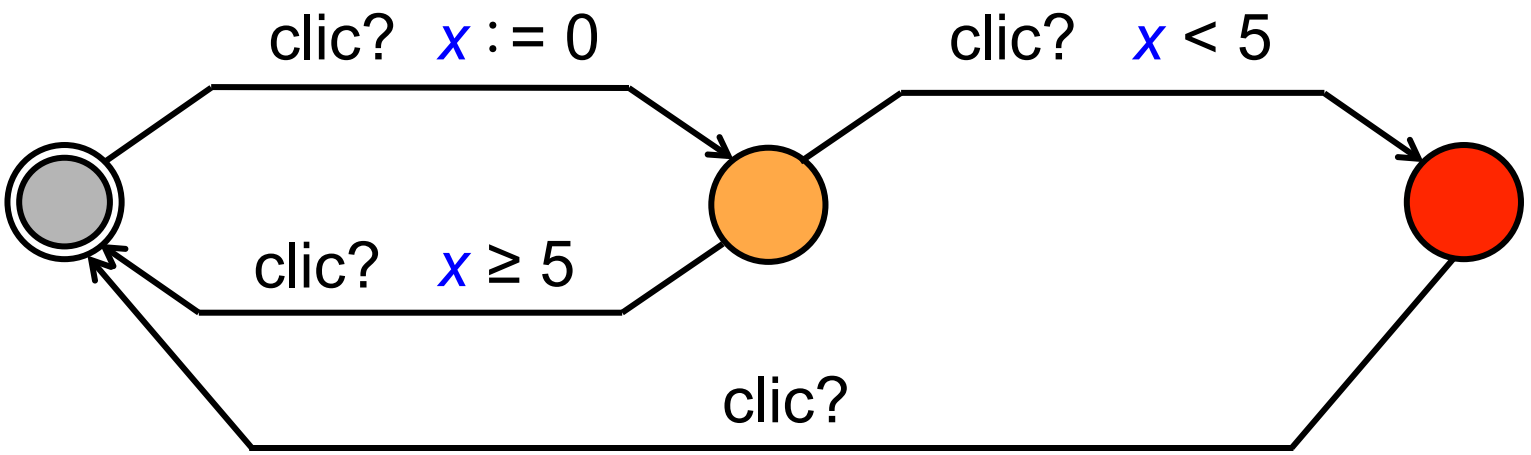
- Modéliser et vérifier des applications où le temps joue un rôle central
  - systèmes cyber-physiques, protocoles de télécommunication, etc.
- Des applications importantes
  - transports, robots, équipements industriels, etc.
  - protocoles réseaux dépendant du temps (ex. Bang & Olufsen, ABB, TDMA, etc), multimedia
  - ordonnancement de tâches (scheduling)
- Deux grandes questions
  - comment modéliser? → automates temporisés, automates hybrides
  - comment vérifier ? → abstraction par zones

Un système de référence : UPPAAL (Kim Larsen et. al.)
UPPAAL = UPPsala + AALborg

Agenda

1. Les automates temporisés
2. Horloges, trajectoires et régions
3. Logiques temporisées
4. Exemples UPPAAL
5. Vérification par zones et DBMs
6. Les automates hybrides

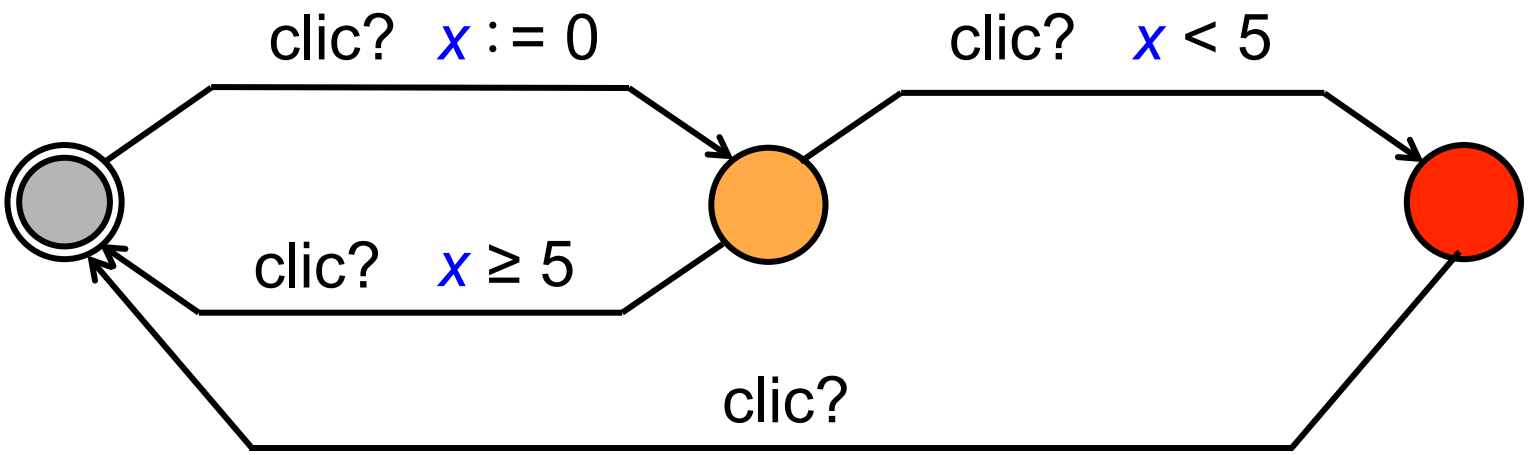
Automates temporisés : lampe à double-clic



[AD] Rajeev Alur et Davis Dill, *A Theory of Timed Automata*

Ici x est une variable réelle positive appelée horloge,
Mais je trouve ce nom bien trop surchargé, cf. cours précédents,
je dirai chronomètre ou chrono à la place si confusion.

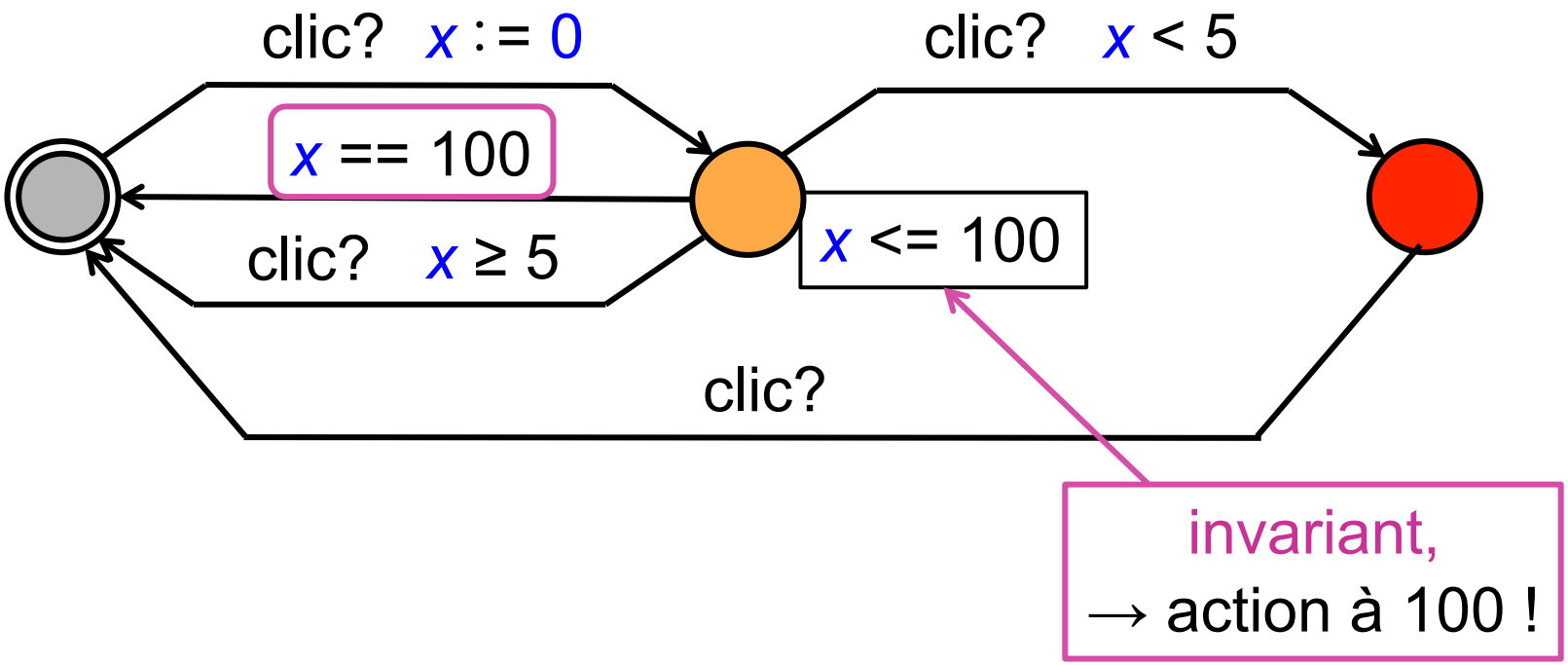
Automates temporisés : lampe à double-click



0 :
0.4 : clic? → x := 0
7.3 : clic? x == 6.9 →

0 :
0.4 : clic? → x := 0
4.3 : clic? x == 3.9 →
7.6 : clic? x == 7.2 →

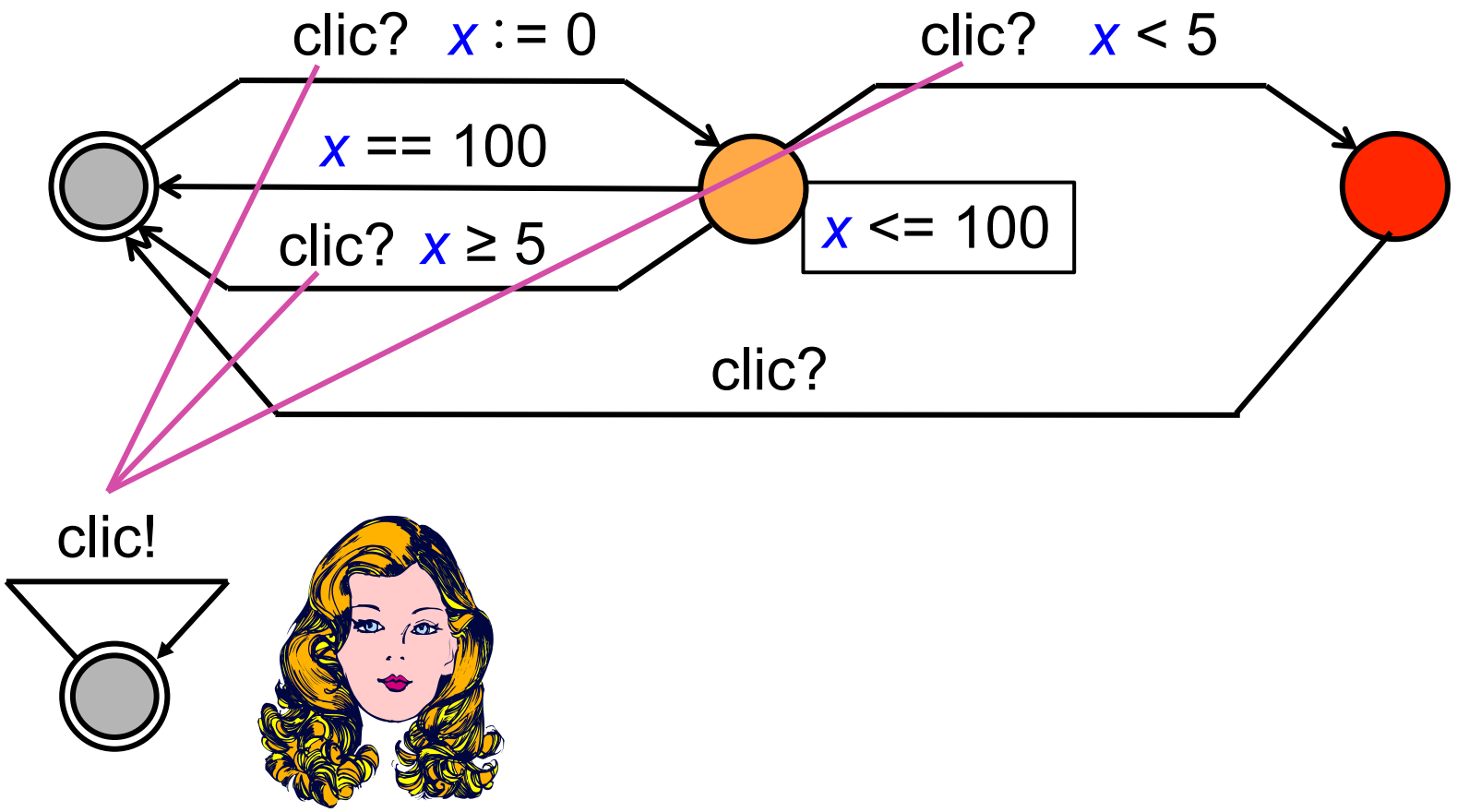
La même lampe avec minuterie partielle



0 :
0.4 : clic? → x := 0
100.4 : x == 100 →

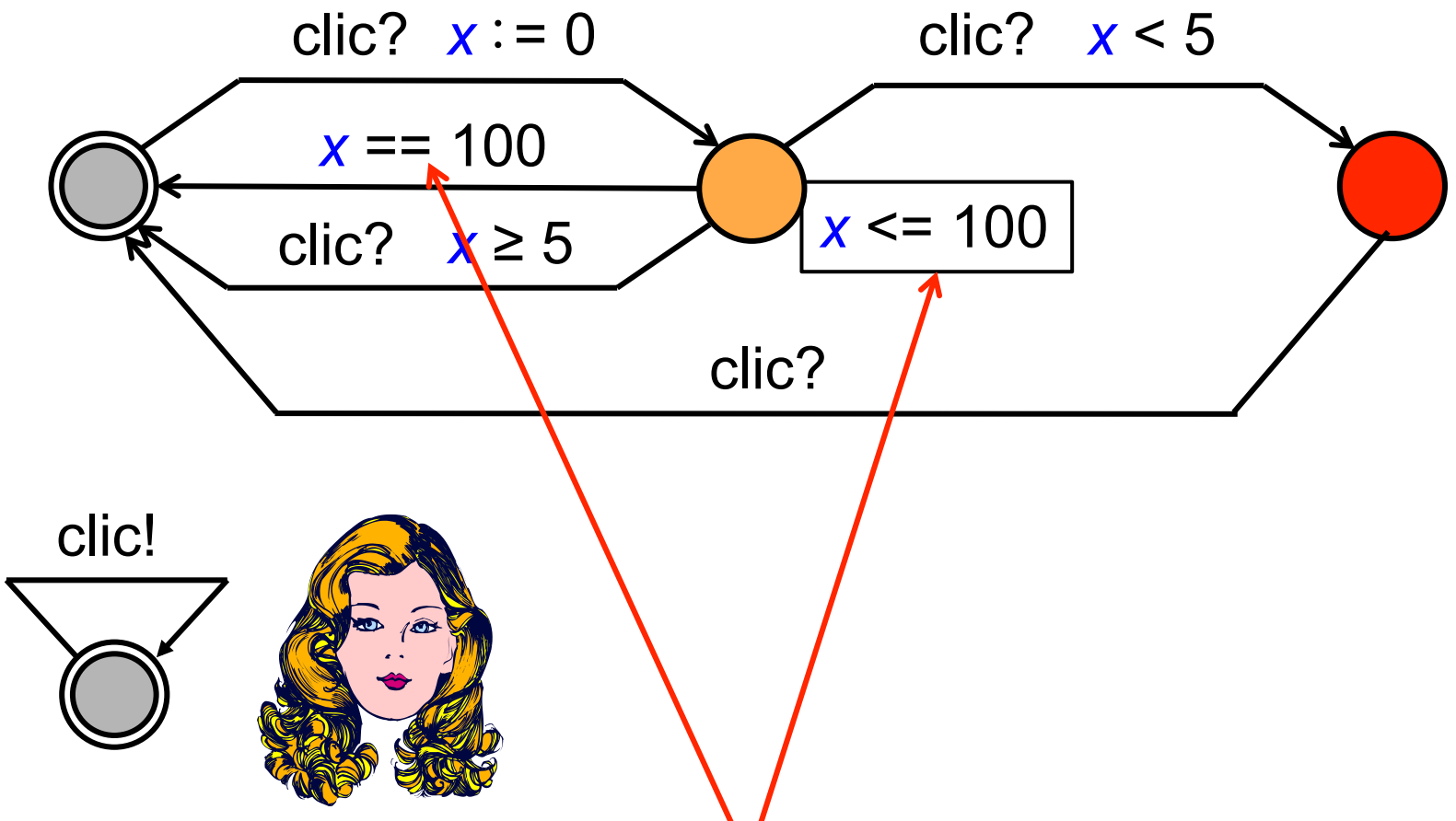
0 :
0.4 : clic? → x := 0
4.3 : clic? → x == 3.9
142.7 : clic? → x == 142.3

Attention, modèle, pas programme !



