PRODEMO
Implementation of Model Checking for EPL

Jan van Eijck

December 13, 2013

Abstract

This report gives an implementation of a simplified probabilistic epistemic logic, together with updates with public announcement, public changes, and generic S5 action models.

The report implements a model checker for epistemic probability logic, documents the implementation, and demonstrates how it can be used to keep track of information flow about aleatory acts among multiple agents.

We give examples in the area of multi-agent probabilistic reasoning, including urn problems from elementary probability theory, transferred to a multi-agent setting, coin tossing problems, the modelling of noisy bit transfer, the solution of a puzzle by Lewis Carroll, and the solution of the Monty Hall problem. The final chapter makes the connection with Bayesian learning.

Contents

1 Introduction **2**

2 Lotteries and Probability Distributions **4**

3 Models **12**

4 Blissful Ignorance **26**

5 Formulas and Truth: Implementation **33**

6 Public Announcements **42**

7 Public Change **46**

8 Generic Update **49**

9 A Puzzle of Lewis Carroll **57**

10 The Hypochondriac **61**

11 The Puzzle of Monty Hall **64**

12 Model Transformations **68**

13 Urn Problems **72**

14 Noisy Data Transfer **81**

15 Bayesian Learning as Knowledge Growth **85**

Chapter 1

Introduction

> Dans les choses qui ne sont que vraisemblables, la différence des données que chaque homme a sur elles, est une des causes principales de la diversité des opinions que l'on voit régner sur les mêmes objects.
>
> Laplace [Lap14]

According to Laplace, probability theory is nothing but common sense reduced to calculation (Laplace, quoted in [Jay03]). If probability theory is the mathematical theory about uncertain events, and if epistemic logic is the tool for reasoning about knowledge and ignorance, then probabilistic epistemic logic or epistemic probability theory should be the theory of what agents with different information can know about the probabilities of uncertain events, about what other agents know about these probabilities, and so on. This should make an eminently practical tool.

The relation between probability and knowledge is simply this:

> Agent a knows φ iff the probability a assigns to φ equals 1.

In short: knowledge equals certainty. The set-up of our system will be such that ($P_a\varphi$ is the probability that agent a assigns to φ):

Certainty implies Truth $P_a\varphi = 1 \rightarrow \varphi$.

Positive Introspection into Certainty $P_a\varphi = 1 \to P_a(P_a\varphi = 1) = 1$.

Negative Introspection into Certainty $P_a\varphi < 1 \to P_a(P_a\varphi < 1) = 1$.

This means that the certainty operator is an $S5$ operator, and that we can define $K_a\varphi$ as $P_a\varphi = 1$.

The proposal in this report is related to earlier proposals on combining knowledge and probability [FH94, Koo03b, Koo03a, BGK09], but unlike these proposals, the present proposal equates knowledge with certainty, and uses the epistemic relation to build probability distributions on knowledge cells from lotteries.

A lottery is an assignment of positive rational numbers to the set of all possible worlds. Take any non-empty subset of worlds, and normalize the lottery for this. This gives a probability distribution for the subset. Thus, an agent a in some world w considers only the worlds in the epistemic equivalence class of w possible, and the subjective probabilities of the worlds in this class are determined by the a-lottery. This equates certainty with knowledge.

Useful overviews of the mathematics of uncertainty are [Par94] and [Hal03].

Probability theory and decision making is the topic of the wonderful [Kör08], and probability theory is promoted *as* logic in [Jay03]. An illuminating philosophical history of probability theory is [Hac75].

Chapter 2

Lotteries and Probability Distributions

The key concept that we start out from is that of a *lottery* on a set of possible worlds. The concept of a lottery is borrowed from utility theory, and was extensively used in Savage's famous book on the foundations of statistics [Sav72].

A W-lottery l is a function from a set of worlds W to the set of positive rationals, i.e., $l : W \to \mathbb{Q}^+$. Two W-lotteries l, l' are equivalent if for some $q \in \mathbb{Q}^+$, $l' = (\lambda p \mapsto q * p) \cdot l$. A W-lottery l is normalized on $B \subseteq W$ if $\sum\{l(w) \mid w \in B\} = 1$.

If we have a lottery $l : W \to \mathbb{Q}^+$ and a block $B \subseteq W$ in a partition of W, then this determines a probability distribution P on B, by means of (we assume that $B \neq \emptyset$):

$$P(w) = \frac{l(w)}{\sum\{l(w') \mid w' \in B\}}.$$

This operation allows us to combine knowledge and probability. Not only can we say that a does not know whether a coin is showing heads, but we can also model the fact that a knows that the coin has bias $\frac{2}{3}$ towards heads. Or we can say that a entertains two possibilities: the possibility that the coin is fair and the possibility that the coin has bias $\frac{2}{3}$ towards heads.

Module Declaration This paper is also the documentation of a literate [Knu92] Haskell [Jon03] program for epistemic probability model checking.

