

Introduction à la vérification automatique des systèmes concurrents (2)

Anca Muscholl et Marc Zeitoun
LIAFA, Univ. Paris 7 & CNRS
EJC Marne-La-Vallée, 04/2003

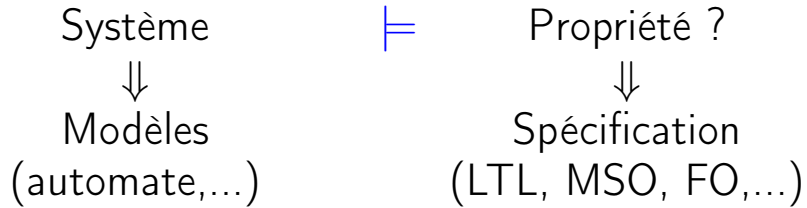
Dans cet exposé...

- Vérification des systèmes concurrents : outils et problèmes.
- Un modèle concurrent : les Message Sequence Charts (MSC)
- Traces de Mazurkiewicz et logiques temporelles
 - Difficulté des logiques globales
 - Vérification des logiques locales.

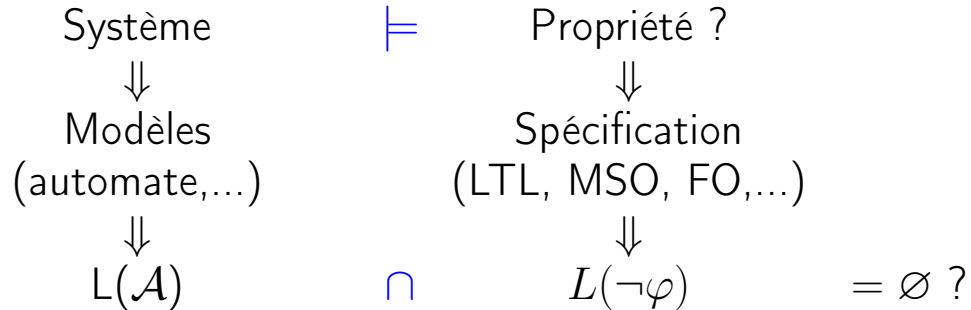
Systemes séquentiels (rappels)

Systeme \models Propriété ?

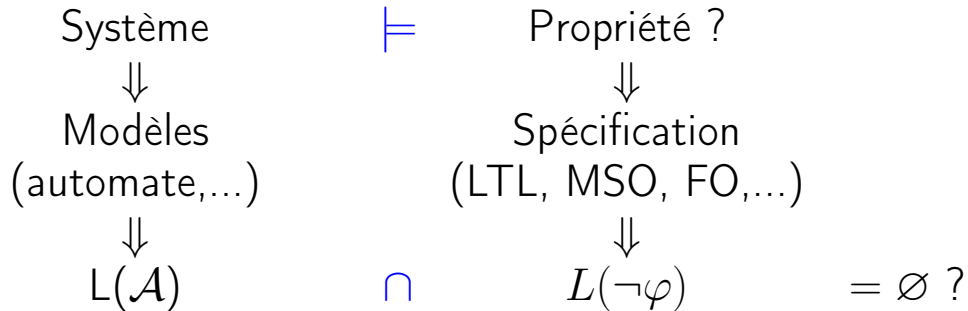
Systemes séquentiels (rappels)



Systemes séquentiels (rappels)



Systemes séquentiels (rappels)



- Souvent, le model-checking se ramène au **test du vide sur un automate**
- Souvent, les modèles peuvent être codés par les spécifications
- \Rightarrow Problèmes intéressants sur les langages de spécification
 - Expressivité (FO)
 - Problème du vide (satisfaisabilité d'une spécification).

Systèmes concurrents

- Explosion du **nombre d'états** lors de la mise en parallèle de processus.
- L'explosion combinatoire peut être artificielle.
- Difficulté de **modélisation** des systèmes.
- \Rightarrow Intérêt de travailler **directement** sur des modèles concurrents.

Systèmes concurrents

- Explosion du **nombre d'états** lors de la mise en parallèle de processus.
- L'explosion combinatoire peut être artificielle.
- Difficulté de **modélisation** des systèmes.
- \Rightarrow Intérêt de travailler **directement** sur des modèles concurrents.

Monde séquentiel	Monde concurrent
Mots	Ordres partiels étiquetés
Automates	Automates asynchrones (mémoire partagée) Automates communicants (canaux) Réseaux de Petri HMSCs...
Logiques FO, MSO, LTL	FO, MSO, ?

Traces de Mazurkiewicz

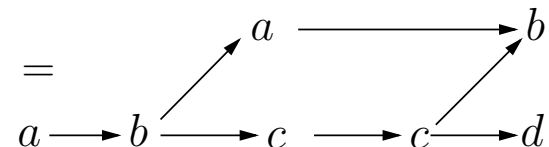
$D \subseteq \Sigma \times \Sigma$: dépendance **réflexive**, **symétrique**. $I = \Sigma \times \Sigma \setminus D$.
Trace = ordre partiel étiqueté (V, \leq, ℓ) , où \leq est **contraint** par ℓ .

$$\begin{aligned} \ell(e) D \ell(f) &\implies e \leq f \text{ ou } f \leq e \\ e \triangleleft f &\implies \ell(e) D \ell(f) \end{aligned}$$

Traces de Mazurkiewicz

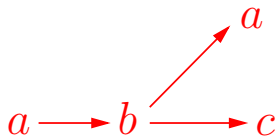
$D \subseteq \Sigma \times \Sigma$: dépendance **réflexive**, **symétrique**. $I = \Sigma \times \Sigma \setminus D$.
Trace = ordre partiel étiqueté (V, \leq, ℓ) , où \leq est **contraint** par ℓ .

$$\begin{aligned} \ell(e) D \ell(f) &\implies e \leq f \text{ ou } f \leq e \\ e < f &\implies \ell(e) D \ell(f) \end{aligned}$$

$$(\Sigma, D) = a \text{ --- } b \text{ --- } c \text{ --- } d \quad (V, \leq, \ell) =$$


Linéarisation = mot respectant l'ordre \leq : $abca**cb**d$, mais pas $abcab**cb**d$.
Une linéarisation donnée correspond à **une et une seule** trace

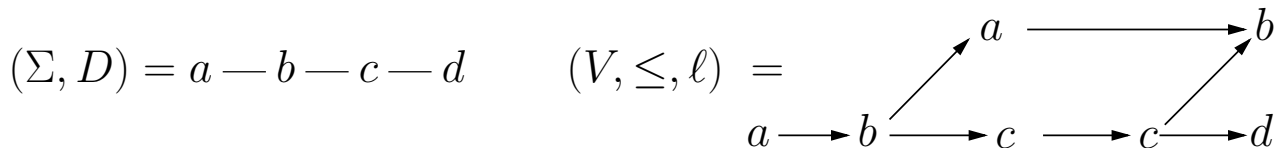
$\mathbb{M}(\Sigma, D)$ monoïde de traces.



Traces de Mazurkiewicz

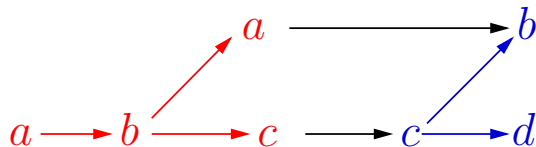
$D \subseteq \Sigma \times \Sigma$: dépendance **réflexive**, **symétrique**. $I = \Sigma \times \Sigma \setminus D$.
 Trace = ordre partiel étiqueté (V, \leq, ℓ) , où \leq est **contraint** par ℓ .

$$\begin{aligned} \ell(e) D \ell(f) &\implies e \leq f \text{ ou } f \leq e \\ e < f &\implies \ell(e) D \ell(f) \end{aligned}$$



Linéarisation = mot respectant l'ordre \leq : $abca**cb**d$, mais pas $abcab**cb**d$.
 Une linéarisation donnée correspond à **une et une seule** trace

$\mathbb{M}(\Sigma, D)$ monoïde de traces.



Langages de traces : reconnaissable vs. rationnel

- $L \subseteq \mathbb{M}(\Sigma, D)$ $\text{Lin}(L)$ = ensemble des linéarisations de traces de L
- L est rationnel si $\exists R \subseteq \text{Rat}(\Sigma^*) \cap \text{Lin}(L)$ tq $t \in L \Rightarrow \text{Lin}(t) \cap R \neq \emptyset$
- L est reconnaissable si $\text{Lin}(L)$ est rationnel sur Σ^* .
- Reconnaissable \Rightarrow rationnel. $a \mid b, (ab)^* \in \text{Rat} \setminus \text{Rec}$.

Langages de traces : reconnaissable vs. rationnel

- $L \subseteq \mathbb{M}(\Sigma, D)$ $\text{Lin}(L)$ = ensemble des linéarisations de traces de L
- L est rationnel si $\exists R \subseteq \text{Rat}(\Sigma^*) \cap \text{Lin}(L)$ tq $t \in L \Rightarrow \text{Lin}(t) \cap R \neq \emptyset$
- L est reconnaissable si $\text{Lin}(L)$ est rationnel sur Σ^* .
- Reconnaissable \Rightarrow rationnel. $a \mid b, (ab)^* \in \text{Rat} \setminus \text{Rec}$.

Théorème[O85] Si un automate \mathcal{A} sur (Σ, D) a ses boucles étiquetées par des traces connexes, alors $\text{Lin}(L(\mathcal{A}))$ est rationnel. Autrement dit, $L(\mathcal{A}) \subseteq \mathbb{M}(\Sigma, D)$ est reconnaissable.

Si un langage $L \subseteq \mathbb{M}(\Sigma, D)$ est reconnaissable, alors il existe un tel automate qui reconnaît $\text{Lin}(L)$.

Langages de traces : reconnaissable vs. rationnel

Théorème Soient $K, L \subseteq \mathbb{M}(\Sigma, D)$ rationnels, on ne peut pas décider si

- $L \cap K = \emptyset$, $L \subseteq K$ (model-checking),
- L est reconnaissable,
- $L = \mathbb{M}(\Sigma, D)$.
- ...

Langages de traces : reconnaissable vs. rationnel

Théorème Soient $K, L \subseteq \mathbb{M}(\Sigma, D)$ rationnels, on ne peut pas décider si

- $L \cap K = \emptyset$, $L \subseteq K$ (model-checking),
- L est reconnaissable,
- $L = \mathbb{M}(\Sigma, D)$.
- ...

Dém. PCP $\{(u_1, v_1), \dots, (u_k, v_k)\}$ sur A^* .

Soit $B = b_1, \dots, b_k$ tel que $A \times B \subseteq I$

Soit $L[u] = (u_1 b_1 + \dots + u_k b_k)^+ \subseteq A^* \times B^*$.

$$= \{(u_{i_1} \dots u_{i_p}, b_{i_1} \dots b_{i_p})\}^+$$

PCP(u, v) se réduit à $L[u] \cap L[v] = \emptyset$.

