Probabilistic CCP

Prakash Panangaden
McGill University

Outline

- CCP : from logic to computation
- Closure operators
- Probabilistic CCP

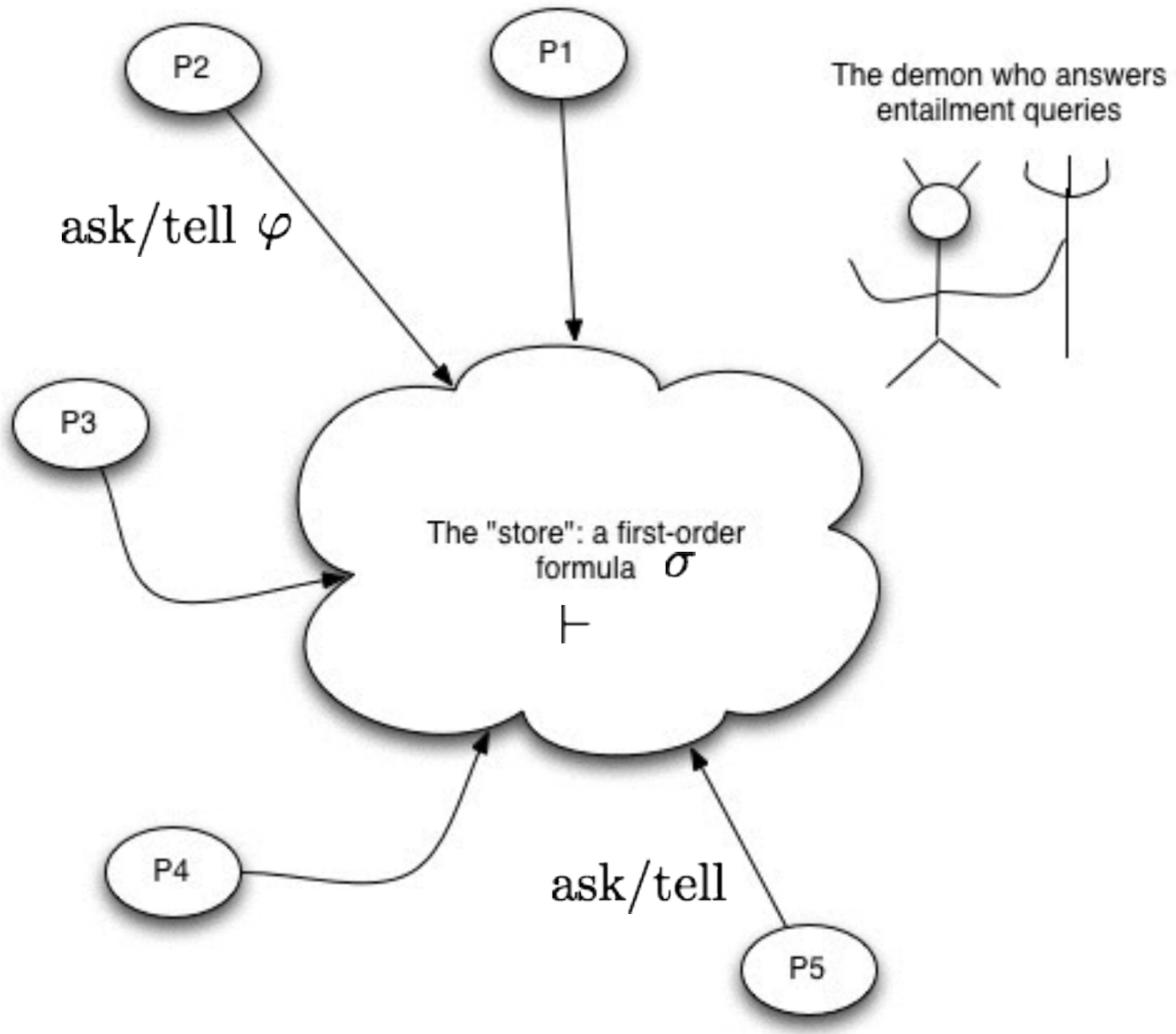
Thanks

- To a great gang of collaborators: Samson Abramsky, Costin Badescu, Richard Blute, Kostas Chatzikokolakis, Gheorghe Comanici, Alex Bouchard-Côté, Philippe Chaput, Vincent Danos, Josée Desharnais, Monica Dinculescu, Abbas Edalat, Norm Ferns, Vineet Gupta, Chris Hundt, Radha Jagadeesan, Dexter Kozen, Kim Larsen, François Laviolette, Radu Mardare, Catuscia Palamidessi,Joelle Pineau, Gordon Plotkin, and Doina Precup.
- Also to my CCP collaborators: Nax Mendler, Robert Seely, Phil Scott, Vijay Saraswat, Martin Rinard, Frank Valencia and Sophia Knight.

Special Thanks to

- Dexter for putting together this awesome conference
- The PC chairs (Bart, Sam and Alexandra) and the whole PC
- Mike Mislove: the heart and soul of MFPS
- Cornell, where I found the love of my life.

Concurrent Constraint Programming



CCP processes

$\mathsf{ask}(\phi)$: does the current store (σ) entail ϕ?

$\mathsf{tell}(\phi)$: add ϕ to the current store.

$P_1 || P_2$: run P_1 and P_2 in parallel.

$\mathsf{new}\ X\ \mathsf{in}\ P$: fresh local variable; $\nu X.P$.

recursive procedures

Underlying (first-order) language and $\vdash$. The constraint system.

The demon answers entailment queries somehow.

If $\sigma \nvdash \phi$ then process *suspends*.

Examples

More detailed syntax for ask: $\mathsf{ask}(\phi) \to P$.

$$\mathsf{tell}(X = 1)||(\mathsf{ask}(X > 0) \to (\mathsf{tell}(Y = 17)||\mathsf{ask}\ldots)$$

Henceforth, I will skip ask and tell.

$$P_1 : (X > 1)||[(Y > 0) \to ((X > 2)||((Y > 1) \to (X = 17)))]$$

$$P_2 : (X > 1) \to [(Y > 0)||(X > 2) \to (Y = 2)]$$

