Une introduction à la programmation probabiliste (1/2)

École Jeunes Chercheuses et Jeunes Chercheurs du GdrIM

Guillaume Baudart et Christine Tasson

Institut Supérieur de l'Aéronautique et de l'Espace SUPAERO

Nantes, Juin 2024

Introduction

Langages et Modèles

La Programmation Probabiliste

Paradigme de programmation qui permet de modéliser l'incertitude par des distributions de probabilité et de les manipuler à l'aide de processus stochastiques. La génération et l'inférence sont automatisées.

De nombreux langages dédiés :

Stan, Gen.jl, Pyro,...

Modèles en

- vision (génération d'image),
- robotique (planification),
- santé (épidémiologie),
- sciences sociales (sondages d'opinion),...

Aujourd'hui

Principes de la programmation probabiliste

- Programmes, variables aléatoires et simulation
- Conditionnement et inférence Bayésienne

Modélisation à l'aide de **muPPL**

- https://github.com/gbdrt/mu-ppl
- Exemples et exercices

Sémantique

- Méthode formelle
- Sémantique à noyau
- Limites pour l'ordre supérieur

Programmation probabiliste

Énumération

Que fait ce programme ?

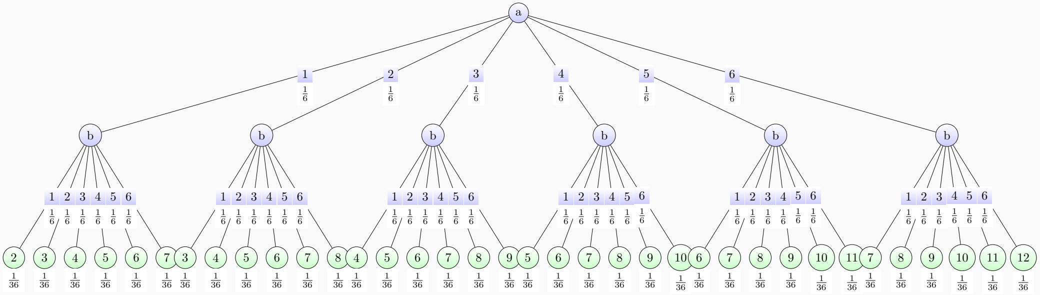
```
def dice() → int:
    a = sample(RandInt(1, 6), name=”a”)
    b = sample(RandInt(1, 6), name=”b”)
    return a + b
```

Que fait ce programme ?

```
def dice() → int:
    a = sample(RandInt(1, 6), name="a")
    b = sample(RandInt(1, 6), name="b")
    return a + b
```

Il simule la **variable aléatoire** dice :
« La somme des valeurs renvoyées par deux dés non pipés indépendants. »

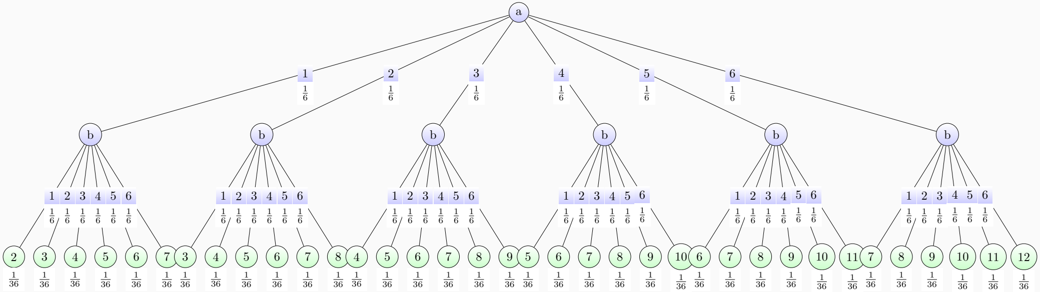
À chaque exécution, l'opérateur sample **simule** une variable aléatoire de loi uniforme sur $\{1, \ldots, 6\}$.



Comment calculer la loi de la variable aléatoire associée à ce programme ?

```
def dice() → int:
    a = sample(RandInt(1, 6), name="a")
    b = sample(RandInt(1, 6), name="b")
    return a + b

with Enumeration():
    dist: Categorical[int] = infer(dice)
```

L'**énumération** de tous les échantillons possibles permet de calculer la loi dist de la variable aléatoire dice.

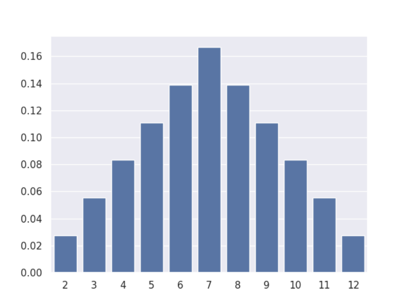


Quelle est la loi de la variable aléatoire associée à ce programme ?

```
def dice() → int:
    a = sample(RandInt(1, 6), name="a")
    b = sample(RandInt(1, 6), name="b")
    return a + b

with Enumeration():
    dist: Categorical[int] = infer(dice)
    print(dist.stats())
    viz(dist)
    plt.show()
```

La loi discrète :

$$
\begin{aligned}
[\![\mathrm{dice}]\!] : &= \mathbb{P}(\mathrm{dice}() = k) \\
&= \sum_{a=1}^{6} \sum_{b=1}^{6} \frac{1}{36} \mathbb{1}_{\{a+b=k\}}
\end{aligned}
$$



Que fait ce programme ?