Message Sequence Charts

- MSCs et High-level Message sequence charts (HMSCs)

Message Sequence Charts

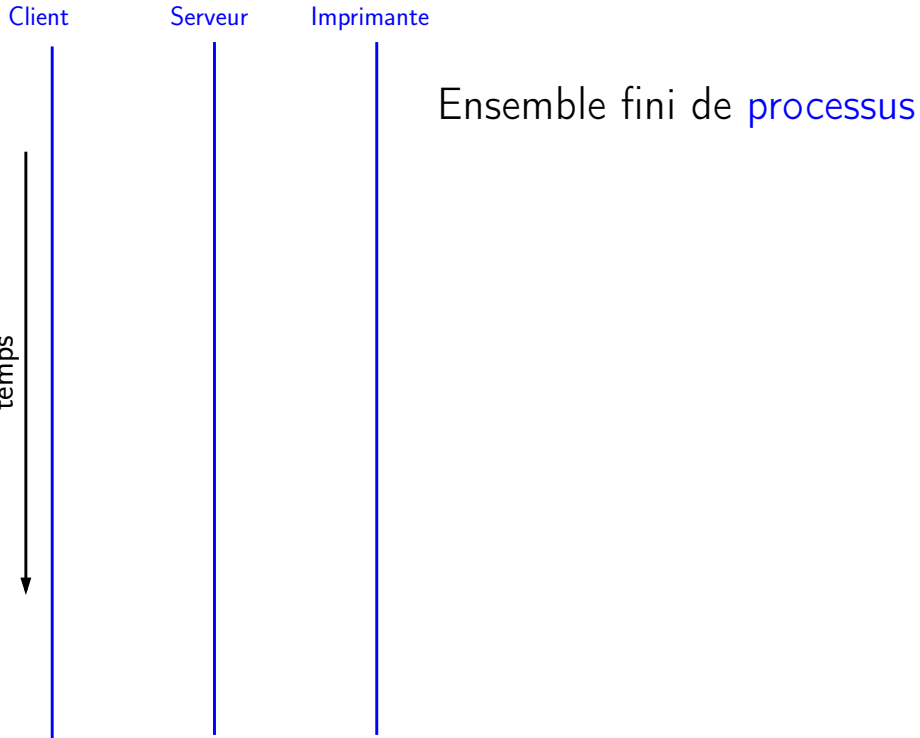
- MSCs et High-level Message sequence charts (HMSCs)

Deux problèmes sur des modèles concurrents à états infinis...

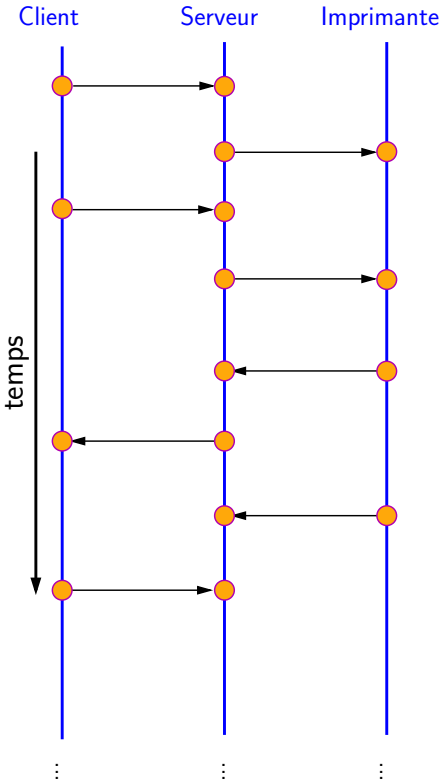
- **Implémentabilité** de langages de HMSCs par des CFMs
- **Model-checking** de HMSCs par des HMSCs

Des réponses pour des **sous-classes de HMSCs à espace d'états infini**

Message sequence charts (MSC)



Message sequence charts (MSC)



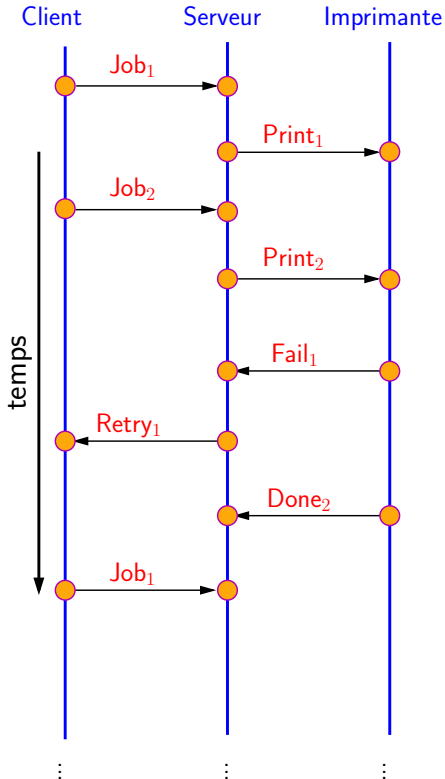
Ensemble fini de processus

Canaux point à point, fiables, FIFO

Types d'événements : envoi $k!_{m,j}$ / réception $k?_{m,j}$

Chaque événement est situé sur un processus

Message sequence charts (MSC)



Ensemble fini de processus

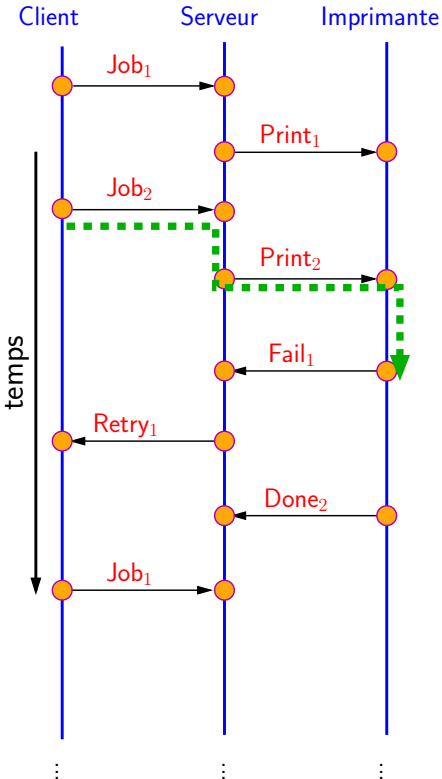
Canaux point à point, fiables, FIFO

Types d'événements : envoi $k!_{m,j}$ / réception $k?_{m,j}$

Chaque événement est situé sur un processus

Contenus de messages

Message sequence charts (MSC)



Ensemble fini de processus

Canaux point à point, fiables, FIFO

Types d'événements : envoi $k!_{m,j}$ /réception $k?_{m,j}$

Chaque événement est situé sur un processus

Contenus de messages

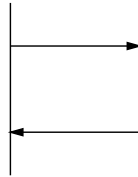
Ordre partiel sur les événements (Ordre visuel \leq)

- Ordre total \leq_p sur les événements de p ,
- 'envoi' précède 'réception' correspondante
→ relation acyclique

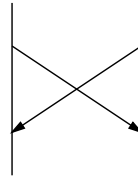
Qu'est-ce qu'un Message sequence chart (MSC) ?

- Description **graphique** d'une exécution d'un système multi-processus
- Le temps est discret, l'accent est mis sur les **communications**.
- Utilisé lors des phases de conception des protocoles.
 - définition d'exigences (complétude)
 - pas de mauvais comportements (consistance)
- **Standardisé** ITU Z120 (1996, 2000), fait partie d'UML, outils disponibles (ObjectGeode, RationalRose, Slim/SoFat)
- Applications **pratiques** (génération automatique de squelette de code)

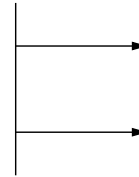
Exemples de MSCs



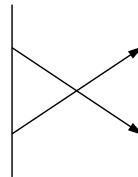
OK



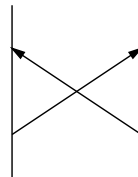
OK



OK



PAS FIFO



NON ! (CYCLIQUE)

Concaténation asynchrone de MSCs

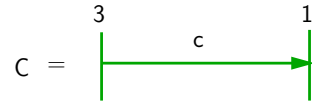
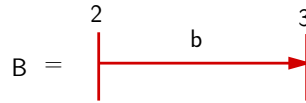
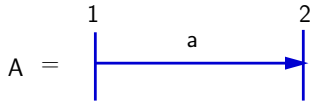
Concaténation processus par processus, **SANS** synchronisation

$A \cdot B$: “coller B sous A”

Concaténation asynchrone de MSCs

Concaténation processus par processus, **SANS** synchronisation

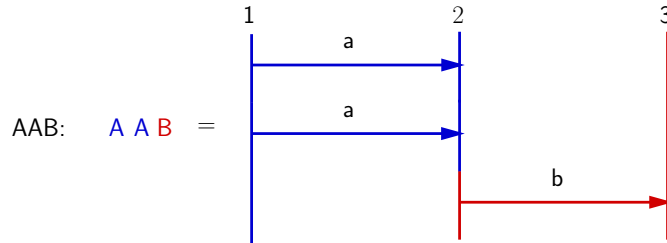
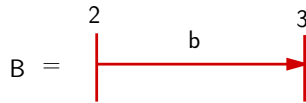
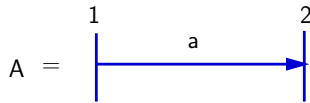
$A \cdot B$: "coller B sous A"



Concaténation asynchrone de MSCs

Concaténation processus par processus, **SANS** synchronisation

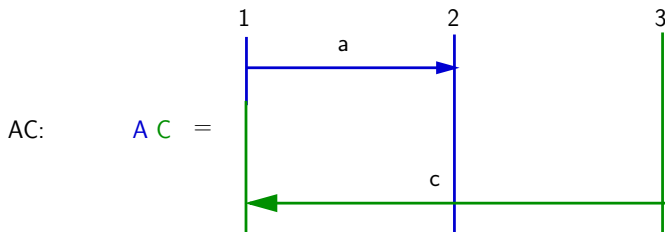
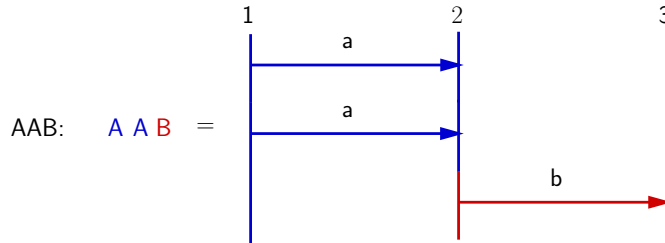
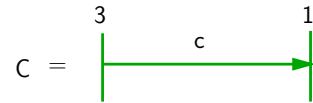
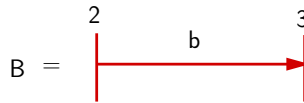
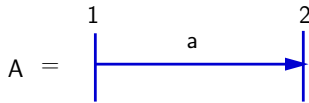
$A \cdot B$: "coller B sous A"



Concaténation asynchrone de MSCs

Concaténation processus par processus, **SANS** synchronisation

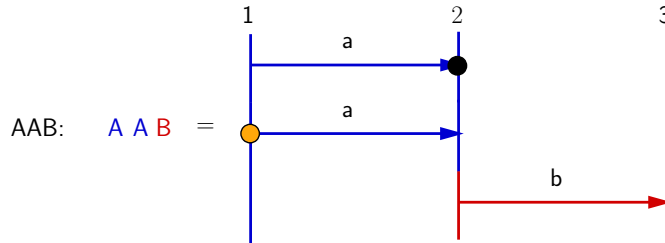
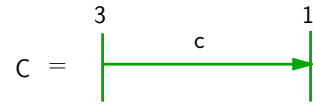
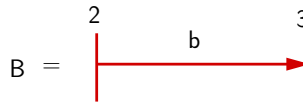
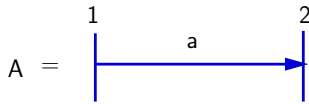
A · B : "coller B sous A"



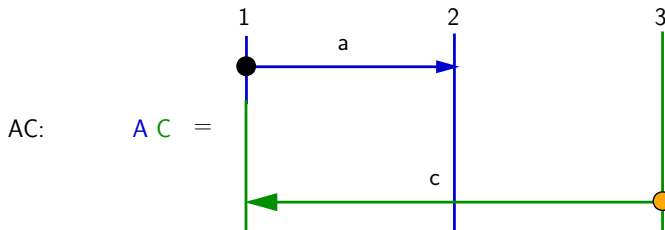
Concaténation asynchrone de MSCs

Concaténation processus par processus, **SANS** synchronisation

$A \cdot B$: "coller B sous A"