Here is the declaration of the main module.

```
module PRODEMO

where

import Data.List
```

Type Declarations, Helper Functions A probability is a rational number, a distribution is a list of probabilities.

```
type Prob = Rational

certain :: Prob
certain = 1/1

type Dist = [Prob]
```

Normalize a probability distribution:

```
normalize :: Dist -> Dist
normalize ps = [ p / s | p <- ps ] where s = sum ps
```

Normalize a probability distribution over a list of (atomic) events:

```
norm :: [(a,Prob)] -> [(a,Prob)]
norm xs = let
  ps = map snd xs
  s  = sum ps
 in [ (x, p / s ) | (x,p) <- xs ]
```

The basic concept in the logic of knowledge is that of epistemic uncertainty. If I am uncertain about whether a coin that has just been tossed is showing head or tail, this can be pictured as two situations related by my uncertainty.

In a probabilistic setting, if we know nothing about the probability distribution of a set of situations that we consider possible, we can apply the principle of indifference, and assign all possibilities the same probability:

```
indif :: [a] -> [(a,Prob)]
indif xs = map (\ x -> (x,p)) xs
   where p = 1 / fromIntegral (length xs)
```

So if we know nothing about the bias of a coin, the assumption that it is fair is an application of the principle of indifference.

Equivalence relations on a set of situations S can be implemented as partitions of S, where a partition is a family X_i of sets with the following properties (let I be the index set):

- For each $i \in I$, $\emptyset \neq X_i \subseteq S$.
- For $i \neq j$, $X_i \cap X_j = \emptyset$.
- $\bigcup_{i \in I} X_i = S$.

Here is a datatype for equivalence relations, viewed as partitions (lists of lists of items):

```
type Erel a = [[a]]
```

The block of an item in a partition:

```
bl :: Eq a => Erel a -> a -> [a]
bl r x = head (filter (elem x) r)
```

The restriction of a partition to a domain:

```
restrict :: Eq a => [a] -> Erel a -> Erel a
restrict domain =  nub . filter (/= [])
                   . map (filter (flip elem domain))
```

To add probabilities to this, one way to go is to lift the basic type from `a` to `(a,Prob)`.

Say there are two urns, U and V. U contains one black marble and two white marbles, V contains one black marble and one white marble. This is common knowledge among a, b and c. Now a selects one of the urns, without revealing which one to b, c. Then b picks a marble from it, without revealing the marble to a, c. How can we represent this situation? (See also [Ell61] for a discussion of this kind of example in the context of uncertainty and risk.)

Relevant possibilities are $(U, b), (U, w), (V, b), (V, w)$. But what are the probability distributions and the epistemic accessibilities?

a can distinguish the set $\{(U, b), (U, w)\}$ from the set $\{(V, b), (V, w)\}$. b can distinguish the set $\{(U, b), (V, b)\}$ from the set $\{(U, w), (V, w)\}$. For c any member of the set is possible: $\{(U, b), (U, w), (V, b), (V, w)\}$.

The probabilities the agents assign are:

$$a : \{(U, b) : \frac{1}{3}, (U, w) : \frac{2}{3}\}, \{(V, b) : \frac{1}{2}, (V, w) : \frac{1}{2}\},$$
$$b : \{(U, b) : \frac{1}{2}, (V, b) : \frac{1}{2}\}, \{(U, w) : \frac{1}{2}, (V, w) : \frac{1}{2}\},$$
$$c : \{(U, b) : \frac{1}{6}, (U, w) : \frac{2}{6}, (V, b) : \frac{1}{4}, (V, w) : \frac{1}{4}\}.$$

Figure 2.1 has a picture of this situation, with the probability information encoded in a lottery over the set of worlds in the model. In the picture, the uncertainty relation for a is given by solid lines, that for b by dashed lines, and that for c by dotted lines.

Now suppose we only have a truth value for the colour of the selected marble. Then the probability that the colour is black for a either is $\frac{1}{3}$ or $\frac{1}{2}$. For b it is either 0 or 1. For c it is $\frac{5}{12}$. This results in a different representation

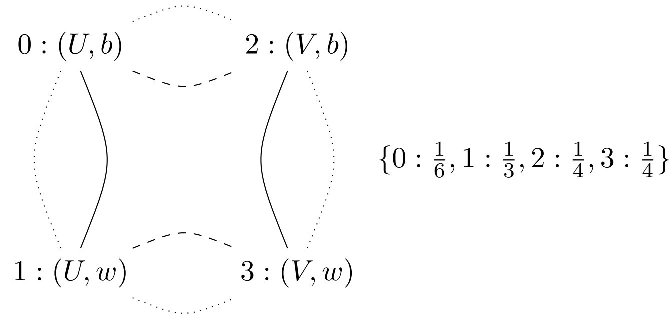


Figure 2.1: Model with uncertainty about urns

of the same situation, given in Figure 2.2. In this figure, the agents not only confuse worlds, but also lotteries. Agents b and c cannot distinguish between the two lotteries.

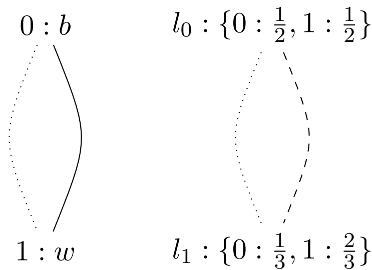


Figure 2.2: Model with uncertainty about lotteries

We will see below how these models also represent the probability assignments of agents to probability assignments of other agents.