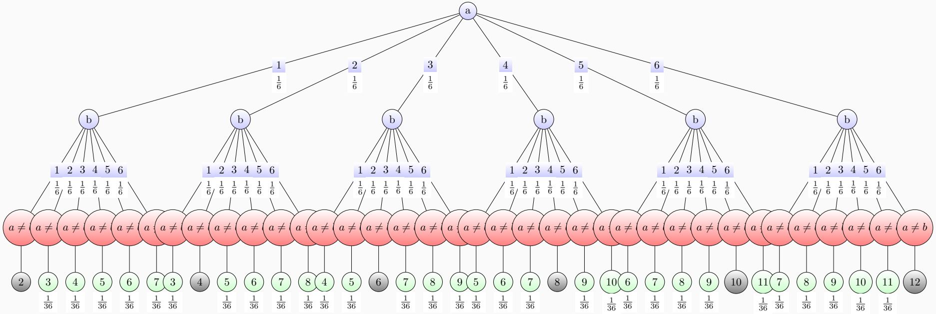
```
def hard_dice() → int:
    a = sample(RandInt(1, 6), name="a")
    b = sample(RandInt(1, 6), name="b")
    assume (a != b)
    return a + b
```

Que fait ce programme ?

```
def hard_dice() → int:
    a = sample(RandInt(1, 6), name="a")
    b = sample(RandInt(1, 6), name="b")
    assume (a != b)
    return a + b
```

Il simule la **variable aléatoire** :
« La somme des valeurs de deux dés indépendants non pipés, sachant que ces valeurs sont distinctes. »

L'opérateur **assume** rejette les échantillons qui ne vérifient pas sa propriété.



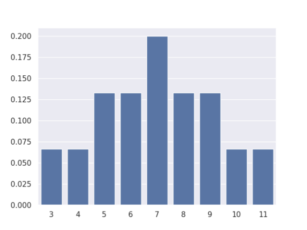
Quelle est la loi associée au programme ?

```
def hard_dice() → int:
    a = sample(RandInt(1, 6), name="a")
    b = sample(RandInt(1, 6), name="b")
    assume (a != b)
    return a + b

with Enumeration():
    dist: Categorical[int] = infer(hard_dice)
    print(dist.stats())
    viz(dist)
    plt.show()
```

La loi conditionnelle :

$$[\![\text{hard\_dice}]\!]\,(k) = \mathbb{P}(a+b = k \mid a \neq b)$$

$$= \sum_{a \neq b \in \{1,\ldots,6\}^2} \frac{\frac{1}{36}\mathbb{1}_{\{a+b=k\}}}{\sum_{a \neq b \in \{1,\ldots,6\}^2} \frac{1}{36}}$$



Programmation probabiliste

Simulation de Monte-Carlo

Que fait ce programme ?

```
def disk() → Tuple[float, float]:
    x = sample(Uniform(-1, 1))
    y = sample(Uniform(-1, 1))
    assume (x**2 + y**2 < 1)
    return (x, y)
```

Que fait ce programme ?

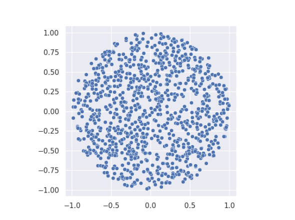
```
def disk() → Tuple[float, float]:
    x = sample(Uniform(-1, 1))
    y = sample(Uniform(-1, 1))
    assume (x**2 + y**2 < 1)
    return (x, y)
```

Il simule la **variable aléatoire** de loi conditionnelle :
« *Les coordonnées d'un point tirées uniformément dans un carré,*
sachant que sa distance au centre est strictement inférieure à 1. »

Comment calculer la loi de ce programme ?

```
def disk() → Tuple[float, float]:
    x = sample(Uniform(-1, 1))
    y = sample(Uniform(-1, 1))
    assume (x**2 + y**2 < 1)
    return (x, y)

with RejectionSampling(num_samples=1000):
    dist: Empirical = infer(disk)
    x, y = zip(*dist.samples)
    sns.scatterplot(x=x, y=y)
```

Simulation de Monte-Carlo :
N-échantillon : $(X_1, \ldots, X_N)$ *i.i.d.* de même loi que disk



Loi des grands nombres : l'espérance est la limite de la moyenne empirique du N-échantillon

$$\frac{1}{N}\sum_{i=1}^{N} g(X_i) \xrightarrow[N\to\infty]{} \mathbb{E}(g(X)).$$

On peut alors approcher la moyenne, la fonction de répartition, les moments de la loi de disk

