Partial inconsistency and vector semantics: sampling, animation, and program learning

Michael Bukatin

Nokia Corporation, Cambridge, MA

Joint work with Ralph Kopperman and Steve Matthews

- - -

29th Summer Conference on Topology and its Applications, Special Session on Asymmetry and its Applications, July 24, 2014

Electronic coordinates

These slides are linked from my page on partial inconsistency and vector semantics of programming languages:

http://www.cs.brandeis.edu/~bukatin/partial_inconsistency.html

E-mail:

`bukatin@cs.brandeis.edu`

Abstract

Scott domains tend to get embedded into vector spaces in the context of partial inconsistency. One example of this is partially inconsistent interval numbers, another is that if one allows negative probability, then probabilistic power domains are embedded into vector spaces of signed measures (Kozen semantics for probabilistic programs).

This is very attractive from the applied point of view, especially if not only denotations of programs, but single runs form vector spaces. This, in turn, points to sampling-based operational semantics of probabilistic programs and to computations via generalized animation, which can be viewed as fuzzy sampling.

We overview rapid progress in the probabilistic programming languages and architectures based on Markov Chain Monte Carlo sampling and recent results suggesting that program learning is more tractable in this setting than in the setting of deterministic programs.

Outline

1. Vector semantics of probabilistic programs
2. Sampling semantics
   - Generalized animations
   - Program learning
3. MCMC sampling: recent progress
   - Probabilistic programming languages
   - Program learning: sampling the samplers
   - MCMC and neuro

Partial inconsistency landscape

Group and vector space semantics of programming languages compatible with Scott domain semantics.

Scott domains tend to get embedded into vector spaces in the context of partial inconsistency.

One example of this is partially inconsistent interval numbers.

Another example is that if one allows **negative probability**, then probabilistic power domains are embedded into vector spaces of signed measures (Kozen semantics for probabilistic programs).

Kozen semantics

One can think about probabilistic programs as transformers from the probability distributions on the space of inputs to the probability distributions on the space of outputs.

It is fruitful to replace the space of probability distributions by the space of signed measures.

D. Kozen, Semantics of Probabilistic Programs, Journal of Computer and System Sciences **22** (3), 328–350 (1981)

Kozen semantics

One defines $\nu < \mu$ iff $\mu - \nu$ is a positive measure.

The space of signed measures is a vector lattice (a Riesz space) and a Banach space, so people call this structure a Banach lattice.

Denotations of programs are continuous linear operators with finite norms.

The probabilistic powerdomain is embedded into the positive cone of this Banach lattice.

Hahn-Jordan decomposition

$\mu^+ = \mu \vee 0, \mu^- = \mu \wedge 0, \mu = \mu^+ + \mu^-$, holds, since it's a theorem for all lattice-ordered groups.

Defining $\nu \sqsubseteq \mu$ iff $\nu^+ \leq \mu^+$ and $\nu^- \leq \mu^-$, one also obtains $\mu = \mu^+ \sqcup \mu^-$, making this an instance of the "bilattice pattern".

Hilbert space

The structure of Hilbert space on signed measures can be obtained via reproducing kernel methods.

Chapter 4 of A. Berlinet, C. Thomas-Agnan, Reproducing Kernel Hilbert Spaces in Probability and Statistics, Kluwer Academic Publishers, Boston (2001)

Linear combinations of programs

Take $0 < \alpha < 1$ and **random** being a generator of uniformly distributed reals between 0 and 1.

if random $< \alpha$ **then P else Q** yields a linear combination of programs **P** and **Q**.

To allow negative coefficients one needs to consider computing a negative and a positive channel in parallel (computations are marked as negative or positive).

Program learning, program decompositions, program transformations?

The situations when one can consider linear combinations of single execution runs should be especially attractive.

Sampling semantics is one possibility here – the input is literally a sampling of a probability distribution, and so is the output, and the whole thing is, in some sense, a "probabilistic dataflow architecture".

To implement linear combinations of probabilistic programs with positive coefficients one can simply execute those programs in parallel, and the values of coefficients can be controlled by changing the relative execution speed of those programs.

To allow negative coefficients one again needs to use a negative and a positive channel

Fuzzy sampling and animations

Fuzzy samplings where points are taken with real coefficients might be even more attractive.

One can think about them as generalized animations, where points might be indexed by a more sophisticated index set than a discretized rectangle.

Here one might allow negative coefficients and avoid the need for a separate channel (speaking in terms of animation this means that 0 is at some grey level, between black and white).

One can leverage existing animations, digital and physical (such as light reflections and refractions in water), as computational oracles.

Expressive power

Music is a fast animation (typically on the index set of 2 points for usual stereo).

Very short programs can express complex dynamics (demo at the end of the talk if there is time).

A typical DJ setup and a typical VJ software (e.g. Resolume), alllow to take linear combinations of transformed music and animations.