Consider the following example from a textbook on probability theory [Gne75]. There are five urns with the following compositions: 2 urns with 2 white and 3 black balls each, 2 urns with 1 white and 4 black balls each, and one urn with 4 white balls and 1 black ball. A ball is chosen from one of the urns taken at random. It turns out to be white. What is the probability (after the experiment) that the ball was taken from the last urn? Figure 2.3 gives a picture of a model for this with a lottery over lotteries. There is just a single agent, and accessibility is represented by paths of solid lines in the picture.

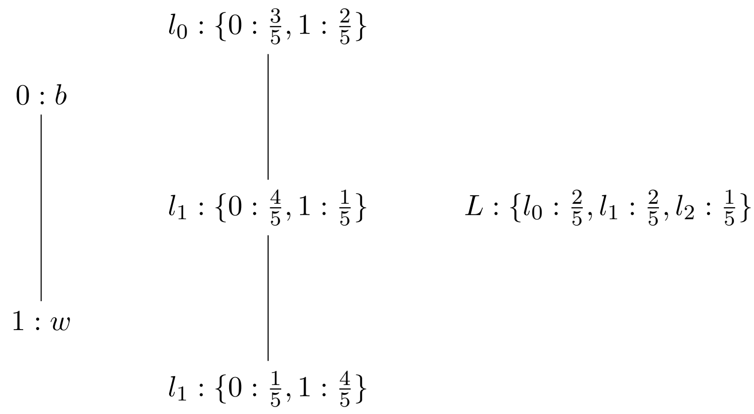


Figure 2.3: Model for the urn problem from [Gne75]

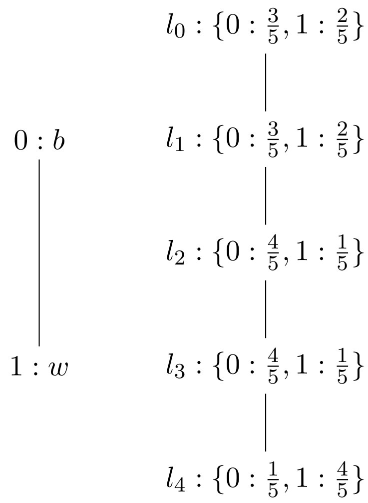


Figure 2.4: Alternative model for the urn problem from [Gne75]

However, in this report, I will not allow such lotteries over lotteries. Figure 2.4 gives an alternative representation of the problem, with each possible urn represented by a lottery. These lotteries can be incorporated in the valuation by introducing proposition letters to distinguish them.

Lotteries We will implement lotteries as lists.

```
type Lottery a = [(a,Prob)]
```

A lottery is a table. The following operation lifts tables to (partial) functions:

```
table2fct ::  Eq a => [(a,b)] -> a -> b
table2fct t = \ x -> maybe undefined id (lookup x t)
```

Use this to define an enumeration function:

```
enum :: Eq a => [a] -> a -> Int
enum xs = table2fct (zip xs [0..])
```

From Lotteries and Equivalence Relations on Worlds to Probability Distributions From lotteries to probability distributions:

```
restrictL :: Eq a => [a] -> Lottery a -> Lottery a
restrictL domain lot = let
   lot0 = filter (\ (w,_) -> elem w domain) lot
  in
   norm lot0
```

Probability of Events Probability computation of an event, where the event is viewed as a list of atomic events. It is not assumed that the lottery is normalized.

```
eventProb :: Eq a => [a] -> Lottery a -> Prob
eventProb xs ys = let
  zs = norm ys
  ps = map (table2fct zs) xs
 in
  sum ps
```

Chapter 3

Models

Suppose Alice is tossing a coin while Bob is watching. Both know that the coin can either be fair or biased (say, with bias $\frac{2}{3}$ towards heads). Bob does not know which coin Alice is using, but Alice knows.

A reasonable representation is as a model M with $W = \{H, T\}$. Representing the accessibilities as partitions, $\sim_A = \sim_B = \{\{H, T\}\}$. Two lotteries $l_0 = \{(H, \frac{1}{2}), (T, \frac{1}{2})\}$ and $l_1 = \{(H, \frac{2}{3}), (T, \frac{1}{3})\}$, and $\approx_A = \{\{l_0\}, \{l_1\}\}$, $\approx_B = \{\{l_0, l_1\}\}$. Figure 3.1 gives a picture (solid line for Alice, dashed lines for Bob).

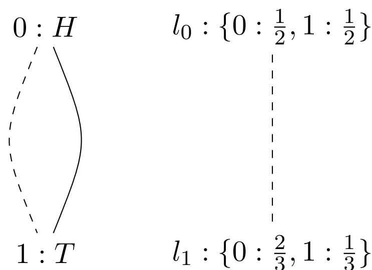


Figure 3.1: Model with uncertainty about coin bias

It is our goal in this paper to represent knowledge rather than belief. The actual probability distribution(s) over the actual worlds, as determined by the actual lotteries, can be viewed as objective probability.

For example, if a coin flip actually resulted from a toss with a coin with bias

$\frac{2}{3}$ towards heads, then this is an objective probability, regardless of the fact that some agent(s) may confuse it with a different probability distribution.

