Uncertain<\text{T}>
A First-Order Type for Uncertain Data

James Bornholt
Supervisor: Steve Blackburn
Uncertain<T>: A First-Order Type for Uncertain Data

James Bornholt
Australian National University
u4842199@anu.edu.au

Todd Mytkowicz
Microsoft Research
toddm@microsoft.com

Kathryn S. McKinley
Microsoft Research
mckinley@microsoft.com
Sensors

Approximate computing

Big data

Machine learning

uncertain data
struct Geocoordinate {
    double Latitude;
    double Longitude;
}

Geocoordinate Loc = GetGPSLocation();
uncertain data + discrete type = ???
uncertain data + discrete type = who cares?
uncertain data + discrete type = uncertainty bug
uncertain data + discrete type = uncertainty bug
errors that occur when applications pretend that uncertain data is certain
treating estimates as facts

```c
struct Geocoordinate {
    double Latitude;
    double Longitude;
    double HorizontalAccuracy;
}
```

95% of apps ignore accuracy!
computation compounds error
computation compounds error
computation compounds error

Walking speed (km/h)

Time

Usain Bolt
false positives in questions

\[
\text{if } (\text{Speed} > 60) \\
\text{IssueSpeedingTicket();}
\]
uncertainty bugs

Treating estimates as facts
Computation compounds error
False positives in questions

Caused by poor programming language abstractions

Uncertainty should not be abstracted away
related work

- Uncertain data
  - Sensors, measurements, probabilistic models

Flexible

Simple

Developer computations
related work

Flexible - Simple

Developer computations

No abstraction

Current abstractions

Uncertain data
Sensors, measurements, probabilistic models
related work

Flexiable → Simple

Developer computations

No abstraction

Probabilistic programming

Current abstractions

Uncertain data
Sensors, measurements, probabilistic models
probabilistic programming

Reasoning about probabilistic models

\[
\begin{align*}
\text{earthquake} &= \text{Bernoulli}(0.0001) \\
\text{burglary} &= \text{Bernoulli}(0.001) \\
\text{alarm} &= \text{earthquake or burglary}
\end{align*}
\]

\[
\begin{align*}
\text{if (earthquake)} \\
\quad \text{phoneWorking} &= \text{Bernoulli}(0.7) \\
\text{else} \\
\quad \text{phoneWorking} &= \text{Bernoulli}(0.99)
\end{align*}
\]
What is $\Pr[\text{phoneWorking}=v \mid \text{alarm}=\text{True}]$, for each possible value of $v$ (i.e. True and False)?
inference is expensive

Some paths of execution are very unlikely

![Chart showing the relationship between number of samples and time to query for different probabilities of an earthquake.]
related work

Flexible → Simple

Developer computations

No abstraction

Probabilistic programming

Current abstractions

Probabilistic data
Sensors, measurements, probabilistic models
related work

Developer computations

No abstraction
Probabilistic programming
**Uncertain<T>**
Current abstractions

Probabilistic data
Sensors, measurements, probabilistic models
Uncertain\(<T>\) is an uncertain type abstraction.

Encapsulates distributions, like prior work.

But focuses on an accessible interface.

For everyday programmers, Uncertain\(<T>\) enables programs that are more concise, expressive, and correct.
using Uncertain<T>

Identify the distribution

Compute with the distribution

Ask questions using conditionals

Improve the quality of estimates
identifying the distribution

Many library programmers already know the distribution they need to return!
identifying the distribution

Many library programmers already know the distribution they need to return!

“Get the estimated accuracy of this location, in meters. We define accuracy as the radius of 68% confidence. [...] In statistical terms, it is assumed that location errors are random with a normal distribution.”

—Android
representing distributions

\[
\text{Norm}(x; \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma}} \exp\left\{ -\frac{(x - \mu)^2}{2\sigma^2} \right\}
\]
representing distributions

Store probability density functions?

\[ \text{Norm}(x; \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} \exp \left\{ -\frac{(x - \mu)^2}{2\sigma^2} \right\} \]

Two problems:

1. Even simple operations are complex:

\[ f_{X+Y}(z) = \int_{-\infty}^{\infty} f_Y(z - x) f_X(x) \, dx \]

2. Many interesting distributions don’t have PDFs
representing distributions

Store probability density functions?

\[
\text{Norm}(x; \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma}} \exp \left\{ -\frac{(x - \mu)^2}{2\sigma^2} \right\}
\]

