MOdélisation et VALidation

Franck Cassez

IRCCyN/CNRS UMR 6597,

1 rue de la Noë,

B.P. 92101, 44321 Nantes Cedex 03

e-mail : Franck.Cassez@irccyn.ec-nantes.fr

6 février 2002

Quelques caractéristiques des systèmes embarqués

- *réactivité* (contrôler un environnement)
  systèmes d'exploitation, protocoles de communication
- *complexité* (activités parallèles, systèmes répartis et communicants)
  réseaux, BDs réparties
- *criticité* (enjeux humains ou financiers)
  transports, nucléaire, téléphone, médical, etc
- *non-interopérabilité* (. . . embarqués . . .)
  robots marsiens, etc

$\Longrightarrow$ méthodes de développement rigoureuses et *vérification*

Méthodes Formelles [8, 23]

Comment et quoi vérifier ?

quoi : des *spécifications formelles* sur un *modèle formel* du système

comment :

- **démonstration automatique :** . . . pas automatique – universel (on put théoriquement tout prouver) – outils : theorems provers
- **model-cheking :** automatique – limitation : taille du système – outils : model-chekers
- **test :** pas exhaustif – possible sur des implémentations de grande taille – outils : générateurs de tests

Vérification d'un système réactif

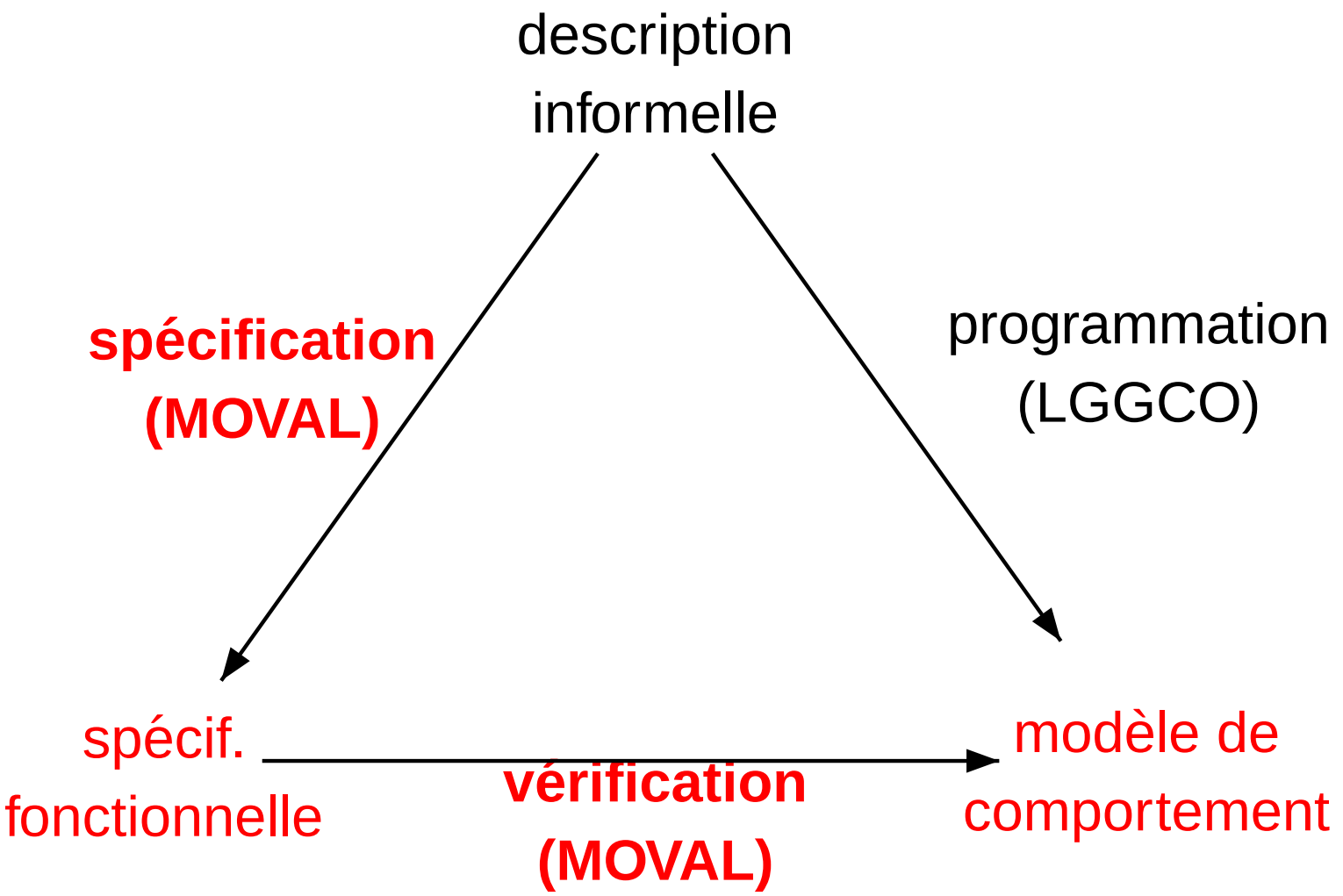


Table des matières

Première partie : Modélisation et spécification des systèmes concurrents

Chapitre 1 : Machines à états

Sommaire

1.1– Systèmes de transitions

Définition 1 (Système de transitions (SdeT) [4]) *Un SdeT S c'est :*

- Q *un ensemble (fini) d'états,*
- s_0, *un état initial*
- A *un alphabet (fini) d'actions,*
- $\longrightarrow \subseteq Q \times A \times Q$ *une relation de transition.*

$\sigma = s_0 \xrightarrow{l_0} s_1 \dots s_n \xrightarrow{l_n} s_{n+1} \dots$ *est un chemin (exécution) de* S *si* $\forall i \geq 0, (s_i, l_i, s_{i+1}) \in \longrightarrow$

- $l_0 l_1 \cdots l_n \cdots$ *est la trace de* σ
- s *est accessible* $\iff \exists s_0 \xrightarrow{w} s,\ w \in A^*$
- $Reach(S)$ *est l'ensemble des états accessibles dans* S
- *Un SdeT étiqueté est un SdeT avec une fonction* $L : Q \to P$. □

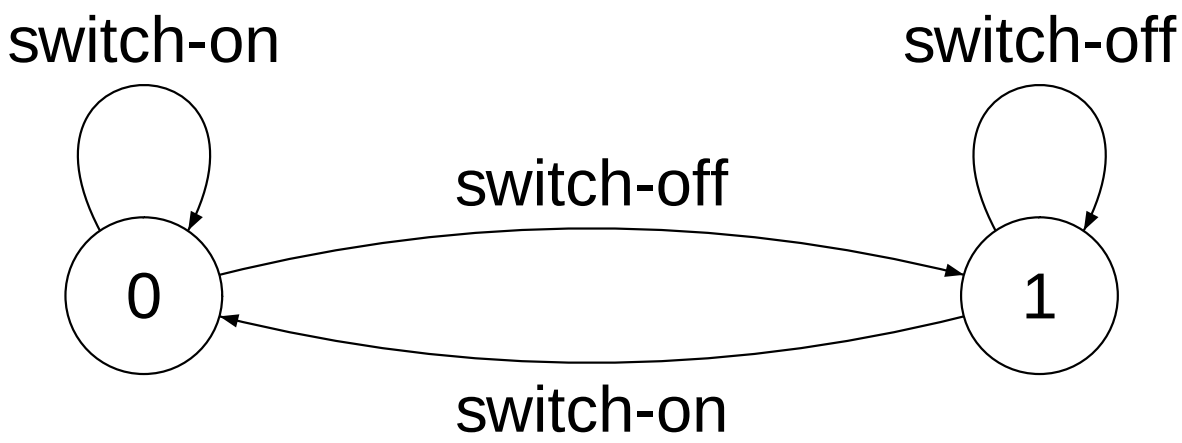


FIG. 1.1 – SdeT représentant un interrupteur

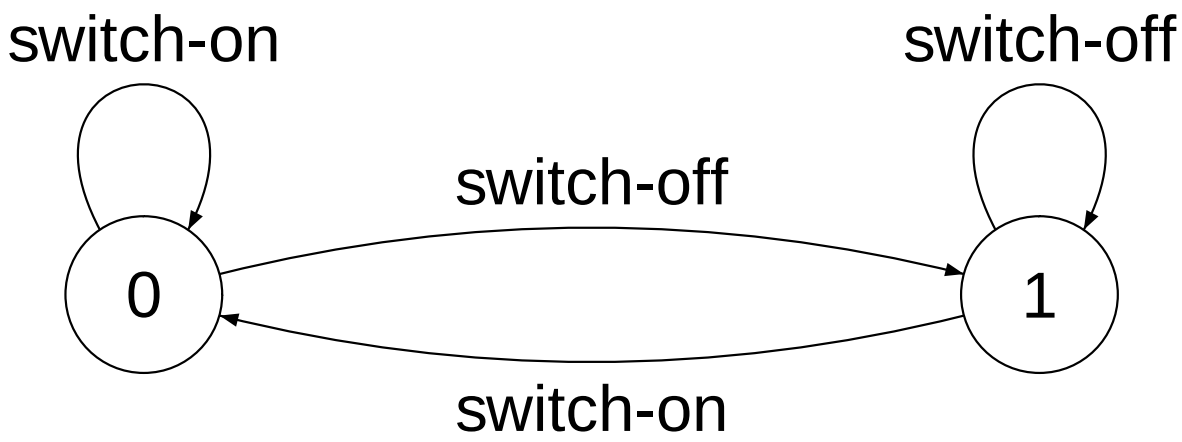


FIG. 1.2 – SdeT étiqueté représentant un interrupteur

$$L(0) = \{\mathsf{On}, \mathsf{Init}\} \qquad L(1) = \{\mathsf{Off}\}$$
$$L^{-1}(\mathsf{On}) = \{0\} \qquad L^{-1}(\mathsf{Off}) = \{1\}$$

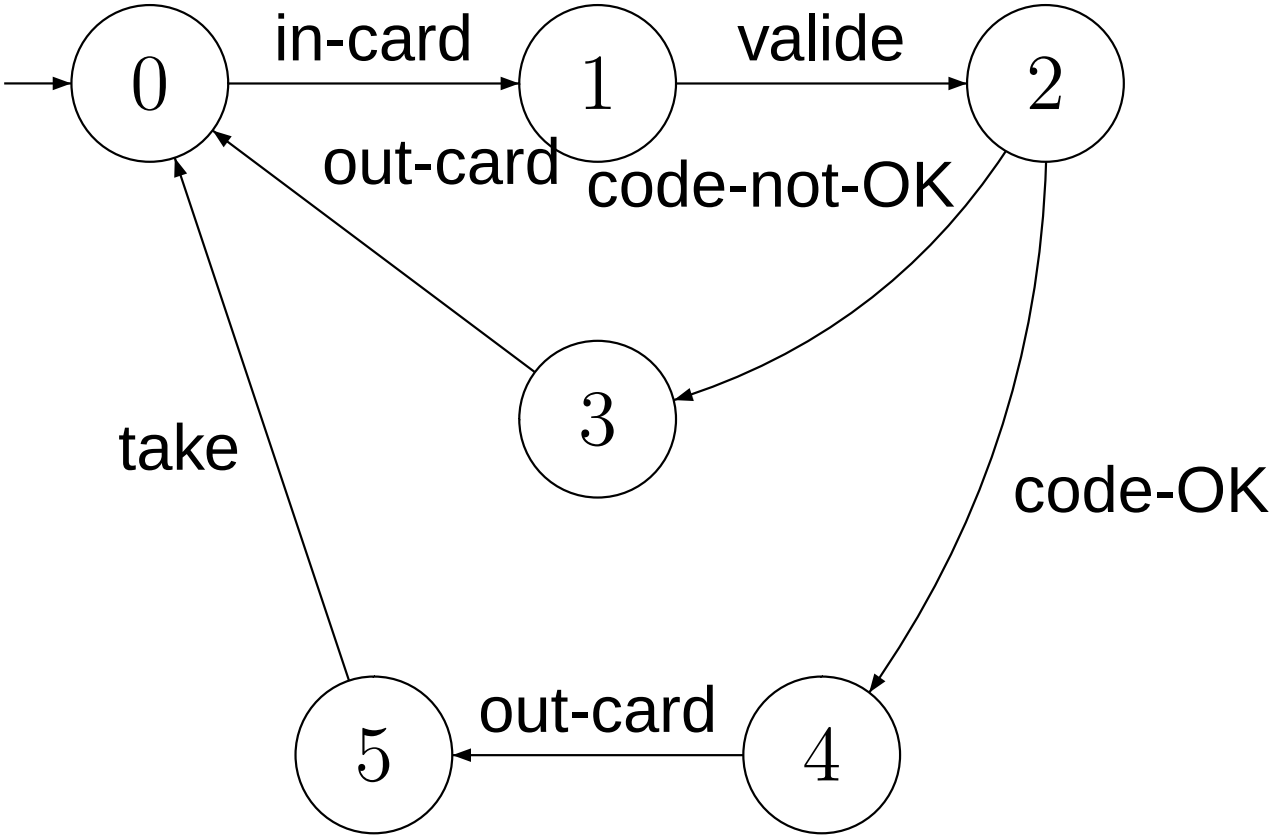


FIG. 1.3 – SdeT représentant le fonctionnement d'un GAB

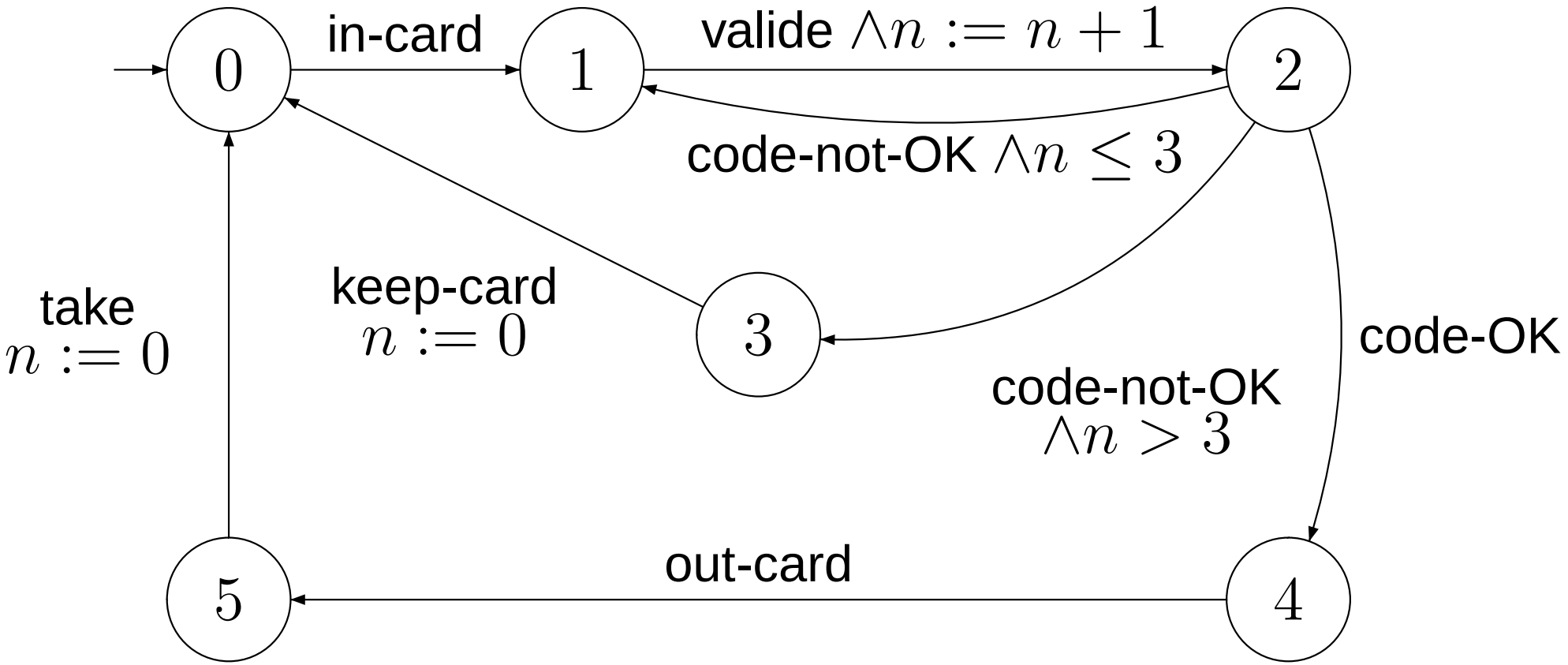


FIG. 1.4 – Un GAB plus perfectionné

Utilisation de variables discrètes

- ensemble V de valeurs *fini* $\Longleftrightarrow$ SdeT *fini*
  $v := k$
  $v := [i, j], \ldots$
- description concise
- nombre réel d'états $\leq |V| \times |Q|$
- système réel $=$ *dépliage* du système avec variables
- variantes des modèles : voir [20]

Exemple : introduire une/des variables pour le code.

1.2– Exécutions d'un SdeT

langage : le *langage $L(S)$ accepté* par un SdeT S est l'ensemble des traces de toutes les exécutions du SdeT à partir de s_0

arbre : l'*arbre d'exécution* $A(S)$ associé à un SdeT est défini par :

- la *racine* : $(0, s_0)$
- si $(n, s) \in A(S)$ et $(s, e, s') \in \longrightarrow$ alors $(s', n+1)$ est un *fils* de (n, s)

Remarque : $L(S)$ comprend des mots finis et/ou infinis – $A(S)$ peut être un arbre infini.

1.3– Synchronisation des SdeTs

- un SdeT = description *d'un composant* d'un système
- un système réel = un ensemble de modules *interagissants*
- but : construire le système *global* à partir des sous-systèmes
- moyen : décrire *l'interaction* entre les sous-systèmes

formalisation : produit synchronisé de SdeT à la Arnold-Nivat [4]

1.3.1– Produit libre de SdeTs

Définition 2 (Produit libre (ou cartésien) [4]) $S_i = (Q_i, s_0^i, A_i, \longrightarrow_i)$, n *SdeTs. Le produit libre* $S_1 \parallel S_2 \cdots \parallel S_n$ *des SdeTs* S_i *est un SdeT* $S = (Q, s_0, A, \longrightarrow)$ *défini par :*

- $Q = Q_1 \times Q_2 \times \cdots \times Q_n$,
- $s_0 = (s_0^1, s_0^2, \ldots, s_0^n)$,
- $A = A_1 \times A_2 \times \cdots \times A_n$,
- $\longrightarrow \subseteq Q \times A \times Q$:

$$(q_1, \ldots, q_n) \xrightarrow{(a_1,\ldots,a_n)} (q'_1, \ldots, q'_n) \iff \forall i, q_i \xrightarrow{a_i} q'_i$$

□

1.3.2– Produit synchronisé de SdeTs

Définition 3 (Produit synchronisé [4]) $S_i = (Q_i, s_0^i, A_i, \longrightarrow_i)$, n *SdeTs.* $I \subseteq A_1 \times A_2 \times \cdots \times A_n$ *une contrainte de synchronisation. Le produit synchronisé* $(S_1 \parallel S_2 \cdots \parallel S_n)_{Sync}$ *des SdeTs* S_i *est un SdeT* $S = (Q, s_0, A, \longrightarrow)$ *défini par :*

- $Q = Q_1 \times Q_2 \times \cdots \times Q_n$,
- $s_0 = (s_0^1, s_0^2, \ldots, s_0^n)$,
- $A = A_1 \times A_2 \times \cdots \times A_n$,
- $\longrightarrow \subseteq Q \times A \times Q$:

$$(q_1, \ldots, q_n) \xrightarrow{(a_1,\ldots,a_n)} (q'_1, \ldots, q'_n) \iff \begin{cases} (a_1, \ldots, a_n) \in I \\ \wedge \left(\forall i, q_i \xrightarrow{a_i} q'_i\right) \end{cases}$$

□

Communication par envoi/réception de messages

- but : décrire *envoi/réception* de messages : communication *synchrone*
- principe : étiquettes de transitions envoi $=!m$, réception $=?m$
  implicitement : $!m$ initie la communication
- description de la communication : synchronisation de $!m$ et $?m$

Action “ne rien faire”

- nouvelle étiquette : $\bullet =$ “ne rien faire”
- modification de la définition 1 : $\forall q \in S, q \xrightarrow{\bullet} q$ et alphabet $A \cup \{\bullet\}$
- modélisation de *l'asynchronisme* entre systèmes : produit synchronisé (3) avec des $\bullet$: $(\bullet, \ldots, \bullet, a_k, \bullet, \ldots, \bullet, a_j, \bullet, \ldots, \bullet)$

Exemple : Le GAB (1.4) et un utilisateur

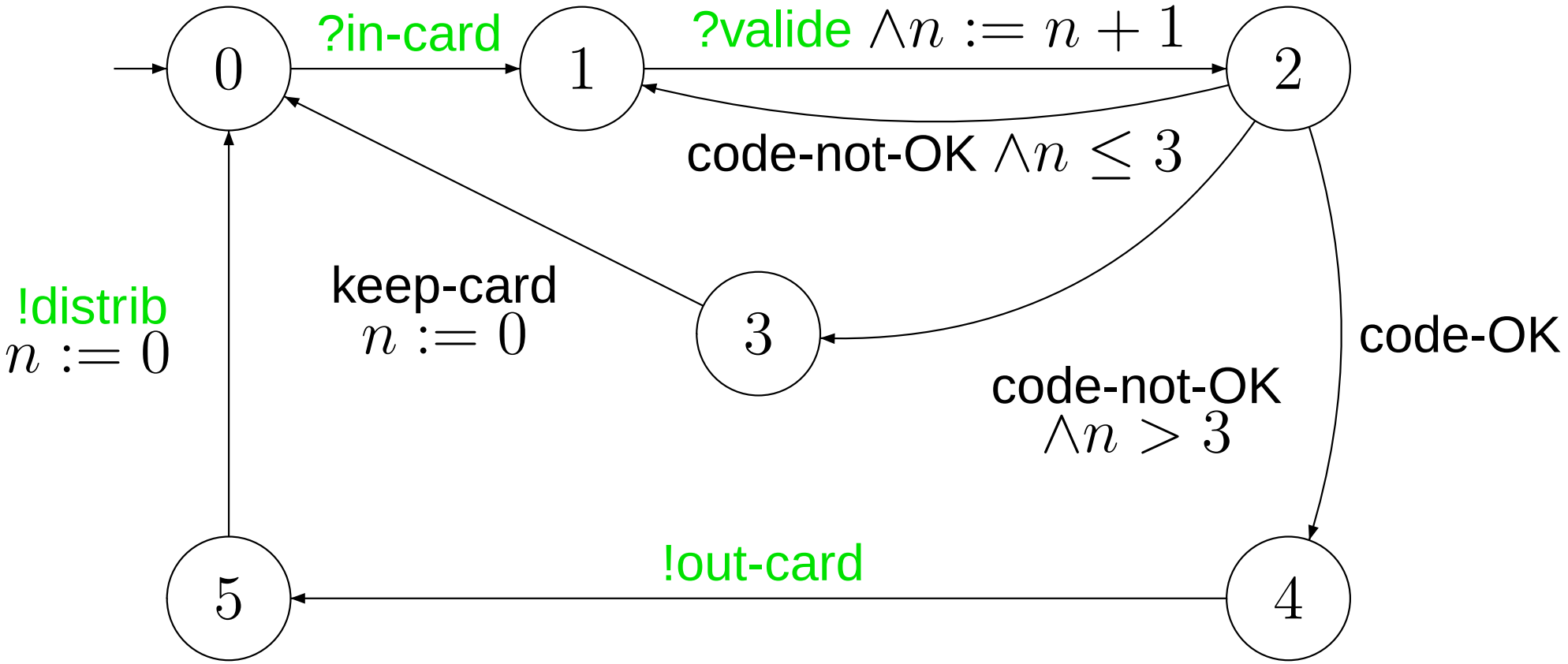


FIG. 1.5 – GAB avec messages

Exemple : Le GAB et un utilisateur (suite)

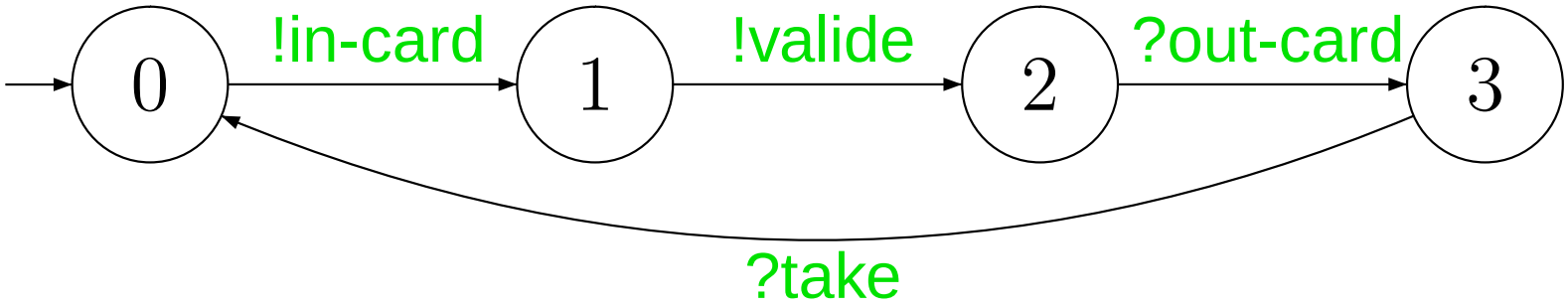


FIG. 1.6 – Un bon utilisateur

Contrainte de synchronisation I pour (GAB $\times$ U) :

(	?in-card,	!in-card)
(	?valide,	!valide)
(	!out-card,	?out-card)
(	!take,	?take)
(	code-OK,	•)
(	code-not-OK,	•)
(	keep-card,	•)

Exemple : Le GAB et un utilisateur (suite)

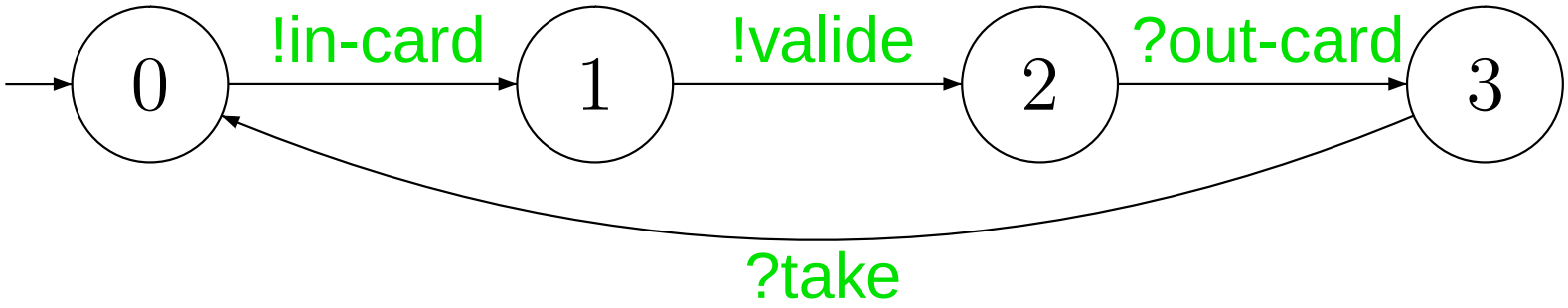


FIG. 1.7 – Un bon utilisateur

Contrainte de synchronisation I pour (GAB $\times$ U) :

(	?in-card,	!in-card	)	in
(	?valide,	!valide	)	valide
(	!out-card,	?out-card	)	out
(	!take,	?take	)	take
(	code-OK,	•	)	code-O
(	code-not-OK,	•	)	code-not-OK
(	keep-card,	•	)	keep-card

Renommage

– synchronisation à la Arnold-Nivat avec *renommage*
– définition 3 modifiée : f fonction *partielle* de synchronisation $f : A_1 \times A_2 \times \cdots \times A_n \longrightarrow A$ et

$$(q_1, \ldots, q_n) \xrightarrow{b} (q'_1, \ldots, q'_n) \iff \begin{cases} f(a_1, \ldots, a_n) = b \\ \wedge \left(\forall i, q_i \xrightarrow{a_i} q'_i\right) \end{cases}$$

Action invisible

– code-Ok, code-non-Ok, keep-card non vues par l'utilisateur (internes à GAB)
– action spéciale : τ