● et ● sont **concurrents**

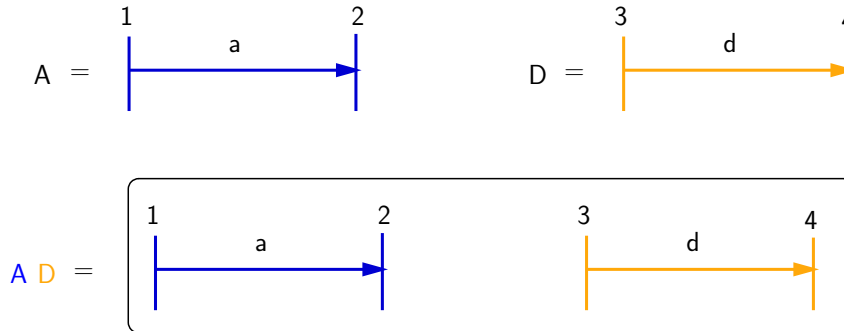


Dans une linéarisation de $<$,

● peut être exécuté **avant** ●
même s'il apparaît dans un
facteur ultérieur

Concaténation asynchrone et dépendance

Concaténation « parallèle » de MSCs sur des processus disjointes



$P(M)$ = ensemble des processus communicants dans M

$$M_1 \text{ D } M_2 \text{ ssi } P(M_1) \cap P(M_2) \neq \emptyset$$

Remarque

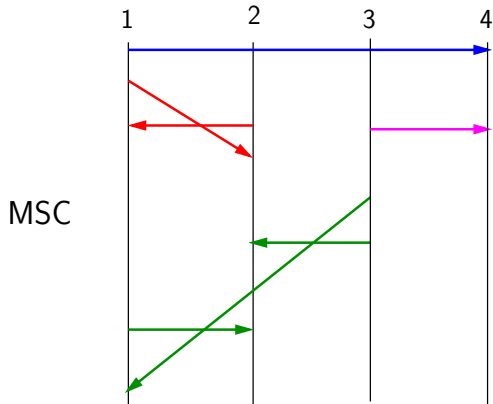
$$M_1 \text{ I } M_2 \implies M_1 M_2 = M_2 M_1$$

Traces et MSCs

Un MSC M est *atomique* si $M = AB$ implique $M = A$ ou $M = B$.

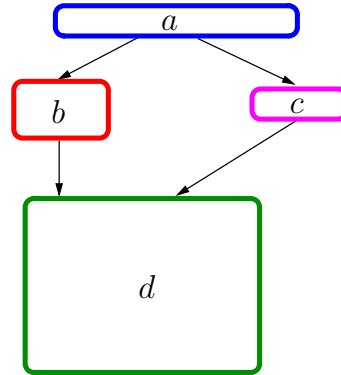
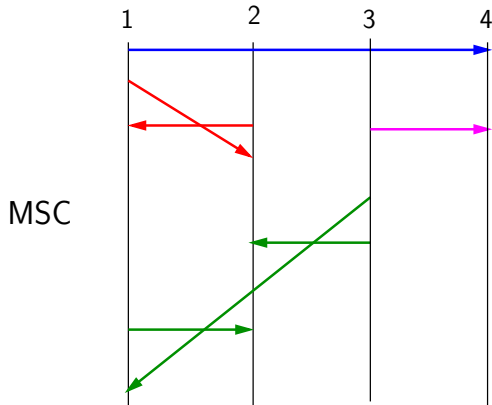
Traces et MSCs

Un MSC M est **atomique** si $M = AB$ implique $M = A$ ou $M = B$.

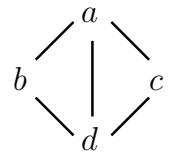


Traces et MSCs

Un MSC M est **atomique** si $M = AB$ implique $M = A$ ou $M = B$.



trace sur



High-level MSC (HMSC)

HMSC = système de transitions $G = \langle V, R, v^0, v^f, \lambda \rangle$, où

- V : ensemble de nœuds,
- $R \subseteq V \times V$: ensemble de transitions,
- nœud initial v^0
- nœud terminal v^f
- chaque nœud v est étiqueté par un MSC fini $\lambda(v)$.

Exécution de G : étiquetage $\lambda(v_0)\lambda(v_1)\cdots\lambda(v_k)$ d'un chemin réussi
 $v^0 = v_0 \rightarrow v_1 \rightarrow \cdots \rightarrow v_k = v^f$

HMSCs régulier [HMKT00]

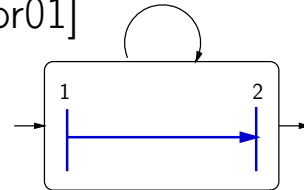
G HMSC $\text{Lin}(G) = \{\text{linéarisations des exécutions de } G\} \subseteq \text{Events}^*$

Définition G est un HMSC régulier ssi $\text{Lin}(G)$ is régulier

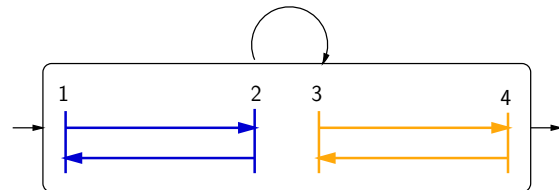
Remarque G régulier \implies G à canaux bornés

Causes caractéristiques de non régularité [Mor01]

- Canaux non bornés



- Pas reconnaissable dans \mathcal{M}



But : vérifier et implémenter des HMSCs éventuellement non réguliers.

HMSCs coopératifs

Un MSC M est **atomique** si $M = AB$ implique $M = A$ ou $M = B$.

Un MSC M est **lié** si $M = AB$ et $A \perp B$ implique $M = A$ ou $M = B$.

Hypothèse : les nœuds des HMSCs considérés sont **liés**.

Un HMSC $G = \langle V, R, v^0, v^f, \lambda \rangle$ est

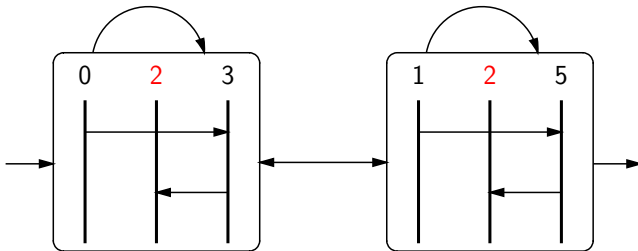
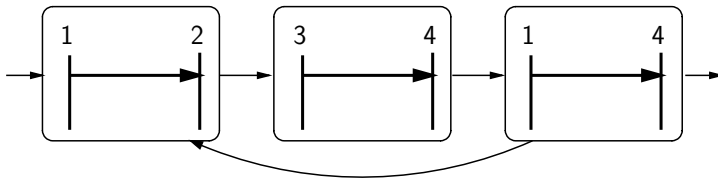
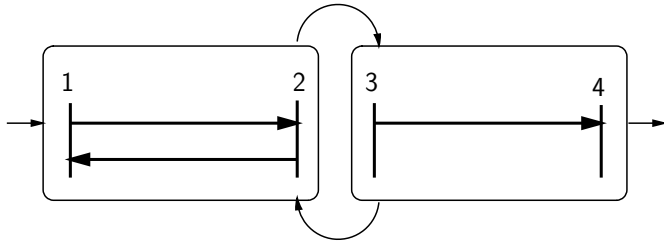
- **Localement coopératif** si pour toute arête $v \rightarrow w$, $\lambda(v)\lambda(w)$ est lié
- **Globalement coopératif** si toute **boucle** est étiquetée par un MSC lié

Tester qu'un HMSC est

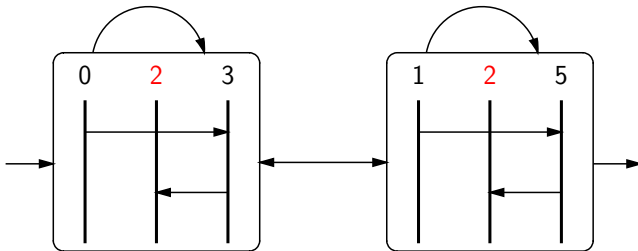
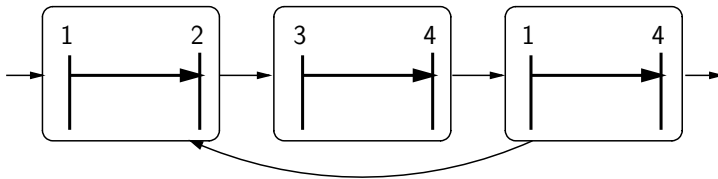
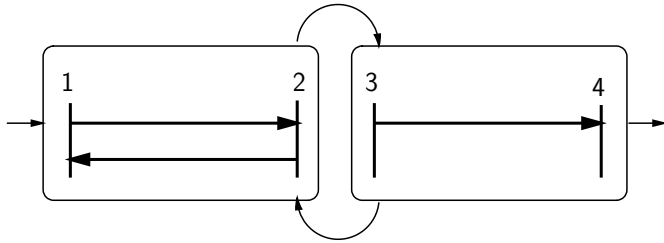
- globalement coopératif est co-NP-complet [MP99]
- localement coopératif est polynomial

Ces HMSCs décrivent des systèmes à nombre **infini d'états**.

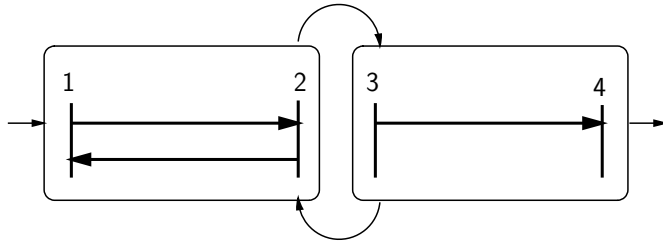
HMSCs coopératifs : exemples



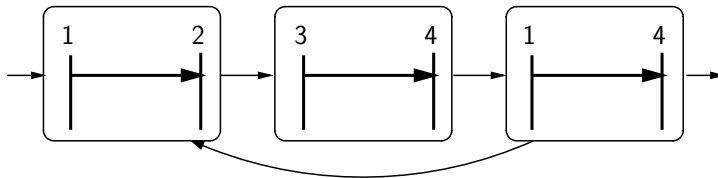
HMSCs coopératifs : exemples



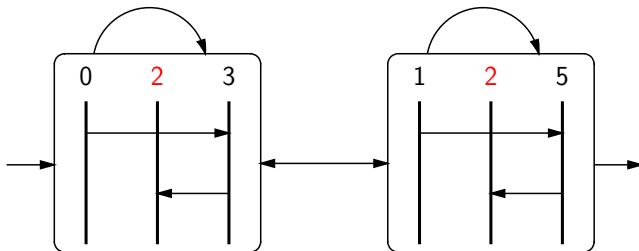
HMSCs coopératifs : exemples



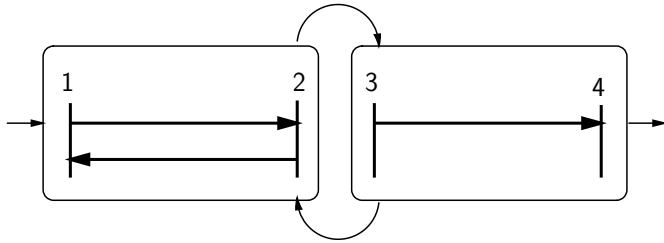
non coopératif



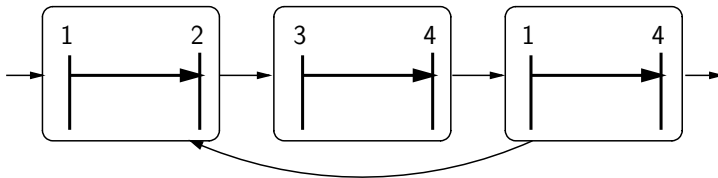
non localement coopératif
globalement coopératif



