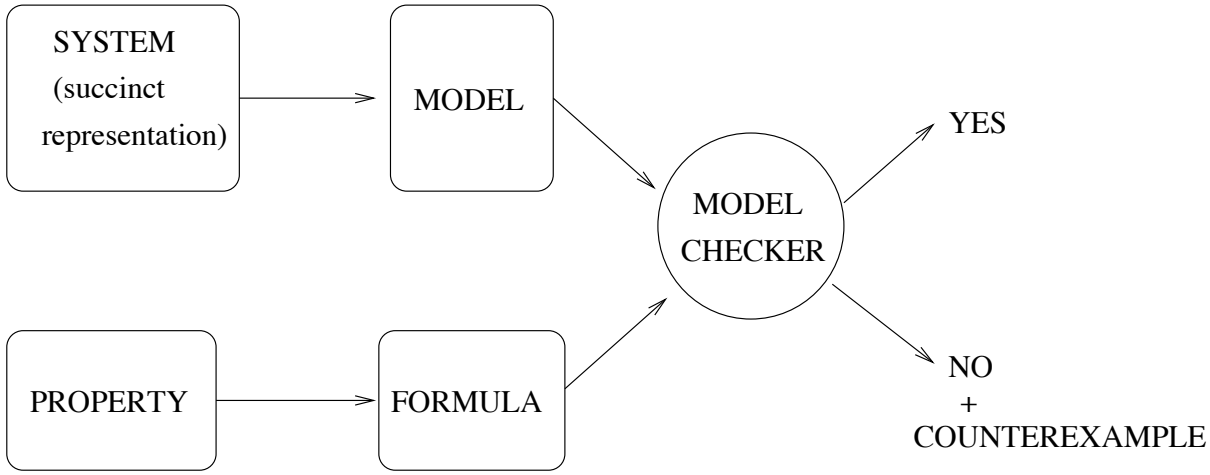
# Logique, Complexité et Vérification

Richard Lassaigne
Logique mathématique,
CNRS-Université Paris 7

**Logique et Vérification**

**Vérification par Model Checking**
**Model Checking et Automates**

**Vérification probabiliste et Approximation**




Entrée :

- Modèle $\mathcal{M} = (S, R)$ $R \subseteq S^2$ (relation de transition)
- Etat initial $s_0$
- Formule $\varphi$

Sortie :

- OUI si $(\mathcal{M}, s_0) \models \varphi$
- NON avec trace d'erreur si $(\mathcal{M}, s_0) \not\models \varphi$

## Machine à Etats Finis Etendue (EFSM)

Une **EFSM** est donnée par $\mathcal{M}_F = (Q, V, Gardes, Actions, \Delta, init)$ :

- $Q$ est un ensemble fini d'états de contrôle
- $V = \{v_1, v_2, \ldots, v_m\}$ est un ensemble de variables
  Chaque $v_i \in V$ prend ses valeurs dans un ens. fini $D_i$
- $Gardes = \{G_1, G_2, \ldots, G_r\}$ est un ens. de prédicats
  $G_i(v_1, v_2, \ldots, v_m)$, évalués à $vrai$ ou $faux$ suivant une affectation $\alpha \in D_1 \times D_2 \times \cdots \times D_m$ des variables
  exemple de garde : $v_2 \geq v_4$
- $Actions = \{act_1, act_2, \ldots, act_d\}$ est un ens. de fonctions
  $act_i : D_1 \times D_2 \times \cdots \times D_m \longrightarrow D_1 \times D_2 \times \cdots \times D_m$
  qui associe à une affectation une autre affectation
  exemple d'action : $v_3 = v_1 + v_2 + 1$
- $\Delta \subseteq Q \times Gardes \times Actions \times Q$ est un ens. de transitions
- $init = (q_0, \alpha_0)$ est un état initial avec une affectation initiale $\alpha_0$

## Système de Transition associé à une EFSM

### Modèle de Kripke

Une **EFSM** fournit une représentation *succincte* du système de transition associé $\mathcal{M} = (S, R, init)$ :

- $S = Q \times (D_1 \times D_2 \times \cdots \times D_m)$ est l'ensemble des états de $\mathcal{M}$
- $R \subseteq S^2$ est la relation de transition définie par
  pour tout état $s = (q, \alpha) \in S$ et $s' = (q', \alpha') \in S$, $(s, s') \in R$ ssi
  il existe $(q, G, act, q') \in \Delta$ t.q. $G(\alpha) = vrai$ et $act(\alpha) = \alpha'$
- $init = (q_0, \alpha_0) \in S$ est aussi l'état initial de $\mathcal{M}$