Que fait ce programme ?

```
def position(o: float) → Tuple[float, float]:
    x = sample(Uniform(-1, 1))
    y = sample(Uniform(-1, 1))
    d2 = x**2 + y**2
    observe(Gaussian(d2, 0.1), o)
    return (x, y)
```

Que fait ce programme ?

```
def position(o: float) → Tuple[float, float]:
    x = sample(Uniform(-1, 1))
    y = sample(Uniform(-1, 1))
    d2 = x**2 + y**2
    observe(Gaussian(d2, 0.1), o)
    return (x, y)
```

Le programme position représente la loi de « la position d'un point sachant la mesure *bruitée* du carré de sa distance au centre. »

Étant donnée la position (X, Y) **a priori** uniforme sur $[-1, 1]^2$,
on **observe** la variable aléatoire Z de loi gaussienne centrée sur le carré de la distance de (X, Y) au centre.

Le programme position représente la loi de (X, Y) sachant $Z = o$.

Comment calculer la loi de ce programme ?

```
def position(o: float) → Tuple[float, float]:
    x = sample(Uniform(-1, 1))
    y = sample(Uniform(-1, 1))
    d2 = x**2 + y**2
    observe(Gaussian(d2, 0.1), o)
    return (x, y)

with ImportanceSampling(num_particles=10000):
    dist: Categorical = infer(position, 0.5)
```

$$X \sim \text{Uniform(-1,1)}$$
$$Y \sim \text{Uniform(-1,1)}$$
$$Z \sim \text{Gaussian}(X^2 + Y^2, 0.1)$$

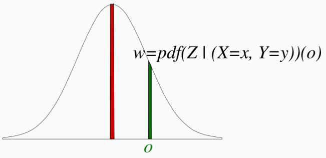
position est une variable aléatoire de loi conditionnelle : (X, Y) sachant $Z = o$.

Simulation de Monte-Carlo

N-échantillon *i.i.d.* de loi (X, Y)

observe pondère chaque exécution par la densité de Z en o sachant la position $(X, Y) = (x, y)$ (vraisemblance).

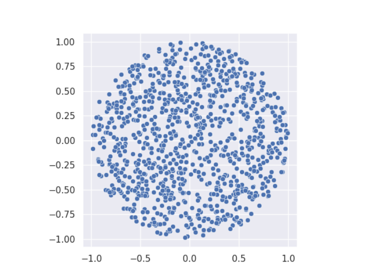
La loi des grands nombres permet d'approcher les caractéristiques de la loi de (X, Y) sachant $Z = o$



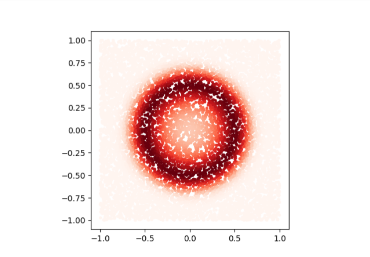
Conditionnement par rejet (hard) et par pondération (soft)

```
def disk() → Tuple[float, float]:
    x = sample(Uniform(-1, 1))
    y = sample(Uniform(-1, 1))
    d2 = x**2 + y**2
    assume (d2 < 1)
    return (x, y)

with RejectionSampling(num_samples=10000):
    dist: Empirical = infer(disk)
```

```
def position(o: float) → Tuple[float, float]:
    x = sample(Uniform(-1, 1))
    y = sample(Uniform(-1, 1))
    d2 = x**2 + y**2
    observe(Gaussian(d2, 0.1), o)
    return (x, y)

with ImportanceSampling(num_particles=10000):
    dist: Categorical = infer(position, 0.5)
```



Simulation de Monte-Carlo

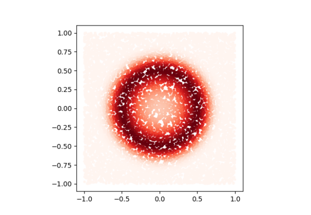


Comment calculer la loi de ce programme ?

```
def position(o: float) → Tuple[float, float]:
    x = sample(Uniform(-1, 1))
    y = sample(Uniform(-1, 1))
    d2 = x**2 + y**2
    observe(Gaussian(d2, 0.1), o)
    return (x, y)

with ImportanceSampling(num_particles=10000):

    dist: Categorical = infer(position, 0.5)
    w = dist.probs
```

Simulation de Monte-Carlo : N-échantillons iid : $(X_i, Y_i), i \leq N$ pondérés par :

$w_i = \mathbb{P}(Z = o | (X_i, Y_i) = (x_i, y_i))$



Loi des grands nombres : l'espérance est la limite de la moyenne empirique du N-échantillon :

$$\frac{\frac{1}{N}\sum_i g(X_i, Y_i) w_i}{\frac{1}{N}\sum_i w_i} \xrightarrow[N \to \infty]{} \frac{\mathbb{P}(Z = o | (X_i, Y_i) = (x, y))}{\mathbb{P}(Z = o)} = \mathbb{E}(g(X, Y) | Z = o).$$

