Modélisation de programmes probabilistes la bibliothèque ALEA en COQ

Christine Paulin-Mohring

contributions de Philippe Audebaud, David Baelde, Pierre Courtieu
projet ANR SCALP (2007-2011)

Université Paris-Sud, INRIA Saclay - Île-de-France

Janvier 2014

Motivations

- Comportements probabilistes présents dans de nombreuses situations
  - modélisation d'environnements
  - efficacité des algorithmes
  - équité, anonymat dans les protocoles
  - analyse de sécurité
- Analyse probabiliste délicate
  - des outils formels pour raisonner sur des probabilités

Organisation du cours

1. Représenter des programmes aléatoires
   - Principes généraux
2. Formalisation dans la bibliothèque ALEA
   - Mise en œuvre en COQ

Cours 1 : Représenter des programmes aléatoires

1. Introduction
   - Quelques exemples
   - Éléments de probabilités
2. Représentations de programmes aléatoires
   - Point de vue opérationnel
   - Programmes vus comme des lois de probabilité
3. Raisonner sur des programmes aléatoires
   - Logique de Hoare probabiliste
   - Terminaison probabiliste

Quelques usages des probabilités

- Algorithmes approchés (Monte-Carlo)
  - algorithmes simples (expression, complexité)
  - solutions approchées
- Algorithmes avec comportement probabiliste (Las Vegas)
  - meilleure performance moyenne
  - casser la symétrie dans les programmes distribués
- Sécurité
  - Anonymat
  - Analyse des risques d'attaques

Algorithmes approchés *simples* : primalité

Test de primalité de Rabin-Miller d'un entier p

- r et s impair tels que $p-1=2^r s$
- a quelconque $0<a<p$
- lorsque p est premier :
  1. $a^{p-1}\equiv 1(p)$ (test de Fermat)
  2. si $a^{2^j s}\equiv 1(p)$ avec $0<j\leq r$ alors $a^{2^{j-1}s}\equiv \pm 1(p)$
- si p n'est pas premier : $|\{0<a<p|test(a,p)\}|\leq \frac{(p-1)}{2}$
- choisir a de manière aléatoire
  - si le test échoue, p n'est pas premier
  - si le test réussit, la probabilité que p soit premier est au moins $\frac{1}{2}$
  - répéter le test et combiner les résultats

Algorithmes approchés *simples* : graphes

Coupure minimale dans un multi-graphe non orienté

- partition des nœuds en 2 ensembles liés par un nombre minimal d'arêtes.

Algorithme

graphe avec n nœuds :

- contracter le graphe selon une arête (A, B) choisie aléatoirement
  - les boucles sont supprimées
- recommencer $n-2$ fois : il reste deux sommets X et Y
- la partition est donnée par les images inverses de X et Y

Algorithmes approchés *simples* : graphes

Analyse

soit $\mathcal{C}$ les arêtes d'une coupure minimale,

- si $(A, B) \notin \mathcal{C}$, alors $\mathcal{C}$ coupure minimale du graphe contracté
- si $|\mathcal{C}| = k$ alors tous les nœuds sont au moins de degré k
  $\rightarrow$ au moins $\frac{mk}{2}$ arêtes pour m nœuds
- probabilité de ne pas choisir une arête de $\mathcal{C}$ si m nœuds
  $\rightarrow$ $\geq 1 - \frac{2}{m} = \frac{m-2}{m}$
- probabilité de préserver $\mathcal{C}$ lors des $n-2$ étapes :
  $\rightarrow$ $\geq \frac{n-2}{n} \times \frac{n-3}{n-1} \times \frac{n-4}{n-2} \times \cdots \times \frac{2}{4} \times \frac{1}{3} = \frac{2}{n(n-1)} \geq \frac{2}{n^2}$
- exécuter cet algorithme de l'ordre de $\frac{n^2}{2}$ fois
- garder à chaque fois la coupure la plus petite
- probabilité d'erreur $\leq 1/e$

Amélioration de la complexité

- L'algorithme reste exact
- L'exécution est aléatoire, choix aléatoire de certains éléments de l'algorithme
- Quicksort pour trier n éléments
  - le choix du pivot influe sur la complexité
  - chaque sous-partie doit rester de taille au moins $\frac{n}{4}$
  - il y a $\frac{n}{2}$ pivots acceptables
  - choisir le pivot de manière aléatoire
  - p_{ij} probabilité de comparer les éléments de rang i et j
  - pas de comparaison si i et j sont séparés par un pivot, c'est-à-dire qu'un élément de rang k avec $i < k < j$ a été choisi avant $p_{ij} = \frac{2}{j-i+1}$
  - le nombre total de comparaisons est inférieur à $2n\Sigma_{k=1}^{n} \frac{1}{k}$

Algorithme distribué : consensus

Se mettre d'accord sur un choix de manière distribuée

- solutions déterministes : nombre d'étapes proportionnel au nombre de participants n
- casser la symétrie

Algorithme de Rabin

- n processus, le processus k a une mémoire p_k entière
- deux résultats possibles, associés à des mémoires r_0 et r_1 qui peuvent contenir un entier, variables $\texttt{ok}_i, i = 0, 1$
- initialisation à 0 des mémoires, $\texttt{ok}_i = \texttt{false}$
- chaque processus k choisit aléatoirement i par lequel il commence

Algorithme de consensus

Étape élémentaire : le processus k “visite” i

- si $\texttt{ok}_i$ alors k enregistre le choix i et s’arrête
- sinon
  - si $p_k < r_i$ alors $p_k := r_i$ et k visite $1 - i$
  - si $p_k > r_i$ alors $\texttt{ok}_i := \mathit{true}$, k enregistre le choix i et s’arrête
  - si $p_k = r_i$ alors soit n pair immédiatement plus grand que p_k,
    - k choisit aléatoirement entre n et $n + 1$
    - il met cette valeur dans r_i et p_k
    - k visite ensuite $1 - i$

Analyse

- A chaque étape :
  - ensemble de processus $\mathtt{out}_i$ qui visite i
  - ensemble de processus $\mathtt{in}_i$ qui a choisi i
- Invariants
  - $k \in \mathtt{out}_i \cup \mathtt{in}_i \Rightarrow p_k \leq r_{1-i}$
  - si $\mathtt{in}_i \neq \emptyset$ alors $\mathtt{max}(p_k | k \in \mathtt{in}_i) > r_i$
- correction : le programme s'arrête lorsque $\mathtt{out}_0 = \mathtt{out}_1 = \emptyset$ auquel cas forcément l'un des deux ensembles $\mathtt{in}_0, \mathtt{in}_1$ est vide, tout le monde a fait le même choix
- terminaison :
  - tant que le choix n'est pas fait, il y a des écritures sur r_0 ou r_1
  - terminaison avec un variant probabiliste (décroissance avec une probabilité minorée par $p > 0$)

Anonymat

- Protocole Crowds
  - transmettre un message sans que le destinataire puisse en connaître l'origine
  - à chaque étape on choisit de transmettre au destinataire avec une probabilité p ou bien de le faire suivre à une autre personne en charge de poursuivre la distribution
- Dining Cryptographers
  - 3 cryptographes dînent ensemble
  - ils veulent savoir si l'un d'entre eux a payé l'addition sans révéler lequel
  - chaque couple de cryptographes choisit un booléen aléatoirement
  - chaque cryptographe calcule b par xor des deux booléens choisis
  - il révèle b s'il n'a pas payé et $\neg b$ sinon
  - les cryptographes font le xor des résultats
  - si le score est 0 alors aucun n'a payé, sinon l'un d'entre eux a payé mais les deux autres ne peuvent savoir qui

Analyse de sécurité

- les tirages aléatoires apparaissent quasi systématiquement dans les procédures (éviter le rejeu)
- dans les analyses symboliques de protocoles cryptographiques, le cryptage est considéré comme incassable
- comment exprimer cette propriété ?
- exemple de propriété de sécurité
  - on ne peut distinguer deux messages quelconques une fois qu'ils sont cryptés

Analyse de sécurité

Schéma IND-CPA (chosen plain text attack) pour du cryptage à clé publique

```
let (sk,pk) = keygen(η)
in let (m0,m1) = A(pk) and b = flip
in let m = crypt(pk,if b then m0 else m1)
in let b' = A'(pk,m0,m1,m) in b=b'
```

- $\mathcal{A}$ et $\mathcal{A}'$ sont des adversaires polynomiaux arbitraires dont la connaissance est limitée
- La probabilité que ce programme réponde **vrai** doit être : $\frac{1}{2} + \epsilon(\eta)$ avec $\epsilon(\eta)$ négligeable ($\leq \frac{1}{\eta^k}$ pour tout k, lorsque η est assez grand).
- La fonction **crypt** doit être aléatoire (sinon la connaissance de m_0, m_1 et pk donne la solution).

