Self-applicable probabilistic inference without interpretive overhead

Oleg Kiselyov
FNMOC
oleg@pobox.com

Chung-chieh Shan
Rutgers University
ccshan@rutgers.edu

16 April 2009

Patrick Hughes

Patrick Hughes

Probabilistic inference

$$\left.\begin{array}{l}\Pr(W) \\ \Pr(F|W) \\ \text{Observed evidence } F\end{array}\right\} \text{Compute } \Pr(W\,|\,F)\text{, etc.}$$

Declarative probabilistic inference

Model (what) | Inference (how)

$$\left.\begin{array}{l}\Pr(W) \\ \Pr(F|W) \\ \text{Observed evidence } F\end{array}\right\} \text{Compute } \Pr(W \mid F)\text{, etc.}$$

Declarative probabilistic inference

	Model (what)	Inference (how)
Toolkit (BNT)	invoke $\rightarrow$	distributions, conditionalization, ...
Language (BLOG)	random choice, evidence observation, ...	$\leftarrow$ interpret

Declarative probabilistic inference

	Model (what)	Inference (how)
Toolkit (BNT)	use existing facilities: libraries, compilers, types, debugging	add custom procedures: just sidestep or extend
Language (BLOG)	succinct and natural: sampling procedures, relational programs	compile models to more efficient inference code

Declarative probabilistic inference

	Model (what)	Inference (how)
Toolkit (BNT)	use existing facilities: libraries, compilers, types, debugging	add custom procedures: just sidestep or extend
Language (BLOG)	succinct and natural: sampling procedures, relational programs	compile models to more efficient inference code
Today: best of both worlds	invoke →	← interpret
	models *of inference*: theory of mind	deterministic parts of models run *at full speed*

Express both models and inference as programs in the same general-purpose language.

Outline

- **Expressivity (colored balls)**
  - Memoization
- Inference (music)
  - Reifying a model into a search tree
  - Importance sampling with look-ahead
- Self-interpretation (implicature)
  - Variable elimination
  - Particle filtering
  - Theory of mind

Colored balls

An urn contains an unknown number of balls—say, a number chosen from a [uniform] distribution. Balls are equally likely to be blue or green. We draw some balls from the urn, observing the color of each and replacing it. We cannot tell two identically colored balls apart; furthermore, observed colors are wrong with probability 0.2. How many balls are in the urn? Was the same ball drawn twice?

(Milch et al. 2007)

Colored balls

```
type color = Blue | Green
```

```
dist [(0.5, Blue); (0.5, Green)]
```

Colored balls

```
type color = Blue | Green
```

```
let ball_color = memo (function b ->
      dist [(0.5, Blue); (0.5, Green)])
```

Colored balls

```
type color = Blue | Green
```

```
let nballs = 1 + uniform 8 in
let ball_color = memo (function b ->
      dist [(0.5, Blue); (0.5, Green)])
```

Colored balls

```
type color = Blue | Green
```

```
let nballs = 1 + uniform 8 in
let ball_color = memo (function b ->
      dist [(0.5, Blue); (0.5, Green)]) in
let observe = function o ->
    if o <> observed_color (ball_color (uniform nballs))
    then fail ()
```

Colored balls

```
type color = Blue | Green
```

```
let nballs = 1 + uniform 8 in
let ball_color = memo (function b ->
      dist [(0.5, Blue); (0.5, Green)]) in
let observe = function o ->
    if o <> observed_color (ball_color (uniform nballs))
    then fail ()
```

Colored balls

```
type color = Blue | Green
let opposite_color = function Blue -> Green
                            | Green -> Blue
let observed_color = function c ->
    dist [(0.8, c); (0.2, opposite_color c)]
```

```
let nballs = 1 + uniform 8 in
let ball_color = memo (function b ->
      dist [(0.5, Blue); (0.5, Green)]) in
let observe = function o ->
    if o <> observed_color (ball_color (uniform nballs))
    then fail ()
```

