Modélisation Probabiliste et Apprentissage par Renforcement

Benoît Delahaye

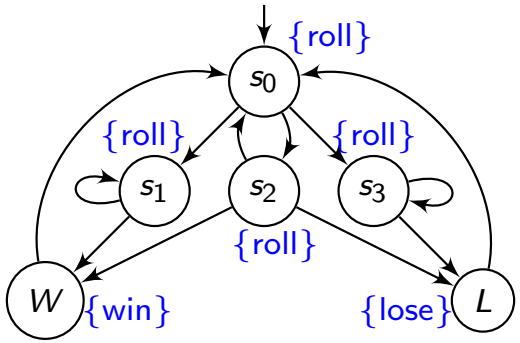
Nantes Université, LS2N UMR 6004

Chapitre 2 : Vérification Probabiliste

Outline

Outline

Rappels (?)



Système de transitions

Un Système de transitions est un tuple $M = (S, I(, Act), T, AP, L)$ avec

- S un ensemble d'états, et $I \subseteq S$ des états initiaux,
- Act un ensemble d'actions,
- $T \subseteq S(\times Act) \times S$ une relation de transition,
- AP un ensemble de labels,
- $L : S \mapsto 2^{AP}$ une fonction d'étiquetage.

Chemins, Traces

(Fragment de) Chemin

Un chemin est une succession infinie d'états $\pi = s_0, s_1, \ldots$ telle que pour tout i, $(s_i, s_{i+1}) \in T$.
Un fragment de chemin est un chemin fini.

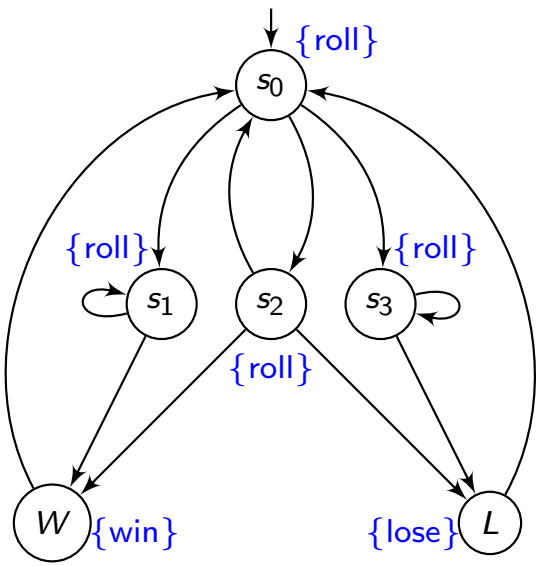
Les traces sont les projections des chemins sur les étiquetages.

Trace

La trace correspondant à un (fragment de) chemin $\pi = s_0 s_1 s_2 \ldots$ est $\mathit{trace}(\pi) = L(s_0)L(s_1)L(s_2)\ldots$.

Les traces d'un ST C sont donc l'ensemble des traces correspondant à tous ses chemins.

Exercice



Exercice 1

1. Donner un chemin de l'automate ci-contre
2. Pouvez-vous caractériser l'ensemble des chemins ?
3. Donner une trace
4. Caractériser l'ensemble des traces
5. En général, y a-t-il plus de traces que de chemins ?

Propriétés linéaires

Les propriétés les plus simples à vérifier pour un ST sont donc celles qui caractérisent son ensemble de traces.

Propriété Linéaire

Une propriété linéaire φ sur l'ensemble de propositions atomiques AP est un sous-ensemble de $(2^{AP})^{\omega}$.
$\Rightarrow$ φ **est un ensemble de mots infinis sur l'alphabet** 2^{AP}.

Satisfaction

Un ST C satisfait une propriété φ ($C \models \varphi$) ssi. toutes les traces de C sont dans φ.

Exercice 2

Donner une propriété linéaire pour le ST précédent. Le ST la satisfait-il ?

Sûreté

Les propriétés de **sûreté** (**safety**) sont du type *rien de mauvais ne doit arriver*.

- Un bon exemple est l'absence de blocages.
- Un contre-exemple est toujours basé sur une trace *finie*.

Exercice 3

Pour le ST précédent, donner des traces satisfaisant et ne satisfaisant pas les propriétés de sûreté suivantes :

1. Le système ne passe jamais par un état étiqueté "*lose*"
2. Il n'y a jamais plus de 3 *roll* entre 2 *win*

Sûreté et Invariants

Un exemple de propriété de sûreté est l'*invariance linéaire*.

Invariant

Une propriété linéaire φ est un *invariant* s'il existe une formule *propositionnelle* (vérifiable sur un élément de 2^{Prop}) θ telle que

$$\varphi = \{E_0 E_1 E_2 \ldots \in (2^{AP})^\omega \mid \forall i, E_i \models \theta\}$$

On dit que θ est l'*invariant d'état* associé à φ.

Exercice 4

Identifier l'invariant d'état associé à la propriété 1 de l'exercice précédent.
Proposer un algorithme pour vérifier les propriétés d'invariance linéaire pour un système de transitions quelconque.

Algorithme de vérification d'invariant (Largeur)

Algorithme naïf : vérifier que tous les états atteignables vérifient l'invariant d'état.

$AV := I$; $V := \emptyset$; $ok := True$
tant que $AV \neq \emptyset$ *et* ok **faire**
  soit $s \in AV$
  $AV := AV \setminus \{s\}$
  $V := V \cup \{s\}$
  si $s \models \theta$ **alors**
    pour tous les s' *tel que* $s \longrightarrow s'$ **faire**
      si $s' \notin V \cup AV$ **alors**
        $AV := AV \cup \{s'\}$
      fin
    fin
  sinon
    $ok := False$
  fin
fin
Return ok

Algorithme de vérification d'invariant (Profondeur)

$V := \emptyset$; Pile $P := \varepsilon$
$ok :=$ True
tant que $I \setminus V \neq \emptyset \wedge ok$ **faire**
  soit $s \in I \setminus V$
  visiter(s, P, V, ok)
fin
si ok **alors**
  return(Ok !)
sinon
  return(nOk, reverse(P))
fin

procédure *visiter(s, P, V, ok)*
  push(s, P) ; $V := V \cup \{s\}$
  répéter
    $s' :=$top(P)
    si $s' \not\models \theta$ **alors**
      $ok :=$ *False*
    sinon
      si *Post*$(s') \subseteq V$ **alors**
        pop(P)
      sinon
        soit $s'' \in$ Post$(s') \setminus V$
        push(s'', P)
        $V := V \cup \{s''\}$
      fin
    fin
  jusqu'à $(P = \varepsilon) \vee \neg ok$;
fin

Autres propriétés de sûreté

Toutes les propriétés de sûreté ne sont pas des invariants.

Exemple : *Il n'y a jamais plus de 3 roll entre 2 win.*

Les contre-exemples sont bien tous issus de traces finies, mais cette propriété ne peut pas être exprimée en tant qu'invariant.

Sûreté

Une propriété linéaire φ est une propriété de **sûreté** si et seulement si, pour toute trace $\sigma \in ((2^{AP})^\omega \setminus \varphi$, il existe un préfixe fini $\widehat{\sigma}$ tel que

$$\varphi \cap \{\sigma' \in (2^{AP})^\omega \mid \widehat{\sigma} < \sigma'\} = \emptyset$$

Etant donnée φ, on note $bad(\varphi)$ l'ensemble des préfixes finis $\widehat{\sigma}$ de ses contre-exemples.

Exercice

Exercice 5

1. Prouver formellement que toute propriété d'invariance linéaire est une propriété de sûreté.

2. Soit φ une propriété de sûreté qui n'est pas un invariant. En supposant qu'il existe un automate $\mathcal{A}$ tel que $\mathcal{L}(\mathcal{A}) = \mathit{bad}(\varphi)$, proposer un algorithme permettant de vérifier si un ST donné satisfait φ.

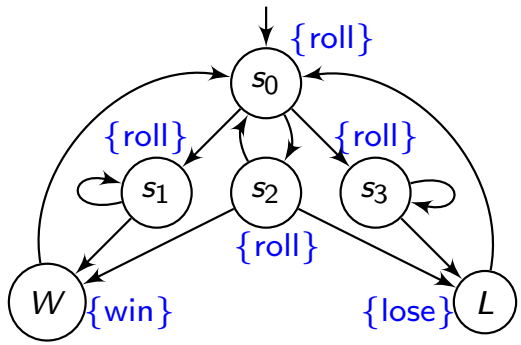
Vivacité

Il est facile de satisfaire une propriété de sûreté : il suffit de ne rien faire.
$\Rightarrow$ On a besoin d'autre chose.

Les propriétés de **vivacité** (**liveness**) sont du type *quelque chose de bien va finir par se passer*.

- Un bon exemple est la terminaison d'un programme.
- Un contre-exemple est toujours une trace *infinie*.

Exemple



- On finit toujours par obtenir un *win*.
- On obtient une infinité de *win* et une infinité de *lose*.

Q : Comment vérifier qu'une telle propriété est satisfaite de manière *automatique* ?