Agents We assume an infinite number of agents, with names a, b, c, d, e for the first five of them:

```
data Agent = Ag Int deriving (Eq,Ord)

a,b,c,d,e :: Agent
a = Ag 0; b = Ag 1; c = Ag 2; d = Ag 3; e = Ag 4

instance Show Agent where
  show (Ag 0) = "a"; show (Ag 1) = "b";
  show (Ag 2) = "c"; show (Ag 3) = "d" ;
  show (Ag 4) = "e";
  show (Ag n) = 'a': show n
```

Basic Propositions We assume an infinite set of basic propositions

$$p_0, p_1, \ldots, q_0, q_1, \ldots, r_0, r_1, \ldots, s_0, s_1, \ldots$$

Call this set **P**. In specific cases below we assume that some finite subset of **P** is used as the vocabulary of a model.

```
data Prp = P Int | Q Int | R Int | S Int deriving (Eq,Ord)
instance Show Prp where
  show (P 0) = "p"; show (P i) = "p" ++ show i
  show (Q 0) = "q"; show (Q i) = "q" ++ show i
  show (R 0) = "r"; show (R i) = "r" ++ show i
  show (S 0) = "s"; show (S i) = "s" ++ show i
```

Standard Epistemic Models A standard epistemic model for a set $\mathbf{P}$ of propositions and a set A of agents is a tuple (W, V, R) where

- W is a non-empty set of worlds,
- V is a valuation function that assigns to every $w \in W$ a subset of $\mathbf{P}$.
- R is a function that assigns to every agent $a \in A$ an equivalence relation R_a on W.

It is convenient to single out a subset $U \subseteq W$ as the actual worlds of the model.

An epistemic language $\mathcal{L}_0$ for this is the language of multi-agent epistemic logic given by

$$\varphi ::= \top \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid K_a\varphi$$

where p ranges over a set $\mathbf{P}$ of basic propositions and a ranges over a set of agents A.

Truth definition. Let $M = (W, V, R)$, let $w \in W$:

$$\begin{array}{rcl} M, w \models p & \text{iff} & p \in V(w) \\ M, w \models \neg\varphi & \text{iff} & \text{it is not the case that } M, w \models \varphi \\ M, w \models \varphi_1 \wedge \varphi_2 & \text{iff} & M, w \models \varphi_1 \text{ and } M, w \models \varphi_2 \\ M, w \models K_a\varphi & \text{iff} & \text{for all } w' \text{ with } wR_aw' : M, w' \models \varphi. \end{array}$$

In our implementation, we want to be general in our choice of valuations, so we make the type of basic proposition a parameter of the model. Also, we make the set of agents a component of the model.

A datatype for (standard) epistemic models. The first element is the list of agemts, the second element the list of states, the third element the valuation, the fourth element the list of epistemic relations for the agents, and the final element the list of actual situations (a constraint on the real world in the model). The type of the model determines the type of object that gets assigned by the valuation function. If the type is α then the valuation assigns lists of objects of type α.

```
type World = Int
```

```
data EM a = Mo
             [Agent]
             [World]
             [(World,[a])]
             [(Agent,Erel World)]
             [World]  deriving (Eq,Show)
```

If there are several actual states, this means that these are all candidates for the real world.

Epistemic Probability Models To change a standard epistemic model into an epistemic probability model, we assign to each agent a lottery over the set of worlds of the model. These lotteries represent subjective probabilities. If all agents assign the same lottery, this can be taken as a representation of objective probability.

So an epistemic probability model is a tuple (W, V, R, L) where

- (W, V, R) is a standard multi S5 epistemic model.
- L is function that assigns to each agent a a lottery L_a over W.

Again, it is convenient to mark a subset $U \subseteq W$ as the actual worlds of the model.

This choice of model for epistemic probability logic rests on the assumption that subjective probabilities are symmetric, and are linked to epistemic accessibility. Subjective probabilities are symmetric in the sense that the probability that gets assigned to w from the perspective of w' in the same epistemic accessibility block is the same as the probability in the reverse direction. Probabilities are linked to epistemic accessibility by normalizing the lotteries for each epistemic partition block. This turns a lottery into a list of probability distributions.

Epistemic probability language $\mathcal{L}$: a logical form language for epistemic probability statements. Assume p ranges over a set of basic propositions $\mathbf{P}$ and a ranges over a set A of agents. Assume q ranges over $\mathbb{Q}$.

$$\varphi \quad ::= \quad \top \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid t_1 + \cdots + t_n \geq q$$

$$t \quad ::= \quad q \mid tP_a\varphi$$

Terms of the form $q_1 P_a \varphi_1 + \cdots + q_n P_a \varphi_n$ were proposed in [FH94]. Boolean combinations of $q_1 P_a \varphi_1 + \cdots + q_n P_a \varphi_n \geq q$ can express all linear inequalities (see [Hal03]).

Abbreviations: We employ the usual abbreviations for $\bot$, $\varphi_1 \vee \varphi_2$, $\varphi_1 \rightarrow \varphi_2$, $\varphi_1 \leftrightarrow \varphi_2$.