Two problems:

1. Even simple operations are complex:

\[
f_{X+Y}(z) = \int_{-\infty}^{\infty} f_Y(z - x) f_X(x) \, dx
\]

2. Many interesting distributions don’t have PDFs
representing distributions

Random sampling: two birds with one stone

Simple operations are simple (e.g., +)

More distributions can be represented

Later: how to implement random sampling
computing with distributions

Propagating uncertainty through calculations automatically with operator overloading

A key advantage of random sampling: computation is simply* lifting of the original operators
computing with distributions

Propagating uncertainty through calculations automatically with operator overloading

A key advantage of random sampling: computation is simply* lifting of the original operators

If \( x \) a sample of \( X \) and \( y \) a sample of \( Y \) then \( x+y \) a sample of \( X+Y \)
computing with distributions

* The caveat is that this only works if the operands are independent

If not, we need to know something about how the variables are related

This is an issue for all probabilistic programming
induced dependencies

\[ A = X + Y \quad \text{(X,Y independent)} \]
\[ B = A + X \]
induced dependencies

We can distinguish inherent dependencies from programmer-induced dependencies

\[
A = X + Y \quad (X, Y \text{ independent})
\]

\[
B = A + X
\]

When evaluating B, both operands depend on X, so they are not independent

Lazy evaluation to the rescue!
induced dependencies

We can distinguish inherent dependencies from programmer-induced dependencies

\[ A = X + Y \quad (X,Y \text{ independent}) \]
\[ B = A + X \]

When evaluating \( B \), both operands depend on \( X \), so they are not independent

Lazy evaluation to the rescue!
asking questions

```java
if (Speed > 60)
    IssueSpeedingTicket();
```
comparing means

\[
\text{if } \text{Speed.E}() > 60 \text{ }
\]

\[
\text{IssueSpeedingTicket();}
\]
comparing evidence

if ((Speed > 60).E() > 0.95)
   IssueSpeedingTicket();
comparing evidence

> is a lifted operator

\[
\text{if } ((\text{Speed} > 60).E() > 0.95) \\
\text{IssueSpeedingTicket();}
\]
comparing evidence

```csharp
if ((Speed > 60).E() > 0.95) IssueSpeedingTicket();
```

![Graph showing the probability distribution of speed with a threshold for speeding.](image-url)
comparing evidence

mean of Uncertain<bool>

if ((Speed > 60).E() > 0.95)
IssueSpeedingTicket();
comparing evidence

= number in [0,1]

if ((Speed > 60).E() > 0.95)
IssueSpeedingTicket();
comparing evidence

\[
\text{if } ((\text{Speed} > 60).E() > 0.95) \text{ then } \text{IssueSpeedingTicket();}
\]
is there a >95% chance that Speed > 60?

$$\text{if } ((\text{Speed} > 60).E() > 0.95) \text{ \underline{IssueSpeedingTicket()};}$$
The threshold allows the programmer to balance false positives and false negatives. Higher thresholds give fewer false positives, but more false negatives.

```c
if ((Speed > 60).E() > 0.95)
    IssueSpeedingTicket();
```
improving estimates

Uncertain\texttt{T} is Bayesian: error distributions track degrees of belief about the value of a variable

\[
\Pr[H|E] = \frac{\Pr[E|H] \Pr[H]}{\Pr[E]}
\]

Bayes’ theorem: use prior knowledge to improve estimates
improving estimates

\[ \Pr[H|E] = \frac{\Pr[E|H] \Pr[H]}{\Pr[E]} \]
improving estimates

$$\Pr[H|E] = \frac{\Pr[E|H] \Pr[H]}{\Pr[E]}$$

[Graph showing likelihood distribution]
improving estimates

$$Pr[H|E] = \frac{Pr[E|H] Pr[H]}{Pr[E]}$$

likelihood  prior
improving estimates

$$\Pr[H|E] = \frac{\Pr[E|H] \Pr[H]}{\Pr[E]}$$

<table>
<thead>
<tr>
<th>Location</th>
<th>Density</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>0.05</td>
<td>0.00</td>
</tr>
<tr>
<td>0.10</td>
<td>0.00</td>
</tr>
<tr>
<td>0.15</td>
<td>0.00</td>
</tr>
<tr>
<td>0.20</td>
<td>0.00</td>
</tr>
<tr>
<td>0.25</td>
<td>0.00</td>
</tr>
</tbody>
</table>

