Probabilistic programming using first-class stores and first-class continuations

Oleg Kiselyov
FNMO C
oleg@pobox.com

Chung-chieh Shan
Rutgers University
ccshan@rutgers.edu

ML workshop
September 26, 2010
I have exactly two kids. What is the probability that
At least one is a girl. my older kid is a girl?
Probabilistic inference

Model (what) 

Pr(Reality) 

Reality → Obs, Result 

obs 

Inference (how) 

\{ Pr(Result | Obs = obs) \}

I have exactly two kids. What is the probability that At least one is a girl. my older kid is a girl?
Declarative probabilistic inference

Model (what)                      Inference (how)

Pr(Reality)
Reality → Obs, Result            Pr(Result | Obs = obs)
obs

I have exactly two kids.          What is the probability that
At least one is a girl.           my older kid is a girl?

```
normalize (exact_reify (fun () ->
    let flip = fun p ->
        true 2 = 3
        false 1 = 3
        true 1 = 2
        false 1 = 4
    in dist [(p, true); (1.-.p, false)]
    in let girl1 = flip 0.5 in
        let girl2 = flip 0.5 in
        if girl1 || girl2
            then girl1
            else fail ()
))
```
Declarative probabilistic inference  
(UAI 2009, DSL 2009)

Model (what) ............................................. Inference (how)

\[ \Pr(\text{Reality}) \]
\[ \text{Reality} \rightarrow \text{Obs, Result} \]
\[ \text{obs} \]

\[ \Pr(\text{Result} \mid \text{Obs} = \text{obs}) \]

I have exactly two kids. What is the probability that
At least one is a girl. my older kid is a girl?

Models and inference as interacting programs in the same general-purpose language
Declarative probabilistic inference

Model (what)  Inference (how)

Pr(Reality)
Reality → Obs, Result \{ Pr(Result | Obs = obs) \}
obs

I have exactly two kids. What is the probability that my older kid is a girl?
At least one is a girl.

let flip = fun p ->
dist [(p, true);
    (1.-.p, false)]
in let girl1 = flip 0.5 in
let girl2 = flip 0.5 in
if girl1 || girl2
then girl1 else fail ()

Models and inference as interacting programs in the same general-purpose language
Declarative probabilistic inference

Model (what)  Inference (how)

\[
\begin{align*}
\Pr(\text{Reality}) \\
\text{Reality} \rightarrow \text{Obs, Result} \\
\text{obs}
\end{align*}
\]

\[
\begin{align*}
\Pr(\text{Result} \mid \text{Obs} = \text{obs})
\end{align*}
\]

I have exactly two kids. What is the probability that my older kid is a girl?

What is the probability that at least one is a girl?

\[
\text{normalize (exact_reify (fun () ->}
\begin{array}{|c|c|}
\hline
\text{true} & 1/2 \\
\text{false} & 1/4 \\
\hline
\end{array}
\text{Models and inference as interacting programs in the same general-purpose language}
\]
I have exactly two kids. What is the probability that at least one is a girl and my older kid is a girl?

```
let flip = fun p ->
  dist [(p, true);
        (1.-.p, false)]
in let girl1 = flip 0.5 in
  let girl2 = flip 0.5 in
  if girl1 || girl2
  then girl1 else fail ()
```

Models and inference as interacting programs in the same general-purpose language
Declarative probabilistic inference  
(UAI 2009, DSL 2009)

Model (what)  
Pr(Reality)  
Reality → Obs, Result  

Inference (how)  
Pr(Result | Obs = obs)

I have exactly two kids. What is the probability that my older kid is a girl?

What is the probability that at least one is a girl?

```
normalize (exact_reify (fun () ->
  let flip = fun p ->
    dist [(p, true); (1.-.p, false)]
  in let girl1 = flip 0.5 in
    let girl2 = flip 0.5 in
    if girl1 || girl2
    then girl1 else fail ()
))
```

Expressive models and efficient inference as interacting programs in the same general-purpose language.
Outline

▶ Expressive models
  Reuse existing infrastructure
  Nested inference

Efficient inference
  First-class continuations
  First-class stores
Motivic development in Beethoven sonatas (Pfeffer 2007)

Source motif

\begin{music}
\begin{tabular}{c}
\begin{music}
\horizontalaccidental&\horizontalaccidental&\horizontalaccidental&\verticalaccidental&\verticalaccidental&\end{music}
\end{tabular}
\end{music}
Motivic development in Beethoven sonatas (Pfeffer 2007)

Source motif

\[ \text{\includegraphics{source_motif.png}} \]
Motivic development in Beethoven sonatas

Source motif:

\begin{music}
\clef \G
\begin{musicstaffs}
\musicnote D\sharp 
\musicnote D\sharp 
\musicnote D\sharp 
\musicnote E\flat 
\musicnote F\natural 
\musicnote G
\musicnote A
\musicnote G
\musicnote F
\musicnote D\natural
\end{musicstaffs}
\end{music}
Motivic development in Beethoven sonatas