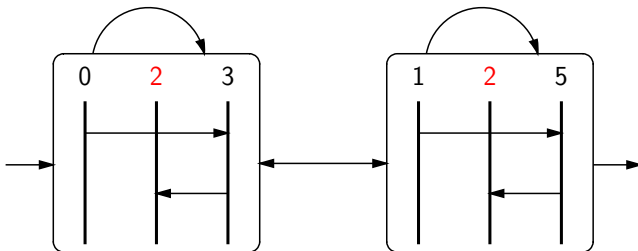
HMSCs coopératifs : exemples



non coopératif



non localement coopératif
globalement coopératif



localement coopératif

Problème d'implémentabilité de HMSC

Distribution du contrôle

Problème d'implémentabilité

Modèle d'implementation : Automates communicants (CFM)
avec **contrôle distribué**

Un CFM **C** génère une suite d'événements $L(\mathbf{C})$

Un CFM **C** est une **implémentation** d'un HMSC **G** si $L(\mathbf{C}) = \text{Lin}(\mathbf{G})$

Problème d'implémentabilité

Modèle d'implémentation : Automates communicants (CFM)
avec **contrôle distribué**

Un CFM **C** génère une suite d'événements $L(\mathbf{C})$

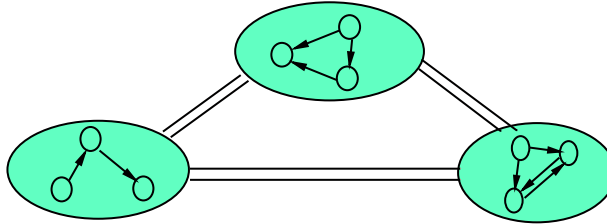
Un CFM **C** est une **implémentation** d'un HMSC **G** si $L(\mathbf{C}) = \text{Lin}(\mathbf{G})$

Problème d'implémentabilité

- Donnée : Un HMSC **G**.
- Question : Existe-t-il une implémentation pour **G**?
Si oui, l'implémentation est-elle calculable ?

▷ **Distribuer le contrôle centralisé d'un HMSC sur ses processus**

Automates communicants (CFM)

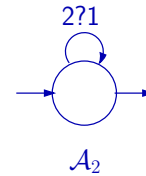
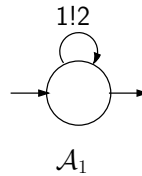


$\langle \mathcal{A}_k, \mathcal{B}_{i,j}, \mathcal{C} \rangle_{1 \leq i,j,k \leq n} =$ automates finis communiquant via canaux FIFO

- $\mathcal{A}_k = (\Sigma_k, Q_k, i_k, \rightarrow_k, F_k)$ automate représentant le processus k
- Canaux **fiables**, **FIFO** point à point $\mathcal{B}_{i,j}$ de i à j $i \neq j$
- Alphabet fini \mathcal{C} de **contenus de messages**.
- Alphabets d'événements $\Sigma_k = \bigcup_{j \neq k, m \in \mathcal{C}} \{k!_m j, k?_m j\}$ $1 \leq k \leq n$

Automates communicants (CFM)

Exemple : $C = \langle \mathcal{A}_1, \mathcal{A}_2 \rangle$

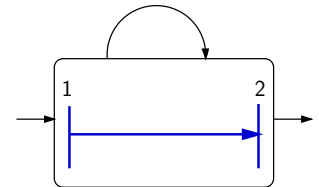


Calcul

- commence dans l'état $s^0 = (i_1, \dots, i_n)$, finit en $\prod F_k$, canaux vides.
- une transition par $k!_{\mathbf{m}}j$ ajoute le message \mathbf{m} en fin de canal $k \rightarrow j$
- une transition par $k?_{\mathbf{m}}j$ supprime le message \mathbf{m} du buffer $j \rightarrow k$

$L(C) =$ ensemble des suites d'étiquettes de transitions le long de calculs.

Le CFM de l'exemple est une implémentation de



Automates communicants et machines de Turing

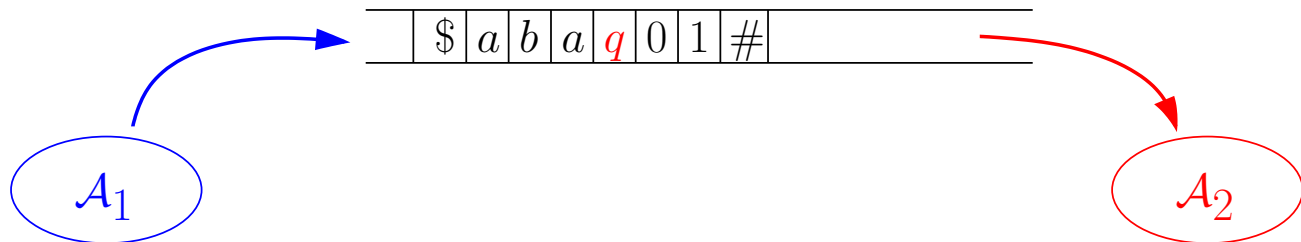
Machine de Turing $M \rightsquigarrow$ CFM \mathcal{A} simulant M . Ex. $(q, 0) \rightarrow (p, A, \text{left})$

\mathcal{A}_1

\mathcal{A}_2

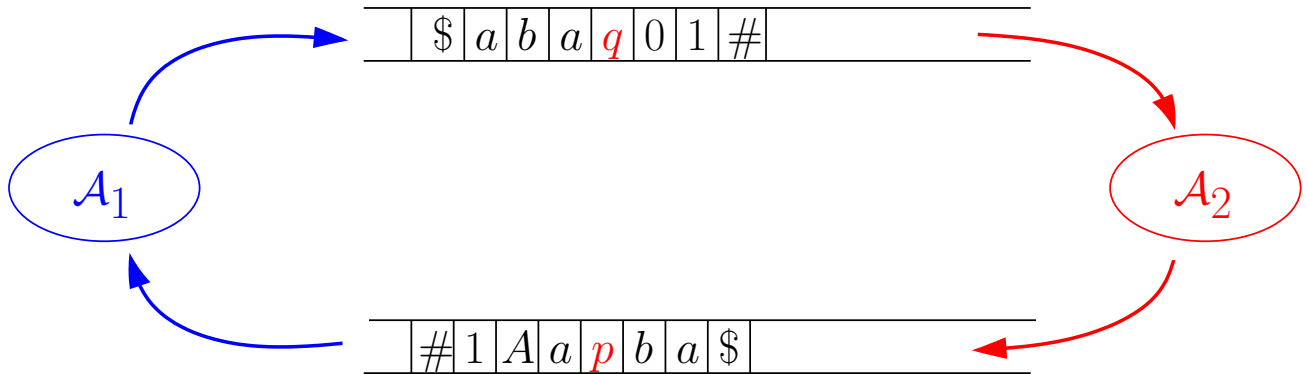
Automates communicants et machines de Turing

Machine de Turing $M \rightsquigarrow$ CFM \mathcal{A} simulant M . Ex. $(q, 0) \rightarrow (p, A, \text{left})$



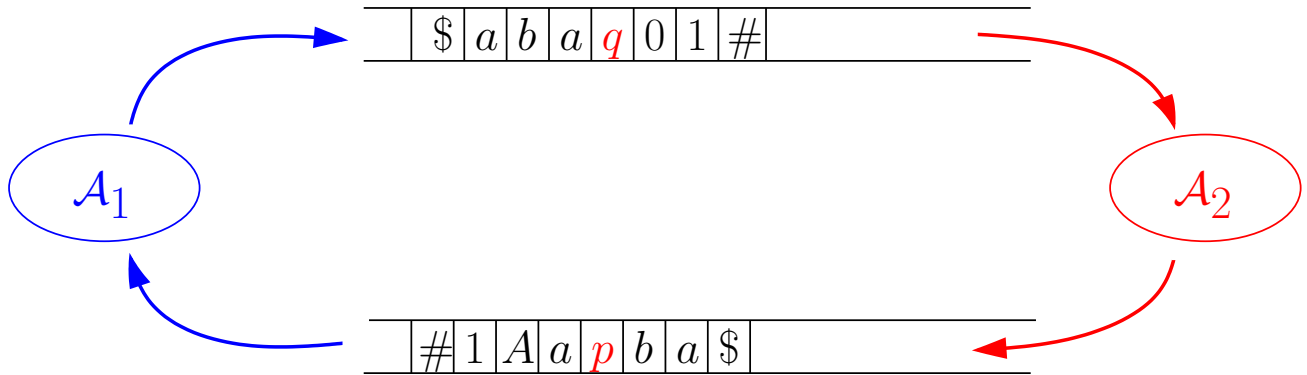
Automates communicants et machines de Turing

Machine de Turing $M \rightsquigarrow$ CFM \mathcal{A} simulant M . Ex. $(q, 0) \rightarrow (p, A, \text{left})$



Automates communicants et machines de Turing

Machine de Turing $M \rightsquigarrow$ CFM \mathcal{A} simulant M . Ex. $(q, 0) \rightarrow (p, A, \text{left})$



Étant donné un CFM, on ne peut pas décider si

- un message envoyé sera reçu,
- une configuration est accessible,
- une configuration peut atteindre un état final.
- . . .

CFM standard [AEY00]

Pour un HMSC G , le CFM standard de G est obtenu ainsi

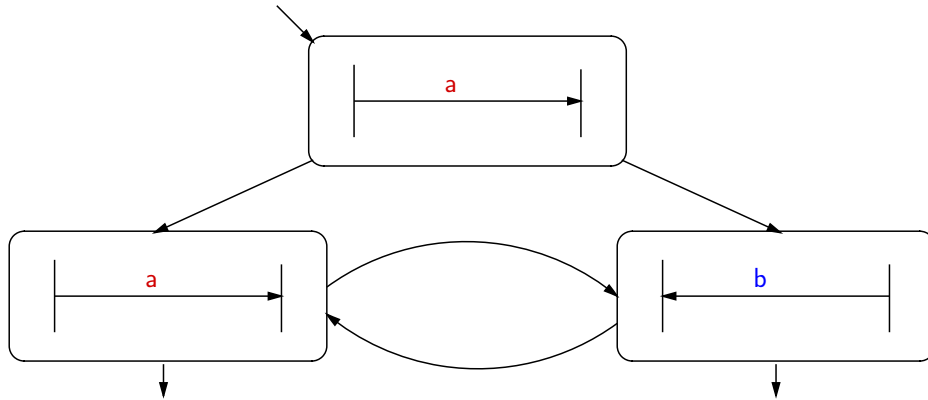
\mathcal{A}_k reconnaît la projection de $L(G)$ sur le processus k

- Le CFM standard de G reconnaît **au moins** le langage de G
- Implémentabilité \iff implémentabilité par le CFM standard [Mo02]
- L'implémentabilité des HMSCs réguliers est **indécidable** [AEY00]

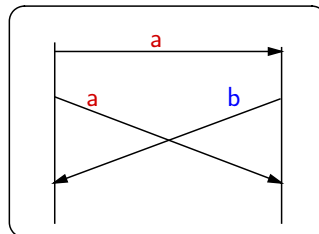
Note L'implémentabilité est décidable pour des CFMs non-FIFO [Mo02]

CFM standard : scénarios impliqués

Le CFM standard de

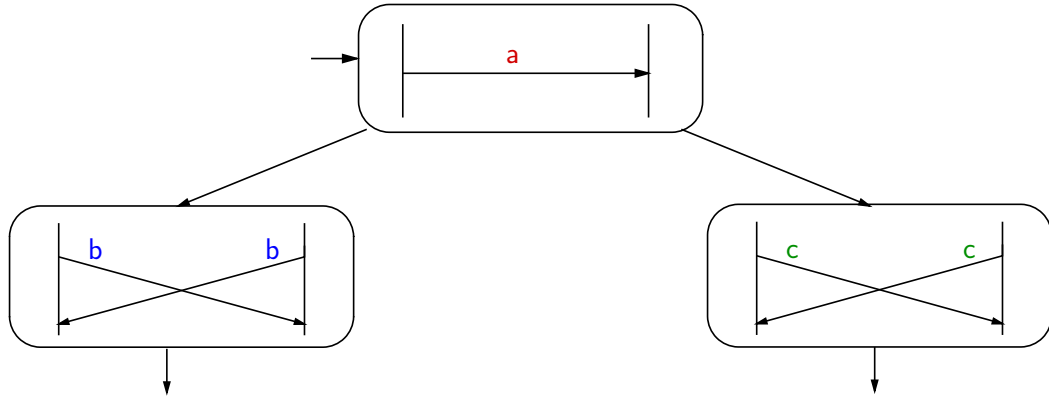


produit un nouveau MSC (non désiré) :

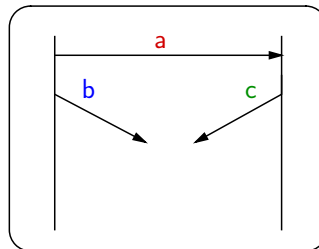


CFM standard : blocages

Standard CFM of



produit un blocage



Implémentation des HMSCs: résultats

- On ne peut pas décider si un HMSC, même régulier, est implémentable

Mais...