posterior → likelihood → prior
improving estimates

\[
\Pr[H|E] = \frac{\Pr[E|H] \Pr[H]}{\Pr[E]}
\]

posterior
likelihood
prior

<table>
<thead>
<tr>
<th>Location</th>
<th>Density</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>0.05</td>
<td>0.05</td>
</tr>
<tr>
<td>0.10</td>
<td>0.10</td>
</tr>
<tr>
<td>0.15</td>
<td>0.15</td>
</tr>
<tr>
<td>0.20</td>
<td>0.20</td>
</tr>
<tr>
<td>0.25</td>
<td>0.25</td>
</tr>
</tbody>
</table>

E[Location]
implementing Uncertain<T>

Two key insights in the design inform an efficient implementation

1. Distributions are random samples
   Suggests lazy evaluation

2. All evaluations end up in expected values
   Suggests hypothesis tests
lazy evaluation

Uncertain<T> uses random sampling, but how?

Option 1: store a vector of N samples

|   | 5.6 | 2.8 | 6.4 | 4.9 | 4.9 | 5.1 | 4.3 | 5.0 | ... | 4.6 |
**lazy evaluation**

Uncertain<T> uses random sampling, but how?

Option 1: store a vector of N samples

<table>
<thead>
<tr>
<th></th>
<th>5.6</th>
<th>2.8</th>
<th>6.4</th>
<th>4.9</th>
<th>4.9</th>
<th>5.1</th>
<th>4.3</th>
<th>5.0</th>
<th>…</th>
<th>4.6</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>4.0</td>
<td>3.2</td>
<td>1.1</td>
<td>3.5</td>
<td>3.9</td>
<td>3.4</td>
<td>4.7</td>
<td>3.8</td>
<td>…</td>
<td>2.2</td>
</tr>
</tbody>
</table>
lazy evaluation

Uncertain<T> uses random sampling, but how?

Option 1: store a vector of N samples

\[
\begin{array}{cccccccccc}
A & 5.6 & 2.8 & 6.4 & 4.9 & 4.9 & 5.1 & 4.3 & 5.0 & \ldots & 4.6 \\
+ & 4.0 & 3.2 & 1.1 & 3.5 & 3.9 & 3.4 & 4.7 & 3.8 & \ldots & 2.2 \\
\downarrow & & & & & & & & & & \\
A+B & & & & & & & & & & \\
\end{array}
\]
lazy evaluation

Uncertain<T> uses random sampling, but how?

Option 1: store a vector of N samples

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th></th>
<th>B</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5.6</td>
<td>2.8</td>
<td>6.4</td>
<td>4.9</td>
</tr>
<tr>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td></td>
<td>4.0</td>
<td>3.2</td>
<td>1.1</td>
<td>3.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A + B = 

| A+B |     |     |     |     |     |     |     |     |     |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|
|     |     |     |     |     |     |     |     |     |
|     |     |     |     |     |     |     |     |     |
|     |     |     |     |     |     |     |     |     |
|     |     |     |     |     |     |     |     |     |
|     |     |     |     |     |     |     |     |     |
|     |     |     |     |     |     |     |     |     |
|     |     |     |     |     |     |     |     |     |
|     |     |     |     |     |     |     |     |     |
|     |     |     |     |     |     |     |     |     |
|     |     |     |     |     |     |     |     |     |

...
lazy evaluation

Uncertain\(<T>\) uses random sampling, but how?

Option 1: store a vector of N samples

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>5.6</td>
<td>2.8</td>
<td>6.4</td>
<td>4.9</td>
<td>4.9</td>
<td>5.1</td>
<td>4.3</td>
<td>5.0</td>
<td>...</td>
</tr>
<tr>
<td>+</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>4.0</td>
<td>3.2</td>
<td>1.1</td>
<td>3.5</td>
<td>3.9</td>
<td>3.4</td>
<td>4.7</td>
<td>3.8</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A+B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
lazy evaluation

Uncertain<T> uses random sampling, but how?