Source motif:

\[\text{Source motif} \quad \begin{array}{c}
\text{\includegraphics[width=\textwidth]{motif.png}}
\end{array}\]
Motivic development in Beethoven sonatas (Pfeffer 2007)

Source motif

\[\text{\uparrow} \quad \text{infer} \quad \text{\downarrow}\]

Destination motif

Pfeffer 2007 (30 sec) 93 100 28 80 98 100 63

HANSEI (90 sec) 98 100 29 87 94 100 77

HANSEI (30 sec) 92 99 25 46 72 95 61

Importance sampling using lazy stochastic lists.
Motivic development in Beethoven sonatas

(Pfeffer 2007)

Source motif

\[ \text{infer} \]

Destination motif

<table>
<thead>
<tr>
<th>Motif pair</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>% correct</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pfeffer 2007 (30 sec)</td>
<td>93</td>
<td>100</td>
<td>28</td>
<td>80</td>
<td>98</td>
<td>100</td>
<td>63</td>
</tr>
<tr>
<td>HANSEI (90 sec)</td>
<td>98</td>
<td>100</td>
<td>29</td>
<td>87</td>
<td>94</td>
<td>100</td>
<td>77</td>
</tr>
<tr>
<td>HANSEI (30 sec)</td>
<td>92</td>
<td>99</td>
<td>25</td>
<td>46</td>
<td>72</td>
<td>95</td>
<td>61</td>
</tr>
</tbody>
</table>

Importance sampling using lazy stochastic lists.
Noisy radar blips for aircraft tracking

Blips present and absent

infer

Number of planes

Probability

0 1 2 3 4 5 6 7

Particle filter using lazy stochastic coordinates.
Noisy radar blips for aircraft tracking  
(Milch et al. 2007)

Blips present and absent  
$t = 1$

Particle filter using lazy stochastic coordinates.
Noisy radar blips for aircraft tracking

(Milch et al. 2007)

Blips present and absent

\( t = 1, \ t = 2 \)

Number of planes

Probability

Particle filter using lazy stochastic coordinates.
Noisy radar blips for aircraft tracking

(Milch et al. 2007)

Blips present and absent
\( t = 1, \ t = 2, \ t = 3 \)

Number of planes

Particle filter using lazy stochastic coordinates.
Models as programs in a general-purpose language

Reuse existing infrastructure!

- Rich libraries: lists, arrays, database access, I/O, ...
- Type system
- Functions as first-class values
- Compiler
- Debugger
- Memoization

Implemented independently in Haskell, Scheme, Ruby, Scala ...
Choose a coin that is either fair or completely biased for \texttt{true}.

\begin{verbatim}
let biased = flip 0.5 in
let coin = fun () -> flip 0.5 || biased in
\end{verbatim}
Choose a coin that is either fair or completely biased for true.

\[
\text{let \textit{biased} = \text{flip} 0.5 \text{ in}\\
\text{let \textit{coin} = fun () \rightarrow \text{flip} 0.5 \text{ || \textit{biased}} \text{ in}\\
\]

Let \( p \) be the probability that flipping the coin yields true.

What is the probability that \( p \) is at least 0.3?
Models that invoke nested inference

Choose a coin that is either fair or completely biased for true.

let biased = flip 0.5 in
let coin = fun () -> flip 0.5 || biased in

Let $p$ be the probability that flipping the coin yields true.

What is the probability that $p$ is at least 0.3?
Answer: 1.

at_least 0.3 true (exact_reify coin)
Models that invoke nested inference

```ocaml
exact_reify (fun () ->

Choose a coin that is either fair or completely biased for true.

let biased = flip 0.5 in
let coin = fun () -> flip 0.5 || biased in

Let $p$ be the probability that flipping the coin yields true.

What is the probability that $p$ is at least 0.3?
Answer: 1.

at_least 0.3 true (exact_reify coin)
```
Choose a coin that is either fair or completely biased for \textit{true}.

\begin{verbatim}
let biased = flip 0.5 in
let coin = fun () -> flip 0.5 || biased in
\end{verbatim}

Let $p$ be the probability that flipping the coin yields \textit{true}.
\textbf{Estimate $p$ by flipping the coin twice.}
What is the probability that our estimate of $p$ is at least 0.3?
\textbf{Answer: 7/8.}

\begin{verbatim}
 at_least 0.3 true (sample 2 coin) )
\end{verbatim}
Models that invoke nested inference

```
exact_reify (fun () ->

    Choose a coin that is either fair or completely biased for true.
    let biased = flip 0.5 in
    let coin = fun () -> flip 0.5 || biased in

    Let \( p \) be the probability that flipping the coin yields true.
    Estimate \( p \) by flipping the coin twice.
    What is the probability that our estimate of \( p \) is at least 0.3?
    Answer: 7/8.

    at_least 0.3 true (sample 2 coin)
    )

Returns a distribution, using dist like models do.
Works with observation, recursion, memoization.
Metareasoning without interpretive overhead.
```
Outline

Expressive models
  Reuse existing infrastructure
  Nested inference

Efficient inference
  First-class continuations
  First-class stores
Reifying a model into a lazy search tree

```
type 'a branch = V of 'a | C of (unit -> 'a tree)
and 'a tree = (prob * 'a branch) list
```

```
.3  .2  .5
 /   /   |
.8   .2   .6
 /   /   |
true false .3
```

not syntax tree
not call tree
Reifying a model into a lazy search tree

Depth-first enumeration = exact inference
Random dive = rejection sampling
Dive with look-ahead = importance sampling
Represent a probability and state monad (Filinski 1994) using first-class delimited continuations, aka clonable threads:

- Model runs inside a thread.
- \texttt{dist} clones the thread.
- \texttt{fail} kills the thread.