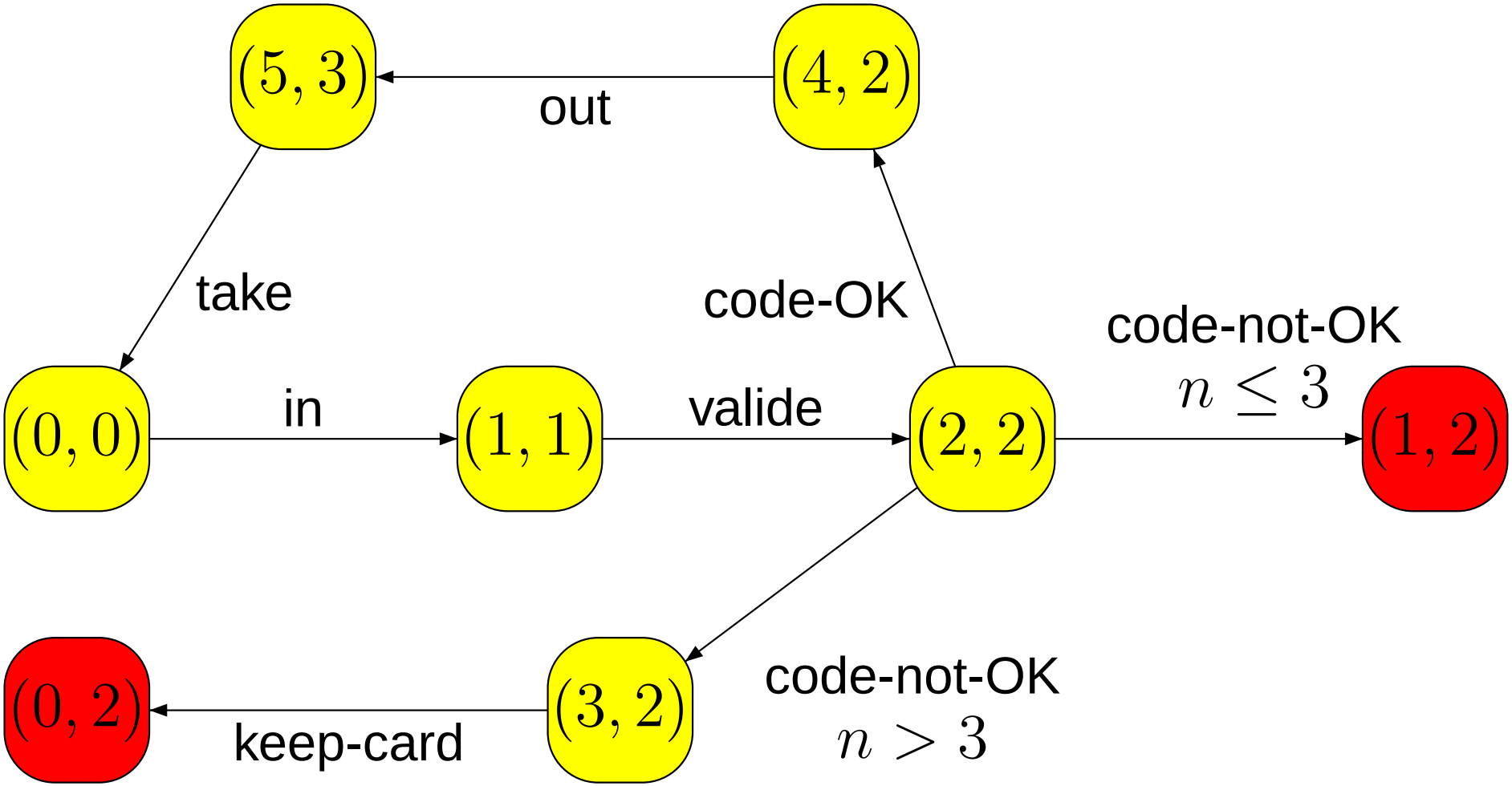


FIG. 1.8 – Produit synchronisé GAB $\times$ U – contrainte I

Synchronisation par variable(s) partagée(s)

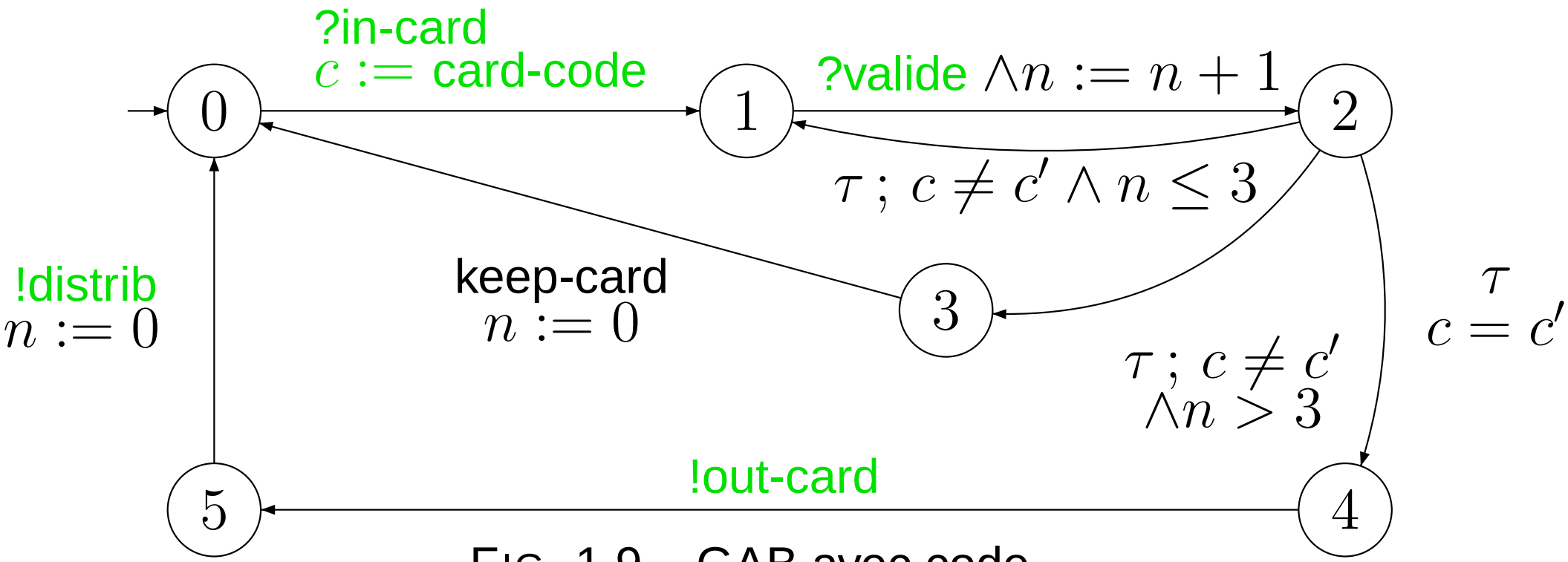


FIG. 1.9 – GAB avec code

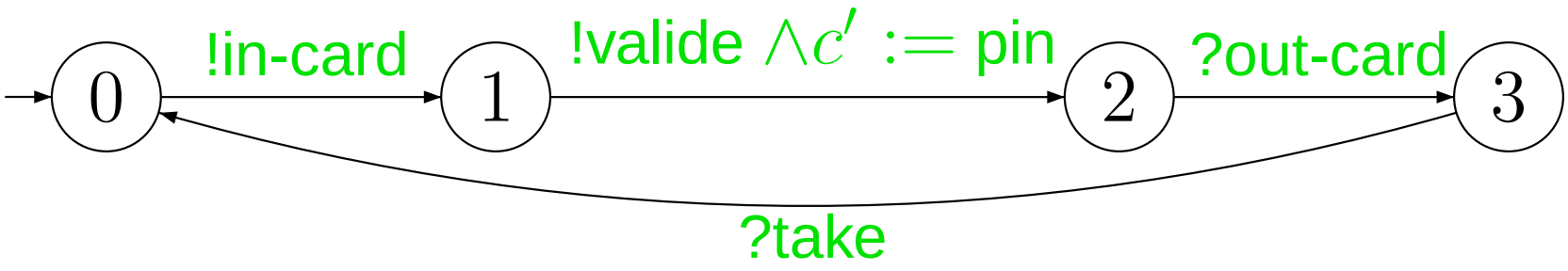


FIG. 1.10 – Utilisateur avec code PIN

Exécutions infinies

- action $\bullet$ toujours possible
- deadlock $=$ *état dont on ne sort jamais*
- exécution finie *complétée* en une exécution infinie
  $s \xrightarrow{a_1 a_2 \cdots a_n} s'$ devient $s \xrightarrow{a_1 a_2 \cdots a_n} s' \xrightarrow{\bullet} s' \xrightarrow{\bullet} \ldots$
- étude des exécutions infinies

Etats "puits"

deadlock : *blocage non souhaité* du système

- états sources d'aucune transition
- états desquels on ne peut sortir

état final : état *normal* d'arrêt

différence $\Longrightarrow$ utiliser étiquetage (1) : $L(s) = terminal$

1.4– Déterminisme – non déterminisme

Définition 4 (SdeT déterministe) *Un SdeT* $S = (Q, s_0, A, \rightarrow)$ *est déterministe ssi*

$$\forall s, s', s'' \in S, \forall a \in A, s \xrightarrow{a} s' \wedge s \xrightarrow{a} s'' \Longrightarrow s' = s''$$

Sinon il est non déterministe. □

– SdeT déterministe : traces identiques $\Longrightarrow$ exécutions identiques (une exécution possible)
– SdeT non déterministe : choix non déterministe du système (parmi un nombre fini de choix)

Exemple : Abstraction non déterministe du GAB

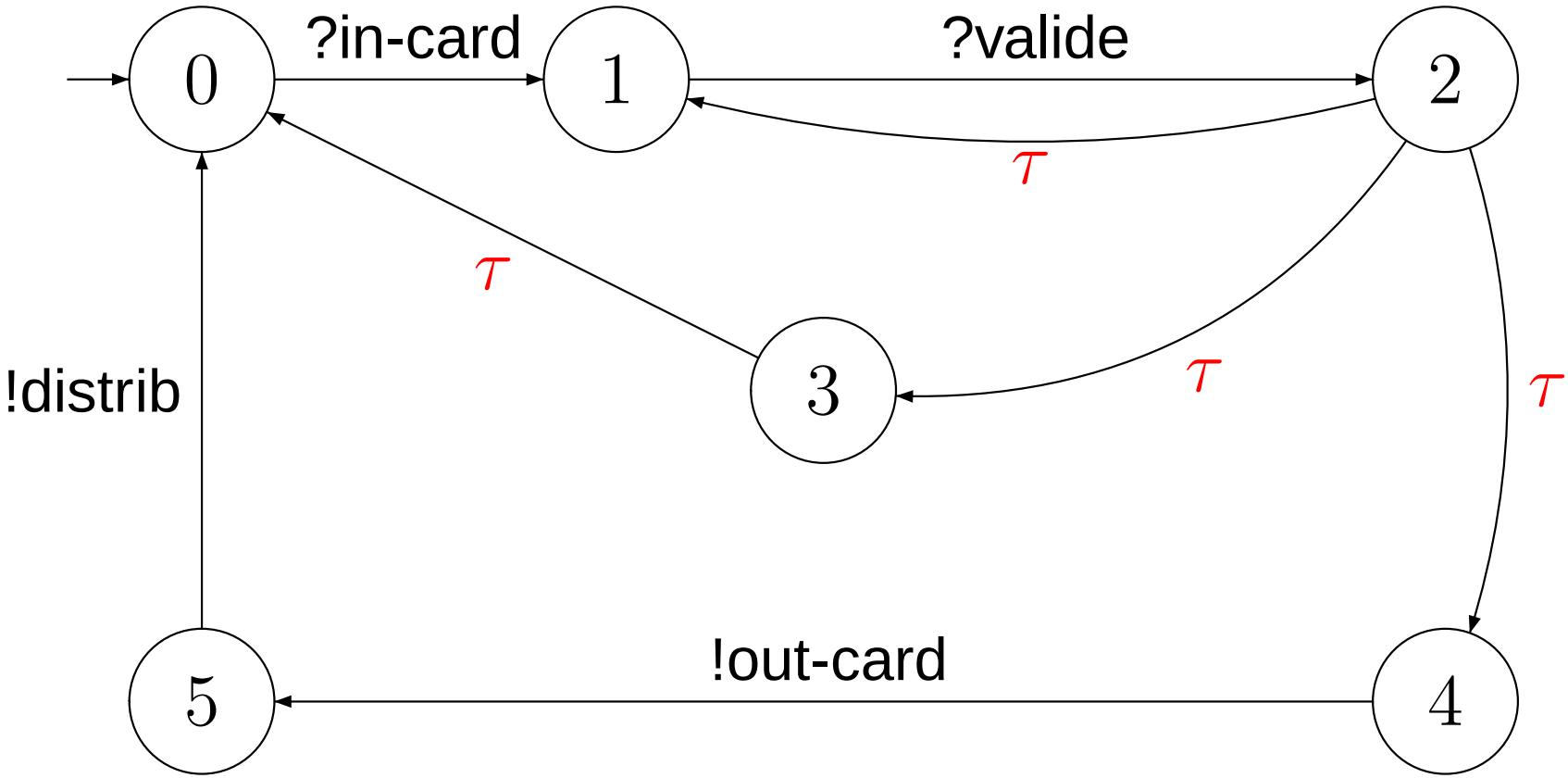


FIG. 1.11 – GAB non déterministe

1.5– Equité (Fairness) [4, 8, 10]

– S un SdeT : toute exécution (infinie) de $S \Longrightarrow$ une *transition* de S est *infiniment souvent tirée*
– $S = S_1 \parallel S_2$: exécution infinie de $S \not\Longrightarrow$ infiniment souvent une transition de S_1 (S_2) soit tirée

équité forte : toute transition *infiniment souvent tirable* est *infiniment souvent tirée*

équité faible : toute transition *toujours tirable à partir d'un certain moment* est *infiniment souvent tirée*

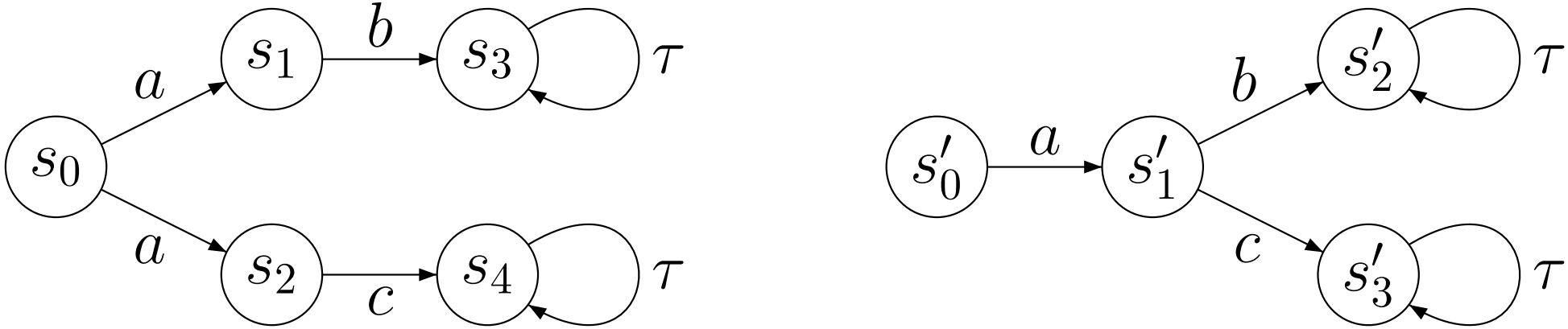
Exo : Définir formellement les critères d'équité. Relation équités forte et faible.

1.6– Equivalences de systèmes de transitions [4]

- but : définir des *équivalences* pour 2 SdeTs S et S'
- critères :
  - mêmes *traces* ($L(S) = L(S')$)
  - mêmes *structures*
- *formalisation* de ces critères
- *propriétés* et *relations* entre les critères
  exemple : *équivalence de traces* et *bisimulation*

1.6.1– Equivalence de traces

- $S = (Q, s_0, A, \rightarrow)$ et $S' = (Q', s'_0, A, \rightarrow')$ des SdeTs
- $L(S, q) = \{$traces des chemins commençants en q dans $S\}$
- $q \in S, q' \in S',\ q \overset{t}{\approx} q' \iff L(S, q) = L(S', q')$
- $S \overset{t}{\approx} S' \iff s_0 \overset{t}{\approx} s'_0$

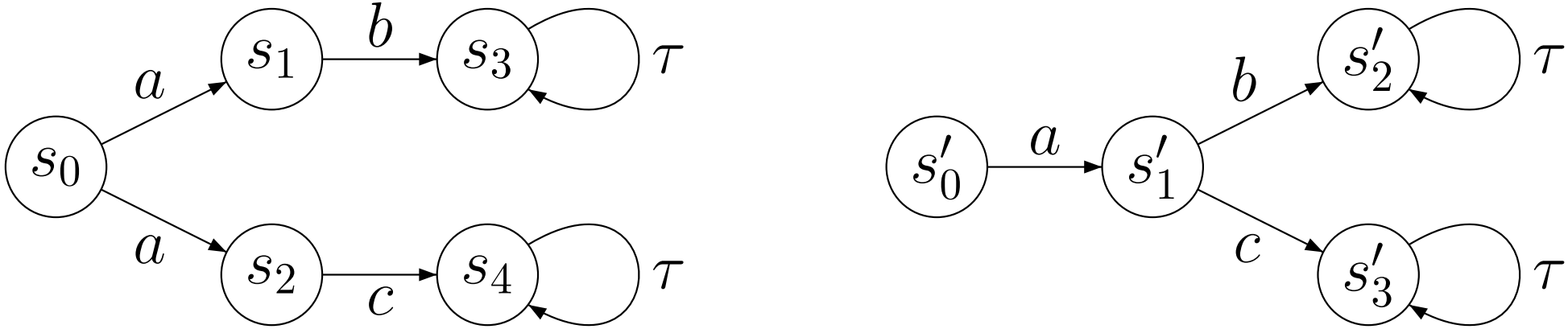


(a) Automate S (b) Automate S'

FIG. 1.12 – Automates traces-équivalents

1.6.2– Equivalence de traces

- $S = (Q, s_0, A, \rightarrow)$ et $S' = (Q', s'_0, A, \rightarrow')$ des SdeTs
- $L(S, q) = \{$traces des chemins commençants en q dans $S\}$
- $q \in S, q' \in S',\ q \overset{t}{\approx} q' \iff L(S, q) = L(S', q')$
- $S \overset{t}{\approx} S' \iff s_0 \overset{t}{\approx} s'_0$



(a) Automate S (b) Automate S'

FIG. 1.13 – Automates traces-équivalents

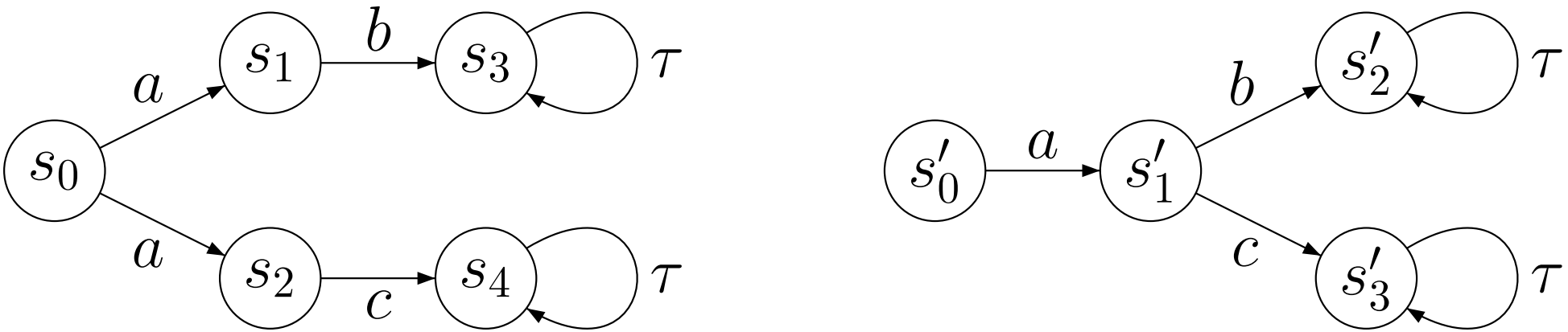
Extensions : traces acceptantes, refusantes, . . . autres ?

Bisimulation

- $S = (Q, s_0, A, \rightarrow)$ et $S' = (Q', s'_0, A, \rightarrow')$ des SdeTs
- S (resp. S') peut *mimer* chaque transition de S' (resp. S)
- vu de *l'extérieur* S et S' sont *indiscernables*

Bisimulation

- $S = (Q, s_0, A, \rightarrow)$ et $S' = (Q', s'_0, A, \rightarrow')$ des SdeTs
- S (resp. S') peut *mimer* chaque transition de S' (resp. S)
- vu de *l'extérieur* S et S' sont *indiscernables*



(a) Automate S (b) Automate S'

FIG. 1.15 – Automates non bisimilaires

Définition 5 (Relation de bisimulation) $S = (Q, s_0, A, \rightarrow)$, $S' = (Q', s'_0, A', \rightarrow)$ *deux SdeTs,* $\mathcal{R} \subseteq S \times S'$ *est une relation de bisimulation entre* S *et* S' *ssi* $\mathcal{R}$ *est totale et* $\forall s \in S, s' \in S',\ s\mathcal{R}s'$

$$\forall (s \xrightarrow{a} s_1) \Longrightarrow \exists (s' \xrightarrow{a} s'_1) \wedge s_1 \mathcal{R} s'_1 \tag{1.1}$$

$$\forall (s' \xrightarrow{a} s'_1) \Longrightarrow \exists (s \xrightarrow{a} s_1) \wedge s_1 \mathcal{R} s'_1 \tag{1.2}$$

S et S' sont en bisimulation ssi il existe une relation de bisimulation $\mathcal{R}$ entre S et S' avec $s_0 \mathcal{R} s'_0$. Si uniquement 1.1 alors $\mathcal{R}$ est une relation de simulation entre S' et S et S' simule S. □

Exo : S' simule S et S simule $S' \Longrightarrow S$ et S' sont bisimilaires ?
Relation entre équivalence de traces et bisimulation ?

Chapitre 2 : Les Logiques Temporelles – temps discret

Sommaire

Logiques temporelles ?

– langages de *spécification* de *propriétés* d'un système réactif
– but : spécifier des *comportements dynamiques*
formules *non statiques* ; la valeur de vérité *change* dans le temps
exemple : toute demande d'argent sera satisfaite – on ne reçoit de l'argent que si on a entré le bon code – l'imprimante 1 imprime – la porte de l'ascenseur ne peut s'ouvrir que si le la cabine est arrêtée, . . .

Logiques adaptées : logiques temporelles [8, 10, 15, 5, 26, 20]

– logique temporelle linéaire (LTL) : propriétés des exécutions
– logique temporelle arborescente (CTL 2.2, CTL* 2.3) : propriétés des arbres d'exécution
– μ-calcul **??** : calcul de points fixes

Types de propriétés

– Propriété $=$ ensemble d'exécutions
– Temps $=$ *discret* ; instants $=$ états successifs des *exécutions* (1)

sûreté (safety) : "quelque chose de mauvais n'arrive jamais"
P propriété de sûreté ssi :

$$\sigma \in P \iff \textit{tous les préfixes finis de } \sigma \in P$$

Types de propriétés

– Propriété $=$ ensemble d'exécutions
– Temps $=$ *discret* ; instants $=$ états successifs des *exécutions* (1)

sûreté (safety) : "quelque chose de mauvais n'arrive jamais"
P propriété de sûreté ssi :

$$\sigma \in P \iff \textit{tous les préfixes finis de } \sigma \in P$$

vivacité (liveness) : "quelque chose de bon est toujours possible"
P propriété de vivacité ssi :

$$\textit{toute exécution finie } \sigma \textit{ peut être étendue en } \sigma' \in P$$

Types de propriétés

– Propriété = ensemble d'exécutions
– Temps = *discret* ; instants = états successifs des *exécutions* (1)

sûreté (safety) : "quelque chose de mauvais n'arrive jamais"
P propriété de sûreté ssi :

$$\sigma \in P \iff \textit{tous les préfixes finis de } \sigma \in P$$

vivacité (liveness) : "quelque chose de bon est toujours possible"
P propriété de vivacité ssi :

$$\textit{toute exécution finie } \sigma \textit{ peut être étendue en } \sigma' \in P$$

Théorème [1] : Toute propriété est la conjonction d'une propriété de sûreté et d'une propriété de vivacité.

Interprétation des formules de logiques temporelles

– *modèle* : $S = (Q, s_0, A, \rightarrow, L)$ SdeT étiqueté 1
ensemble de *propriétés atomiques* sur les états
$AP = \{L(q), q \in Q\} \cup \{q, q \in Q\} \cup \{\text{tt}, \text{ff}\}$

– *interprétation* (évaluation) des formules ϕ sur
une **exécution σ du SdeT en LTL**

$$\sigma \models \phi$$

Interprétation des formules de logiques temporelles

– *modèle* : $S = (Q, s_0, A, \rightarrow, L)$ SdeT étiqueté 1
ensemble de *propriétés atomiques* sur les états
$AP = \{L(q), q \in Q\} \cup \{q, q \in Q\} \cup \{\text{tt}, \text{ff}\}$

– *interprétation* (évaluation) des formules ϕ sur
une **exécution σ du SdeT en LTL**

$$\forall \sigma \in Exec(S), \quad \sigma \models \phi \Longleftrightarrow S \models \phi$$

Interprétation des formules de logiques temporelles

– *modèle* : $S = (Q, s_0, A, \rightarrow, L)$ SdeT étiqueté 1
ensemble de *propriétés atomiques* sur les états
$AP = \{L(q), q \in Q\} \cup \{q, q \in Q\} \cup \{\text{tt}, \text{ff}\}$

– *interprétation* (évaluation) des formules ϕ sur
une **exécution σ du SdeT en LTL**

$$\forall \sigma \in Exec(S), \quad \sigma \models \phi \Longleftrightarrow S \models \phi$$

arbre **d'exécution $A(S)$ du SdeT en CTL**

$$A(S) \models \phi$$

Interprétation des formules de logiques temporelles

– *modèle* : $S = (Q, s_0, A, \rightarrow, L)$ SdeT étiqueté 1
ensemble de *propriétés atomiques* sur les états
$AP = \{L(q), q \in Q\} \cup \{q, q \in Q\} \cup \{\text{tt}, \text{ff}\}$

– *interprétation* (évaluation) des formules ϕ sur
une **exécution σ du SdeT en LTL**

$$\forall \sigma \in Exec(S), \quad \sigma \models \phi \iff S \models \phi$$

arbre **d'exécution $A(S)$ du SdeT en CTL**

$$A(S) \models \phi \iff S \models \phi$$

2.1– Logique temporelle linéaire (LTL) [10, 5, 26]

2.1.1– Syntaxe de LTL

objet de l'étude : *séquences d'états* infinies – (déterminisme)

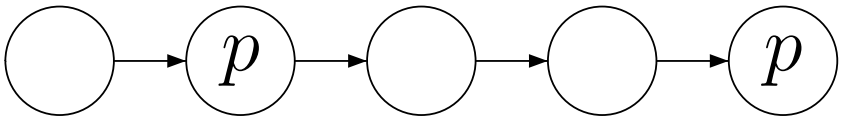
Définition 6 (Formules de LTL) *Les formules de LTL sont définies inductivement par :*

– $\forall p \in AP, p \in LTL$

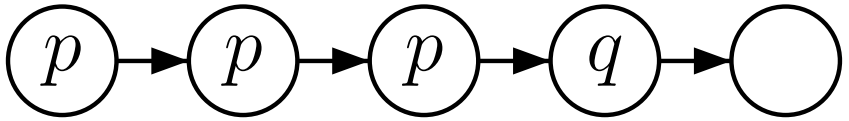
– $p, q \in LTL$, alors $p \vee q, \neg p \in LTL$,

– $p, q \in LTL$, alors $\mathsf{X} p, p \,\mathcal{U}\, q \in LTL$

Sémantique intuitive :



X p X p



$p \,\mathcal{U}\, q$

2.1.2– Sémantique de LTL

- *séquence* $\sigma = s_0 s_1 \dots s_n \dots$
- $\sigma_n = s_n \dots$
- $\forall k \geq 0, L(s_k) \subseteq AP$ (propositions atomiques vraies en s_k)
  et $s_i \in L(s_j) \iff i = j$ et tt $\in L(s_i)$, ff $\notin L(s_i)$

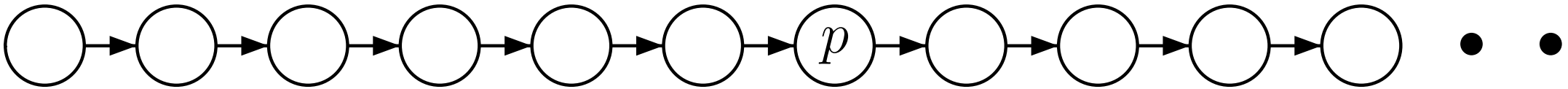
Définition 7 (Sémantique de LTL) *σ une séquence :*

- $p \in AP, \sigma \models p \iff p \in L(s_0)$
- $\sigma \models p \vee q \iff \sigma \models p$ *ou* $\sigma \models q$
- $\sigma \models \neg p \iff \sigma \not\models p$,
- $\sigma \models \mathsf{X} p \iff \sigma_1 \models p$
- $\sigma \models p \mathcal{U} q \iff \exists j, \sigma_j \models q$ *et* $(\forall k < j, \sigma_k \models p)$

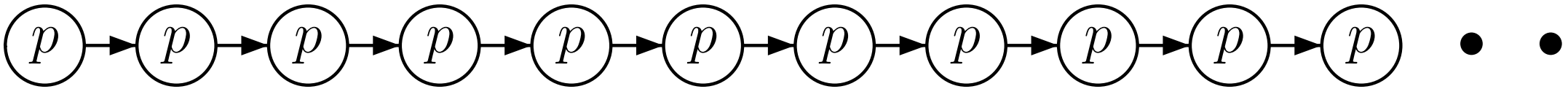
□

2.1.3– Abbréviations utiles

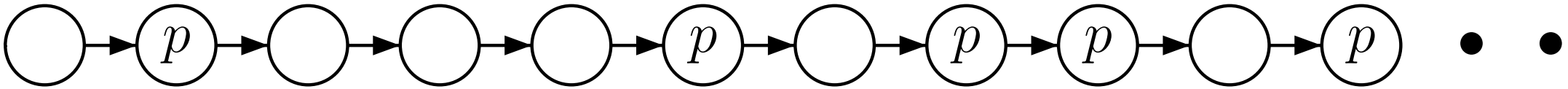
– **fatalement p : F$p \equiv tt\,\mathcal{U}\,p$**



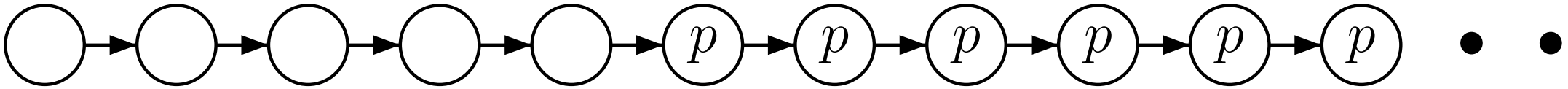
– **toujours p : G$p \equiv \neg$F$\neg p$** (p est un *invariant*)



– **infiniment souvent p : GFp**



– **presque partout p : FGp**



Exemple : Propriétés de $GAB \times U$ (1.8)

– sûreté : jamais carte rendue et code mauvais $\mathbf{G}\neg(U.3 \wedge n > 3)$
– réponse : obtenir de l'argent . . . $\mathbf{G}(U.2 \Longrightarrow \mathbf{F}U.3)$

Exo : Proposer des formules LTL (si possible ?) pour les propriétés :

1. si l'utilisateur obtient de l'argent, $n \leq 3$
2. après une demande l'utilisateur n'obtient de l'argent que si n est resté ≤ 3 depuis sa demande
3. si le GAB revient infiniment souvent dans son état initial, l'utilisateur obtient infiniment souvent de l'argent
4. si l'utilisateur a de l'argent, avant il y a forcément eu une demande
5. il est toujours possible d'obtenir de l'argent
6. toutes les 4 unités de temps on est dans l'état 0

Quelles sont les propriétés vraies sur le SdeT de la Fig. 1.8 ?