Caractérisation de la vivacité

Vivacité

Une propriété linaire φ est une propriété de *vivacité* si et seulement si préfixes(φ)= $(2^{AP})^*$.

$\Rightarrow$ Tout mot fini peut être prolongé en un mot satisfaisant φ.

Exercice 6

Donner des traces satisfaisant et ne satisfaisant pas les propriétés précédentes :

1. On finit toujours par obtenir un *win*.
2. On obtient une infinité de *win* et une infinité de *lose*.

Sûreté VS Vivacité

- La seule propriété qui est à la fois une propriété de vivacité et une propriété de sûreté est la propriété triviale $(2^{AP})^{\omega}$. (**Exercice 7** : Prouver ce résultat)
- Toute propriété linéaire peut être exprimée comme la conjonction d'une propriété de sûreté et d'une propriété de vivacité. (**Exercice 8 à la maison** : Prouver ce résultat)

Outline

Linear Temporal Logic

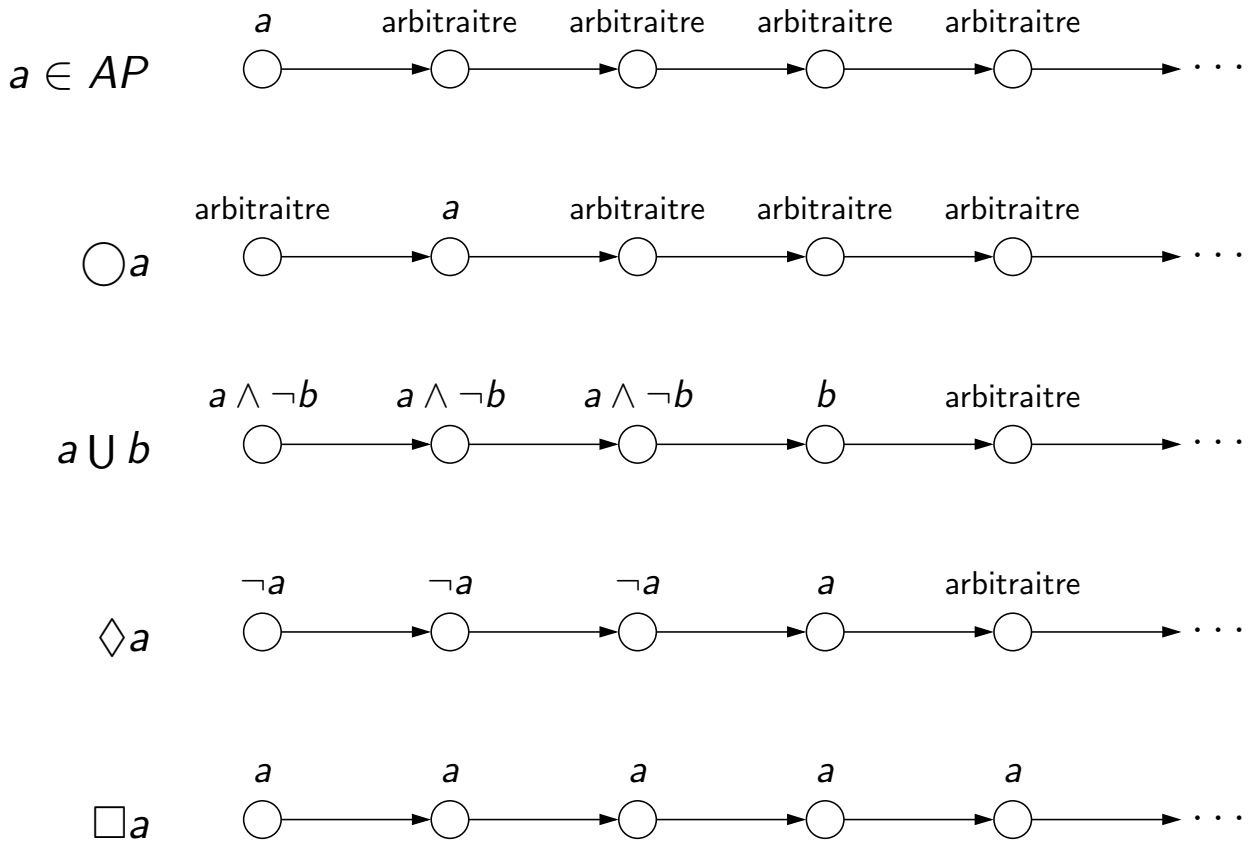
Soient AP un ensemble de propositions atomiques, L une fonction d'étiquetage et $a \in AP$. Les formules LTL sont construites comme suit :

$$\varphi ::= \text{true} \mid a \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \bigcirc\varphi \mid \varphi_1 \,\mathsf{U}\, \varphi_2$$

On utilise aussi les notations suivantes (raccourcis syntaxiques) :

$$\begin{array}{rcll} \varphi_1 \vee \varphi_2 & ::= & \neg(\neg\varphi_1 \wedge \neg\varphi_2) & \\ \varphi_1 \Rightarrow \varphi_2 & ::= & \varphi_2 \vee \neg\varphi_1 & \\ \Diamond\varphi & ::= & \text{true}\,\mathsf{U}\,\varphi & \text{Eventually} \\ \Box\varphi & ::= & \neg\Diamond\neg\varphi & \text{Always} \\ \ldots & ::= & & \end{array}$$

Sémantique (graphique)



Exercice 9

1. Que signifient les formules suivantes ? Proposez une trace les satisfaisant.
   - 1.1 $\varphi_1 ::= \Box\Diamond a$
   - 1.2 $\varphi_2 ::= \Diamond\Box a$
   - 1.3 $\varphi_3 ::= \Box(a \Rightarrow \neg \bigcirc b)$
   - 1.4 $\varphi_4 ::= \Box(a \Rightarrow \bigcirc(a \,\mathsf{U}(b \wedge \bigcirc(b \,\mathsf{U}\, c))))$
2. Traduire les propriétés suivantes en formules de LTL pour $AP = \{a, b, c\}$:

   P_1 : Lorsque a est toujours vraie, b est infiniment souvent vraie.

   P_2 : Lorsque a est vraie dans un état, b ne peut pas être vraie dans le suivant.

   P_3 : c ne peut pas être vraie dans deux états successifs.

   P_4 : On ne peut jamais se trouver dans une configuration où a est vraie et b est fausse.

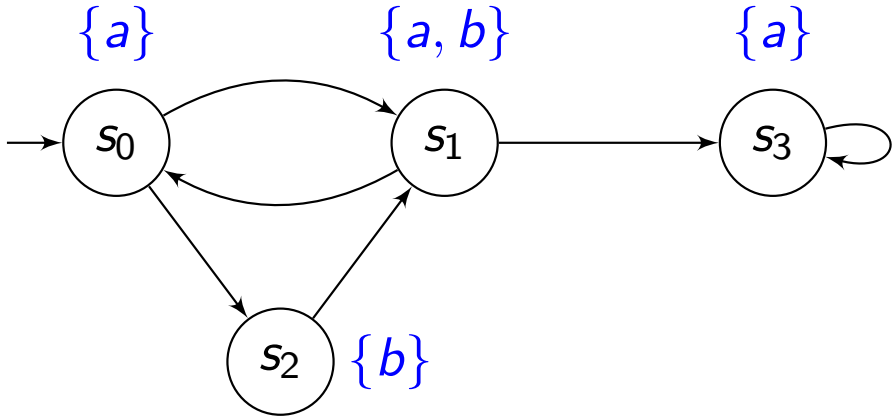
Computation Tree Logic

- Interprétée sur des **arbres d'exécution** plutôt que des traces
- Quantificateurs de chemins :
  - Sur tous les chemins : $\forall$, A
  - Sur au moins un chemin : $\exists$, E

Formule d'état : $\Psi ::= \text{true} \mid a \mid \Psi_1 \wedge \Psi_2 \mid \neg\Psi \mid \exists\varphi \mid \forall\varphi$

Formule de chemin : $\varphi ::= \bigcirc\Psi \mid \Psi_1 \,\mathsf{U}\, \Psi_2 \mid$

Exercice 10



Quels sont les états de l'automate ci-dessus satisfaisant les formules suivantes ?

- $\exists \bigcirc a$
- $\exists \Box a$
- $\exists \Diamond (\exists \Box a)$
- $\exists (a \, \mathsf{U} (\neg a \wedge \forall (\neg a \, \mathsf{U} \, b)))$
- $\forall \bigcirc a$
- $\forall \Box a$
- $\forall (a \, \mathsf{U} \, b)$

Outline

Vérification Probabiliste

Comme la vérification standard,

- Basé sur les traces
- Propriétés d'états / de chemins
- Exhaustif
- Basé sur l'exploration

Mais

- Mesure sur les ensembles de chemins
- Quantification $\rightarrow$ Mesure
- Propriétés *Qualitatives*
  - $\forall \rightarrow \mathbb{P}_{=1}$
  - $\exists \rightarrow \mathbb{P}_{>0}$
- Propriétés *Quantitatives*
  - $\mathbb{P}_{\sim b},\ b \neq 0, 1$

Outline

Probabilistic Computation Tree Logic

Definition (PCTL [Hansson and Jonsson, 1994])

Formules d'états :

$$\Psi ::= \text{true} \mid a \mid \Psi_1 \wedge \Psi_2 \mid \neg\Psi \mid \mathbb{P}_J(\varphi),$$

avec $a \in AP$, φ formule de chemins et $J \subseteq [0,1]$ un intervalle à bornes rationnelles.