Learning the animations

Scott Draves, The Electric Sheep Screen-Saver: A Case Study in Aesthetic Evolution, in Applications of Evolutionary Computing, LNCS 3449, Springer 2005, pp.458–467, http://draves.org/evomusart05/

(Demo at the end of the talk if there is time)

The animations tend to be non-brittle; mutations and crossover tend to produce something meaningful.

A way to incorporate aesthetic criteria into software systems.

Rejection sampling

Consider computing the area under a curve via Monte Carlo simulation.

Say, between x_0 and x_1, and that for all $x \in [x_0, x_1]$, $0 \leq f(x) \leq y$.

Take random numbers X and Y uniformly in $[x_0, x_1]$ and $[0, Y]$.

If $Y \leq f(X)$, the sample is accepted, otherwise it is rejected.

The area is approximated as the ratio of accepted samples multiplied by the area of the rectangle in question, $Y * (x_1 - x_0)$.

Markov Chain Monte Carlo

Informally speaking, the next sample depends on the previously accepted sample.

So, each steps transforms the empirical distribution, and we want the process to converge to its equilibrium distribution (fixpoint).

A variety of flavors.

A large rapidly evolving field (theory and applications).

I am just going to list some especially remarkable examples of recent progress (a small and rather subjective subselection).

Sampling semantics

Variables are probability distributions (usually given by a stream of Monte Carlo samples).

Program is written in terms of distribution transformers.

One of the first well-known languages of this kind was Church, written in Scheme and being a probabilistic dialect of Scheme.

Now we have a powerful successor of Church called Venture.

The variety of probabilistic programming systems

http://probabilistic-programming.org

Venture:

http://probcomp.csail.mit.edu/venture/

Details of Venture engine (key aspects of implementation, the theory and practice of efficient inference):

Vikash Mansinghka, Daniel Selsam, Yura Perov, Venture: a higher-order probabilistic programming platform with programmable inference, 2014, http://arxiv.org/abs/1404.0099

The MIT Probabilistic Computing Project

http://probcomp.csail.mit.edu/

Various projects leveraging power of Venture, including

Generative Probabilistic Graphics Programming:

http://probcomp.csail.mit.edu/gpgp/

(Computer vision as a Bayesian inverse problem to computer graphics)

Computer vision as an inverse problem to rendering

Vikash Mansinghka, Tejas Kulkarni, Yura Perov, Joshua Tenenbaum, Approximate Bayesian Image Interpretation using Generative Probabilistic Graphics Programs, 2013, http://arxiv.org/abs/1307.0060

(CAPTCHAs, 3D road models)

Tejas Kulkarni, Vikash Mansinghka, Pushmeet Kohli, Joshua Tenenbaum, Inverse Graphics with Probabilistic CAD Models, 2014, http://arxiv.org/abs/1407.1339

(3D human pose estimation, 3D object shape reconstruction)

Probabilistic program learning

Sampling the samplers might be easier than sampling the underlying objects directly.

Sampling probabilistic programs might be easier than sampling deterministic programs.

If necessary, one can sample probabilistic programs which in turn sample other probabilistic or deterministic programs.

Example of probabilistic program learning

Roger Grosse, Ruslan Salakhutdinov, William Freeman, Joshua Tenenbaum, Exploiting compositionality to explore a large space of model structures, 2012, http://arxiv.org/abs/1210.4856

Considering a context-free grammar describing a large class of models and using a relatively simple greedy strategy of traversing a fragment of the space of all such models proves surprisingly effective.

This paper is what got me really convinced that "sampling the samplers" is likely to work.

Natural laguage output

Inspired by this work: M. Ganesalingam, W. T. Gowers, A fully automatic problem solver with human-style output, 2013, http://arxiv.org/abs/1309.4501

"This paper describes a program that solves elementary mathematical problems, mostly in metric space theory, and presents solutions that are hard to distinguish from solutions that might be written by human mathematicians."

Natural language output for learned models

James Robert Lloyd, David Duvenaud, Roger Grosse, Joshua Tenenbaum, Zoubin Ghahramani, Automatic Construction and Natural-Language Description of Nonparametric Regression Models, 2014, http://arxiv.org/abs/1402.4304

Does brain perform probabilistic sampling?

E.g. research by Wolfgang Maass,
http://www.igi.tugraz.at/maass/publications.html

Reference [221], W. Maass, Noise as a resource for computation and learning in networks of spiking neurons, Proceedings of the IEEE, **102**(5):860-880, 2014.

This scheme can leverage noise, and so allows to use low-powered circuits with high level of noise (errors).

see also his references [207], [208], [219] (starting from 2011)

Electronic coordinates

These slides are linked from my page on partial inconsistency and vector semantics of programming languages:

http://www.cs.brandeis.edu/~bukatin/partial_inconsistency.html

E-mail:

`bukatin@cs.brandeis.edu`