Colored balls

```
type color = Blue | Green
let opposite_color = function Blue -> Green
                            | Green -> Blue
let observed_color = function c ->
    dist [(0.8, c); (0.2, opposite_color c)]
```

```
let nballs = 1 + uniform 8 in
let ball_color = memo (function b ->
      dist [(0.5, Blue); (0.5, Green)]) in
let observe = function o ->
    if o <> observed_color (ball_color (uniform nballs))
    then fail ()
```

Colored balls

```
type color = Blue | Green
let opposite_color = function Blue -> Green
                            | Green -> Blue
let observed_color = function c ->
    dist [(0.8, c); (0.2, opposite_color c)]

let model_nballs = function obs () ->
 let nballs = 1 + uniform 8 in
 let ball_color = memo (function b ->
       dist [(0.5, Blue); (0.5, Green)]) in
 let observe = function o ->
     if o <> observed_color (ball_color (uniform nballs))
     then fail () in
 Array.iter observe obs; nballs
```

Colored balls

```
type color = Blue | Green
let opposite_color = function Blue -> Green
                            | Green -> Blue
let observed_color = function c ->
    dist [(0.8, c); (0.2, opposite_color c)]

let model_nballs = function obs () ->
 let nballs = 1 + uniform 8 in
 let ball_color = memo (function b ->
       dist [(0.5, Blue); (0.5, Green)]) in
 let observe = function o ->
     if o <> observed_color (ball_color (uniform nballs))
     then fail () in
 Array.iter observe obs; nballs

normalize (sample_reify 17 10000 (model_nballs
 [|Blue;Blue;Blue;Blue;Blue;Blue;Blue;Blue;Blue;Blue|]))
```

Colored balls

```
type color = Blue | Green
let opposite_color = function Blue -> Green
                            | Green -> Blue
let observed_color = function c ->
    dist [(0.8, c); (0.2, opposite_color c)]

let model_nballs = function obs () ->
 let nballs = 1 + uniform 8 in
 let ball_color = memo (function b ->
      dist [(0.5, Blue); (0.5, Green)]) in
 let observe = function o ->
     if o <> observed_color (ball_color (uniform nballs))
     then fail () in
 Array.iter observe obs; nballs

normalize (sample_reify 17 10000 (model_nballs
 [|Blue;Blue;Blue;Blue;Blue;Blue;Blue;Blue;Blue;Blue|]))
```

Outline

Expressivity (colored balls)
- Memoization

▶ **Inference (music)**
- Reifying a model into a search tree
- Importance sampling with look-ahead

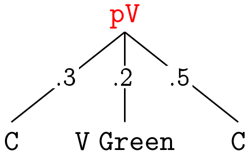
Self-interpretation (implicature)
- Variable elimination
- Particle filtering
- Theory of mind

Reifying a model into a search tree

C

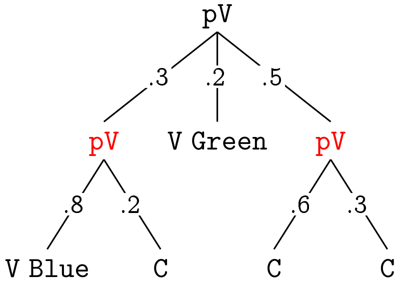
```
type 'a vc = V of 'a | C of (unit -> 'a pV)
and  'a pV = (float * 'a vc) list
```

Reifying a model into a search tree



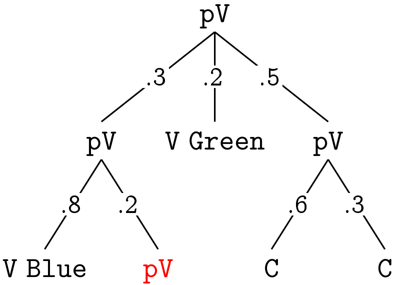

```
type 'a vc = V of 'a | C of (unit -> 'a pV)
and  'a pV = (float * 'a vc) list
```

