Theory of mind and bounded rationality without interpretive overhead

Chung-chieh Shan (Rutgers ⇔ Aarhus), with Oleg Kiselyov
Theory of mind

- False-belief (Sally-Anne) task
- Gricean reasoning
- $p$-beauty contest
- Focal points in coordination games
- Information cascade
- Securities trading
- Plausibly deniable bribing (Pinker, Nowak, Lee)

Crucial for collaboration among human and computer agents!
Want executable models.
Modeling minds as programs

An agent’s intention

A rational agent’s beliefs

A probability distribution

A utility function

A program

A stochastic program

Bounded rationality

Approximate inference

The amount of detail varies.
Modeling minds as programs

An agent’s intention  A rational agent’s beliefs  desires  Bounded rationality
  |  A probability distribution  A utility function  |
  |  A program  A stochastic program  Approximate inference  |

val random : random
val dist : random -> (prob * 'a) list -> 'a
val fail : random -> 'a

let flip random p = dist random [ p, true; 1.-.p, false ] in
let x = flip random 0.5 in
let y = flip random 0.5 in
if x || y then (x,y) else fail random

The amount of detail varies.
Modeling minds as programs

An agent’s intention

A rational agent’s beliefs
desires

A bounded-rational agent’s

A probability distribution function

A utility function

A program

A stochastic program

Approximate inference

theory of mind
theory of rational mind
theory of bounded-rational mind

about programs
about stochastic programs
about approximate inference

The amount of detail varies. Encapsulated weighted search.
Marr’s computational vs algorithmic models

A computational model of the modeler nests an algorithmic model of the modelee. For arbitrary nesting, implement inference as a stochastic program. Run deterministic code at full speed, to avoid slowdown exponential in the nesting depth (e.g., quantifier depth, plys).

stochastic program
(e.g., grammar)
Marr’s computational vs algorithmic models

approximate inference
(e.g., comprehension)

stochastic program
(e.g., grammar)
Marr’s computational vs algorithmic models

**stochastic program** (e.g., don’t go to jail)

**approximate inference**
(e.g., comprehension)

**stochastic program**
(e.g., grammar)
Marr’s computational vs algorithmic models

**approximate inference** (e.g., plan utterance)

**stochastic program** (e.g., don’t go to jail)

**approximate inference** (e.g., comprehension)

**stochastic program** (e.g., grammar)

Run deterministic code at full speed, to avoid slowdown exponential in the nesting depth (e.g., quantifier depth, plys).
A computational model of the modeler nests an algorithmic model of the modelee.

For arbitrary nesting, implement inference as a stochastic program.

Run deterministic code at full speed, to avoid slowdown exponential in the nesting depth (e.g., quantifier depth, plys).
How to eliminate interpretive overhead

Filinski: given a programming language with *delimited control*, add *layered side effects* (probabilities, memoization, etc.) while still running deterministic code at full speed.

- With delimited control, threads of execution can be suspended, resumed, copied, discarded.
- Represent stochastic programs not as data but as normal programs that suspend when they want randomness.
- Convert a stochastic program to a lazy tree of execution traces without interpretive overhead.
- Inference operates on this lazy tree. Implemented in OCaml.
- Inference is itself a stochastic program (e.g., importance sampling): it suspends when it wants randomness.
- Intuitions for nesting: sandboxes, virtualization, randomness adapters, mock objects.
How to eliminate interpretive overhead

Filinski: given a programming language with delimited control, add layered side effects (probabilities, memoization, etc.) while still running deterministic code at full speed.

- With delimited control, threads of execution can be suspended, resumed, copied, discarded.
- Represent stochastic programs not as data but as normal programs that suspend when they want randomness.
- Convert a stochastic program to a lazy tree of execution traces without interpretive overhead.
- Inference operates on this lazy tree. Implemented in OCaml.
- Inference is itself a stochastic program (e.g., importance sampling): it suspends when it wants randomness.
- Intuitions for nesting: sandboxes, virtualization, randomness adapters, mock objects.
How to eliminate interpretive overhead

Filinski: given a programming language with delimited control, add layered side effects (probabilities, memoization, etc.) while still running deterministic code at full speed.

► With delimited control, threads of execution can be suspended, resumed, copied, discarded.

► Represent stochastic programs not as data but as normal programs that suspend when they want randomness.

► Convert a stochastic program to a lazy tree of execution traces without interpretive overhead.

► Inference operates on this lazy tree. Implemented in OCaml.

► Inference is itself a stochastic program (e.g., importance sampling): it suspends when it wants randomness.

► Intuitions for nesting: sandboxes, virtualization, randomness adapters, mock objects.
Filinski: given a programming language with *delimited control*, add *layered side effects* (probabilities, memoization, etc.) while still running deterministic code at full speed.