Formules de chemins :

$$\varphi ::= \bigcirc\Psi \mid \Psi_1 \,\mathsf{U}\, \Psi_2 \mid \Psi_1 \,\mathsf{U}^{\leq n}\, \Psi_2,$$

avec Ψ, Ψ_1 et Ψ_2 des formules d'états, et $n \in \mathbb{N}$.

Vérification de PCTL [Baier and Katoen, 2008]

Algorithme inductif sur la formule. Soit s un état.

Pour les cas de base $\Psi \equiv \text{true} \mid a \mid \Psi_1 \wedge \Psi_2 \mid \neg\Psi$, identique à CTL.

Pour $\Psi \equiv \mathbb{P}_J(\varphi)$, on a $s \models \Psi$ ssi

$$\Pr(s \models \varphi) = \mu(\{\omega = s \ldots \in S^\omega \mid \omega \models \varphi\}) \in J$$

Nécéssité d'avoir une mesure unique μ sur les chemins.

Vérification de PCTL [Baier and Katoen, 2008]

Algorithme inductif sur la formule. Soit s un état.

Pour les cas de base $\Psi \equiv \text{true} \mid a \mid \Psi_1 \wedge \Psi_2 \mid \neg\Psi$, identique à CTL.

Pour $\Psi \equiv \mathbb{P}_J(\varphi)$, on a $s \models \Psi$ ssi

$$\Pr(s \models \varphi) = \mu(\{\omega = s \ldots \in S^\omega \mid \omega \models \varphi\}) \in J$$

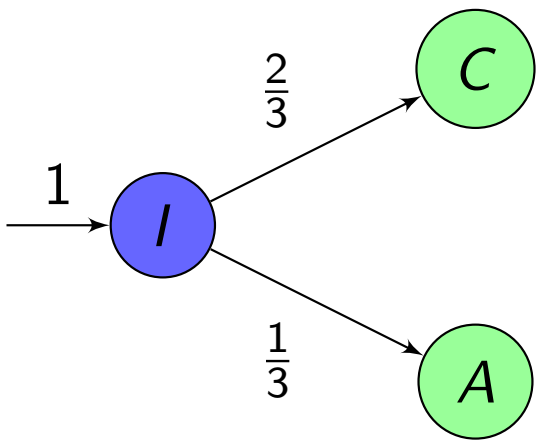
Nécéssité d'avoir une mesure unique μ sur les chemins.

Complexité : Linéaire dans la taille de la formule et polynomial dans la taille du modèle (états)

Mesure du Next

Si $\varphi = \bigcirc \Psi'$, alors

$$\Pr(s \models \bigcirc \Psi') = \sum_{s' \in \mathsf{Sat}(\Psi')} P(s, s').$$



Mesure du Until

Système d'équations linéaires.

1. Identification des états "simples" ($\Pr(s) = 0$ ou 1) $\Rightarrow S_0, S_1$.
2. Réduction du problème aux états d'intérêt $S_?$ $\Rightarrow$ Matrice A.

$$A = (P(s,t))_{s,t \in S_?}$$

3. Probabilité de gagner en 1 coup depuis $S_?$: vecteur b.

$$b = (P(s, S_1))_{s \in S_?}$$

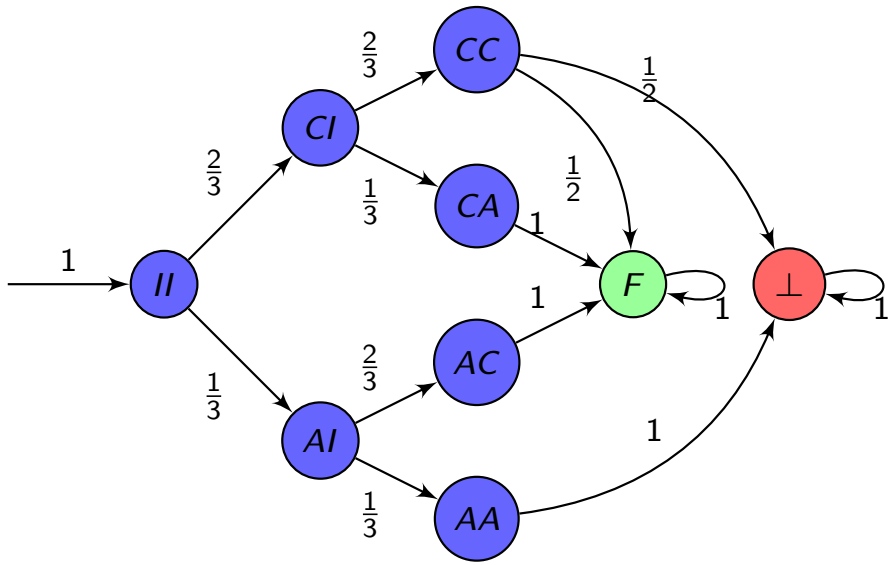
$$\Pr(s \models \Psi_1 \,\mathsf{U}\, \Psi_2) = x(s),$$

avec x le plus petit point fixe de $\Gamma(y) = A \cdot y + b$.

$$\Pr(s \models \Psi_1 \,\mathsf{U}^{\leq n}\, \Psi_2) = x_n(s),$$

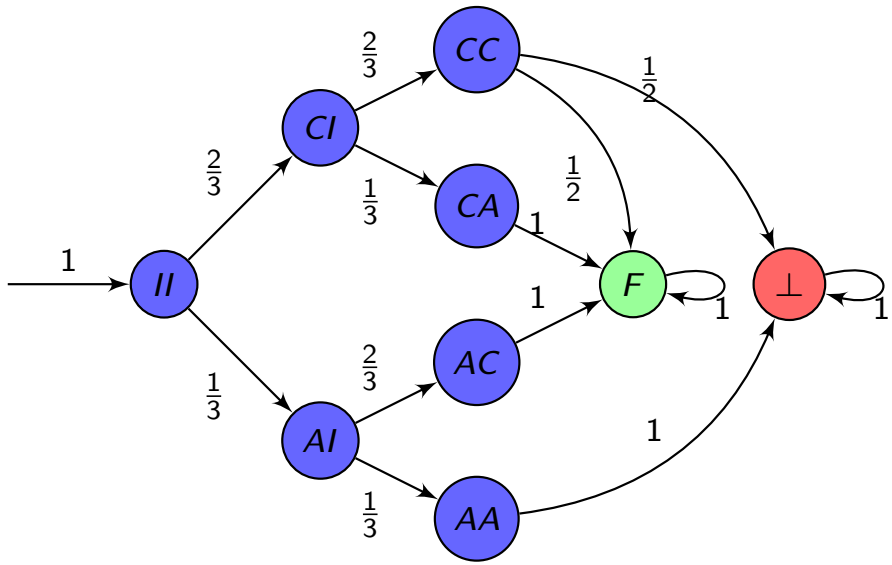
avec $x_n = \Gamma^n(x_0)$ et $x_0 = (0, \ldots 0)$.

Exemple : $\varphi \equiv \mathit{blue}\ \mathsf{U}\ \mathit{green}$



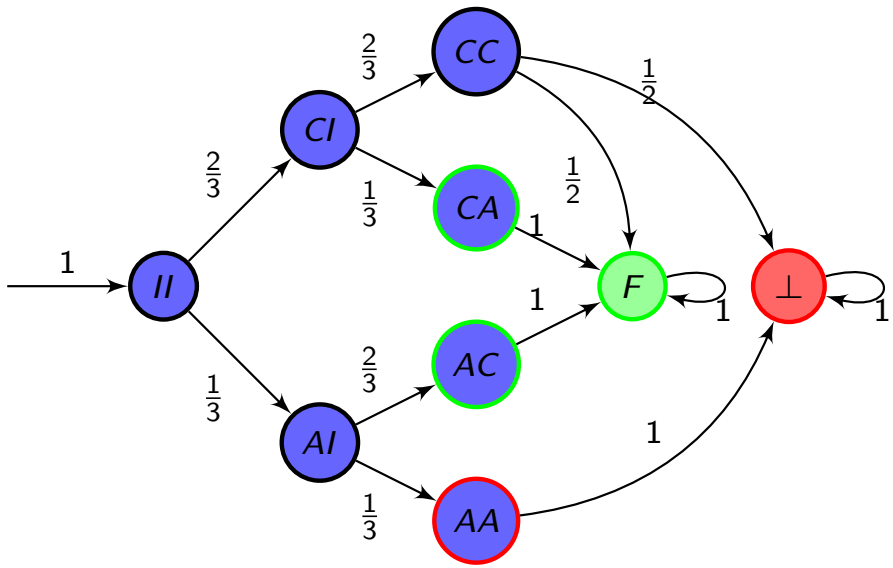
Exemple : $\varphi \equiv \mathit{blue}\ \mathsf{U}\ \mathit{green}$