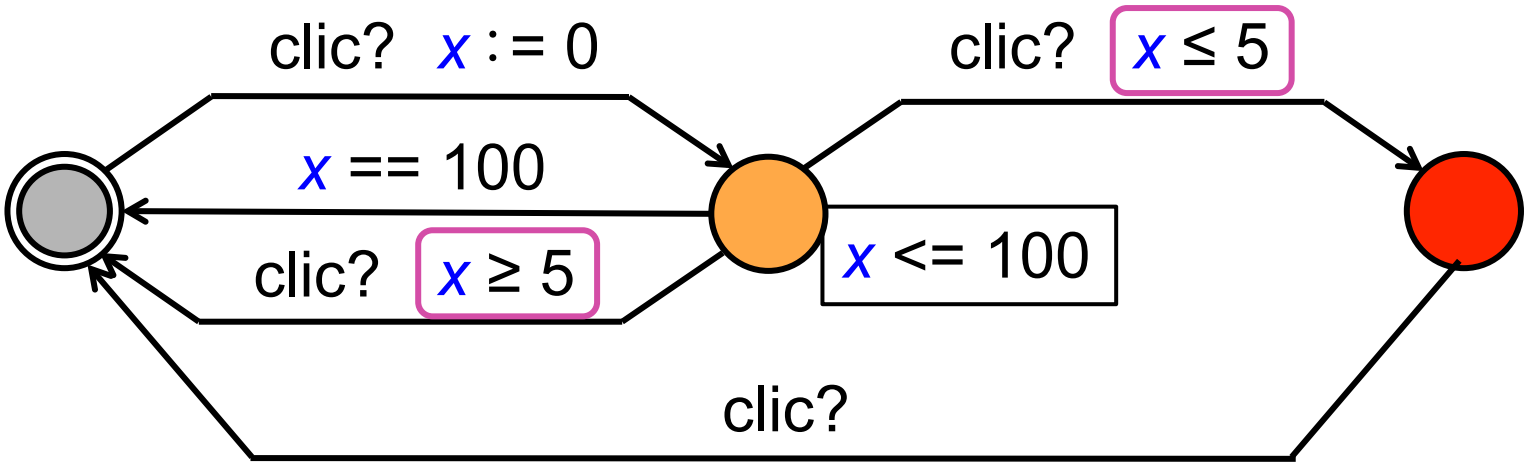
Réseau d'automates à la CCS, descriptif et pas prescriptif :
action = rendez-vous clic!-clic? instantané mais asynchrone

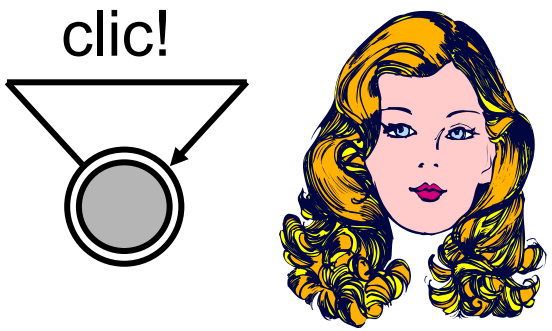
Attention, spécification, pas programme !



Seuls les invariants peuvent prescrire des comportements

Non-déterminisme autorisé partout





Quand plusieurs transitions sont possibles,
on peut prendre n'importe laquelle

Un réseau d'automates temporisés

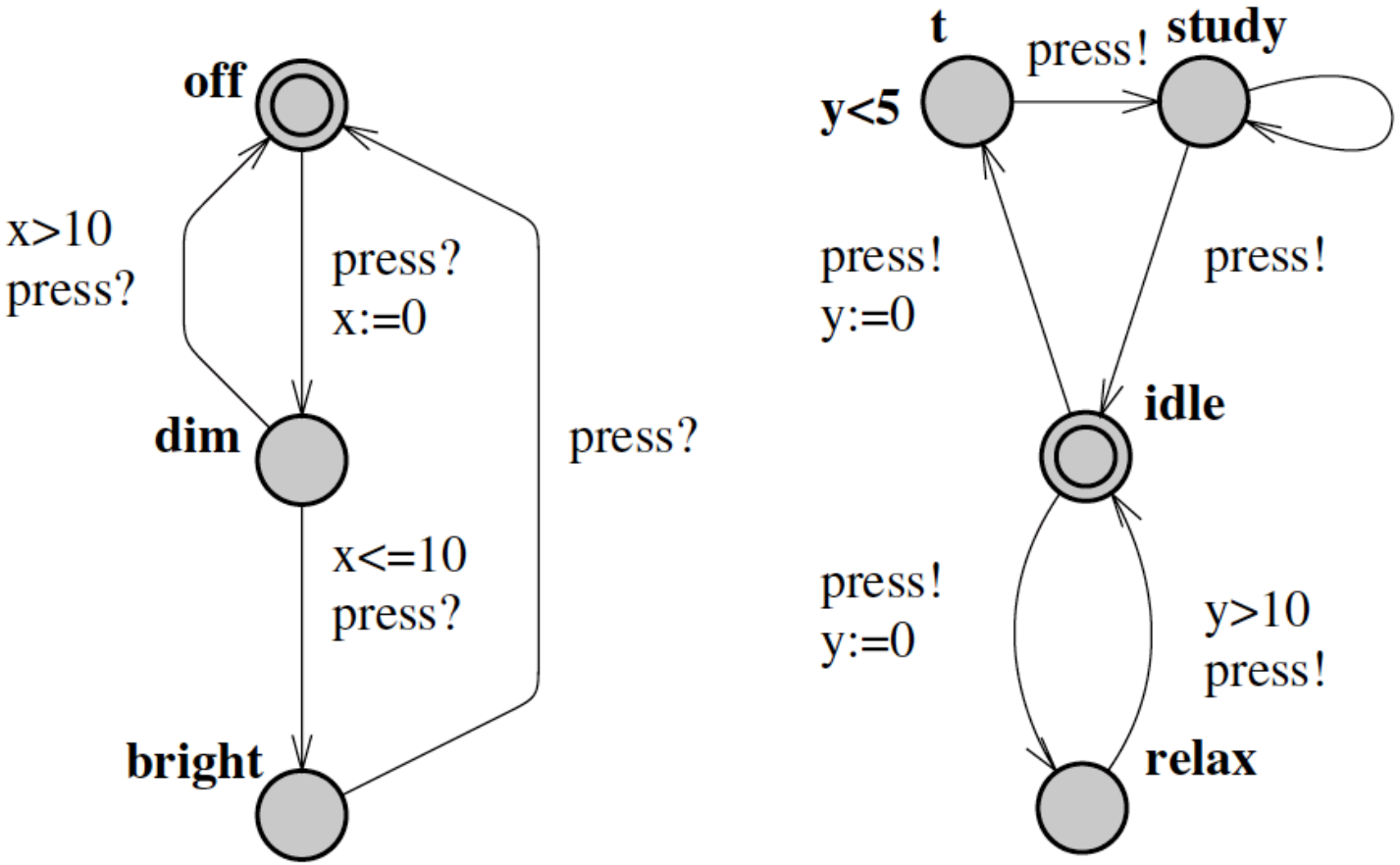
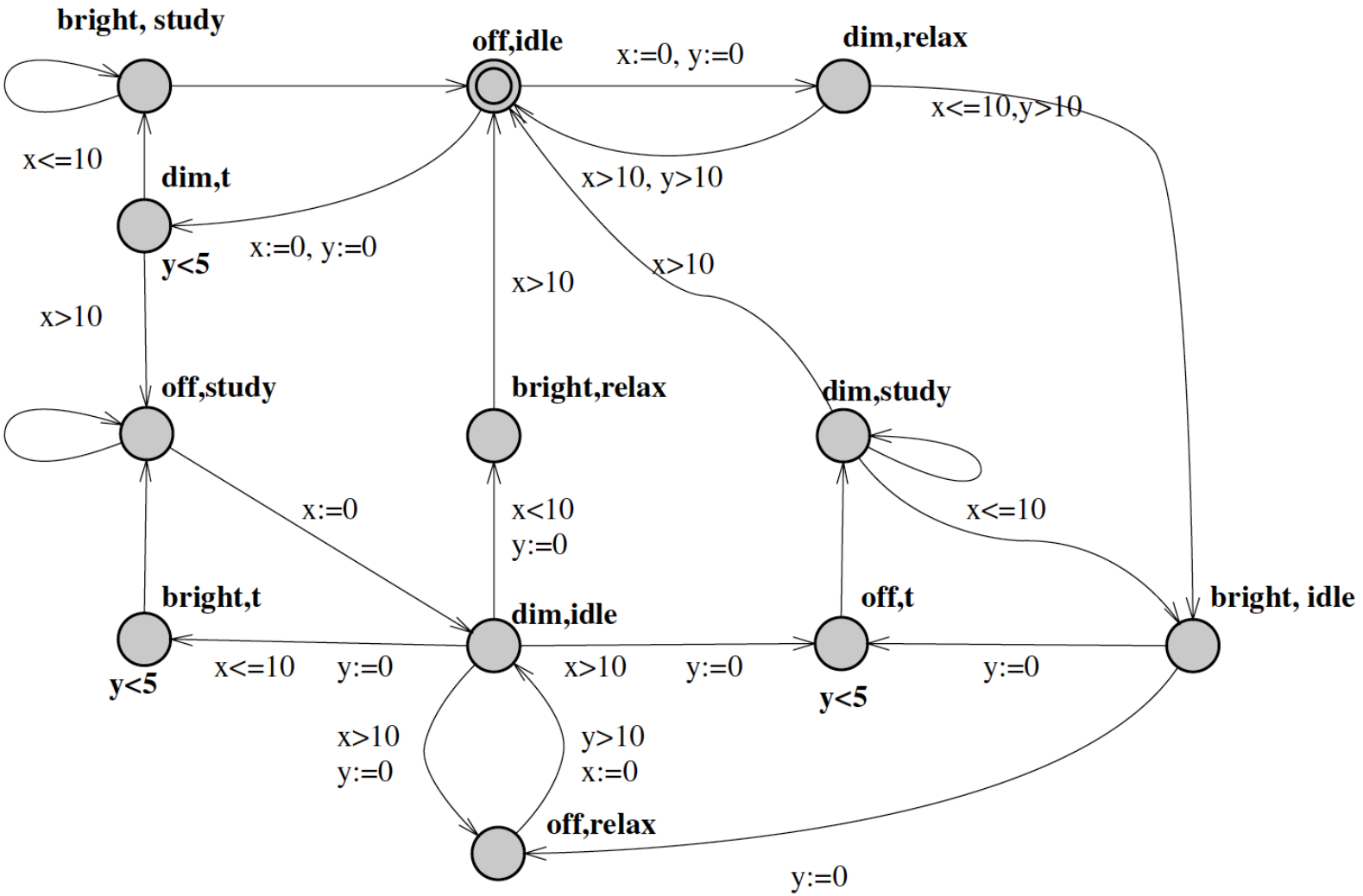


Fig. 14. Network of Timed Automata.

Pas essentiel en théorie, mais essentiel pour l'utilisation pratique !

source Bengtsson et Wang Yi [BWY]

Expansion du réseau en un seul automate



source Bengtsson et Wang Yi [BWY]

Agenda

1. Les automates temporisés
2. Horloges, trajectoires et régions
3. Logiques temporisées
4. Exemples UPPAAL
5. Vérification par zones et DBMs
6. Les automates hybrides

Horloges

1. Une horloge est une variable réelle positive ou nulle.
   On se donne un ensemble fini $C = \{0, x, y, z, \ldots\}$ d'horloges.
   L'horloge spéciale 0 est toujours nulle
2. Une valuation de C est une application $v : C \rightarrow R^+$
3. Les horloges (sauf 0) avancent toutes à la même vitesse.
   On pose $(v + d)(x) = v(x) + d$ pour tout délai $d \neq 0$
4. Une contrainte d'horloges est une conjonction d'inéquations de la forme
   $x - y \lhd k$
   ou k est un entier relatif et $\lhd$ est $\leq, <, =, \geq$ ou $>$
   L'ensemble des contraintes d'horloge est noté $\Phi(C)$

Note : on peut réduire les contraintes à $\leq$ et $<$ car

$$x - y = k \Leftrightarrow x - y \leq k \wedge y - x \leq -k$$
$$x - y \geq k \Leftrightarrow y - x \leq -k$$
$$x \leq k \Leftrightarrow x - 0 \leq k$$

Actions et traces

On se donne un ensemble fini Σ d'actions

Une trace temporelle est une séquence finie ou infinie

$$(a_0,t_0),(a_1,t_1),\ldots,(a_p,t_p),\ldots$$

où $a_i \in \Sigma$, $t_i \in R^+$ et $i < j \Rightarrow t_i \leq t_j$ (temps croissant)

Dans un réseau, les actions sont composées :

- rendez-vous asynchrones à la CCS (Milner)
- synchronisation et diffusion en temps nul à la Esterel, en utilisant les invariants de places pour ajouter du prescriptif
- ou encore mélange des deux, mêlant asynchronisme et synchronisme dans un seul formalisme

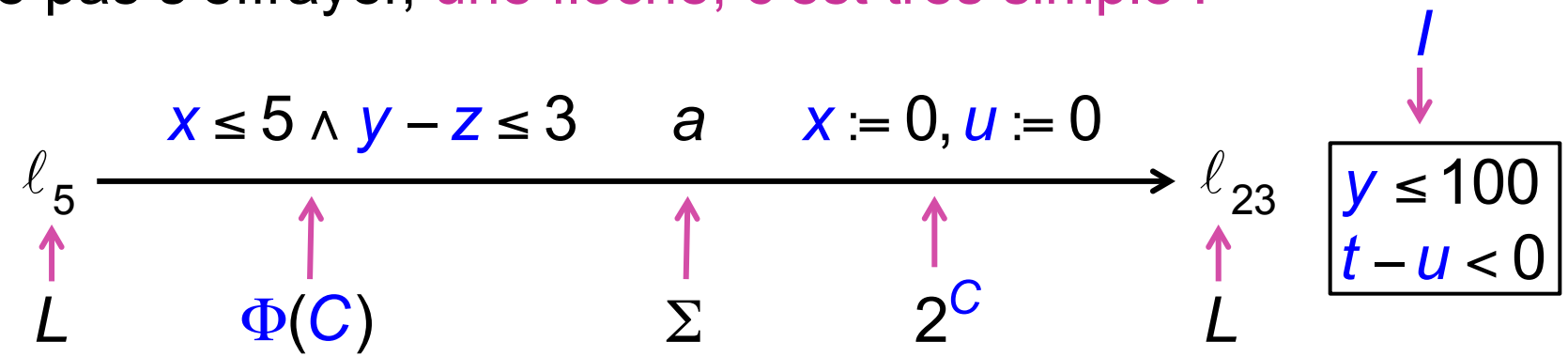
Cf. exemples à venir des trains et du protocole de Fischer

Automate temporisé

- Un automate temporisé A est défini par $A = (L, \ell_0, C, \Sigma, I, E)$ où :

  1. $L = \{\ell_i\}$ est un ensemble fini de *places*
  2. ℓ_0 est la *place initiale*
  3. C est l'ensemble des *horloges*
  4. Σ est l'ensemble des *actions*
  4. $I : L \rightarrow \Phi(C)$ est l'ensemble des *invariants de places*
  5. $E \subseteq L \times \Phi(C) \times \Sigma \times 2^C \times L$ est l'ensemble des *flèches*

- Ne pas s'effrayer, une flèche, c'est très simple !