2.1.4– Equité 1.5 en LTL

$S = (Q, s_0, A, \rightarrow)$ un SdeT avec propriétés sur les *transitions* et les *états* : $L : Q \times A \times Q \rightarrow P$; $L((s, a, s')) \in P$

séquence $= (\ \textit{état}.\ \textit{transitions})^{\omega}$

$L(s) = \{\text{enabled}(t), t = (s, a, s') \in \rightarrow\}$ $\quad$ $L(t) = \{\text{Exec}(t), t \in \rightarrow\}$

équité faible : **FG** enabled$(t) \Longrightarrow$ **GF** Exec(t)

différence avec **FG** enabled$(t) \Longrightarrow$ **F** Exec(t) ?

équité forte : **FG** enabled$(t) \Longrightarrow$ **FG** Exec(t)

Exo :

1. exprimer le *blocage* dans un état s en LTL ;
2. exprimer les critères d'équités forte et faible en LTL pour des SdeTs avec
   - *propriétés* sur les *états* uniquement ;
   - *propriétés* sur les *transitions* uniquement.

2.1.5– Quelques équivalences de formules

$$p \Longrightarrow \mathbf{F}p \qquad \mathbf{X}p \Longrightarrow \mathbf{F}p \qquad \mathbf{G}p \Longrightarrow \mathbf{F}p$$

$$p\mathcal{U}q \Longrightarrow \mathbf{F}q \qquad \mathbf{FG}p \Longrightarrow \mathbf{GF}p \qquad \mathbf{FF}p \equiv \mathbf{F}p$$

$$\mathbf{F}p \equiv p \vee \mathbf{XF}p \qquad \mathbf{G}p \equiv p \wedge \mathbf{XG}p$$

$$p\mathcal{U}q \equiv q \vee (p \wedge \mathbf{X}(p\mathcal{U}q))$$

2.2– Logique temporelle arborescente : CTL [8, 15, 10]

objet de l'étude : *arbres d'exécutions* (infinis) – (indéterminisme)

2.2.1– Syntaxe de CTL

Définition 8 (Formules de Computation Tree Logic) *Les formules de CTL sont les formules d'états définies inductivement par :*

– $\forall p \in AP, p \in$ *formules d'état*

– $p, q \in$ *formules d'état CTL, alors* $p \vee q, \neg p \in$ *formules d'état,*

– $p \in$ *formules de chemin de CTL, alors* $\mathbf{E}p, \mathbf{A}p \in$ *formules d'états,*

– $p, q \in$ *formules d'états de CTL, alors* $\mathsf{X}p, p\,\mathcal{U}\,q \in$ *formules de chemin*

2.2.2– Sémantique de CTL

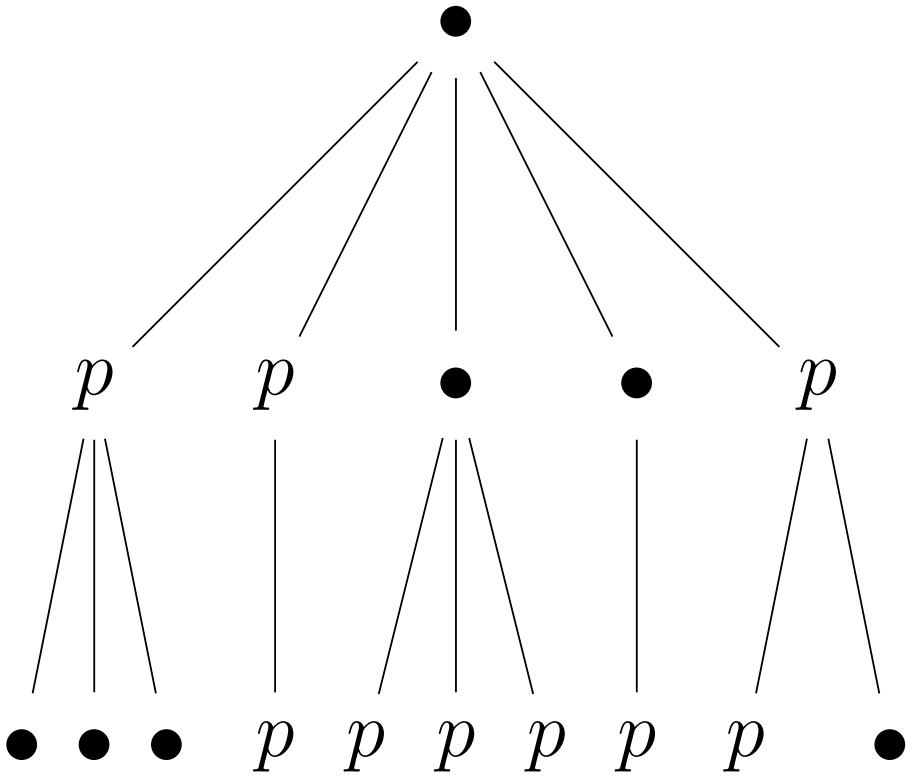
- arbre d'exécution $\equiv$ *relation binaire* R (totale)
- chemin $\sigma = s_0 s_1 \dots s_n \iff \forall i, (s_i, s_{i+1}) \in R \qquad \sigma_i = s_i \dots$
- $\forall k \geq 0, L(s_k) \subseteq AP$ (propositions atomiques vraies en s_k)

Définition 9 (Sémantique de CTL) *σ une séquence :*

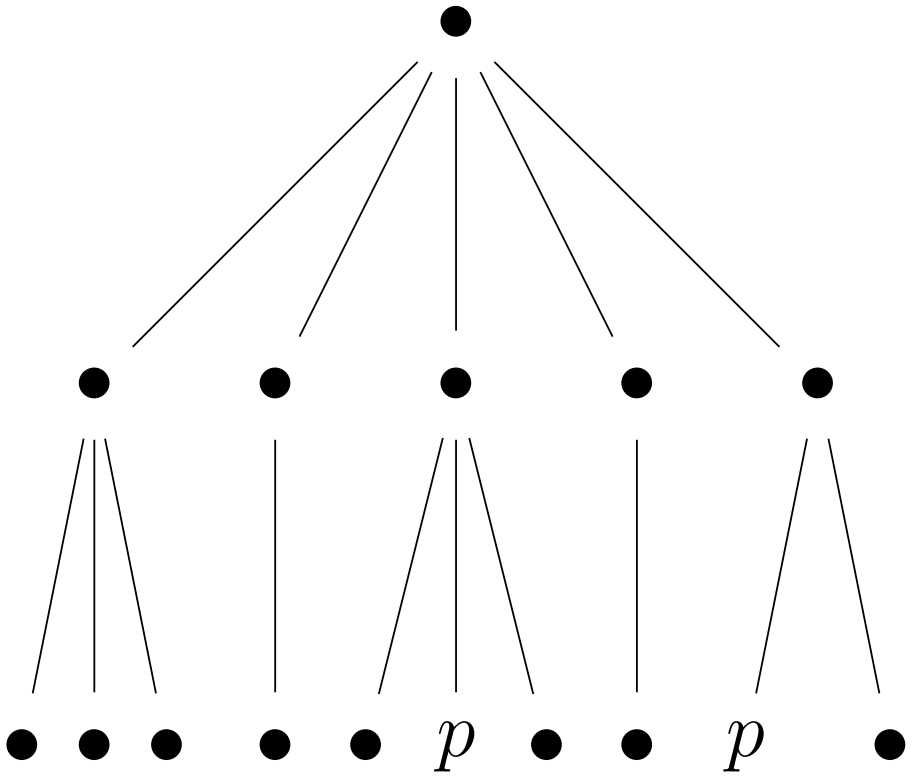
- $p \in AP, s_0 \models p \iff p \in L(s_0)$
- $s_0 \models p \vee q \iff s_0 \models p$ *ou* $s_0 \models q \qquad s_0 \models \neg p \iff s_0 \not\models p$,
- $s_0 \models \mathbf{E}p \iff \exists \sigma = s_0 \dots, \sigma \models p$ *(p formule de chemins)*
- $s_0 \models \mathbf{A}p \iff \forall \sigma = s_0 \dots, \sigma \models p$ *(p formule de chemins)*
- $\sigma \models \mathsf{X}p \iff \sigma_1 \models p$ *(p formule d'état)*
- $\sigma \models p \,\mathcal{U}\, q \iff \exists j, \sigma_j \models q$ *et* $(\forall k < j, \sigma_k \models p)$ *(p et q formules d'états)*

2.2.3– Abbréviations utiles

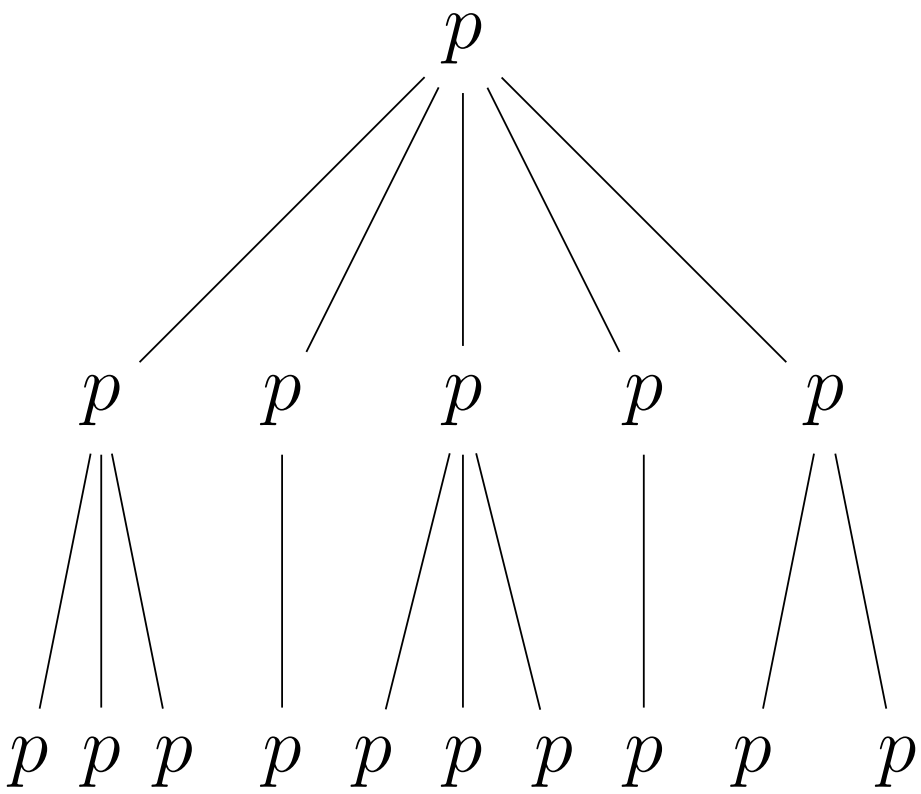
$\mathbf{AF}p \equiv \mathbf{A}(\mathrm{tt}\,\mathcal{U}\,p)$



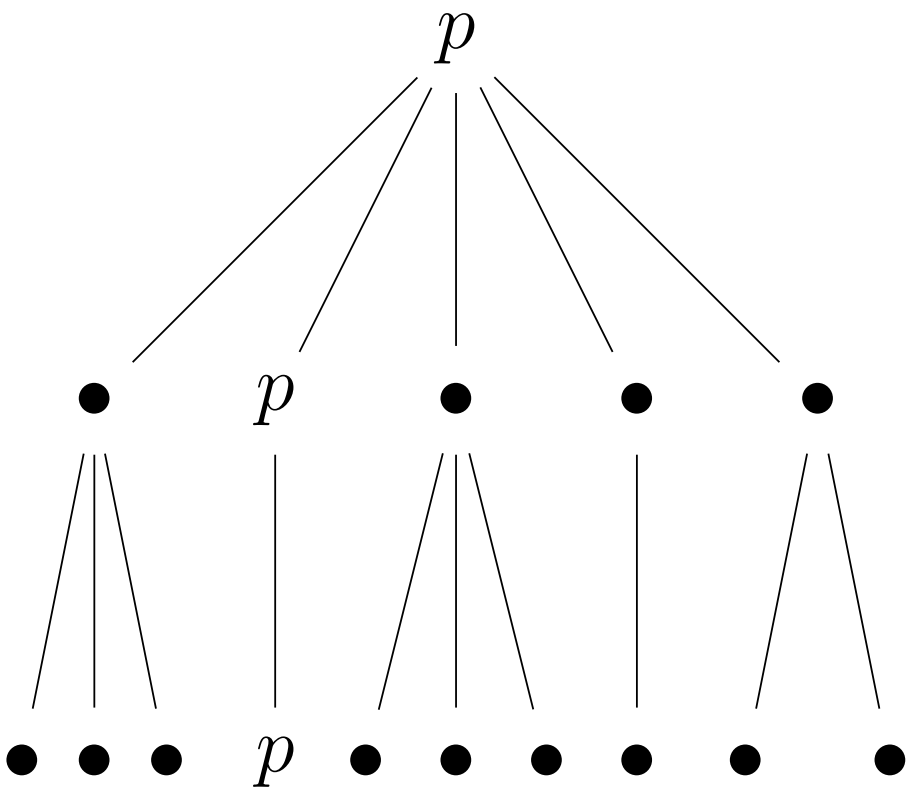
$\mathbf{EF}p \equiv \mathbf{E}(\mathrm{tt}\,\mathcal{U}\,p)$



AG$p \equiv \neg$**EF**$\neg p$



EG$p \equiv \neg$**AF**$\neg p$



Exemples de propriétés de $GAB \times U$ (1.8)

- sûreté : jamais carte rendue et code mauvais $\mathbf{AG}\neg(U.3 \wedge n > 3)$
- il est possible d'obtenir de l'argent après chaque demande $\mathbf{EG}(U.2 \Longrightarrow \mathbf{AF}U.3)$
- on obtient toujours de l'argent $\mathbf{AG}(U.2 \Longrightarrow \mathbf{AF}U.3)$
- de tout état, on peut revenir à l'état initial $\mathbf{AG}\,(\mathbf{EF}\mathit{init})$

Exo : Proposer des formules CTL (si possible ?) pour les propriétés :

1. si l'utilisateur obtient de l'argent, $n \leq 3$,
2. après une demande il est possible que l'utilisateur obtienne de l'argent
3. après une demande l'utilisateur n'obtient de l'argent que si n est resté ≤ 3 depuis sa demande
4. si le GAB revient infiniment souvent dans son état initial, l'utilisateur obtient infiniment souvent de l'argent
5. il est toujours possible d'obtenir de l'argent

Quelles sont les propriétés vraies sur le SdeT de la Fig. 1.8 ?

2.2.4– Equité en CTL

– en LTL : **FG** et **GF**

impossible en CTL !

– Cf. syntaxe de CTL (8) : **F** et **G** ne peuvent être emboîtés
– infiniment souvent p sur tous les chemins : **AGAF**p
– il existe un chemin avec infiniment souvent p :
EGEFp ? **EGAF**p ?
– il existe un chemin avec infiniment souvent p_1 et p_2 . . .
– extension de CTL : Fair CTL $=$ CTL avec une sémantique *fair*
définir les chemins équitables

2.2.5– Quelques équivalences de formules

$$\mathbf{AG}p \equiv p \wedge \mathbf{AXAG}p \qquad \mathbf{EG}p \equiv p \wedge \mathbf{EXEG}p$$

$$\mathbf{AF}p \equiv p \vee \mathbf{AXAF}p \qquad \mathbf{EF}p \equiv p \vee \mathbf{EXEF}p$$

$$\mathbf{A}(p\mathcal{U}q) \equiv q \vee (p \wedge \mathbf{AXA}(p\mathcal{U}q)) \qquad \mathbf{E}(p\mathcal{U}q) \equiv q \vee (p \wedge \mathbf{EXE}(p\mathcal{U}q))$$

2.3– LTL + CTL $\subseteq$ CTL*[8, 10]

2.3.1– Syntaxe de CTL*

Définition 10 (Formules de CTL*) *Les formules de CTL* sont définies inductivement par :*

(s_1) $\forall p \in AP, p \in$ *formules d'état*

(s_2) $p, q \in$ *formules d'état, alors* $p \vee q, \neg p \in$ *formules d'état,*

(s_3) $p \in$ *formules de chemin, alors* **E**p, **A**$p \in$ *formules d'état,*

(p_1) $p \in$ *formules d'état, alors* $p \in$ *formules de chemin,*

(p_2) $p, q \in$ *formules de chemin, alors* $p \vee q, \neg p \in$ *formules de chemin*

(p_3) $p, q \in$ *formules de chemin, alors* **X**$p, p \mathcal{U} q \in$ *formules de chemin*

□

2.3.2– Sémantique de CTL*

Définition 11 (Sémantique de CTL*) *La sémantique des formules de CTL* est définie inductivement par :*

(s_1) $p \in AP$, *formule d'état,* $s_0 \models p \iff p \in L(s_0)$

(s_2) $s_0 \models p \vee q \iff s_0 \models p$ *ou* $s_0 \models q$ $\qquad s_0 \models \neg p \iff s_0 \not\models p$,

(s_3) $s_0 \models \mathbf{E}p \iff \exists \sigma = s_0 \ldots, \sigma \models p$ (p *formule de chemins*)

(s'_3) $s_0 \models \mathbf{A}p \iff \forall \sigma = s_0 \ldots, \sigma \models p$ (p *formule de chemins*)

(p_1) $p \in$ *formules d'état,* $s_0 \models p \iff \sigma = s_0 \ldots \models p$

(p_2) $\sigma \models p \vee q \iff \sigma \models p$ *ou* $\sigma \models q$

(p_3) $\sigma \models \mathsf{X}p \iff \sigma_1 \models p$ (p *for. de chemin*)

(p'_3) $\sigma \models p \,\mathcal{U}\, q \iff \exists j, \sigma_j \models q$ *et* $(\forall k < j, \sigma_k \models p)$ (p *et* q *for. de chemin*) □

LTL, CTL, CTL* ?

	nature du temps	équité	outils
LTL	linéaire	oui	SPIN [13, 14]
CTL	arborescent	non	SMV [21]
CTL*	linéaire et arborescent	oui	

Autres logiques

– TLA (Lamport) [16]
– μ-calcul [4] (MEC [3])

Deuxième partie : Algorithmes de model-checking

Vérification sur modèle : Model-checking [8, 7, 22]

- S un SdeT étiqueté
- ϕ une formules de LTL, CTL ou CTL*
- question : S est-il un *modèle* de ϕ.

$$\boxed{S \models \phi}$$

- réponse *oui* ou *non* : *algorithme de model-checking*
  si oui ! ! si *non* $\Longrightarrow$ *contre exemple*

Chapitre 3 : Model-checking de LTL

Sommaire

Figures

3.1– Automates de Büchi [25]

Définition 12 (Automate de Büchi) *Un automate de Büchi A c'est :*

- Σ *alphabet fini,*
- Q *ensemble fini d'états*
- $\longrightarrow \subseteq Q \times \Sigma \times Q$ *relation de transition,*
- s_0 *: état initial,*
- F *ensemble d'états accepteurs*

exécution $=$ *séquence infinie de transitions (de* $\longrightarrow$*)*

exécution admissible $=$ *exécution passant infiniment souvent par un état de* F

w **accepté par** $A \equiv w$ *est la trace d'une exécution admissible de* A

langage accepté par A, $L(A) = \{w,$ *acceptés par* $A\}$

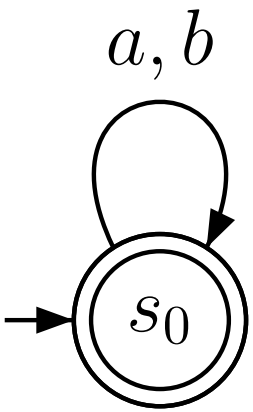


FIG. 3.1 – Automate acceptant $(a \cup b)^\omega$

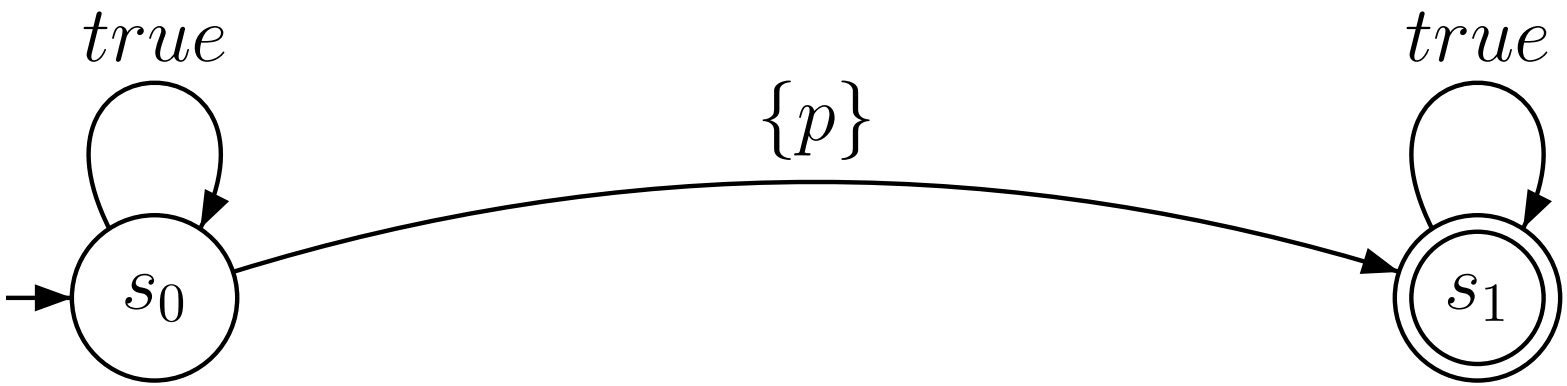


FIG. 3.2 – Automate acceptant $\Diamond p$

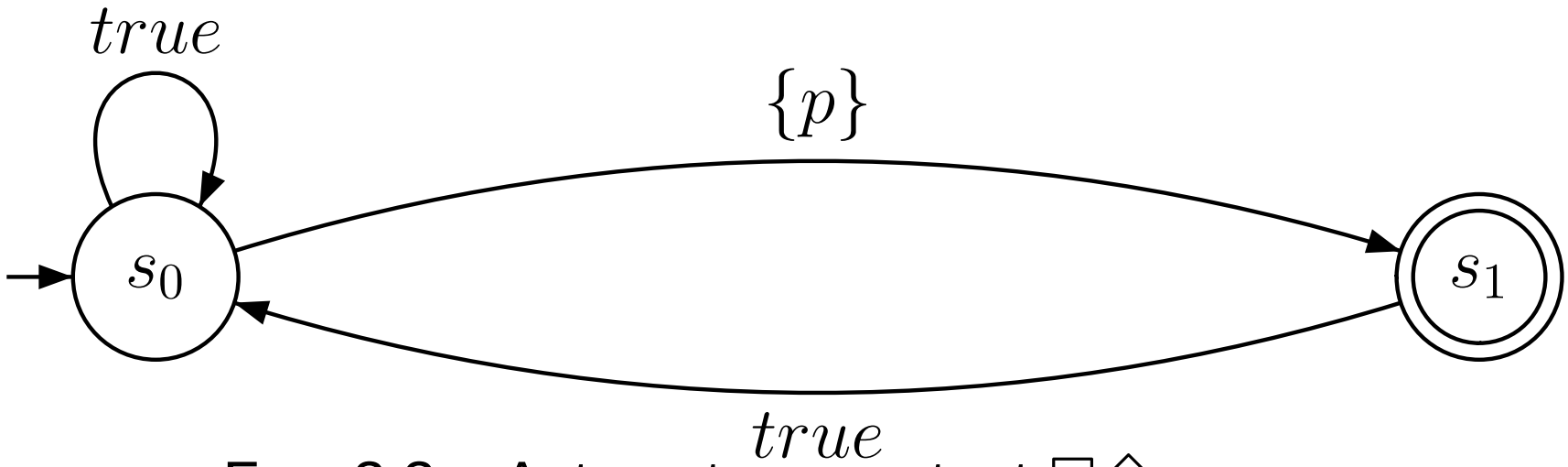


FIG. 3.3 – Automate acceptant $\Box\Diamond p$

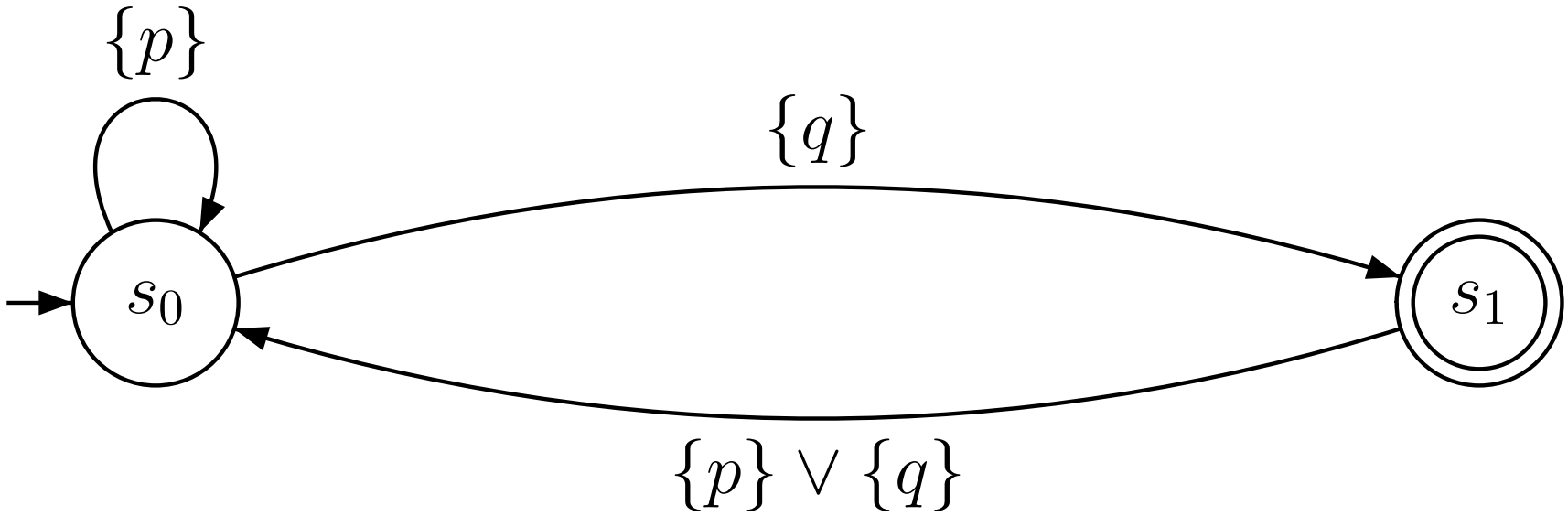


FIG. 3.4 – Automate acceptant $\Box(p \,\mathcal{U}\, q)$

Définition 13 (Automate de Büchi Généralisé) *Un automate de Büchi généralisé A c'est un automate de Büchi où :*

– I *est l'ensemble des états initiaux,*

– $F = \{F_1, \ldots, F_n\}$ *est une famille d'ensembles d'états accepteurs*

exécution admissible $=$ *exécution issue d'un état de* I *et passant infiniment souvent par un état de chaque* F_i

w **accepté par** $A \equiv w$ *est la trace d'une exécution admissible de* A

langage accepté par A, $L(A) = \{w,$ *acceptés par* $A\}$

Théorème 1 (Expressivité des Büchi généralisés) *Tout langage accepté par un automate de Büchi généralisé est accepté par un automate Büchi.* □

3.2– Principe du model-checking pour LTL [8, 7, 22]

- ϕ une *propriété* $\longrightarrow$ $B_{\neg\phi}$ l'automate de Büchi acceptant les *exécutions satisfaisant* $\neg\phi$
- S un système de transitions $\longrightarrow$ S est un automate de Büchi où *tous les états* sont *accepteurs*
- synchronisation de $S \times B_{\neg\phi}$:

$$(q, s) \xrightarrow{L(q)} (q', s')$$

- (q, s) accepteur $\Longleftrightarrow$ s est accepteur dans $B_{\neg\phi}$
- Algorithme de model-cheking : $L(S \times B_{\neg\phi}) = \emptyset$?
  1. *chercher cycle* sur un *état accepteur*
  2. *chemin* d'un *état initial* à cet *état accepteur*

3.3– Complexité du model-cheking de LTL

- S un SdeT : $|S| = |Q| + |T|$
- $|\phi| =$ nombre de sous formules de ϕ
- $|B_{\neg\phi}|$ est en $\mathcal{O}(2^{|\phi|})$
- $|S \times B_{\neg\phi}|$ est en $\mathcal{O}(|S|.|B_{\neg\phi}|)$
- $L(S \times B_{\neg\phi}) = \emptyset$ est en $\mathcal{O}(|S \times B_{\neg\phi}|)$

Le *model-checking de LTL* est en $\mathcal{O}(|S|.2^{|\phi|})$

3.4– Model-checking de LTL ”à la volée”(on-the-fly)

– but : éviter de construire $S \times B_{\neg\phi}$ complétement
– algorithme :
  1. générer les états de $S \times B_{\neg\phi}$ en DFS jusqu’à trouver un état accepteur
  2. chercher un cycle à partir de cet état accepteur
– en mémoire uniquement le chemin courant
– si $L(S \times B_{\neg\phi}) = \emptyset$ on ne gagne rien . . .
– sinon : algorithme utilisable sur des SdeT infinis

3.5– Construction de B_ϕ

- principe : q un état de $B_\phi \longleftrightarrow q \subseteq \{$ sous formules de $\phi \}$
  toute *exécution admissible* à partir de q dans B_ϕ *satisfait* les formules de q
- états initiaux de B_ϕ : q tel que $\phi \in q$
- $Cl(\phi) =$ ensemble des sous-formules de ϕ et leur négation ($\neg\neg\varphi = \varphi$)
- état q de B_ϕ est un *ensemble maximal* vérifiant
  - $\varphi \in q$ ou $\neg\varphi \in q$ (mais *pas les deux* !)
  - $\varphi \vee \psi \in q$ ssi $\varphi \in q$ ou $\psi \in q$
  - si $\varphi \mathcal{U} \psi \in q$ alors $\varphi \in q$ ou $\psi \in q$
  - si $\varphi \mathcal{U} \psi \notin q$ alors $\psi \notin q$

Exo : construire $Cl(\mathbf{F}p)$, p proposition atomique.