Programmation probabiliste

Modèles hiérarchiques

Que fait ce programme ?

```
def success(s:int) → int:
    n = sample(RandInt(10, 20))
    d_success = Binomial(n, 0.5)
    observe(d_success, s)
    return n
```

Que fait ce programme ?

```
def success(s:int) → int:
    n = sample(RandInt(10, 20))
    d_success = Binomial(n, 0.5)
    observe(d_success, s)
    return n
```

Loi a priori : X uniforme entre 10 et 20

Loi conditionnelle : S le nombre de succès parmi X lancers d'une pièce équilibrée

Vraisemblance :
$\mathbb{P}(S = s | X = n) = \mathrm{Binomial(n,\ 0.5)}(s)$

Le programme success représente la **loi a posteriori** : loi conditionnelle de X sachant $S = s$

« Combien de fois (entre 10 *et* 20*) a-t-on lancé une pièce équilibrée sachant qu'on a obtenu s succès ? »*

Comment calculer la loi de ce programme ?

```
def success(s:int) → int:
    n = sample(RandInt(10, 20))
    observe(Binomial(n, 0.5), s)
    return n

with ImportanceSampling(num_particles=100):
    dist: Categorical = infer(success, 10)
```

Simulation de Monte-carlo :
N-échantillon $(X_1, \dots, X_N)$ iid de loi X uniforme entre 10 et 20.
Pondération : vraisemblance
$w_i = \mathbb{P}(S = s | X_i = n_i)$

Formule de Bayes :

$$\mathbb{P}(X = n | S = s) = \frac{\mathbb{P}(S = s | X = n)\ \mathbb{P}(X = n)}{\mathbb{P}(S = s)} = \frac{\mathbb{P}(S = s | X = n)\ \mathbb{P}(X = n)}{\displaystyle\sum_{k=10}^{20} \mathbb{P}(S = s | X = k)\mathbb{P}(X = k)}$$

Loi des grands nombres : l'espérance est la limite de la moyenne empirique.

$$\frac{\sum_{i=1}^{N} g(X_i) w_i}{\sum_{i=1}^{N} w_i} \xrightarrow[N \to \infty]{} \mathbb{E}(g(X) | S = s).$$

Calcul de l'espérance

Loi des grands nombres : $h(X_i) = w_i = \mathbb{P}(S = s|X_i)$

$$\lim_{N\to\infty} \frac{1}{N} \sum_{i=1}^{N} h(X_i) = \mathbb{E}(h(X)) = \sum_{k=10}^{20} \mathbb{P}(S = s|X = k)\mathbb{P}(X = k) = \mathbb{P}(S = s)$$

Loi des grands nombres : $\rho(X_i) = \frac{w_i g(X_i)}{\mathbb{P}(S=s)}$

$$\begin{aligned} \lim_{N\to\infty} \frac{1}{N} \sum_{i=1}^{N} \rho(X_i) &= \mathbb{E}(\rho(X)) = \sum_{k=10}^{20} \frac{\mathbb{P}(S = s|X = k)\mathbb{P}(X = k)g(k)}{\mathbb{P}(S = s)} \\ &= \sum_{k=10}^{20} \mathbb{P}(X = k|S = s)g(k) = \mathbb{E}(g(X)|S = s) \end{aligned}$$

Conclusion :

$$\frac{\sum g(X_i)w_i}{\sum w_i} \xrightarrow[N\to\infty]{} \mathbb{E}(g(X)|S = s).$$

Programmation probabiliste

Exercices de modélisation

Exercices - https://github.com/gbdrt/mu-ppl

Que modélise ce programme ?

```
def coin(obs: list[int]) → float:
    p = sample(Uniform(0, 1))
    for o in obs:
        observe(Bernoulli(p), o)
    return p

with ImportanceSampling(num_particles=10000):
    dist: Categorical = infer(coin, [0, 0, 0, 0, 0, 0, 0, 0, 1, 1])
```

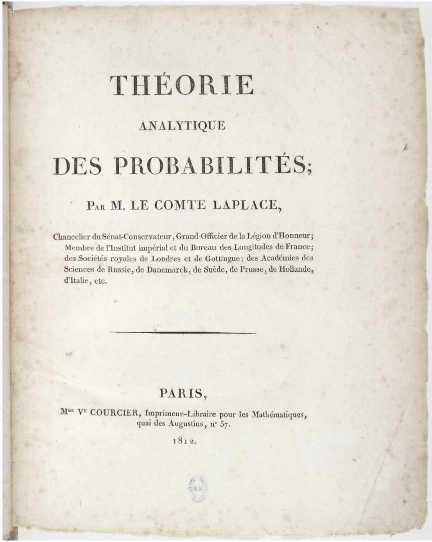
Écrire des programmes modélisant les variables aléatoires suivantes :

Le nombre de lancé d'une pièce biaisée avant d'obtenir *face*.