Transitions et comportements temporels

- Il y a deux sortes de transitions :

  1. Passage du temps sans bouger, en préservant les invariants :

     $(\ell, v) \rightarrow (\ell, v + d)$

     pour tout v et d tel que v et $v + d$ satisfont $I(\ell)$

  2. Prise d'une flèche autorisée

     $(\ell, v) \rightarrow (\ell', v')$

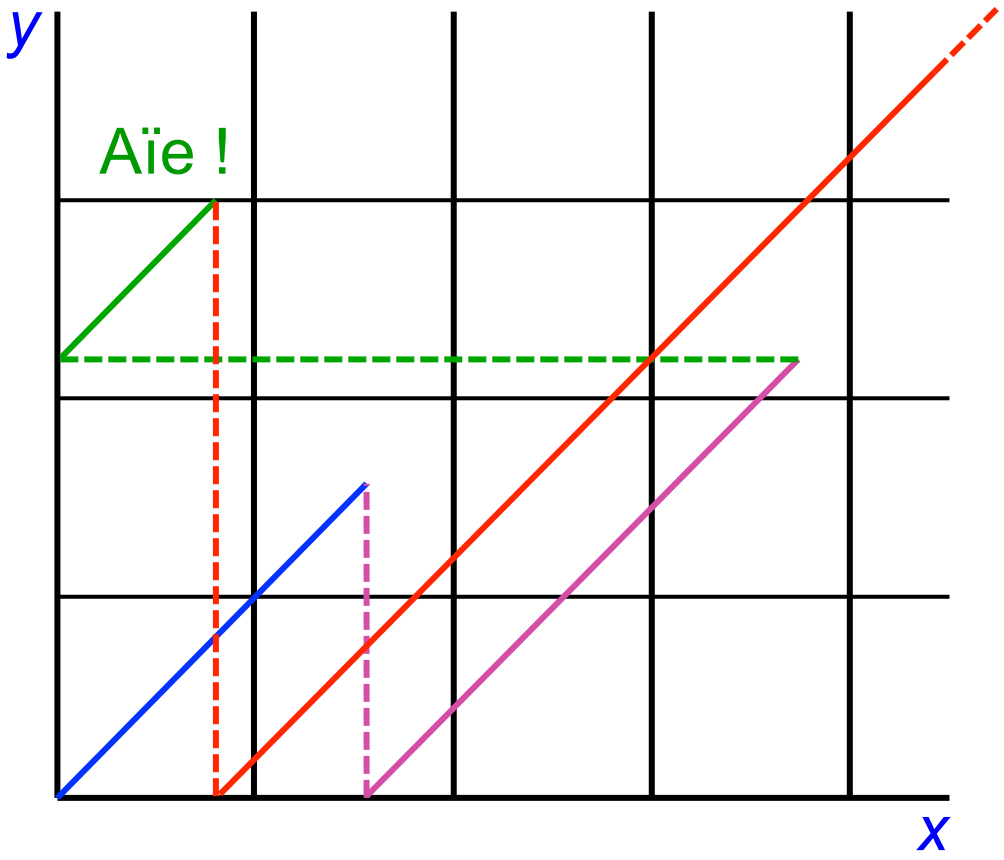
     s'il y a une transition de la forme $\ell, I \xrightarrow{\phi, a, r} \ell', I'$

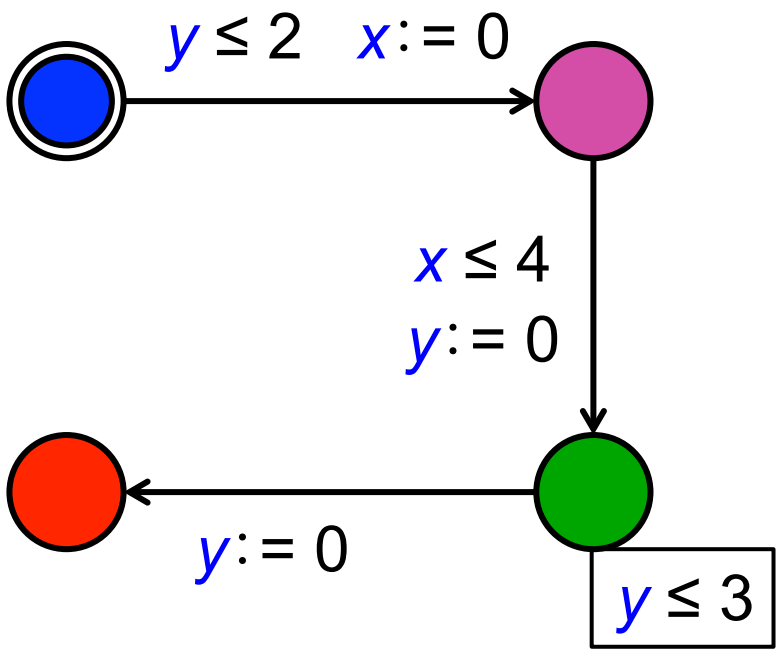
     telle que v satisfait I et ϕ et $v' = v[r := 0]$ satisfait I'

- Un comportement temporel est simplement une suite de transitions légales

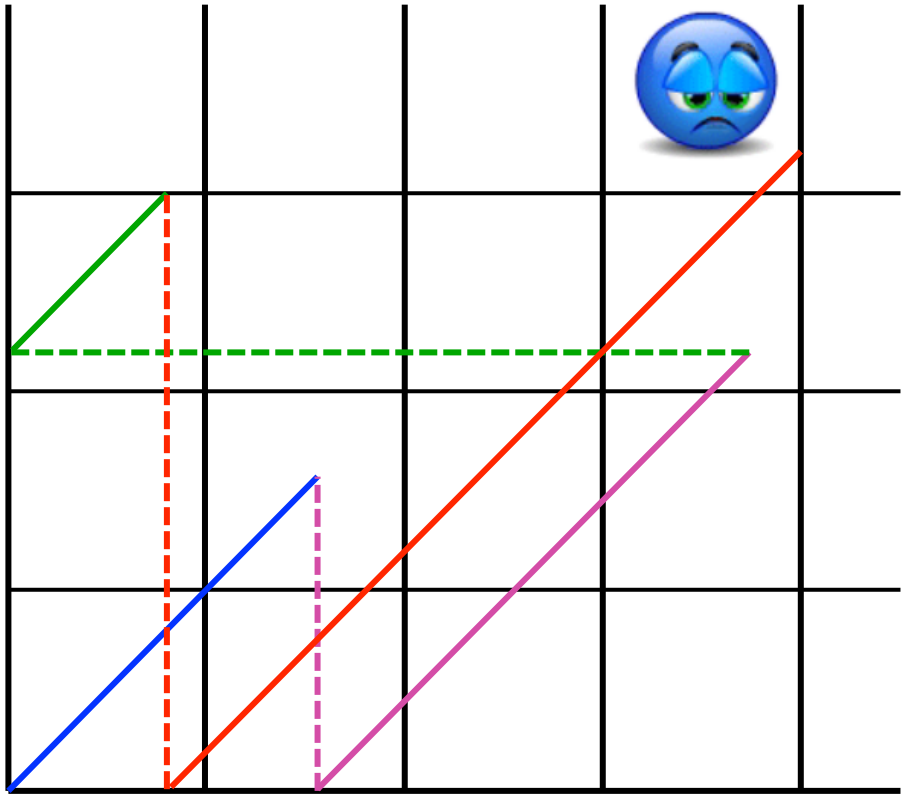
Dans un réseau d'automates, les places sont composites, les invariants sont conjoints et les conditions des flèches aussi

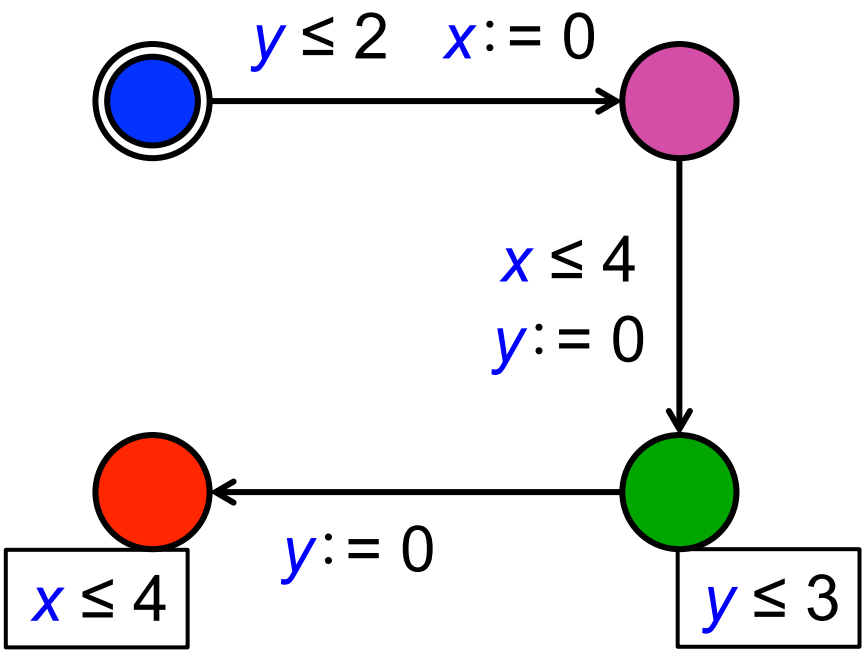
Comportements bi-horloges



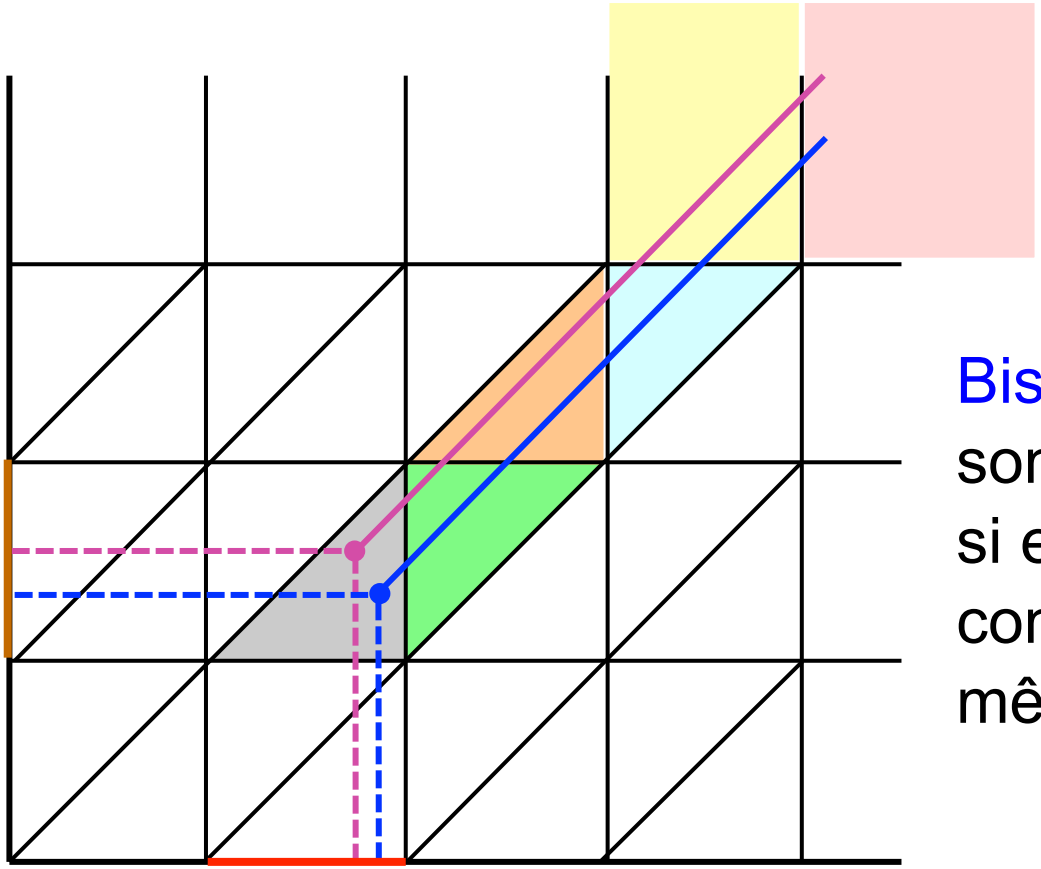


Comportements bi-horloges





Bisimulation et régions



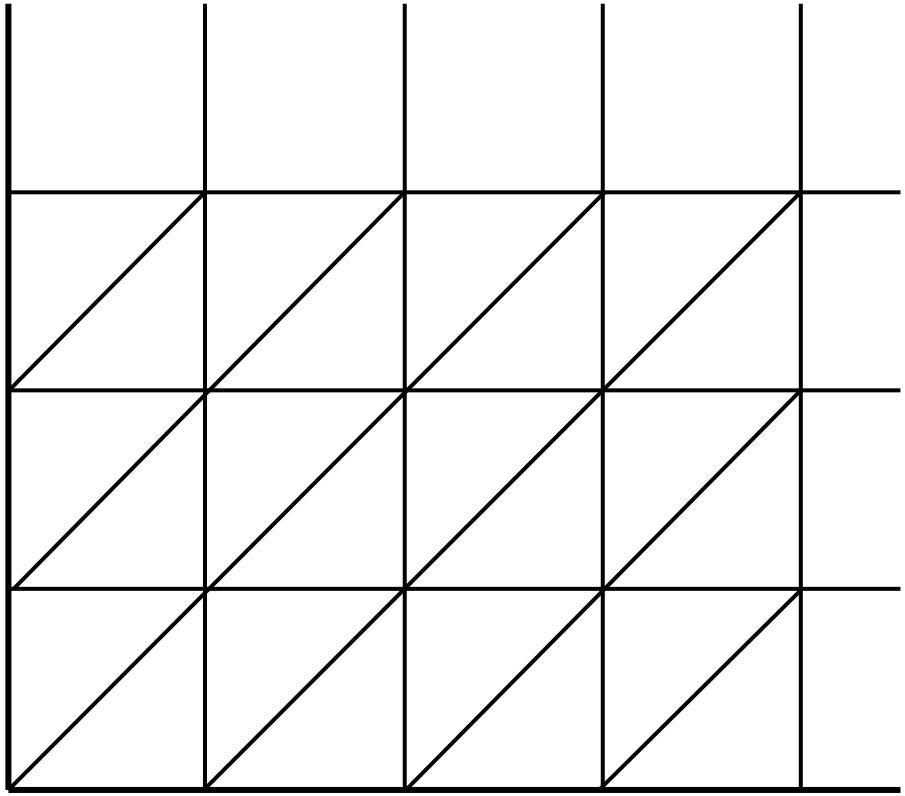
Régions :
point, bord de triangle
triangle, colonnes, etc.

Bisimilarité : deux points sont dans la même région si et seulement si ils conduisent aux mêmes régions

Pour un automate donné, on n'a besoin que d'un nombre fini de régions : inutile d'aller plus loin que la plus grande constante du réseau d'automates !

Décidabilité de l'atteignabilité

Théorème (Alur & Dill) : Pour tout automate temporisé, l'atteignabilité d'une région donnée par un automate est décidable



Mais beaucoup de régions !

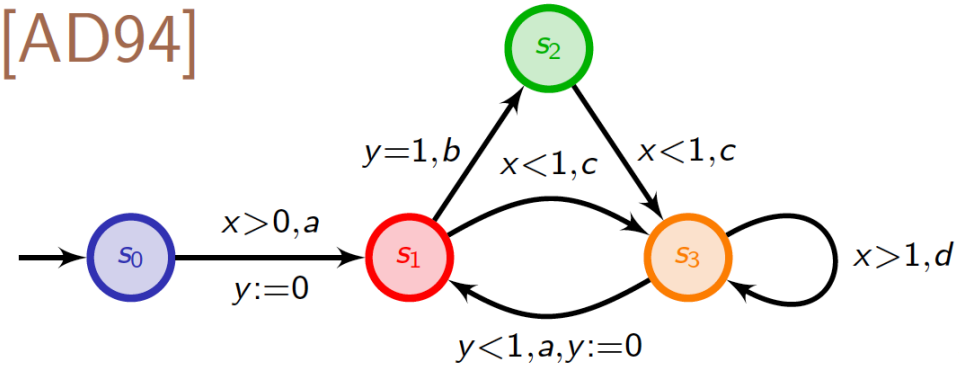
- 20 points
- 7 lignes finies
- 24 triangles
- 7 colonnes infinies
- 3 lignes infinies
- 1 coin infini

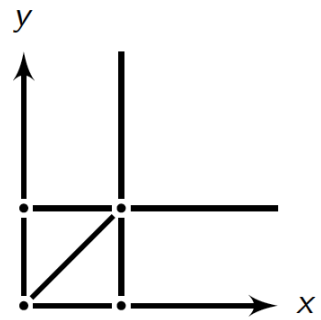
nb de régions exponentiel dans le nb d'horloges

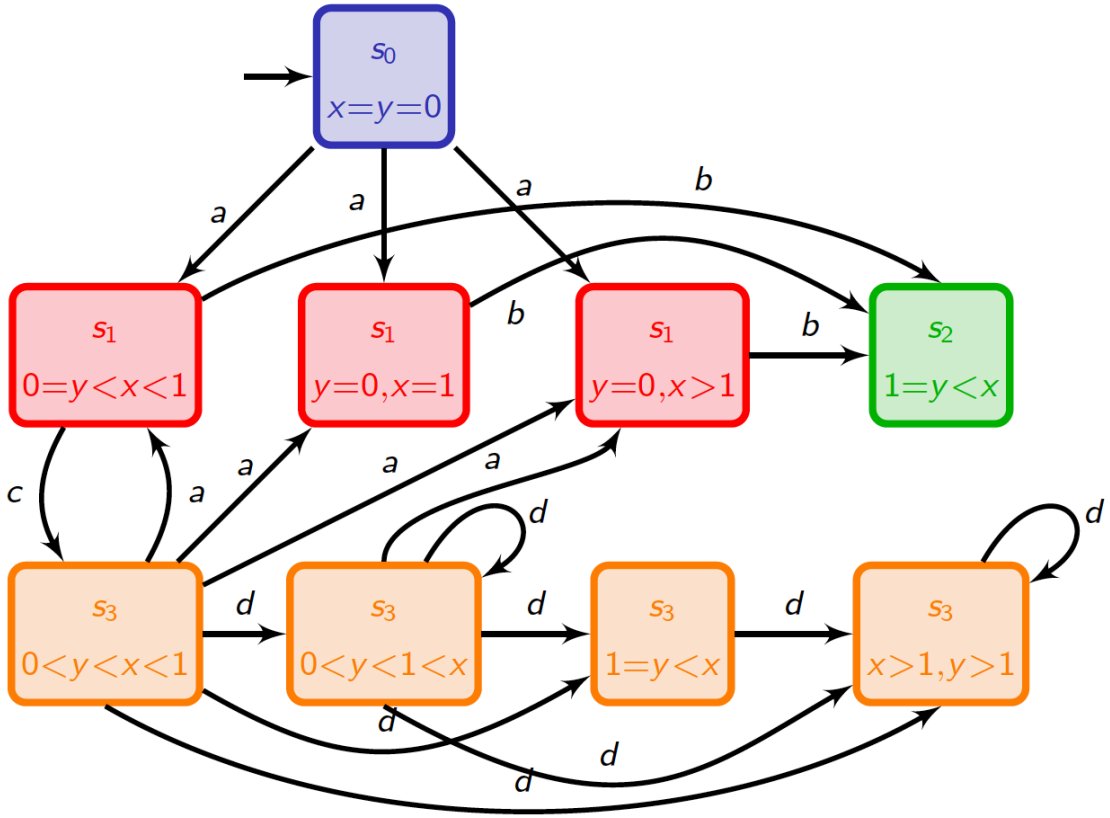
Alur&Dill : pour n horloges, borne sup : $n! \times 2^n \times \Pi_x (2c_x + 2)$