Reifying a model into a search tree




```
type 'a vc = V of 'a | C of (unit -> 'a pV)
and  'a pV = (float * 'a vc) list
```

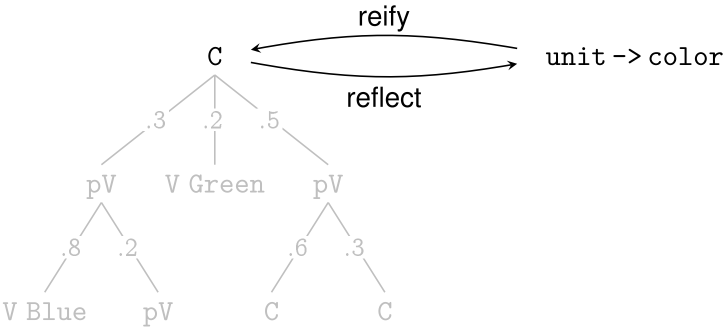
Reifying a model into a search tree




```
type 'a vc = V of 'a | C of (unit -> 'a pV)
and  'a pV = (float * 'a vc) list
```

Depth-first traversal is exact inference by brute-force enumeration.

Reifying a model into a search tree




```
type 'a vc = V of 'a | C of (unit -> 'a pV)
and  'a pV = (float * 'a vc) list
```

Inference procedures cannot access models' source code.

Reifying a model into a search tree

$$\begin{tikzcd} \mathtt{C} \arrow[r, bend right, "\text{reflect}"'] & \mathtt{unit \to color} \arrow[l, bend right, "\text{reify}"'] \end{tikzcd}$$

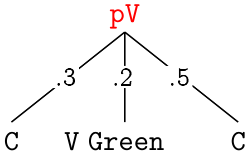
Implemented	
by representing	(Filinski 1994)
a state monad transformer	(Moggi 1990)
applied to a probability monad	(Giry 1982)
using `shift` and `reset`	(Danvy & Filinski 1989)
to operate on first-class	(Felleisen et al. 1987)
delimited continuations	(Strachey & Wadsworth 1974)

- model runs inside `reset` (like an exception handler)
- `dist` and `fail` perform `shift` (like throwing an exception)
- `memo` mutates thread-local storage