Chaque exécution de $\mathcal{M}$ définit un chemin (infini)

$\sigma = (s_0, s_1, \ldots, s_i, \ldots)$ t.q. $s_0 = init$ et $(s_i, s_{i+1}) \in R$ $(i \geq 0)$

## Système de Transition étiqueté

- On suppose que l'on peut observer certaines propriétés intéressantes des états du système :
  soit $P = \{p_1, p_2, \ldots, p_n\}$ un ens. de propositions atomiques
- Soit $L : S \longrightarrow \Sigma_P$ où l'alphabet $\Sigma_P = 2^P$ :
  pour $s \in S$, $L(s)$ est l'ens. des propositions vraies en $s$
- On obtient ainsi un système de transition étiqueté :
  $\mathcal{M}_P = (S, \Sigma_P, R, L, init)$
- A tout chemin $\sigma = (s_0, s_1, \ldots, s_i, \ldots)$ de $\mathcal{M}_P$,
  on peut associer un mot infini
  $L(\sigma) = L(s_0)L(s_1)\ldots L(s_i)\cdots \in (\Sigma_P)^\omega$
- $\omega$-langage associé à une **EFSM** :
  $L_P(\mathcal{M}) = L(\mathcal{M}_P) = \{L(\sigma) \ / \ \sigma$ est un chemin de $\mathcal{M}_P\}$

## Logique Temporelle Linéaire (LTL)

Les formules de LTL sont construites à l'aide de

- propositions atomiques : $P = \{p_1, p_2, \ldots, p_n\}$
- connecteurs propositionnels : $\neg, \vee$ $(\wedge, \rightarrow)$
- opérateurs temporels : $X$ (Next) $U$ (Until)

L'ensemble des formules LTL est défini par induction :

- toute proposition atomique $p_i$ est une formule,
- si $\varphi$ et $\psi$ sont des formules,
- alors $\neg\varphi$, $\varphi \vee \psi$, $X\varphi$ et $\varphi U \psi$ sont des formules

Autres opérateurs temporels :

$$F\psi \equiv (trueU\psi) \ (true \equiv (p_1 \vee \neg p_1))$$
$$G\psi \equiv \neg F \neg \psi$$

## Logique Temporelle Linéaire (LTL)

On interprète les formules de LTL sur les chemins
Chemin $\sigma = (s_0, s_1, \ldots, s_i, \ldots)$ : on note $\sigma^i$ le chemin $(s_i, s_{i+1}, \ldots)$

La relation de satisfaction pour les formules LTL est définie par induction :

- $\sigma \models p_i$ ssi $p_i \in L(s_0)$
- $\sigma \models \neg\psi$ ssi $\sigma \not\models \psi$
- $\sigma \models \varphi \vee \psi$ ssi $\sigma \models \varphi$ ou $\sigma \models \psi$
- $\sigma \models X\psi$ ssi $\sigma^1 \models \psi$
- $\sigma \models \varphi U \psi$ ssi $(\exists i \geq 0)$ t.q. $\sigma^i \models \psi$ et $(\forall j < i)$ $\sigma^j \models \varphi$

La relation de satisfaction est étendue aux systèmes de transition : $\mathcal{M}_P \models \psi$ ssi pour tout chemin $\sigma$ de $\mathcal{M}_P$, $\sigma \models \psi$

## Automates de Büchi

- Si l'on ajoute un ensemble $F \subseteq S$ d'états acceptants
  à un modèle de Kripke, on obtient un automate de Büchi :
  $\mathcal{A} = (S, \Sigma, R, L, Init, F)$

- Un chemin $\sigma = (s_0, s_1, \ldots, s_i, \ldots)$ est acceptant
  ssi pour une infinité de $i$, $s_i \in F$

- On considère un ens. d'états initiaux $Init \subseteq S$
  Un chemin $\sigma$ doit avoir son origine $s_0 \in Init$

- Le langage associé à l'automate $\mathcal{A}$ est le langage :
  $L(\mathcal{A}) = \{L(\sigma) \ / \ \sigma$ est un chemin acceptant de $\mathcal{A}\}$
- Un mot infini $w \in (\Sigma)^\omega$ est accepté par $\mathcal{A}$ si $w \in L(\mathcal{A})$