When P_1 and P_2 are run in parallel they will engage in a *dialogue*.

Ask is a *synchronizer*: perhaps it should have been called *await*.

Constraint system examples

Kahn-McQueen style dataflow: Define a language with function symbols for *cons*, *first* and *rest* and appropriate predicate symbols and entailment relations to model streams.

CCP is an **asynchronous** programming paradigm.

Herbrand: terms in a first-order language with equality.
Constraints are equality statements or their conjunctions.
Example entailment:
$f(X,Y) = f(Z, g(U,V)) \vdash (X = Z) \wedge (Y = g(U,V))$

Rational intervals.

The partial order of the denotational semantics
is now something with which one can program.

Programs update the store, but only in a monotonic way.

Demons answer entailment queries: the store can
be thought of as the theory that it generates.

The collection of possible stores can be ordered by
inclusion of the theories they define.

This collection forms a complete algebraic lattice
if we make some assumptions about constraint systems.

Closure operators: a digression

Given a poset $(S, \leq)$, a **closure operator** is a monotone function $f : S \to S$ such that $\forall x \in S, x \leq f(x)$ and $f(x) = f(f(x))$: inflationary and idempotent.

A closure operator can be reconstructed from its set of fixed points:

$$f(x) = \min(x\uparrow \bigcap \hat{f})$$

where $\hat{f}$ is the set of fixed points of f, and $x\uparrow$ is the set of elements in S greater than or equal to x.

One can think of closure operators as functions or as sets.

We will consider closure operators on complete algebraic lattices.

We will work with Scott-continuous closure operators.

A *set* of closure operators on a lattice has a least common fixed point.

