Modélisation Probabiliste et Apprentissage par Renforcement

Benoît Delahaye

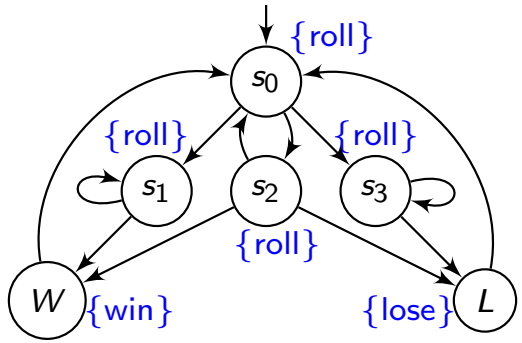
Nantes Université, LS2N UMR 6004

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.

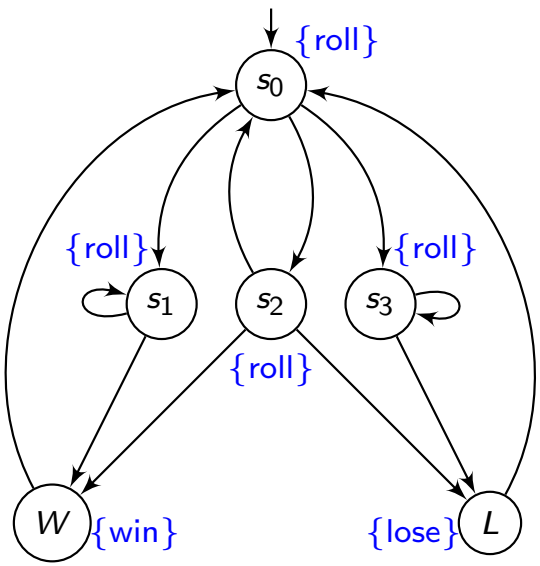
Les traces sont les projections des chemins sur les étiquetages.

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.

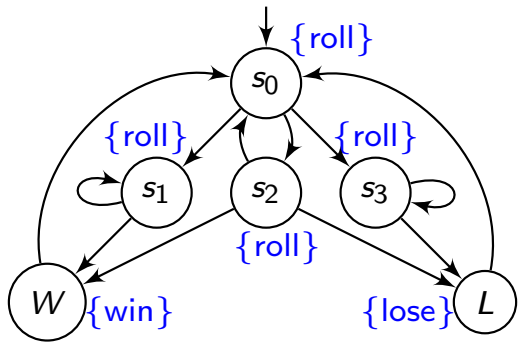
Vivacité

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

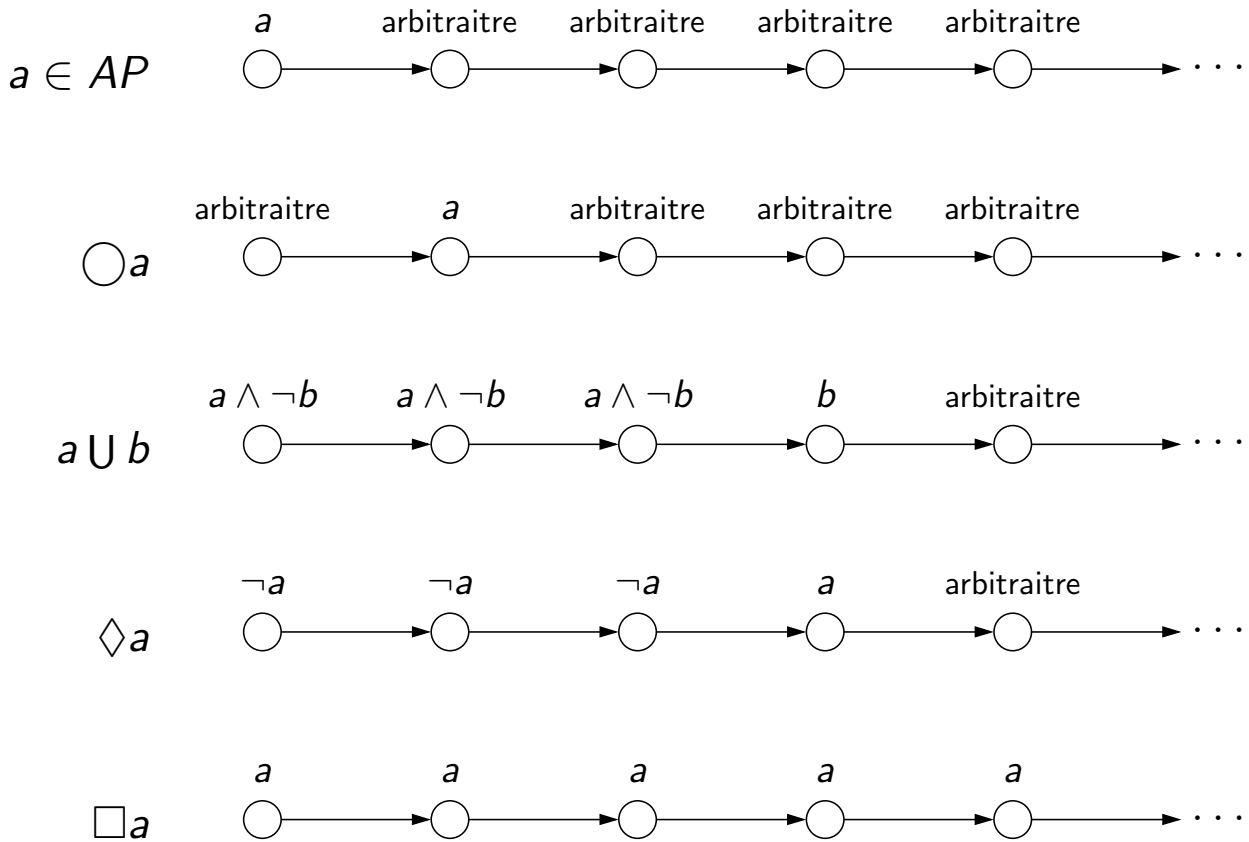
Soient AP un ensemble de propositions atomiques, L une fonction d'étiquetage et $a \in AP$. Les formules LTL sont construites comme suit :

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

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.

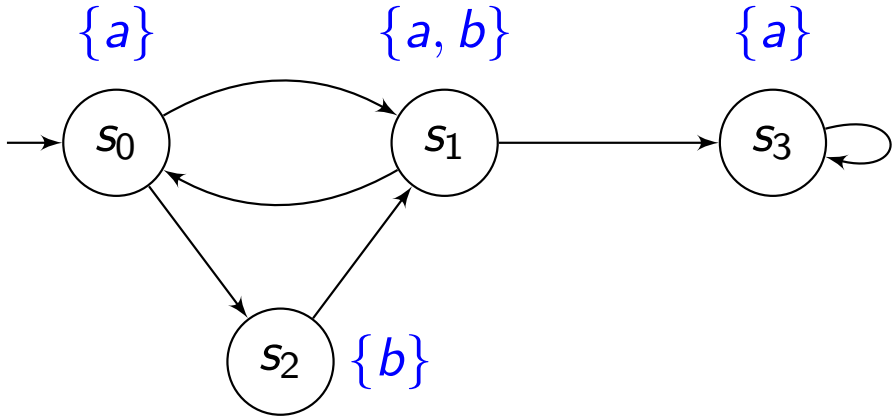
Computation Tree Logic

- 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 $n \in \mathbb{N}$.

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.