Cours 1 : Représenter des programmes aléatoires

Probabilités discrètes

- ensemble fini ou dénombrable A de résultats possibles d'une expérience (programme)
  - lancer deux dés à 6 faces :
    $A_1 = \{(1,1);(1,2);(2,1);..;(6,6)\}, \mid A_1 \mid = 36$
    $A_2 = \{\{1,1\};..;\{6,6\}\}, \mid A_2 \mid = 21$
- valeurs positives $(p_a)_{a\in A}$: p_a la probabilité que le résultat soit a
  - dés non biaisés : $p_{(i,j)} = \frac{1}{36} \quad p_{\{i,i\}} = \frac{1}{36}, p_{\{i,j\}} = \frac{2}{36}$
- probabilité : $\Sigma_{a\in A} p_a = 1$
  - sous-probabilité, si l'expérience peut ne pas aboutir : $\Sigma_{a\in A} p_a < 1$
- Cas fini, distribution uniforme : $p_a = \frac{1}{|A|}$
- Cas dénombrable : $A = \mathbb{N}$, $p_k = \frac{1}{2^{k+1}}$

Évènements

- un évènement est une partie de A
  - tirer un double $X = \{(1,1),(2,2)\ldots,(6,6)\}$
- une loi de probabilité est une application de l'ensemble des évènements dans $[0,1]$, $\text{prob}(X) = \Sigma_{a \in X} p_a$
- une probabilité est additive : si $(X_i)_i \in I$ ensemble dénombrable d'évènements disjoints 2 à 2 alors $\text{prob}(\cup_{i \in I} X_i) = \Sigma_{i \in I} \text{prob}(X_i)$
- opérations sur les évènements :
  - union : $\text{prob}(X \cup Y) = \text{prob}(X) + \text{prob}(Y) - \text{prob}(X \cap Y)$
  - complément : $\text{prob}(A \backslash X) = 1 - \text{prob}(X)$
  - intersection : X et Y sont indépendants si $\text{prob}(X \cap Y) = \text{prob}(X) \times \text{prob}(Y)$
    - $X = \{(1,x) | x = 1 \ldots 6\}$ et $Y = \{(x,2) | x = 1 \ldots 6\}$
  - évènement presque sûr : probabilité 1
  - évènement négligeable : probabilité 0

Variable aléatoire (réelle)

Transformation de l'expérience aléatoire, mesure du résultat

- application $f : A \to \mathbb{R}$
  - somme des valeurs des dés
- en général on ne mentionne pas la dépendance par rapport à l'espace de probabilité
  - $\text{prob}(f \geq 3) = \text{prob}(\{a \in A | f(a) \geq 3\}) = 1 - \frac{1}{36}$
- espérance (moyenne) :
  $\mathbb{E}(f) = \Sigma_{a \in A} f(a) p_a = \Sigma_{x \in \mathcal{I}m(f)} x \, \text{prob}(f = x) \in \mathbb{R} \cup \{\infty\}$
- variance ("distance" de l'observation à la moyenne) :
  $\mathbb{V}(f) = \mathbb{E}((f - \mathbb{E}(f))^2) = \mathbb{E}(f^2) - \mathbb{E}(f)^2$
- plus généralement une variable aléatoire peur prendre des valeurs dans un espace mesurable, typiquement $\mathbb{R}^n$ ou même $\mathbb{R}^\infty$ (marches aléatoires)

Cas général (continu)

- $\mathbb{B}^\infty$, suites infinies de booléens est non-dénombrable
  - programme qui tire à pile ou face un nombre non borné de fois
- probabilité d'un résultat élémentaire $a \in A$ non significative
  - probabilité uniforme sur $[0,1]$: $\text{prob}(x \in [a,b]) = b - a$, $\text{prob}(x = a) = 0$
- Tribu $\mathcal{T} \subseteq \wp(A)$: ensemble non vide d'évènements
  - stable par union dénombrable
  - stable par complémentaire

Propriétés

- $A, \emptyset \in \mathcal{T}$
- stable par intersection dénombrable

Exemple

- tribu borélienne (espace topologique) engendrée par les ouverts
- $\mathbb{R}^n$: tribu engendrée par des pavés $]a_1, b_1[\times \ldots \times]a_n, b_n[$

Probabilité sur $(A, \mathcal{T})$

- Définition : application $\mathcal{T} \to [0,1]$
  - $\text{prob}(A) = 1$
  - $(X_i)_{i \in I}$ fini ou dénombrable, disjoints 2 à 2 : $\text{prob}(\cup_{i \in I} X_i) = \Sigma_{i \in I} \text{prob}(X_i)$
- Variable aléatoire (réelle) : application $f \in A \to \mathbb{R}$ mesurable
  - les fonctions caractéristiques : $\mathbb{1}_X, X \in \mathcal{T}$
  - les combinaisons linéaires de fonctions mesurables
    - fonctions en escalier : $\Sigma_{i=1}^{n} \alpha_i \mathbb{1}_{X_i}$
  - les limites supérieures de suites dénombrables de fonctions mesurables

Espérance

- Espérance d'une variable aléatoire réelle positive $\mathbb{E}(f) \in [0, \infty]$
  - $\mathbb{E}(\mathbb{1}_X) = \text{prob}(X)$
  - $\mathbb{E}(\Sigma_{i \in I} \alpha_i f_i) = \Sigma_{i \in I} \alpha_i \mathbb{E}(f_i)$
  - $\mathbb{E}(\sup_{i \in I} f_i) = \sup_{i \in I} \mathbb{E}(f_i)$
- Propriétés
  - linéarité : $\mathbb{E}(\alpha f + \beta g) = \alpha \mathbb{E}(f) + \beta \mathbb{E}(g), \mathbb{E}(0) = 0$
  - monotonie : si $f \leq g$ alors $\mathbb{E}(f) \leq \mathbb{E}(g)$
  - continuité, suite croissante de fonctions : $\mathbb{E}(\lim_{n \to \infty} f_n) = \lim_{n \to \infty} \mathbb{E}(f_n)$
- Simulation (approche l'espérance en moyennant les expériences)
  - suite $(f_n)_{n \in \mathbb{N}}$ de variables aléatoires indépendantes (de même loi)
  - $g_n = \frac{1}{n} \Sigma_{i=0}^{n-1} f_n$
  - $\text{prob}(|g_n - \mathbb{E}(f)| \geq \epsilon) \longrightarrow_{n \to \infty} 0$
  - $\lim_{n \to \infty} g_n$ converge vers $\mathbb{E}(f)$ avec probabilité 1

Intermède : jeux de Monty-Hall

Principe du jeu

- un cadeau est caché dans l'une des trois boîtes identiques A, B, C
- le joueur choisit une des boîtes
- le meneur du jeu ouvre alors une des deux boîtes restantes qui est vide
- le joueur a alors la possibilité de changer la boîte qu'il va ouvrir
- que doit-il faire ?
- quelles sont ses chances de gagner ?

Analyse du jeu de Monty-Hall

- espace de probabilité
- situation où le changement est gagnant
- situation où le changement est perdant

Cours 1 : Représenter des programmes aléatoires

1. Introduction
   - Quelques exemples
   - Éléments de probabilités
2. Représentations de programmes aléatoires
   - Point de vue opérationnel
   - Programmes vus comme des lois de probabilité
3. Raisonner sur des programmes aléatoires
   - Logique de Hoare probabiliste
   - Terminaison probabiliste

Langage

Un mini-langage fonctionnel avec des primitives aléatoires (random)

- constantes et fonctions primitives : c
- primitives aléatoires : `random`, `flip`
- conditionnelle : **if** b **then** e_1 **else** e_2
- liaison locale : **let** $x = e_1$ **in** e_2
- abstraction : **fun** $(x : \tau) \rightarrow e$
- application : $(e_1\ e_2)$

deux évaluations du même programme peuvent donner des valeurs différentes

Approche monadique

- expression aléatoire : représente un calcul de type τ
- interprétation : expression purement fonctionnelle de type $[\tau]$

Opérateurs monadiques/Primitives aléatoires

$\texttt{unit} : \tau \to [\tau]$ $\quad$ $\texttt{bind} : [\tau] \to (\tau \to [\sigma]) \to [\sigma]$

$\texttt{random} : \tau$ interprété comme un objet fonctionnel de type $[\tau]$.

Expressions

Calcul $p : \tau$	Valeur fonctionnelle $[p] : [\tau]$
let $x = a$ **in** b	$\texttt{bind}\ [a]\ (\textbf{fun}\ (x : \sigma) \to [b])$
fun $(x : \sigma) \Rightarrow t$	**fun** $(x : \sigma) \Rightarrow [t]$
$(t\ u)$	$\texttt{bind}\ [u]\ [t]$
if b **then** e_1 **else** e_2	$\texttt{bind}\ [b]$ $\quad$ **fun** $(x : \texttt{bool}) \Rightarrow$ **if** x **then** $[e_1]$ **else** $[e_2]$

Point de vue opérationnel

- **random** : ρ
- chaque évaluation de **random** est indépendante, suit la même loi
- suite infinie $\pi \in \rho^\infty$ comme variable globale
- évaluation de **random** : consomme le premier élément de π
- probabilité sur l'ensemble ρ^∞ des séquences infinies de tirages indépendants selon **random**
  - tribu engendrée par des cônes $C_m = \{mw | w \in \rho^\infty\}, m \in \rho^*$
- programme $e : \tau \quad \rightarrow$ variable aléatoire dans $\rho^\infty \rightarrow \tau$

$$\begin{array}{ccc} e : \tau & \rightsquigarrow & [e] : \rho^\infty \rightarrow \tau \times \rho^\infty \\ \text{prob}(P(e)) & \rightsquigarrow & \text{prob}(\{\pi \in \rho^\infty | P(\mathbf{fst}([e]\ \pi))\}) \end{array}$$

- applications
  - Haskell
  - HOL : modéliser et prouver des algorithmes aléatoires (J. Hurd)
  - simulation (si on sait produire une séquence)

L'approche de Joe Hurd avec HOL

- *Formal Verification of Probabilistic Algorithms*. PhD thesis, University of Cambridge, 2002.
- *Verification of the Miller-Rabin Probabilistic Primaly test*. J. of Logic and Algebraic Programming, special issue on Probabilistic Techniques for the Design and Analysis of Systems, Elsevier Science, 2003.
- *A formal approach to probabilistic termination.* In *TPHOLs'02*, 2002.