1. Identifier S_0, S_1 et $S_?$



Exemple : $\varphi \equiv \mathit{blue}\ \mathsf{U}\ \mathit{green}$

1. Identifier S_0, S_1 et $S_?$

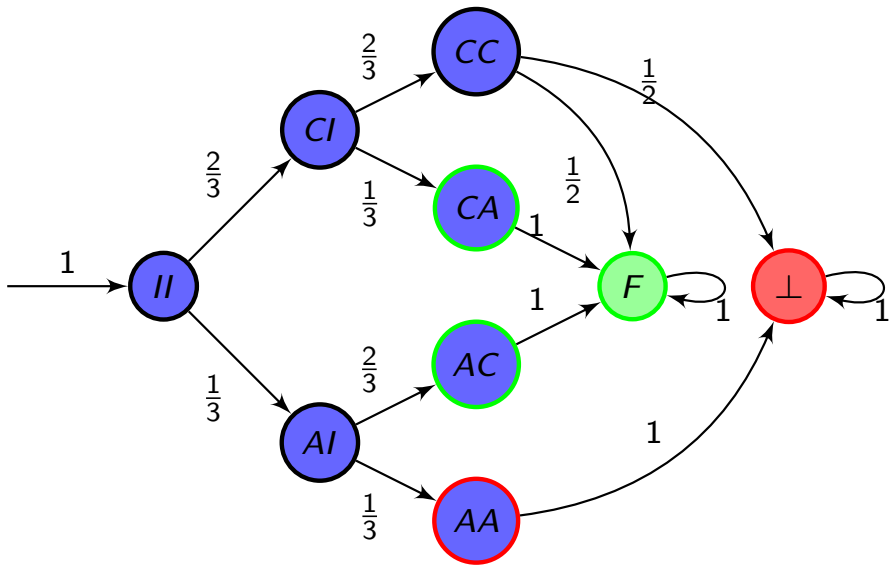


Exemple : $\varphi \equiv \mathit{blue}\ \mathsf{U}\ \mathit{green}$

1. Identifier S_0, S_1 et $S_?$
2. Matrice

$$A = \begin{pmatrix} 0 & \frac{2}{3} & \frac{1}{3} & 0 \\ 0 & 0 & 0 & \frac{2}{3} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

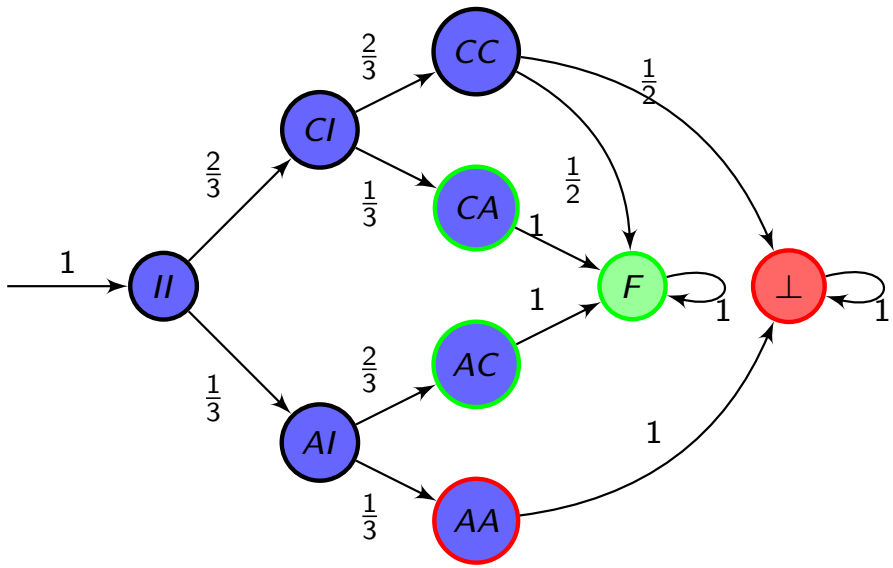
3. Vecteur
   $b = \begin{pmatrix} 0 & \frac{1}{3} & \frac{2}{3} & \frac{1}{2} \end{pmatrix}$



Exemple : $\varphi \equiv \mathit{blue}\ \mathsf{U}\ \mathit{green}$

1. Identifier S_0, S_1 et $S_?$
2. Matrice

$$A = \begin{pmatrix} 0 & \frac{2}{3} & \frac{1}{3} & 0 \\ 0 & 0 & 0 & \frac{2}{3} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$



3. Vecteur
$b = \begin{pmatrix} 0 & \frac{1}{3} & \frac{2}{3} & \frac{1}{2} \end{pmatrix}$

4. Résolution de $\Gamma(y) = A \cdot y + b$:

$$\mathbb{P}(\mathit{II}) = \frac{2}{3}, \mathbb{P}(\mathit{CI}) = \frac{2}{3}, \mathbb{P}(\mathit{AI}) = \frac{2}{3}, \mathbb{P}(\mathit{CC}) = \frac{1}{2}$$

Exercices

Exercice 11

1. Reprendre le modèle permettant de simuler un dé à 6 faces avec des pièces de monnaie et considérer la propriété $\mathbb{P}(\Diamond 1)$.
   - 1.1 Identifier les ensembles S_0, S_1 et $S_?$.
   - 1.2 Construire la matrice A et le vecteur b.
   - 1.3 Résoudre le système linéaire (avec Python par exemple)
2. Faire de même avec le modèle du Craps et la propriété $\mathbb{P}(\Diamond W)$.
3. Calculer la probabilité de $\mathbb{P}(\Diamond^{\leq 10} W)$.
4. Adapter le modèle et la propriété pour calculer la probabilité de gagner sans jamais faire de 6 ou de 8.

Exercice 12

1. Proposer une méthode générique pour identifier les ensembles S_0, S_1 et $S_?$ pour une propriété $\mathbb{P}(\Diamond s)$, où s est un état.
2. Implémenter cette méthode dans votre projet.

PCTL pour les MDP/PA [Baier and Katoen, 2008]

Plusieurs mesures de probabilité $\Rightarrow$ Nécéssité de filtrer sur les adversaires.
On a alors

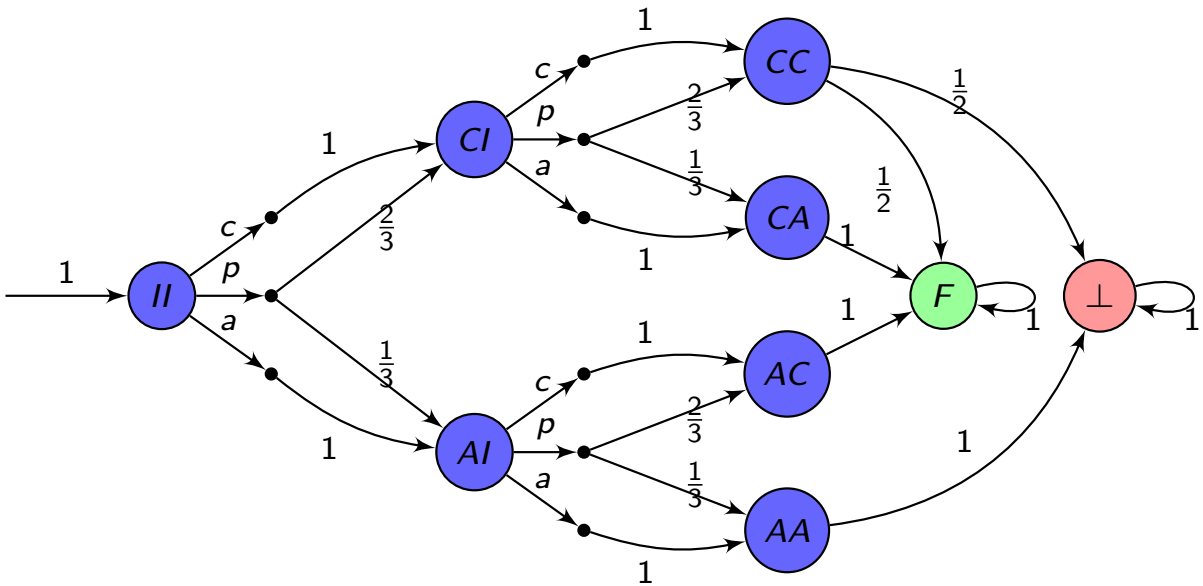
$$\begin{aligned} s \models \mathbb{P}_J^{\max}(\varphi) \Leftrightarrow & \quad \sup_{\mathfrak{S} \in \mathrm{adv}(\mathcal{M})} \Pr^{\mathcal{M}_{\mathfrak{S}}}(s \models \varphi) \in J \\ s \models \mathbb{P}_J^{\min}(\varphi) \Leftrightarrow & \quad \inf_{\mathfrak{S} \in \mathrm{adv}(\mathcal{M})} \Pr^{\mathcal{M}_{\mathfrak{S}}}(s \models \varphi) \in J \end{aligned}$$