Reifying a model into a lazy search tree

reflect ◦ simplify ◦ reify = table, chart, bucket
reflect ◦ sample ◦ reify = particle filter
The library so far

```ocaml
let reify m = reset (fun () -> [(1.0, V (m ())])

let dist ch = shift (fun k ->
  List.map (fun (p,v) -> (p, C (fun () -> k v))) ch)
```
The library so far

type 'a branch = V of 'a | C of (unit -> 'a tree)
and 'a tree = (prob * 'a branch) list

let prompt = new_prompt ()

let reify m = reset prompt (fun () -> [(1.0, V (m ())])

let dist ch = shift prompt (fun k ->
    List.map (fun (p,v) -> (p, C (fun () -> k v))) ch)
First-class continuations

type req = Done | Choice of (prob * (unit -> req)) list

let reify m =
    let answer = ref None in
    let rec interp req = match req with
        | Done ->
            let Some v = !answer in [(1.0, V v)]
        | Choice ch ->
            List.map (fun (p,m) ->
                (p, C (fun () -> interp (m ()))))
            ch
    in interp (reset prompt (fun () ->
        answer := Some (m (); Done))

let dist ch = shift prompt (fun k ->
    Choice (List.map (fun (p,v) -> (p, fun () -> k v)) ch))
type gender = Female | Male

let kid = memo (fun n -> dist [(0.5, Female);
                               (0.5, Male)])
in if kid 1 = Female || kid 2 = Female
  then kid 1 else fail ()
Memoization

type gender = Female | Male

let kid = memo (fun n -> dist [(0.5, Female); (0.5, Male)])
in if kid 1 = Female || kid 2 = Female then kid 1 else fail ()

Used to speed up inference (ICFP 2009)

by delaying choices until observed
Memoization

type gender = Female | Male

let kid = memo (fun n -> dist [(0.5, Female); (0.5, Male)])
in if kid 1 = Female || kid 2 = Female
  then kid 1 else fail ()

Used to speed up inference (ICFP 2009)

by delaying choices until observed
Memoization

type gender = Female | Male

let kid = memo (fun n -> dist [(0.5, Female); (0.5, Male)])
in if kid 1 = Female || kid 2 = Female
then kid 1 else fail ()

Used to speed up inference (ICFP 2009)
and to express nonparametric distributions (Goodman et al. 2008)

Lazy evaluation is memo (fun () -> ...)

Each search-tree node must keep its own store (‘thread-local’) Nesting creates regions of memo cells (ICFP 2006)
Memoization

type gender = Female | Male

let kid = memo (fun n -> dist [(0.5, Female); (0.5, Male)])
in if kid 1 = Female || kid 2 = Female
then kid 1 else fail ()

Used to speed up inference (ICFP 2009)
and to express nonparametric distributions (Goodman et al. 2008)

Lazy evaluation is 
Each search-tree node
Nesting creates regions

Delimited Dynamic Binding

Oleg Kiselyov
FNMOC
oleg@pobox.com

Chung-chieh Shan
Rutgers University
ccshan@cs.rutgers.edu

Amr Sabry
Indiana University
sabry@indiana.edu

Abstract
Dynamic binding and delimited control are useful together in many settings, including Web applications, database cursors, and mobile code. We examine this pair of language features to show that the semantics of their interaction is ill-defined yet not expressive enough for the need.
module Memory = struct
  type 'a loc
  type t
  val newm : t
  val new_loc : unit -> 'a loc
  val mref : 'a loc -> t -> 'a (* throws Not_found *)
  val mset : 'a loc -> 'a -> t -> t
end
let reify m =
    let answer = ref None in
    let rec interp req = match req with
    | Done ->
        let Some v = !answer in [(1.0, V v)]
    | Choice ch ->
        List.map (fun (p,m) ->
            (p, C (fun () -> interp (m ()))))
        ch
    in
    let mem = !thread_local in
    thread_local := Memory.newm;
    let req = reset prompt (fun () ->
        answer := Some (m ()) ; Done) in
    thread_local := mem;
    interp req
Recap

Expressive models and efficient inference as interacting programs in the same general-purpose language

We want first-class delimited continuations and (garbage-collector support for) first-class stores

HANSEI  http://okmij.org/ftp/kakuritu/
Recap

Expressive models and efficient inference as interacting programs in the same general-purpose language

We want first-class delimited continuations and (garbage-collector support for) first-class stores

HANSEI  http://okmij.org/ftp/kakuritu/