## Réduction du problème du model checking à celui de l'intersection vide de 2 langages

1. Convertir une formule LTL $\psi$ $(= \neg\varphi)$ en un automate de Büchi $\mathcal{A}_\psi$ t.q. $L(\psi) = L(\mathcal{A}_\psi)$ (Vardi,Wolper,Courcoubetis,Yannakakis)
2. Déterminer si $\mathcal{M} \models \varphi$ en vérifiant si l'intersection des 2 langages $L(\mathcal{M}_P) \cap L(\mathcal{A}_\psi)$ est vide

$$\mathcal{M}_P \models \varphi \quad \Leftrightarrow \quad L(\mathcal{M}_P) \subseteq L(\mathcal{A}_\varphi)$$

$$\mathcal{M}_P \models \varphi \quad \Leftrightarrow \quad L(\mathcal{M}_P) \cap L(\mathcal{A}_{\neg\varphi}) = \emptyset$$

**Déterminer si** $L(\mathcal{M}_P) \cap L(\mathcal{A}_\psi) = \emptyset$

Etant donnés $\mathcal{M}_P = (S, \Sigma_P, R, L, init)$ et
$\mathcal{A}_\psi = (Q, \Sigma_P, R', L', Init', F')$,
on définit l'automate produit :

$$\mathcal{M}_P \otimes \mathcal{A}_\psi = (S^\otimes, \Sigma_P, R^\otimes, L^\otimes, Init^\otimes, F^\otimes)$$

- $S^\otimes = \{(s,q) \in S^\otimes \ / \ L(s) \text{ satisfait } L'(q)\}$
- $R^\otimes = \{((s,q),(s',q')) \in S^\otimes \times S^\otimes \ / \ (s,s') \in R \text{ et } (q,q') \in R'\}$
- $L^\otimes((s,q)) = L(s)$
- $Init^\otimes = \{(init,q) \in S^\otimes \ / \ q \in Init'\}$
- $F^\otimes = \{(s,q) \in S^\otimes \ / \ q \in F'\}$

*Proposition 1* : $L(\mathcal{M}_P \otimes \mathcal{A}_\psi) = L(\mathcal{M}_P) \cap L(\mathcal{A}_\psi)$

**Déterminer si** $L(\mathcal{M}_P) \cap L(\mathcal{A}_\psi) = \emptyset$

*Proposition 2* : $L(\mathcal{M}_P \otimes \mathcal{A}_\psi = \emptyset$ ssi
il n'existe pas de cycle accessible dans l'automate $\mathcal{M}_P \otimes \mathcal{A}_\psi$
contenant un état acceptant

*Idée* : Etant donnés la FSM $\mathcal{M}_F$ et l'automate $\mathcal{A}_\psi$,
on explore l'espace des états de $\mathcal{M}_P \otimes \mathcal{A}_\psi$,
mais seulement en construisant les états si nécessaire :
si l'on trouve un mauvais cycle, on arrête

**Déterminer si** $L(\mathcal{M}_P) \cap L(\mathcal{A}_\psi) = \emptyset$

- On commence avec l'ensemble $Init^{\otimes}$

- Pour $s \in S^{\otimes}$, on peut calculer
  l'ens. de ses successeurs efficacement
  On effectue une recherche en profondeur d'abord (DFS)
  pour obtenir tous les états accessibles de $\mathcal{M}_P \otimes \mathcal{A}_\psi$

- Pour obtenir les mauvais cycles, on calcule
  les composantes fortement connexes en utilisant
  l'algorithme standard DFS et on vérifie s'il existe
  une composante fortement connexe accessible
  contenant un état acceptant

# Complexité

$O(|\mathcal{M}|.|\varphi|)$ (Branching Time Temporal Logic **CTL**)

ou

$O(|\mathcal{M}|.2^{|\varphi|})$ (Linear Time Temporal Logic **LTL**)

Problème :

Phénomène d'explosion de l'espace des états

(le problème n'est pas le temps mais l'espace)