Propriété

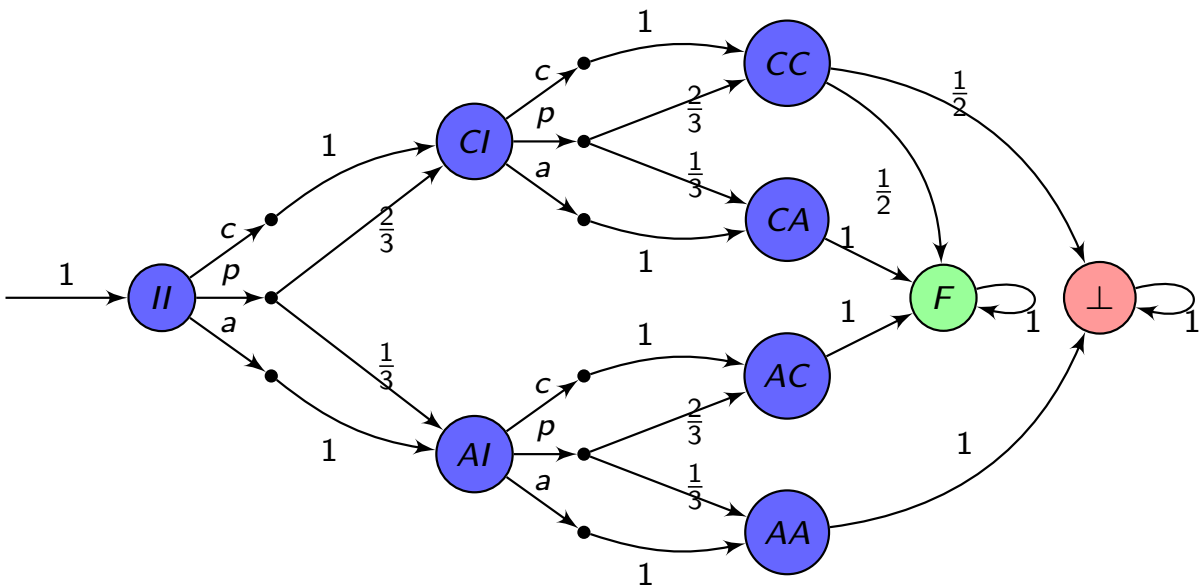
Il suffit de considérer les adversaires **positionnels**