Implémentation des HMSCs: résultats

- On ne peut pas décider si un HMSC, même régulier, est implémentable

Mais...

- En permettant aux messages de l'implémentation de contenir de l'information supplémentaire (bornée), on peut implémenter
 - ▷ les HMSCs globalement coopératifs
 - ▷ les HMSCs localement coopératifs, sans blocage
- D'autres classes peuvent être implémentées sans blocage et très efficacement (implémentation de taille linéaire).

Model-checking de HMSC

Model-checking de HMSCs

- Système à vérifier : HMSC **S**
- Propriété à vérifier : donnée par un autre HMSC **P**
 - Model-checking **positif** : $\text{Executions}(\mathbf{S}) \subseteq \text{Executions}(\mathbf{P})?$
 - Model-checking **négatif**: $\text{Executions}(\mathbf{S}) \cap \text{Executions}(\mathbf{P}) = \emptyset?$

Model-checking de HMSCs

- Système à vérifier : HMSC **S**
- Propriété à vérifier : donnée par un autre HMSC **P**
 - Model-checking **positif** : $\text{Executions}(\mathbf{S}) \subseteq \text{Executions}(\mathbf{P})$?
 - Model-checking **négatif** : $\text{Executions}(\mathbf{S}) \cap \text{Executions}(\mathbf{P}) = \emptyset$?

En général, ces deux problèmes sont **indécidables**
(comme de nombreux problèmes sur les **exécutions**)

Idées

- Restrictions syntaxiques raisonnables sur les HMSCs pour lesquelles model-checking et implémentabilité sont décidables/efficaces
- Pour le model-checking, capturer tous les HMSCs par un ensemble régulier de représentants.

Model-checking de HMSCs coopératifs : résultats

Le model-checking $\text{negatif}(\cap)$ / $\text{positif}(\subseteq)$

- a. de HMSCs localement coopératifs, étiquetés par des **atomes** est **NLOG-complet**/**PSPACE-complet**.
- b. de HMSCs localement coopératifs est **PSPACE-complet**/**[PSPACE-EXPSPACE]**
- c. de HMSCs globalement coopératifs est **PSPACE-complet**/**EXPSPACE-complet**

Model-checking de HMSCs coopératifs : bornes sup

$$\text{Lin}^a(G) = \text{Lin}(G) \cap [\text{Lin}(\text{Atom}(G))]^*$$

- $\text{Lin}^a(G)$ est un ensemble de représentants de $\text{Lin}(G)$.
- $\implies \text{Lin}^a(G) \cap \text{Lin}^a(H) \neq \emptyset$ ssi $L(G) \cap L(H) \neq \emptyset$
- Utiliser l'unicité de la décomposition atomique modulo commutation.
Pour les HMSCs localement coopératifs d'étiquettes **atomiques**, pas de commutation entre MSCs successifs \implies « vrais » automates

Model-checking de HMSCs coopératifs : bornes sup

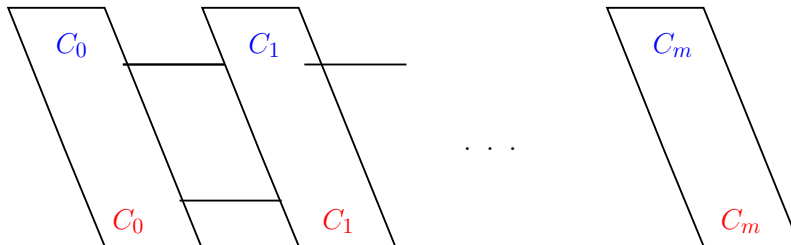
$$\text{Lin}^a(G) = \text{Lin}(G) \cap [\text{Lin}(\text{Atom}(G))]^*$$

- $\text{Lin}^a(G)$ est un ensemble de représentants de $\text{Lin}(G)$.
- $\implies \text{Lin}^a(G) \cap \text{Lin}^a(H) \neq \emptyset$ ssi $L(G) \cap L(H) \neq \emptyset$
- Utiliser l'unicité de la décomposition atomique modulo commutation.
Pour les HMSCs localement coopératifs d'étiquettes **atomiques**, pas de commutation entre MSCs successifs \implies « vrais » automates
- G coopératif $\implies \text{Lin}^a(G)$ régulier, et on peut construire un automate le reconnaissant [MP99]
- On se ramène à la complexité de l'intersection et de l'inclusion sur les automates.

Model-checking de HMSCs coopératifs : bornes inf

Intersection de loc. coopératifs : simulation d'une MT PSPACE.

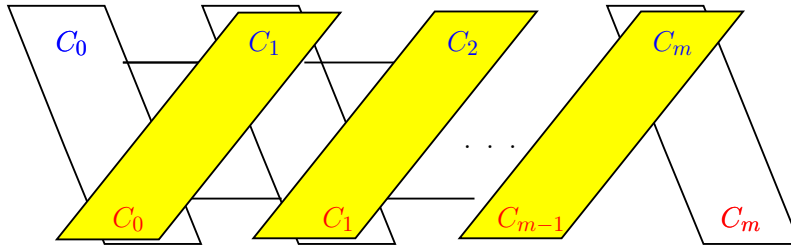
Calcul $C_0 \vdash C_1 \vdash \dots \vdash C_m$, $|C_i| = k = O(|w|^n)$



Model-checking de HMSCs coopératifs : bornes inf

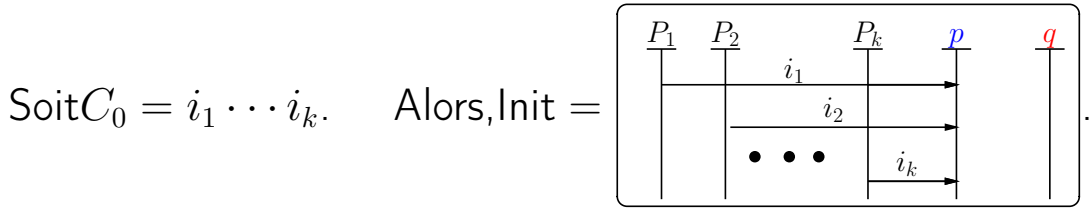
Intersection de loc. coopératifs : simulation d'une MT PSPACE.

Calcul $C_0 \vdash C_1 \vdash \dots \vdash C_m$, $|C_i| = k = O(|w|^n)$

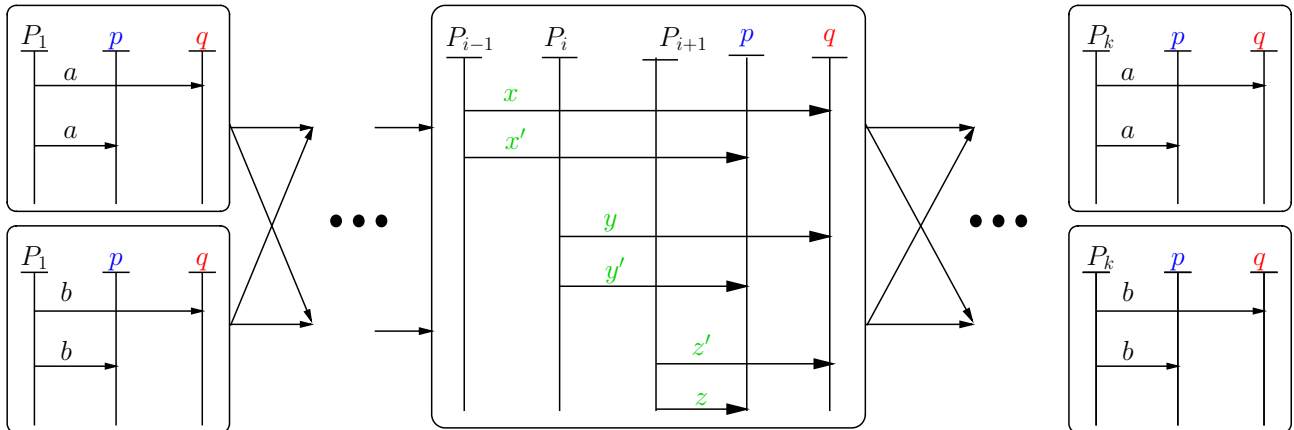


Pour forcer un calcul acceptant $C_i \vdash C_{i+1}$

Le langage du HMSC jaune est **Init** **Trans*** **Accept**.



Trans fait, de façon non-déterministe, une transition $xyz \rightarrow x'y'z'$ à une position



Résumé (FIFO)

Classe \mathcal{C}	G.Coop	L.Coop	L.Coop + atom	Loc.Choice
$\in \mathcal{C}?$	co-NP ⁰	P	P	P
$L \cap K \neq \emptyset?$	PSPACE	NLOG ¹	NLOG	NLOG
$L \subseteq K?$	EXPSPACE	PSPACE ¹	PSPACE	PSPACE
Implem. weak	Undec ^{2,3}			
Implem. safe	EXPSPACE ²	PSPACE ²		
Implem.+data	Yes	$ G ^{O(\wp)}$	$ G ^{O(\wp)}$	$ G $

0) [MP99] 1) if \wp constant 2) [Loh02] 3) [AEY]

- Model-checking et implémentabilité peuvent être **décidables** (efficacement)
- Les ordres partiels et traces une intuition

Logiques sur les traces

Logiques temporelles et concurrence : motivations

Logique temporelle = langage de spécification

- facile d'utilisation,
- idéalement expressif et raisonnablement vérifiable.

Tout ou rien

Pour un système concurrent, une propriété ne doit pas dépendre de l'ordre **observé** entre actions indépendantes.

Expressivité Comparaison avec la logique du premier ordre $FO(<)$.

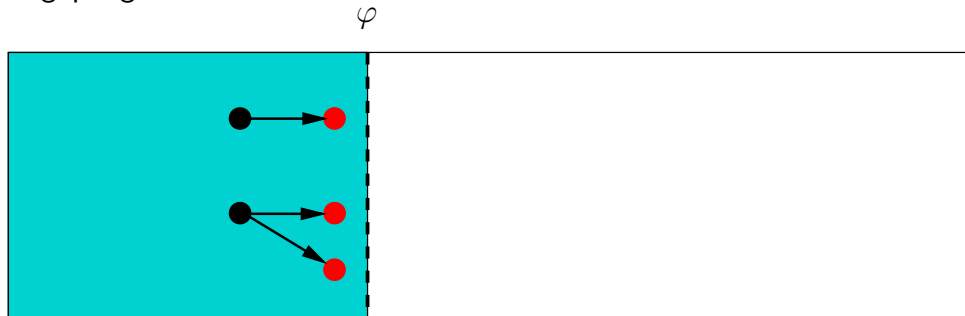
$[\exists x \forall y (y \geq x) \wedge P_a(x)]$ définit le langage aA^* , ie $\mathbf{X}a$

Théorème $LTL_A(\mathbf{X}, \mathbf{U}) = FO_A(<) =$ Langages sans-étoile

La logique du premier ordre est cependant beaucoup plus compacte.