Programmes probabilistes dans HOL

Programme déterministe, accède une suite infinie de tirages aléatoires de valeurs booléennes.

$$\mathbb{B}^\infty : \text{ensemble des suites infinies de booléens}$$

Un programme aléatoire prend en argument une suite $s \in \mathbb{B}^\infty$, consomme un segment initial de s pour calculer son résultat et renvoie la partie de la séquence non utilisée.

Variable aléatoire	$X : \alpha$	$[X] : \mathbb{B}^\infty \to \alpha \times \mathbb{B}^\infty$
Programme	$p : \alpha \to \beta$	$[p] : \alpha \to \mathbb{B}^\infty \to \beta \times \mathbb{B}^\infty$

Transformation monadique

suite infinie : variable globale transformée par effet de bord.

$$[\alpha] = \mathbb{B}^\infty \to \alpha \times \mathbb{B}^\infty$$

```
unit (a : α) : [α]
     = fun s → (a,s)
bind (a : [α])(f : α → [β]) : [β]
     = fun s → let (x,s') = a s in f x s'
flip : [bool]
     = fun s →(hd s,tl s)
```

Exemple

Calculer le nombre de piles dans n tirages à pile ou face.

- Programme ML

```
let rec nbp n = if n=0 then 0
else let p = nbp (n-1) in if flip() then p+1 else p
```

- Interprétation fonctionnelle

```
let rec nbp n s = if n=0 then (0,s)
else let (p,s1) = nbp (n-1) s in
     let (b,s2) = flip s1 in
     if b then (p+1,s2) else (p,s2)
```

- Notations monadiques

```
let rec nbp n = if n=0 then return 0
else do p ←nbp (n-1);
     do b ←flip;
     if b then return (p+1) else return p
```

Programme probabiliste dans HOL

- théorie de la mesure sur $\mathbb{B}^\infty$ formalisée dans HOL
- prob : $\mathcal{P}(\mathbb{B}^\infty) \to \mathbb{R}$
  - fonction partielle sur un ensemble $\mathcal{T}$ d'évènements clos par complément et union
- programme aléatoire $p : \alpha$
  $\to$ fonction HOL $[p] : [\alpha] = \mathbb{B}^\infty \to \alpha \times \mathbb{B}^\infty$
- probabilité que p satisfasse Q : prob($\{s \mid Q(\texttt{fst}\ ([p]\ s))\}$)
- conditions pour que $f : [\alpha]$ représente un programme probabiliste
  - $F = \{s \mid Q(\texttt{fst}\ (f\ s))\}$ est mesurable ($F \in \mathcal{T}$)
  - Indépendance : la suite de valeurs booléennes est consommée
- indépendance préservée par les opérations monadiques.

Test de primalité de p

- $p-1=2^r s \quad 0<a<p$
- $\texttt{test}\ p\ a =$
  $a^{(p-1)} \equiv 1(p) \wedge \forall j, 0<j \leq r, a^{2^j s} \equiv 1(p) \Rightarrow a^{2^{j-1} s} \equiv \pm 1(p)$
- si p premier : $\forall a. 0<a<p \Rightarrow \texttt{test}\ p\ a$
- si p est composite alors $|\{a|(\texttt{test}\ p\ a)\}| \leq \frac{(p-1)}{2}$

Programme

```
let rec miller_rabin p n =
if n = 0 then true
else let a = random (p-1) in
if test p a then miller_rabbin p (n-1) else false
```

Preuve dans HOL

- résultats avancés en théorie des groupes
- utilisation de tirages booléens qui approximent **`random`** pour garder une fonction totale
  - choix d'un entier entre 2 et $p-1$ avec probabilité $\geq \frac{1}{p-1}$
- p est premier : $\forall n\, s, \texttt{miller\_rabin}\ p\ n\ s = \texttt{true}$
- p est composé : $\forall n, \text{prob}(\texttt{miller\_rabin}\ p\ n = \texttt{false}) \geq 1 - \frac{1}{2^n}$

Terminaison

- Les programmes modélisées à l'aide des opérateurs monadiques sont transformés en fonctions totales.
- Une fonction totale de type $[\alpha]$ n'utilise qu'un nombre borné de tirages booléens :
  - la probabilité de chaque valeur est de la forme $\frac{m}{2^n}$.
- De nombreux programmes probabilistes ne peuvent pas s'exprimer ainsi : `random(3)` (probabilité $\frac{1}{3}$).
- Modélisation de fonctions qui terminent avec probabilité 1.

Principe de la construction

Boucle probabiliste avec condition déterministe :

$$\mathbf{body} : \alpha \to \mathbb{B}^\infty \to \alpha \times \mathbb{B}^\infty \qquad \mathbf{cond} : \alpha \to \mathbb{B}$$

Représentation

- Itération bornée :

```
let rec iter n a = if n=0 then unit a
else if cond a then bind (body a) (iter (n-1))
     else unit a
```

- Arrêt : `stop n a= not (cond (iter n a))`
- Boucle :

```
let loop a =
if ∃n,stop n a then iter (μn.stop n a) a else dum
```

Raisonner sur les boucles

- Propriété

```
loop a = if cond a then bind (body a) loop
         else unit a
```

- Terminaison presque sûre :

$$\forall a, \forall^* s, \exists n, (\texttt{stop}\ n\ a\ s)$$

Notation presque sûre

- $\forall^* s, \phi(s)$: ϕ presque sûrement
- $\{s \mid \phi(s)\} \in \mathcal{T} \wedge \text{prob}(\{s \mid \phi(s)\}) = 1$

- La propriété de terminaison presque sûre permet d'établir l'indépendance de la fonction `loop`.

Autres conditions de terminaison

Propriété équivalente :

$$\exists p > 0, \forall a. \text{prob}(\{s \mid \exists n. (\texttt{stop}\ n\ a\ s)\}) \geq p$$

Condition suffisante (variant f entier positif)

$$\exists p > 0, \forall a, \texttt{cond}\ a \Rightarrow \text{prob}(\{s \mid f\ (\texttt{fst}\ (\texttt{body}\ a\ s)) < f\ a\}) \geq p$$

Exemple : Bernoulli (p)

- Tirer vrai avec probabilité p et faux avec probabilité $1-p$.
- considérer la séquence aléatoire $s = (s_i)_{i\geq 1}$ comme l'écriture binaire du nombre $\Sigma_{i=1}^{\infty} \frac{s_i}{2^i}$
- ce nombre est inférieur à p avec probabilité p
- comparer avec l'écriture binaire de p

$$0 \leq \Sigma_{i=1}^{\infty} \frac{s_i}{2^i} \leq 1$$

$$\begin{array}{rcl} 0 + \Sigma_{i=2}^{\infty} \frac{s_i}{2^i} \leq p & \Leftrightarrow & \Sigma_{i=1}^{\infty} \frac{s_{i+1}}{2^i} \leq 2 \times p \\ \frac{1}{2} + \Sigma_{i=2}^{\infty} \frac{s_i}{2^i} \leq p & \Leftrightarrow & \Sigma_{i=1}^{\infty} \frac{s_{i+1}}{2^i} \leq 2 \times p - 1 \end{array}$$

Code

```
let rec bernoulli p =
let x = flip in
 if p < 1/2
 then
     if x then false else bernoulli (2×p)
 else
     if x then bernoulli (2×p-1) else true
```

Une chance sur deux de terminer à chaque itération

Exemple : marche aléatoire dimension 1

- entier n
- si $n > 0$, alors n croit ou décroît aléatoirement
- nombre d'étapes pour arriver à 0 à partir de n
- code :

```
let rec walk n k =
if n=0 then k
else if flip then walk (n+1) (k+1)
     else walk (n-1) (k+1)
```

Terminaison de la marche aléatoire

Analyse de $\pi_{i,j}$ probabilité en partant de i d'atteindre j.

$$\pi_{p+i,p} = \pi_{i,0} \qquad \pi_{i,0} = \pi_{i,1} \times \pi_{1,0} = \pi_{1,0}^{i}$$

$$\pi_{1,0} = \frac{1}{2}\pi_{2,0} + \frac{1}{2} = \frac{1}{2}\pi_{1,0}^{2} + \frac{1}{2}$$