Construction de B_ϕ (suite)

- *étiquettes* des transitions $=$ *propositions atomiques*
- *relation de transition* : $q \xrightarrow{a} q'$ ssi
  - $a =$ propositions atomiques de q
  - si $\mathbf{X}\phi \in q$ (resp. $\notin q$) alors $\phi \in q'$ (resp. $\notin q'$)
  - si $\phi \mathcal{U} \psi \in q$ et $\psi \notin q$ alors $\phi \mathcal{U} \psi \in q'$
  - si $\phi \mathcal{U} \psi \notin q$ et $\phi \in q$ alors $\phi \mathcal{U} \psi \notin q'$
- *états accepteurs* :
  - u_i l'ensemble des sous formules de la forme $p_i \mathcal{U} q_i$ de $Cl(\phi)$
  - $\mathbf{GF} u_i \Longrightarrow \mathbf{GF} q_i$
  - $\mathbf{FG} \neg u_i$ OK

  la famille des états accepteurs est $F_i = \{q, \neg u_i \in q\} \cup \{q, q_i \in q\}$

Exemple : $\phi = \mathsf{F}p \equiv \mathsf{tt}\mathcal{U}p$

- $Cl(\phi) = \{p, \neg p, \mathsf{tt}\mathcal{U}p, \neg(\mathsf{tt}\mathcal{U}p)\}$
- états *cohérents* :

	cohérent ?	**initial ?**	**numéro**
$p, \mathsf{tt}\mathcal{U}p$	oui	oui	s_1
$p, \neg(\mathsf{tt}\mathcal{U}p)$	non	–	–
$\neg p, \mathsf{tt}\mathcal{U}p$	oui	oui	s_2
$\neg p, \neg(\mathsf{tt}\mathcal{U}p)$	oui	non	s_3

- famille d'*états accepteurs* : $F = \{s_1, s_3\}$

Exemple : $\phi = \mathbf{F}p \equiv \mathrm{tt}\mathcal{U}p$ (suite)

– *relation de transition* :

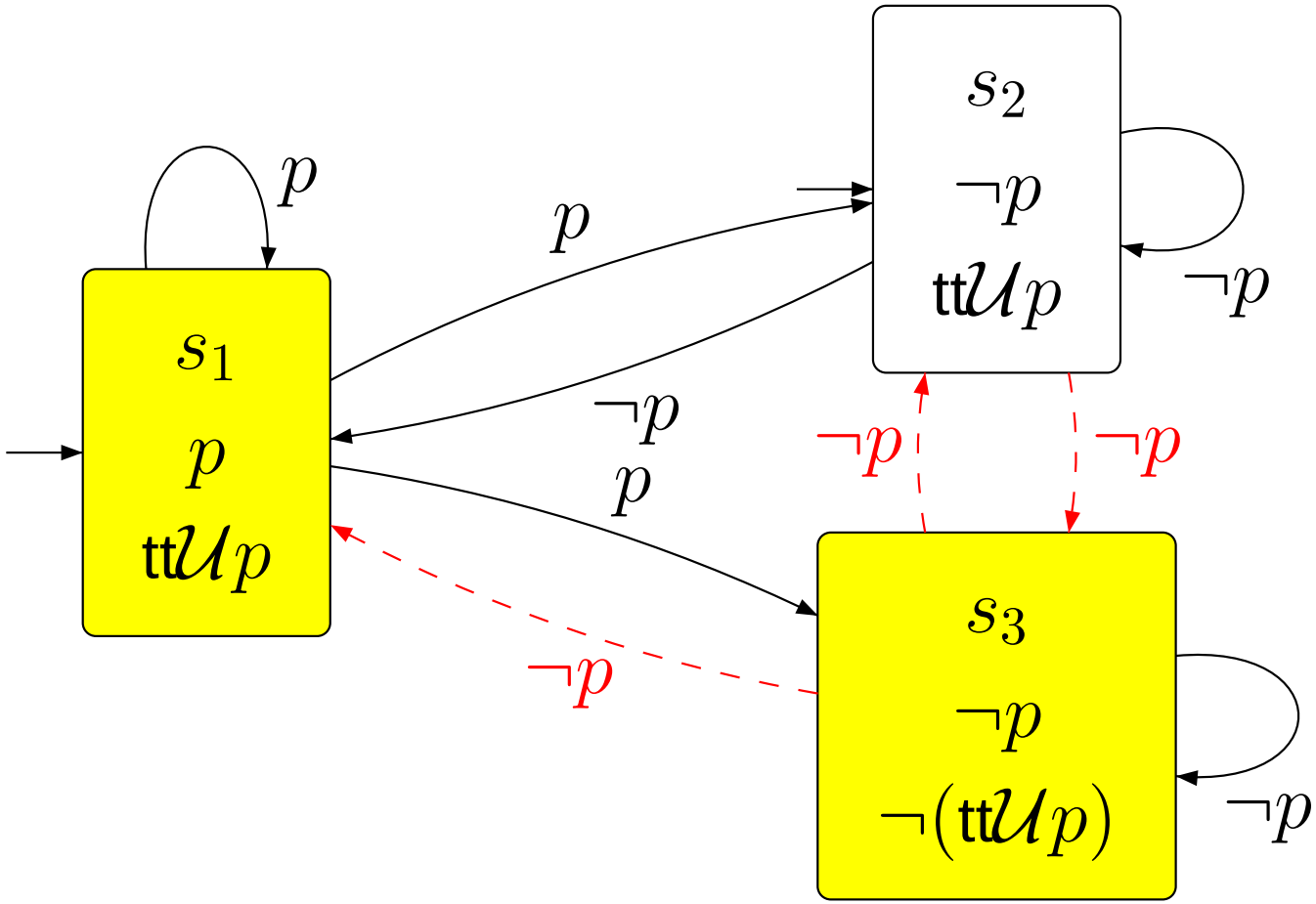


FIG. 3.5 – Automate de Büchi acceptant $\mathbf{F}p$

Chapitre 4 : Model-checking de CTL

Sommaire

4.1– Principe du Model-checking pour CTL [8, 7, 15]

Principe : étiquetage des états $s \in S$ par les sous-formules vraies en s

- $p \in AP$, $p \in L(s)$ ou $p \notin L(s)$
- $p \wedge q$, $\neg p$ ajout de $p \wedge q$ à $L(S)$ si $p, q \in L(s)$, de $\neg p$ si $p \notin L(s)$,
- $p =$ **EX**q, si $\exists (s,t) \in R, q \in L(t)$, ajout de q à $L(s)$
- $p =$ **AX**q, si $\forall (s,t) \in R, q \in L(t)$, ajout de q à $L(s)$
- $p =$ **E**$(q \,\mathcal{U}\, r)$: faire $Card(S)$ fois (arrêt !)
  1. si $r \in L(s)$ ajout de **E**$(q \,\mathcal{U}\, r)$ à $L(s)$
  2. $\forall s, q \in L(s)$ *et* $\exists (s,t) \in R,$ **E**$(q \,\mathcal{U}\, r) \in L(t)$ ajout de **E**$(q \,\mathcal{U}\, r)$ à $L(s)$,
- $p =$ **A**$(q \,\mathcal{U}\, r)$: faire $Card(S)$ fois (arrêt !)
  1. si $r \in L(s)$ ajout de **A**$(q \,\mathcal{U}\, r)$ à $L(s)$
  2. $\forall s, q \in L(s)$ *et* $\forall (s,t) \in R,$ **A**$(q \,\mathcal{U}\, r) \in L(t)$ ajout de **A**$(q \,\mathcal{U}\, r)$ à $L(s)$,

Cas des opérateurs A et E

- $\mathbf{A}(q\,\mathcal{U}\,r) = r \vee \mathbf{AXA}(q\,\mathcal{U}\,r)$ ($\mathbf{XA}(q\,\mathcal{U}\,r)$ est une formule de chemin)
- $\mathbf{E}(q\,\mathcal{U}\,r) = r \vee \mathbf{EXE}(q\,\mathcal{U}\,r)$
- réduction à accessibilité **bornée** : $[\mathbf{A}_0(q\,\mathcal{U}\,r)](s) \equiv s \models r$ et

$$s \models [\mathbf{A}_{i+1}(q\,\mathcal{U}\,r)] \equiv s \models r \vee (q \wedge \mathbf{AX}[\mathbf{A}_i(q\,\mathcal{U}\,r)])$$

Propriété : $s \models \mathbf{A}(q\,\mathcal{U}\,r) \iff s \models \mathbf{A}_{Card(S)}(q\,\mathcal{U}\,r)$

$\Longleftarrow$: évident !

$\Longrightarrow$: contraposée

$s \not\models \mathbf{A}_{\leq Card(S)}(q\,\mathcal{U}\,r) \Longrightarrow \exists \sigma = s_0 \ldots$ tel que $\sigma \not\models p$ et $|\sigma| = Card(S)$

$\Longrightarrow \exists s, \sigma(i) = \sigma(j) = s$ (“lemme du gonflement”)

et $\sigma' = \sigma(0..i)\sigma(i..j)^\omega \not\models \mathbf{A}(q\,\mathcal{U}\,r)$

4.2– Complexité du model-cheking de CTL

– pour une (sous) formule φ de S :
maximum $\mathcal{O}(|Q| + |T|)$
– étiquetage pour toutes les sous formules : $\mathcal{O}((|Q| + |T|).|\phi|)$

Le *model-checking de CTL* est en $\mathcal{O}(|S|.|\phi|)$

Chapitre 5 : Model-checking symbolique

Sommaire

Explosion combinatoire

- S un SdeT : n états,m variables booléennes, k variables entières $\in [0..9]$
- nombre d'états possibles : $n.2^m.10^k$
- ex : $n = 10, m = 10, k = 10 \approx 10^5.10^{10}$ états
- espace mémoire : 1 octet/état ! ! $\Longrightarrow$ 10^5 *Go*
  temps : 10^8 transitions/seconde ... $\approx 10^7.10^8$ transitions ... ≈ 115 jours

Problème de l'Explosion Combinatoire

- *produit* synchronisé *d'automates* $A_1, A_2, \cdots A_n$: taille de la *description* $|A_1| + |A_2| + \cdots + |A_n|$
  taille de $A_1 \times A_2 \times \cdots \times A_n$ en $e^{\sum log(|A_i|)}$
- nécessité *regrouper les états*
  $\Longrightarrow$ *représentation symbolique* de l'espace des *états*

Principe du model-checking symbolique

- *représenter* des ensembles d'états de manière *concise*
  regroupement des états en *classes* d'équivalence
- faire des *opérations* (ou *calculs*) sur des *ensembles* d'états
  une opération $\equiv$ exploration de plusieurs transitions *simultanément*
- principe :
  1. écrire les *algos* de model-checking sur des *ensembles d'états*
  2. utiliser un *codage* pour les *ensembles d'états* tel que
  3. *opérations* de l'algo de model-checking *efficaces* sur le codage
- application au model-checking de systèmes infinis 6

5.1– Calcul symbolique d'ensemble d'états

– logique *CTL* ; $S = (Q, s_0, A, \rightarrow, L)$ un SdeT *étiqueté*
– $Sat(\varphi)$: *états* de S *satisfaisant* φ

$$\begin{aligned} \varphi \in AP, Sat(\varphi) &= \{q \in Q, \varphi \in L(s)\} \\ Sat(\neg\varphi) &= Q \setminus Sat(\varphi) \\ Sat(\varphi \vee \psi) &= Sat(\varphi) \cup Sat(\psi) \\ Sat(\mathbf{EX}\varphi) &= Pre(Sat(\varphi)) \end{aligned}$$

– $X \subseteq Q$, $Pre(X) = \{q \in Q, \exists q' \in X, (q, q') \in \rightarrow\}$
(états permettant d'atteindre X en *une* transition)
$Post(X) = \{q' \in Q, \exists q \in X, (q, q') \in \rightarrow\}$
– $Pre^*(X) = \bigcup_{i=0}^{\infty} Pre^i(X)$, avec $Pre^0 = Id$ et $Pre^{i+1} = Pre \circ Pre^i$
(états permettant d'atteindre X en un *nombre fini* de transitions)

Calcul symbolique (suite)

$$Sat(\mathbf{AX}\varphi) \;=\; Q \setminus Pre\big(Q \setminus Sat(\varphi)\big)$$

– opérateurs $\mathbf{E}\varphi_1\mathcal{U}\varphi_2$ et $\mathbf{A}\varphi_1\mathcal{U}\varphi_2$ (Cf. page 50) :

$$\mathbf{E}\varphi_1\mathcal{U}\varphi_2 \;\equiv\; \varphi_2 \vee \big(\varphi_1 \wedge \mathbf{EX}\big(\mathbf{E}\varphi_1\mathcal{U}\varphi_2\big)\big)$$
$$\mathbf{A}\varphi_1\mathcal{U}\varphi_2 \;\equiv\; \varphi_2 \vee \big(\varphi_1 \wedge \mathbf{AX}\big(\mathbf{A}\varphi_1\mathcal{U}\varphi_2\big)\big)$$

Calcul symbolique (suite)

$$Sat(\mathbf{AX}\varphi) \;=\; Q \setminus Pre\big(Q \setminus Sat(\varphi)\big)$$

– opérateurs $\mathbf{E}\varphi_1\mathcal{U}\varphi_2$ et $\mathbf{A}\varphi_1\mathcal{U}\varphi_2$ (Cf. page 50) :

$$\mathbf{E}\varphi_1\mathcal{U}\varphi_2 \;\equiv\; \varphi_2 \vee \big(\varphi_1 \wedge \mathbf{EX}(\mathbf{E}\varphi_1\mathcal{U}\varphi_2)\big)$$
$$\mathbf{A}\varphi_1\mathcal{U}\varphi_2 \;\equiv\; \varphi_2 \vee \big(\varphi_1 \wedge \mathbf{AX}(\mathbf{A}\varphi_1\mathcal{U}\varphi_2)\big)$$

Calcul symbolique (suite)

$$Sat(\mathbf{AX}\varphi) \;=\; Q \setminus Pre\big(Q \setminus Sat(\varphi)\big)$$

– opérateurs $\mathbf{E}\varphi_1\mathcal{U}\varphi_2$ et $\mathbf{A}\varphi_1\mathcal{U}\varphi_2$ (Cf. page 50) :

$$\mathbf{E}\varphi_1\mathcal{U}\varphi_2 \;\equiv\; \varphi_2 \vee \big(\varphi_1 \wedge \mathbf{EX}(\mathbf{E}\varphi_1\mathcal{U}\varphi_2)\big)$$
$$\mathbf{A}\varphi_1\mathcal{U}\varphi_2 \;\equiv\; \varphi_2 \vee \big(\varphi_1 \wedge \mathbf{AX}(\mathbf{A}\varphi_1\mathcal{U}\varphi_2)\big)$$

– ensemble 2^Q ordonné par $\subseteq$: $(2^Q, \subseteq)$,
– $Sat(\mathbf{E}\varphi_1\mathcal{U}\varphi_2)$ est le *plus petit point fixe* de la fonction f :

$$f(Y) = Sat(\varphi_2) \cup \big(Sat(\varphi_1) \cap Sat(\mathbf{EX}Y)\big)$$

Points fixes de fonctions monotones

Théorème 2 (Knaster-Tarski [19]) *Q un ordre partiel complet avec $\bot$ élement minimal, $f : Q \to Q$ monotone, alors f admet un plus petit point fixe $\mu(f)$ et $\mu(f) = \cup_{i=0}^{\infty} f^i(\bot) = f^*(\bot)$. Si Q est fini, ordonné par l'inclusion, $f : Q \to Q$ monotone, alors*
$\exists k \leq |Q|, \quad \mu(f) = f^k(\emptyset)$. □

- $Sat(\mathbf{E}\varphi_1 \mathcal{U} \varphi_2)$ est un point fixe de
  $f(Y) = Sat(\varphi_2) \cup \big(Sat(\varphi_1) \cap Sat(\mathbf{EX}Y)\big)$
- $Sat(\mathbf{E}\varphi_1 \mathcal{U} \varphi_2)$ est le plus petit point fixe :
  1. $q \in f^n(\emptyset) \equiv \exists k \leq n, q = s_0, \sigma = s_0 s_1 s_2 \cdots s_k,$
     $\sigma \in Exec(S), \forall i < k \quad s_i \models \varphi_1, s_k \models \varphi_2$
  2. $q \in Sat(\mathbf{E}\varphi_1 \mathcal{U} \varphi_2) \Longrightarrow \exists k \in \mathbb{N}, q \in f^k(\emptyset)$
  3. $Sat(\mathbf{E}\varphi_1 \mathcal{U} \varphi_2) \subseteq \cup_{i=0}^{\infty} f^i(\emptyset)$

Représentation symbolique des ensembles d'états

- représenter $Sat(p)$, $\forall p \in AP$
- définir Pre sur représentation *symbolique*
- opérations $\cup$, $\cap$, $\setminus$ sur représentation symbolique
- calcul de plus petit point fixe : calcul *itératif* de Pre

$$Sat(\mathbf{E}\varphi_1 \mathcal{U} \varphi_2) = \mu X.Sat(\varphi_2) \cup \Big(Sat(\varphi_1) \cap Pre\big(Sat(X)\big) \Big)$$

- *arrêt* du calcul de point fixe : test *d'égalité* entre deux représentations

Binary Decisions Diagrams (BDD) [15]

5.2– Binary Decision Diagrams (BDD)

– but : représenter de façon *compacte* les *expressions booléennes*
– $b_1 \Longrightarrow (b_2 \vee b_3)$; arbre de *choix* : n variables $\Longrightarrow 2^{n+1} - 1$ nœuds

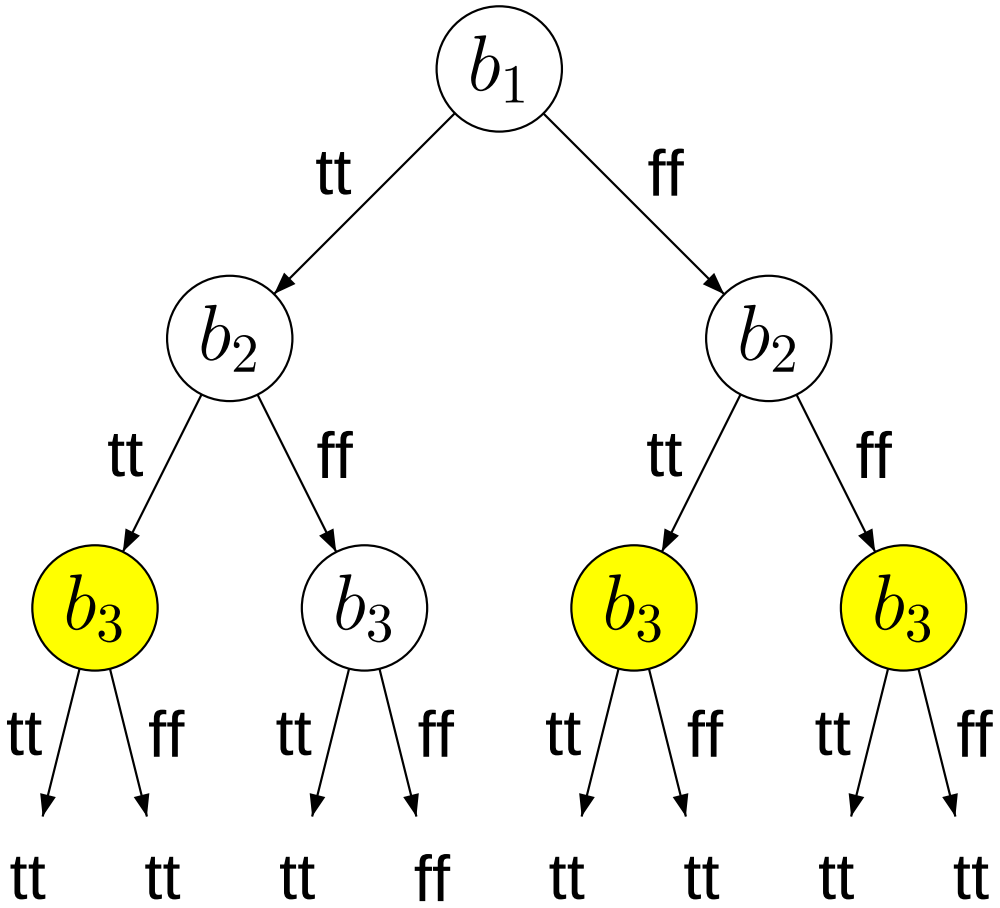
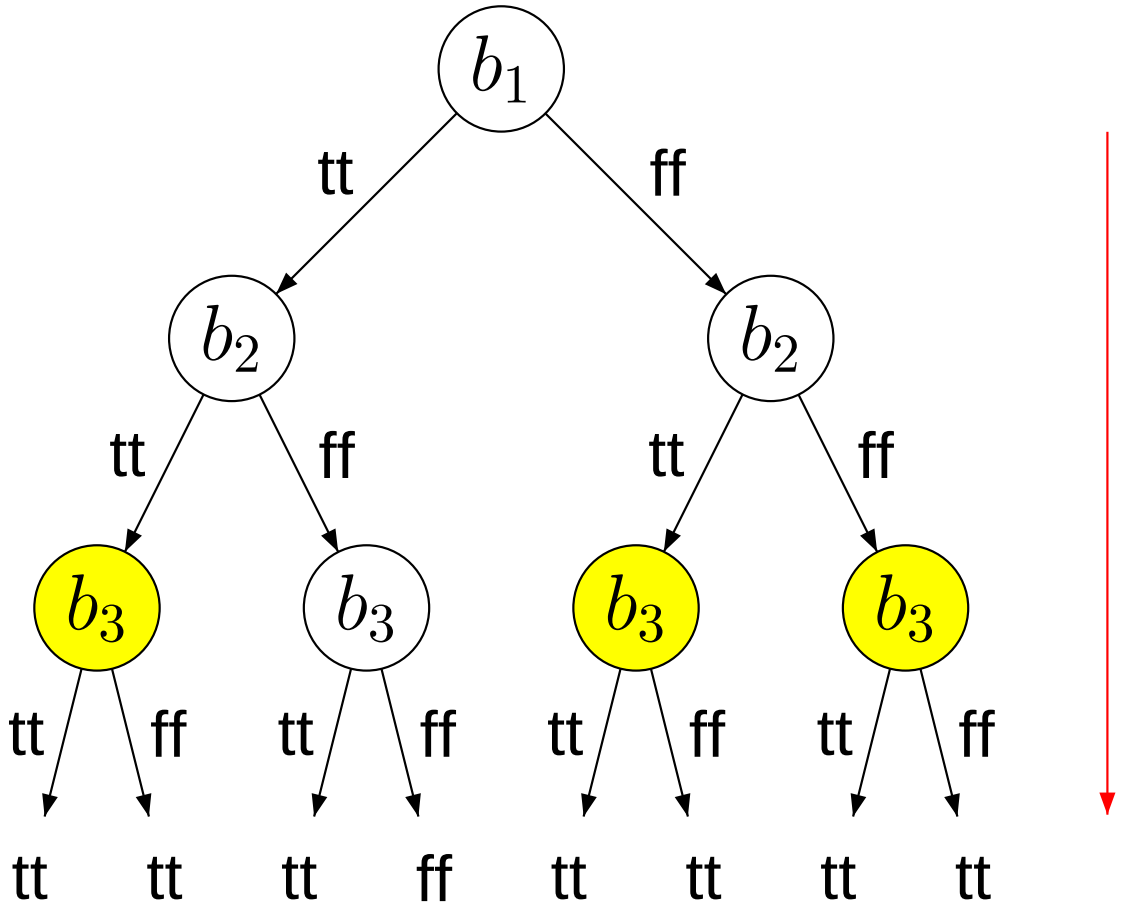


FIG. 5.1 – Arbre de choix pour $b_1 \Longrightarrow (b_2 \vee b_3)$

5.3– Binary Decision Diagrams (BDD)

– but : représenter de façon *compacte* les *expressions booléennes*
– $b_1 \Longrightarrow (b_2 \vee b_3)$; arbre de *choix* : n variables $\Longrightarrow 2^{n+1} - 1$ nœuds