We use $t < t'$ for $\neg t \geq t'$, we use $t > t'$ for $\neg t' \geq t$, we use $t \leq t'$ for $t' \geq t$, we use $t = t'$ for $t \geq t' \wedge t \leq t'$, we use $t \neq t'$ for $t > t' \vee t < t'$, and finally, we use $P_a(\varphi_1|\varphi_2) = q$ for $qP_a(\varphi_2) = P_a(\varphi_1 \wedge \varphi_2)$.

$P_a \varphi = q$ expresses that the probability of φ according to a equals q.

$P_a(\varphi_1|\varphi_2) = q$ expresses that according to a, the probability of φ_1, conditional on φ_2, equals q.

Truth definition for epistemic probability language: We define truth together with a function $[\![\cdot]\!]_w$ from terms to values in $[0..1] \subseteq \mathbb{Q}$, using an auxiliary probability function

$$P : A \times W \times \mathcal{L} \rightarrow [0..1] \subseteq \mathbb{Q}$$

We write $P(a, w, \varphi)$ as $P_{a,w}(\varphi)$. The functions $[\![\cdot]\!]$ and P are defined by mutual recursion, together with the truth definition.

Let $M = (W, V, R, L)$, let $w \in W$.

$$\begin{array}{rcl}
M, w \models \top & & \text{always} \\
M, w \models p & \text{iff} & p \in V(w) \\
M, w \models \neg\varphi & \text{iff} & \text{it is not the case that } M, w \models \varphi \\
M, w \models \varphi_1 \wedge \varphi_2 & \text{iff} & M, w \models \varphi_1 \text{ and } M, w \models \varphi_2 \\
M, w \models t_1 + \cdots + t_n \geq q & \text{iff} & [\![t_1]\!]_w + \cdots + [\![t_n]\!]_w \geq q
\end{array}$$

$$\begin{array}{rcl}
[\![q]\!]_w & := & q \\
[\![qP_a\varphi]\!]_w & := & q \times P_{a,w}(\varphi).
\end{array}$$

$$P_{a,w}(\varphi) = \frac{\sum\{l_a(u) \mid wR_a u \text{ and } M, u \models \varphi\}}{\sum\{l_a(u) \mid wR_a u\}}.$$

$P_{a,w}(\varphi)$ gives the probability that a assigns to φ in w; this is normalized for the set of all worlds that a confuses with w.

It follows from the reflexivity of R_a plus the fact that all lottery values are positive that the denominator in the definition of $P_{a,w}(\varphi)$ is non-zero, so $P_{a,w}(\varphi)$ is well-defined.

The new element in an epistemic probability model is a table of lotteries, one for each agent.

```
data Pem a  = MO
             [Agent]
             [World]
             [(World,[a])]
             [(Agent,Erel World)]
             [World]
             [(Agent,Lottery World)]
 deriving (Eq,Show)
```

Extracting an epistemic relation from a model:

```
rel :: Agent -> Pem a -> Erel World
rel ag (MO _ _ _ rels _ _) = table2fct rels ag
```

Extracting the lottery for an agent from a model:

```
lot :: Agent -> Pem a -> Lottery World
lot ag (MO _ _ _ _ _ lots) =  table2fct lots ag
```

Extracting the set of actual worlds from a model:

```
actual :: Pem a -> [World]
actual (MO _ _ _ _ as _ ) = as
```

Lookup for a world, given a knowledge cell, in a lottery:

```
lookupLot :: Eq a => Lottery a -> [a] -> a -> Prob
lookupLot lot xs x = let
   v = table2fct lot x
   vs = map (table2fct lot) xs
  in
   v / (sum vs)
```

Note that non-probabilistic epistemic models are a special case in the new setting. A classical epistemic model can be lifted to an epistemic probability model by assigning each agent a lottery that is indifferent between all worlds. This boils down to the assumption of a uniform distribution of probabilities for each block in each epistemic partition, and common knowledge among the agents that all agents apply the principle of indifference.

If $M = (W, V, R)$ is an epistemic model, then M^{indif} is the epistemic probability model (W', V', R', L') where

$$\begin{aligned} W' &= W \\ V' &= V \\ R' &= R \\ L &= \lambda a \lambda w \in W \mapsto 1 \end{aligned}$$

Explanation: M^{indif} is the epistemic probability model that is the result of putting a uniform probability distribution on the worlds in M, and making this uniform probability distribution common knowledge. This can be viewed as making it common knowledge that all agents apply the principle of indifference to all facts they are uncertain about.

Figure 3.2 gives an example of a standard epistemic model with one agent uncertain about p and the other agent uncertain about q, and a single uniform lottery shared by the agents.

Conversely, if $M = (W, V, R, L)$ is an epistemic probability model, then we can map this to an epistemic model M° by putting $M^\circ = (W, V, R)$, i.e., M° is the result of removing the lottery information from M.