Terminaison presque sûre

$$\pi_{1,0} = 1 = \pi_{i,0}$$

Conclusion : modélisation HOL

- Modélisation monadique simple
- Espace probabiliste clair, formalisé en HOL
- Possibilité de simulation en CAML des programmes
- Raisonnement formel
- Modélisation des programmes qui terminent presque sûrement possible mais plus complexe à mettre en œuvre
  - Codage de l'itération
  - Preuve de terminaison
  - Équation de définition
- Travaux dans la continuité de Sofiène Tahar et al. (Concordia U)
  - Variables aléatoires continues
  - Propriétés statistiques
  - Chaînes de Markov

Cours 1 : Représenter des programmes aléatoires

1. Introduction
   - Quelques exemples
   - Éléments de probabilités
2. Représentations de programmes aléatoires
   - Point de vue opérationnel
   - Programmes vus comme des lois de probabilité
3. Raisonner sur des programmes aléatoires
   - Logique de Hoare probabiliste
   - Terminaison probabiliste

Approche transformation de mesures

D. Kozen, G. Plotkin, C. Jones, C. Morgan . . .

- Dexter Kozen. Semantics of probabilistic programs. *Journal of Computer and System Sciences*, 1981.
- Dexter Kozen. A probabilistic PDL. In *15th ACM Symposium on Theory of Computing*, 1983.
- Claire Jones and Gordon Plotkin. A probabilistic powerdomain of evaluations. In *Proceedings of the Fourth Annual Symposium on Logic in Computer Science*, Pacific Grove, California, 1989. IEEE Comp. Soc. Press.
- Annabelle McIver and Carroll Morgan. *Abstraction, Refinement and Proof for Probabilistic Systems*. Monographs in Computer Science. Springer, 2005.

Approche transformation de mesures

- cadre impératif `x:=random` ou $\mathtt{P} \ +_p \ \mathtt{Q}$.
- un programme aléatoire p est non-déterministe, sa sémantique est un ensemble d'états
- analyse de la distribution de cet ensemble
- étant donnée une distribution sur les états initiaux, le programme détermine une distribution des résultats
- interpréter un programme aléatoire p comme un transformateur de mesures sur l'ensemble des états S
- partir de l'état s correspond à la mesure de Dirac δ_s
- pour connaître la probabilité que le résultat du programme p satisfasse ϕ en partant de s, il suffit de mesurer l'ensemble ϕ par la mesure associée au programme p en partant de δ_s

Interprétation d'un langage fonctionnel aléatoire

On reprend notre mini langage fonctionnel avec primitives aléatoires

- $p : \tau$ représente un ensemble de valeurs de type τ
- un programme $p : \tau$ correspond à une distribution μ_p sur τ
- si $Q : \tau \to \texttt{Prop}$, alors $\text{prob}(p \in Q) = \mu_p(Q)$

Transformation monadique

Calcul	Valeur fonctionnelle
$p : \tau$	$[p] : [\tau]$
	?**unit** : $\tau \to [\tau]$?**bind** : $[\tau] \to (\tau \to [\sigma]) \to [\sigma]$?[**random**] : $[\tau]$

Quelle monade ?

- La sortie du programme suit une loi de probabilité qui ne dépend que du programme.

$$\begin{aligned} e : \tau &\rightsquigarrow [e] : 2^{\tau} \to [0,1] \\ \mathrm{prob}(P(e)) &\rightsquigarrow ([e]\ P) \end{aligned}$$

Approche directe pour une analyse quantitative.

$$\texttt{flip} : \texttt{bool} \rightsquigarrow \textbf{fun}\ P \Rightarrow \tfrac{1}{2}\langle P(\texttt{true})\rangle + \tfrac{1}{2}\langle P(\texttt{false})\rangle$$

```
if flip then 0
else if flip then 1
else 2
```

$$\rightsquigarrow \textbf{fun}\ P \Rightarrow \tfrac{1}{2}\langle P(0)\rangle + \tfrac{1}{4}\langle P(1)\rangle + \tfrac{1}{4}\langle P(2)\rangle$$

Construction de la monade

Les ensembles $P : 2^\tau$ sont généralisés par des fonctions $f : \tau \to [0,1]$.

$$[\tau] = (\tau \to [0,1]) \to [0,1]$$

- forme de double négation et de continuation

$$\begin{aligned}
&\texttt{unit}\,(x:\tau):[\tau]\\
&\qquad = \textbf{fun}\,(f:\tau\to[0,1])\to\ (f\ x)\\
&\texttt{bind}\,(\mu_a:[\tau])(\mu_b:\tau\to[\sigma]):[\sigma]\\
&\qquad = \textbf{fun}\,(f:\sigma\to[0,1])\to\ (\mu_a\,(\textbf{fun}\ \ (x:\tau)\to\ \ (\mu_b\ x\ f)))
\end{aligned}$$

- vérifie les égalités monadiques : $(\texttt{bind}\,(\texttt{unit}\ x)\ \mu) = (\mu\ x)$
- interprétation en terme de mesure
  - mesure de Dirac
  - indépendance des appels aléatoires
- idée analogue proposée par Ramsey, Pfeffer en Haskell (expectation monad)

Interprétation des primitives aléatoires

$$
\begin{array}{lcl}
\texttt{random}(n) & : & [\texttt{int}] \\
 & = & \textbf{fun}\, f \to \Sigma_{i=1}^{n} (f\ i)/n \\
\texttt{flip} & : & [\texttt{bool}] \\
 & = & \textbf{fun}\, f \to \frac{1}{2}(f\ \texttt{true}) + \frac{1}{2}(f\ \texttt{false}) \\
e_1 +_p e_2 & : & [\tau] \\
 & = & \textbf{fun}\, f \to p \times ([e_1]\ f) + (1-p) \times ([e_2]\ f)
\end{array}
$$

Interprétation fonctionnelle

Transforme un programme fonctionnel avec **random** en une mesure

- Code aléatoire

```
let rec prime p n =
if n = 0 then true
else if test p (random (p-1)) then prime p (n-1)
else false
```

- Interprétation fonctionnelle

```
let rec prime_fun p n =
if n = 0 then return true
else do a ←random (p-1); do b ←test p a;
     if b then prime_fun p (n-1) else return false
```

Calcul fonctionnel

- Le programme fonctionnel calcule la probabilité que le résultat satisfasse une propriété (décidable)
- Le programme est faux quand il répond vrai alors que p n'est pas premier

```
let prime_correct p b =
    if b then if exact_prime p then 1. else 0. else 1.
```

Calculer la probabilité de succès

Probabilité que le programme se comporte correctement après n itérations.

```
let evaluate p n = prime_fun p n (prime_correct p)
```

```
# evaluate 23 1;;
- : float = 1
# [evaluate 9 0; evaluate 9 1;evaluate 9 2;evaluate 9 3];;
- : float list = [0.; 0.75; 0.9375; 0.984375]
```

Inefficace ! test tous les cas possibles pour **random** ! !

Exercice

Dans votre langage (fonctionnel) préféré :

- programmer le jeu de Monty-Hall
  - choix aléatoire de l'endroit où est le cadeau, de la réponse du joueur et de la boîte dévoilée
  - résultat : solution et réponse finale du joueur
- programmer les opérateurs monadiques **`bind`**, **`unit`**, **`random`**
- programmer la mesure associée au programme Monty-Hall
- tester les probabilités de succès avec différentes stratégies

Récursion générale

Un programme qui termine seulement avec probabilité 1 :

```
let rec fix_test x =
if flip then fix_test (x+1) else x
```

```
# fix_test 1;;
- : int = 1
# fix_test 1;;
- : int = 2
# fix_test 1;;
- : int = 2
# fix_test 1;;
- : int = 3
# fix_test 1;;
- : int = 1
```

Interprétation fonctionnelle

```
let rec fix_fun x =
do b ←flip; if b then fix_fun (x+1) else return x
```

```
# fix_test_fun 1 (fun n -> 1.);;
Stack overflow during evaluation (looping recursion?).
```

il faut construire des limites

Modélisation de programmes probabilistes la bibliothèque ALEA en COQ

Christine Paulin-Mohring

contributions de Philippe Audebaud, David Baelde, Pierre Courtieu
projet ANR SCALP (2007-2011)

Université Paris-Sud, INRIA Saclay - Île-de-France

Janvier 2014

Cours 1 : Représenter des programmes aléatoires

1. Introduction
   - Quelques exemples
   - Éléments de probabilités
2. Représentations de programmes aléatoires
   - Point de vue opérationnel
   - Programmes vus comme des lois de probabilité
3. Raisonner sur des programmes aléatoires
   - Logique de Hoare probabiliste
   - Terminaison probabiliste

Rappels

- ini-langage fonctionnel avec des primitives aléatoires
  - constantes et fonctions primitives : c
  - primitives aléatoires : `random`, `flip`
  - conditionnelle : **if** b **then** e_1 **else** e_2
  - liaison locale : **let** $x = e_1$ **in** e_2
  - abstraction : **fun** $(x : \tau) \rightarrow e$
  - application : $(e_1\ e_2)$
- les primitives aléatoires exécutées déterminent l'espace de probabilité
  - le programme est une variable aléatoire sur cet espace

Monade associée

$$[\tau] = (\tau \to [0,1]) \to [0,1]$$

$$\begin{array}{ll}
\texttt{unit} & (x : \tau) : [\tau] \\
& = \textbf{fun}\,(f : \tau \to [0,1]) \to\ (f\ x) \\
\texttt{bind} & (\mu_a : [\tau])(\mu_b : \tau \to [\sigma]) : [\sigma] \\
& = \textbf{fun}\,(f : \sigma \to [0,1]) \to\ (\mu_a\ (\textbf{fun}\ \ (x : \tau) \to\ \ (\mu_b\ x\ f))) \\
\texttt{random}(n) & : [\texttt{int}] \\
& = \textbf{fun}\, f \to \Sigma_{i=1}^{n}(f\ i)/n \\
\texttt{flip} & : [\texttt{bool}] \\
& = \textbf{fun}\, f \to \tfrac{1}{2}(f\ \texttt{true}) + \tfrac{1}{2}(f\ \texttt{false})
\end{array}$$

- on peut faire le calcul si l'espace de probabilité est fini
- que faire pour des programmes qui ne terminent que presque sûrement ?