Option 1: store a vector of N samples

\[
\begin{array}{cccccccccc}
\text{A} & 5.6 & 2.8 & 6.4 & 4.9 & 4.9 & 5.1 & 4.3 & 5.0 & \ldots & 4.6 \\
+ & + & + & + & + & + & + & + & + & + \\
\text{B} & 4.0 & 3.2 & 1.1 & 3.5 & 3.9 & 3.4 & 4.7 & 3.8 & \ldots & 2.2 \\
\downarrow & & & & & & & & & & \\
\text{A+B} & 9.6 & & & & & & & & \ldots \\
\end{array}
\]
lazy evaluation

Uncertain<T> uses random sampling, but how?

Option 1: store a vector of N samples

\[
\begin{array}{cccccccccc}
A & 5.6 & 2.8 & 6.4 & 4.9 & 4.9 & 5.1 & 4.3 & 5.0 & \ldots & 4.6 \\
+ & + & + & + & + & + & + & + & + & + & + \\
B & 4.0 & 3.2 & 1.1 & 3.5 & 3.9 & 3.4 & 4.7 & 3.8 & \ldots & 2.2 \\
\downarrow & & & & & & & & & & \\
A+B & 9.6 & & & & & & & & \ldots & \\
\end{array}
\]
lazy evaluation

Uncertain<T> uses random sampling, but how?

Option 1: store a vector of N samples

<table>
<thead>
<tr>
<th>A</th>
<th>5.6</th>
<th>2.8</th>
<th>6.4</th>
<th>4.9</th>
<th>4.9</th>
<th>5.1</th>
<th>4.3</th>
<th>5.0</th>
<th>…</th>
<th>4.6</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>4.0</td>
<td>3.2</td>
<td>1.1</td>
<td>3.5</td>
<td>3.9</td>
<td>3.4</td>
<td>4.7</td>
<td>3.8</td>
<td>…</td>
<td>2.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>A+B</td>
<td>9.6</td>
<td>6.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
lazy evaluation

Uncertain\(<T>\) uses random sampling, but how?

Option 1: store a vector of \(N\) samples

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>+</th>
<th>B</th>
<th>A+B</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>5.6</td>
<td>+</td>
<td>4.0</td>
<td>9.6</td>
</tr>
<tr>
<td></td>
<td>2.8</td>
<td>+</td>
<td>3.2</td>
<td>6.0</td>
</tr>
<tr>
<td></td>
<td>6.4</td>
<td>+</td>
<td>1.1</td>
<td>7.5</td>
</tr>
<tr>
<td></td>
<td>4.9</td>
<td>+</td>
<td>3.5</td>
<td>8.4</td>
</tr>
<tr>
<td></td>
<td>4.9</td>
<td>+</td>
<td>3.9</td>
<td>8.8</td>
</tr>
<tr>
<td></td>
<td>5.1</td>
<td>+</td>
<td>3.4</td>
<td>8.5</td>
</tr>
<tr>
<td></td>
<td>4.3</td>
<td>+</td>
<td>4.7</td>
<td>9.0</td>
</tr>
<tr>
<td></td>
<td>5.0</td>
<td>+</td>
<td>3.8</td>
<td>8.8</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>+</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>4.6</td>
<td>+</td>
<td>2.2</td>
<td>6.8</td>
</tr>
</tbody>
</table>
lazy evaluation

Suppose an oracle tells us the “right” sample size for a particular operation (we’ll invent this oracle shortly!) How do we satisfy this sample size?

Uncertain<T> represents distributions with sampling functions, returning a new sample on each invocation

Operators combining distributions are lazy, constructing a symbolic expression tree
evaluating expression trees
evaluating expression trees

```javascript
var A = GetReading()
var B = GetReading()
var Sum = A + B
if ((Sum > 10).E() > 75%):
    Alert()
```
evaluating expression trees

```javascript
var A = GetReading();
var B = GetReading();
var Sum = A + B
if ((Sum > 10).E() > 75%) {
  Alert();
}
```
evaluating expression trees

```javascript
var A = GetReading()
var B = GetReading()
var Sum = A + B
if ((Sum > 10).E() > 75%):
    Alert()
```
hypothesis tests

How do we decide the “right” sample size for a particular operation?