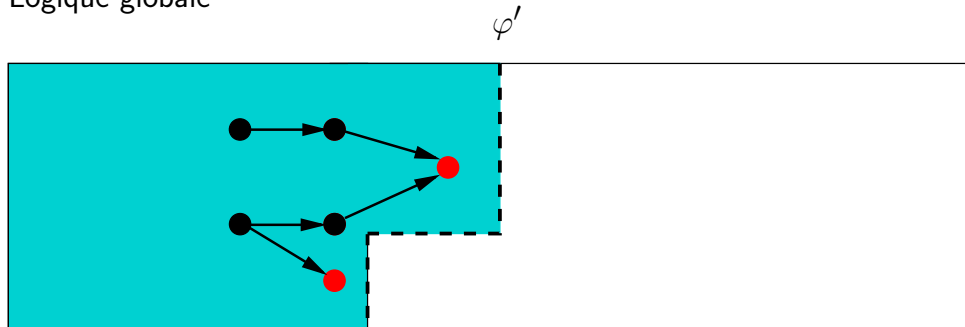
LTL sur les traces : local vs. global

Logique globale



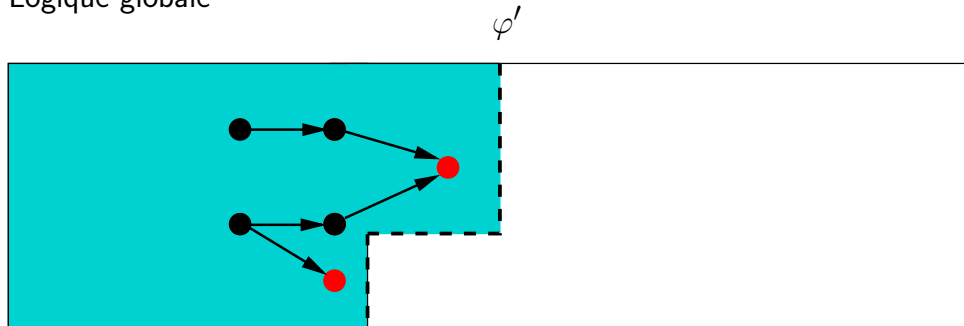
LTL sur les traces : local vs. global

Logique globale



LTL sur les traces : local vs. global

Logique globale

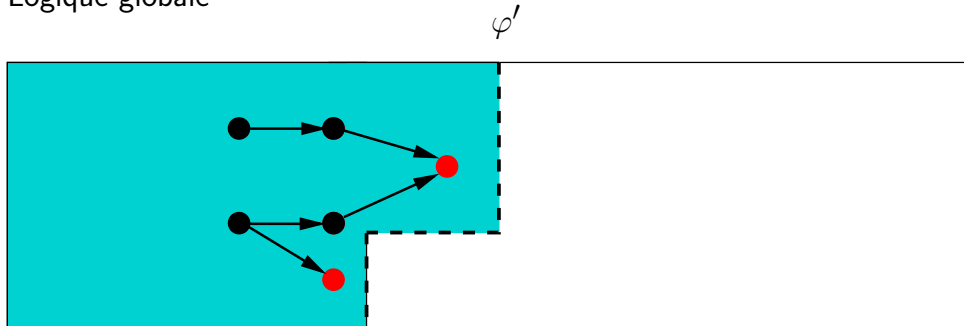


Logique locale

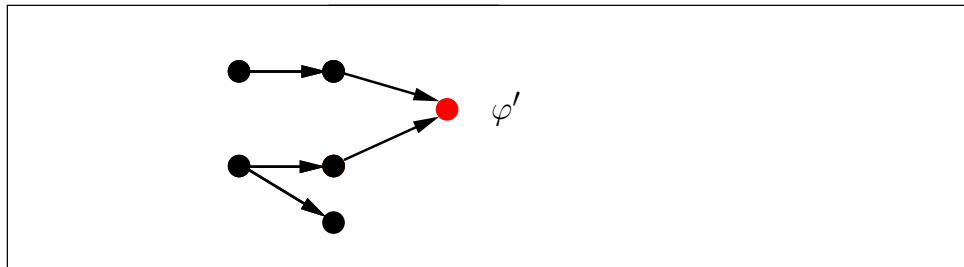


LTL sur les traces : local vs. global

Logique globale



Logique locale



Logiques globales : opérateur Next

Logiques globales : travaillent sur les **préfixes** des traces

Logiques globales : opérateur Next

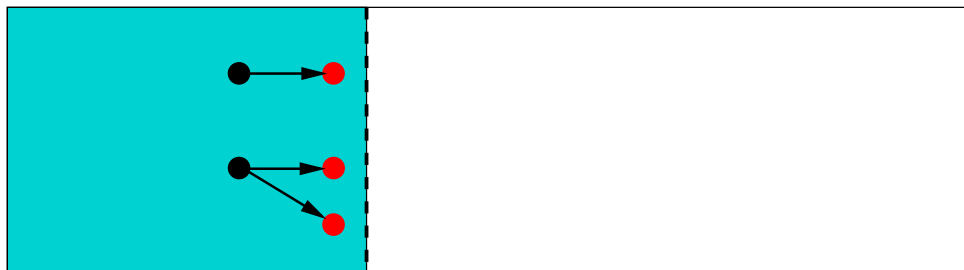
Logiques globales : travaillent sur les **préfixes** des traces

Plusieurs sommets pouvant continuer un préfixe

⇒ plusieurs opérateurs Next : $(\mathbf{X}_a)_{a \in \Sigma}$

Next

$\mathbf{X}_a \phi$



Logiques globales : opérateur Next

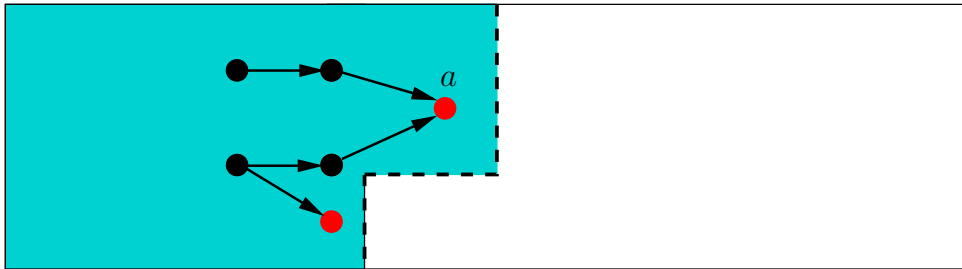
Logiques globales : travaillent sur les **préfixes** des traces

Plusieurs sommets pouvant continuer un préfixe

⇒ plusieurs opérateurs Next : $(\mathbf{X}_a)_{a \in \Sigma}$

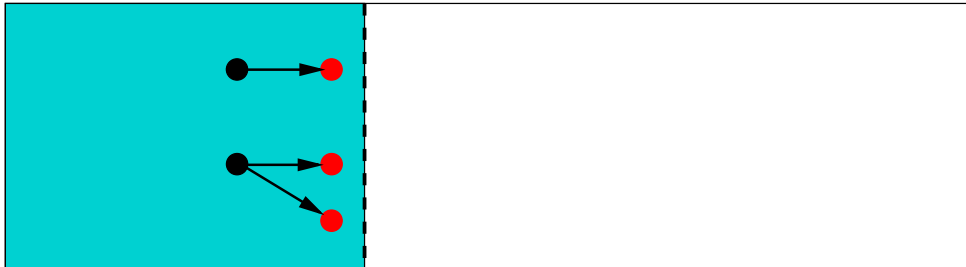
Next

$\mathbf{X}_a \phi$ ϕ



Logiques globales : les 'Until'

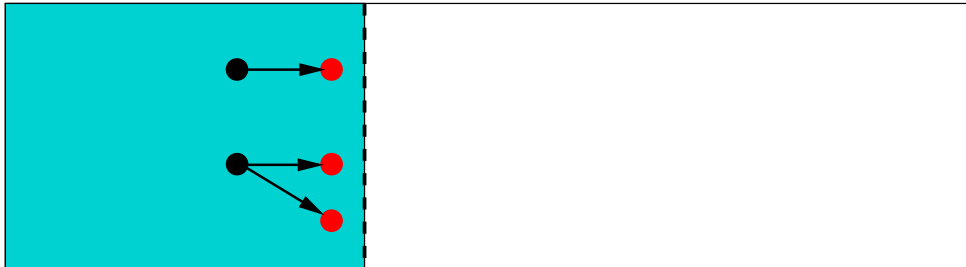
Until universel



Logiques globales : les 'Until'

Until universel

$\alpha \mathbf{AU} \beta$

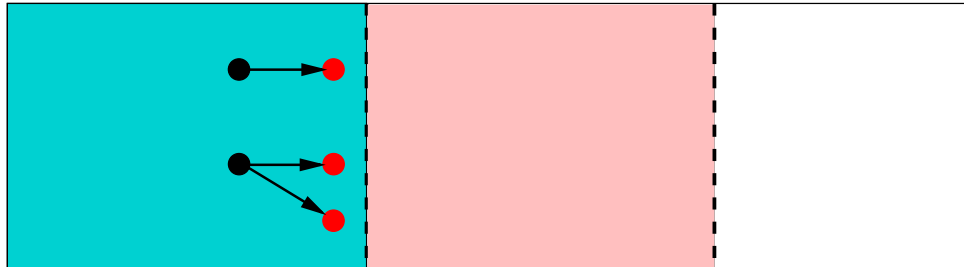


Logiques globales : les 'Until'

Until universel

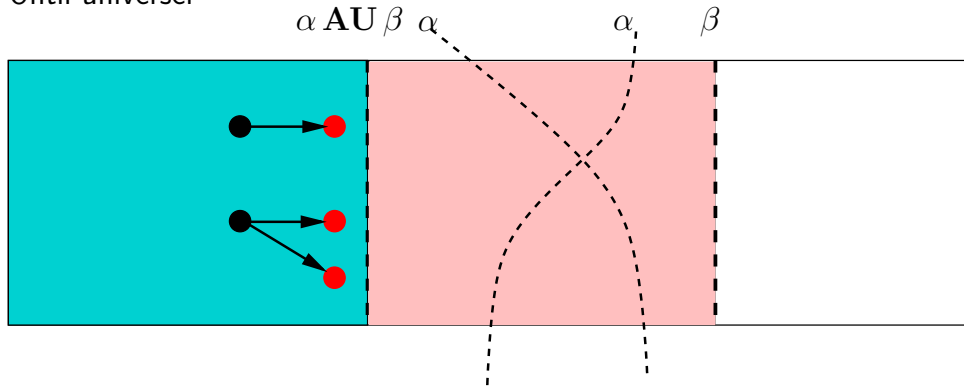
$\alpha \mathbf{AU} \beta$

β



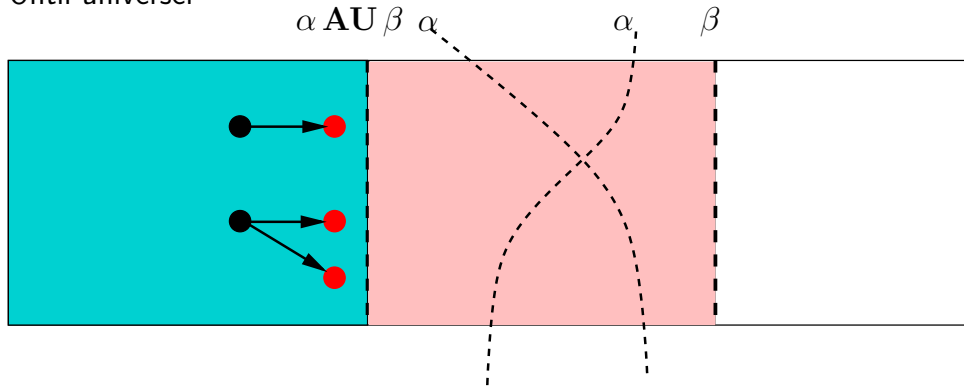
Logiques globales : les 'Until'