Automate symbolique des régions

An example [AD94]







Agenda

1. Les automates temporisés
2. Horloges, trajectoires et régions
3. Logiques temporisées
4. Exemples UPPAAL
5. Vérification par zones et DBMs
6. Les automates hybrides

Rappels de logique temporelle (25/03/2015)

- Quantification d'états : A= tous, E=existe
- Quantification de chemins : G=toujours, F=un jour, U=jusqu'à
- Logique CTL (*Computation Tree Logic*)

  expressions d'états, AG ϕ, AF ϕ, EG ϕ, EF ϕ, E(ϕUϕ')
- Logique d'UPPAAL : CTL restreinte + temps → décidable

  E <> ϕ : EF ϕ, possible = il existe un chemin où ϕ sera vrai

  A[ ] ϕ : AG ϕ, toujours, abréviation de ¬E <> ¬ϕ

  E[ ] ϕ : EG ϕ, possiblement toujours

  A <> ϕ : AF ϕ, toujours un jour, abréviation de ¬E[ ] ¬ϕ

  $\phi \rightsquigarrow \phi'$: AG ($\phi \Longrightarrow$ AF ϕ')

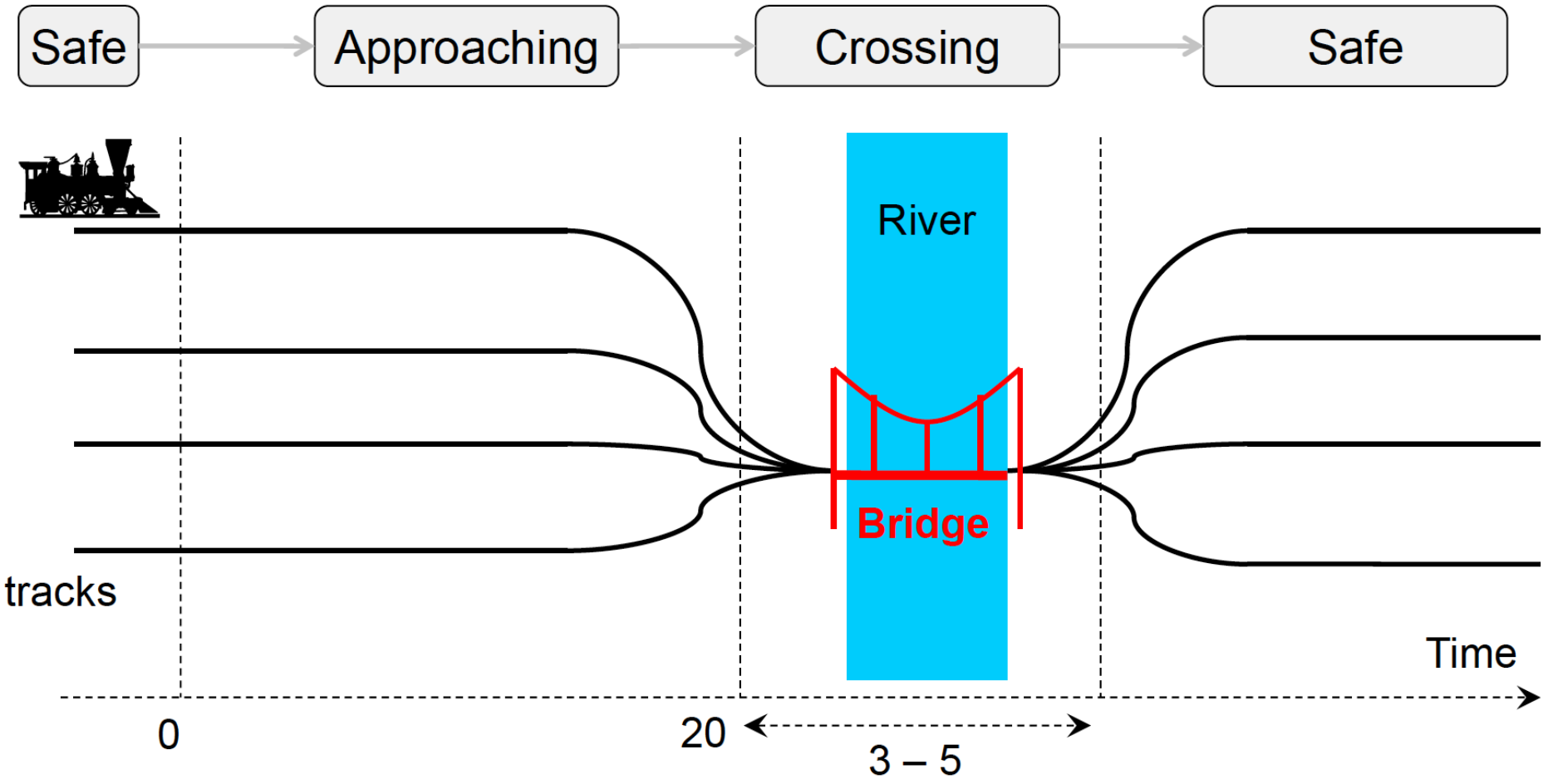
  si ϕ devient vrai, ϕ' deviendra toujours (vrai un jour)

Les horloges peuvent apparaître dans ϕ

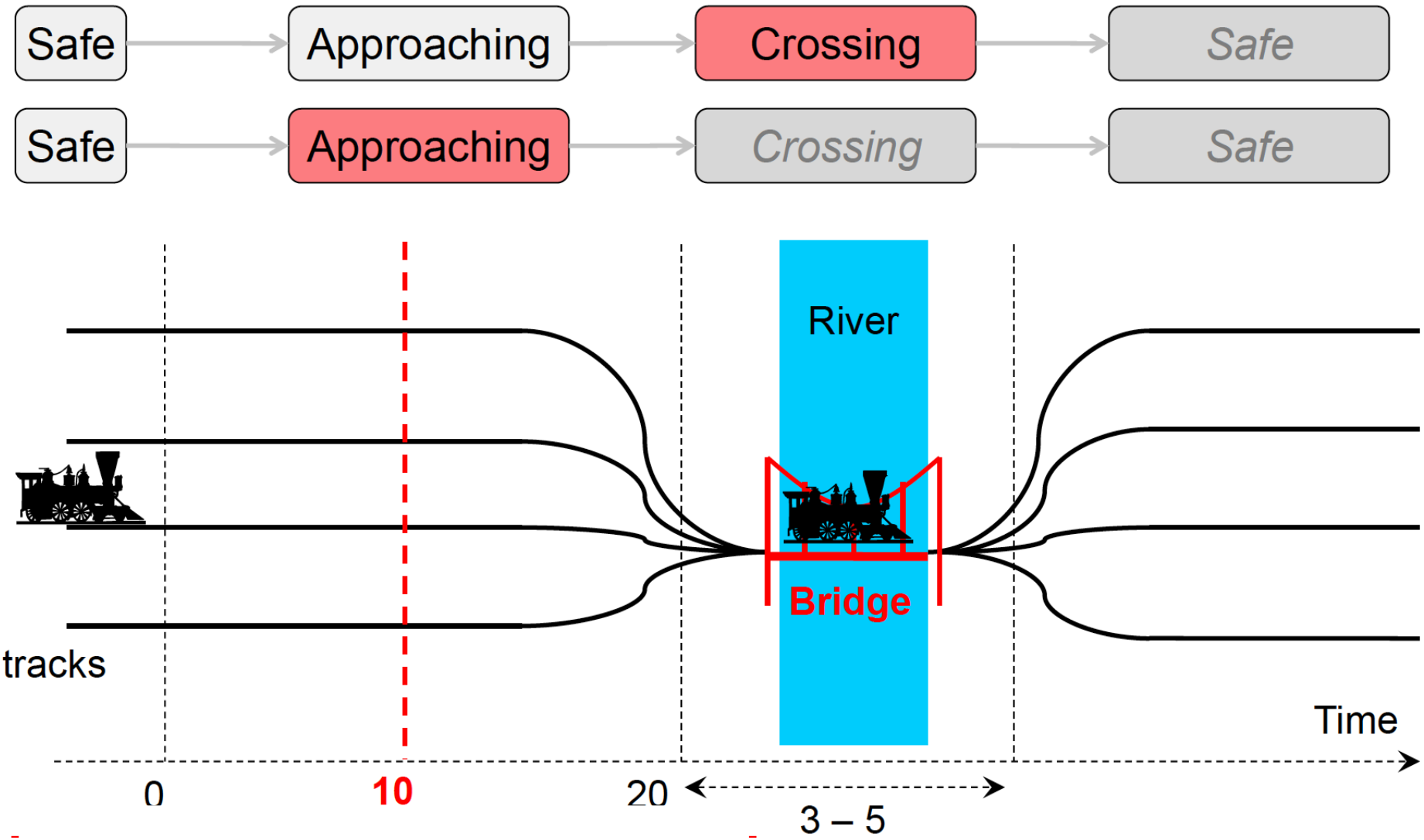
Agenda

1. Les automates temporisés
2. Horloges, trajectoires et régions
3. Logiques temporisées
4. Exemples UPPAAL
5. Vérification par zones et DBMs
6. Les automates hybrides