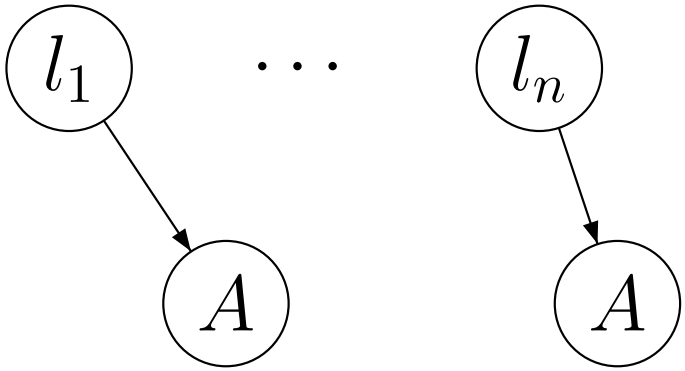


ordered BDD (OBDD)
$b_1 < b_2 < b_3$

FIG. 5.2 – Arbre de choix pour $b_1 \Longrightarrow (b_2 \vee b_3)$

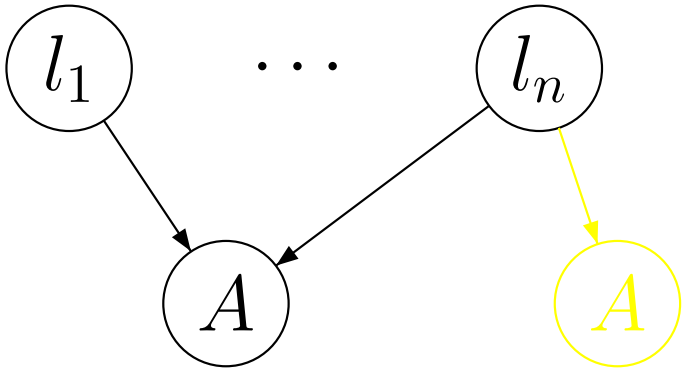
Réduction de l'arbre de choix

1. *regrouper* les sous *arbres identiques*



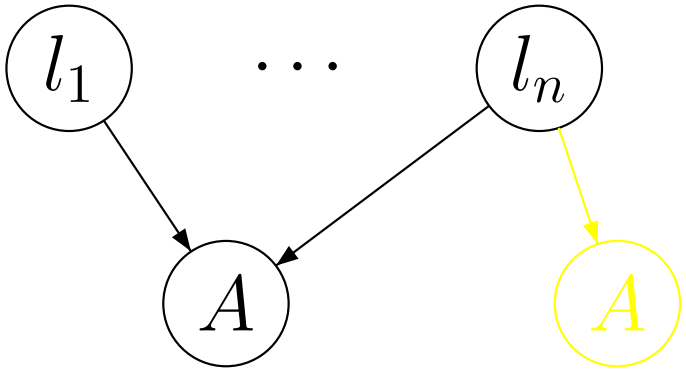
Réduction de l'arbre de choix

1. *regrouper* les sous *arbres identiques*

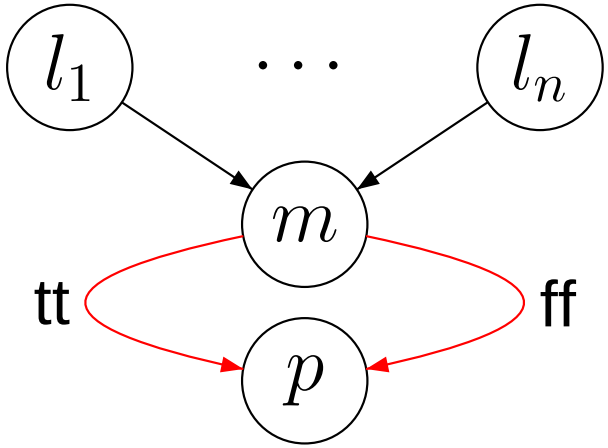


Réduction de l'arbre de choix

1. *regrouper* les sous *arbres identiques*

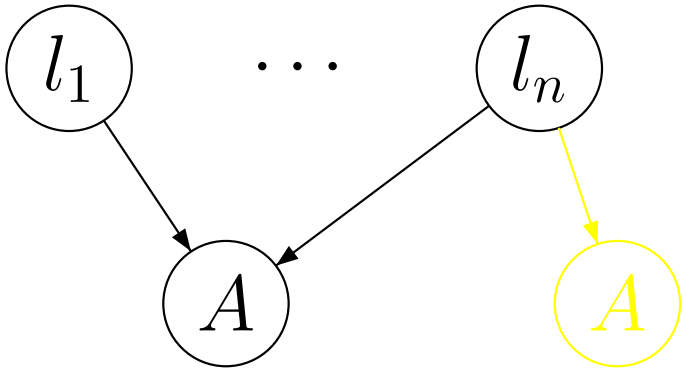


2. *enlever* les *faux choix*

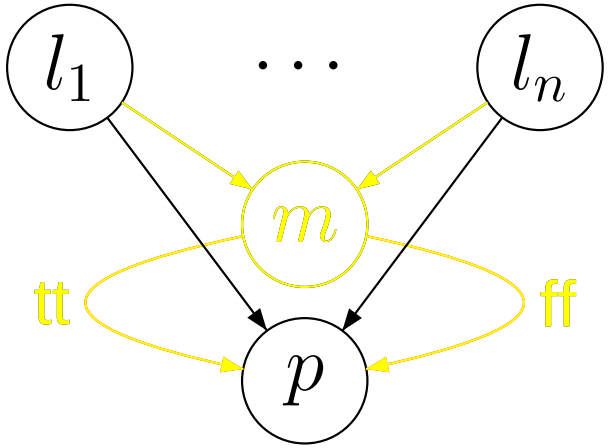


Réduction de l'arbre de choix

1. *regrouper* les sous *arbres identiques*

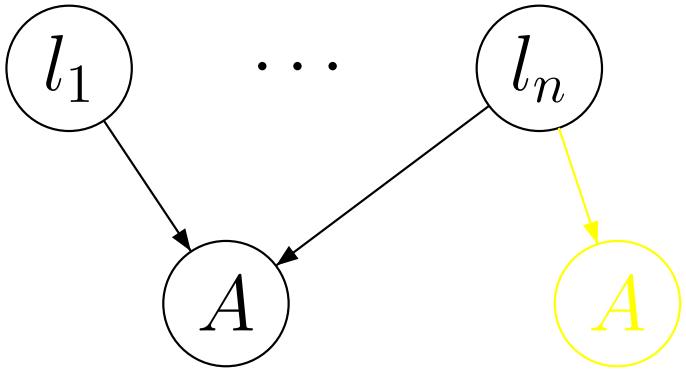


2. *enlever* les *faux choix*

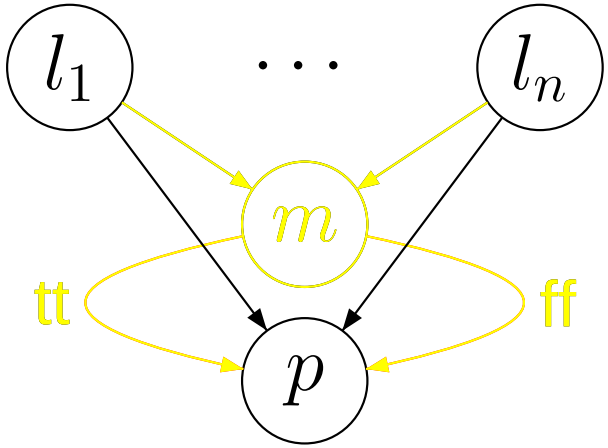


Réduction de l'arbre de choix

1. *regrouper* les sous *arbres identiques*



2. *enlever* les *faux choix*



– *réduction* OBDD $=$ *itérer* 1) et 2) au *maximum*
– on obtient un *Reduced* OBDD (*DAG*, Direct Acyclic Graph, $\neq$ arbre)

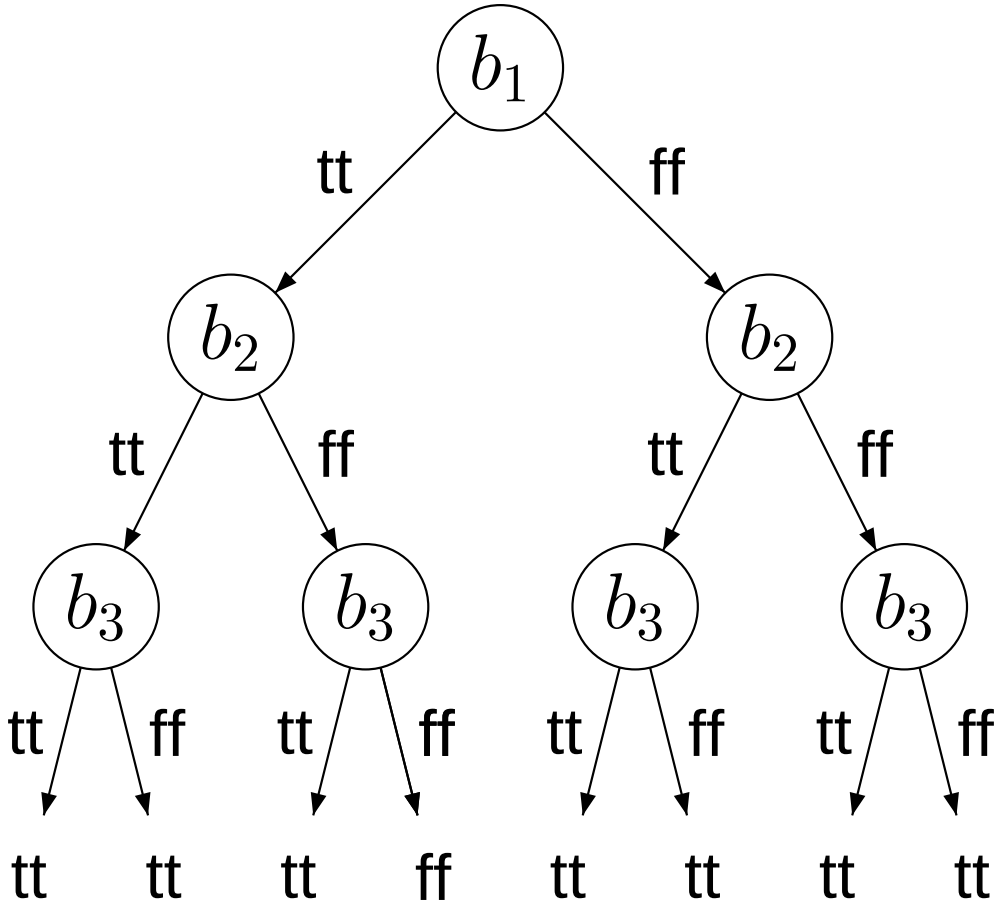
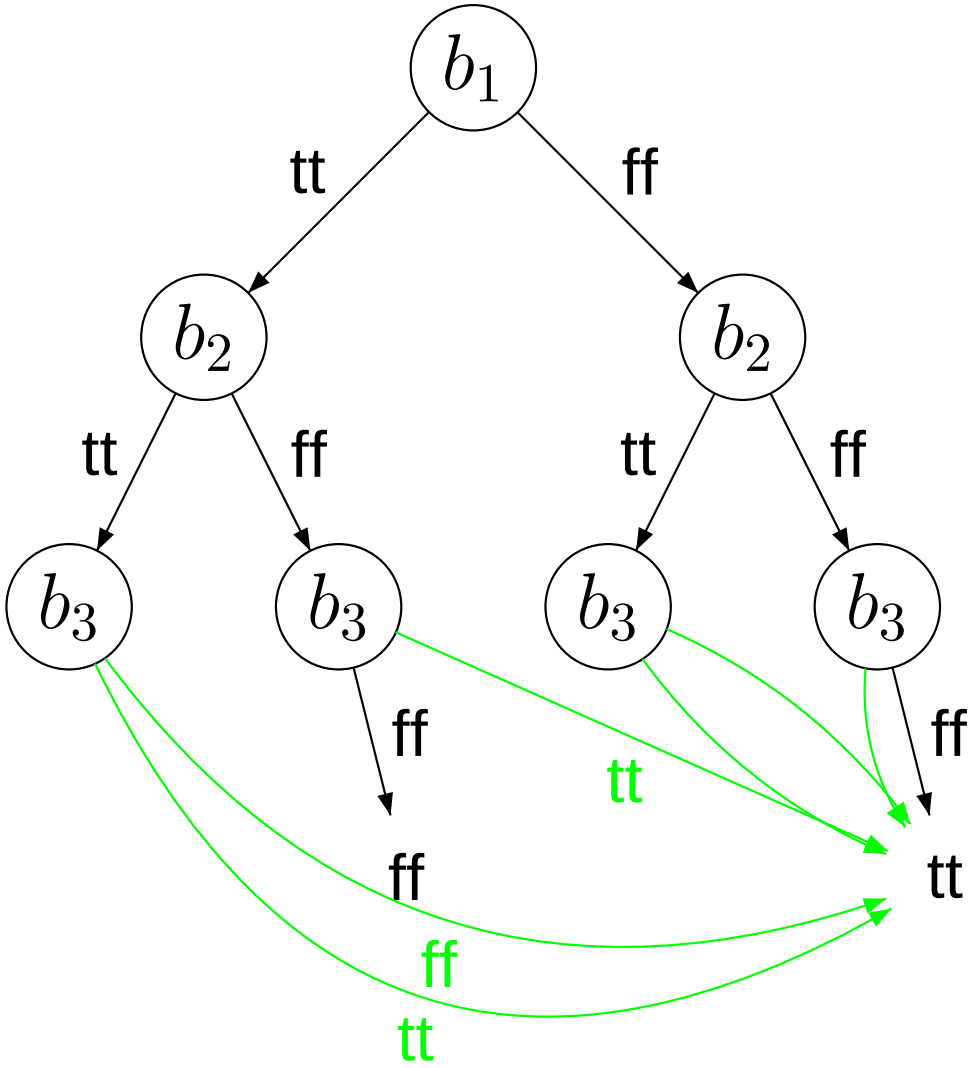
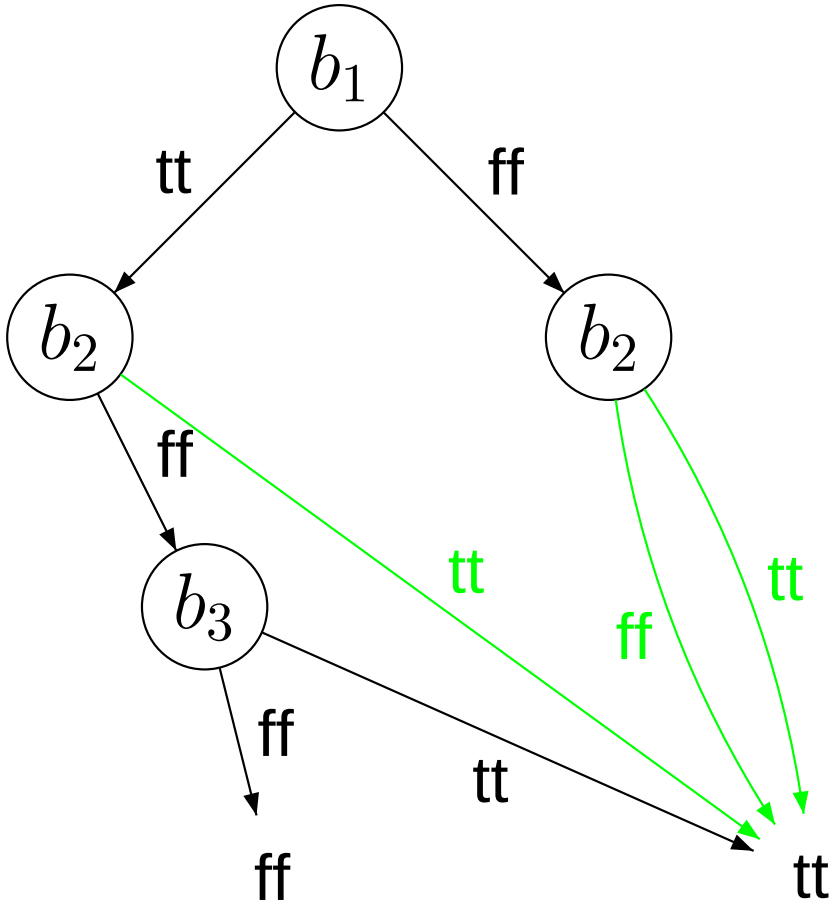
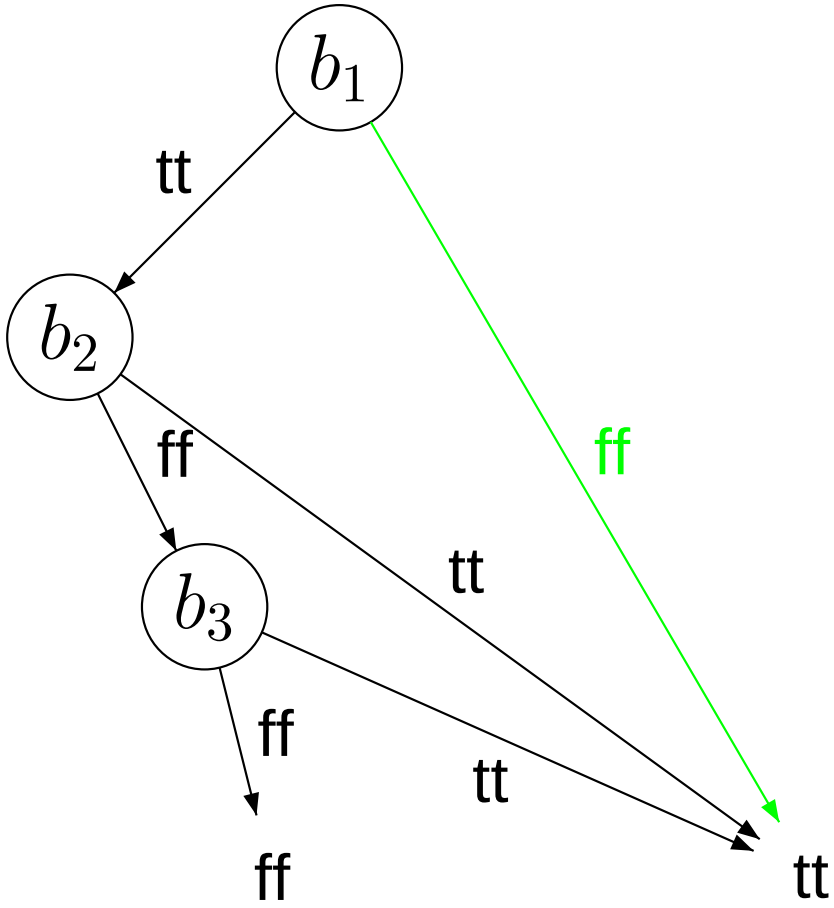


FIG. 5.3 – Réduction de l'OBDD de $b_1 \Longrightarrow (b_2 \vee b_3)$


b_1
tt
ff
b_2
b_2
tt
ff
tt
ff
b_3
b_3
b_3
b_3
ff
ff
tt
ff
tt
ff
tt


b_1
tt
ff
b_2
b_2
ff
tt
ff
tt
b_3
ff
tt
ff
tt


b_1
tt
b_2
ff
ff
tt
b_3
ff
tt
ff
tt

Intérêts des ROBDDs

- *diminution* de la *taille* de *l'arbre* de choix
- *représentation canonique* :

  Théorème 3 (Canonicité des ROBDDs) *n variables booléennes ordonnées $b_1 < b_2 < \cdots < b_n$. F une expression booléenne sur les b_i. Alors il existe un unique ROBDD bdd(f) représentant f.* □
- conséquences :
  - bdd$(f) =$ bdd$(\mathsf{ff}) \iff f = \mathsf{ff}$
  - bdd$(\neg f) =$ échanger tt et ff dans bdd(f)
  - $f_1 = f_2 \iff$ bdd$(f_1) =$ bdd(f_2)

    possibilité de *partage* en *mémoire* des sous *arbres* :

    bdd$(f_1) \equiv$ bdd$(f_2) \iff adr($bdd$(f_1)) = adr($bdd$(f_2))$
  - test *d'égalité* en *temps constant* par comparaison des *adresses*

Influence de l'ordre des variables

- taille des ROBDDs *sensible* à l'*ordre des variables*
- ex : construire les BDDs de $(b_1 \wedge b_2) \vee (b_3 \wedge b_4) \vee (b_5 \wedge b_6)$ avec les ordres :
  1. $b_1 > b_2 > b_3 > b_4 > b_5 > b_6$
  2. $b_1 > b_3 > b_5 > b_2 > b_4 > b_6$
- trouver un ordre *optimal* ? . . . trop couteux en *temps*
- il existe des *stratégies* (heuristiques) pour *trouver* des *bons ordres*

5.4– Opérations sur les ROBDDs

- un ROBDD $\beta \equiv$ nœud racine de β
- $\beta \equiv \text{nil} \iff \beta = \text{ff}$; $\beta_1 \equiv \beta_2 \iff adr(\beta_1) = adr(\beta_2)$ *(temps constant)*
- $\bar{\beta}$: *échanger* les nœuds tt et ff (reste un ROBDD)
- *opérations binaires* : $\beta_1 \, op \, \beta_2$
  ordres *compatibles* sur β_1 et β_2 (*même ordre* sur les variables *communes*)

Théorème 4 (Shannon) *f, g des formules booléennes ; x une variable ; alors* $f \equiv (x \wedge f[x := \mathit{tt}]) \vee (\neg x \wedge f[x := \mathit{ff}])$ *et*

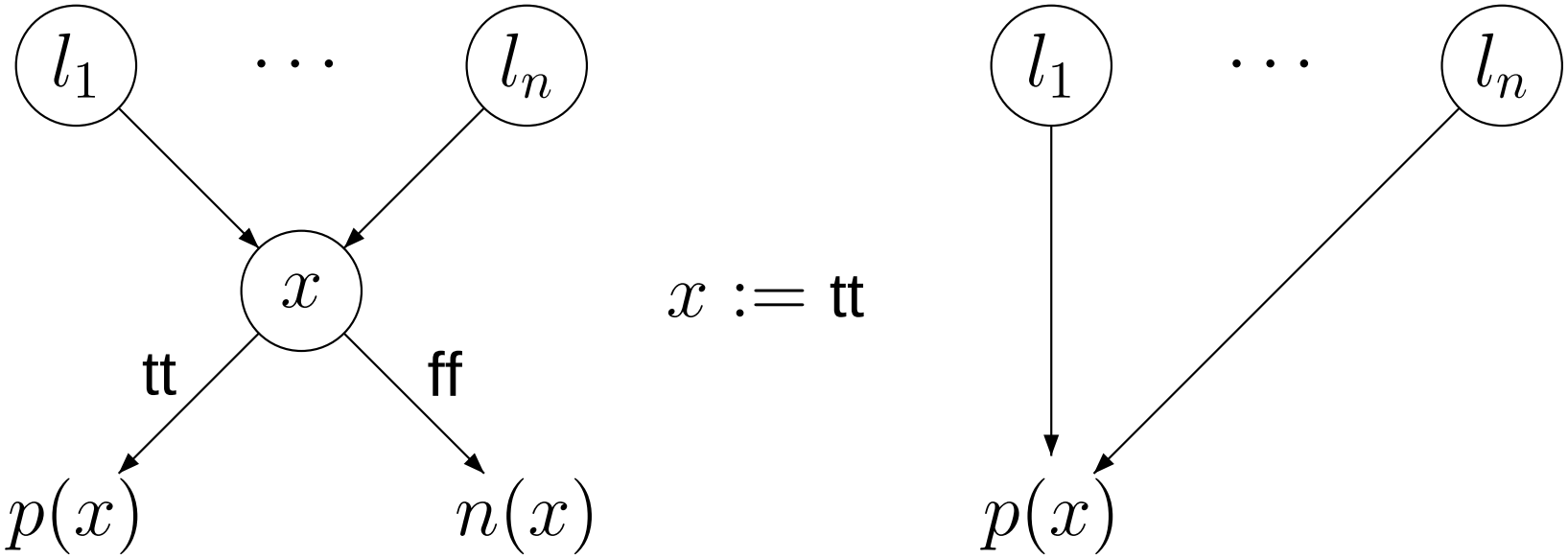
$$\begin{aligned} f \, op \, g \;\; = \;\; & \big(x \wedge (f[x := \mathit{tt}] \, op \, g[x := \mathit{tt}])\big) \\ & \vee \big(\neg x \wedge (f[x := \mathit{ff}] \, op \, g[x := \mathit{ff}])\big) \end{aligned}$$

– application aux (R)OBDDs : β_1, β_2 deux ROBDDs ;
  $p(\beta) =$ partie fille *tt* issue *de* β ; $n(\beta) =$ partie fille *ff* issue de β
  x variable de β : $\beta = (x \wedge p(\beta)) \vee (\neg x \wedge n(\beta))$
  x_i la variable de β_i :
  1. $x_1 = x_2$:
     $\beta_1 \, op \, \beta_2 = \big(x_1 \wedge (p(\beta_1) \, op \, p(\beta_2))\big) \vee \big(\neg x_1 \wedge (n(\beta_1) \, op \, n(\beta_2))\big)$
  2. $x_1 < x_2$:
     $\beta_1 \, op \, \beta_2 = \big(x_1 \wedge (p(\beta_1) \, op \, \beta_2)\big) \vee \big(\neg x_1 \wedge (n(\beta_1) \, op \, \beta_2)\big)$
– tt $\cap \, n = n$, ff $\cap \, n =$ ff, tt $\cup \, n =$ tt, ff $\cup \, n = n$
– appliquer la *réduction* après op
  (nouveau BDD pas forcément réduit)

exemple : $(b_1 \Longrightarrow b_2) \wedge (b_2 \vee b_3)$

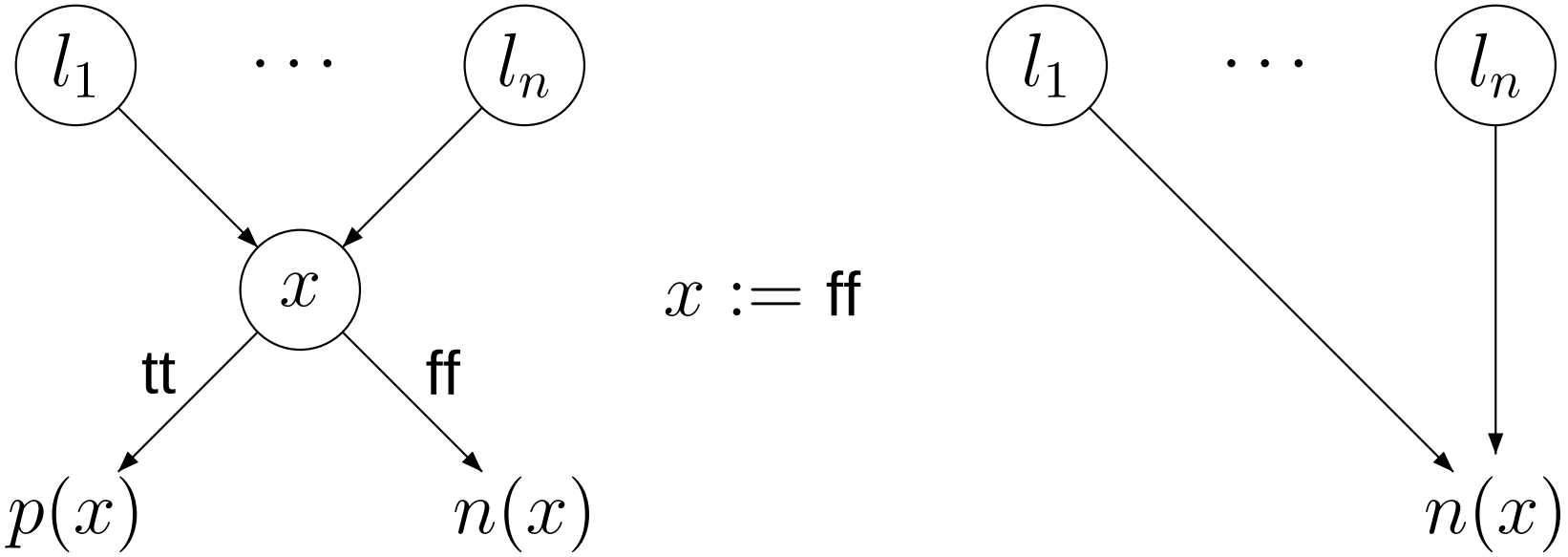
Opérations sur les ROBDDs (suite)

– *projection* (ou restriction) : ROBDD représentant $f[x := \text{tt}]$ ou $f[x := \text{ff}]$ à partir du ROBDD de f