Importance sampling with look-ahead

`C` Probability mass $p_c = 1$

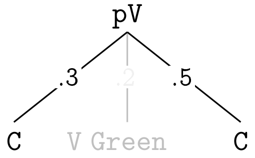
Importance sampling with look-ahead



Probability mass $p_c = 1$

1. Expand one level.

Importance sampling with look-ahead

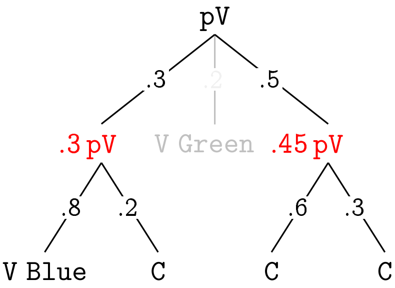


Probability mass $p_c = 1$

$(.2, \texttt{Green})$

1. Expand one level.
2. Report shallow successes.

Importance sampling with look-ahead

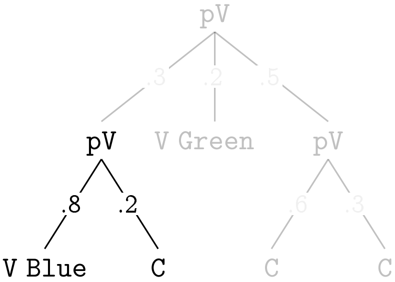


Probability mass $p_c = .75$

$(.2, \texttt{Green})$

1. Expand one level.
2. Report shallow successes.
3. Expand one more level and tally open probability.

Importance sampling with look-ahead

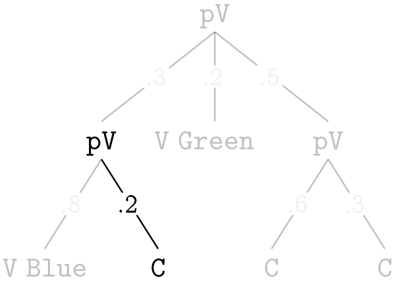


Probability mass $p_c = .75$

$(.2, \texttt{Green})$

1. Expand one level.
2. Report shallow successes.
3. Expand one more level and tally open probability.
4. Randomly choose a branch and go back to 2.

Importance sampling with look-ahead

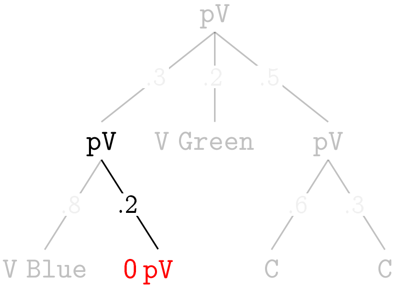


Probability mass $p_c = .75$

$(.2, \texttt{Green})$ $(.6, \texttt{Blue})$

1. Expand one level.
2. Report shallow successes.
3. Expand one more level and tally open probability.
4. Randomly choose a branch and go back to 2.

Importance sampling with look-ahead

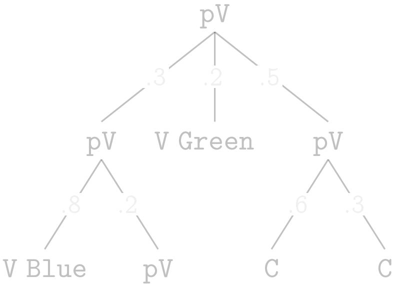


Probability mass $p_c = 0$

$(.2, \texttt{Green})\ (.6, \texttt{Blue})$

1. Expand one level.
2. Report shallow successes.
3. Expand one more level and tally open probability.
4. Randomly choose a branch and go back to 2.

Importance sampling with look-ahead



Probability mass $p_c = 0$

$(.2, \texttt{Green})$ $(.6, \texttt{Blue})$

1. Expand one level.
2. Report shallow successes.
3. Expand one more level and tally open probability.
4. Randomly choose a branch and go back to 2.

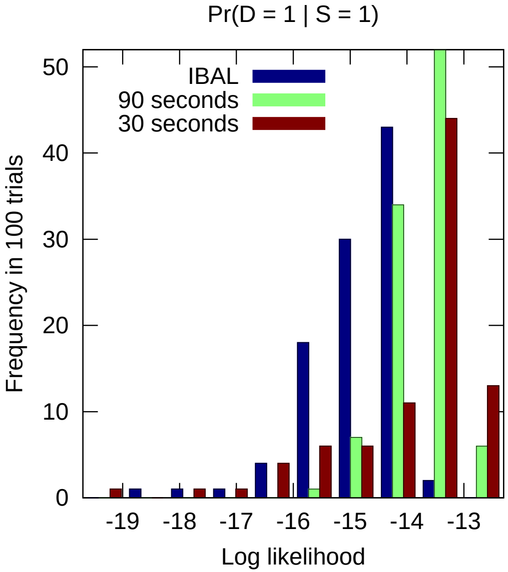
Music model

Pfeffer's test of importance sampling (2007):
motivic development in early Beethoven piano sonatas

$$\begin{tikzcd} \text{Source motif } S \arrow[d, "\text{recursively divide}"'] & \text{Destination motif } D \\ \text{Random binary tree} \arrow[r, "\text{recursively transpose or delete}"'] & \text{Random binary tree} \arrow[u, "\text{recursively concatenate}"'] \end{tikzcd}$$

Want $\Pr(D = \cdots \mid S = \cdots)$.
Exact inference and rejection sampling are infeasible.
Implemented using lists with stochastic parts.