Une pièce équilibrée à partir d'une pièce biaisée, *en utilisant le conditionnement par rejet.*

La régression linéaire $y = ax + b$ d'un nuage de points.

Le problème de Laplace.



THÉORIE

ANALYTIQUE

DES PROBABILITÉS;

PAR M. LE COMTE LAPLACE,

Chancelier du Sénat-Conservateur, Grand-Officier de la Légion d'Honneur; Membre de l'Institut impérial et du Bureau des Longitudes de France; des Sociétés royales de Londres et de Gottingue; des Académies des Sciences de Russie, de Danemarck, de Suède, de Prusse, de Hollande, d'Italie, etc.

PARIS,

Mme Ve COURCIER, Imprimeur-Libraire pour les Mathématiques, quai des Augustins, n° 57.

1812.

Quelle est la probabilité d'avoir plus de filles que de garçons à Paris qu'à Londres ?

28. C'est principalement aux naissances, que l'analyse précédente est applicable, et l'on peut en déduire non-seulement pour l'espèce humaine, mais pour toutes les espèces d'êtres organisés, des résultats intéressans. Jusqu'ici les observations de ce genre n'ont été faites en grand nombre, que sur l'espèce humaine : nous allons soumettre au calcul, les principales.

Considérons d'abord les naissances observées à Paris, à Londres, et dans le royaume de Naples. Dans l'espace des 40 années écoulées depuis le commencement de 1475, époque où l'on a commencé à distinguer à Paris, sur les registres, les naissances des deux sexes, jusqu'à la fin de 1784, on a baptisé dans cette capitale, 393386 garçons, et 377555 filles, les enfans trouvés étant compris dans ce nombre : cela donne à peu près $\frac{25}{24}$ pour le rapport des baptêmes des garçons à ceux des filles.

Dans l'espace des 95 années écoulées depuis le commencement de 1664 jusqu'à la fin de 1758, il est né à Londres, 737629 garçons, et 698958 filles ; ce qui donne $\frac{19}{18}$ à peu près, pour le rapport des naissances des garçons à celles des filles.

Enfin, dans l'espace des neuf années écoulées depuis le commencement de 1774 jusqu'à la fin de 1782, il est né dans le royaume de Naples, la Sicile non-comprise, 782352 garçons, et 746821 filles ; ce qui donne $\frac{22}{21}$ pour le rapport des naissances des garçons à celles des filles.

Les plus petits de ces nombres de naissances, sont relatifs à Paris ; d'ailleurs, c'est dans cette ville que les naissances des garçons et des filles, approchent le plus de l'égalité. Par ces deux raisons, la probabilité que la possibilité de la naissance d'un garçon surpasse $\frac{1}{2}$, doit y être moindre qu'à Londres et dans le royaume de Naples. Déterminons numériquement cette probabilité.

Nommons p le nombre des naissances masculines observées à Paris, q celui des naissances féminines, et x la possibilité d'une naissance masculine, c'est-à-dire la probabilité qu'un enfant qui doit naître, sera un garçon ; $1-x$ sera la possibilité d'une naissance féminine, et l'on aura la probabilité que sur $p+q$ naissances,

48

. . .

la probabilité que la possibilité des baptêmes des garçons est plus grande à Londres qu'à Paris, a pour expression,

$$1-\frac{i.c^{-\frac{1}{2i^2}}}{\sqrt{2\pi}}\cdot\cfrac{1}{1+\cfrac{i^2}{1+\cfrac{2i^2}{1+\cfrac{3i^2}{1+\cfrac{4i^2}{1+\text{etc.}}}}}}.\quad(\mu)$$

En faisant dans cette formule,

$$p=393386\,,\quad q=377555\,,$$
$$p'=737629\,,\quad q'=698958\,,$$

elle devient

$$1-\frac{1}{328269}.$$

Il y a donc 328268 à parier contre un, qu'à Londres, la possibilité des baptêmes des garçons est plus grande qu'à Paris. Cette probabilité approche tellement de la certitude, qu'il y a lieu de rechercher la cause de cette supériorité.

Parmi les causes qui peuvent la produire, il m'a paru que les baptêmes des enfans trouvés, qui font partie de la liste annuelle des baptêmes à Paris, devaient avoir une influence sensible sur le rapport des baptêmes des garçons à ceux des filles ; et qu'ils devaient diminuer ce rapport, si, comme il est naturel de le croire, les parens des campagnes environnantes, trouvant de l'avantage à retenir près d'eux les enfans mâles, en avaient envoyé à l'hospice des Enfans trouvés de Paris, dans un rapport moindre que celui des naissances des deux sexes. C'est ce que le relevé des registres de cet hospice m'a fait voir avec une très-grande probabilité. Depuis le commencement de 1745 jusqu'à la fin de 1809, on y a baptisé 163499 garçons et 159405 filles, nombre dont le rapport est $\frac{39}{38}$, et diffère trop du rapport $\frac{25}{24}$ des baptêmes des garçons et des filles à Paris, pour être attribué au simple hasard.