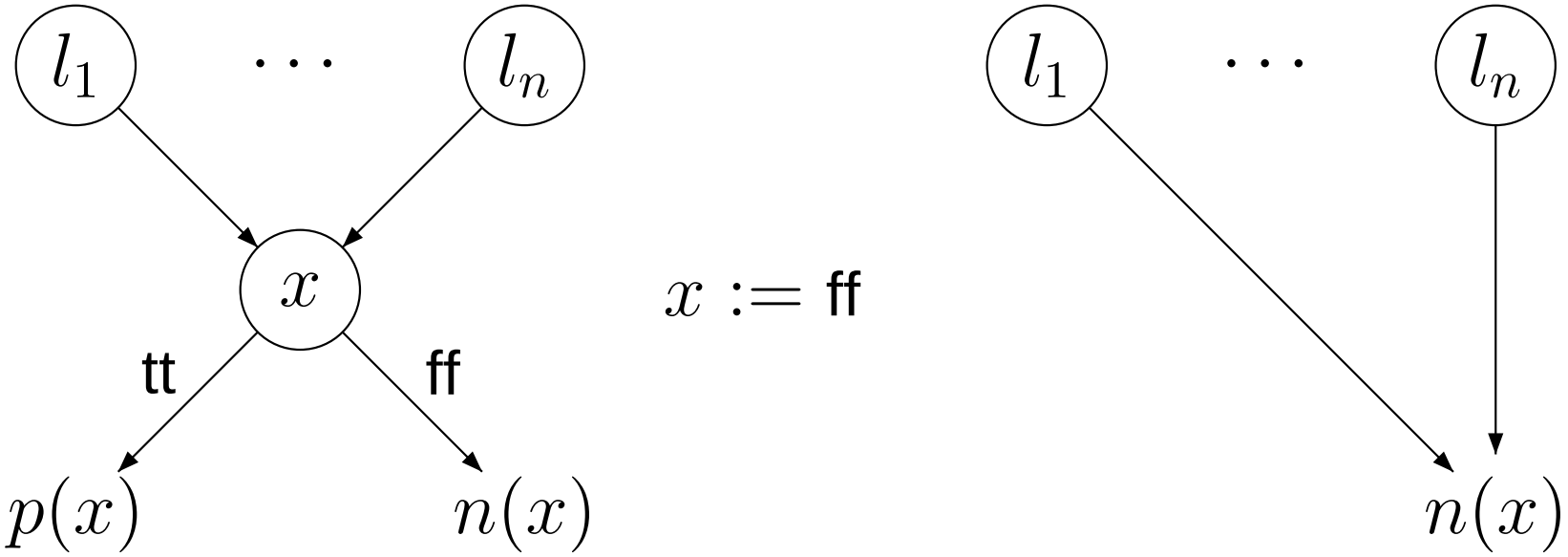
Opérations sur les ROBDDs (suite)

– *projection* (ou restriction) : ROBDD représentant $f[x := \text{tt}]$ ou $f[x := \text{ff}]$ à partir du ROBDD de f



Opérations sur les ROBDDs (suite)

– *projection* (ou restriction) : ROBDD représentant $f[x := \text{tt}]$ ou $f[x := \text{ff}]$ à partir du ROBDD de f



– *abstraction* : $\exists x.f \equiv f[x := \text{tt}] \vee f[x := \text{ff}]$
faire uniquement $\vee$ sur les sous arbres issus de x
Remarque : $\forall x.f \equiv f[x := \text{tt}] \wedge f[x := \text{ff}]$

5.5– Model-checking à base de ROBDDs

- codage des *états* de Q : *vecteur de n bits* $b_1 \cdots b_n$ tel que $|Q| \leq 2^n$
  exemple : 5 états $\longrightarrow$ 4 bits
  variables entières idem
- $s \in Q$ codé en $\bar{b}$ et $bdd(Q) = \cup_{s \in Q} bdd(s)$
- codage des *transitions* : *vecteur de longueur double* $b_1 \cdots b_n b'_1 \cdots b'_n$
  $(s, s') \in \longrightarrow \Longrightarrow b_1 \cdots b_n b'_1 \cdots b'_n$ dans $bdd(\longrightarrow)$
- $bdd(\longrightarrow) = \cup_{(s,s') \in \longrightarrow} bdd((s, s'))$
- calcul de $\cap, \cup, \setminus$: OK
- *calcul de $Pre(S)$* :
  1. *primer* le $bdd(S)$ en $bdd(S)'$ avec variables b_i devient b'_i
  2. calculer $\alpha = bdd(S)' \cap bdd(\longrightarrow)$
  3. calculer $\exists \bar{b}'.\alpha$

Application du model checking à base de BDDs

- model checking de *CTL* : SMV [21]
- problèmes et améliorations :
  - $bdd(\rightarrow)$ ... grand ! Garder les BDDs de chaque transition t_i
    $Pre(S) = \cup_{t_i} Pre_{t_i}(S)$
  - *ordre* des variables : heuristiques !
- autres applications des BDDs : conception de circuits

Troisième partie : Systèmes temporisés

Chapitre 6 : Automates Temporisés

Sommaire

Temps quantitatif

- SdeT : temps *logique* et *discret*
- certaines *propriétés* dépendent de *contraintes temporelles quantitatives*

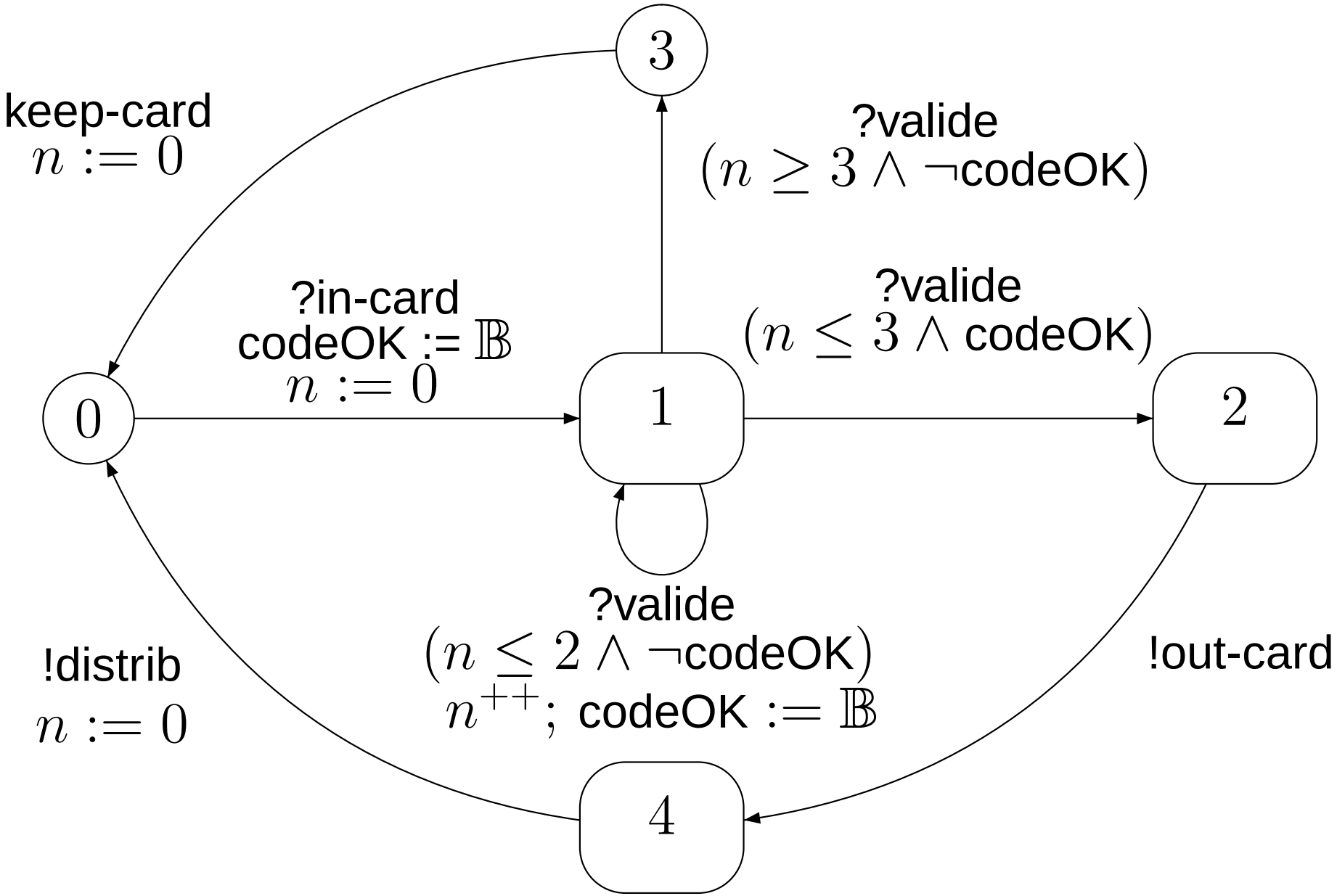
  ex : GAB tel que trois essais maxi et code bon en *moins de 6 secondes*
- *mesure* du temps :
  - temps *discret* dans $\mathbb{N}$: ajout d'une *transition* $tick$ comptant les unités

    $tick =$ variable entière . . . explosion combinatoire !
  - temps *continu* dans $\mathbb{R}^+$

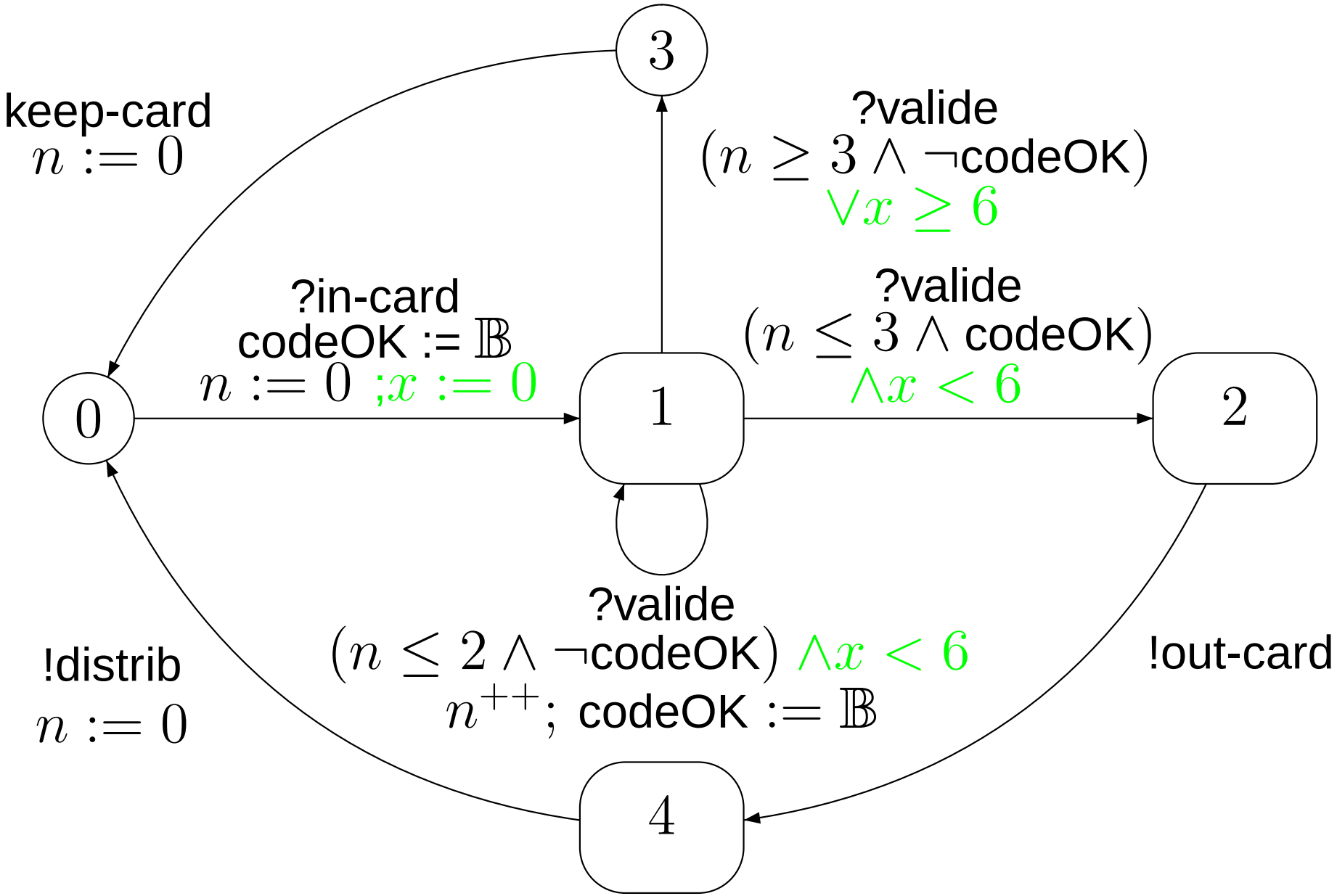
    SdeT $+$ ensemble d'*horloges réelles* (à valeurs ≥ 0) $\Longrightarrow$

automates temporisés [2]

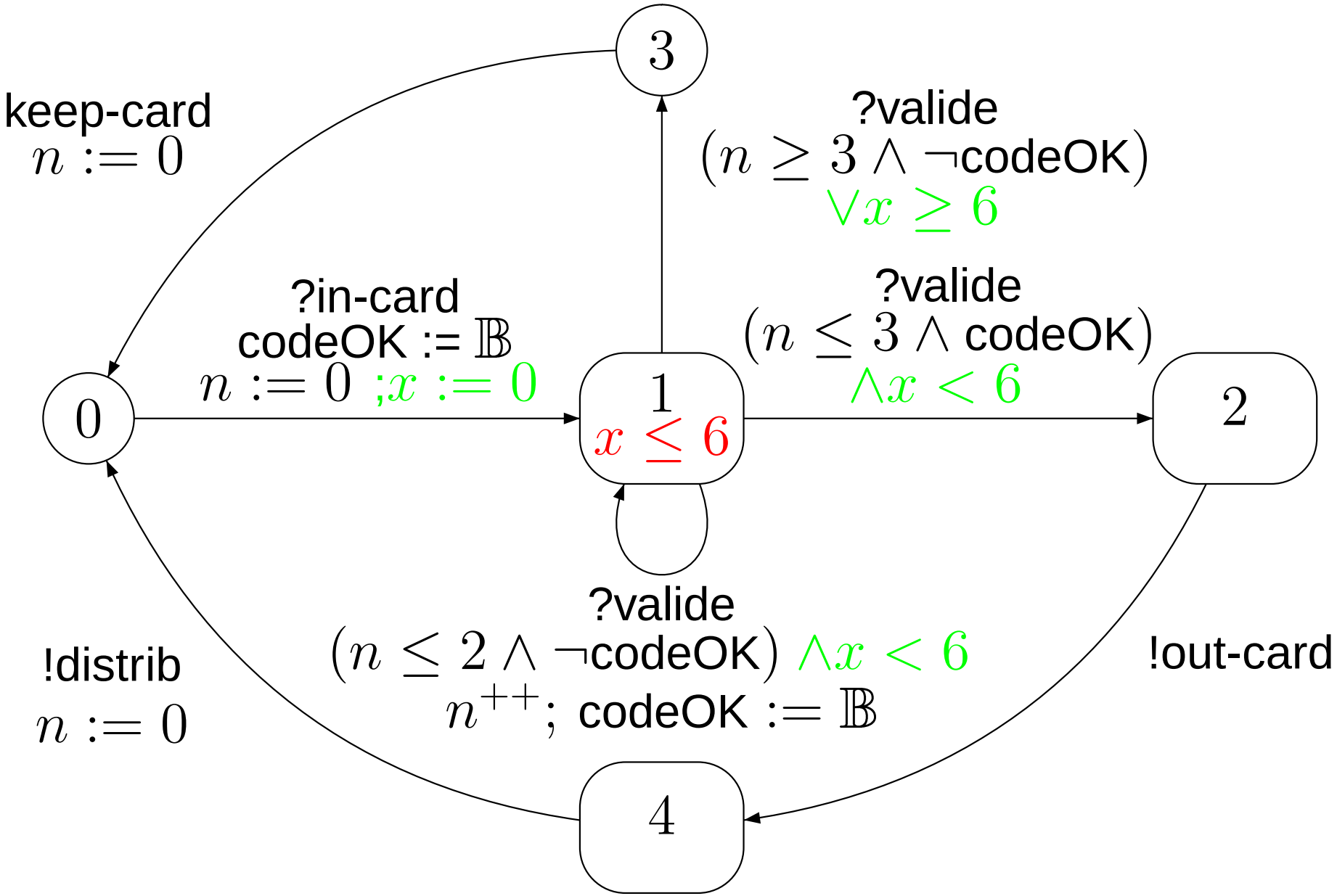
GAB temporisé



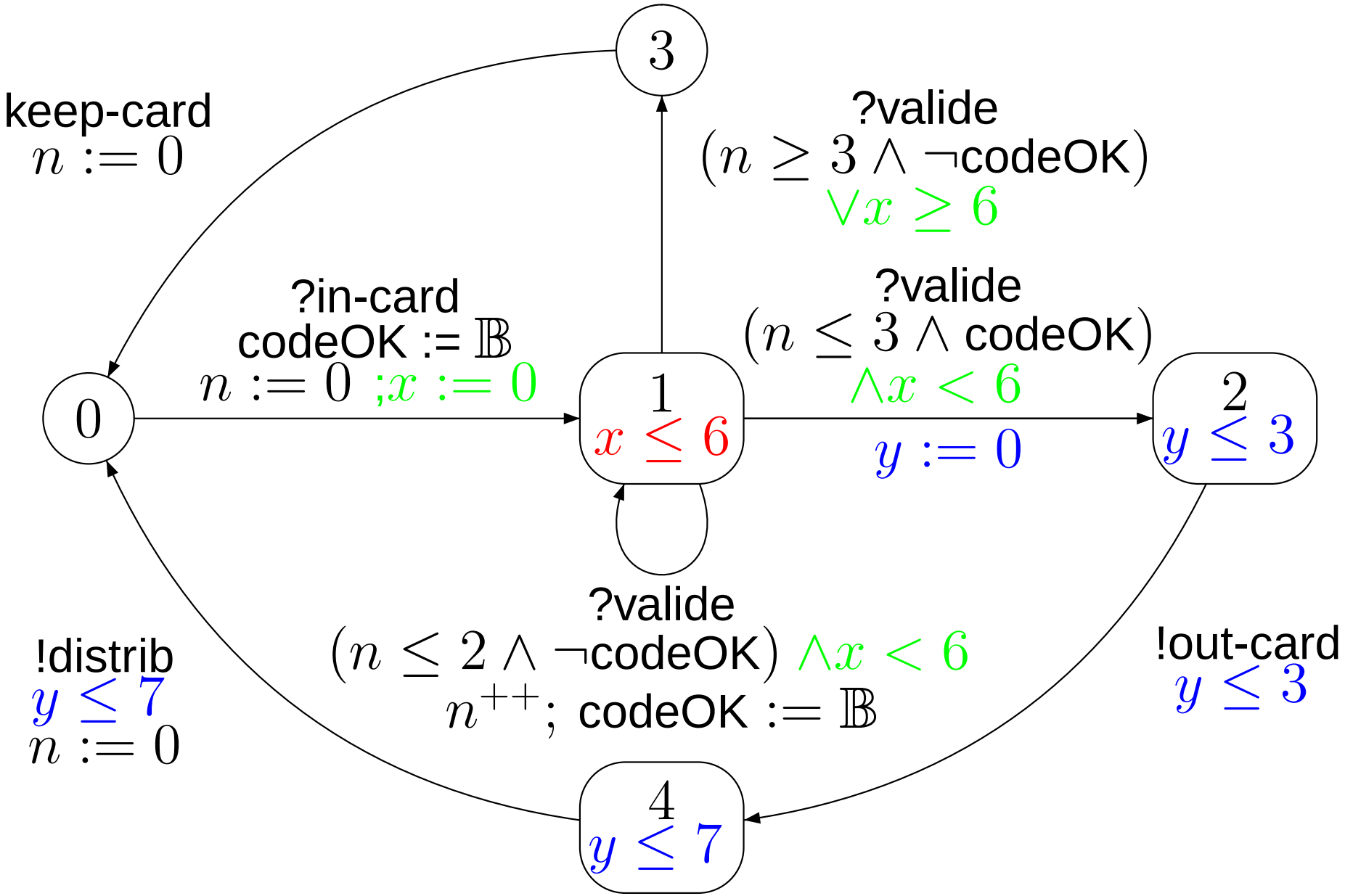
GAB temporisé



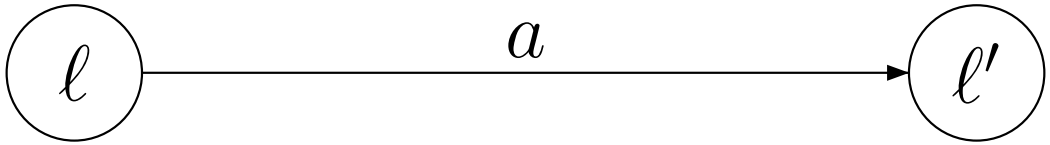
GAB temporisé



GAB temporisé

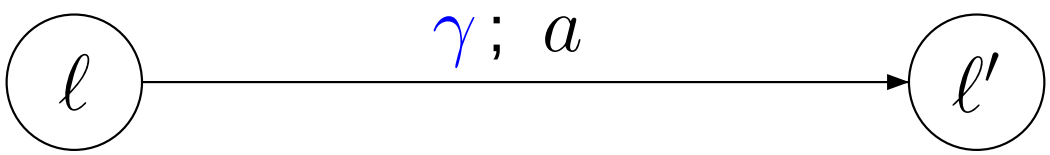


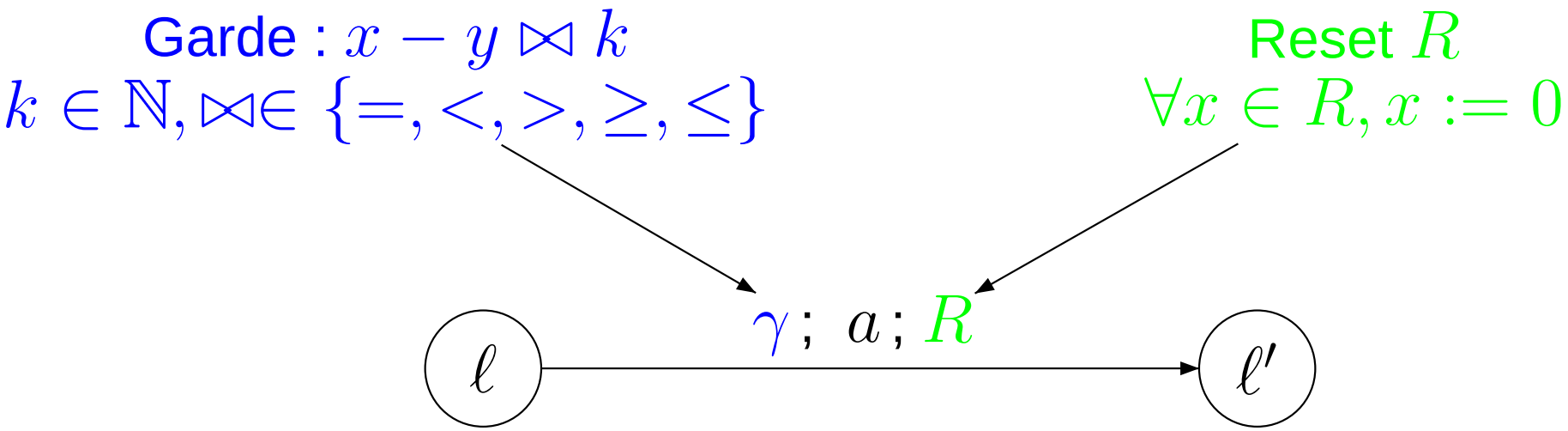
6.1– Automate temporisé

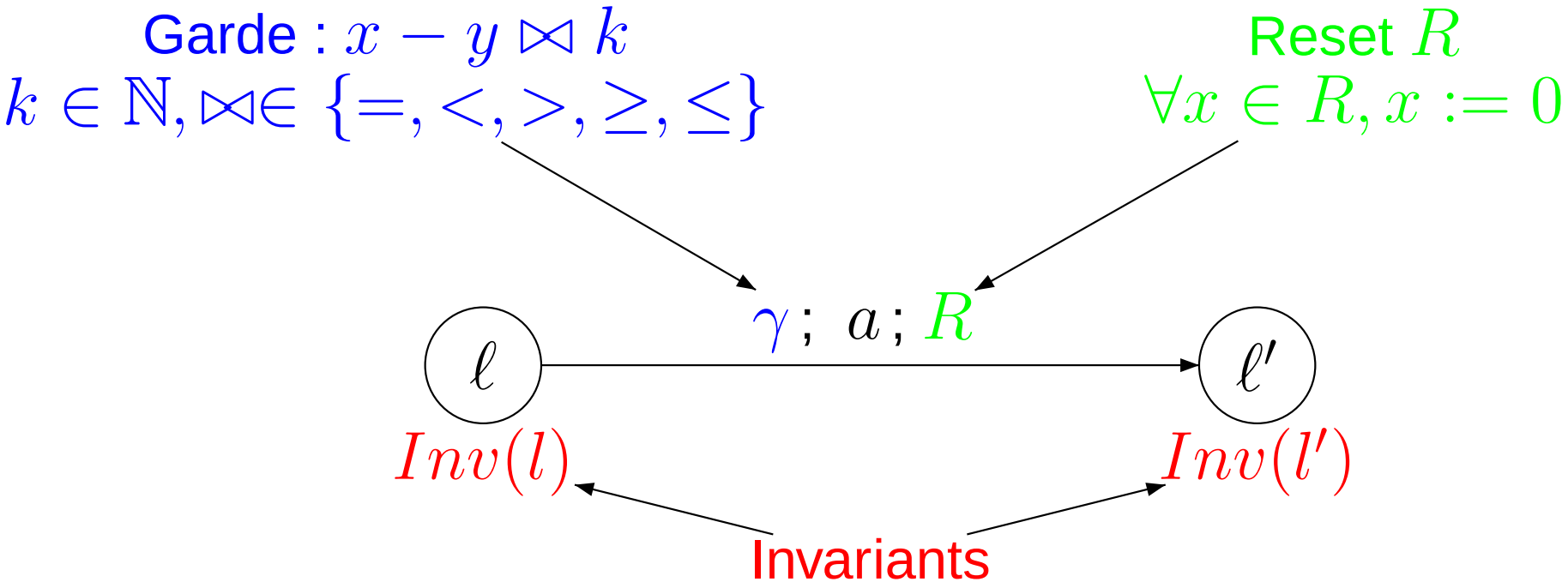


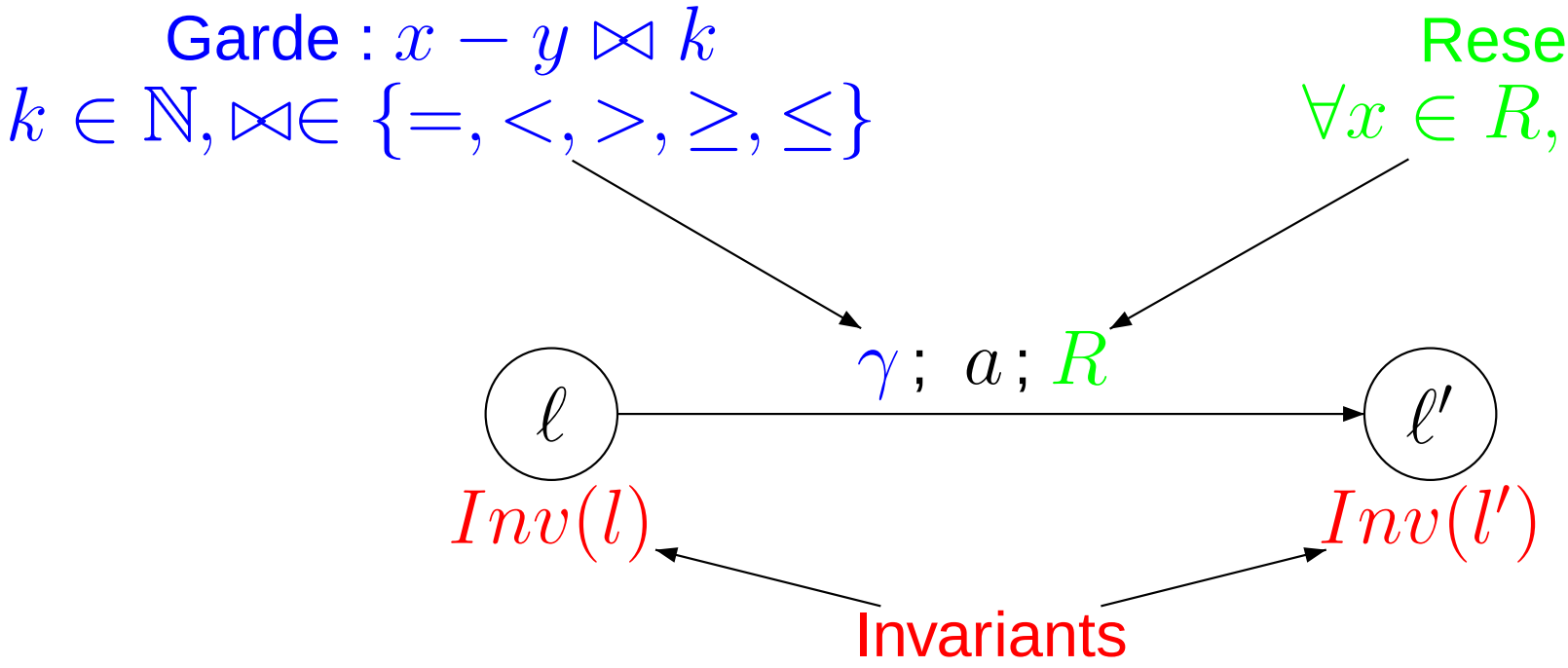
Garde : $x - y \bowtie k$

$k \in \mathbb{N}, \bowtie \in \{=, <, >, \geq, \leq\}$




Garde : $x - y \bowtie k$
$k \in \mathbb{N}, \bowtie \in \{=, <, >, \geq, \leq\}$
Reset R
$\forall x \in R, x := 0$
$\gamma\,;\ a\,;\ R$
ℓ
ℓ'