Next, we wish to argue for the equation of knowledge and certainty. Define a translation $t : \mathcal{L}_0 \to \mathcal{L}$ from the language of multi-agent epistemic logic to

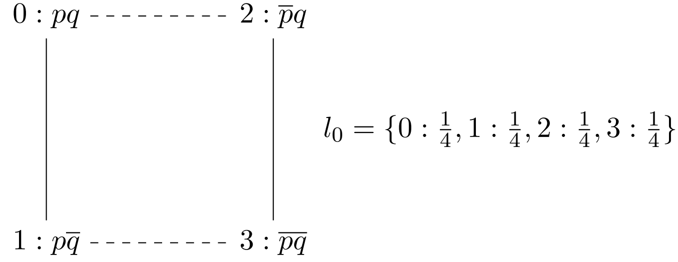


Figure 3.2: Model with indifference lottery

the language of epistemic probability logic by means of:

$$\begin{aligned} t(p) &:= p \\ t(\neg\varphi) &:= \neg t(\varphi) \\ t(\varphi_1 \wedge \varphi_2) &:= t(\varphi_1) \wedge t(\varphi_2) \\ t(K_a\varphi) &:= P_a t(\varphi) = 1 \end{aligned}$$

This translates knowledge statements of $\mathcal{L}_0$ into certainty statements of $\mathcal{L}$, and allows us to prove the following Certainty Theorem.

Theorem 1 (Certainty) *For any epistemic probability model*

$$M = (W, V, R, L),$$

any world-index pair w for M, any $\varphi \in \mathcal{L}_0$:

$$M^\circ, w \models \varphi \text{ iff } M, w \models t(\varphi).$$

Proof. Induction on the structure of φ. The only case we have to check is $K_a\varphi$.

$M^\circ, w \models K_a\varphi$ iff (truth definition of $K_a\varphi$) for all $u \in W^\circ$ with $wR^\circ_a u$:

$$M^\circ, u \models \varphi$$

iff (induction hypothesis) for all $u \in W$ with $wR_a u$:

$$M, u \models t(\varphi)$$

iff (def of $P_{a,w}$ in M)

$$P_{a,w} t(\varphi) = 1$$

iff (definition of t)

$$M, w, i \models t(K_a\varphi).$$

□

This theorem motivates the following abbreviation for $\mathcal{L}$:

Use $K_a\varphi$ for $P_a\varphi = 1$.

The abbreviation $K_a\varphi$ for $P_a\varphi = 1$ reflects the equation of knowledge and certainty. It follows immediately from the Certainty Theorem that the certainty operator $P_a\varphi = 1$ is an S5 operator.

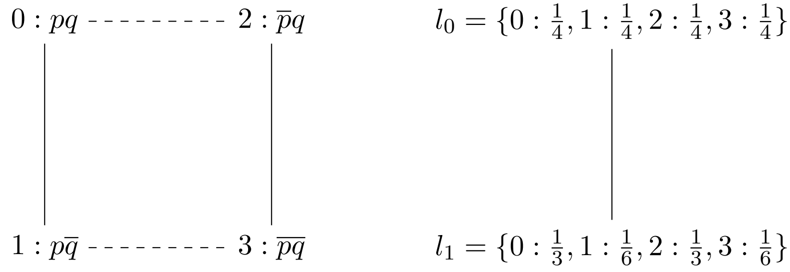


Figure 3.3: Model with uncertainty about q bias

Look at example model M in Figure 3.3. In this model, at world 0 and lottery l_0, the probability that a (represented by solid lines) assigns to p is 1, so K_ap is true at $0, l_0$. K_aq is false at $0, l_0$, for the probability that a assigns to q is less than 1. It equals the average of the probabilities that a assigns to q for the two lotteries. For lottery l_0, the probability a assigns to q is $\frac{1}{2}$. For lottery l_1, it is $\frac{2}{3}$. So the formula $q_a = \frac{1}{2}$ is true at $0, l_0$, and so is the formula $P_aq = \frac{7}{12}$. The probability that a assigns to $P_aq = \frac{7}{12}$ at $0, l_0$ equals 1, so $K_a(P_aq = \frac{7}{12})$ is also true at $0, l_0$. On the other hand, $K_a(q_a = \frac{1}{2})$ is false at $0, l_0$, for a confuses the two lotteries.

Note that agents do not have to know whether a coin flip was made with a biased coin or not in order to be able to compute a probability.

Lift epistemic models to probabilistic epistemic models.

```
liftEM :: EM a -> Pem a
liftEM (Mo agents worlds val rels actuals) =
  MO agents worlds val rels actuals lots
  where
    lots  = [ (a,indif worlds) | a <- agents ]
```

An example classical epistemic model, where a knows about p, but b, c do not:

```
ex0 :: EM Prp
ex0 = Mo
 [a,b,c]
 [0,1]
 [(0,[P 0]),(1,[])]
 [(a,[[0],[1]]),
  (b,[[0,1]]),
  (c,[[0,1]])]
 [0,1]
```

Lifting this to a probabilistic epistemic model:

```
example0 = liftEM ex0
```

This gives:

```
*PRODEMO> example0
MO [a,b,c] [0,1] [(0,[p]),(1,[])]
[(a,[[0],[1]]),(b,[[0,1]]),(c,[[0,1]])]
[0,1]
[(a,[(0,1 % 2),(1,1 % 2)]),
 (b,[(0,1 % 2),(1,1 % 2)]),
 (c,[(0,1 % 2),(1,1 % 2)])]
```

The two outcomes are equally likely, and this is common knowledge. But only a knows the outcome. This could be a representation of a coin flip with a fair coin, with a observing the outcome, while b and c are watching (but cannot see the outcome themselves).