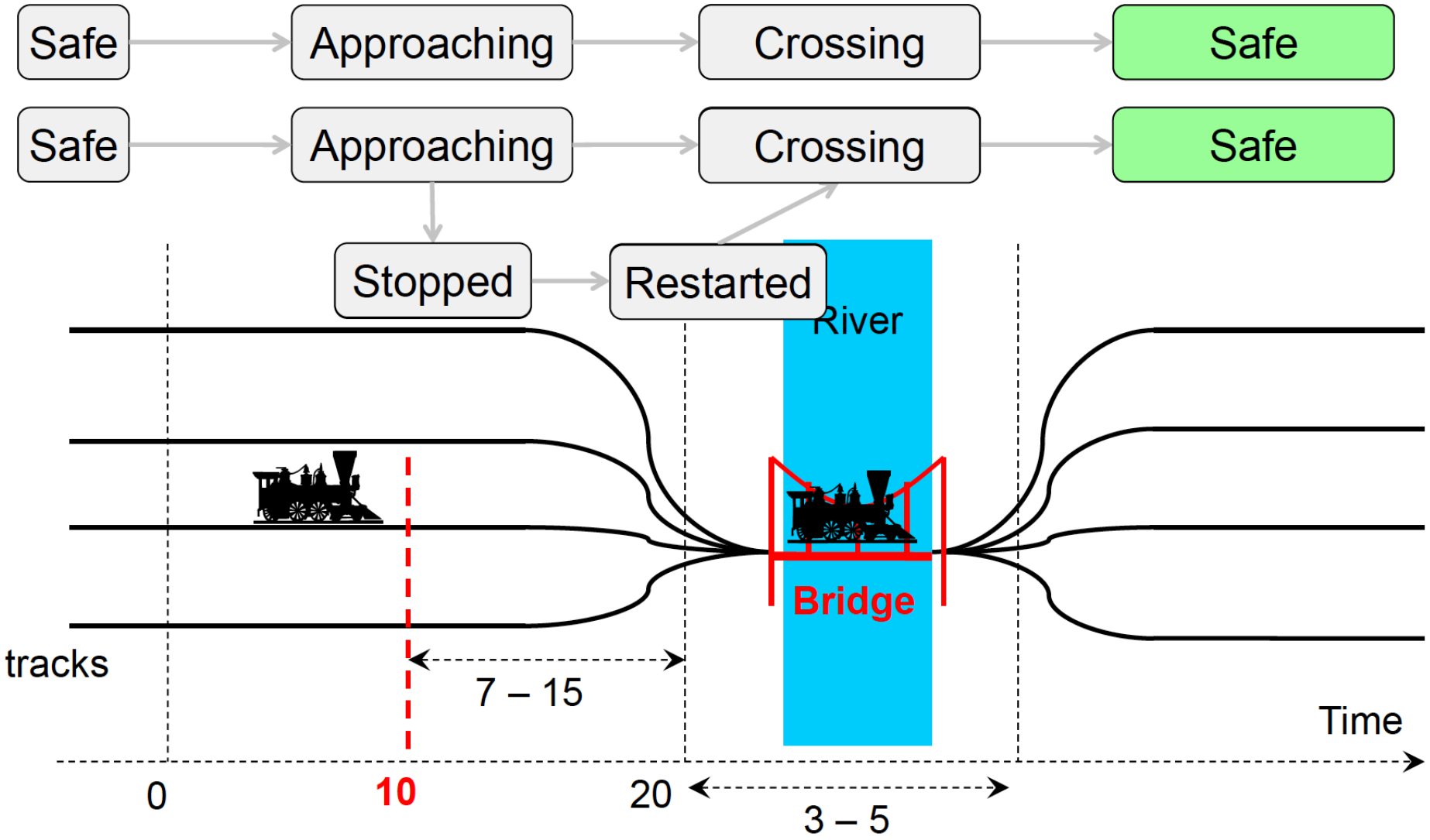
Des trains et un pont



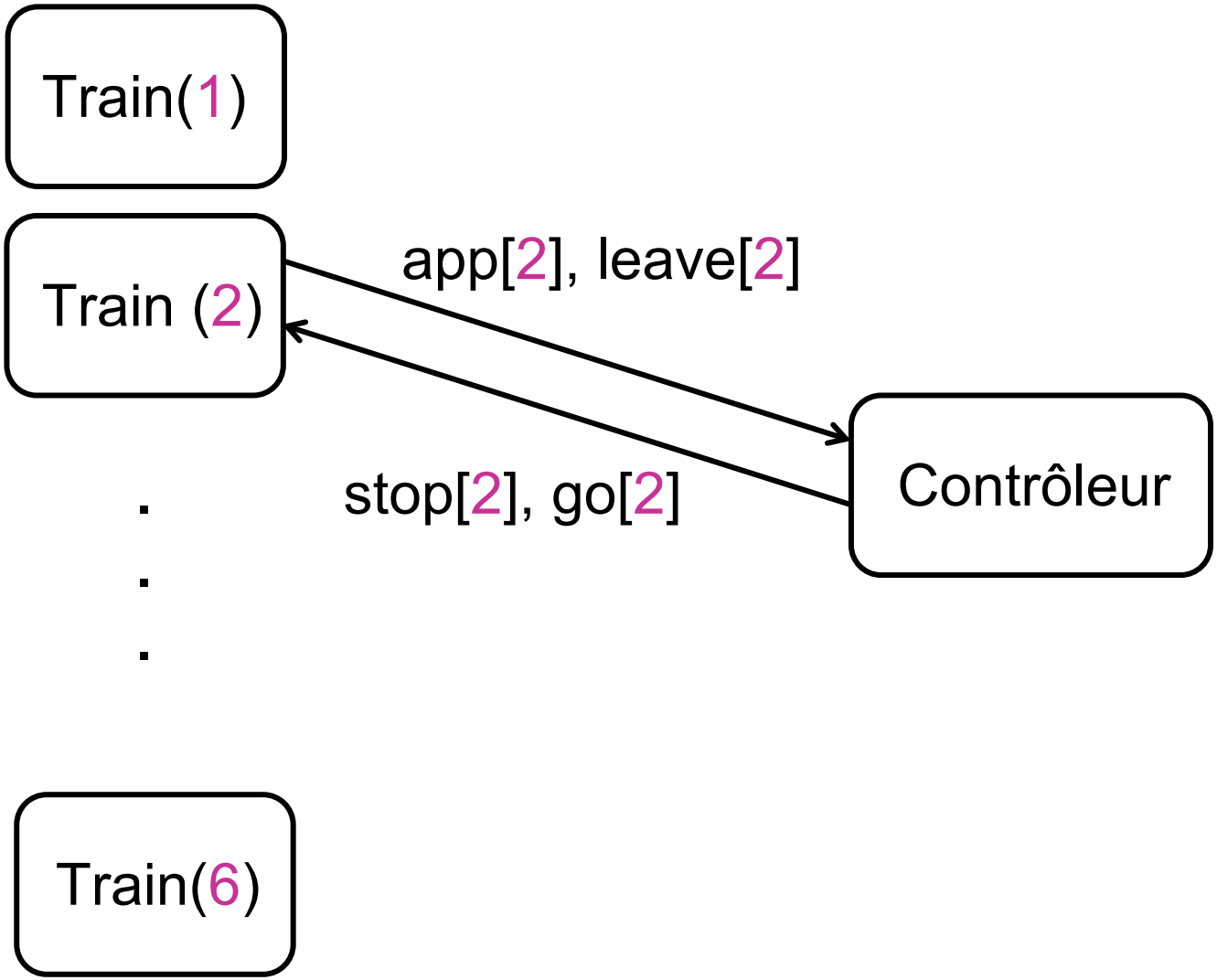
Des trains et un pont



Des trains et un pont



Architecture



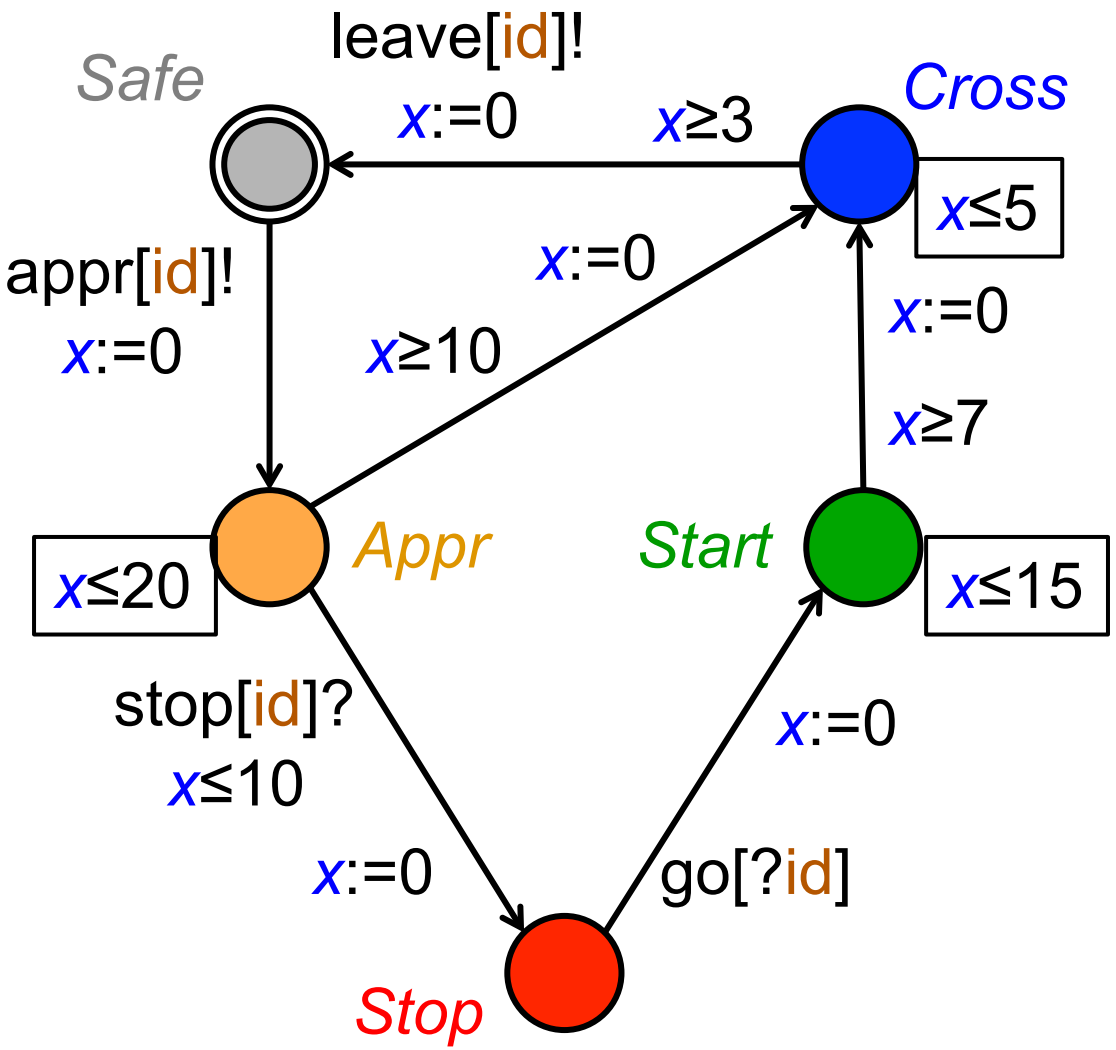
Déclarations de données globales

```
const int N = 6 ; // nombre de trains
typedef int[0..N-1] id_t ; // identificateurs de trains

chan app[N], stop[N], leave[N] ; // canaux de rendez-vous
urgent chan go[N] ;  // canal de rendez-vous urgent
                     // (aucun délai possible)
```

Comportement d'un train d'identificateur id

clock x;



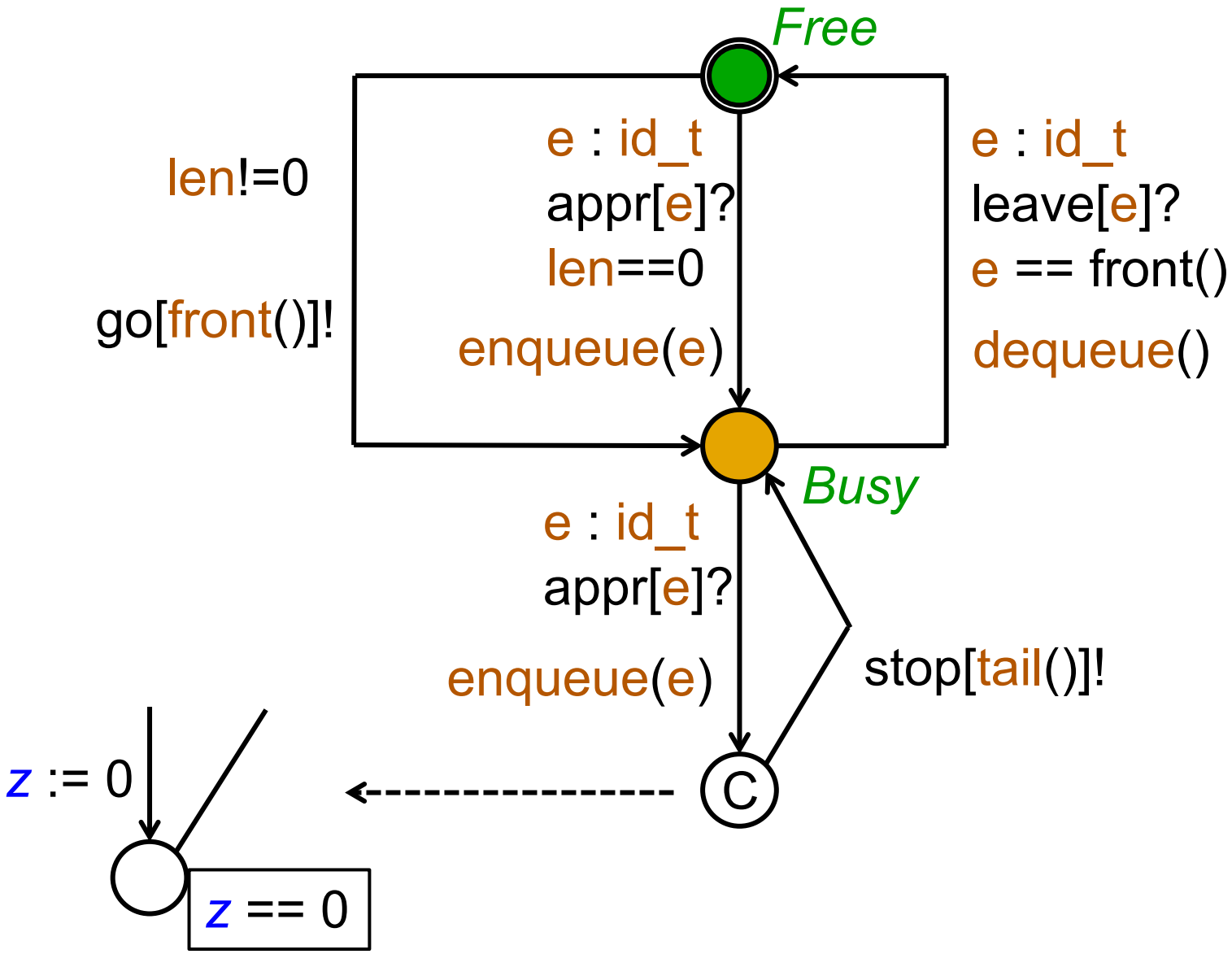
Données du contrôleur

```
id_t list [N+1];
int [0,N] len;
void enqueue (id_t elt) {
   list [len++] = elt;
}
void dequeue () {
   int i = 0;
   len −= 1;
   while (i<len) {
      list[i] = list[i+1];
      i++;
   }
   list[i] = 0;
}
```

```
id_t front () {
   return list [0]
}

id_t tail () {
   return list [len−1];
}
```

Comportement du contrôleur



Propriétés de sûreté prouvées par UPPAAL

- Pas d'accident sur le pont :

  $\forall i, j$ t.q.$1 \le i \le N, 1 \le j \le N$,

  $AG(Train(i).Cross \wedge Train(j).Cross \Longrightarrow i = j)$

- La file d'attente ne peut jamais déborder

  $AG(Gate.list[N] == 0)$;

Mais attention, pour que ces propriétés soient vraies, il suffit que tous les trains soient arrêtés !

Propriétés de vivacité prouvées par UPPAAL

- Tout train pourra passer le pont :

  $\forall i$ t.q.$1 \le i \le N$, EF(Train(i).Cross)

- Tout train qui s'approche finira par passer :

  $\forall i$ t.q.$1 \le i \le N$, AG(Train(i).Appr $\Rightarrow$ AF Train(i).Cross)

- Le système ne peut pas se bloquer :

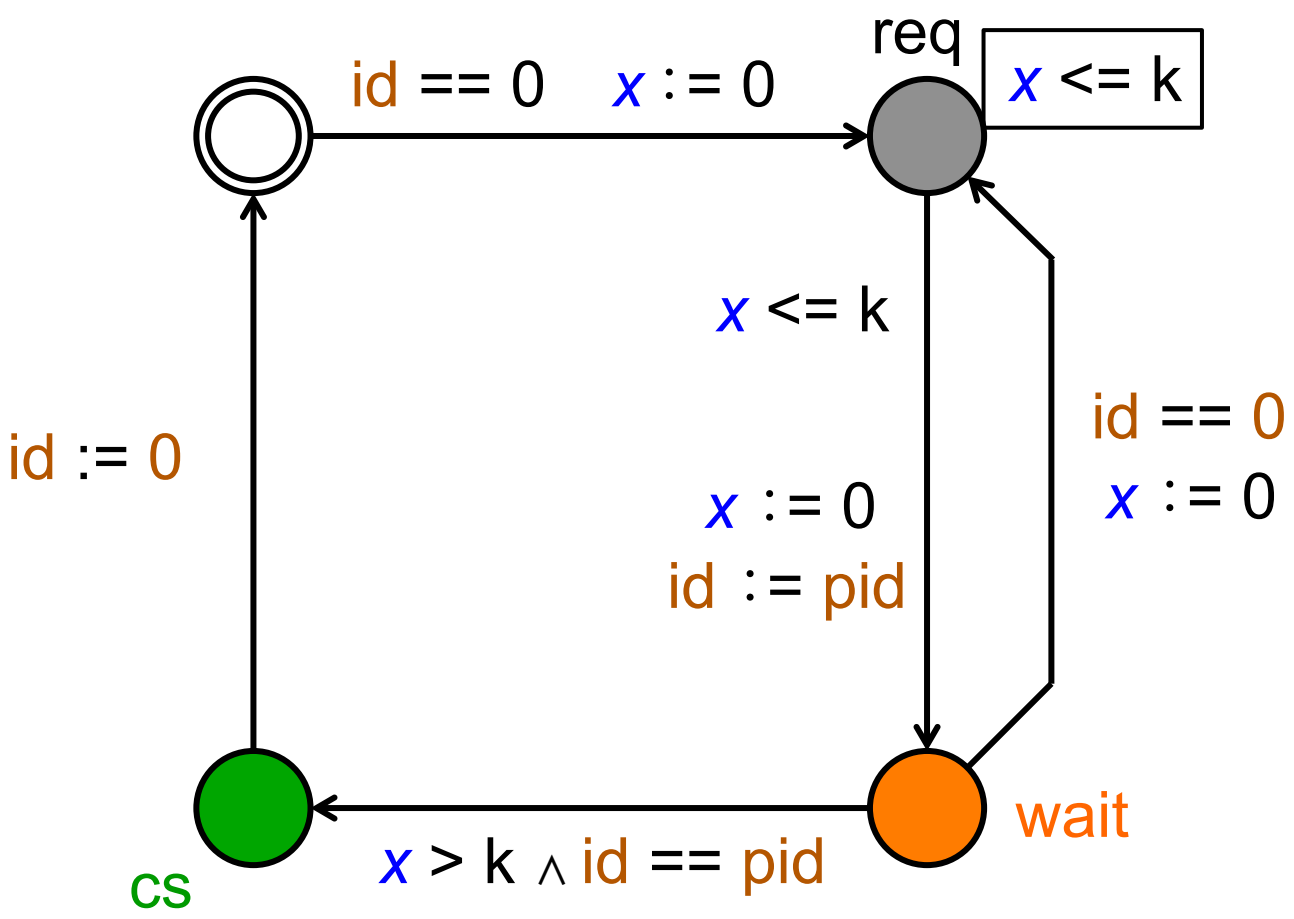
  $\Box$ ~ deadlock

Protocole de Fischer

- N participants veulent accéder à une ressource partagée, qui ne peut être utilisée que par un à la fois
- Compliqué et coûteux avec des passages de messages, mais facile et élégant en utilisant le temps !

- Idée : chaque participant est identifié par un nombre pid > 0. Les participants partagent une variable partagée id, qui vaut 0 si la ressource est libre.
- Quand le processus pid décide de demander la ressource, et à condition que la ressource soit libre, il met son pid dans id et démarre son chrono.
- Quand un temps k>0 est écoulé, il peut prendre la ressource si id vaut toujours id. Sinon, il peut réessayer quand id vaut de nouveau 0

Protocole de Fischer en UPPAAL



Un participant est identifié par son pid (ils sont tous pareils)

La preuve du protocole

1. Exclusion mutuelle

   $AG(\forall i, j.\ P(i).cs \wedge P(j).cs \Rightarrow i = j)$

```
A[] forall (i:id_t) forall (j:id_t)
                P(i).cs && P(j).cs imply i==j
```

2. Vivacité

   $AG(P(1).req \Rightarrow AF(P(1).wait))$

```
P(1).req --> P(1).wait
```

```
A[] not deadlock
```

Mais `P(1).req --> P(1).cs` n'est pas vrai,
car le pauvre P(1) peut se faire toujours doubler !