Garde : $x - y \bowtie k$
$k \in \mathbb{N}, \bowtie \in \{=, <, >, \geq, \leq\}$
Reset R
$\forall x \in R, x := 0$
$\gamma\,;\ a\,;\ R$
ℓ
ℓ'
$Inv(l)$
$Inv(l')$
Invariants



- *Horloges* : $C = \{x, y, \ldots\}$, $\dot{x} = \dot{y} = \cdots = 1$
- *Garde* $=$ *combinaison booléenne* de $x - y \bowtie k$
- *Invariants* $=$ *conjonctions* de $x \bowtie k$, $k \in \mathbb{N}$
- *évolution continue* : $v : C \to \mathbb{R}^+, t \in \mathbb{R}^+, (v + t)(x) = v(x) + t$

Définition formelle

$\mathcal{G} =$ ensemble des *contraintes de gardes*
$\mathcal{I} =$ ensemble des *contraintes d'invariants*

Définition 14 (Automate temporisé) *Un automate temporisé H est un tuple (N, l_0, C, A, E, Inv) avec :*

- N *ensemble fini de localités,*
- $l_0 \in N$ *la localité initiale,*
- C *un ensemble fini d'horloges (à valeur dans $\mathbb{R}^+$),*
- A *l'alphabet des actions,*
- $E \subseteq N \times \mathcal{C} \times A \times 2^C \times N$ *un ensemble fini de transitions ;*
  $e = \langle l, \gamma, a, R, l' \rangle \in E : \gamma =$ *garde de e, a action de e, R reset set.*
- $Inv \in \mathcal{I}$ *donne l'invariant associé à $l : \forall l \in N, Inv(l) = \bigwedge c \leq r$ avec $c \in C$ et $r \in \mathbb{N}$.* □

6.2– Sémantique des automates temporisés

Définition 15 (Sémantique) *La sémantique d'un AT H est un SdeT (temporisé) $S_H = (Q, q_0, B, \rightarrow)$ avec $Q = N \times (\mathbb{R}_+)^C$, $q_0 = (l_0, \overline{0})$ est l'état initial, $B = A \cup \delta(\mathbb{R}^+)$ l'alphabet d'actions et $\rightarrow$ définie par :*

$$(l, v) \xrightarrow{a} (l', v') \quad \text{ssi} \quad \begin{cases} \exists (l, \gamma, a, R, l') \in E \text{ s.t.} \\ \gamma(v) = \text{tt},\ v' = v[R \mapsto 0] \text{ et} \\ Inv(l')(v') = \text{tt} \end{cases}$$

$$(l, v) \xrightarrow{\delta(t)} (l', v') \quad \text{ssi} \quad \begin{cases} l = l' \qquad v' = v + t \quad \text{et} \\ \forall 0 \leq t' \leq t,\ Inv(l)(v + t') = \text{tt} \end{cases}$$

Une exécution d'un AT H est un chemin de S_A issu de q_0.
$[\![H]\!]$= ensemble des exécutions de H. □

Exemple

$$
\begin{aligned}
((0,0,\mathsf{ff});0,0) &\xrightarrow{\delta(1.2)} ((0,0,\mathsf{ff});x=1.2,y=1.2)\\
&\xrightarrow{\text{?in-card}} ((1,0,\mathsf{tt});x=0,y=1.2)\\
&\xrightarrow{\delta(2.134)} ((1,0,\mathsf{tt});x=2.134,y=3.334)\\
&\xrightarrow{\text{?valide}} ((2,0,\mathsf{tt});x=2.134,y=0)\\
&\xrightarrow{\delta(2.9999)} ((2,0,\mathsf{tt});x=5.1339,y=2.9999)\\
&\xrightarrow{\text{?out-card}} ((4,0,\mathsf{tt});x=5.1339,y=2.9999)\\
&\dots
\end{aligned}
$$

Exemple

$$((0,0,\mathsf{ff});x=y=0) \xrightarrow{\delta(1.2)} ((0,0,\mathsf{ff});x=1.2,y=1.2)$$

$$\xrightarrow{\text{?in-card}} ((1,0,\mathsf{tt});x=0,y=1.2)$$

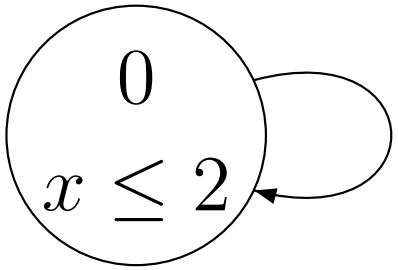
$$\cdots$$

- forme *canonique* des exécutions : $s \xrightarrow{\delta(t)} s' \xrightarrow{a} s'' \equiv s \rightarrow^{t}_{a} s''$
  $\sigma = s_0 \rightarrow^{t_1}_{a_1} s_1 \rightarrow^{t_2}_{a_2} \cdots \rightarrow^{t_n}_{a_n} s_n \cdots$
- *traces temporisées* : $tr(\sigma) = (a_1,t_1)(a_2,t_2)\cdots(a_n,t_n)$
  durée de $\sigma = \sum_{i=1}^{n} t_i$
  $Untimed(\sigma) = a_1 a_2 \cdots a_n$
- *langage temporisé* $LT(H)$ accepté par H :

$$LT(H) = \{w \in (Q \times \mathbb{R}^+)^*, \exists \sigma \in [\![H]\!], w = tr(\sigma)\}$$

6.3– Propriétés des automates temporisés

caractère *Zénon* :



$a\,;\, x \leq 2$

$\exists$ un algorithme décidant le caractère Zénon [11]

– *infinité* de transitions discrètes en un temps *fini*

– $0 \xrightarrow{\delta(1)} 0 \xrightarrow{\delta(\frac{1}{2})} 0 \xrightarrow{\delta(\frac{1}{4})} \cdots \xrightarrow{\delta(\frac{1}{n^2})} \cdots$

divergence du *temps* :

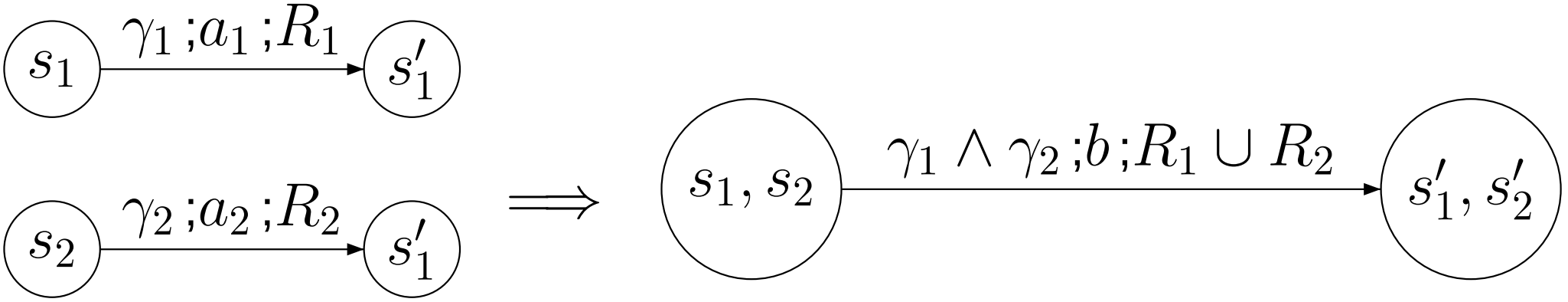
– $\rho = (a_1, t_1)(a_2, t_2) \cdots (a_n, t_n) \cdots$ une exécution

– $\forall t \in \mathbb{R}^+, \exists i, \sum_{k=1}^{i} t_i > t$

6.4– Produit synchronisé d'automates temporisés

Produit syntaxique

- *synchronisation* avec *renommage* : $f(a_1, a_2) = b$
- $Inv\big((s, s')\big) = Inv(s) \wedge Inv(s')$
- *asynchronisme* : chaque AT peut *implicitement* faire $s \xrightarrow{\bullet} s$
- *construction* de $A_1 \times A_2 \Longrightarrow$ AT

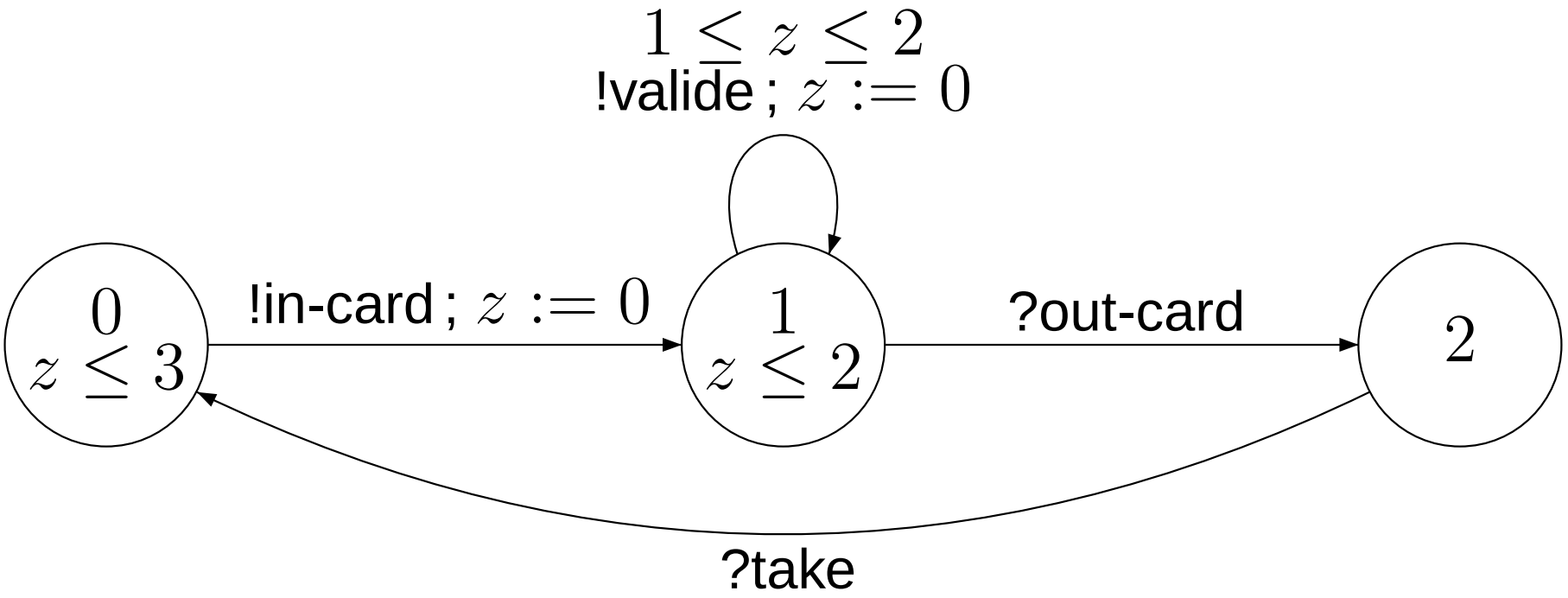


Caractérisation sémantique

Définition 16 $H_1, \ldots, H_n$ *n AT avec* $H_i = (N_i, l_{i,0}, C_i, A_i, E_i, Inv_i)$, *et* f *une fonction de synchronisation.*
La sémantique de $(H_1| \ldots |H_n)_f$ *est un SdeT* $S = (Q, q_0, B, \rightarrow)$ *avec*

- $B = \cup_i A_i \bigcup \mathbb{R}_+$
- $Q = N_1 \times \ldots \times N_n \times (\mathbb{R}_+)^C$,
- $q_0 = ((l_{1,0}, \ldots, l_{n,0}), \overline{0})$
- *et* $\rightarrow$ *définie par :*
  - $(\overline{l}, v) \xrightarrow{b} (\overline{l'}, v')$ *ssi* $\exists (a_1, \ldots, a_n) \in A_1 \times A_2 \times \cdots \times A_n$ *tel que* $f(a_1, \ldots, a_n) = b$ *et* $\forall i \ (l_i, v_i) \xrightarrow{a_i} (l'_i, v'_i)$
  - $(\overline{l}, v) \xrightarrow{\delta(t)} (\overline{l}, v')$ *ssi* $\forall i \in [1..n], (l_i, v_i) \xrightarrow{\delta(t)} (l_i, v'_i)$ □

Le GAB et un utilisateur temporisé

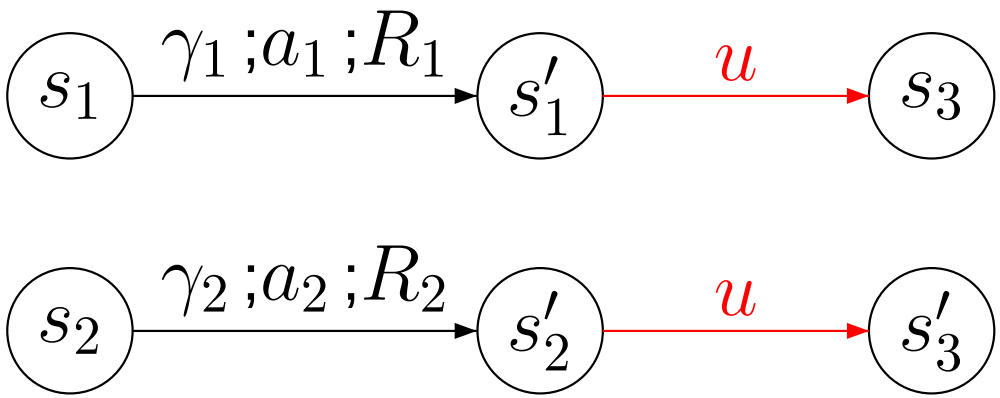


– *synchronisation* f définie par :

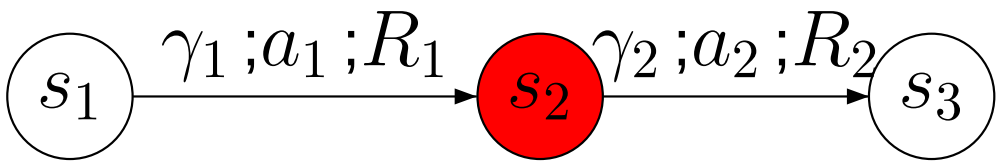
(	?in-card,	!in-card)	in
(	?valide,	!valide)	valide
(	!out-card,	?out-card)	out
(	!take,	?take)	take
(	keep-card,	•)	keep-card

6.5– Extensions simples

– *transitions (actions) urgentes* : tirées *dés que possible*



– *états atomiques (committed)* : *atomicité* entre a_1 et a_2 (section critique)



– $=$ *extensions syntaxiques* seulement (n'augmentent pas pouvoir d'expression des ATs)

6.6– Extensions . . . moins simples

Updatable Timed Automata [6]

- extensions sur les *contraintes* :
  $x + y < k$ model-cheking de TCTL *non décidable*
- extensions sur les *affectations*
  - affectations du type $x := x - 1, x + 1, y + c$ ou $x := [y + c, \infty]$:
    en génaral *model-checking* de TCTL *non décidable*
  - $\exists$ des sous classes *diagonal-free* où MC décidable
- extensions sur les *types d'horloges* :
  - horloges *suspendues* : *stopwatch* automata ; vitesse $\in \{0, 1\}$
  - cas général : *vitesse* $\neq 1 =$ *automates hybrides* ([12])
- *paramétrage* des valeurs des constantes des automates
  automates *paramétrés*

Chapitre 7 : Logique Temporelle TCTL – temps continu

Sommaire

Logique temporelle quantitative

- but : exprimer des *propriétés* contenant des informations *quantitatives*
- comment : extension de CTL en *Timed CTL* [2]
- exemple : tt$\mathcal{U}_{<2}q$
  q sera vraie dans moins de 2 unités de temps
- *interprétation* des formules de la logique sur des *systèmes de transitions temporisés*
- pas d'opérateurs *next* (X) car temps *continu*

7.1– Syntaxe de TCTL

on suppose donné un *ensemble d'horloges* $C \cup C'$
$C' =$ horloges de formule
$\bowtie \in \{<, \leq, =, \geq, >\}$

Définition 17 (Formules de TCTL) *Les formules de TCTL sont les formules définies inductivement par :*

– $\forall p \in AP, p \in$ *TCTL*
– $x, y \in C \cup C'$, $x - y \bowtie k \in$ *TCTL*
– $p, q \in$ *TCTL, alors* $p \vee q, \neg p \in$ *TCTL,*
– $p \in$ *TCTL, alors* $\mathbf{E} p \, \mathcal{U}_{\bowtie k} \, q, \mathbf{A} p \, \mathcal{U}_{\bowtie k} \, q \in$ *TCTL*
– $p \in$ *TCTL,* $c \in C'$ *alors* $c.p \in$ *TCTL*

□

Exemples de formules de TCTL

- **E**$p\,\mathcal{U}_{\geq 0}\,q \equiv$ **E**$p\,\mathcal{U}\,q$ *en CTL*
- abbréviations usuelles : **EF**$_{\bowtie k}p =$ **E** tt$\,\mathcal{U}_{\bowtie k}\ p$ (idem pour **EG**, **AF**, **AG**)
- x une horloge de l'automate : **AG**$(x \leq 5)$
- obtention de l'argent avec le GAB en moins de k unités de temps : **AG**$(U.2 \Longrightarrow$ **AF**$_{\leq k}U.3)$
- avec une horloge de formule c : **AG**$(c.(U.2 \Longrightarrow$ **AF**$(U.3 \wedge c \leq k)))$

7.2– Sémantique de TCTL

C les *horloges de* H
$L : N \to 2^{AP}$ *propositions atomiques* associées à chaque localité
ν une *valuation* de $C \to \mathbb{R}^+$
η valuation des *horloges de formules* de $C' \to \mathbb{R}^+$

Définition 18 (Sémantique de TCTL)

- $p \in AP, (s, \nu, \eta) \models p \iff p \in L(s)$
- $x, y \in C \cup C', (s, \nu, \eta) \models x - y \bowtie k \iff (\nu \cup \eta)(x - y) \bowtie k$
- $(s, \nu, \eta) \models p \vee q \iff (s, \nu, \eta) \models p$ **ou** $(s, \nu, \eta) \models q$
- $(s, \nu, \eta) \models \neg p \iff (s, \nu, \eta) \not\models p$

– $(s,\nu,\eta) \models \mathbf{E}p\,\mathcal{U}_{\bowtie k}\, q \iff$ *il existe une exécution*
$\sigma = (s,\nu,\eta) \xrightarrow[a_1]{t_1} (s_1,\nu_1,\eta_1) \xrightarrow[a_2]{t_2} \cdots \xrightarrow[a_n]{t_n} (s_n,\nu_n,\eta_n)$ *telle que*
$(s_n,\nu_n,\eta_n) \models q$ *et* $\forall i < n, \forall t \leq t_i, (s_i,\nu_i+t,\eta_i+t) \models p$ *et*
$\forall t < t_n, (s_n,\nu_n+t,\eta_n+t) \models p$ *et* $\sum t_i \bowtie k$

– $(s,\nu,\eta) \models \mathbf{A}p\,\mathcal{U}_{\bowtie k}\, q \iff$ *pour toute exécution*
$\sigma = (s,\nu,\eta) \xrightarrow[a_1]{t_1} (s_1,\nu_1,\eta_1) \xrightarrow[a_2]{t_2} \cdots \xrightarrow[a_n]{t_n} (s_n,\nu_n,\eta_n)$ *on a*
$(s_n,\nu_n,\eta_n) \models q$ *et* $\forall i < n, \forall t \leq t_i, (s_i,\nu_i+t,\eta_i+t) \models p$ *et*
$\forall t < t_n, (s_n,\nu_n+t,\eta_n+t) \models p$ *et* $\sum t_i \bowtie k$

– $(s,\nu,\eta) \models c.p \iff (s,\nu,\eta[c := 0])) \models p$

Si $(l_0,\nu_0,\eta_0) \models p$ *alors* $H \models p$. □

7.3– Autres logiques temporelles quantitatives

- μ-calcul temporisée : T_μ [11]
- sous ensembles de T_μ : L_ν, L_μ etc,
- intérêts des sous ensembles :
  - expression des *propriétés de sûreté*
  - *algorithmes* de *model-checking* plus *efficaces* que TCTL ou T_μ
  - applications : *UPPAAL* [24]

Chapitre 8 : Model-Cheking de TCTL

Sommaire

8.1– Automate des régions

– problème : *calculer* l'espace des *états atteignables* $Reach(H)$
$Reach(H) = \{(q, v),\ (q_0, \overline{0}) \xrightarrow{w} (q, v) \in [\![\ H\ ]\!]\}$
ensemble d'états *infini* !

– solution [2] : calcul du *graphe des régions*
représentation symbolique de l'espace des états

– calcul d'un automate $[H]$ à partir de H

1. *états* de $[H] = (q, R)$ où $R = [v]$ *ensemble* de valuations
$[v] =$ *classe d'équivalence associée* à v
$(q, [v]) =$ *états* satisfaisant les *mêmes propriétés de TCTL*
2. propriété de $[H]$:

$$(q, v) \xrightarrow{l}_H (q', v') \iff (q, [v]) \xrightarrow{l}_{[H]} (q', [v'])$$

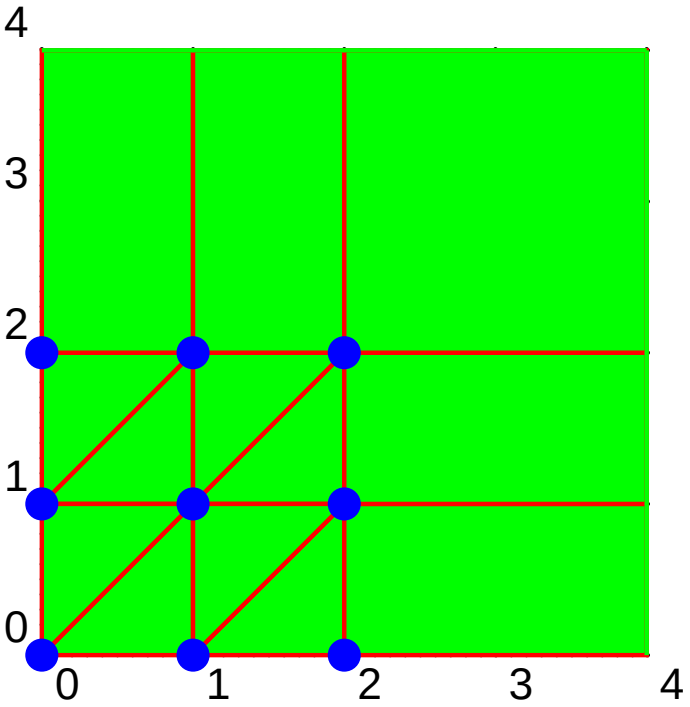
8.1.1– Partition de l'espace d'états

- C ensemble des horloges de l'automate ; $M(x)$ la *plus grande constante* comparée à x
- définition de $\approx$ sur les *valuations* de l'automate : $v \approx v'$ ssi
  - $\forall x \in C, \lfloor v(x) \rfloor = \lfloor v'(x) \rfloor$ ou $\big(v(x) > M(x)$ et $v'(x) > M(x)\big)$
  - $\forall x, x' \in C, v(x) \leq M(x) \wedge v(x') \leq M(x')$
    1. $fract(v(x)) \leq fract(v(x')) \iff fract(v'(x)) \leq fract(v'(x'))$ et
    2. $fract(v(x)) = 0 \iff fract(v'(x)) = 0$
- $\approx$ est une *relation d'équivalence* sur $\mathbb{R}_+^C$; $[v] =$ *région*
- $\mathbb{R}_+^C / \approx$ est *fini*

Théorème 5 (Equivalence pour TCTL) *$v \approx v'$ alors*

$$\forall \varphi \in TCTL, (s, v) \models \varphi \iff (s, v') \models \varphi.$$ □

Régions pour 2 variables x, y et $M(x) = M(y) = 2$



- nombre de régions borné par : $\mathcal{O}(|C|!.2^{|C|}.\Pi_{x \in C}(2.M(x)+2))$
- ici : 45 (≤ 288)
- *exo* : dénombrer exactement les régions !

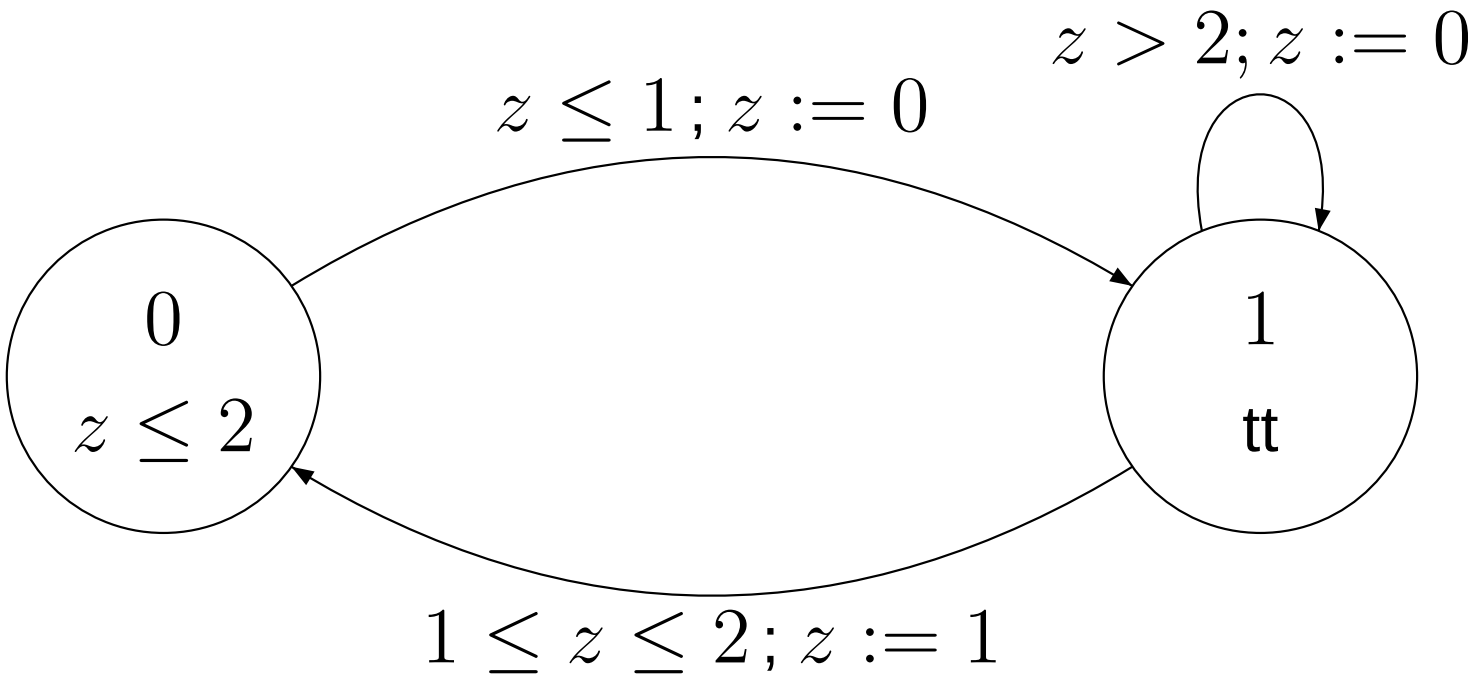
8.1.2– Graphe des régions

- *états* de $G(H)$: $N \times \mathbb{R}_+^C / \approx$
- $succ_\delta([v]) =$ prochaine région dans la *direction* $\overline{1}$ à partir de $[v]$
- transitions :

discrètes : $(l,[v]) \xrightarrow{a} (l',[v'])$ ssi $\begin{cases} (l,\gamma,a,R,l') \in E \\ \gamma([v]) = \text{tt} \\ [v'] = [v][R := 0] \\ Inv(l')[v'] = \text{tt} \end{cases}$

continues : $(l,[v]) \xrightarrow{\delta} (l,[v'])$ ssi $\begin{cases} Inv(l)([v]) = \text{tt} \\ Inv(l)([v']) = \text{tt} \\ [v'] = succ([v]) \end{cases}$

Exo : Calculer le graphe des régions de l'automate suivant :