Here is a model where a knows about p (i.e., a can distinguish between p worlds and $\neg p$ worlds), b knows about q, and c knows about neither.

```
ex1 :: EM Prp
ex1 = Mo
 [a,b,c]
 [0..3]
 [(0,[P 0,Q 0]),(1,[P 0]),(2,[Q 0]),(3,[])]
 [(a,[[0,1],[2,3]]),
  (b,[[0,2],[1,3]]),
  (c,[[0..3]])]
 [0..3]
```

Lifting this to a probabilistic setting:

```
example1 = liftEM ex1
```

This gives:

```
\begin{verbatim}
MO [a,b,c] [0,1,2,3]
[(0,[p,q]),(1,[p]),(2,[q]),(3,[])]
[(a,[[0,1],[2,3]]),(b,[[0,2],[1,3]]),(c,[[0,1,2,3]])]
[0,1,2,3]
[(a,[(0,1 % 4),(1,1 % 4),(2,1 % 4),(3,1 % 4)]),
 (b,[(0,1 % 4),(1,1 % 4),(2,1 % 4),(3,1 % 4)]),
 (c,[(0,1 % 4),(1,1 % 4),(2,1 % 4),(3,1 % 4)])]
```

This is the model from Figure 3.2 (except for the fact that a third agent c is present). In this example model, any world is equally likely to be the real

world, and all probability distributions are uniform. In 0, p and q are true, in 1 p is true, in 2 q is true, and in 3 both p and q are false.

In this model, agent a can distinguish p-worlds from $\neg p$ worlds, but not q-worlds from $\neg q$ worlds. With b it is the other way around, while c can make neither of these distinctions. This is all common knowledge.

This could be a model of two tosses with fair coins, with the outcome p or $\neg p$ of the first toss observed by a while b, c are present, and the outcome q or $\neg q$ of the second toss observed by b while a, c are present.

```
*PRODEMO> rel a example1
[[0,1],[2,3]]
*PRODEMO> rel b example1
[[0,2],[1,3]]
*PRODEMO> rel c example1
[[0,1,2,3]]
```

It is also common knowledge that all combinations of p and q are equally likely:

```
*PRODEMO> lot a example1
[(0,1 % 4),(1,1 % 4),(2,1 % 4),(3,1 % 4)]
*PRODEMO> lot b example1
[(0,1 % 4),(1,1 % 4),(2,1 % 4),(3,1 % 4)]
*PRODEMO> lot c example1
[(0,1 % 4),(1,1 % 4),(2,1 % 4),(3,1 % 4)]
```

c and a are both willing to bet with each other about q. c and b are both willing to bet with each other about p. c will not bet about p with a, nor about q with b. a will not bet about q with b. b will not bet about p with a.

Figure 3.4 gives an example involving two different lotteries, expressing that two probability distributions are possible. Agent a knows the value of p but cannot distinguish the two lotteries, agents b does not know the value of p and cannot distinguish the two lotteries (dashed lines in the picture), agent c does not know the value of p (a dotted line in the picture) but can distinguish the two lotteries.

The example model of Figure 3.4 is implemented as follows:

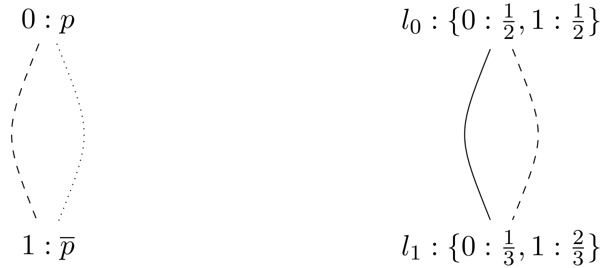


Figure 3.4: Model with two possible probability distributions for p

```
example2 :: Pem Prp
example2 = MO
 [a,b,c]
 [0,1,2,3]
 [(0,[P 0]), (1,[]),(2,[P 0]), (3,[])]
 [(a,[[0,2],[1,3]]),
  (b,[[0,1,2,3]]),
  (c,[[0,1],[2,3]])]
 [0,1,2,3]
 [(a,[(0,1/2),(1,1/2),(2,1/3),(3,2/3)]),
  (b,[(0,1/2),(1,1/2),(2,1/3),(3,2/3)]),
  (c,[(0,1/2),(1,1/2),(2,1/3),(3,2/3)])]
```

In this example, the distribution between states 0 and 1 is either given by a fair coin flip, or given by a coin flip with bias $\frac{2}{3}$ towards 1. c can distinguish between these two possibilities, the other agents cannot. a knows the outcome of the toss, while b and c do not. This is like the urn model given in Figures 2.1 and 2.2.

```
*PRODEMO> rel a example2
[[0],[1]]
*PRODEMO> rel b example2
[[0,1]]
*PRODEMO> rel c example2
[[0,1]]
*PRODEMO> lots a example2
```

```
[[[(0,1 % 2),(1,1 % 2)]],[[(0,1 % 3),(1,2 % 3)]]]
*PRODEMO> lots b example2
[[[(0,1 % 2),(1,1 % 2)],[(0,1 % 3),(1,2 % 3)]]]
*PRODEMO> lots c example2
[[[(0,1 % 2),(1,1 % 2)]],[[(0,1 % 3),(1,2 % 3)]]]
```

Chapter 4

Blissful Ignorance

Blissful ignorance is the state where you don't know anything, but you know also that there is no reason to worry, for you know that it is common knowledge that nobody knows anything.