Point fixes généraux

let rec `f (x:`σ`) : `τ` = F f x`

- Interprétation : $F : (\sigma \to [\tau]) \to \sigma \to [\tau]$
- Structure de cpo sur $\sigma \to [\tau]$ héritée de $[0, 1]$
- Construction :

$$f_0 \equiv \textbf{fun}\,(x : \sigma)\,(f : \tau \to [0,1]) \to 0$$
$$f_{n+1} \equiv (F\ f_n) \quad \textbf{fix}\ F \equiv \sqcup_{n\in\mathbb{N}}\ f_n$$

- Propriétés :

$$f \leq g \Rightarrow (F\ f) \leq (F\ g) \quad \Longrightarrow \quad f_n \leq f_{n+1}$$
$$F\ (\sqcup_{n\in\mathbb{N}} f_n) \leq \sqcup_{n\in\mathbb{N}} (F\ f_n) \quad \Longrightarrow \quad F\ (\textbf{fix}\ F) = (\textbf{fix}\ F)$$

- On peut avoir $\textbf{fix}\ F\mathbb{1} < 1$ si la fonction ne termine pas sûrement

Conclusion sur le point de vue distribution

- une approche qui ne nécessite pas de théorie des probabilités
- fonctionne dans le cadre de variables aléatoires discrètes
  - sinon restreindre le formalisme à des fonctions mesurables
- prise en compte de points fixes généraux
- mesures de fonctions à valeur dans $[0, 1]$
  - pourrait s'étendre à $\mathbb{R}^+ \cup \{\infty\}$
  - on peut aussi considérer une action sur des intervalles
- possibilité de calculer mais pour une analyse quantitative, pas le résultat du programme

Cours 1 : Représenter des programmes aléatoires

1. Introduction
   - Quelques exemples
   - Éléments de probabilités
2. Représentations de programmes aléatoires
   - Point de vue opérationnel
   - Programmes vus comme des lois de probabilité
3. Raisonner sur des programmes aléatoires
   - Logique de Hoare probabiliste
   - Terminaison probabiliste

Raisonnement élémentaire

- Les propriétés de monotonie, linéarité, complément des mesures associées aux primitives sont préservées par les constructions monadiques **`bind`**, **`unit`** et point fixe
- Construire la mesure $[e]$ associée au programme e
- Construire la fonction caractéristique associée à la propriété à tester
- Raisonner en utilisant les identités monadiques et les propriétés des primitives aléatoires

Sémantique axiomatique

Action des programmes sur les prédicats $\{P\}e\{Q\}, P \Rightarrow \mathbf{wp}(e, Q)$

Cas fonctionnel

Pour un programme fonctionnel pur e, on veut prouver $P \Rightarrow Q(e)$.

Cas probabiliste

propriétés de la forme

- la probabilité que e satisfasse Q est plus grande que p
- peut s'écrire $p \leq [e]\, \mathbb{1}_Q$

Règles admissibles

utiliser la sémantique pour justifier l'admissibilité des règles

- application

$$\frac{k \leq [a](f) \quad \forall x, f\ x \leq [e\ x](g)}{k \leq [e\ a](g)}$$

- conditionnelle

$$\frac{k_1 \leq [e_1](f) \quad k_2 \leq [e_2](f)}{k_1 \times [b](\mathbb{1}_{.=\texttt{true}}) + k_2 \times [b](\mathbb{1}_{.=\texttt{false}}) \leq [\textbf{if } b \textbf{ then } e_1 \textbf{ else } e_2](f)}$$

- liaison locale

$$\frac{1 \leq [a](\mathbb{1}_P) \quad \forall x, (P\ x) \Rightarrow k \leq [b](f)}{k \leq [\textbf{let } x = a \textbf{ in } b](f)}$$

Justification de schémas généraux

- trouver une valeur de type τ qui vérifie une propriété Q
- itérer un programme probabiliste pour améliorer le résultat
- données :
  - deux programmes aléatoires $p_1, p_2 : \tau$
  - une fonction de choix $\texttt{choice} : \tau \to \tau \to \tau$
  - le choix est bon : $\forall x, (Q\ x) \Rightarrow Q\ (\texttt{choice}\ x\ y)$ et $\forall y, (Q\ y) \Rightarrow Q\ (\texttt{choice}\ x\ y)$
- nouveau programme

  let x = p_1 **in let** y = p_2 **in** **choice** x y
- analyse (nouvelle estimation meilleure que k_1 et k_2) :
  - $k_1 \leq [p_1](\mathbb{1}_Q)$ et $k_2 \leq [p_2](\mathbb{1}_Q)$ implique $k_1(1-k_2)+k_2 \leq [p](\mathbb{1}_Q)$
- généralisation à $q : \tau \to [0,1]$
  - hypothèse $\forall x\ y, \texttt{min}((q\ x)+(q\ y), 1) \leq q\ (\texttt{choice}\ x\ y)$
  - les deux programmes doivent terminer avec probabilité 1

Cours 1 : Représenter des programmes aléatoires

1. Introduction
   - Quelques exemples
   - Éléments de probabilités
2. Représentations de programmes aléatoires
   - Point de vue opérationnel
   - Programmes vus comme des lois de probabilité
3. Raisonner sur des programmes aléatoires
   - Logique de Hoare probabiliste
   - Terminaison probabiliste

Programmes récursifs

- si la récursion est structurelle alors elle “passe” aux distributions
  - exemple : répéter n fois un test aléatoire
  - le raisonnement se fait par récurrence
  - pas de non-terminaison induite par le point-fixe
- dans le cas général il faut utiliser la construction de point fixe
  - prouver la continuité pour obtenir l’équation de point fixe
  - exemples : marche aléatoire, boucle **`while`**

Règle axiomatique pour le point fixe

let rec `f x = F f x`

- $f : \sigma \to \tau, [f] : \sigma \to [\tau]$
- $q : \tau \to [0,1]$ la fonction à mesurer
- soit une suite croissante de fonctions $(l_i)_i$ de type $\sigma \to [0,1]$ telle que $\forall x, l_0\ x = 0$.

$$\frac{(\forall x, l_n\ x \leq [f\ x](q)) \Rightarrow \forall x, l_{n+1}\ x \leq [F\ f\ x](q)}{\forall x, \sqcup_n\ (l_n\ x) \leq [f\ x](q)}$$

Exemple : Bernoulli

Calcule `true` avec probability p.

```
let rec bern p =
    if flip then if p < 1/2 then false else bern(2p-1)
            else if p < 1/2 then bern(2p) else true
```

$$\forall p, \sqcup_n (p - \frac{1}{2^n}) \leq [\texttt{bern}\ p](\mathbb{1}_{\texttt{true}})$$

Preuve

- hypothèse : $\forall p, (p - \frac{1}{2^n}) \leq [\texttt{bern}\ p](\mathbb{1}_{\texttt{true}})$
- on simplifie le corps de `bern` p

```
if flip then if p < 1/2 then false else bern(2p-1)
        else if p < 1/2 then bern(2p) else true
```

  - si $p < \frac{1}{2}$, on montre $p - \frac{1}{2^{n+1}} \leq \frac{1}{2}[\texttt{bern}(2p)](\mathbb{1}_{\texttt{true}})$
  - si $\frac{1}{2} \leq p$, on montre $p - \frac{1}{2^{n+1}} \leq \frac{1}{2}[\texttt{bern}(2p-1)](\mathbb{1}_{\texttt{true}}) + \frac{1}{2} \times 1$
- on conclut $p \leq [\texttt{bern}\ p](\mathbb{1}_{\texttt{true}})$
- de manière analogue $(1 - p) \leq [\texttt{bern}\ p](\mathbb{1}_{\texttt{false}})$
- on conclut $1 \leq [\texttt{bern}\ p](\mathbb{1})$
- `bern` termine avec probabilité 1