$\Rightarrow$ La vérification de PCTL sur les MDPs à états finis est *décidable*

Exemple

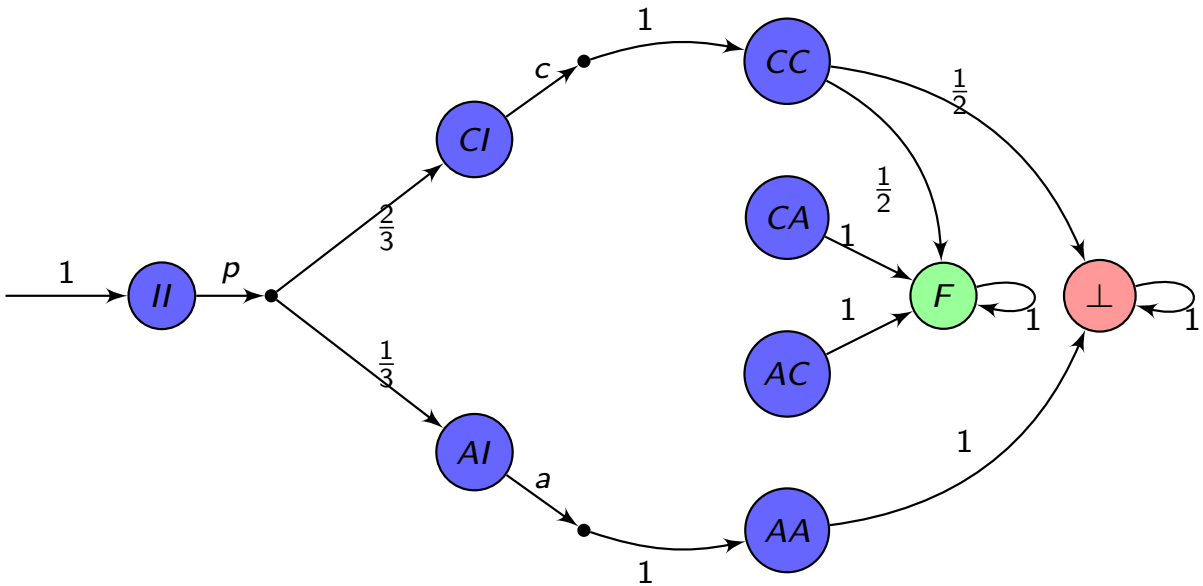


Exemple



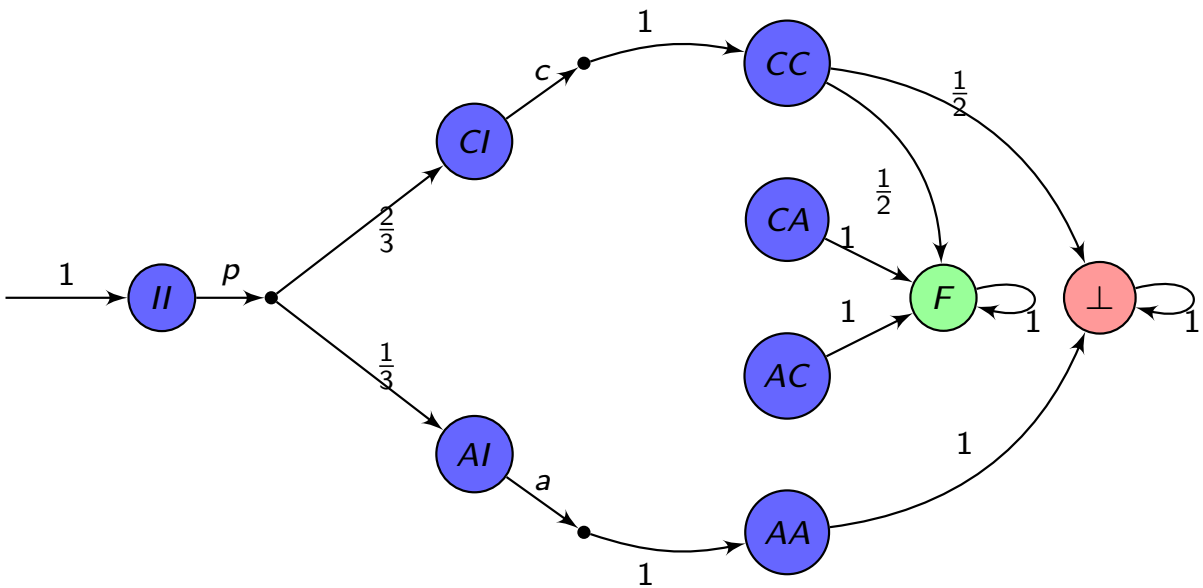
1. Choisir un adversaire $\mathfrak{S}$

Exemple



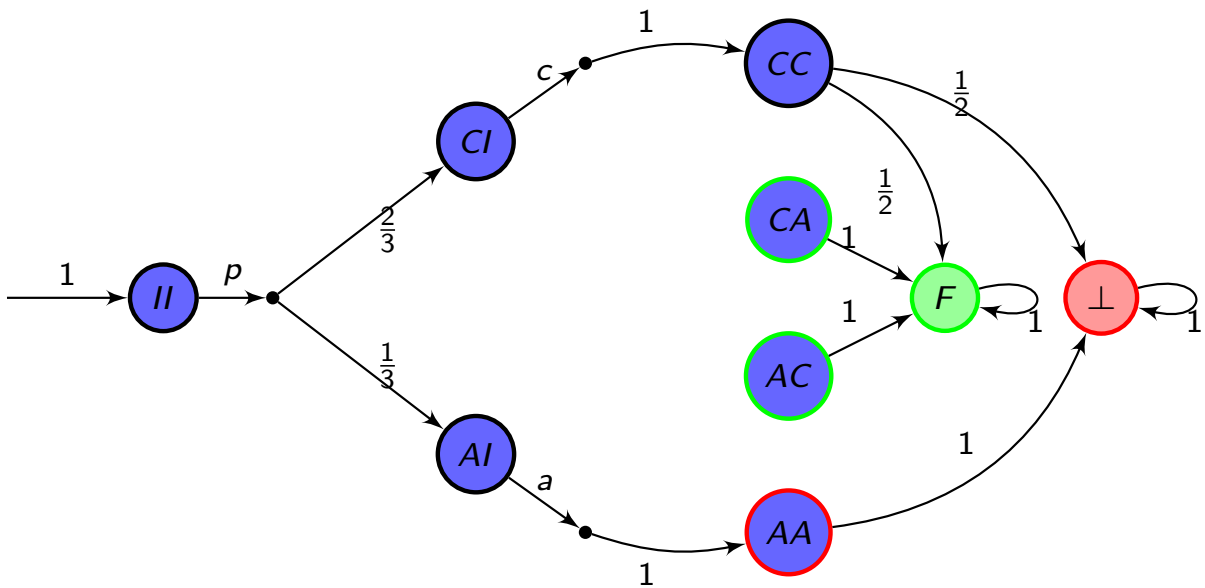
1. Choisir un adversaire $\mathfrak{S}$

Exemple



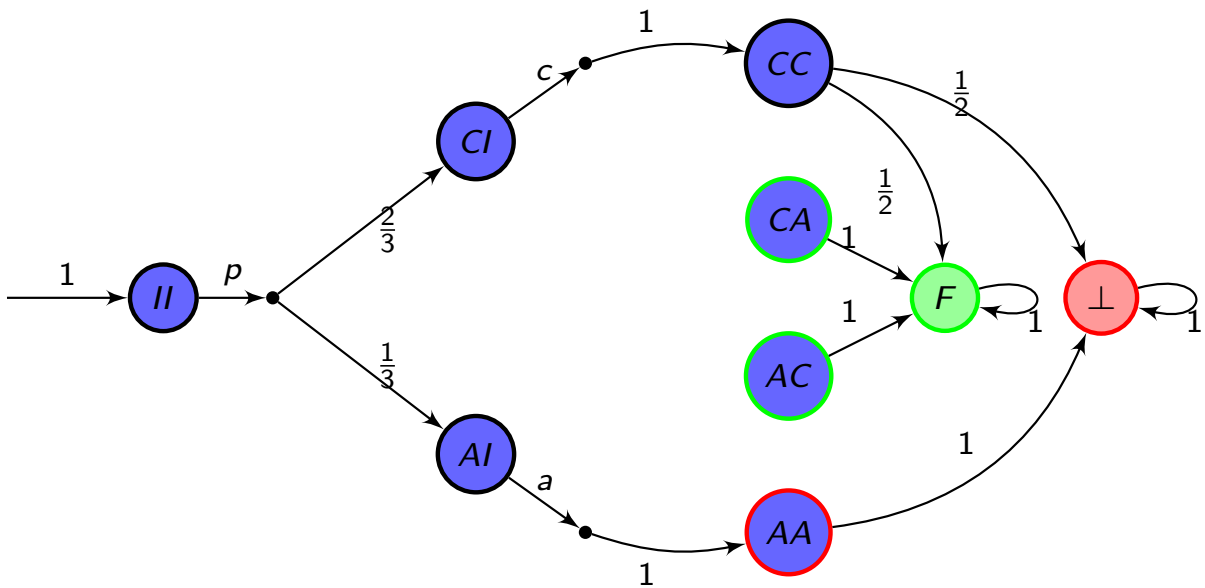
1. Choisir un adversaire $\mathfrak{S}$
2. Identifier S_0, S_1 et $S_?$

Exemple



1. Choisir un adversaire $\mathfrak{S}$
2. Identifier S_0, S_1 et $S_?$

Exemple



1. Choisir un adversaire $\mathfrak{S}$
2. Identifier S_0, S_1 et $S_?$
3. Résoudre le système linéaire

$$\mathbb{P}_{\mathfrak{S}}(II) = \frac{1}{3}, \mathbb{P}_{\mathfrak{S}}(CI) = \frac{1}{2}, \mathbb{P}_{\mathfrak{S}}(AI) = 0, \mathbb{P}_{\mathfrak{S}}(CC) = \frac{1}{2}$$

Programmation linéaire

Calcul de $\mathbb{P}^{max}(\Psi_1 \,\mathsf{U}\, \Psi_2)$

1. Identification des états "simples" ($\mathbb{P}^{max}(\Psi_1 \,\mathsf{U}\, \Psi_2) = 0$ ou 1) $\Rightarrow S_0, S_1$.
2. Réduction du problème aux états d'intérêt $S_?$ $\Rightarrow$ Programme linéaire

$$\forall s \in S_?, a \in Act(s), x_s \geq \sum_{t \in S} P(s,a,t) \cdot x_t,$$

$$\text{avec } \sum_{s \in S} x_s \text{ minimal.}$$

Cela revient à trouver la solution minimale d'une équation $\mathbf{A} \cdot x \geq \mathbf{b}$.

Exercice

Exercice 13

Considérer un MDP $\mathcal{M} = (S, Act, P, \iota_{init}, AP, L)$ et la propriété $\mathbb{P}^{max}(\Diamond s^*)$ avec $s^* \in S$ un état donné.

1. Proposer une définition pour la matrice $\mathbf{A}$ et le vecteur $\mathbf{b}$ du programme linéaire $\mathbf{A} \cdot x \geq \mathbf{b}$.
2. Réfléchir à l'implémentation de cet algorithme de vérification en Python. Quels outils et bibliothèques utiliser ?
3. Implémenter cet algorithme dans votre projet.

Exercice 14

Expliquer comment adapter les algorithmes de vérification pour le calcul de la récompense attendue pour des modèles de récompense Markoviens (MC et MDP). Implémenter ces algorithmes dans votre projet.

Outline

Continuous Stochastic Logic [Baier et al., 2003]

Definition (CSL (transitoire))

Formules d'états :

$$\Psi ::= \text{true} \mid a \mid \Psi_1 \wedge \Psi_2 \mid \neg\Psi \mid \mathbb{P}_J(\varphi),$$

avec $a \in AP$, φ formule de chemins et $J \subseteq [0,1]$ un intervalle à bornes rationnelles.

Formules de chemins :

$$\varphi ::= \bigcirc^I \Psi \mid \Psi_1 \, \mathsf{U}^I \, \Psi_2,$$

avec Ψ, Ψ_1 et Ψ_2 des formules d'états, et $I \subseteq \mathbb{R}_+$ un intervalle à bornes rationnelles.

Et aussi $\mathcal{S}_J(\Psi)$ (à l'équilibre)

Satisfaction des opérateurs $\bigcirc^I$ et U^I (CTMC)

Soit $\pi = s_0 t_0 s_1 t_1 \ldots$ un chemin

- $\pi \models \bigcirc^I \Psi$ ssi $s_1 \models \Psi$ et $t_0 \in I$
- $\pi \models \Psi_1 \, \mathrm{U}^I \, \Psi_2$ ssi il existe $t \in I$ et $k \geq 0$ tel que
  - $t_0 + \ldots + t_{k-1} \leq t < t_0 + \ldots + t_k$
  - $s_k \models \Psi_2$
  - $s_i \models \Psi_1$ pour $0 \leq i \leq k-1$ (ou k si $t_0 + \ldots + t_{k-1} < t$)

Vérification de CSL sur les CTMC

1. Transformation de la CTMC en une DTMC *uniformisée*
2. Utilisation des probabilités de Poisson
3. Calcul des probabilités transitoires à l'aide d'une somme infinie
4. Complexité
   - Linéaire par rapport à la formule
   - Polynomial dans le nombre d'états
   - Linéaire par raport à la plus grande constante temporelle de la formule
   - Linéaire par rapport à l'élément le plus grand de la matrice génératrice

Voir [Baier et al., 2003, Katoen et al., 2001, Baier et al., 2000] pour les détails.

Extension aux PTA [Kwiatkowska et al., 2002]

Ajout de non-déterminisme, horloges explicites, syntaxe différente.

CSL n'est pas la logique la plus appropriée (voire PTCTL), mais fonctionne.

Graphe des régions = MDP à états finis

1. Construction du graphe des régions
2. Transformation/Adaptation de la formule (si besoin)
3. Vérification de PCTL dans ce cadre (Adversaires min et max)

Complexité : Algorithmes largement exponentiels mais améliorations possibles

Mais aussi

- Autres logiques (PTCTL, CSLTA, ...)
- Autres modèles (Automates Stochastiques, réseaux d'Automates, ...)
- Vérification des propriétés à l'équilibre
- ...

Outline

Outils

- PRISM [Kwiatkowska et al., 2011]
- UPPAAL [Larsen et al., 1997]
- MRMC [Katoen et al., 2005]
- SPIN [Holzmann, 1997]
- ...

Outline

Motivation

Le model-checking probabiliste souffre des défauts du model-checking standard :

- Indécidabilité pour modèles complexes (Hybrides, paramétrés, etc.)
- Explosion de l'espace d'états
- Ressources nécéssaires (temps, mémoire, etc.)

$\Rightarrow$ Solution approchée : **Model checking statistique (SMC)**
[Younes and Simmons, 2002, Legay et al., 2010]

- Efficace (la complexité ne dépend pas de la taille du modèle)
- Précision contrôlée
- Vérification de modèles indécidables
- Non limité par les logiques
- Non limité par les modèles

SMC : But

But : Estimer la *probabilité* qu'un modèle M *satisfasse une propriété* φ à partir *d'échantillons*.