30. Déterminons, d'après les principes précédens, les probabilités

lités

Programme probabiliste pour le problème de Laplace.

```
def laplace(f1: int, g1: int, f2: int, g2: int) → float:
    p = sample(Uniform(0, 1), name="p")
    q = sample(Uniform(0, 1), name="q")
    observe(Binomial(f1 + g1, p), g1, name="f1")
    observe(Binomial(f2 + g2, q), g2, name="f2")
    return q > p


# Paris 1745 − 1784
fp = 377555
gp = 393386
# Londres 1664 − 1758
fl = 698958
gl = 737629
with ImportanceSampling(num_particles=100000):
    dist: Categorical = infer(laplace, fp, gp, fl, gl)
    print(dist.stats())
```

Sémantique

Introduction

Qu'est-ce que la sémantique

Sémantique dénotationnelle

Les programmes dénotent ou représentent des fonctions agissant sur des valeurs ou sur des états de la machine.

Christopher Strachey (1916-1975) *Towards a formal semantics (1964)*

Dana Scott (1932–) and Christopher Strachey *Towards a mathematical Semantics of Computer Languages (1971)*

Sémantique opérationnelle

Système de transition dont les états sont les états d'une machine abstraite (machine de Turing, automate à piles, machine de Krivine,. . .), des termes (réécriture),. . .

Peter Landin (1930-2009) *The Mechanical Evaluation of Expressions (1966)*

Sémantique probabiliste

Programmes probabilistes

- Décrire des modèles statistiques
- Calculer exactement ou approcher les caractéristiques de variables aléatoires

Méthodes formelles

- Comment prouver qu'un programme probabiliste décrit le bon modèle statistique ?
- Comment prouver que l'implémentation d'un algorithme d'inférence est correcte ?
- Comment prouver la correction des transformations de programmes ?

Dexter Kozen, *Semantics of probabilistic programs, 1979*

Sémantique

Syntaxe

Syntaxe d'un sous-ensemble simplifié de mu-PPL

$$t ::= \mathrm{None} \mid \mathrm{bool} \mid \mathrm{int} \mid \mathrm{real} \mid t \times t \mid \mathrm{dist}(t)$$

$$e ::= c \mid x \mid (e, e) \mid op(e)$$

$$\begin{aligned} s ::=\ & \mathrm{pass} \mid x = e \mid x = f(e) \mid \mathrm{if}\ e\text{: } s\ \mathrm{else}\text{: } s \mid s;\ s \\ & \mid x = \mathrm{sample}(e) \mid \mathrm{factor}(e) \end{aligned}$$

$$d ::= \mathrm{def}\ f(x)\text{: } s\ \mathrm{return}\ e \mid d\ d$$

```
def assume(p):
    factor(0 if p else -np.inf) # zero probability

def observe(d, v):
    factor(d.log_prob(v)) # log-likelihood of v w.r.t. d
```

Sémantique

Sémantique des types

Sémantique des types

Espace mesurable

Soit X un ensemble. Une tribu Σ_X est un ensemble de parties de X, contenant X, stable par complétaire et unions dénombrables. Les éléments de Σ_X sont nommés ensembles mesurables.

Interprétation des types

$$
\begin{array}{lcll}
[\![t]\!] & = & t,\ \Sigma_t & \textit{Espace mesurable} \\[0.5em]
[\![\mathrm{None}]\!] & = & \{*\},\ \mathcal{P}(\{*\}) & \textit{l'espace discret à un seul point} \\
[\![\mathrm{bool}]\!] & = & \mathbb{B},\ \mathcal{P}(\mathbb{B}) & \textit{l'espace discret à deux points } \mathrm{true} \textit{ et } \mathrm{false} \\
[\![\mathrm{int}]\!] & = & \mathbb{N},\ \mathcal{P}(\mathbb{N}) & \textit{l'espace discret des entiers} \\
[\![t_1 \times t_2]\!] & = & [\![t_1]\!] \otimes [\![t_2]\!] & \textit{l'espace produit } t_1 \times t_2,\ \Sigma_{t_1} \otimes \Sigma_{t_2} \\
[\![\mathrm{dist}(t)]\!] & = & \textbf{Meas}([\![t]\!]) & \textit{l'espace mesurable des mesures sur } [\![t]\!]
\end{array}
$$

Sémantique des types

Espace produit $X \otimes Y$

Univers : $X \times Y$

Tribu : la plus petite tribu engendrée par les pavés $U \times V$ pour $U \in \Sigma_X$ et $Y \in \Sigma_Y$

Espace mesurable des mesures $\mathbf{Meas}(X)$

Une **mesure** sur Y est une fonction $\mu : \Sigma_Y \to \mathbb{R}^+$ σ-additive : $\mu(\uplus U_i) = \sum \mu(U_i)$

Univers : mesures sur X

Tribu : engendrée par $\{\mu | \mu(U) < r\}$ $\forall U \in \Sigma_X, r \in \mathbb{R}^+\}$

Interprétation des expressions

Une **fonction mesurable** $f : X \to Y$ est une fonction dont l'image inverse de tout mesurable de Y est un mesurable de X.