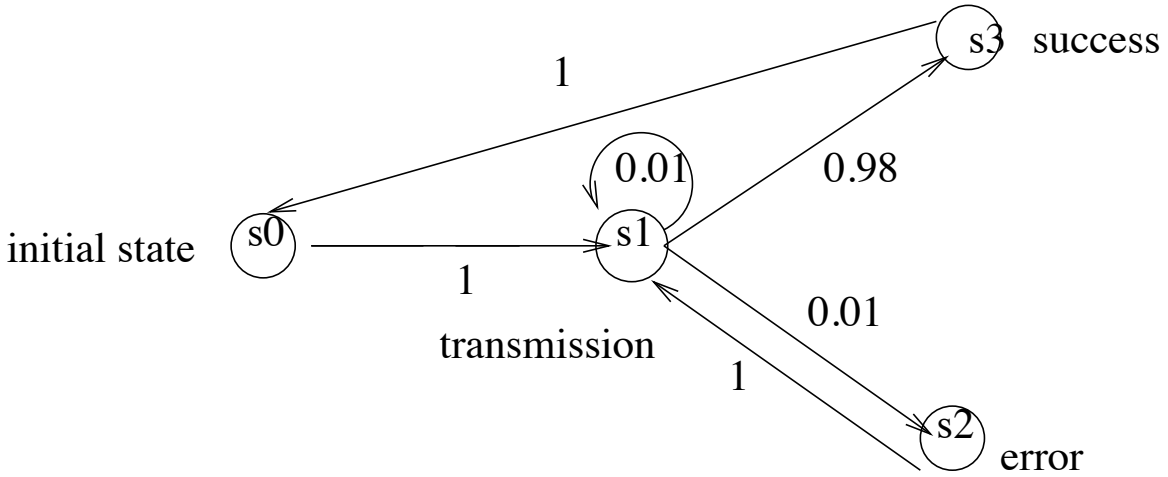
Méthodes classiques :

- Représentation symbolique (OBDD)
- Méthodes basées sur l'utilisation de SAT-solvers (Bounded Moded Checking)
- Abstraction

# Systèmes de Transition Probabilistes

Entrée :

- Modèle $\mathcal{M} = (S, P, L)$ et état initial $s_0$
- $P : S^2 \longrightarrow [0, 1]$ Fonction de probabilité
- $L : S \longrightarrow 2^{AP}$ (étiquetage des états)
- Formula $\psi$ (**LTL**)




Sortie : $Prob_{\Omega}[\psi]$

Exemple : $\psi \equiv transmission$ Until $success$

($\Omega$ espace probabiliste des chemins d'exécution d'origine $s_0$)

## **Espace probabiliste** :

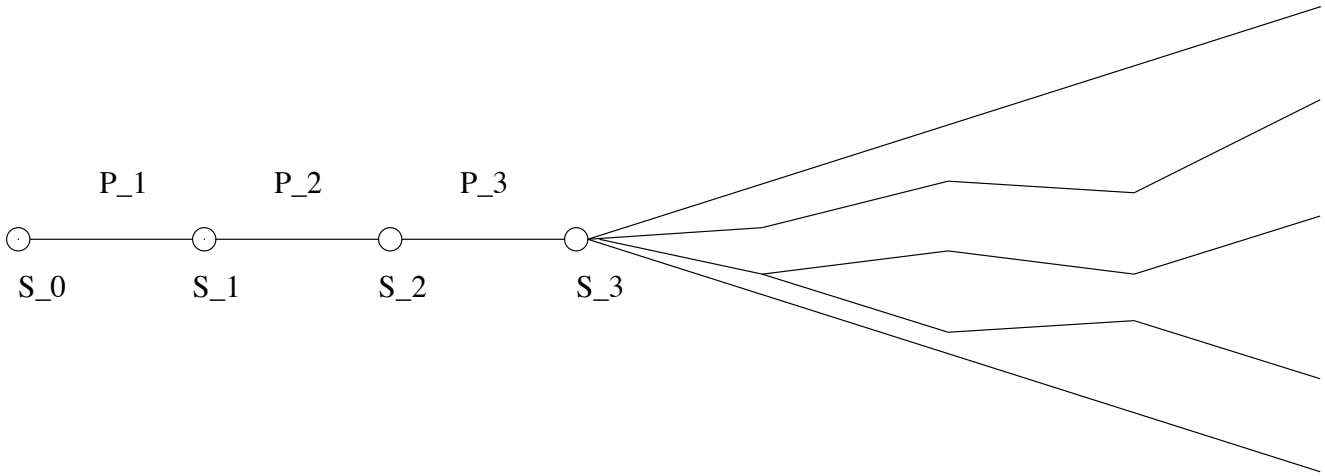
Cône des extensions d'un chemin fini $\rho = (s_0, s_1, \ldots, s_n)$ :

$$Prob(\{\sigma/\sigma \text{ est un chemin et } (s_0, s_1, \ldots, s_n) \text{ est un préfixe of } \sigma\}) =$$

$$\prod_{i=1}^{n} P(s_{i-1}, s_i)$$

La mesure peut être définie sur la famille borélienne engendrée par les ensembles $\{\sigma/\rho$ préfixe de $\sigma\}$ où $\rho$ est un chemin fini.