Principe de preuve par commutation

- soit $F : (\sigma \to [\tau]) \to (\sigma \to [\tau]), q : \tau \to [0,1]$
- on suppose qu'il existe $G : (\sigma \to [0,1]) \to (\sigma \to [0,1])$ tel que

$$\forall f\ x, (F\ f\ x)(q) = G\ (\textbf{fun}\ y \to (f\ y)(q))\ x$$

- alors $(\texttt{fix}\ F\ x)(q) \equiv \texttt{fix}\ G\ x$
- Application à Bernoulli
  - **let rec** `b q p =` **if** `p<`$\frac{1}{2}$ **then** $\frac{1}{2}$`(q false)+`$\frac{1}{2}$`b(2p)` **else** $\frac{1}{2}$`b(2p-1)+`$\frac{1}{2}$`(q true)`
  - On a $[\texttt{bern}\ p](q) \equiv \texttt{b}\ q\ p$ (commutation)
  - avec $q = \mathbb{1}$, on montre $\texttt{b}\ \mathbb{1}\ p = 1$
  - Bernoulli termine avec probabilité 1

Invariant pour les boucles

```
let rec loop s =
  if cond s then let s' = body s in loop s' else s
```

- on veut borner $[\texttt{loop}\, s](q)$
- soit des fonctions (invariants) $\phi, \psi \in S \to [0,1]$
- $x \& y = max(x + y - 1, 0)$ se comporte comme une conjonction sur les popriétés logiques
- règles dérivées, la terminaison de la boucle entre en jeu

$$\frac{\forall s, \phi\, s \leq [\texttt{cond}\, s](\mathbb{1}_{\texttt{true}}) \times [\texttt{body}\, s](\phi) + [\texttt{cond}\, s](\mathbb{1}_{\texttt{false}}) \times q\, s}{\forall s, \phi\, s \& [\texttt{loop}\, s](\mathbb{1}) \leq [\texttt{loop}\, s](q)}$$

$$\frac{\forall s, [\texttt{cond}\, s](\mathbb{1}_{\texttt{true}}) \times [\texttt{body}\, s](\psi) + [\texttt{cond}\, s](\mathbb{1}_{\texttt{false}}) \times q\, s \leq \psi s}{\forall s, [\texttt{loop}\, s](q) \leq \psi\, s}$$

Invariant pour les boucles

Cas d'une condition non probabiliste

$$\frac{\forall s, \mathbf{cond}\, s \Rightarrow \phi\, s \leq [\mathbf{body}\, s](\phi) \quad \forall s, \neg \mathbf{cond}\, s \Rightarrow \phi\, s \leq q\, s}{\forall s, \phi\, s \& [\mathtt{loop}\, s](\mathbb{1}) \leq [\mathtt{loop}\, s](q)}$$

$$\frac{\forall s, \mathbf{cond}\, s \Rightarrow \phi s \leq [\mathbf{body}\, s](\phi)}{\forall s, \phi\, s \& [\mathtt{loop}(s)](\mathbb{1}) \leq [\mathtt{loop}\, s](\phi\, s \& \mathbb{1}_{\mathtt{cond} .= \mathtt{false}})}$$

Autre condition de terminaison

Problème abstrait d'un algorithme de construction de labyrinthe

- `rdata` probabiliste, les autres fonctions sont déterministes

```
let rec iter u =
if end u then out u
else let d = rdata in
     if cond u d then iter u else iter (step u d)
```

- hypothèses

$$[\mathtt{rdata}](\mathbb{1}) = 1$$
$$k < 1 \qquad \forall u, \neg \mathtt{end}\, u \Rightarrow [\mathtt{rdata}](\mathbb{1}_{\mathtt{cond}\, u .= \mathtt{true}}) \leq k$$
$$\forall u\, x, \neg \mathtt{end}\, u \Rightarrow \neg \mathtt{cond}\, u\, x \Rightarrow \mathtt{size}(\mathtt{step}\, u\, x) < \mathtt{size}\, u$$
$$\forall u, \mathtt{size}\, u = 0 \Rightarrow \mathtt{end}\, u$$

- résultat : $\forall u, [\mathtt{iter}\, u](\mathbb{1}) = 1$

Autre condition de terminaison (preuve)

suite $(p_{x,n})$, x représente la taille et n les itérations

$$
\begin{aligned}
p_{x,0} &= 0 \\
p_{0,n} &= 1 \\
p_{(x+1),(n+1)} &= k \times p_{(x+1),n} + (1-k) \times p_{x,n}
\end{aligned}
$$

- décroissance en x : $p_{x+1,n} \leq p_{x,n}$
- croissance en n : $p_{x,n} \leq p_{x,n+1}$
- convergence vers 1 : $\sqcup_{n \in \mathbb{N}} p_{x,n} = 1$
- $\sqcup_{n \in \mathbb{N}} p_{\mathtt{size}\, u,n} \leq [\mathtt{iter}\, u](\mathbb{1})$

Conclusion

- Le cadre fonctionnel est très naturel
- Possibilité dans les cas simples de calculer systématiquement les probabilités (terminaison, efficacité)
- Avantage de la structure de cpo de $[0, 1]$ pour construire des point-fixes
- L'existence de programmes qui peuvent ne pas terminer presque sûrement complique les formules
- Sémantique alternative correspondant à de la correction partielle
- L'interprétation explicite la structure probabiliste des programmes, les arguments de correction demandent souvent des propriétés avancées
- Vision idéale des générateurs aléatoires primitifs
- Ne traite pas directement des programmes parallèles avec combinaison de choix probabiliste et non-déterministe (cf travail de McIver et Morgan)

4. Bibliothèque ALEA
   - Préliminaires : cpo
   - Modélisation de l'intervalle $[0, 1]$
   - Modélisation des distributions
   - Automatisation des preuves
5. Exemples d'utilisation
   - Sécurité
   - Equivalence
6. Conclusion

Architecture de la bibliothèque

- bibliothèque relativement “autonome” (indépendante de $\mathbb{R}$)
  - 8000 lignes définitions
  - 10000 lignes de preuves
- principaux outils Coq utilisés
  - type classes (ordre, monotonie, continuité)
  - setoid rewrite
- plusieurs versions depuis environ 10 ans. . .
  - dernière version V8 (mai 2013)
  - disponible `http://www.lri.fr/~paulin/ALEA`
- modélisation de la sémantique (shallow embedding)

Modélisation d'ordres

- ordre sur A défini par deux relations $\leq$ et $\equiv$
  1. $\forall x, x \leq x$
  2. $\forall x\, y, (x \leq y \wedge y \leq x) \Leftrightarrow x \equiv y$
  3. $\forall x\, y\, z, x \leq y \Rightarrow (y \leq z) \Rightarrow x \leq z$
- ordre strict $x < y := x \leq y \wedge x \not\equiv y$
- ordre point à point sur les fonctions
- monotonie des fonctions
- type $O_1 \xrightarrow{m} O_2$ des fonctions monotones
- combinateurs sur les fonctions monotones
- suites monotones

ω-cpo

Structure avec un plus petit élément et une borne sup pour les suites croissantes

Class `cpo` ‘{o :Ord D} : **Type** := `mk_cpo`
{`D0` : D ;
`lub` : ∀(f :nat $\xrightarrow{m}$D), D ;
`Dbot` : ∀x :D, `D0` ≤ x ;
`le_lub` : ∀(f : `nat` $\xrightarrow{m}$D) (n :`nat`), f n ≤ `lub` f ;
`lub_le` : ∀(f : `nat` $\xrightarrow{m}$D) (x :D), (∀n, f n ≤ x) →`lub` f ≤ x}.

Notation ”0” := `D0`.

Point-fixe

Construit par itération

Fixpoint `iter` ‘{c : cpo D} (f : D $\xrightarrow{m}$ D) n {**struct** n} : D :=
  match n **with** O $\to$0 | `S` m $\to$f (`iter` f m) **end**.

Instance iter_mon : $\forall$‘{c : `cpo` D} (f : D $\xrightarrow{m}$ D), `monotonic` (`iter` f).

Definition `fixp` ‘{c : `cpo` D} (f : D $\xrightarrow{m}$ D) : D := `lub` (`iter` f).

Propriété :

Lemma fixp_le : $\forall$‘{c : `cpo` D} (f : D $\xrightarrow{m}$ D), `fixp` f $\leq$ f (`fixp` f).

Continuité :

Class `continuous` ‘{c1 :`cpo` D1} ‘{c2 :`cpo` D2} (f :D1 $\xrightarrow{m}$ D2) :=
  `conti` : $\forall$h :`nat`$\to$D1, `monotonic` h $\to$f (`lub` h) $\leq$ `lub` (f@h).

Lemma fixp_eq : $\forall$‘{c : `cpo` D} (f : D $\xrightarrow{m}$ D) {mf :`continuous` f},
    `fixp` f == f (`fixp` f).