Un **Environnement** Γ associe à une liste de variables typées, des valeurs du bon type.

Une **Expressions** $\Gamma \vdash e : t$ calculée dans un environnement Γ s'évalue en une valeur de type t.

Elle est interprétée par une fonction mesurable

$$[\![e]\!] : \Gamma \to t$$

$$\begin{array}{lcl}
[\![c]\!]^{\varphi}_{\gamma} & = & c \\
[\![x]\!]^{\varphi}_{\gamma} & = & \gamma(x) \\
[\![op(e)]\!]^{\varphi}_{\gamma} & = & op([\![e]\!]^{\varphi}_{\gamma}) \\
[\![(e_1, e_2)]\!]^{\varphi}_{\gamma} & = & ([\![e_1]\!]^{\varphi}_{\gamma}, [\![e_2]\!]^{\varphi}_{\gamma})
\end{array}$$

Interprétation des commandes

Un **noyau** $k : X \rightsquigarrow Y$, est une mesure sur Y paramétrée en X

$k : X \times \Sigma_Y \to \mathbb{R}^+$

pour tout $U \in \Sigma_Y$, $k(-, U) : X \to \mathbb{R}^+$ est mesurable

pour tout $x \in X$, $k(x, -)$ est une mesure sur Y

Une **commande** s transforme un environnement en un nouvel environnement.

Elle est interprétée par un noyau :

$$[\![s]\!] : \Gamma \rightsquigarrow \Gamma$$

Interprétation d'une sélection de commandes

$$[\![s]\!] : \Gamma \rightsquigarrow \Gamma$$

$$\begin{array}{lcl}
[\![\mathrm{pass}]\!]_\gamma^\varphi(U) & = & \delta_\gamma(U) \\
[\![x = e]\!]_\gamma^\varphi(U) & = & \delta_{\gamma+\left[x\leftarrow [\![e]\!]_\gamma^\varphi\right]}(U) \\
[\![\mathrm{if}\ e\colon s_1\ \mathrm{else}\colon s_2]\!]_\gamma^\varphi & = & [\![s_1]\!]_\gamma^\varphi(U)\ \mathit{si}\,[\![e]\!]_\gamma^\varphi\ \ \mathit{sinon}\,[\![s_2]\!]_\gamma^\varphi(U) \\
[\![x = \mathrm{sample}(e)]\!]_\gamma^\varphi(U) & = & \displaystyle\int [\![e]\!]_\gamma^\varphi(dv)\ \delta_{\gamma+[x\leftarrow v]}(U) \\
[\![x = f(e)]\!]_\gamma^\varphi(U) & = & \displaystyle\int \mu(dv)\ \delta_{\gamma+[x\leftarrow v]}(U) \\
 & & \mathit{avec}\ \mu = \varphi(f)([\![e]\!]_\gamma^\varphi) \\
[\![\mathrm{factor}(e)]\!]_\gamma^\varphi(U) & = & [\![e]\!]_\gamma^\varphi\ \ \delta_\gamma(U) \\
[\![s_1;\ s_2]\!]_\gamma^\varphi(U) & = & \displaystyle\int [\![s_1]\!]_\gamma^\varphi(d\gamma_1)\ \ [\![s_2]\!]_{\gamma_1}^\varphi(U)
\end{array}$$

Declaration et Inférence

$$
\begin{aligned}
[\![\mathrm{def}\ f(x)\colon s\ \mathrm{return}\ e]\!]\,\varphi &= \varphi + [f \leftarrow \lambda v.\ \lambda U.\ k(v, U)] \\
&\quad \textit{avec}\ k(v, U) = \int [\![s]\!]^{\varphi}_{[x\leftarrow v]}(d\gamma)\ \delta_{[\![e]\!]^{\varphi}_{\gamma}}(U) \\
[\![d_1\ d_2]\!]^{\varphi} &= [\![d_2]\!]^{\varphi_1}\ \textit{avec}\ \varphi_1 = [\![d_1]\!]^{\varphi}
\end{aligned}
$$

$$
\begin{aligned}
[\![\mathrm{infer}(f,\ e)]\!]^{\varphi}_{\gamma}(U) &= \begin{cases} \dfrac{\mu(U)}{\mu(\top)} & \textit{si}\ 0 < \mu(\top) < \infty \\[2ex] \text{Erreur} & \textit{sinon} \end{cases} \\
&\quad \textit{avec}\ \mu(U) = \varphi(f)([\![e]\!]^{\varphi}_{\gamma})
\end{aligned}
$$

Exemples

# mu defined on [a, b] x = sample(mu)	≡	x = sample(Uniform(a, b)) observe(mu, x)

def coin():
    x = sample Bernoulli(0.5)
    return x
	≡	def coin():
		    x = sample(Gaussian(0, 1))
		    return x > 0

factor(a); ... ; factor(b)	≡	... ; factor(a * b)
x = sample(dx); y = sample(dy)	≡	y = sample(dy); x = sample(dx)