Consider $f, g : L \to L$. What is the least common fixed point?

$$\mathsf{lub}\{f(\bot), g(f(\bot)), f(g(f(\bot)), g(f(g(f(\bot))), \ldots\}$$

Hmmm, looks like a "dialogue."

We can construct the least common fixed point *above* any given x, using x in place of $\bot$ above.

This gives a function — we will write it as $f||g$ — which is a closure operator. In fact: $\widehat{f||g} = \hat{f} \cap \hat{g}$.

Denotational Semantics

A store is modelled by the theory it generates: an entailment-closed set of formulas. The inclusion order is the information order.

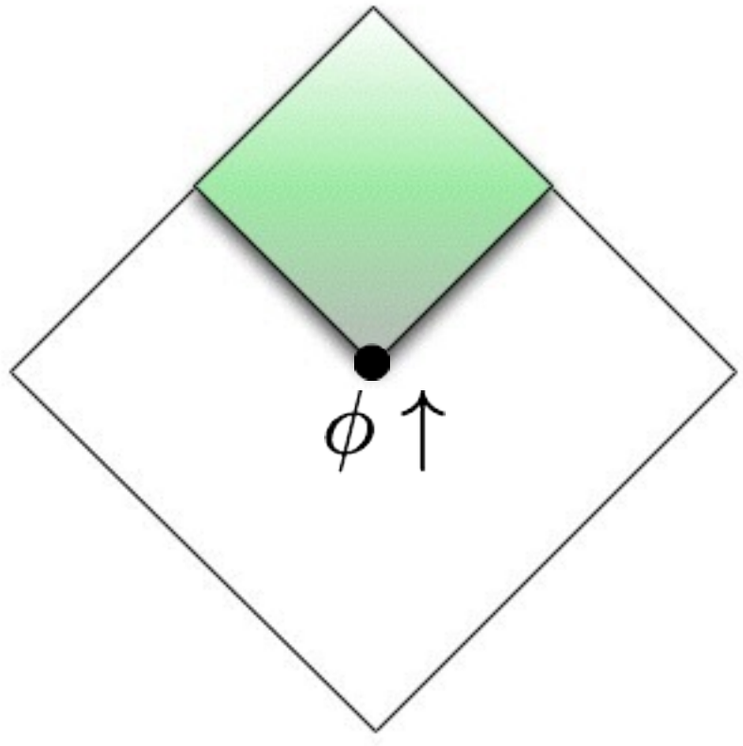
A CCP process *adds* information to the store. If we run the same process a second time it will not add anything new to the store.

A CCP process denotes a (Scott-continuous) closure operator.

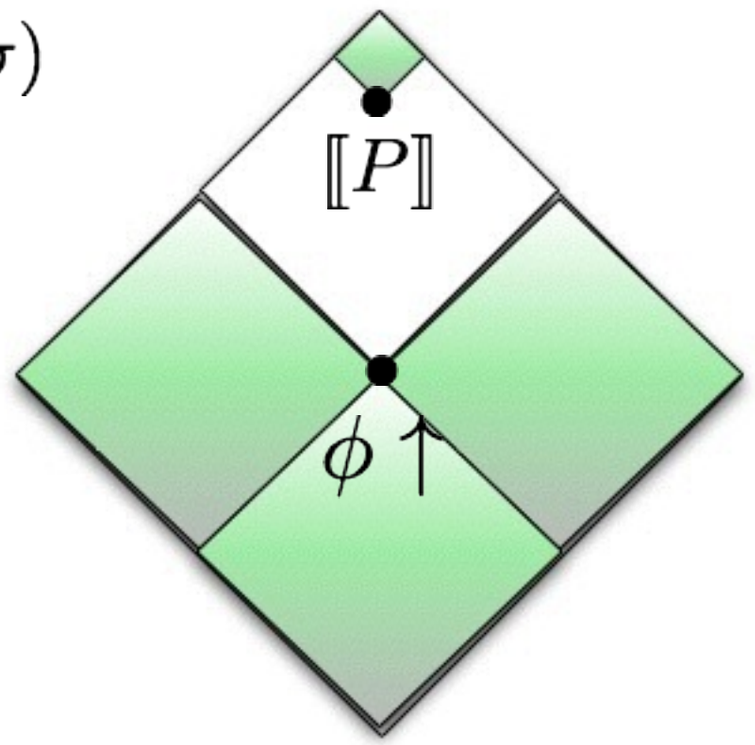
One can think about closure operators as functions *or* as sets of fixed points.

