April 17, 2000

James Cheney

Overview and Discussion

Theory

Probability, Programming, and Type
My background

I’ve been working on term compression

Given a term (proof, exp, program, ...), how do you compress it?

Short answer: use gzip

Long answer: Adapt and extend standard compression algorithms to terms

e.g. Huffman coding, Lempel-Ziv parsing
But ultimately aiming for automatic data format generation and
implementation only on the implementation angle

Example: Compressing giant datatypes means writing giant, for

to implement and reason about by hand

Models interesting enough for good compression are too complex

Recent work based on statistical models of terms

Story so far
Algorithms: average cases, compression/error-correcting coding, AI: Bayes nets, probabilistic NL parsing, fuzzy logic

Algorithms: hashing, quicksort, crytopgraphy

Probability is the theory used to reason about it

Randomness is a core concept in computer science

Why is this interesting?
Why is this not trivial?

Has not been studied from the perspective of type theory

Randomness is (seemingly) inherently nonconstructive

Probability theory only compatible with decidable fragment of constructive logic

Not clear if/how probability theory generalizes to include undecidable event spaces (such as \( \mathbb{R} \))

Yet necessary to reason about, e.g., Ensemble randomized gossip
Theorems we’d like to prove

∀p > 0.∀L : {0, 1} List.P(0) = p ∧ L ∼ P ⇒ E(#0(L)) = \frac{1}{p}

∃C : PrefixCode(T, B).∀X : T.X ∼ P ⇒ H(X) ≤ E(|C(X)|) ≤ H(X) + 1

Problem: Sloppy notation. Want to make more concrete in typed setting

Examples: What type does E have? P? H?
Computational approach

Consider ML ADTs for ’a rand and ’a prob

Random objects and probability distributions are constructed from simple objects and from outside sources of data

Claim: can build useful, complex random objects and distributions up from simple ones

Example: source (fn ar => (ar.choose(),ar.choose())) has type ’a rand -> (’a*’a) rand

Claim: Computational theory of probability should support these types (plus other constraints)
Random values

signature RAND =

  sig
    type 'a rand
    type seed = (int*int)
    val bool_rand : seed -> bool rand
    val int_rand : seed -> int rand
    (* ... *)
    val choose : 'a rand -> 'a
    val source : ('a rand -> 'b) -> 'a rand -> 'b rand
  end
Probability distributions

signature PROB =
  sig
    type 'a prob
    val bernoulli : real -> bool prob
    val uniform_range : int * int -> int prob
    (* ... *)
    val prob : ''a prob -> ''a -> real
    val expect : ('a -> real) -> 'a prob -> real
    (* ... *)
    val generate : 'a prob -> real rand -> 'a rand
    val sample : 'a rand -> int -> 'a prob
  end
Logical approaches

There are two main paths to mixing probability theory and logic:

- **External** (probabilistic logics [Nilsson])

  Assign probabilities (or ranges) to predicates by calculating the probability that the statement is true in a model selected according to some distribution

- **Internal** (logics of probability [Fagin, Halpern])

  Add axioms for probability, along with basic axioms for reasoning about linear inequalities, to first order logic.
External approaches

There are three levels at which we can introduce a general form of uncertainty into NuPRL.

- Make *judgements* uncertain: \( H \vdash^\alpha G \equiv P(H \vdash G) = \alpha \)

- Make *types/propositions* uncertain: \( x_i :^{\alpha_i} H_i \vdash G^\beta \equiv P(H_i) = \alpha_i \vdash P(G) = \beta \)

- Make *values* uncertain: \( x =^\alpha_A y \equiv P(x = y \in A) = \alpha \)

in decreasing order of disruptiveness.
Problems with these approaches

Probabilizing judgements or types seems like overkill

Probability is not truth-functional – you get consistent ranges rather than fixed values

Computing the ranges is hard – large matrix equations

Moreover, this approach requires adding probability theory to the meta-theory, along with real analysis

That would be bad, because it makes the meta-theory nonconstructive! $A \lor \neg A \equiv P(A) + P(\neg A) = 1$
What do we really want?

We really want to be able to reason about uncertain values.

Probabilizing $x \equiv^\alpha y \in A$ still imports classical logic into NuPRL.

Reasoning about probability still “external”; the $\alpha$ is not connected with anything in NuPRL.