Sémantique

Exercices

Exercices : Calculer la sémantique des programmes suivants

```
def stop(p:float):
    n = 0
    while sample(Bernoulli(p)):
        n = n+1
    return n
```

```
def flip(p):
    x = sample(Bernoulli(p), name='x')
    y = sample(Bernoulli(p), name='y')
    assume(x!=y)
    return x
```

```
def coin(obs: list[int]) → float:
    p = sample(Uniform(0, 1))
    for o in obs:
        observe(Bernoulli(p), o)
    return p

with ImportanceSampling(num_particles=10000):
    dist: Categorical = infer(coin, [0, 0, 0, 0, 0, 0, 0, 0, 1, 1])
```

Sémantique

L'ordre supérieur

Quelle sémantique pour l'ordre supérieur ?

Exemples :

Loi d'un modèle récursif.
Loi d'une fonction affine, polynomiale, analytique qui décrit un nuage de points
model : data → (t→ t).

```
def flipRec(p, s=' '):
    x = sample(Bernoulli(p), name='x')
    y = sample(Bernoulli(p), name='y')
    return x if (x!=y) else flipRec(p, s+'_')
```

Aumann (1961)

Il existe X et Y tels que, quelle que soit la tribu sur l'espace des fonctions mesurablce $\mathbf{Meas}(X, Y)$, l'évaluation ev : $\mathbf{Meas}(X, Y) \otimes X \to Y$ n'est pas une fonction mesurable.

Conclusion

Les mesures et noyaux ne conviennent pas à la sémantique des PPL d'ordre supérieur.

(Heureusement, d'autres modèles ont été mis à jour les 10 dernières années (QBS, Cones, . . .)).

La catégorie des espaces mesurables est monoidale symétrique mais pas fermée.

Par l'absurde : Supposons que l'évaluation $\forall X, Z, \mathrm{ev} : Z^X \otimes X \to Z$ soit mesurable quelques soient X, Z.

Espaces mesurables X est $\mathbb{R}$ muni la tribu $\Sigma_X = \mathcal{P}(X)$ de toutes les parties et Y est $\mathbb{R}$ muni de la tribu dénombrable-codénombrable enfendrée par les ensembles dénombrables et dont le complémentaire est dénombrable (close par union et intersection dénombrables).

Fonction diagonale : $h : \begin{cases} (\mathbb{R} \times \mathbb{R}, \mathcal{P}(\mathbb{R}) \otimes \mathcal{C}(\mathbb{R})) & \to \{0, 1\} \\ (x, y) & \mapsto 1 \text{ if } x = y \\ & \mapsto 0 \text{ otherwise} \end{cases},$

$\Lambda(h) : (\mathbb{R}, \mathcal{P}(\mathbb{R})) \to (\{0, 1\}^{\mathbb{R}}, \Sigma_{2^Y})$ est **measurable**

$h = \mathrm{ev} \circ \Lambda(h)$ est **measurable** car composée de fonctions mesurables.

$\Delta = \{(x, y) \in \mathbb{R}^2 \mid x = y\} = h^{-1}(1)$ est measurable dans $\mathcal{P}(\mathbb{R}) \otimes \mathcal{C}(\mathbb{R})$.

La catégorie des espaces mesurables est monoidale symétrique mais pas fermée.

Par l'absurde : Supposons que l'évaluation $\forall X, Y, \mathrm{ev} : Y^X \otimes X \to Y$ est measurable.

Alors $\Delta = \{(x, y) \in \mathbb{R}^2 \mid x = y\} = h^{-1}(\{1\})$ est **measurable** dans $\mathcal{P}(\mathbb{R}) \otimes \mathcal{C}(\mathbb{R})$.

Proposition : Si $W \in \mathcal{P}(\mathbb{R}) \otimes \mathcal{C}(\mathbb{R})$, alors, il existe $B \subseteq \mathbb{R}$ dénombrable tel que

S'il existe $(x, y) \in W$ tel que $y \notin B$, alors $\forall z \notin B$, $(x, z) \in W$.

Preuve : satisfait par les ensembles mesurables de la base et close par union et intersection dénombrables.

Conséquence : Δ satisfait cette propriété, soit B l'ensemble dénombrable. Comme B est dénombrable, on peut trouver $(x, x) \in \Delta$ et $x \notin B$. Comme B est dénombrable, on peut trouver $z \notin B$ et $z \neq x$, donc $(x, z) \in \Delta$ et $(x, z) \notin \Delta$. On arrive à une **contraditonc**

Conclusion

Un programme probabiliste modélise une variable aléatoire

Les algorithmes d'inférence permettent de calculer leurs lois, de les simuler,...

La sémantique permet de prouver la correction des programmes, des transformations de programmes, et d'assurer la validité de l'exécution d'un langage de programmation.

Demain :

- Implémentation d'un PPL
- Algorithmes d'inférence avancés
- Sémantique à densité