Agenda

1. Les automates temporisés
2. Horloges, trajectoires et régions
3. Logiques temporisées
4. Exemples UPPAAL
5. Vérification par zones et DBMs
6. Les automates hybrides

Des régions aux zones

- Régions : formalisme trop fin pour nos besoins

→ Zones : traduisent plus efficacement l'évolution d'un automate

Zone = état + contraintes d'horloges

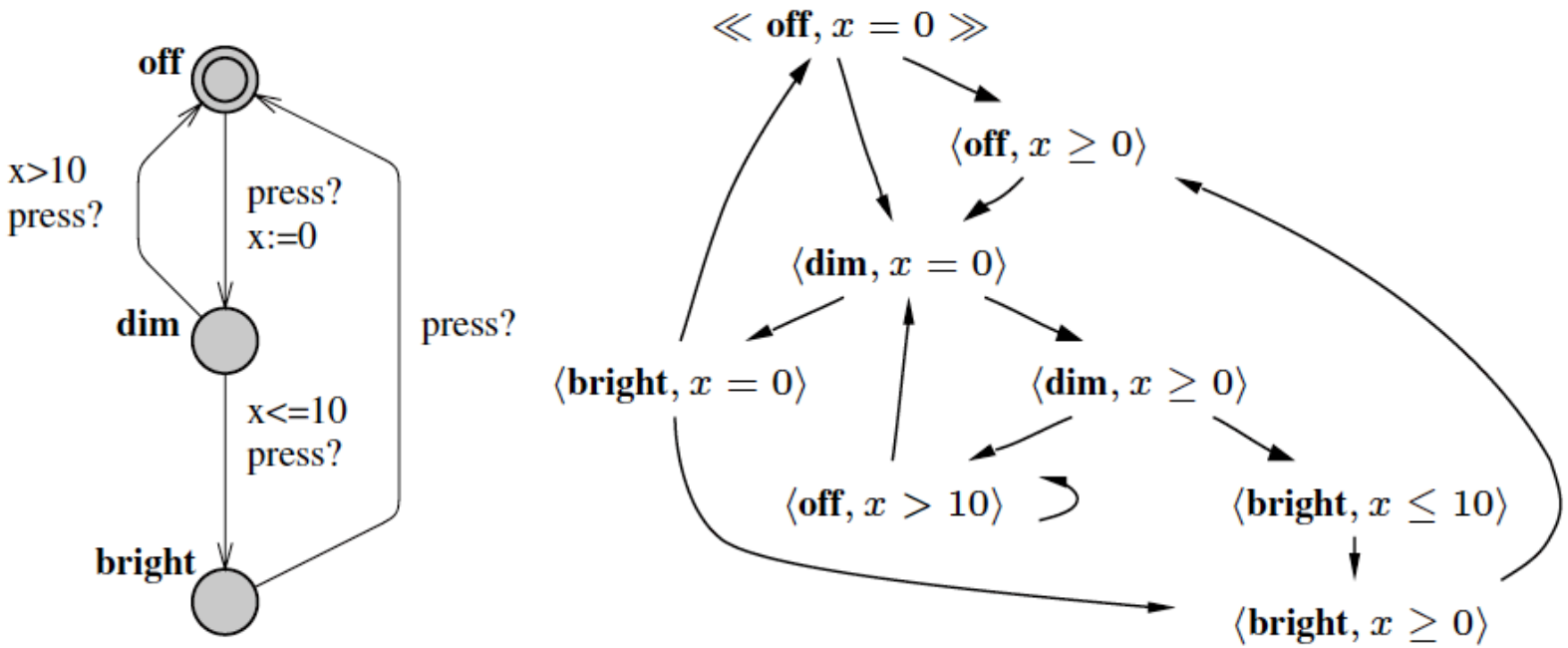


Fig. 3. A Timed Automaton and its Zone Graph.

Mais attention à l'infini !

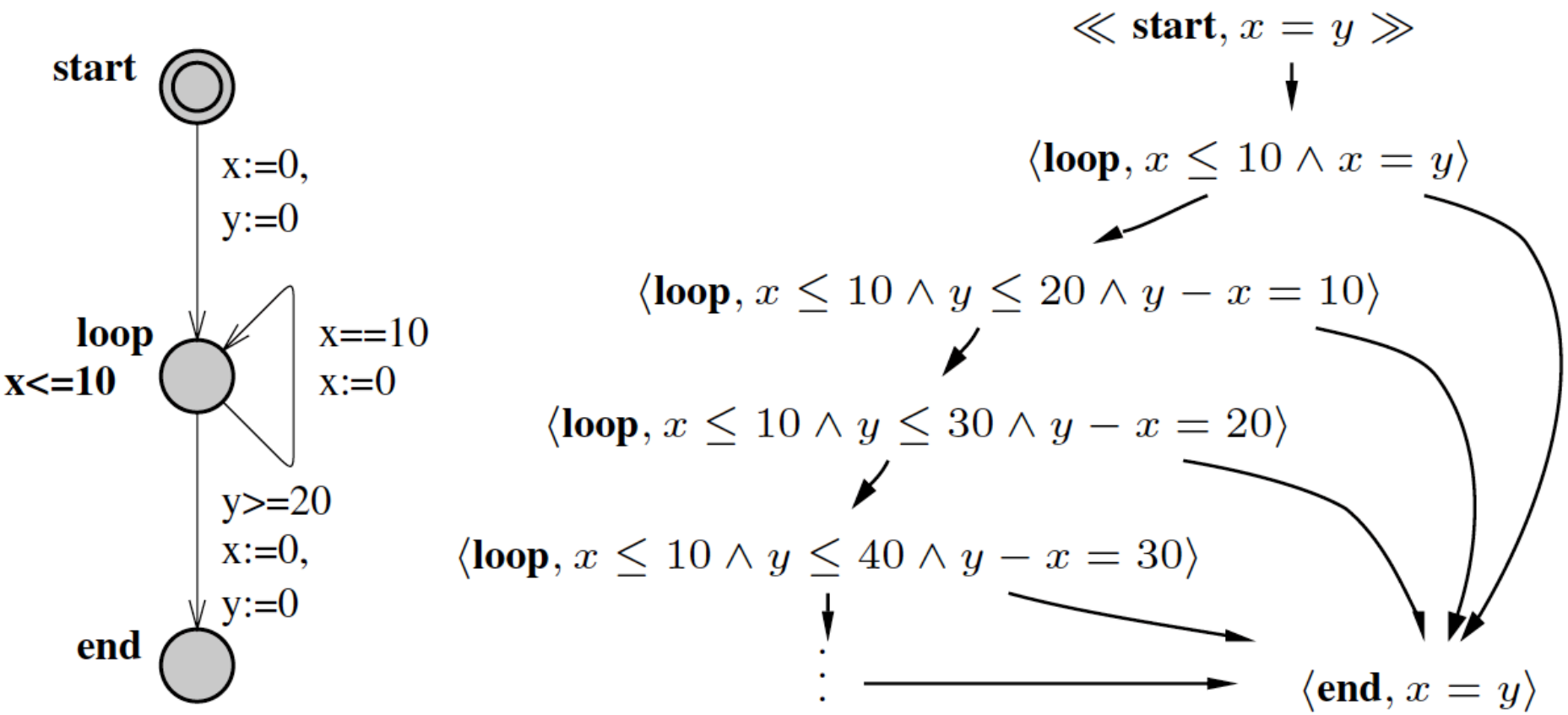


Fig. 4. A Timed Automaton with an Infinite Zone-Graph.

Normaliser = Couper par la plus grande constante (20)

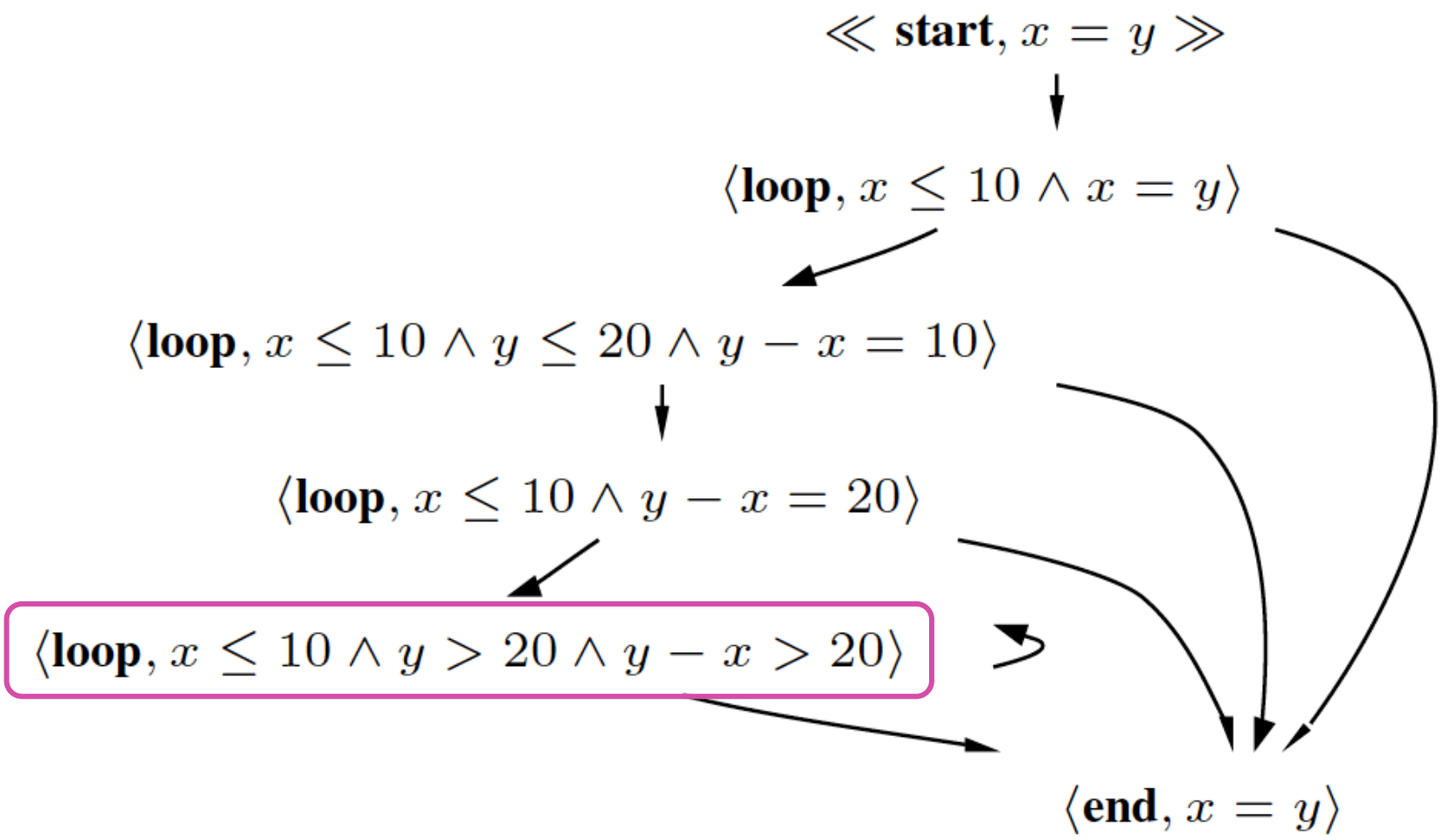


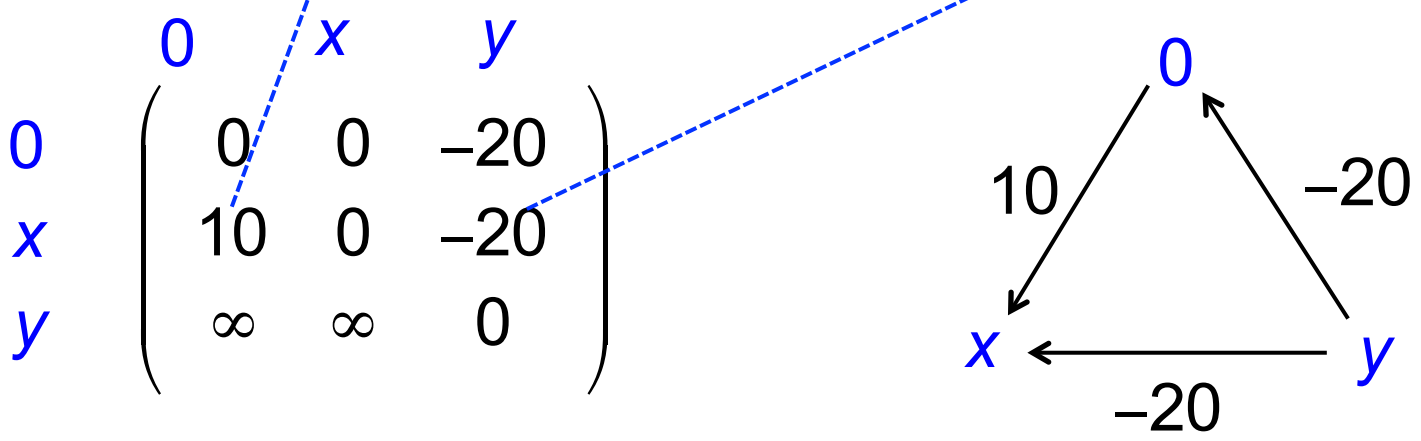
Fig. 5. Normalized Zone Graph for the Automaton in Fig. 4.

Zones, DBMs et graphes

- DBMs (Difference Bound Matrices)
  = représentation efficace des zones
- Graphes = autre vision des DBMs

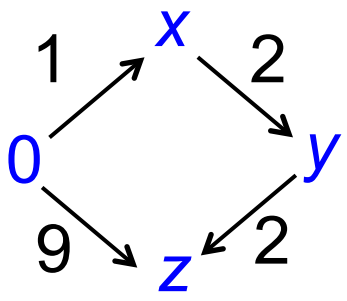
$$x \le 10 \wedge y \ge 20 \wedge y - x \ge 20$$

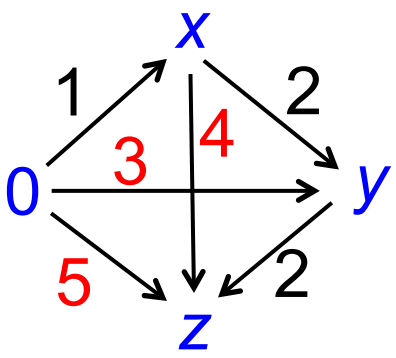
$$\rightarrow\ x - 0 \le 10 \wedge 0 - y \le -20 \wedge x - y \le -20$$



Normalisation par les plus courts chemins

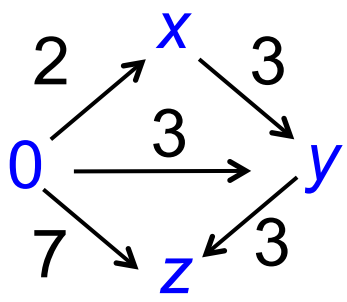
$Z1 : x \le 1 \wedge y - x \le 2 \wedge z - y \le 2 \wedge z \le 9$

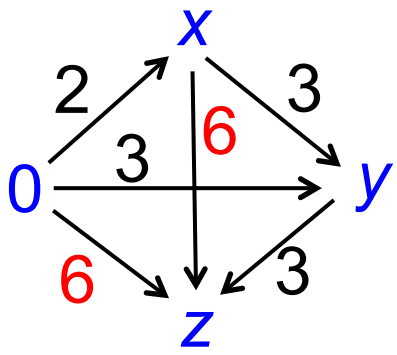




$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & \infty & \infty \\ 3 & 2 & 0 & \infty \\ 5 & 4 & 2 & 0 \end{pmatrix}$$

$Z2 : x \le 1 \wedge y - x \le 3 \wedge z - x \le 3 \wedge z - y \le 3 \wedge z \le 7$





$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & \infty & \infty \\ 3 & 2 & 0 & \infty \\ 6 & 6 & 2 & 0 \end{pmatrix}$$

$Z1 \subset Z2$?

Oui, car $Z2$ supérieur partout !