L'ensemble des chemins $\{\sigma/\sigma(0) = s$ and $\mathcal{M}, \sigma \models \psi\}$ est mesurable (Vardi) et $Prob_\Omega[\psi]$ est bien définie



**Complexité :** (Coucourbetis et Yannakakis, 1995)

**Vérification qualitative (i.e. prob ¿ 0 ?)**

Même **complexité** que **model checking pour LTL**

$$O(|M|.2^{|\psi|})$$

**Vérification Quantitative (i.e. prob = ?)**

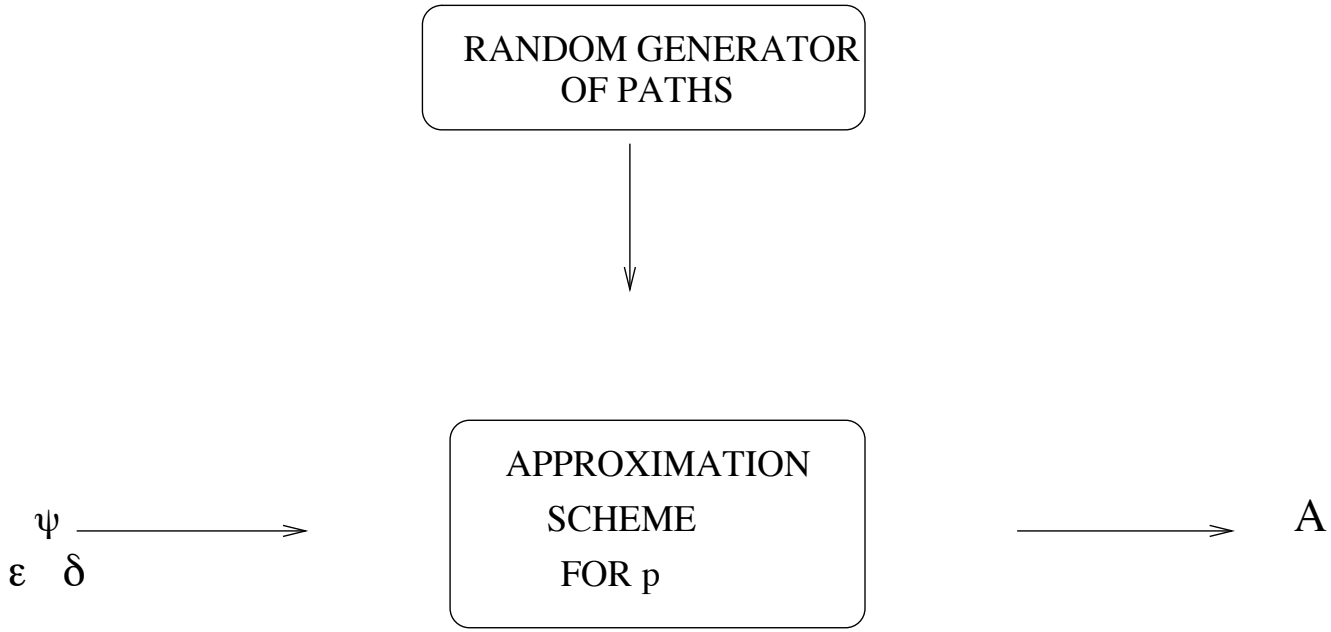
$$O(poly(|M|).2^{|\psi|})$$

**Méthode :** Calculer $Prob_{\Omega}[\psi]$

- Transformer étape par étape la formule et la chaîne de Markov $\mathcal{M}$
- Eliminer les connecteurs temporels un par un
- En préservant la probabilité de satisfaction
- En résolvant un système d'équations linéaires de taille $|M|$.