Modéliser $[0,1]$ (axiomes)

Cpo : $x \equiv y \quad x \leq y \quad 0 \quad \texttt{lub} : (\texttt{nat} \xrightarrow{m} [0,1]) \to [0,1]$

Opérations algébriques primitives : $x \times y \quad x + y \quad 1 - x$

Notation : $1 := 1 - 0$

Axiomes : $+$ et $\times$ sont AC, neutre 0 et 1, monotones, continues

$$0 \neq 1 \quad 1 - (1 - 0) = 0 \quad x \leq y \to 1 - y \leq 1 - x$$

Sous l'hypothèse $x \leq 1 - y$ ($x + y$ sans overflow)

$$(x + y) \times z \equiv x \times z + y \times z$$
$$(1 - (x + y)) + x \equiv (1 - y)$$
$$x + y \leq z + y \to x \leq z$$

Multiplication :

$$1 - (x \times y) \equiv (1 - x) \times y + (1 - y)$$
$$z \not\equiv 0 \to x \times y \leq z \times y \to x \leq z$$

Autres constructions

Constantes : $\frac{1}{n+1}, n \in \mathbb{N}$
Axiomes :

$$\frac{1}{n+1} = 1 - (n \times (\frac{1}{n+1}))$$
$$x \not\equiv 0 \to \exists n, \frac{1}{n+1} \leq x$$

Division : x/y
Axiomes :

$$y \not\equiv 0 \to x \leq y \to y \times (x/y) \equiv x$$
$$y \not\equiv 0 \to y \leq x \to x/y \equiv 1$$
$$y \equiv 0 \to x/y \equiv 0$$

Axiomes classiques

$$\neg\neg(x \leq y) \Rightarrow x \leq y$$

$$x \leq y \vee_c y \leq x$$

Propriétés dérivées

$$1 - (1 - x) \equiv x$$
$$0 \leq x \leq 1$$
$$(1 - x) + x \equiv 1$$
$$1 - x \leq y \rightarrow x + y \equiv 1$$
$$0 \times x \equiv 0$$
$$\ldots$$
$$(\forall t, t < x \rightarrow t \leq y) \rightarrow x \leq y$$
$$(\forall p, x \leq y + \frac{1}{p+1}) \rightarrow x \leq y$$

$[0,1]$: opérations dérivées

- itération des opérations $n \in \mathbb{N}, x \in [0,1] : n \times x, x^n$
- $x \& y := 1 - ((1-x) + (1-y))$ (correspond à $x + y - 1$)
  AC, monotone, continue

$$1 - (x + y) \equiv (1 - x) \& (1 - y)$$
$$x \& y \leq x$$
$$x \leq 1 - y \to x \& y \equiv 0$$
$$x \& 0 \equiv 0 \quad x \& 1 \equiv x$$

- soustraction :
  $x - y := 1 - ((1-x) + y)$ $\quad |x - y| := (x - y) + (y - x)$
- $\mathtt{max}(x,y) := (x - y) + y$ $\quad \mathtt{min}(x,y) := 1 - ((y - x) + (1 - y))$
- sommes et produits finis : $\Sigma_{k=0}^{n-1} f\, k \quad \Pi_{k=0}^{n-1} f\, k$
- sommes et produits infinis : condition $f\, n \leq \Sigma_{k=0}^{n-1} f\, k$

de $[0,1]$ à $\mathbb{R}^+$

- réel représenté par un couple $(n, x), n \in \mathbb{N}, x \in [0,1]$
- double représentation $(n, 1) \equiv (n+1, 0)$
- nécessité de décider si $x \equiv 1$ ($x \leq y$ booléen)
- partie entière, partie décimale
- programmation (complexe) des opérations standard $(+, -, \times, /)$
  - accès aux tactiques semi-ring, semi-field de Coq
- vision relationnelle des bornes sup : `islub` f x

Distribution en Coq

type $(\texttt{distr}\ \tau)$ des distributions sur τ est le type des fonctions $\mu : (\tau \to [0,1]) \to [0,1]$ qui satisfont les propriétés de stabilité :

- compatibilité avec l'addition : $f \leq 1 - g \Rightarrow \mu(f+g) \equiv \mu(f) + \mu(g)$
- compatibilité avec la multiplication : $\mu(k \times f) \equiv k \times \mu(f)$
- compatibilité avec l'inverse : $\mu(1-f) \leq 1 - \mu(f)$
  (μ sous-probabilité)
- continuité : $\mu(\texttt{lub}_n\, f_n) \leq \texttt{lub}_n\, (\mu(f_n))$

Représenté en Coq comme un record dépendant

Constructions monadiques

- **unit** de type $\tau \to \mathbf{distr}\ \tau$
- **bind** de type $\mathbf{distr}\ \tau \to (\tau \to \mathbf{distr}\ \sigma) \to \mathbf{distr}\ \sigma$
- **flip** de type $\mathbf{distr}\ \mathbf{bool}$
- **random** de type $\mathbb{N} \to \mathbf{distr}\ \mathbb{N}$ (tirage dans $[0, n]$)

Propriétés :

- lois monadiques :

$$\mathbf{bind}\ (\mathbf{unit}\ x)\ m \equiv m\ x$$
$$\mathbf{bind}\ \mu\ \mathbf{unit} \equiv \mu$$
$$\mathbf{bind}\ (\mathbf{bind}\ \mu\ M)\ N \equiv \mathbf{bind}\ \mu\ (\mathbf{fun}\ x \to \mathbf{bind}\ (M\ x)\ N)$$

- indépendance : $\mathbf{bind}\ \mu_1\ (\mathbf{fun}\ \_ \to \mu_2)\ f \equiv \mu_1(\mathbb{1}) \times (\mu_2\ f)$
- monotonie : $\dfrac{\mu_1 \leq \mu_2 \quad M_1 \leq M_2}{\mathbf{bind}\ \mu_1\ M_1 \leq \mathbf{bind}\ \mu_2\ M_2}$

Autres constructions probabilistes

- probabilité conditionnelle
- probabilité discrète sur A : $c : \mathbb{N} \to [0,1], p : \mathbb{N} \to A$
- **if** de type **distr bool** $\to$ **distr** $\tau \to$ **distr** $\tau \to$ **distr** τ
- distribution image
  im_distr (*f* :$A \to B$) (*m* :**distr** *A*) : **distr** *B* :=
  bind *m* (**fun** *a* $\Rightarrow$ **unit** (*f a*)).
- distribution produit
  prod_distr (*d1* :**distr** *A*)(*d2* :**distr** *B*) : **distr** (*A*$\times$*B*) :=
  bind *d1* (**fun** *x* $\Rightarrow$ **bind** *d2* (**fun** *y* $\Rightarrow$ **unit** (*x*,*y*))).
  commutation prouvable pour des distributions discrètes

Autres éléments de la bibliothèque

- preuves de règles de sémantique axiomatique
- définition d'une distribution de manière récursive sur un ordre bien fondé
- fonctions caractéristiques
  $cover\ (P : \texttt{set}\ A)(f : A \to [0,1]) :=$
  $\forall\ x, (P\ x \to 1 \leq f\ x) \wedge (\neg\ P\ x \to f\ x \leq 0).$
- distribution uniforme sur un ensemble fini
- choix dans une liste
  Fixpoint *choose* (*l* : *list A*) : *distr A* :=
  match *l* **with**
  | **nil** $\Rightarrow$ *distr_null A*
  | **cons** *hd tl* $\Rightarrow$ **choice** $\frac{1}{\texttt{length}\ l}$ (**unit** *hd*) (*choose tl*)
  end.
- autre exemples de base : loi de Bernoulli, loi binomiale, marche aléatoire

Automatisation des preuves

- tactique de simplification pour les expressions dans $[0, 1]$
- connexion à **`AC-rewrite`**
- tactique de simplification des expressions avec des distributions
- usage intensif de **`setoid-rewrite`** (réecriture sous les lambdas)
- de manière générale l'automatisation reste très artisanale
- peu de support spécifique pour les expressions de programmes

Machine de Turing probabiliste

```
Variables state alph : Type.
Variable accept : state → bool.

Inductive dir := L | R.

Record output := {next :state ;write :alph ;move :dir}.

Variable trans : state → alph → distr output.

Record config := mkT {cur : state ; tape :Z → alph ; pos : Z}.

Definition update (m :config) (o :output) : config
:= mkT (next o)
        (fun z ⇒ if z = (pos m) then write o else tape m z)
        (if move o then pos m - 1 else pos m + 1).
```

Machine de Turing probabiliste

Construction par point fixe

```
let rec run (m:config) : distr config
    if accept (cur m) then return m
    else do out ←trans (cur m) (tape (pos m));
            run (update m out)
```

En Coq :

Definition *Frun* : (*config* → **distr** *config*) $\overset{m}{\to}$ (*config* → **distr** *config*)
:= *mon* (**fun** *f* (*m* :*config*) ⇒
if (**accept** (cur *m*)) **then** **unit** *m* **else**
bind (**trans** (cur *m*) (tape *m* (pos *m*))
(**fun** *out* ⇒ *f* (*update m out*)))).

Definition *run* : *config* → **distr** *config* := *Mfix Frun*.