The shaded region shows the fixed points.

$$[\![\mathsf{tell}(\phi)]\!] = \lambda\sigma.(\sigma \sqcup (\phi)\uparrow)$$



$$[\![\mathsf{ask}(\phi) \to P]\!](\sigma) = \text{if}(\sigma \sqsupseteq \phi\uparrow) \text{ then } [\![P]\!](\sigma)$$



$$[\![P_1||P_2]\!] = [\![P_1]\!]||[\![P_2]\!] \qquad \text{OR} \qquad \widehat{[\![P_1||P_2]\!]} = \widehat{[\![P_1]\!]} \cap \widehat{[\![P_2]\!]}$$

$$[\![\nu X.P]\!] = \exists X.[\![P]\!]$$

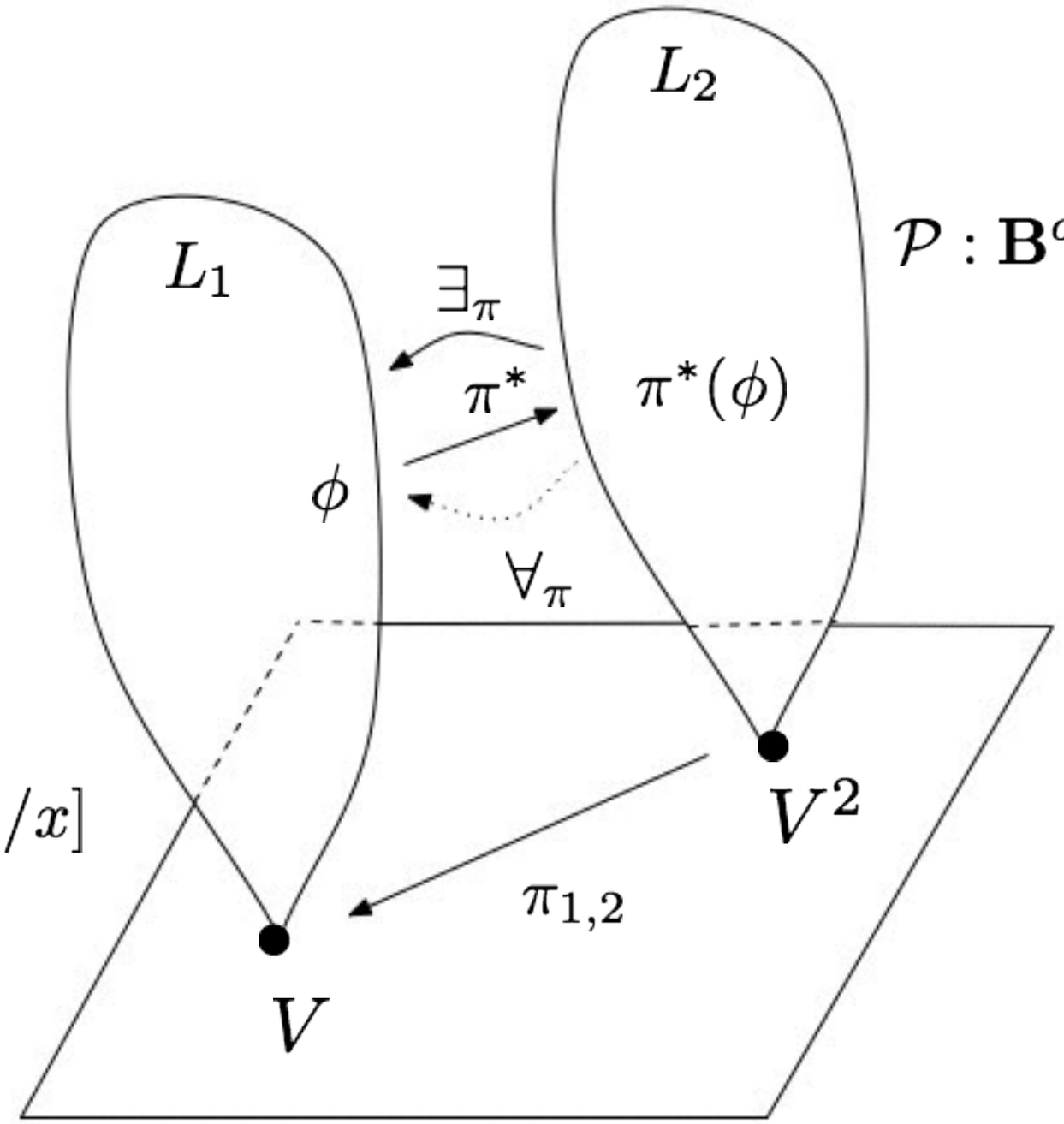
What is $\exists$ on closure operators?