Distributions only evaluated at conditionals, so use hypothesis tests to address sampling error
This code implicitly performs a hypothesis test

```java
if (Speed.E() > 60)
    IssueSpeedingTicket();
```

Start with a base sample size

Continue increasing the sample size until either

1. The null hypothesis is rejected; or
2. A maximum sample size limit is reached (to ensure termination)
smartphone GPS sensors

Many smartphone apps use GPS to calculate distances and speeds

How can Uncertain<T> improve these apps?
int dt = 1;

Geocoordinate LastLocation = GPSLib.GetGPSLocation();
while (true) {
    Sleep(dt); // wait for dt seconds

    Geocoordinate Location = GPSLib.GetGPSLocation();
    double Speed =
        GPSLib.Distance(Location, LastLocation) / dt;

    Display(Speed);
    if (Speed > 5)
        GoodJobMessage();

    LastLocation = Location;
}
int dt = 1;

Geocoordinate LastLocation = GPSLib.GetGPSLocation();
while (true) {
    Sleep(dt);  // wait for dt seconds

    Geocoordinate Location = GPSLib.GetGPSLocation();
    double Speed =
        GPSLib.Distance(Location, LastLocation) / dt;

    Display(Speed);
    if (Speed > 5)
        GoodJobMessage();

    LastLocation = Location;
}


int dt = 1;

Uncertain<Geocoordinate> LastLocation = GPSLib.GetGPSLocation();
while (true) {
    Sleep(dt); // wait for dt seconds

    Uncertain<Geocoordinate> Location = GPSLib.GetGPSLocation();
    Uncertain<double> Speed = GPSLib.Distance(Location, LastLocation) / dt;

    Display(Speed);
    if (Speed > 5)
        GoodJobMessage();

    LastLocation = Location;
}
```csharp
int dt = 1;

Uncertain<Geocoordinate> LastLocation = GPSLib.GetGPSLocation();
while (true) {
  Sleep(dt); // wait for dt seconds

  Uncertain<Geocoordinate> Location = GPSLib.GetGPSLocation();
  Uncertain<double> Speed = GPSLib.Distance(Location, LastLocation) / dt;

  Display(Speed);
  if (Speed > 5)
    GoodJobMessage();

  LastLocation = Location;
}
```
int dt = 1;

Uncertain<Geocoordinate> LastLocation =
    GPSLib.GetGPSLocation();
while (true) {
    Sleep(dt); // wait for dt seconds

    Uncertain<Geocoordinate> Location =
        GPSLib.GetGPSLocation();
    Uncertain<double> Speed =
        GPSLib.Distance(Location, LastLocation) / dt;

    Display(Speed.E().Project());
    if (Speed > 5)
        GoodJobMessage();

    LastLocation = Location;
}
int dt = 1;

Uncertain<Geocoordinate> LastLocation =
    GPSLib.GetGPSLocation();
while (true) {
    Sleep(dt); // wait for dt seconds

    Uncertain<Geocoordinate> Location =
        GPSLib.GetGPSLocation();
    Uncertain<double> Speed =
        GPSLib.Distance(Location, LastLocation) / dt;

    Display(Speed.E().Project());
    ★if (Speed > 5)
        GoodJobMessage();

    LastLocation = Location;
}
int dt = 1;

Uncertain<Geocoordinate> LastLocation = GPSLib.GetGPSLocation();
while (true) {
    Sleep(dt); // wait for dt seconds

    Uncertain<Geocoordinate> Location = GPSLib.GetGPSLocation();
    Uncertain<double> Speed =
        GPSLib.Distance(Location, LastLocation) / dt;

    Display(Speed.E().Project());

    ★ if ((Speed > 5).E() > 0.75)
        GoodJobMessage();

    LastLocation = Location;
}
walking speeds
improved walking speeds
approximate computing

Recent work uses neural networks to approximate functions, trade accuracy for performance.

How to reason about the error this induces?

Neural networks: posterior predictive distribution
Approximate the Sobel operator \( s(\rho) \), calculating gradient of image intensity at a pixel

Evaluate the conditional \( s(\rho) > 0.1 \), with and without Uncertain\(<T>\)
evaluation

Parrot (naive approach)

Mean

90%

Incorrect decisions (%) vs. Confidence level (%)
future work

Sensor applications
  Less accurate sensors to save power

A programming model for uncertainty
  Machine learning for non-experts

Optimisation
  Lazy evaluation a promising target
Uncertainty is a growing problem for non-expert programmers. Existing abstractions are inadequate. Other solutions are either inefficient or inaccessible. Uncertain<T> focuses on accessibility to non-experts, while still being expressive and efficient.

Programmers can make principled decisions under uncertainty.

With Uncertain<T>, non-expert programmers can build programs that are more concise, expressive, and correct.