Until universel

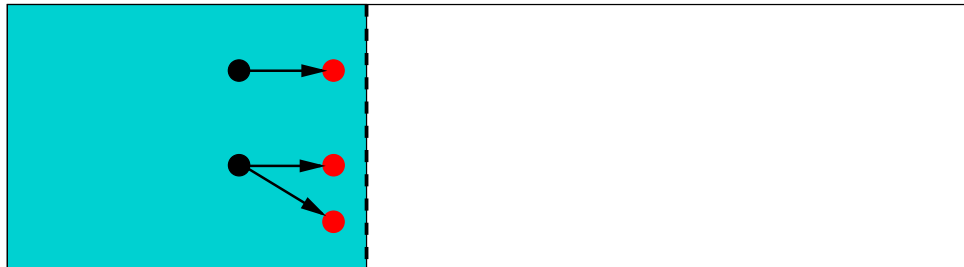


Logiques globales : les 'Until'

Until universel

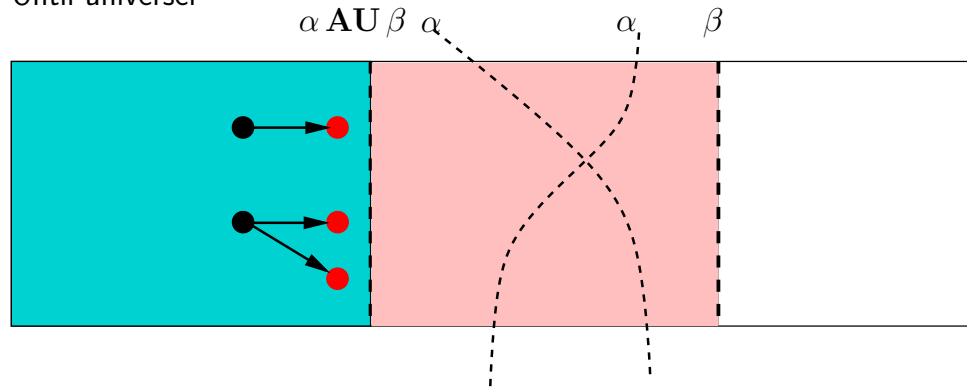


Until existentiel

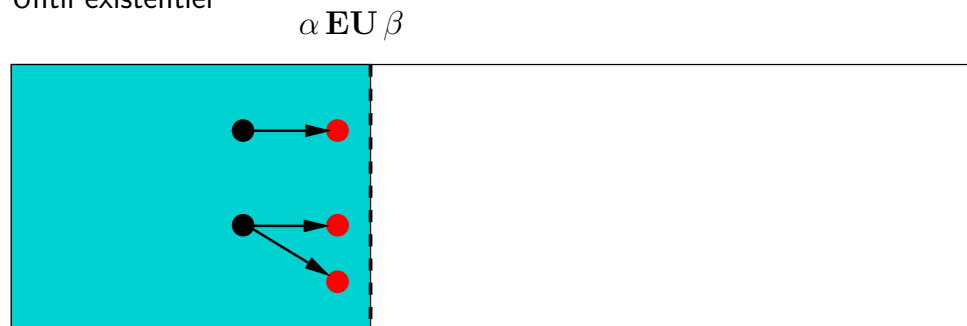


Logiques globales : les 'Until'

Until universel

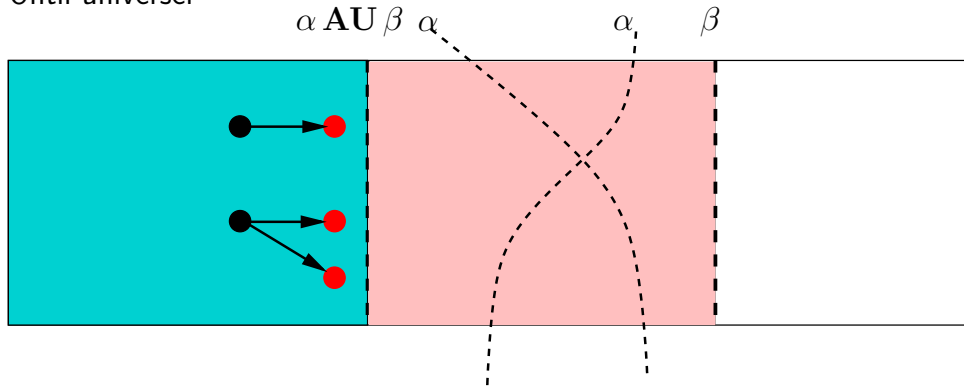


Until existentiel

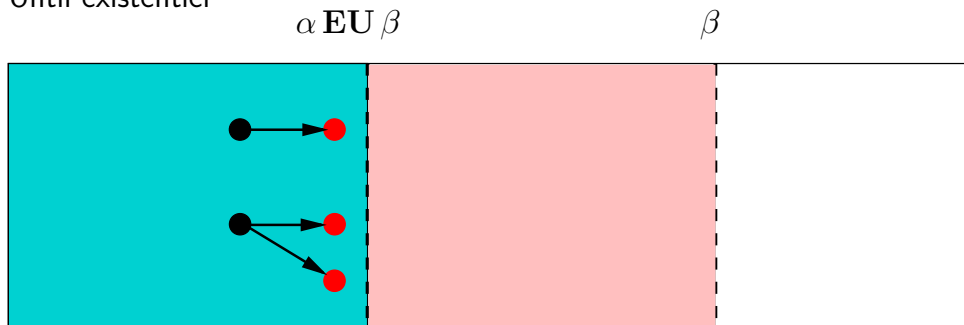


Logiques globales : les 'Until'

Until universel

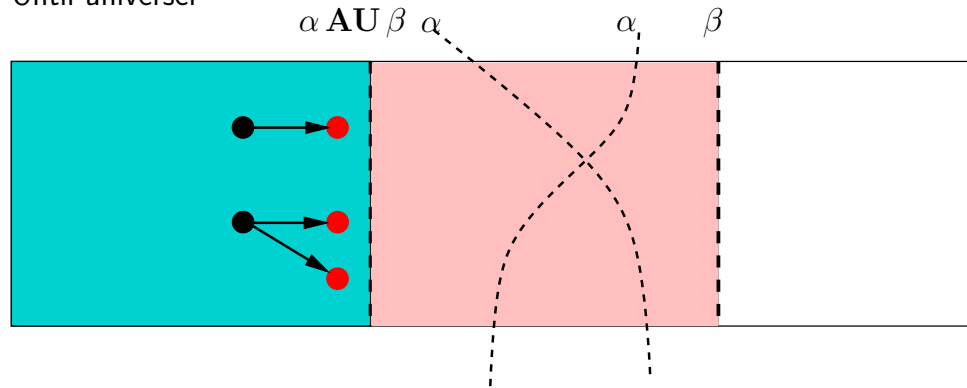


Until existentiel

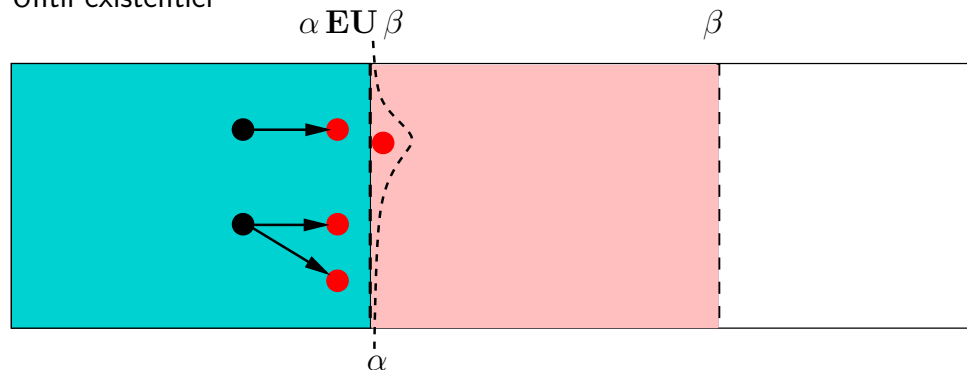


Logiques globales : les 'Until'

Until universel

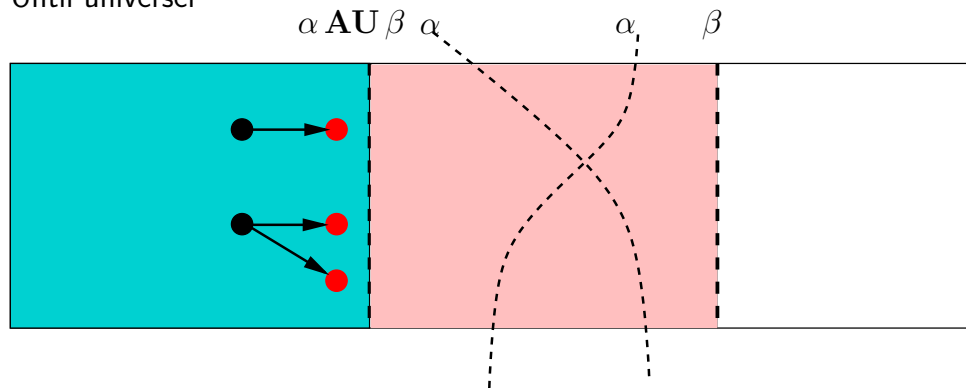


Until existentiel

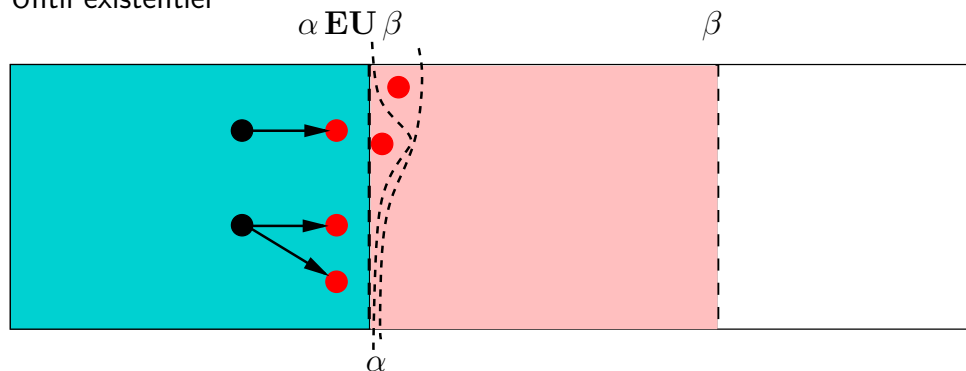


Logiques globales : les 'Until'