Straightforward to define $\exists_X$ on stores.

$$\widehat{[\![\nu X.P]\!]} = \{\sigma | \exists \sigma' \in [\![P]\!].\exists_X.\sigma = \exists_X.\sigma'\}$$

In various papers we organized closure operators into a *hyperdoctrine* with $\exists$ as left-adjoint to substitution.

V basic data type, Var variables, function and relations symbols, terms and formulas as usual. L_n formulas with n free variables.



$\mathcal{P} : \mathbf{B}^{op} \to \wedge - \mathbf{Preord}$

$\exists_\pi \dashv \pi^* \dashv \forall_\pi$

$\pi^*(\phi) = \phi[\pi(\langle x, y \rangle)/x]$

The base $\mathbf{B}$ is a cartesian category generated by V with additional arrows for function symbols.

Generalized existential quantifiers

$$[\exists_f \phi](x) \text{ means } \exists y.[(x = f(y)) \wedge \phi(y)]$$

$$[\exists_\pi \phi](x) = \exists u, v.[(x = \pi(\langle u, v \rangle)) \wedge \phi(u, v)] = \exists v.\phi(x, v)$$

$$[\exists_\Delta \phi](x, x') = \exists u.[(\langle x, x' \rangle = \Delta(u)) \wedge \phi(u)]$$

which is just

$$\phi(x) \wedge (x = x')$$

If we take ϕ to be true we get

$$\exists_\Delta \mathsf{true} = (x = x').$$

The Lattice Hyperdoctrine

$\mathcal{F} : \mathbf{meet - Preord} \rightarrow \mathbf{CAL}$

A *filter* is a meet-closed, upward closed subset of A.

$X \uparrow = \{\psi : \phi_1, \ldots, \phi_n \vdash \psi, \phi_i \in X\}$.

$\mathcal{F}(A)$ is the set of filters on A ordered by inclusion, it forms a complete algebraic lattice.

If $r : A \rightarrow B$ is a monotone function define $\mathcal{F}(r) : \mathcal{F}(A) \rightarrow \mathcal{F}(B)$ by $\mathcal{F}(r)(u) = r(u) \uparrow$.

All the hyperdoctrine structure is preserved by $\mathcal{F}$.

The Closure Operator Hyperdoctrine

$\mathcal{C}l : \mathbf{CAL}^{adj} \to \mathbf{CAL}^{adj}$

$\mathbf{CAL}^{adj}$: Objects are $\mathbf{CAL}$ objects, morphisms are adjoint pairs
$f : A \to B$; $g : B \to A$ with $f \dashv g$.