Echantillons

Traces / Exécutions du modèle

Probabilité

Requiert une mesure de probabilité sur l'espace des échantillons

Satisfaction d'une propriété

- Vérifiable sur un échantillon
  - Borné
  - Linéaire
  - Observable
- Pas nécessairement de logique

$+$ précision, erreur à maîtriser

Attention Uniquement si le modèle est purement probabiliste

Principes du SMC

1. Utiliser le modèle pour générer un échantillon de traces
   $\Rightarrow$ Estimateur de la mesure réelle

2. Mesurer la probabilité de satisfaire la propriété sur cet échantillon
   $\Rightarrow$ Estimation de la probabilité réelle

3. Généraliser l'estimation au modèle initial
   $\Rightarrow$ Précision, taux d'erreur en fonction de la taille de l'échantillon

Types de propriétés

2 types principaux : Propriétés *qualitatives* ou *quantitatives*

- **Quantitatif** : Réponse numérique
  Ex : Calculer $\mathbb{P}(M \models \varphi)$

- **Qualitatif** : Réponse Vrai/Faux
  Ex : $\mathbb{P}(M \models \varphi) > \theta$

Pas besoin de calculer $\mathbb{P}(M \models \varphi)$ pour le problème qualitatif, il existe des algorithmes plus efficaces.

Outline

SMC Quantitatif

Données : Modèle M, propriété φ, précision ε, taux d'erreur δ
But : Estimer la probabilité γ avec laquelle M satisfait φ

Monte Carlo [Robert, 2004]

Expérience : Générer une simulation de M et vérifier si elle satisfait φ

- Variable aléatoire Z, réalisation $z_i = 1$ ssi la i^e expérience est un succès

$\Rightarrow$ variable de Bernouilli de paramètre γ

$$\lim_{N \to \infty} \frac{\sum_{i=1}^{N} z_i}{N} = \gamma$$

Précision et taux d'erreur

Soit $\gamma_N = \frac{\sum_{i=1}^{N} Z_i}{N}$ (moyenne après N essais)

Bornes de Chernoff-Hoeffding [Hoeffding, 1963]

Si $\delta, \varepsilon \in [0, 1]$, alors

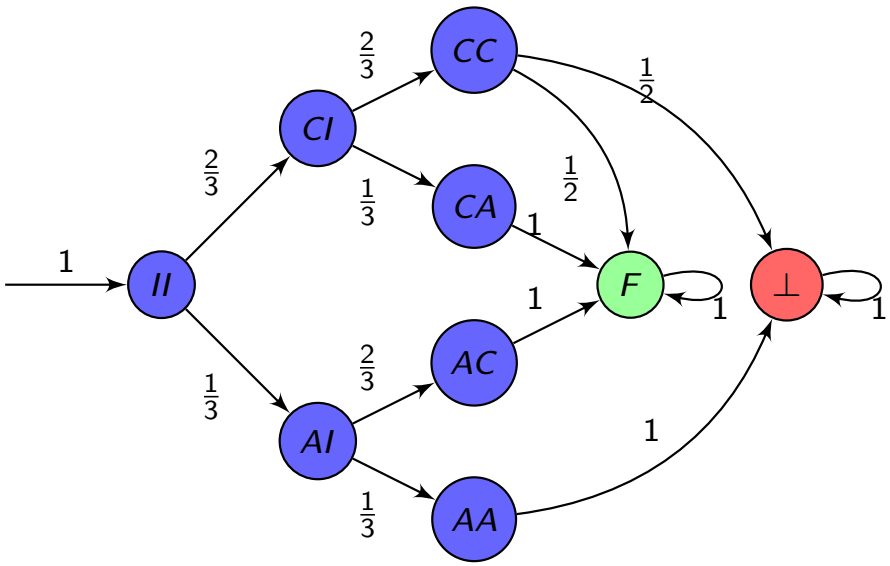
$$N \geq \frac{\ln(2) - \ln(\delta)}{(2\varepsilon)^2} \qquad \Rightarrow \qquad \mathbb{P}(|\gamma_N - \gamma| \geq \varepsilon) \leq \delta$$

Précision ε et taux d'erreur δ guarantis si $N \geq \frac{\ln(2)-\ln(\delta)}{(2\varepsilon)^2}$

Exemple

- $\varphi = \textit{bleu}\ \mathsf{U}^{\leq 5}\ \textit{vert}$
- Précision $\varepsilon = 0.01$
- Erreur $\delta = 0.01$

$\Rightarrow$ $N \geq 13246$



Exercices

Exercice 15

1. Proposer un programme Python qui met en oeuvre les lancers de pièces successifs pour simuler un dé à 6 faces.
2. Implémenter l'algorithme de SMC quantitatif utilisant Monte Carlo dans ce programme.
3. Calculer, par SMC quantitatif, la probabilité d'obtenir chaque chiffre $(1, 2, 3, 4, 5, 6)$ après 5 lancers au maximum. Comparer à la valeur théorique.
   Utiliser un taux d'erreur de 0.05 et une précision de 0.01.

Exercice 16

1. Implémenter l'algorithme de SMC quantitatif dans votre projet de TP.
2. Appliquer cet algo au modèle des lancers de pièces pour simuler un dé à 6 faces.
3. Vérifier que vous obtenez les mêmes résultats que pour l'exercice précédent.

Outline

Test d'Hypothèses [Younes, 2005a]

Soit γ la probabilité avec laquelle M satisfait φ (inconnue)

Le test d'hypothèse a pour but de comparer γ avec une borne θ **sans calculer** γ

Test d'Hypothèse

But : confronter 2 hypothèses

- $H : \gamma \geq \theta$
- $K : \gamma < \theta$

Caractéristique : la **force**, déterminée par 2 paramètres α et β

- α : borne sur l'*erreur de type I* (K acceptée alors que H est vraie)
- β : borne sur l'*erreur de type II* (H acceptée alors que K est vraie)

Problème : Impossible d'optimiser *à la fois* α et β

Région d'indifférence

Afin de garantir des valeurs faibles pour α et β, on utilise une région d'indifférence $[\gamma_1, \gamma_0]$ qui contient θ.
On confronte alors les hypothèses

$$H_0 : \gamma \geq \gamma_0 \qquad \text{VS} \qquad H_1 : \gamma < \gamma_1$$

Si $\gamma \in [\gamma_1, \gamma_0]$ alors le résultat n'a pas d'importance.

En général, on fixe une **précision** ε et on prend $\gamma_1 = \theta - \varepsilon$ et $\gamma_0 = \theta + \varepsilon$.

Résolution du test d'hypothèses

De nombreuses méthodes existent

- Méthodes "statiques" : le nombre d'échantillons est pré-calculé en fonction des paramètres $\alpha, \beta, \varepsilon$
  Ex : Single Sampling Plan $\rightarrow (n, c)$ tel que H_0 est accepté si au moins c échantillons sur n satisfont φ
- Méthodes "dynamiques" : le nombre d'échantillons s'adapte. Critère d'arrêt fixé en fonction des paramètres
  Ex : Sequential Probability Ratio Test

Sequential Probability Ratio Test [Wald, 1945]

Une valeur R_m représentative du ratio de simulations "acceptées" d_m sur le nombre total de simulations m est mise à jour à chaque simulation et comparée à des bornes A et B.

$$R_m = \frac{\gamma_1^{d_m} \cdot (1-\gamma_1)^{m-d_m}}{\gamma_0^{d_m} \cdot (1-\gamma_0)^{m-d_m}},$$

- Dès que $R_m \geq A$, on peut accepter H_1
- Dès que $R_m \leq B$, on peut accepter H_0

Problème : Lier A et B aux paramètres α, β

Sequential Probability Ratio Test [Wald, 1945]

Une valeur R_m représentative du ratio de simulations "acceptées" d_m sur le nombre total de simulations m est mise à jour à chaque simulation et comparée à des bornes A et B.

$$R_m = \frac{\gamma_1^{d_m} \cdot (1-\gamma_1)^{m-d_m}}{\gamma_0^{d_m} \cdot (1-\gamma_0)^{m-d_m}},$$

- Dès que $R_m \geq A$, on peut accepter H_1
- Dès que $R_m \leq B$, on peut accepter H_0

Problème : Lier A et B aux paramètres α, β

En pratique : $A = \frac{(1-\beta)}{\alpha}$ et $B = \frac{\beta}{(1-\alpha)}$ garantissent $\alpha' + \beta' < \alpha + \beta$

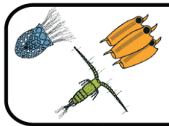
Exercices

Exercice 17

1. Implémenter l'algorithme de SMC qualitatif utilisant SPRT dans le programme de l'exercice 15.
2. On appelle p_i^k la probabilité d'obtenir le chiffre i après k lancers au maximum. Utiliser votre programme pour estimer $\mathbb{P}(p_i^k \geq 0.1), \mathbb{P}(p_i^k \geq 0.14), \mathbb{P}(p_i^k \geq 0.16)$. Qu'observez-vous ?
   Utiliser $\alpha = \beta = 0.01$.
3. Implémenter cet algorithme dans votre projet de TP et le tester sur le modèle du CRAPS.
4. Que faire si le modèle est un MDP ?