8.1.3– Propriétés du graphe des régions

- le *graphe des régions* $G(H)$ associé à H est *fini*
- H *et* $G(H)$ sont *(timed) bisimilaires* :
  $\forall (s,\nu) \xrightarrow{e} (s',\nu') \in [\![H]\!] \quad \exists (s,[\nu]) \xrightarrow{e} (s',[\nu']) \in [\![G(H)]\!]$ et
  $\forall (s,z) \xrightarrow{e} (s',z') \in [\![G(H)]\!] \ \exists \nu \in z, \nu' \in z', \quad (s,\nu) \xrightarrow{e} (s',\nu') \in [\![H]\!]$
- conséquence : $H \models \varphi \iff G(H) \models \varphi$
- et *atteignabilité* d'un état (s,ν) est *décidable* pour les automates temporisés
- $G(H)$ *augmenté* (horloges mesurant le temps) permet le *model-checking* de *TCTL*

8.2– Principe du model-checking de TCTL

– *étiquetage* du graphe des régions $G(H)$ par les *sous formules* de φ *vraies* dans chaque état $(s, [\nu])$
suivant la forme de φ :
  – si pas modalités quantitatives $=$ idem CTL
  – pour $\varphi = \mathbf{EF}_{\leq 1} p$:
    1. ajouter une *nouvelle horloge* z
    2. pour chaque état $(s, [\nu])$ de $G(H)$:
       (a) *calculer* le *graphe des régions* associés à $(s, [\nu][z := 0])$
       (b) *chercher* un *chemin* issu de s tel que p *et* $z \leq 1$

Le *model-checking de TCTL* est en

$$\mathcal{O}(M(\phi).|\phi|.|H|.|C|!.M^{|C|})$$

Variantes des algorithmes de model-checking de TCTL

– *réduction* vers CTL [11]
$\varphi \in TCTL$

$$H \models \varphi \Longleftrightarrow G(H) \models Untimed(\varphi)$$

puis *model-checking* de *CTL* pour $G(H) \models Untimed(\varphi)$

– *algorithmes* à base de *points fixes* [11]
$\neg\varphi \rightarrow R(\varphi)$ espace d'états caractéristique de $\neg\varphi$
calculer $Reach(H) \cap R(\varphi)$

– model-checking *compositionnel* [17] (pour produit d'automates)
$(H_1|H_2|\cdots|H_n)_f \models \varphi \Longleftrightarrow (H_1|H_2|\cdots|H_{n-1})_{f'} \models \varphi/_f$

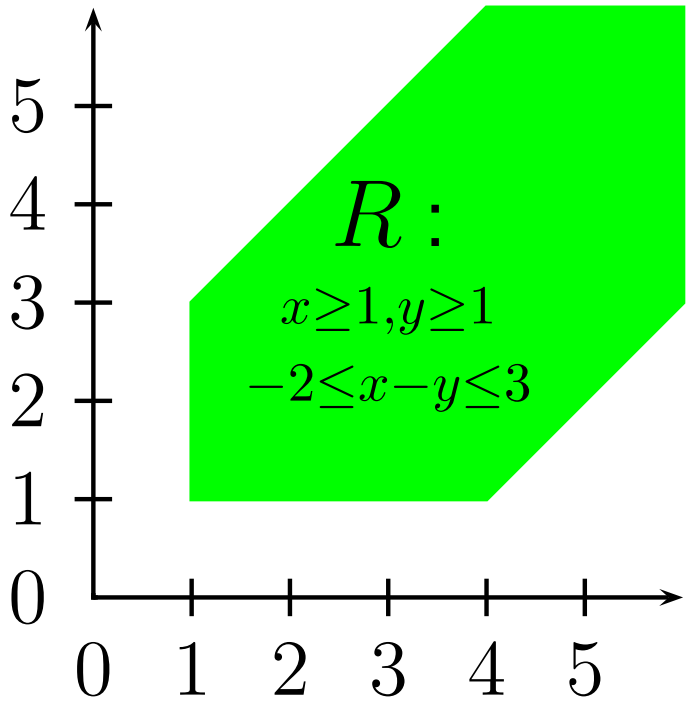
– *model-checking* efficace de *sous ensembles de TCTL* [24]

8.3– Model-checking de TCTL à la volée

- but : *éviter* de *construire* le *graphe* des régions
- solution :
  - sous ensemble de TCTL (propriétés de *sûreté*)
  - *algorithmes* efficaces : *à la volée*
  - *génération* de *contre-exemples* dans le cas où $H \not\models \varphi$
- implanté dans le model-checker *UPPAAL* [24]

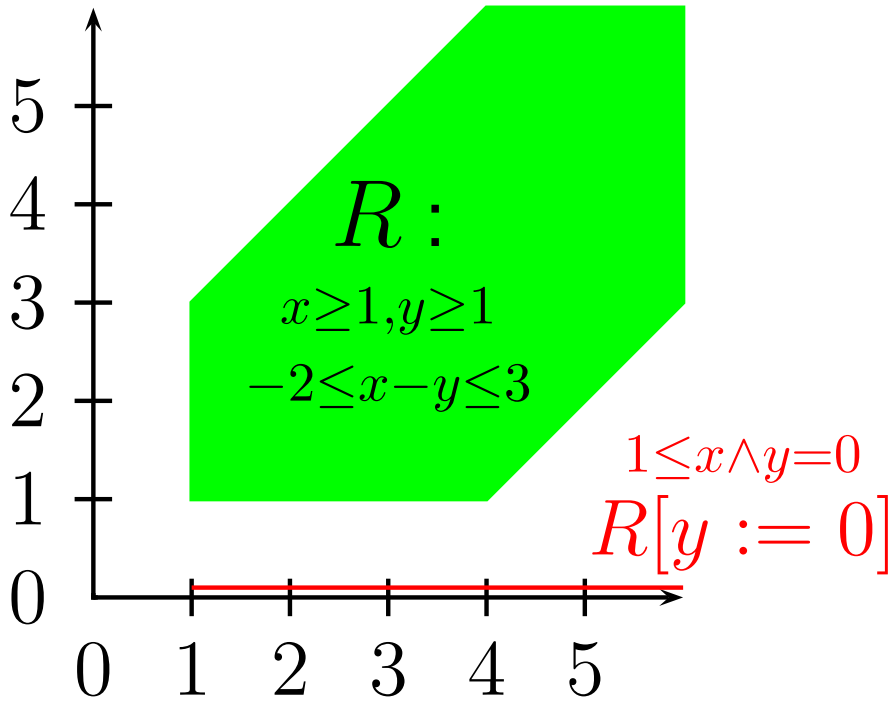
Opérations sur les régions de $\mathbb{R}^n$

- ensemble de variables V
- *projection* : $proj_v(R)$ projection de R sur $V \setminus \{v\}$



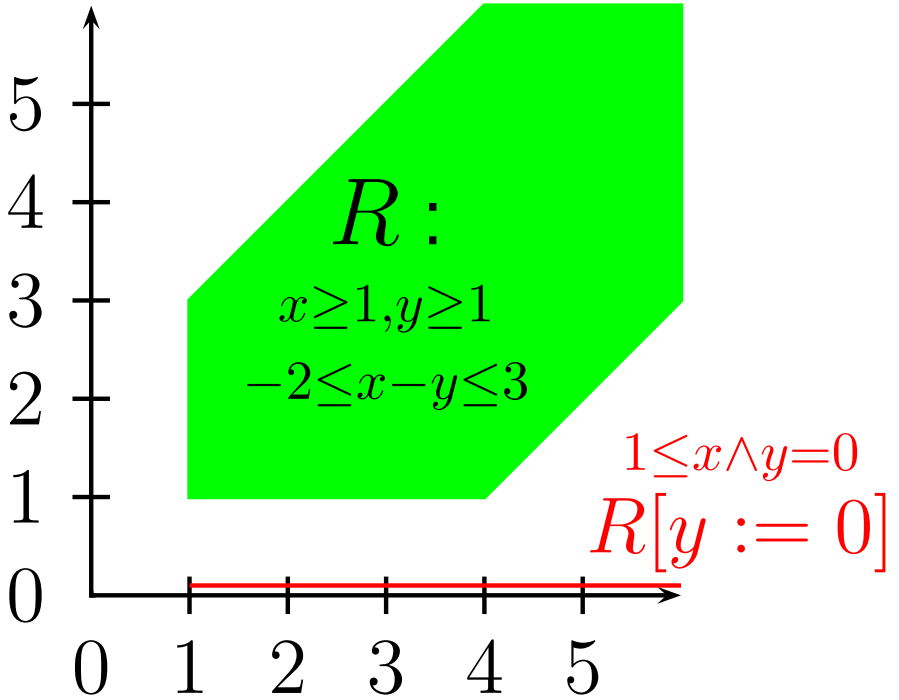
Opérations sur les régions de $\mathbb{R}^n$

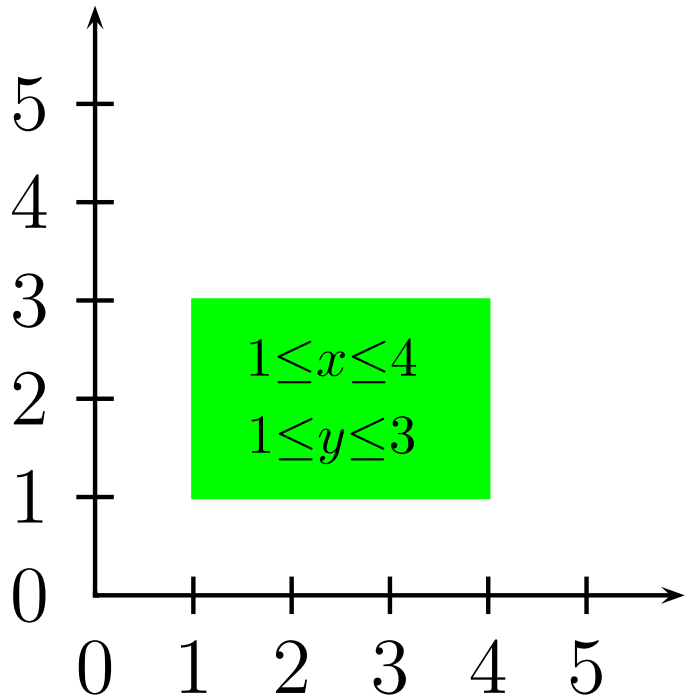
- ensemble de variables V
- *projection* : $proj_v(R)$ projection de R sur $V \setminus \{v\}$



Opérations sur les régions de $\mathbb{R}^n$

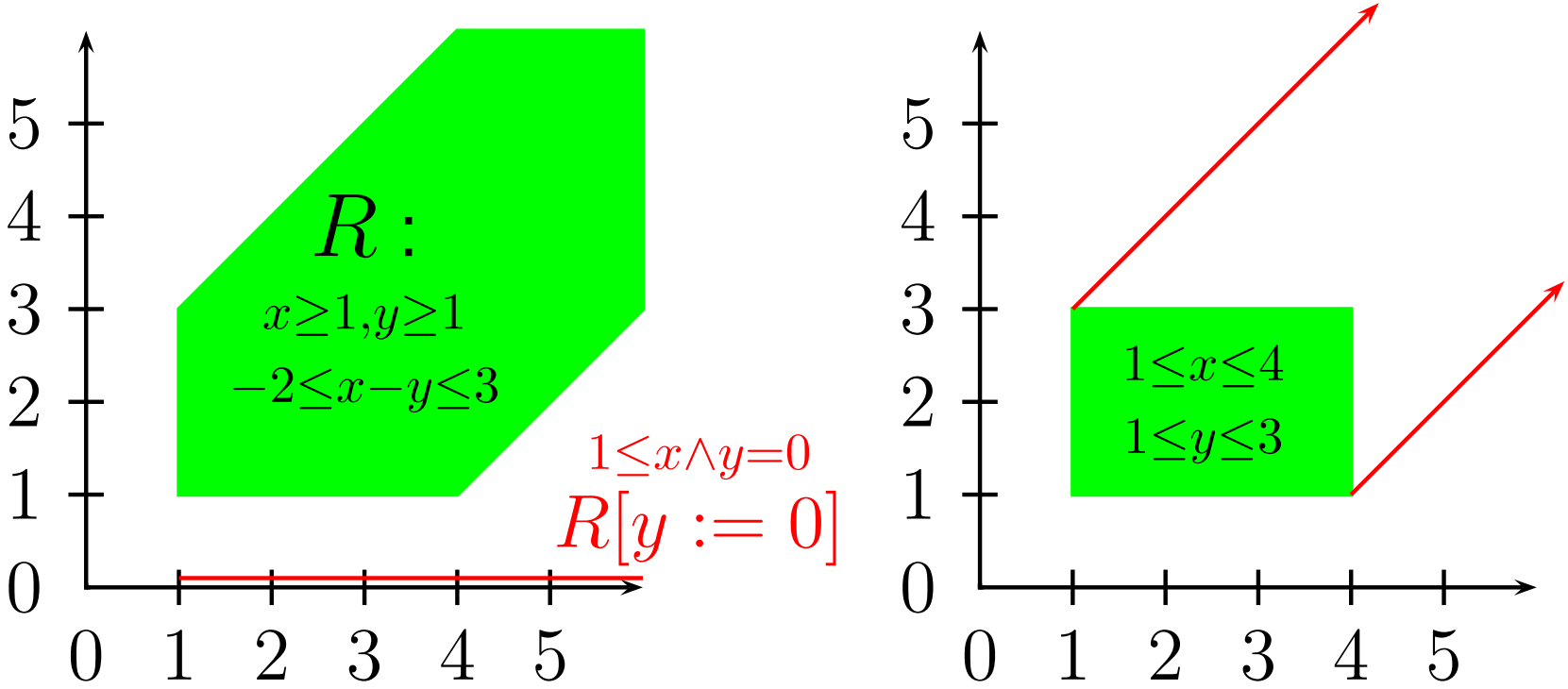
- ensemble de variables V
- *projection* : $proj_v(R)$ projection de R sur $V \setminus \{v\}$
- *futur* : $Fut(R) = \{v + t, v \in R, t \in \mathbb{R}^+\}$





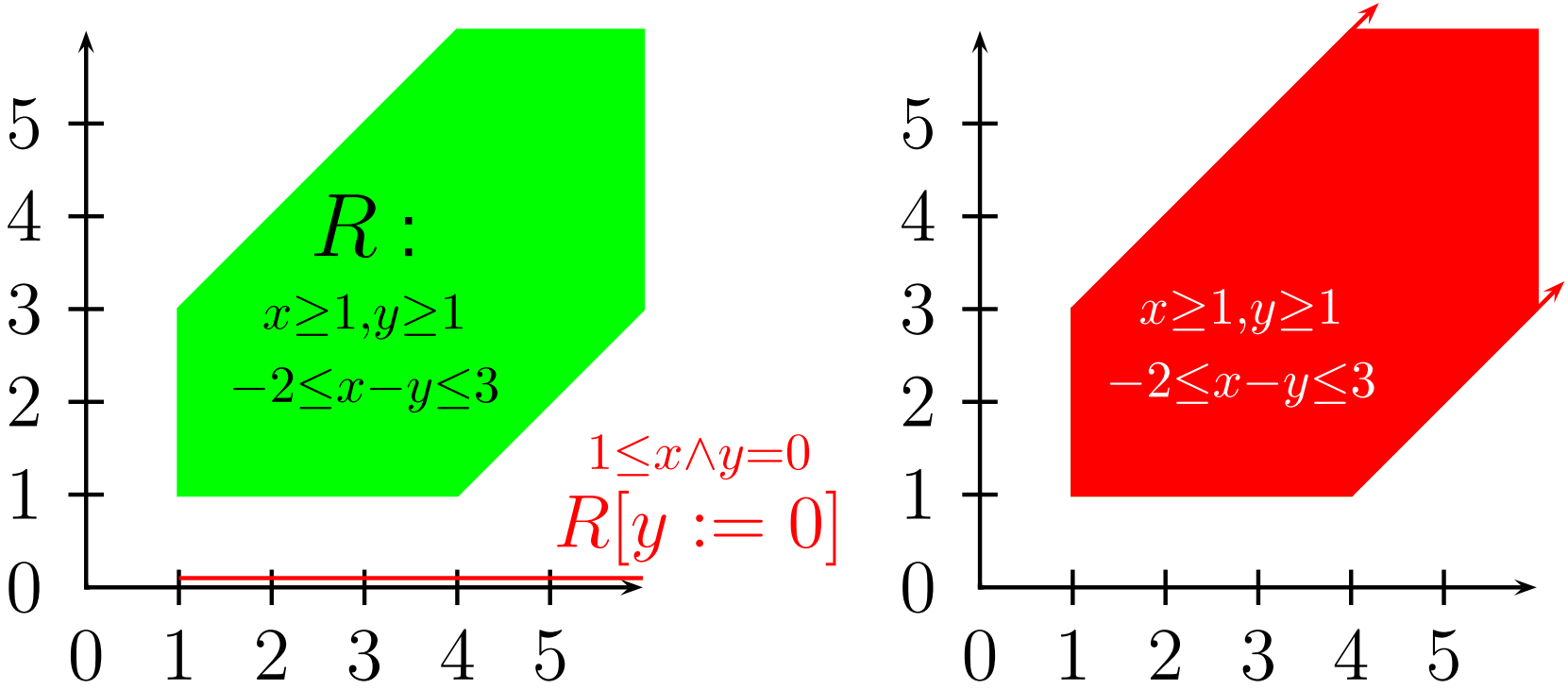
Opérations sur les régions de $\mathbb{R}^n$

- ensemble de variables V
- *projection* : $proj_v(R)$ projection de R sur $V \setminus \{v\}$
- *futur* : $Fut(R) = \{v + t, v \in R, t \in \mathbb{R}^+\}$



Opérations sur les régions de $\mathbb{R}^n$

- ensemble de variables V
- *projection* : $proj_v(R)$ projection de R sur $V \setminus \{v\}$
- *futur* : $Fut(R) = \{v + t, v \in R, t \in \mathbb{R}^+\}$



Normalisation des régions

- $k \in \mathbb{N}$: $norm_k(x - y \leq d) = \begin{cases} x - y \leq d \text{ si } d \leq k \\ x - y \leq \infty \text{ si } d > k \end{cases}$
- $norm_k(d \leq x - y) = \begin{cases} -k \leq x - y \text{ si } d \leq -k \\ d \leq x - y \text{ si } d > -k \end{cases}$
- M la *plus grande constante* entière de H : le *nombre de régions* $norm_M(r), r \in Reach(H)$ est *fini*
- pour graphe d'atteignabilité : uniquement les régions $norm_M(r)$

Forme des régions

- R, R' régions : *combinaison* booléenne de contraintes $x - y \leq k$
- $Fut(R), R \cap R', R \cup R', norm_l(R)$ sont des *combinaisons* booléennes de la forme $x - y \leq k$

Algorithme du calcul de l'espace d'états

– M la plus grande constante entière de H

1. région *initiale* : $I_0 = norm_M(Fut(\overline{0}) \cap Inv(l_0))$;
   $G = G' := \{(l_0, I_0)\}$
2. pour *toute* $(l, Y) \in G'$:
   pour *toute* transition $(l, \gamma, e, R, l') \in H$:
   (a) calculer $X = proj_R(Y \cap \gamma)$
   (b) calculer $X' = nomr_M(Fut(X) \cap Inv(l'))$
   (c) si $X' \neq \emptyset \wedge X' \not\subseteq Y, \forall (l', Y) \in G$ ajouter (l', X') à G'
3. retirer (l, Y) de G' et l'ajouter à G
4. *itérer 2.* jusqu'à $G' = \emptyset$

Théorème 6 (Terminaison [18]) *Pour H un AT, la procédure ci-dessus termine toujours. Les états atteignables sont donnés par G.* □

Algorithme de model-checking à la volée

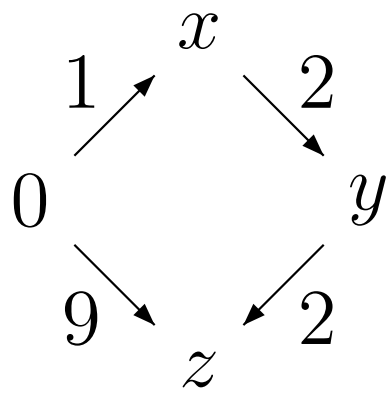
– M la plus grande constante entière de H
– φ un *invariant* ; (L, P) espace d'états associé à $\neg\varphi$

1. région *initiale* : $I_0 = norm_M(Fut(\overline{0}) \cap Inv(l_0))$;
   $G = G' := \{(l_0, I_0)\}$
2. pour *toute* $(l, Y) \in G'$:
   pour *toute* transition $(l, \gamma, e, R, l') \in H$:
   (a) calculer $X = proj_R(Y \cap \gamma)$
   (b) calculer $X' = nomr_M(Fut(X) \cap Inv(l'))$
   (c) si $X' \neq \emptyset \wedge X' \not\subseteq Y, \forall (l', Y) \in G$ ajouter (l', X') à G'
3. retirer (l, Y) de G' et l'ajouter à G
4. *itérer 2.* jusqu'à $G' = \emptyset$ ou $(L, P) \cap G' \neq \emptyset$

Implémentation du model-checking

– DBM : *Difference Bounded Matrices* [9] = *représentation* sous forme de *graphe* (ou matricielle) des régions

$$\begin{aligned} x &\leq 1 \\ y - x &\leq 2 \\ z - y &\leq 2 \\ z &\leq 9 \end{aligned}$$

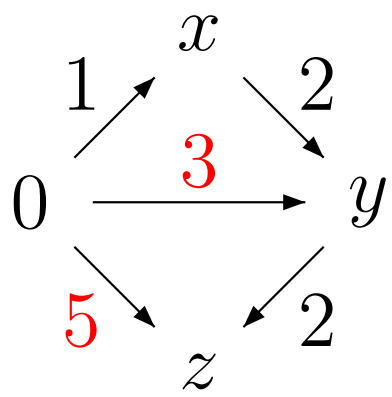


Implémentation du model-checking

– DBM : *Difference Bounded Matrices* [9] = *représentation* sous forme de *graphe* (ou matricielle) des régions

$$\begin{aligned} x &\leq 1 \\ y - x &\leq 2 \\ z - y &\leq 2 \\ z &\leq 5 \\ y &\leq 3 \end{aligned}$$

Forme canonique

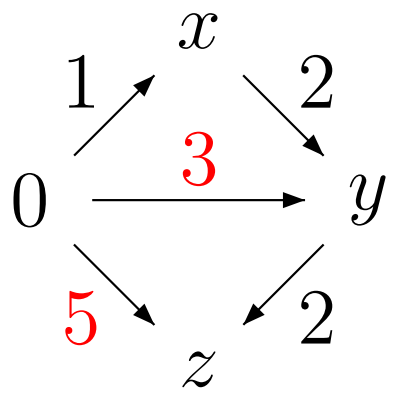


Implémentation du model-checking

– DBM : *Difference Bounded Matrices* [9] = *représentation* sous forme de *graphe* (ou matricielle) des régions

$$\begin{aligned} x &\leq 1 \\ y - x &\leq 2 \\ z - y &\leq 2 \end{aligned}$$

Forme canonique



– définition des *opérations* de $\subseteq, Fut(R), norm_k, proj_R, = \emptyset$ *sur DBMs*

– *DBM* codé en *matrice* $n+1 \times n+1$ si $x_1, x_2 \cdots x_n$ n horloges contrainte $x_i - x_j \leq k \Longrightarrow c_{ij} = k$ et $x_0 = 0$

Inclusion de 2 DBMs

– D_1, D_2 2 DBMs
– $D_1 \subseteq D_2$:
  1. *calculer* les *formes canoniques* de D_1 et D_2
  2. vérifier que chaque *poids de chaque arc* de D_1 est $\leq$ poids correspondant dans D_2

DBM vide

– D un DBM, $D = \emptyset$?
  1. *calculer* la *forme canonique* de D
  2. $D = \emptyset \iff \exists$ *cycle de poids négatif* dans graphe de D

Futur d'une région

1. *calculer* la *forme canonique*
2. *enlever* tous les *arcs* correspondant à des $x \leq k$

Projection d'une région

1. projection sur $V \setminus \{x\}$
2. *calculer* la *forme canonique*
3. *enlever* tous les *arcs de et vers* x
4. mettre x à *zéro*

Quatrième partie : Bibliographie

[1] Bowen Alpern and Fred B. Schneider. Defining liveness. *Information Processing Letters*, 21 :181–185, October 1985. 35-a, 35

[2] Rajeev Alur and David Dill. A theory of timed automata. *Theoretical Computer Science*, 2(126) :183–236, 94. 91, 104, 8.1

[3] A. Arnold. MEC : A system for constructing and analysing transition systems. In J. Sifakis, editor, *Proceedings of the International Workshop on Automatic Verification Methods for Finite State Systems*, volume 407 of *LNCS*, pages 117–132, Berlin, June 1990. Springer. 53

[4] André Arnold. *Systèmes de transitions et sémantique des processus communicants*. Masson, 1992. 1, 1.3, 2, 3, 1.5, 1.6, 53

[5] Éric Audureau, Patrice Enjalbert, and Luis Fariñas Del Cerro. *Logique temporelle – Sémantique et validation de programmes parallèles*. E.R.I. MASSON, 1990. 34, 2.1

[6] P. Bouyer, C. Dufourd, E. Fleury, and A. Petit. Are timed automata updatable ? In *Proc. 12th Int. Conf. Computer Aided Verification (CAV'2000), Chicago, IL, USA, July 2000*, volume 1855 of *Lecture Notes in Computer Science*, pages 464–479. Springer, 2000. 6.6

[7] Edmund M. Clarke, Orna Grumberg, and Dooron A. Peled. *Model-cheking*. MIT PRESS, 1999. 55, 3.2, 4.1

[8] Ouvrage collectif Coordination Philippe Schnoebelen. *Vérification de logiciles – Techniques et outils du model-checking*. Vuibert, Paris, 1999. 2, 1.5, 34, 2.2, 2.3, 55, 3.2, 4.1

[9] D. Dill. Timing assumptions and verification of finite-state concurrent systems. *Lecture Notes in Computer Science*, 407, 1989. In Proc. of Automatic Verification Methods for Finite State Systems. 124

[10] E. Allen Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B Formal Models and Semantics, chapter 16, pages 994–1072. Elsevier Science B.V., 1990. 1.5, 34, 2.1, 2.2, 2.3

[11] T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(2) :193–244, 1994. 6.3, 7.3, 118

[12] Thomas A. Henzinger. The theory of hybrid automata. In *Proceedings, 11$^{\text{th}}$ Annual IEEE Symposium on Logic in Computer Science*, pages 278–292, New Brunswick, New Jersey, 27–30 July 1996. IEEE Computer Society Press. 6.6

[13] Gerald J. Holzmann. *Design and Validation of Computer Protocols*. Prentice Hall, Englewood Cliffs, NJ, 1991. 53

[14] Gerard J. Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*, 23(5) :279–295, May 1997. Special Issue : Formal Methods in Software Practice. 53

[15] Michael R. A. Huth and Mark D. Ryan. *Logic in Computer Science : Modelling and Reasoning about Systems*. Cambridge University Press, Cambridge, England, 2000. 34, 2.2, 4.1, 78

[16] Leslie Lamport. The temporal logic of actions. *ACM Transactions On Programming Languages and Systems*, 16(3) :872–923, May 1994. 53

[17] F. Laroussinie and K. G. Larsen. CMC : A tool for compositional model-checking of real-time systems. In *Proc. IFIP Joint Int. Conf. Formal Description Techniques & Protocol Specification, Testing, and Verification (FORTE-PSTV'98)*, pages 439–456. Kluwer Academic Publishers, 1998. 118

[18] K. G. Larsen, P. Pettersson, and W. Yi. Model-checking for real-time systems. In Horst Reichel (Ed.), editor, *Proceedings of the 10th International Conference on Fundamentals of Computation Theory*, pages 62–88, Dresden, Germany, August 1995. LNCS 965. 6

[19] Jacques Loeckx and Kurt Sieber. *The Foundations of Program Verification (Second edition)*. John Wiley and Sons, New York, NY, 1987. 2

[20] Zohar Manna and Amir Pnueli. *The Temporal Logic of Reactive and Concurrent Systems, Specification*. Springer, 1991. 13, 34

[21] Ken L. Mc Millan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993. 53, 88

[22] Stephan Merz. Model checking. In F. Cassez, C. Jard, M. Ryan, and B. Rozoy, editors, *Modeling and Verification of Parallel Processes (MOVEP'00) – Summer School*, pages 51–70. CNRS/IRCCyN, Ecole Centrale de Nantes, 2000. 55, 3.2

[23] J.-F. Monin. *Comprendre les méthodes formelles. Panorama et outils logiques*. Collection technique et scientifique des télécommunications. Masson, Paris, 1996. 2

[24] Paul Pettersson and Kim G. Larsen. UPPAAL2k. *Bulletin of the European Association for Theoretical Computer Science*, 70 :40–44, February 2000. 7.3, 118, 8.3

[25] Wolfgang Thomas. *Handbook of theoretical computer science*, chapter 4, Automata on infinite objects. Elsevier Science, 1990. 3.1

[26] Pierre Wolper. *Approche logique de l'intelligence artificielle Logique Temporelle*, volume 2 – De la logique modale à la logique des bases de données, chapter 4, Logique Temporelle, pages 179–. DUNOD, 1990. 34, 2.1