$\mathcal{C}l(A)$ is the lattice of all closure operators on A.

$\mathcal{C}l$ preserves all the hyperdoctrinal structure.

The existential quantifier here is exactly what is used to model $\nu x.A$ in the closure operator denotational semantics.

The lub operation is exactly what one uses to model parallel composition.

Probabalistic CCP

New ingredient: **choose** X **from** Dom **in** P

X: local variable, scope is P

Dom is a finite set

- Random variables are hidden
- Each random variable has its own independent probability distribution.

Basic Example

choose X **from** $\{0, 1, 2, 3\}$ **in**

$$[\mathsf{ask}((X = 0) \vee (X = 1)) \rightarrow \ \mathsf{tell}(a)] \ || \ [\mathsf{ask}(X = 2) \rightarrow \ \mathsf{tell}(b)]$$

Produces a with probability 0.5 and b with 0.25.

and true with probability 0.25.

Constraints and conditioning

choose X **from** $\{0, 1, 2, 3\}$ **in**

$\mathsf{tell}(X \leq 2) \;||\; [\mathsf{ask}((X = 0) \vee (X = 1)) \rightarrow \; \mathsf{tell}(a)] \;||$
$[\mathsf{ask}(X = 2) \rightarrow \; \mathsf{tell}(b)]$

Produces a with probability 0.5 and b with probability 0.25, however, it cannot produce **true** because of the constraint on X.

Inconsistent stores are discarded and the probabilities are renormalized.

Probability of $a = \frac{2}{3}$ and probability of $b = \frac{1}{3}$.

The semantics gives the probabilities
conditioned on obtaining a consistent store.

Notational change

I will strop writing ask, tell and $||$.

$$(X \leq 2), [((X = 0) \vee (X = 1)) \rightarrow a], [(X = 2) \rightarrow b]$$

instead of

$$\mathsf{tell}(X \leq 2) \;||\; [\mathsf{ask}((X = 0) \vee (X = 1)) \rightarrow \; \mathsf{tell}(a)] \;||\; [\mathsf{ask}(X = 2) \rightarrow \; \mathsf{tell}(b)]$$

Independence of **choose**

choose X **from** $\{0,1\}$ **in** $[X = Z]$,

choose Y **from** $\{0,1\}$ **in** $[(Z = 1) \to (Y = 1)]$

Four possible execution paths but one is inconsistent with the constraints.

We get the following distribution on the visible variable Z:
$Z = 0(\text{prob} = \frac{2}{3}), \quad Z = 1(\text{prob} = \frac{1}{3})$.

We can get any distribution with rational probabilities
on a finite set this way.

Derived combinator:
choose X **from** Dom **with** f **in** P.

Extended example
Stochastic Petri Nets

1-safe nets, places may have at most one token.

Time is in discrete steps, modelled by a recursive call.

Places randomly choose to which transition they send their token, or not to send it anywhere.

Similarly, an empty place chooses from which transition to accept a token, or not to accept any token.

Constraints on transitions ensure that either all pre and post conditions choose it or none do.

A new marking is computed and a recursive call is made with the new marking.

What about recursion?

We get distributions on continuous spaces.

$$U(l,u,z) :: z \in [l,u],$$
$$\textbf{choose } X \textbf{ from } \{0,1\} \textbf{ in}[$$
$$(X = 0) \to U(l,(u+l)/2,z),$$
$$(X = 1) \to U((u+l)/2,u,z)]$$

Defines the uniform distribution on $[0,1]$.

Actually it defines a measure on the space of binary sequences but this is Borel isomorphic to $[0,1]$.

What about conditioning in the presence of recursion?

$$U(0, 1, z), (z = 0)$$

Intuitively probability of $(z = 0) = 1$, since $z = 0$ is the only possible output.

However, U defines a distribution which assigns probability 0 to $(z = 0)$.

When we try to normalize we get nonsense.

Use the "domain" Luke!