Compression : élimination des arcs redondants

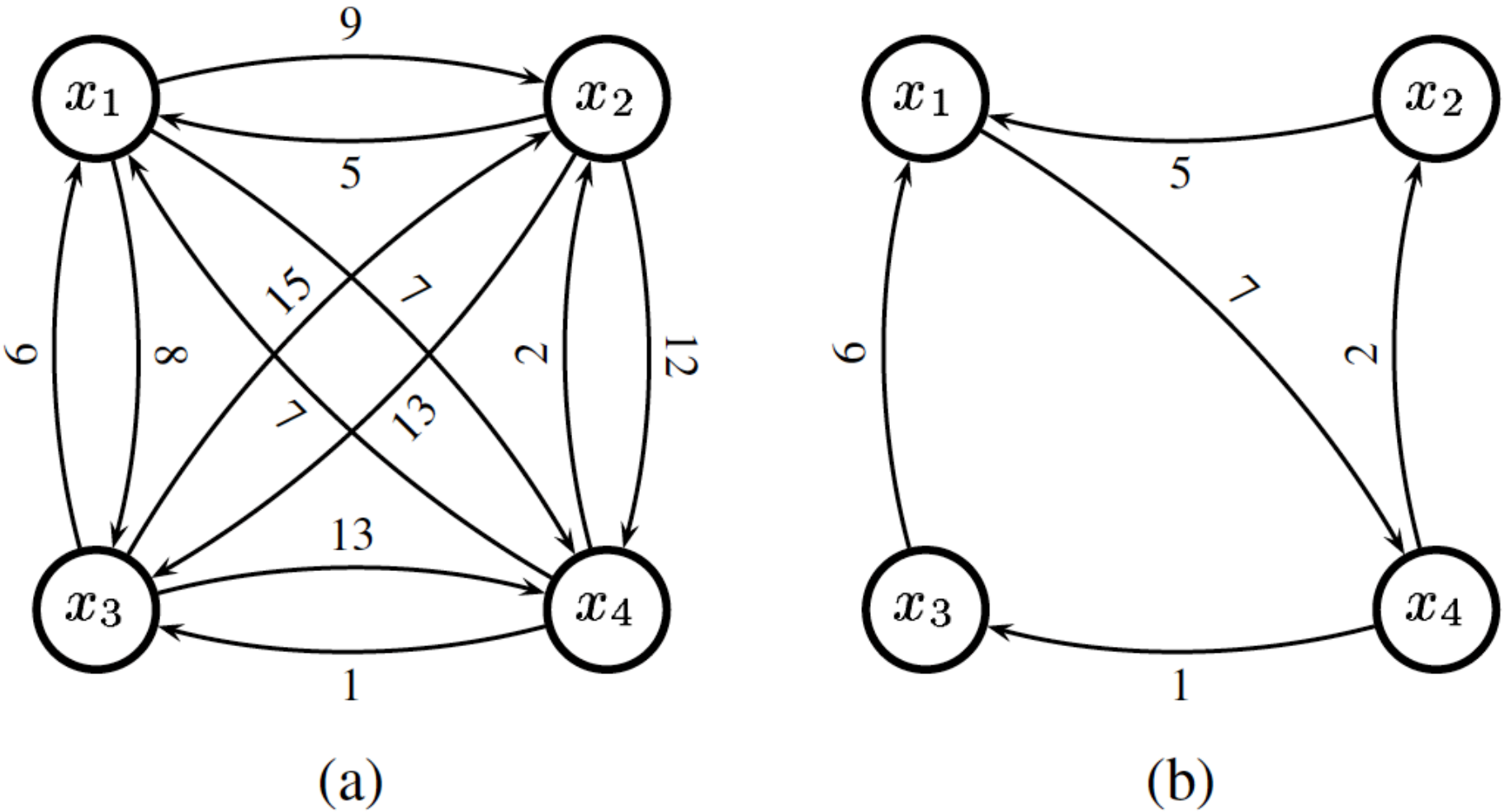


Fig. 9. A zero cycle free graph and its minimal form

Compression : élimination des arcs redondants

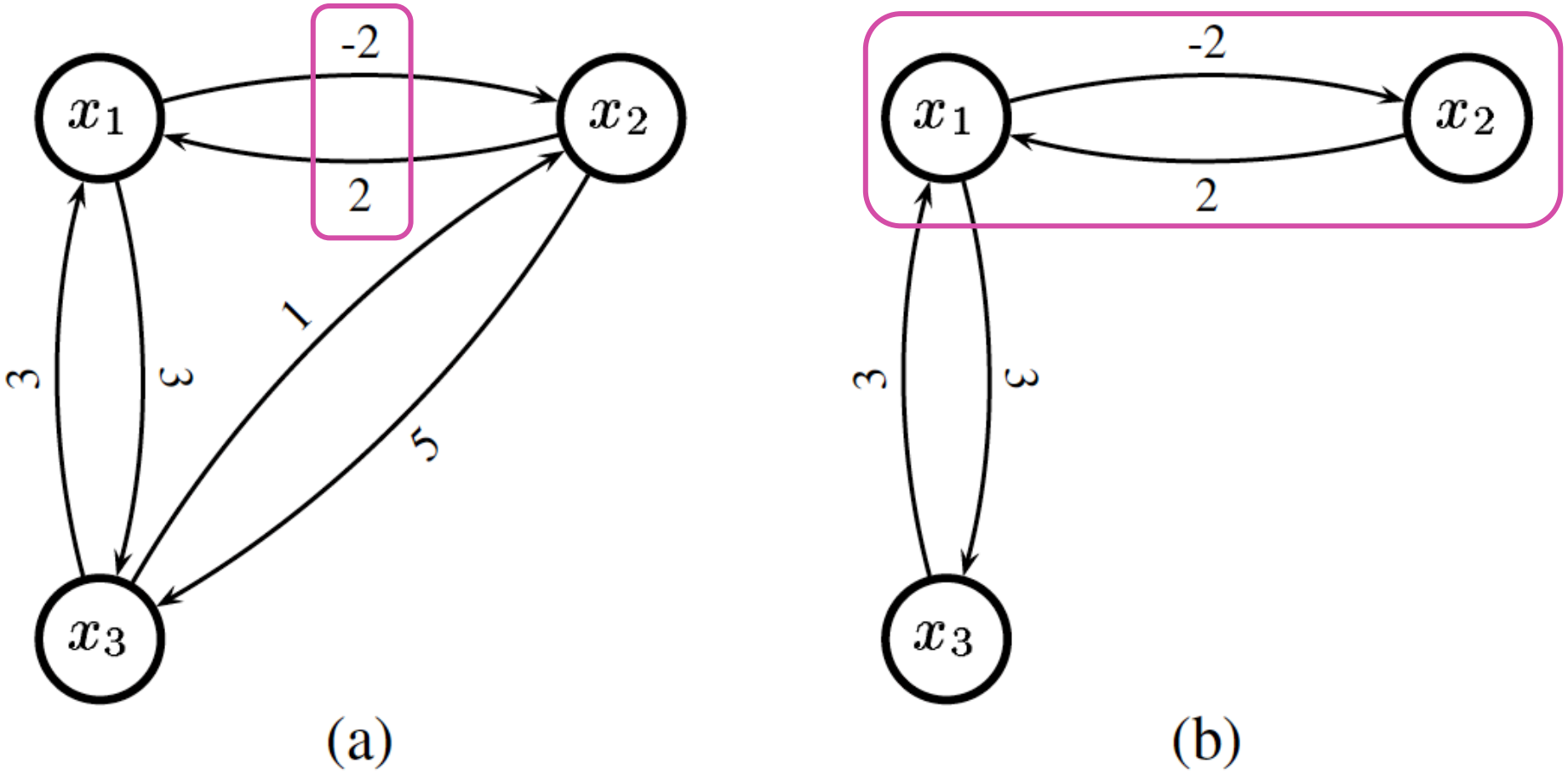
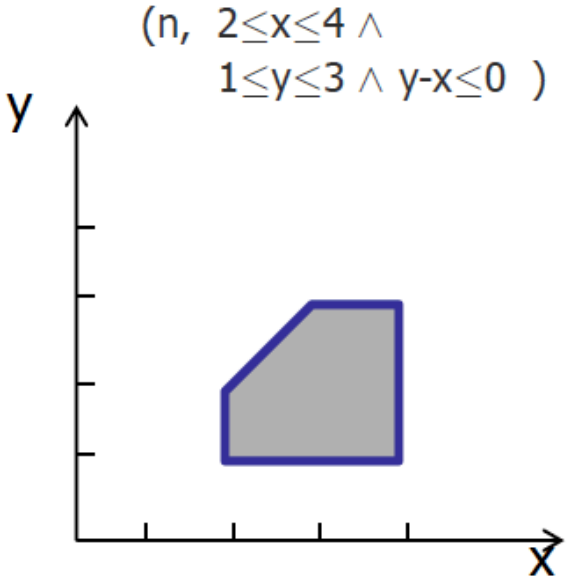


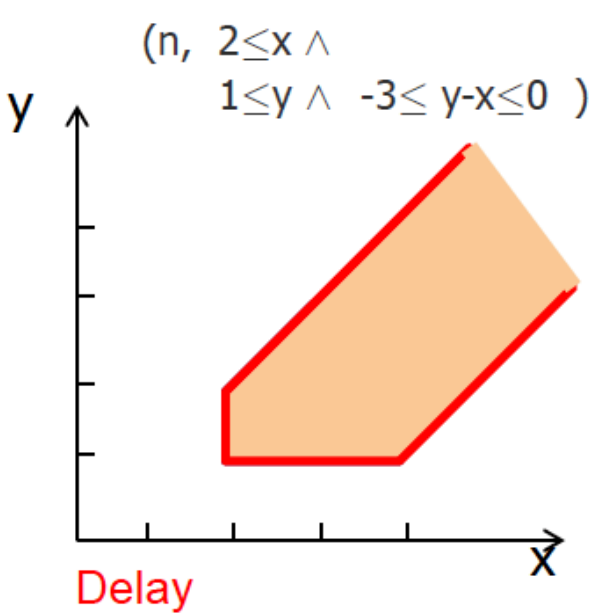
Fig. 10. A graph with a zero-cycle and its minimal form

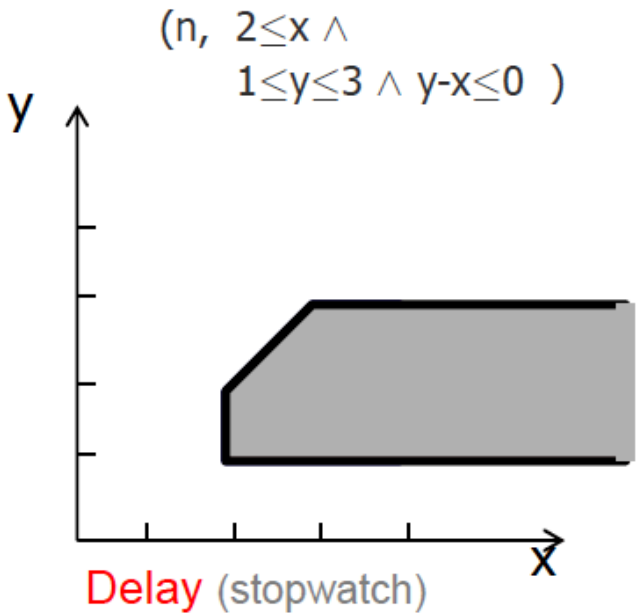
- vide(Z) : existence d'un cycle de poids négatif
- $Z^{\uparrow}$: passage du temps. Mise à l'infini des bornes des horloges en gardant les autres contraintes
- $Z \cap (x_i - x_j \leq c)$: ajout de contrainte. Modifier l'entrée correspondante, puis re-canoniser intelligemment.
- $Z + (x_i := 0)$: reset d'horloge. Mettre à 0 les entrées $(0,i)$ et $(i,0)$ de la matrice correspondante, puis recanoniser intelligemment.
- $\text{norm}_k(Z)$: normalisation par la constante k.
  1. supprimer $x_i - x_j \leq c$ si $c > k$
  2. changer $x_i - x_j \leq c$ *en* $x_i - x_j \leq -k$ si $c < -k$
  3. Recanoniser intelligemment
- ...

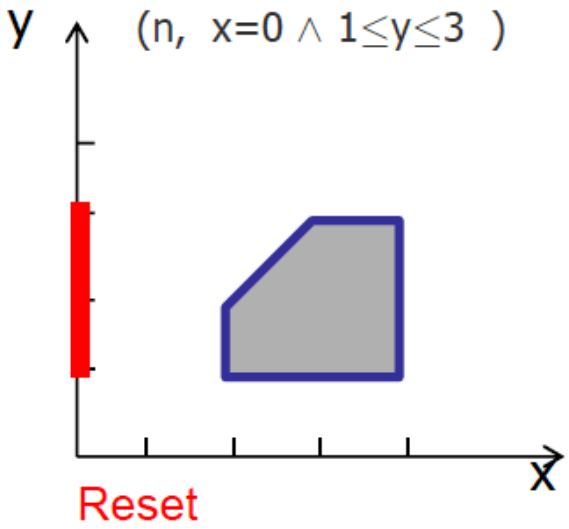
Et tout cela en gérant au mieux la mémoire, surtout les unions de zones à l'aide de structures partagées [BB.]

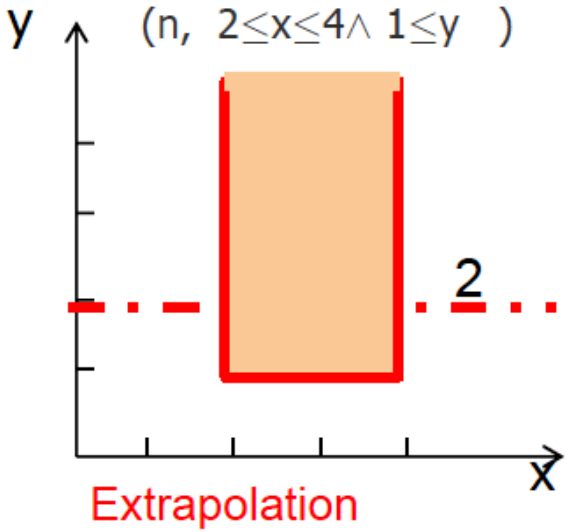
Les opérations vues graphiquement

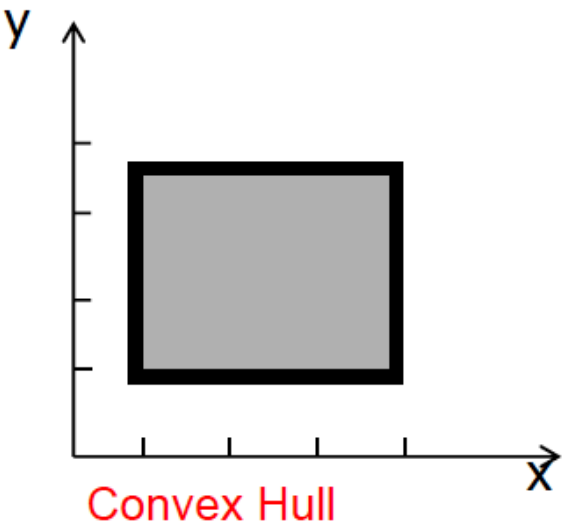












source Kim Larsen

Transformation d'une zone par une transition

$$\ell \xrightarrow{\phi, a, r} \ell'$$

$$Z \to Z' = \mathrm{norm}_K(\mathit{reset}_r(Z^{\uparrow} \cap \phi)) \cap I(\ell')$$

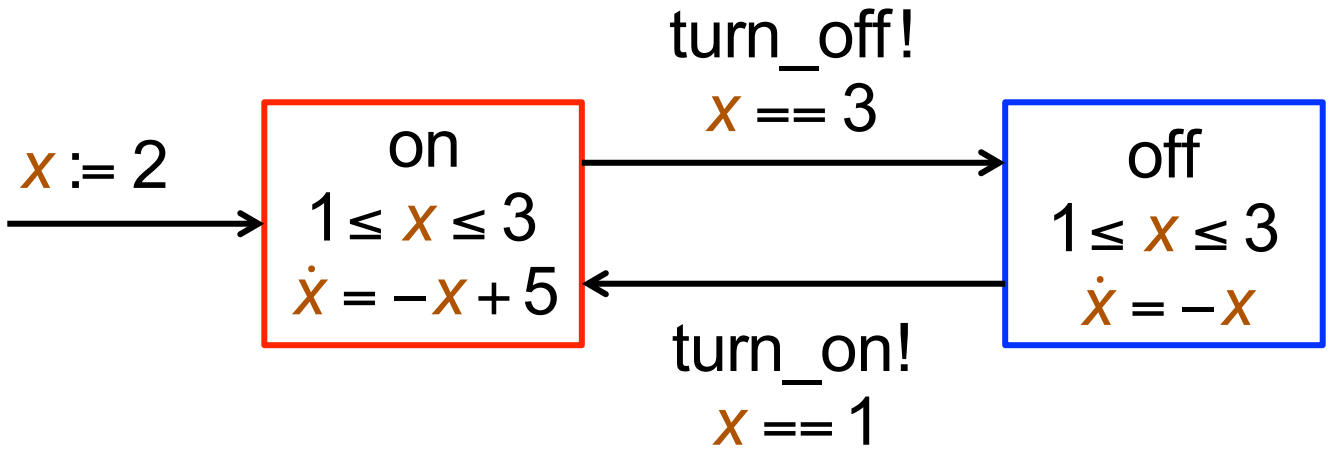
$$\ell_5 \xrightarrow{x \le 5 \wedge y - z \le 3 \quad a \quad x := 0, u := 0} \ell_{23} \qquad \boxed{\begin{array}{l} y \le 100 \\ t - u < 0 \end{array}}$$

$$Z \to Z' = \mathrm{norm}_K(\mathrm{reset}_{x,u}(Z^{\uparrow} \cap (x \le 5 \wedge y - z \le 3))) \cap (y < 100 \wedge t - u < 0)$$

- Model-checking par exploration en avant, avec normalisation par la plus grande constante du problème.
- Fabrication de contre-exemples par retour arrière
- Optimisation poussée, cf. *Uppaal Implementation Secret*

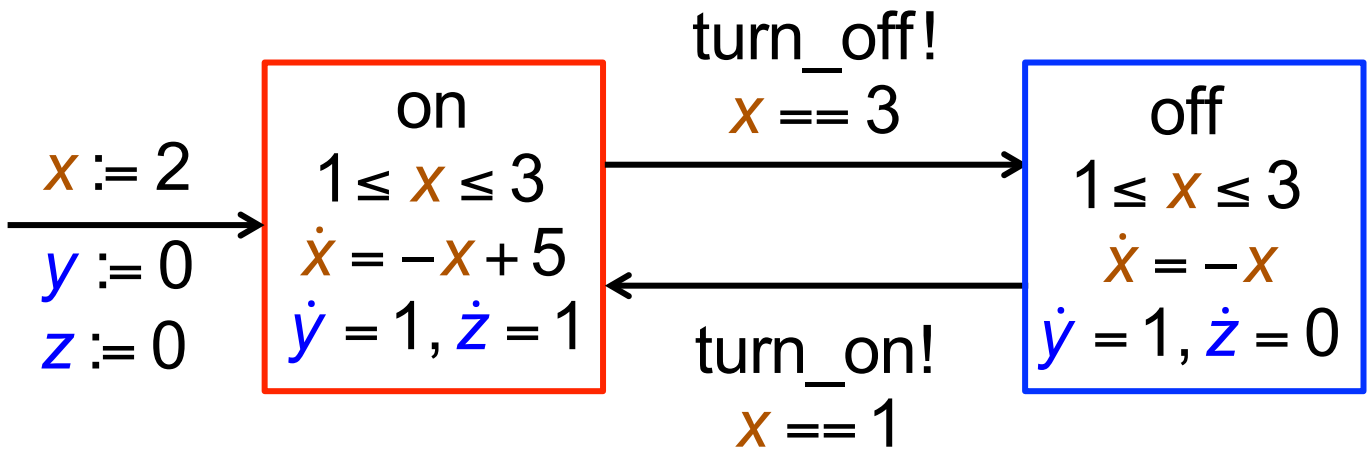
Automates hybrides (Henzinger et. al.)