Typical inference results



	100 inference trials		
	$\ln(\text{Mean})$	$\ln(\text{SD})$	$\#0$
IBAL	-14.6	-15.1	0
90 s	-13.6	-14.4	0
30 s	-13.7	-13.8	13

Outline

Expressivity (colored balls)
- Memoization

Inference (music)
- Reifying a model into a search tree
- Importance sampling with look-ahead

► **Self-interpretation (implicature)**
- Variable elimination
- Particle filtering
- Theory of mind

Models of inference

Inference procedures and models

- are written in the same general-purpose language
- use the same stochastic primitive `dist`

Models of inference

Inference procedures and models

- are written in the same general-purpose language
- use the same stochastic primitive `dist`

so inference procedures can be invoked by models

```
inference (function () ->
  ... inference (function () -> ...) ...)
```

and deterministic parts run at full speed.

Program generation with mutable state and control effects.

Models of inference

Inference procedures and models

- are written in the same general-purpose language
- use the same stochastic primitive `dist`

so inference procedures can be invoked by models

```
inference (function () ->
  ... inference (function () -> ...) ...)
```

and deterministic parts run at full speed.

Program generation with mutable state and control effects.

One common usage pattern: reify-infer-reflect

- Brute-force enumeration becomes *variable elimination*
- Sampling becomes *particle filtering*

Theory of mind

Instances abound:

- False-belief (Sally-Anne) task
- Trading securities
- Teacher's hint to student
- Gricean reasoning in language use

Theory of mind

Instances abound:

- False-belief (Sally-Anne) task
- Trading securities
- Teacher's hint to student
- Gricean reasoning in language use
    1. "Some professors are coming to the party."
    2. "All professors are coming to the party."
    3. "Some but not all professors are coming to the party."

  Trade-off between precision and ease of comprehension?

Theory of mind

Instances abound:

- False-belief (Sally-Anne) task
- Trading securities
- Teacher's hint to student
- Gricean reasoning in language use
  1. "Some professors are coming to the party."
  2. "All professors are coming to the party."
  3. "Some but not all professors are coming to the party."

  Trade-off between precision and ease of comprehension?

Crucial for collaboration among human and computer agents!

Want executable models.

A bounded-rational agent's theory of bounded-rational mind
$\sim$ approximate inference about approximate inference

Marr's computational vs algorithmic models

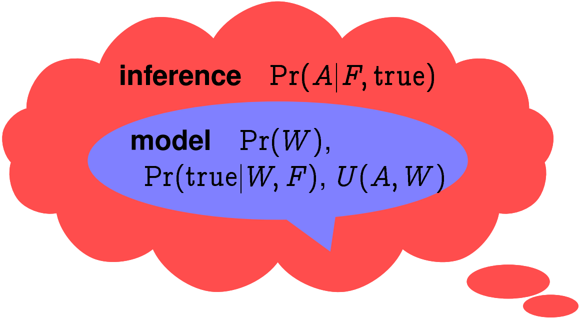
model $\Pr(W)$,
$\Pr(\mathtt{true}|W, F)$, $U(A, W)$