# Schéma probabiliste d'approximation

On désire approximer une probabilité $p$.




$$Pr[(p-\varepsilon) \leq A \leq (p+\varepsilon)] \geq 1-\delta$$

$\varepsilon$ : *paramètre d'approximation (additive)*

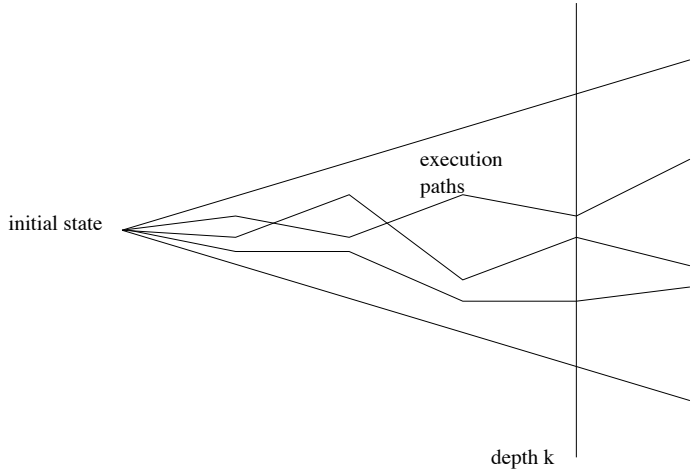
$\delta$ : *paramètre de confiance (algorithme probabiliste)*

Le schéma est dit pleinement polynomial (FPRAS)

si le temps est $poly(|\text{entrée}|, (1/\varepsilon), \log(1/\delta))$

On considère $Prob_k[\psi]$ où :

- l'espace probabiliste est l'espace des chemins de longueur $\leq k$




- $\psi$ exprime une propriété monotone

$$\lim_{k \to \infty} Prob_k[\psi] = Prob_\Omega[\psi]$$

**Algorithme générique d'approximation $\mathcal{GAA}$**

**input :** $\phi, diagramme, \varepsilon, \delta$

$A := 0$

$N := \log(\frac{2}{\delta})/2\varepsilon^2$

Pour $i$ := $1$ à $N$

- Engendrer de manière aléatoire un chemin $\sigma$ de longueur $k$
- Si $\psi$ est vraie sur $\sigma$ alors $A := A + 1$

Retourner $(A/N)$

Algorithme basé sur une estimation de type Monte-Carlo et la borne de Chernoff-Hoeffding

Diagramme : représentation succincte du système (par exemple en Reactive Modules)

## Méthode :

Estimation (Monte-Carlo) + borne de Chernoff-Hoeffding

$X$ variable de Bernoulli $(0, 1)$ avec probabilité de succès $p$

- Faire $N$ tirages aléatoires indépendants $X_1, X_2, \ldots, X_N$
- Estimer $p$ par $\mu = \sum_{i=1}^{N} X_i/N$ avec erreur $\varepsilon$
- La taille de l'échantillon $N$ est telle que la probabilité d'erreur (de l'algorithme) $< \delta$

Borne de Chernoff-Hoeffding :

$$Pr[\mu < p - \varepsilon] + Pr[\mu > p + \varepsilon] < 2e^{-2N\varepsilon^2}$$

Si $N \geq \ln(\frac{2}{\delta})/2\varepsilon^2$, alors

$$Pr[p - \varepsilon \leq \mu \leq p + \varepsilon)] \geq 1 - \delta$$

**Théorème** :

$\mathcal{GAA}$ est un FPRAS pour $Prob_k[\psi]$

**Méthodologie :** Pour approximer $Prob_\Omega[\psi]$

- Choisir $k \approx log|M| \cdot ln(1/\varepsilon)$
- Itérer l'approximation de $Prob_k[\psi]$

**Corollaire** :

L'algorithme de point fixe obtenu en itérant $\mathcal{GAA}$ est un schéma probabiliste d'approximation en espace logarithmique pour $Prob_\Omega[\psi]$

**Remarque :**

- La longueur des chemins nécessaires peut être le diamètre du système
- La vitesse de convergence peut être lente, mais la complexité en espace est logarithmique...

- Implémentation distribuée de la méthode d'approximation [PCMC05]
- Extension (XRM) du langage standard de modélisation des systèmes probabilistes en temps discret et en temps continu (APMC 3.0) [QEST06]
- Nombreux cas d'étude (algorithmes distribués, protocoles de communication, réseaux de capteurs, modèles biologiques,...) [AVoCS04] [AVoCS06] [ISoLA06]
- Outil de vérification et de simulation pour des valeurs réelles des paramètres du modèle (pas d'explosion en espace)
- Intégration avec le model checker probabiliste **PRISM** (Birmingham)
- Collaboration avec le projet Contraintes (INRIA) pour la simulation de modèles biochimiques