- With delimited control, threads of execution can be suspended, resumed, copied, discarded.
- Represent stochastic programs not as data but as normal programs that suspend when they want randomness.
- Convert a stochastic program to a lazy tree of execution traces without interpretive overhead.
- Inference operates on this lazy tree. Implemented in OCaml.
- Inference is itself a stochastic program (e.g., importance sampling): it suspends when it wants randomness.
- Intuitions for nesting: sandboxes, virtualization, randomness adapters, mock objects.
How to eliminate interpretive overhead

Filinski: given a programming language with delimited control, add layered side effects (probabilities, memoization, etc.) while still running deterministic code at full speed.

- With delimited control, threads of execution can be suspended, resumed, copied, discarded.
- Represent stochastic programs not as data but as normal programs that suspend when they want randomness.
- Convert a stochastic program to a lazy tree of execution traces without interpretive overhead.
- Inference operates on this lazy tree. Implemented in OCaml.
- Inference is itself a stochastic program (e.g., importance sampling): it suspends when it wants randomness.
- Intuitions for nesting: sandboxes, virtualization, randomness adapters, mock objects.
How to eliminate interpretive overhead

Filinski: given a programming language with *delimited control*, add *layered side effects* (probabilities, memoization, etc.) while still running deterministic code at full speed.

► With delimited control, threads of execution can be suspended, resumed, copied, discarded.
► Represent stochastic programs not as data but as normal programs that suspend when they want randomness.
► Convert a stochastic program to a lazy tree of execution traces without interpretive overhead.
► Inference operates on this lazy tree. Implemented in OCaml.
► Inference is itself a stochastic program (e.g., importance sampling): it suspends when it wants randomness.
► Intuitions for nesting: sandboxes, virtualization, randomness adapters, mock objects.
How to eliminate interpretive overhead

Filinski: given a programming language with *delimited control*, add *layered side effects* (probabilities, memoization, etc.) while still running deterministic code at full speed.

- With delimited control, threads of execution can be suspended, resumed, copied, discarded.
- Represent stochastic programs not as data but as normal programs that suspend when they want randomness.
- Convert a stochastic program to a lazy tree of execution traces without interpretive overhead.
- Inference operates on this lazy tree. Implemented in OCaml.
- Inference is itself a stochastic program (e.g., importance sampling): it suspends when it wants randomness.
- Intuitions for nesting: sandboxes, virtualization, randomness adapters, mock objects.
What comes in the box?

Represent stochastic programs as normal programs using dist fail laze delay

Represent approximate inference as exploring a lazy tree of execution traces using sample_reify exact_reify collate at_least

Represent theory of mind as recursive invocations of approximate inference using multiple randomness sources (values of type random)
Example: plausibly deniable bribing

```ocaml
# generate random ;;
.<fun random ->
  let make_boolean () =
    laze random (fun () -> flip random 0.5) in
let q_1 = make_boolean () in
let q_2 = make_boolean () in
let q_3 = make_boolean () in
let q_4 = make_boolean () in
let q_5 = make_boolean () in
if not (q_4 () <> q_3 () <> q_1 ()) &&
  (q_2 () <> q_5 () <> q_4 ()) &&
  not (q_2 () <> q_3 () <> q_2 ()) &&
  not (q_3 () <> q_3 () <> q_5 ()) &&
  (q_3 () <> q_4 () <> q_5 ()
then q_5 () && q_4 ()
else fail random>.
```
let predict random innocent problem =
  match collate
    (sample_reify random (Some 2) 5 problem)
  with
  | [] -> fail random (* police rejects sentence *)
  | [_ , false] ->
    if innocent then Ticketed (* naïve driver *)
    else (* police perceives (unambiguous) bribe *)
    if flip random 0.5 then Bribe (* corrupt police *)
    else (* honest police *)
    if at_least 0.01 true (* criminal trial *)
      (sample_reify random (Some 4) 20 problem)
    then Ticketed (* court finds reasonable doubt *)
    else Convicted (* court finds bribe *)
  | _ -> Ticketed (* police does not perceive bribe *)
Example: plausibly deniable bribing

```ocaml
let prefer random = function
  | Ticketed      -> if flip random 0.2 then fail random
  | Bribe         -> ()
  | Convicted     -> fail random

let analyze problem =
  List.map snd (exact_reify problem)

let driver innocent random =
  let problem = .!(generate random) in
  prefer random (predict random innocent problem);
  analyze problem
```
Example: nested computational models

(* innocent driver *)
# collate (sample_reify random (Some 3) 1000
  (driver true)) ;;
[(0.0724, [true; false]); (* 21% *)
  (0.0498, [true]); (* 15% *)
  (0.2183, [false])] (* 64% *)

(* bribing driver *)
# collate (sample_reify random (Some 3) 1000
  (driver false)) ;;
[(0.0768, [true; false]); (* 30% *)
  (0.0457, [true]); (* 18% *)
  (0.1327, [false])] (* 52% *)
Summary

People people model model

- Concise, concrete, composable, **compilable**
- Can model unknown nesting depth

Next steps

- **Applications please!**
- Faster inference: conditional independence; memoization; Markov chain Monte Carlo
- From probability to expected value and maximum utility
- Imperfect information: staging