world $W \in \{$0 come, 1 come, 2 come, 3 come$\} \times \cdots$

action $A \in \{$feed 0, feed 1, feed 2, feed 3$\}$

form $F \subseteq \{$some, all, no, not all$\}$

Marr's computational vs algorithmic models

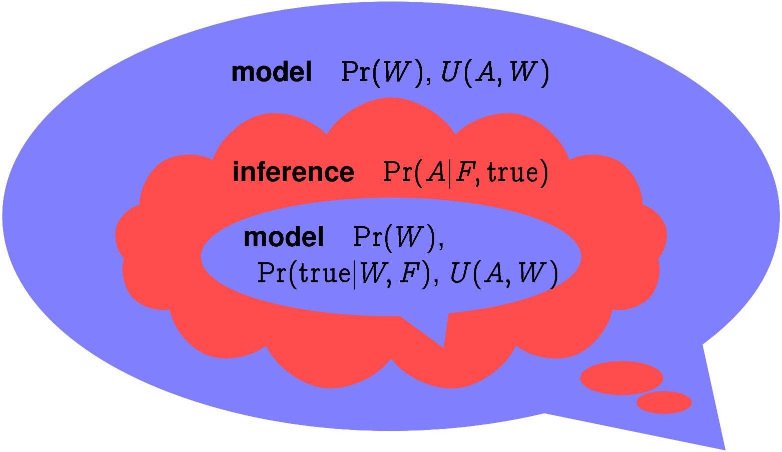


world $W \in$ {0 come, 1 come, 2 come, 3 come} $\times \cdots$

action $A \in$ {feed 0, feed 1, feed 2, feed 3}

form $F \subseteq$ {some, all, no, not all}

Marr's computational vs algorithmic models



world $W \in \{$0 come, 1 come, 2 come, 3 come$\} \times \cdots$

action $A \in \{$feed 0, feed 1, feed 2, feed 3$\}$

form $F \subseteq \{$some, all, no, not all$\}$

Marr's comp

inference $\Pr(F)$

model $\Pr(W), U(A, W)$

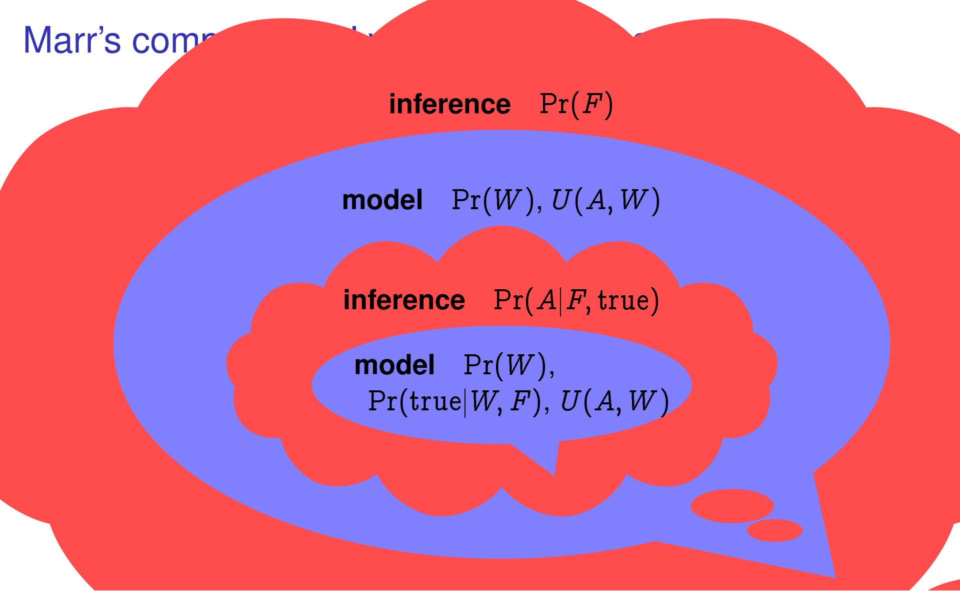
inference $\Pr(A|F, \text{true})$

model $\Pr(W)$,
$\Pr(\text{true}|W, F)$, $U(A, W)$

action

form $F \subseteq$ {, no, not

Marr's comp



A computational model of the modeler nests an algorithmic model of the modelee: invoke inference recursively, without interpretive overhead.

Summary

Express both models and inference as programs in the same general-purpose language.

- Combine strengths of toolkits and standalone languages
- Deterministic parts of models run at full speed
- Models can invoke inference without interpretive overhead
- Theory of mind: inference about approximate inference
- A variety of inference methods: variable elimination, particle filtering, importance sampling, . . . ?