Unwind and consider "finite approximations"
of the recursive program.

The approximation U_n yields $z = 0$ with positive probability,
so $U_n(0, 1, z), (z = 0)$ gives $(z = 0)$ with probability 1.

In the limit we get $(z = 0)$ with probability 1.

Time for a confession to the truth and
reconciliation commission:

We were never completely happy with the semantics
and worked on approximation and metrics instead
for the next 15 years.

Disintegration

The right way to understand this would be via disintegrations.

> One way to engage in rigorous, guilt-free manipulation of conditional distributions is to treat them as disintegrating measures. . . .
> - Chang and Pollard 1997

Alas, I did not learn about them until yesterday.

Kinky example

$$\textbf{new}\ X\ [U(0,1,X),(a \to (X=0)),((X>0) \to b)]$$

Intuitively: **if** a **else** b.

If a is present then b will not be produced,

but if a is not present, b will be produced.

Recall, the semantics throws out inconsistent stores.

What if we ran: **if** a **else** b with **if** b **else** a?

Our semantics would say that this gives no defined answer.

Some remarks on the semantics

Operational semantics in terms of LTS as in CC languages.

Probabilities are computed for finite programs
(by counting paths in the LTS) and normalized.

A preorder is defined for "partial" programs that
arise as syntactic unwindings of recursive programs.

The unwindings give a directed set and we define the
probability of obtaining an output via a limiting process.

The limiting process may not converge in which case
we say that the program is undefined.

A denotational semantics was defined but only for
recursion-free programs.

Integration example

$f : [a, b] \to [c, d]$ a Riemann-integrable function.

$$P :: U(a, b, X), U(c, d, Y),$$

$$((Y < f(X) \to A), (Y > f(X) \to B))$$

Let $R = (b - a) * (d - c)$.

Our semantics gives: probability of $A = (1/R) \times \int_a^b (f - c)$.

More precisely, we get a lower Riemann sum for A and an upper Riemann sum for B.

Not the Cantor set

$$NC(l,u,z) :: NC(l,(u+2l)/3,z),$$
$$NC((2u+l)/3,u,z),$$
$$((u+2l)/3 < z < (2u+l)/3) \rightarrow \textbf{NotCantor}.$$

This program produces the token **NotCantor** if z is not in the Cantor set.

If we run $U(0,1,X), NC(0,1,X)$ we get **NotCantor** with probability 1.

Our semantics successfully produces the right answers for these examples.

Conditioning on a set of measure 0

Use *computational* approximation to provide answers, where Radon and Nikodym would give up.

new X, Y **in** $[U(0,1,X), U(0,1,Y), X = Y]$

Intuition: uniform measure on the diagonal.

Probability theory: undefined.

Our semantics: gives the intuitive answer.

Note: it does **not** depend on the two recursions being unwound at the same "rate".

The last example

$V(l,u,X) :: z \in [l,u]$,

choose X **from** $\{0,1\}$ **with** $\{1/3, 2/3\}$ **in** [

$(X = 0) \to V(l, (2l+u)/3, z)$,

$(X = 1) \to V((2l+u)/3, u, z)]$

Gives the uniform distribution on $[l,u]$.

Now consider $U(0,1,X), V(0,1,Y), (X = Y)$.

Does *not* give the uniform distribution on the diagonal.

$\mathsf{Pr}((0,0.1)) < \mathsf{Pr}((0.9,1.0))$.

Get a "fractal-like" distribution on the diagonal.

Conclusions

Conditional probability is at the "heart" of probabilistic reasoning.

Time for serious dialogue between machine learning researchers and programming language researchers.

Probabilistic CCP still has some good ideas especially for representing continuous spaces and distributions.

We know so much more now, perhaps it is time to revisit the semantics.

Delighted to see the work described by Andy and his associates at MSR.

Thrilled to have Doina Precup and Joelle Pineau as colleagues.

Future Work

I'll tell you about it when I've done it.

Thanks for listening.