Exemple d'application 1/2

Filtration



$$F = \frac{1}{[P] + k_f} . f_{max} . CM^{b_f} . {t_{10f}}^T$$

Ingestion

$$I = F.[P]$$

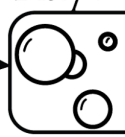
Assimilation

$$A = I.(1 - a_{max} \frac{[P]}{[P] + k_a})$$

Growth

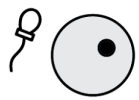
$$\Delta G = A - (R + Ex + Re)$$

Respiration



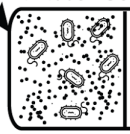
$$R = [R_o . CM^{b_r} . {t_{10r}}^T] . \alpha . \beta$$

Reproduction



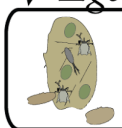
$$Re = [(a_{re} . CM^{b_{re}}) . W_e] . c_{re} . spn$$

Excretion



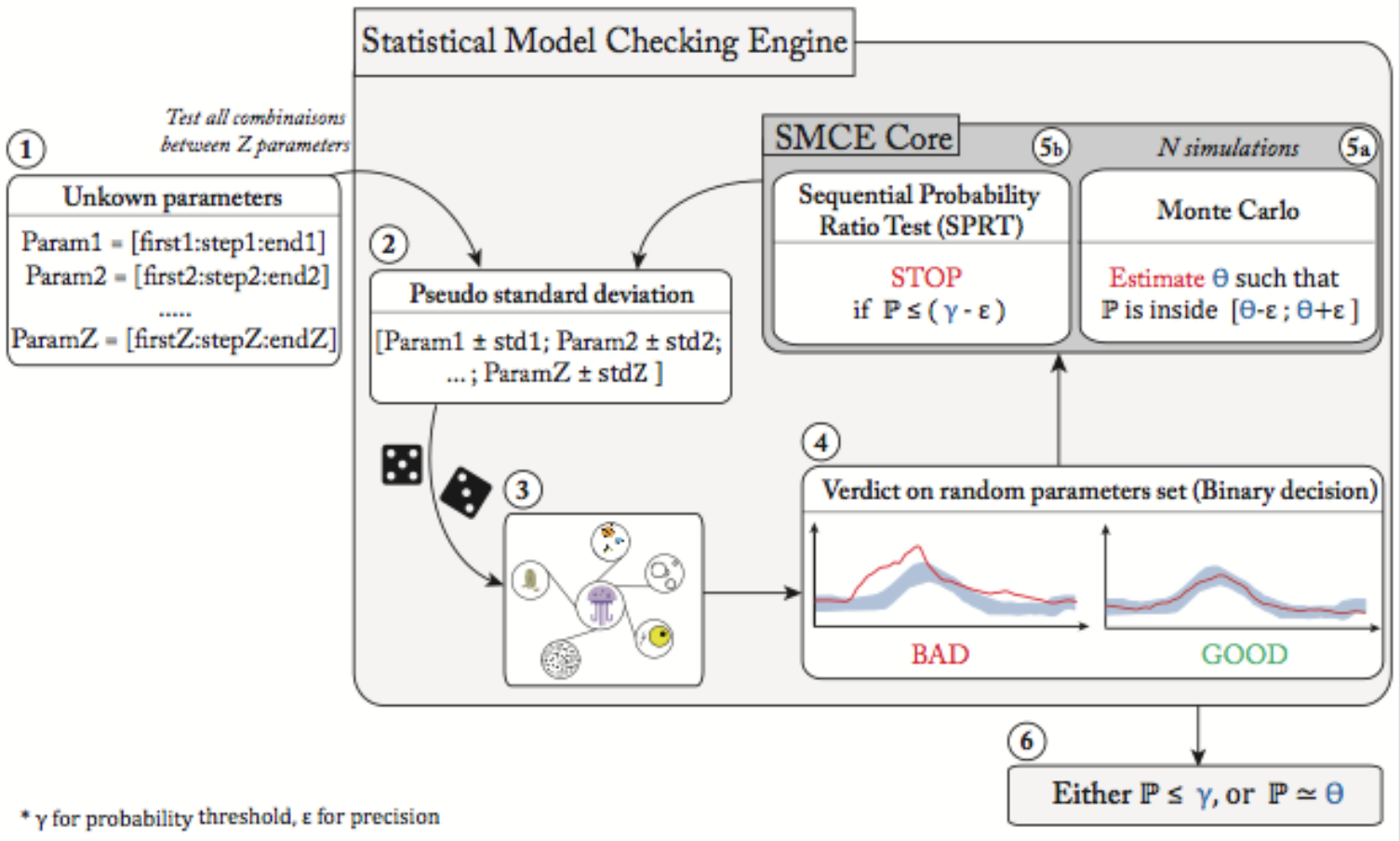
$$Ex = R . c_e$$

Egestion



$$Eg = I - A$$

Exemple d'application 2/2



Outline

- Uppaal SMC (Networks of PTA) [Larsen et al., 1997]
- PRISM (DTMC, CTMC, PTA) [Kwiatkowska et al., 2011]
- Ymer (Generalized Semi-Markov Processes) [Younes, 2005b]
- COSMOS (Linear Hybrid Automata) [Ballarini et al., 2015]
- VESTA (DTMC, CTMC) [Sen et al., 2005]
- PLASMA-LAB (Générique) [Boyer et al., 2013]

References I


Baier, C., Haverkort, B., Hermanns, H., and Katoen, J.-P. (2003).
Model-checking algorithms for continuous-time markov chains.
IEEE Transactions on software engineering, 29(6) :524–541.


Baier, C., Haverkort, B. R., Hermanns, H., and Katoen, J.-P. (2000).
Model checking continuous-time markov chains by transient analysis.
In *CAV*, volume 1855, pages 358–372. Springer.


Baier, C. and Katoen, J.-P. (2008).
Principles of Model Checking.
The MIT Press.


Ballarini, P., Barbot, B., Duflot, M., Haddad, S., and Pekergin, N. (2015).
HASL : A new approach for performance evaluation and model checking from concepts to experimentation.
Performance Evaluation, 90 :53–77.


Boyer, B., Corre, K., Legay, A., and Sedwards, S. (2013).
Plasma-lab : A flexible, distributable statistical model checking library.
In *International Conference on Quantitative Evaluation of Systems*, pages 160–164. Springer.

References II

Hansson, H. and Jonsson, B. (1994).
A logic for reasoning about time and reliability.
Formal aspects of computing, 6(5).

Hoeffding, W. (1963).
Probability inequalities for sums of bounded random variables.
Journal of the American statistical association, 58(301) :13–30.

Holzmann, G. J. (1997).
The model checker spin.
IEEE Transactions on software engineering, 23(5) :279–295.

Katoen, J.-P., Khattri, M., and Zapreevt, I. (2005).
A markov reward model checker.
In *Quantitative Evaluation of Systems, 2005. Second International Conference on the*, pages 243–244. IEEE.

Katoen, J.-P., Kwiatkowska, M., Norman, G., and Parker, D. (2001).
Faster and symbolic CTMC model checking.
In *Process Algebra and Probabilistic Methods. Performance Modelling and Verification*. Springer.

References III


Kwiatkowska, M., Norman, G., and Parker, D. (2011).
Prism 4.0 : Verification of probabilistic real-time systems.
In *Computer aided verification*, pages 585–591. Springer.


Kwiatkowska, M., Norman, G., Segala, R., and Sproston, J. (2002).
Automatic verification of real-time systems with discrete probability distributions.
Theoretical Computer Science, 282(1) :101–150.


Larsen, K. G., Pettersson, P., and Yi, W. (1997).
Uppaal in a nutshell.
International Journal on Software Tools for Technology Transfer (STTT), 1(1) :134–152.


Legay, A., Delahaye, B., and Bensalem, S. (2010).
Statistical model checking : An overview.
In *Proc. Runtime Verification - First International Conference, RV 2010, St. Julians, Malta, November 1-4, 2010. Proceedings*, volume 6418 of *Lecture Notes in Computer Science*, pages 122–135. Springer.


Robert, C. P. (2004).
Monte carlo methods.
Wiley Online Library.

References IV


Sen, K., Viswanathan, M., and Agha, G. (2005).
Vesta : A statistical model-checker and analyzer for probabilistic systems.
In *Quantitative Evaluation of Systems, 2005. Second International Conference on the*, pages 251–252. IEEE.


Wald, A. (1945).
Sequential tests of statistical hypotheses.
The Annals of Mathematical Statistics, 16(2) :117–186.


Younes, H. L. (2005a).
Verification and planning for stochastic processes with asynchronous events.
Technical report, Ph.D. thesis, Carnegie Mellon.


Younes, H. L. (2005b).
Ymer : A statistical model checker.
In *CAV*, volume 3576, pages 429–433. Springer.


Younes, H. L. and Simmons, R. G. (2002).
Probabilistic verification of discrete event systems using acceptance sampling.
In *CAV*, volume 2, pages 223–235. Springer.