Modéliser la securité

Analyse de la securité de programmes utilisant des primitives cryptographiques

- modèle symbolique : un code ne peut pas être cassé sauf si les clés ont fuité
- modèle calculatoire :
  - codage basé sur des problèmes calculatoirement durs
  - un adversaire avec des capacités de calcul polynomiales n'aura qu'un avantage négligeable d'obtenir des informations protégées
  - les preuves de sécurité dans le modèle calculatoire sont compliquées
- sécurité prouvable : estimer formellement la probabilité pour un intrus d'obtenir de l'information confidentielle
- avoir des méthodes générales pour de telles preuves (jeux, logiques)
- utiliser des assistants de preuve pour éviter des erreurs

Sécurité calculatoire d'un algorithme de Tatouage

D. Baelde, P. Courtieu, D. Gross-Amblard, C. Paulin (ITP 2012)
marque non-révoquable

Marquage et détection

- données $\mathcal{M}$
- clé secrète K, support $O \in \mathcal{M}$
  - marquage : $\texttt{mark}(K, O) \in \mathcal{M}$
  - prédicat de détection : $\texttt{detect}([O], K, O') = \top/\bot$
- enlever la marque sans corrompre l'image
  - donnée marquée : $O' \stackrel{def}{=} \texttt{mark}(K, O)$
  - donnée similaire O'' : $\texttt{dist}(O', O'') \leq \gamma$
  - marque enlevée si $\texttt{detect}(O, K, O'') = \bot$

Marquer des chaînes de bits

$\mathcal{M}$: bitstrings taille n $\qquad \mathtt{dist}(O, O') = |O \oplus O'|_1$

Example (Bit-flipping)

- clé secrète K, taille n, seulement k 1-bits
- bits inversés sur les bits 1 de K
  - $\mathtt{mark}(K, O) \stackrel{def}{=} K \oplus O$
  - $\mathtt{detect}(O, K, O') \stackrel{def}{=} \delta \leq |K \wedge (O \oplus O')|_1$

Secret : Probabilité de deviner la clé $\leq 1/\binom{n}{k}$

\+ *Réduction*
enlever la marque proba $p \rightsquigarrow$ deviner la clé proba $p/(\ldots)$

= **Sécurité**
la probabilité d'enlever la marque est négligeable

Réduction

donnée tatouée : $O' = \texttt{mark}(K, O)$
étant donnée une attaque sur la marque : $O'' = A(O')$,
on construit une attaque sur la clé : A'

$$\begin{aligned} \texttt{genkey}() &:= \texttt{BVrandom}^{\texttt{k}}(n, k) \\ \texttt{gensupport}() &:= \texttt{BVrandom}(n) \end{aligned}$$

$\texttt{guess\_key}(A) :=$
$K \xleftarrow{\$} \texttt{genkey}();$
$O \xleftarrow{\$} \texttt{gensupport}();$
$O' \leftarrow \texttt{mark}(K, O);$
$K' \xleftarrow{\$} A(O');$
return$(K = K')$

$\texttt{attack\_mark}(A) :=$
$K \xleftarrow{\$} \texttt{genkey}();$
$O \xleftarrow{\$} \texttt{gensupport}();$
$O' \leftarrow \texttt{mark}(K, O);$
$O'' \xleftarrow{\$} A(O');$
return$(\texttt{dist}(O', O'') < \gamma \wedge$
$\#\texttt{marks}(O, K, O'') < \delta)$

Réduction

Construction de $A'(O', O'')$

$$
\begin{array}{l}
\mathtt{A}'\,(A, O') := \\
\quad O'' \xleftarrow{\$} A(O'); \\
\quad K' \xleftarrow{\$} O' \oplus O'' \\
\quad k_a \leftarrow \mathtt{dist}(O', O''); \\
\quad e \xleftarrow{\$} \mathtt{Random}(\gamma + 2\delta - k); \\
\quad k_m \leftarrow (e + k - k_a)/2; \\
\quad k_i \leftarrow (e - k + k_a)/2; \\
\quad I \xleftarrow{\$} \mathtt{BVrandom}^{\mathtt{k}}_{\mathtt{mask}}(k_i, K'); \\
\quad M \xleftarrow{\$} \mathtt{BVrandom}^{\mathtt{k}}_{\mathtt{mask}}(k_m, \neg K'); \\
\quad \mathbf{return}(M \vee (K' \wedge \neg I))
\end{array}
$$

Théorèmes Coq

Theorem `reduction :`
$\forall A, \mathrm{prob}(\texttt{attack\_mark}\, A) \times \ldots \leq \mathrm{prob}(\texttt{guess\_key}\, (A'\, A))$.

Theorem `secrecy :` $\forall A, \mathrm{prob}(\texttt{guess\_key}\, A) \leq \frac{1}{\binom{n}{k}}$.

Theorem `robustness :`

$$\forall A, \mathrm{prob}(\texttt{attack\_mark}\, A) \leq \frac{(\gamma+2\delta-k)\times\binom{\gamma}{\frac{\gamma}{2}}\times\binom{n-k+\delta}{\delta}}{\binom{n}{k}}.$$

Modélisation de techniques de jeux sur des programmes

Certicrypt (G. Barthe, B. Grégoire, S.Zanella, POPL'09)

$$(G_0, A_0) \stackrel{h_1}{\to} (G_1, A_1) \cdots \stackrel{h_n}{\to} (G_n, A_n)$$

- G_i jeu (aléatoire) modélise l'interaction entre l'attaquant et le système
- A_i évènement
- $\text{prob}_{G_{i-1}}(A_{i-1}) \leq h_i(\text{prob}_{G_i}(A_i))$

Exemple

```
let (sk,pk) = keygen(η)
in let (m0,m1) = A(pk) and b = flip
in let m = crypt(pk,if b then m0 else m1)
in let b' = A'(pk,m0,m1,m) in b=b'
```

Schéma de cryptage ElGamal

Schéma asymétrique : groupe cyclique d'ordre q généré par g.

let `keygen()` = **let** `x=random(q)` **in** (x, g^x)
let `crypt(pk,m)` = **let** `y=random(q)` **in** $(g^y, pk^y \times m)$

Hypothèse (Decisional) Diffie-Hellman
Programme DDH_0 :

let `x=random(q)` **and** `y=random(q)` **in** $\mathcal{B}(g^x, g^y, g^{xy})$

ne peut être distingué du programme DDH_1 :

let `x=random(q)` **and** `y=random(q)` **in**
let `z=random(q)` **in** $\mathcal{B}(g^x, g^y, g^z)$

Preuve

IND-CPA pour ElGamal équivalent au jeu G_0

```
let x=random(q) and y=random(q)
in let (m0,m1) = A(g^x) and b = flip()
in let m = g^{xy} × (if b then m0 else m1)
in let b' = A'(g^x,g^y,m) in b=b'
```

ne peut être distingué du jeu G_1

```
let x=random(q) and y=random(q)
in let (m0,m1) = A(g^x) and b = flip()
in let z=random(q) in let m = g^z
in let b' = A'(g^x,g^y,m) in b=b'
```

cas particulier de

```
let b = flip() in let b'= μ in b=b'
```

probabilité d'avoir `true` est $\frac{1}{2}$.

Transformation de programmes aléatoires

- deux programmes aléatoires $p_1 : A_1$ et $p_2 : A_2$.
- exprimer que les sorties sont dans la relation R ?
  - 2 exécutions du même programme peuvent donner des résultats différents
- simulation probabiliste $A_1 \times A_2$ telle que
  1. μ coincide avec μ_i sur chaque projection $\pi_i := \textbf{fun}\,(x_1, x_2) \Rightarrow x_i$:

$$\pi_i \circ \mu \equiv \mu_i$$

  2. la sortie de μ est dans la relation R :

$$\forall f, (\forall xy, R(x, y) \to f\,(x, y) \equiv 0) \to \mu(f) \equiv 0$$

- p_1 et p_2 coincident sur les fonctions équivalentes sur R :

$$(\forall xy, R(x, y) \to f_1(x) = f_2(y)) \to \mu_1(f_1) \equiv \mu_2(f_2)$$

car $\mu(f_1 \circ \pi_1 - f_2 \circ \pi_2) \equiv 0$ et $\mu(f_i \circ \pi_i) = \mu_i(f_i)$.

Résultats de Certicrypt

- plongement profond d'un langage impératif probabiliste
- prise en compte explicite du temps polynomial et de la consommation de ressources
- logique de Hoare relationnelle probabiliste
- transformations de programmes
- applications : FDH (signatures), OAEP ...
- sémantique construite sur la bibliothèque ALEA
- travaux actuels plutôt orientés vers de la génération d'obligations

Cours 2 : Formalisation dans la bibliothèque ALEA

Bilan

- sémantique formelle pour les programmes aléatoires
  - permet des calculs effectifs de probabilité dans des cas simples
  - adaptée à des raisonnements abstraits probabilistes (terminaison)
  - justification de la correction de méthodes plus automatiques
  - limitation aux variables aléatoires discrètes ?
- moins adaptée à la preuve de programmes spécifiques
  - les manipulations algébriques sur $[0, 1]$ peuvent être délicates (revenir à $\mathbb{R}$?)
  - automatiser les preuves de monotonie et de continuité
  - besoin d'automatisation des raisonnements combinatoires, simplification, approximation
  - traiter spécifiquement les programmes qui terminent presque sûrement

Pistes d'extension

- instancier $[0, 1]$ pour pouvoir calculer dans Coq dans des cas simples
- espérance de variables aléatoires réelles
- théorie des probabilités
  - borne de Bienaymé-Tchebychev
  - chaînes de Markov
  - théorie de l'information
  - ...
- environnement spécifique pour la preuve de programmes probabilistes
- générateurs (pseudo)-aléatoire
- programmes distribués (notion plus faible que celle de probabilité)