Probability, programming, and type theory

Overview and Discussion

James Cheney

April 17, 2000

My background

I've been working on *erm compressio*

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

Short answer: use

Long answer: Adapt and extend

standard compression algorithms to terms

e.g. HuCman coding, Lempel-Ziv parsing

Story so far

Recent work based on *statistical models* of terms

Models interesting enough for good compression are too complex to implement and reason about by hand

Example: Compressing giant datatyJflH00(ks)]TJfl200TDfl(working only on the implementation

But ultimately aiming for automatic data verification

Why is this interesting?

Randomness is a core concept in computer science

Probability is the theory used to reason about it

Algorithms: hashing, quicksort, cryptography

AI: Bayes nets, probabilistic NL parsing, fuzzy logic

Analysis: average cases, compression/error-correcting coding algorithms

Why is this not trivial?

Has not been studied from the

T eorems we'd like to prove

$\forall p > 0$ 0 1 List:P $\overline{p}$

: B) ( ) (( ))
( ) + 1

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

Examples: What type does ? ?

Computational approac

Consider ML ADTs for `'a rand` and `'a prob`

Rand- objects and probability distributions are *constructed* from simple objects and fr- outside sources of data

Claim: can build useful, complex randm objects and distributions up frm 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 cotraints)

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
```

robability 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 and logic:

External (probabilistic logics [Nilsson])

Assign probabilities (or ranges) to formulas.

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 uncerta

Make *judgements* uncertain:

roblems wit t ese approac es

Pro ab ilizing judgements or types seems like overkill

Pro ab ility 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 pro ab ility theory to the meta-theo

What do we really want?

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

Probabilizing $x = y \in A$ still imports classical logic into NuPRL

Reasoning about probability still "external"; the is not connected with anything in NuPRL

We really want an *internal* system for *constructive* probabilistic reasoning

Internal approac

Let's take the programming approach from earlier as a model. Define types

T Prob and T Rand in NuPRL, with similar basic constructs

Let

P_i be suitable probability axioms, such as those of Kol-mogo

T f

Advantages

Now we have an *internal* theory of probability which is closely related to a realistic programming system.

For

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

Workarounds

Add explicit state: T Rand $= RandState$.

Tedious programming

ate. Also, we have to define

$RandState$

Random values as streams

If random processes are *streams* (characterized in TT as co-inductive types), then *ipso facto* maybe streams are random processes.

Example: Unknown

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 $K(x) \geq |x|$. Infinite incompressible sequ

intuition for free-choice sequences?

Interestingly, Martin-Löf's early work in this area.

Aside: Other theories of uncertainty

Dempster-Shafer belief: like standard theory but without "excluded middle"

Fuzzy logic: Real-valued truth values, value of compound formula is pure of values of components (truth-functional)

Possibility/necesity theory: reminiscent of modal logic

Some of these theories might be a fit" to uncertainty in constructive logic, but we should start with the standard theory.

Interfaces for processes and models

Independent: $I(T) = fgen$:

Prob

List:) = : List ( ) = List
Prob

General: ( ) = : : ; :
Rand (; : Prob

Summary

Reasoning about (classical) probability for discrete (decidable) event spaces in NuPRL is straightforward

Although requires further development of real analysis

Reasoning about explicitly random processes isn't.

What is the "right" constructive understanding of random processes and probabilistic models?

I'm sure many people have suggestions.