We really want an internal system for constructive probabilistic reasoning.
Internal approach

Let’s take the programming approach from earlier as a model

Define types $T$ Prob and $T$ Rand in NuPRL, with similar basic constructors and operations

Let $P_i$ be suitable probability axioms, such as those of Kolmogorov.

$$T \text{ Prob} = \{P : T \text{ Event} \rightarrow [0, 1]; \text{Decidable}(T); p1 : P_1(P, T); p2 : P_2(P, T); p3 : P_3(P, T)\}.$$

$$T \text{ Rand} = \text{Unit} \rightarrow T.$$
There's only one little problem...

work on such theories] might make handy tactics. Joseph Halpern has done a lot of
And there are decision procedures for probabilistic theory which
allow us to relate probabilistic distributions and random values. The constructors and operations from the programming model
results of classical probabilistic theory. For appropriate domains, we should be able to prove all the
related to a realistic programming system. Now we have an internal theory of probabilistic which is closely

Advantages
What do we mean by random?

If \( x \) is a random value then we want different occurrences of \( x() \) to possibly result in different values.

But this is impossible if \( x : T \) Rand = \( Unit \rightarrow T \).

So if \( x() = 13 \) somewhere, then all occurrences of \( x() \) reduce to 13.

Defining \( T \) Rand was OK in ML because ML has side effects (ref’s). NuPRL doesn’t.

Yet another conflict between imperative and functional programming.
Workarounds

- Add explicit state: $T \text{ Rand} = \text{RandState} \rightarrow T \times \text{RandState}$.

  This complicates the correspondence with the programming model and forces us to remember to use the new state. Also, now we have to define $\text{RandState}$.

- Use co-induction: $T \text{ Rand} = \nu x : \mathbb{U}.T \times x, \ x.\text{choose} = x.\text{out}$

  Random variables are streams of information generated by some unknown process. Co-inductive types have already been studied in the context of NuPRL [Mendler’s thesis] so we can be sure that adding random value types in this way is safe.
random, but might as well be for all we know. Co-inductive types seem to naturally describe source of input.

Example: Stream of user interface events

Example: Stream generated by a “hidden” process — may not be “truly” random generated by a pseudorandom number generator.

Example: Unknown input stream you want to compress.

Example: Inductive types, then iso facto maybe streams are random processes as streams (characterized in 

Random values as streams
Aside: Kolmogorov complexity

Kolmogorov’s complexity theory offers a more computational view of probability and randomness.

\[ K_M(x) = \min \{ |y| \mid y \in \Sigma^*, M(y) = x \} \ (x \in \Sigma^*, \ M \ a \ universal \ machine) \]

\( x \) is incompressible (random) if \( K(x) \approx |x| \).

Infinite incompressible sequences \( \approx \) intuitionistic free-choice sequences?

Interestingly, Martin-Löf’s early work in this area.
constructive logic, but we should start with the standard theory.

Some of these theories might be a "good fit" to uncertainty in

Possibility/Necessity theory: reminiscent of modal logic

Fuzzy logic: Real-valued truth values, value of compound for-

Clipped middle

Dempster-Shafer belief: like standard theory but without "ex-

Aside: Other theories of uncertainty
Interfaces for processes and models

Independent: \( I(T) = \{gen : T \ \text{Rand}\}, IM(T) = \{prob : T \ \text{Prob}\} \)

Markov: \( M(T) = \{start : T; gen : T \rightarrow T \ \text{Rand}\}, MM(T) = \{start : T, prob : t \rightarrow T \ \text{Prob}\} \)

List: \( L(T) = \{gen : T \ \text{List} \rightarrow T \ \text{Rand}\}, LM(T) = \{prob : T \ \text{List} \rightarrow T \ \text{Prob}\} \)

General: \( G(T, S) = \{start : S; next : S \times T \rightarrow S; gen : S \rightarrow T \ \text{Rand}\}, M(T, S) = \{...; prob : S \rightarrow T \ \text{Prob}\} \)

Starts to look like a state machine...
I'm sure many people have suggestions. Cessess and probabilistic models.

What is the "right" constructive understanding of random pro-

Reasoning about explicitly random processes isn't.

Although requires further development of real analyzes.

event spaces in NUPRL is straightforward

Reasoning about (classical) probability for discrete (decidable)

Summary