Until universel

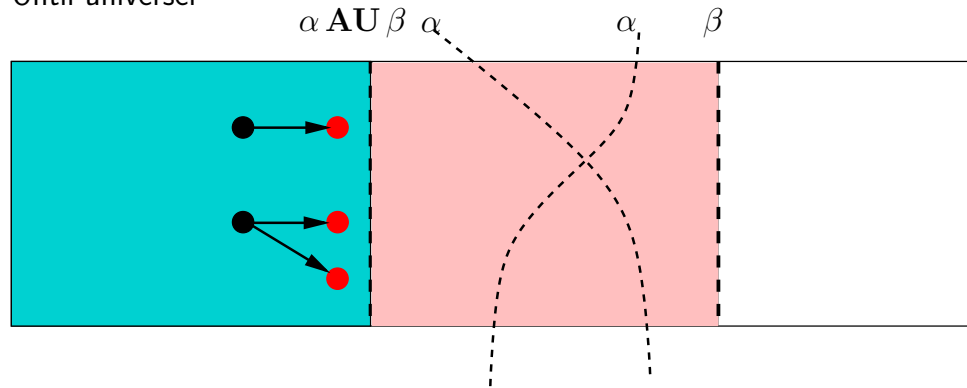


Until existentiel

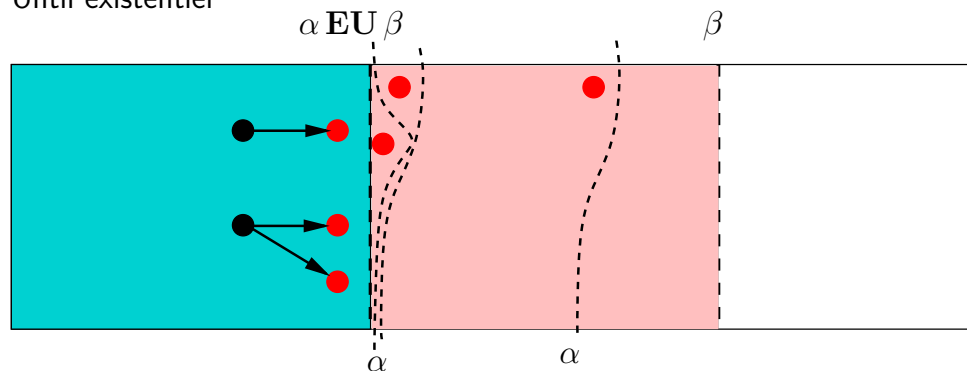


Logiques globales : les 'Until'

Until universel



Until existentiel



Logiques globales : expressivité et complexité

- $LTrL(\mathbf{X}_a, \mathbf{AU})$ est **expressivement complète** [DG01]
- On peut effectivement transformer une formule de $LTrL(\mathbf{X}, \mathbf{AU})$ en automate de Büchi [GMP98]

Mais...

Logiques globales : expressivité et complexité

- $LTrL(\mathbf{X}_a, \mathbf{AU})$ est **expressivement complète** [DG01]
- On peut effectivement transformer une formule de $LTrL(\mathbf{X}, \mathbf{AU})$ en automate de Büchi [GMP98]

Mais...

- la satisfaisabilité de $LTrL(\mathbf{X}, \mathbf{AU})$ est non-élémentaire [W98]

Logiques globales : expressivité et complexité

- $\text{LTrL}(\mathbf{X}_a, \mathbf{AU})$ est **expressivement complète** [DG01]
- On peut effectivement transformer une formule de $\text{LTrL}(\mathbf{X}, \mathbf{AU})$ en automate de Büchi [GMP98]

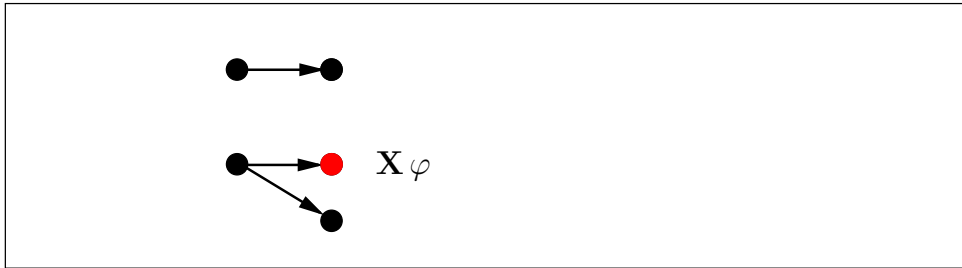
Mais...

- la satisfaisabilité de $\text{LTrL}(\mathbf{X}, \mathbf{AU})$ est non-élémentaire [W98]
- $\text{LTrL}(\mathbf{X}_a, \mathbf{EU})$ est indécidable [APP98]

Logiques locales, opérateurs

Parlent des propriétés liées à l'ordre entre événements du système.

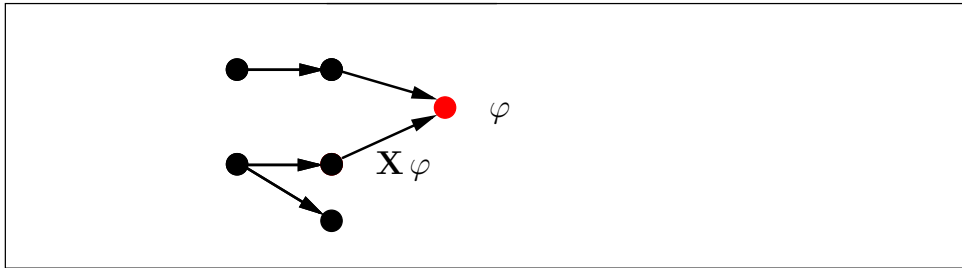
Logique locale, Next



Logiques locales, opérateurs

Parlent des propriétés liées à l'ordre entre événements du système.

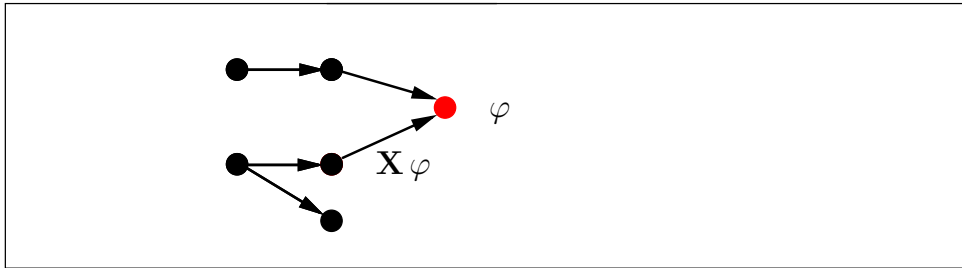
Logique locale, Next



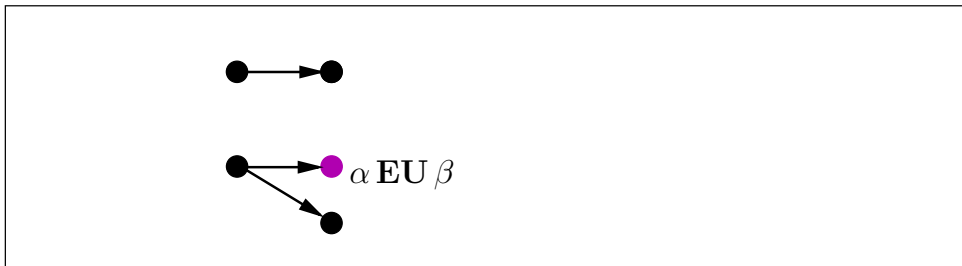
Logiques locales, opérateurs

Parlent des propriétés liées à l'ordre entre événements du système.

Logique locale, Next



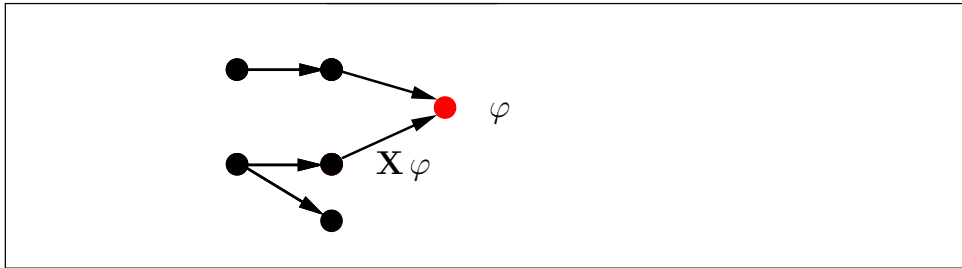
Logique locale, Until existentiel (le Until universel se définit de même)



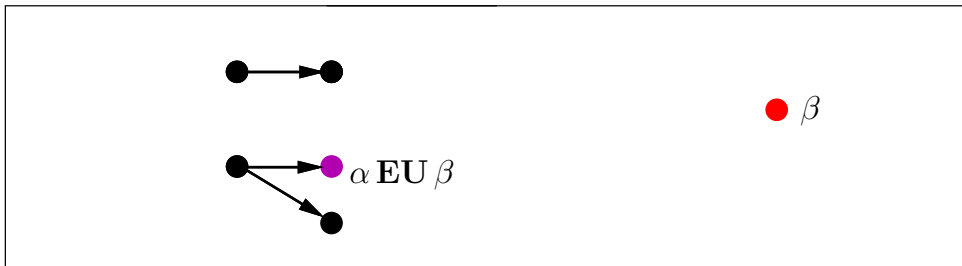
Logiques locales, opérateurs

Parlent des propriétés liées à l'ordre entre événements du système.

Logique locale, Next



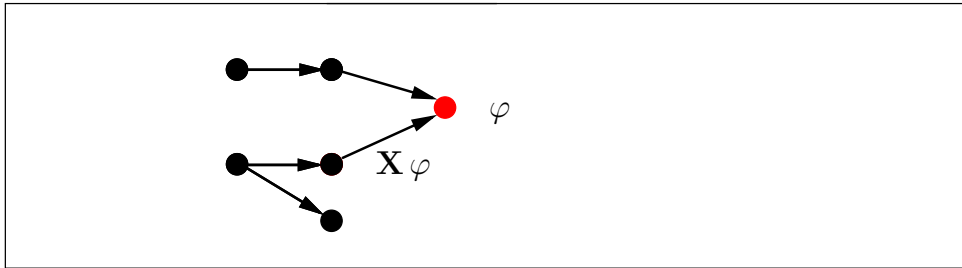
Logique locale, Until existentiel (le Until universel se définit de même)



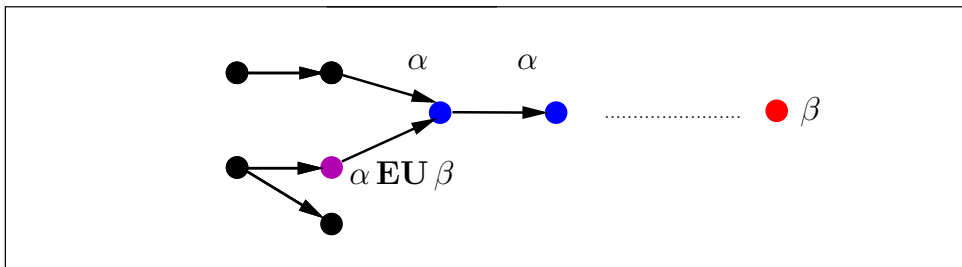
Logiques locales, opérateurs

Parlent des propriétés liées à l'ordre entre événements du système.

Logique locale, Next



Logique locale, Until existentiel (le Until universel se définit de même)



Logiques locales : expressivité et complexité

- **Toute** logique locale dont les modalités sont définissables en MSO est décidable en PSPACE [GK02].
- Les alphabets de traces pour lesquels $\text{LocTL}(\mathbf{X}, \mathbf{AU})$ et $\text{LocTL}(\mathbf{X}, \mathbf{EU})$ sont expressivement complètes sont complètement caractérisés [DG01].

Dans les deux cas, ce sont ceux ne contenant pas $a - b - c - d$ comme sous graphe induit

- Logiques expressivement complètes avec opérateurs de passé [GM02].
- Ouvert : recherche de logique pur-futur expressivement complète.