- Ne plus se contenter des horloges, mais modéliser mieux la physique à l'aide de variables réelles et d'équations différentielles ordinaires (EDOs)



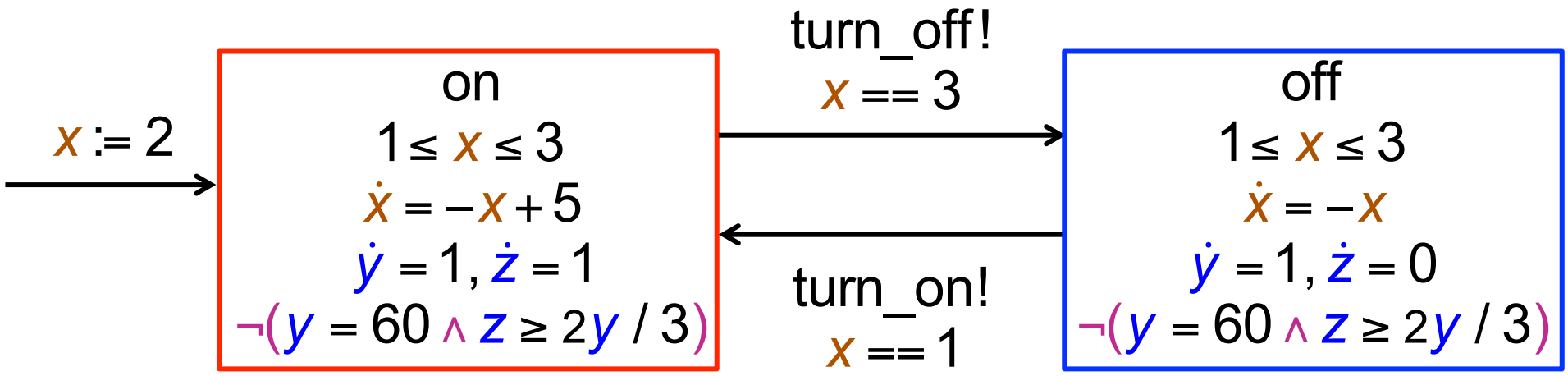
- A démontrer : le radiateur n'est pas allumé plus de 2/3 du temps pendant la première heure

Ajout d'horloges pour compter le temps



y compte le temps total
z compte le temps allumé

Ajout du prédicat de sûreté



Deux solutions pour prouver la sûreté :

1. Traduire les EDO en comportement d'horloges, en étudiant le temps que met x à atteindre les bornes, puis faire comme pour les automates temporisés.
2. Faire des sur-approximations linéaires suffisantes pour prouver les propriétés

1. Résoudre l'équation et introduire des horloges

- Equations successives de x :

Initial : $x := 2 \rightarrow \dot{x} = -x + 5 \Rightarrow x = -3e^{-t} + 5$

$x = 3 \Leftrightarrow t = \ln(3/2)$

on → off : $x = 3 \rightarrow \dot{x} = -x \Rightarrow x = 3e^{-t}$

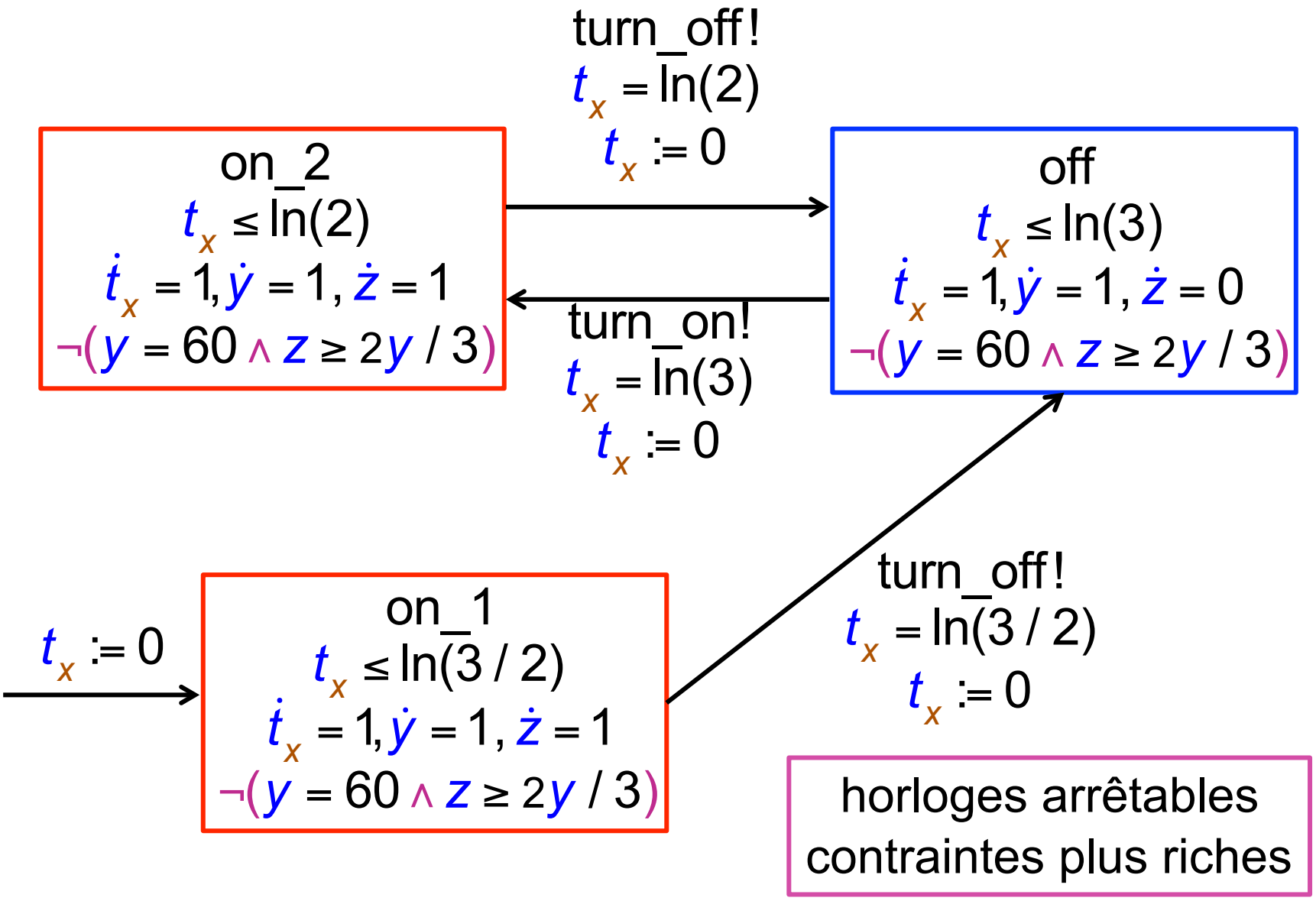
$x = 1 \Leftrightarrow t = \ln(3)$

off → on : $x = 1 \rightarrow \dot{x} = -x + 5 \Rightarrow x = -4e^{-t} + 5$

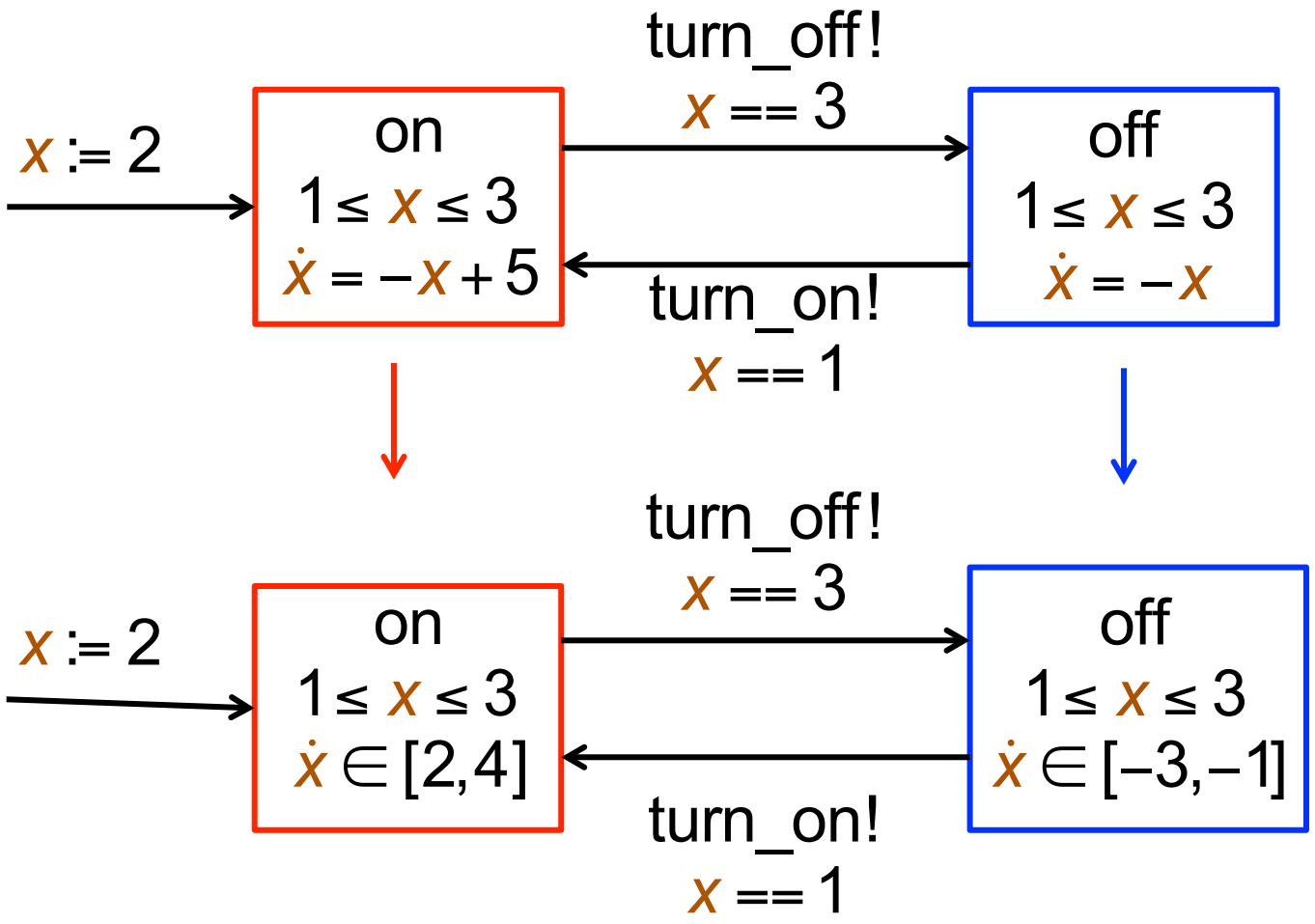
$x = 3 \Leftrightarrow t = \ln(2)$

Remplacement de x par une variable t_x telle que $\dot{t}_x = 1$, i.e., une horloge remise à 0 à chaque transition !

Automate temporisé (enfin, presque)

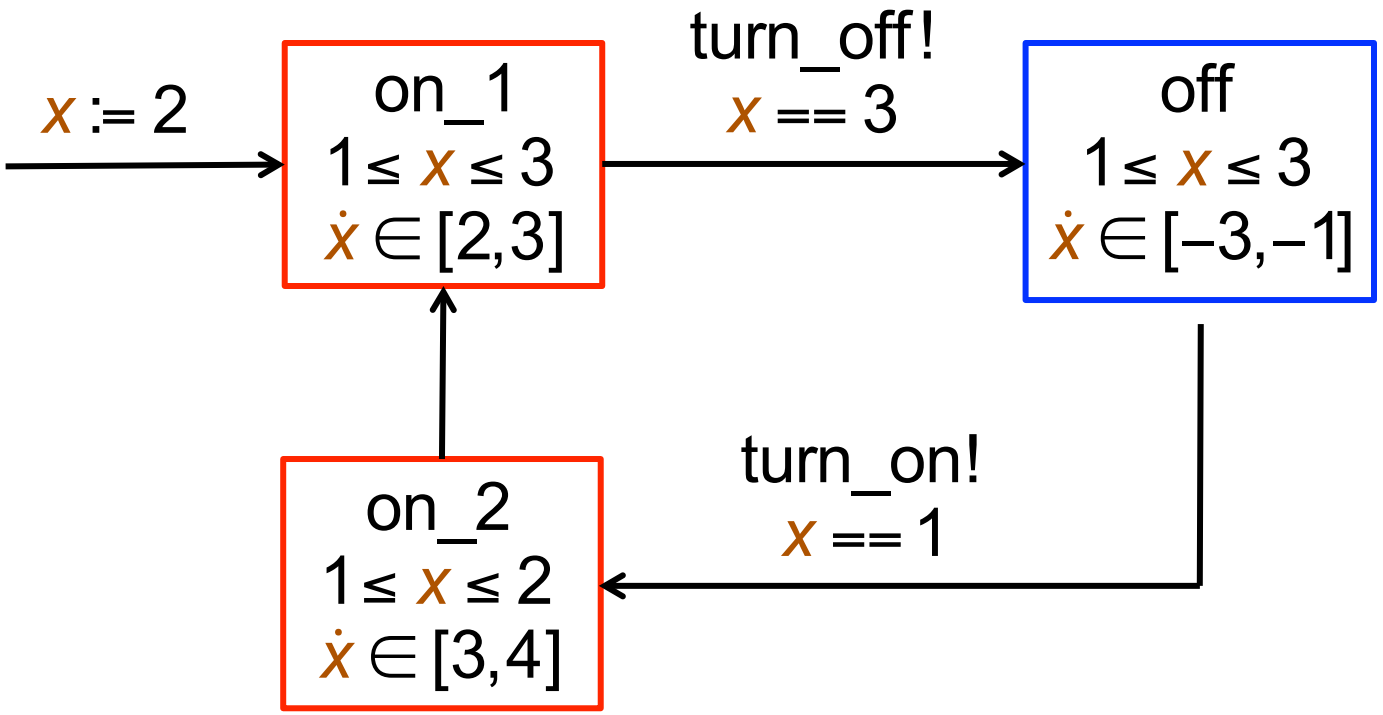


Solution 2 : interpolation linéaire



Cet encadrement suffit à faire la preuve, cf [HOWT]

Plus fin : interpolation + découpage d'état



Plus fin, nécessaire si l'interpolation initiale ne passe pas, mais plus cher à calculer !

Conclusion

- Parler du temps est essentiel dans beaucoup d'applications → automates temporisés et hybrides
- La logique temporelle peut vraiment parler du temps !
  - UPPAAL utilise CTL (simplifiée) + les horloges
  - autres logiques : TLTL = LTL + temps ($\phi U_t \phi'$), indécidable
- Le calcul des régions montre la décidabilité
- Zones et DBMs permettent le calcul efficace
- Extensions : analyses statistiques (UPPAAL), robustesse à l'imprécision du temps (P. Bouyer), etc.
- Les automates hybrides sont plus puissants mais plus complexes

Et UPPAAL est devenu un système de référence

Bibliographie – Automates temporisés

[BDL] G. Behrmann, A. David et K. Larsen,
A Tutorial on UPPAAL
Formal Methods for the Design of Real-Time Systems,
Springer LNCS 3185, pp. 200-236 (2006
http://www.uppaal.com/admin/anvandarfiler/filer/uppaal-tutorial.pdf

[BWY] J. Bengtsson et Wang Yi,
Timed Automata Semantics, Algorithms and Tools.
Lecture Notes on Concurrency and Petri Nets, Springer (2004)

[BFL] P. Bouyer, U. Fahrenberg, K. Larsen,
Verification of Real-Time Systems (2013)
A paraître dans le « Handbook of Model-Checking »

[BB.] G. Behrmann, J. Bengtsson, A. David, K. Larsen, P. Petterson et Wang Yi
UPPAAL Implementation Secrets
Proc. FTRTFT 2002, Springer LNCS 2469, pp. 3-22 (2002)

Bibliographie – Automates hybrides

Rajeev Alur, Costas Courcoubetis, Nicolas Halbwachs, Thomas A. Henzinger, Pei-Hsin Ho, Xavier Nicollin, Alfredo Olivero, Joseph Sifakis, et Sergio Yovine
The algorithmic analysis of hybrid systems.
Theoretical Computer Science, volume 138(1), pages 3–34, 1995.

[HHW] T. Henzinger, Pei-Hsin Ho et H. Won-Toi
Hytech : A Model-Checker for Hybrid Systems
Proc. Computer-Aided Verification CAV'2005

... et une littérature très abondante !