A Kripke model where every agent from agent set A is in blissful ignorance about a (finite) set of propositions P, with $|P| = k$, looks as follows:

$$M = (W, V, R) \text{ where}$$

$$\begin{array}{rcl} W & = & \{0, \ldots, 2^k - 1\} \\ V & = & \text{any surjection in } W \to \mathcal{P}(P) \\ R & = & \{(a, \{W\}) \mid a \in A\}. \end{array}$$

Note that V is in fact a bijection, for $|\mathcal{P}(P)| = 2^k = |W|$. Each agent is completely ignorant, for all members of W are in the same partition block (the equivalence relation R_a is given as a partition).

In a probabilistic setting, we still have to add that all worlds are equally likely, for all agents.

Generating Models for Blissful Ignorance

```
initM :: Ord a => [Agent] -> [a] -> EM a
initM ags props = (Mo ags worlds val accs points)
  where
    worlds  = [0..(2^k-1)]
    k       = length props
    val     = zip worlds (sortL (powerList props))
    accs    = [ (ag,[worlds]) | ag  <- ags        ]
    points  = worlds
```

This uses powerList and sortL (sort by length).

```
powerList  :: [a] -> [[a]]
powerList  [] = [[]]
powerList  (x:xs) =
  (powerList xs) ++ (map (x:) (powerList xs))

sortL :: Ord a => [[a]] -> [[a]]
sortL  = sortBy
  (\ xs ys -> if length xs > length ys
                then LT
              else if length xs < length ys
                then GT
              else compare xs ys)
```

The probabilistic version of this is obtained by a lift:

```
initPM :: Ord a => [Agent] -> [a] -> Pem a
initPM as ps = liftEM (initM as ps)
```

This gives:

```
*PRODEMO> initPM [a,b,c] [P 0, Q 0, R 0]
```

```
MO [a,b,c] [0,1,2,3,4,5,6,7] [(0,[p,q,r]),(1,[p,q]),(2,[p,r]),
(3,[q,r]),(4,[p]),(5,[q]),(6,[r]),(7,[])]
[(a,[[0,1,2,3,4,5,6,7]]),(b,[[0,1,2,3,4,5,6,7]]),
 (c,[[0,1,2,3,4,5,6,7]])]
[0,1,2,3,4,5,6,7]
[(a,[(0,1 % 8),(1,1 % 8),(2,1 % 8),(3,1 % 8),(4,1 % 8),
     (5,1 % 8),(6,1 % 8),(7,1 % 8)]),
 (b,[(0,1 % 8),(1,1 % 8),(2,1 % 8),(3,1 % 8),(4,1 % 8),
     (5,1 % 8),(6,1 % 8),(7,1 % 8)]),
 (c,[(0,1 % 8),(1,1 % 8),(2,1 % 8),(3,1 % 8),(4,1 % 8),
     (5,1 % 8),(6,1 % 8),(7,1 % 8)])]
```

Blissful ignorance about urn situations In the context of probability, urn problems are interesting. We can represent an urn containing m white marbles and n black marbles as (m, n). In the context of probabilistic epistemic logic, this can be viewed as a model of type `(Int,Int)`.

Fix the colours (white or black) of k marbles:

```
distMarbles :: Int -> [(Int,Int)]
distMarbles k = [ (m,n)
  | m <- [0..k], n <- [0..k], m+n == k ]
```

Generate a model with blisfull ignorance about k marbles:

```
initU :: [Agent] -> Int -> EM (Int,Int)
initU ags k = (Mo ags worlds val accs points)
  where
    worlds  = [0..k]
    urns    = map (\ xs -> [xs]) (distMarbles k)
    val     = zip worlds urns
    accs    = [ (ag,[worlds]) | ag  <- ags   ]
    points  = worlds
```

Lift this to a probabilistic version:

```
initU0 :: [Agent] -> Int -> Pem (Int,Int)
initU0 as k = liftEM (initU as k)
```

This gives:

```
*PRODEMO> initU0 [a] 2
M0 [a] [0,1,2] [(0,[(0,2)]),(1,[(1,1)]),(2,[(2,0)])]
[(a,[[0,1,2]])]
[0,1,2]
[[(0,1 % 3),(1,1 % 3),(2,1 % 3)]]
[(a,[[0]])]
[0]
```

See Figure 4.1 for a picture.

$0 : (0, 2)$

$1 : (1, 1)$ $\quad\quad l_0 = \{0 : \frac{1}{3}, 1 : \frac{1}{3}, 2 : \frac{1}{3}\}$

$2 : (2, 0)$

Figure 4.1: Urn model with indifference about marble distribution

There is still something wrong with this, for in general there are $\binom{m+n}{m} = \binom{m+n}{n}$ ways of drawing a sequence of $m+n$ marbles form an urn containing m white and n black marbles. The principle of indifference has led us astray with the initialisation. We have applied it in the wrong way. The correct initialisation recognizes the binomial distribution, as follows:

```
initURN :: [Agent] -> Int -> Pem (Int,Int)
initURN ags k =
   (MO ags worlds val accs points lots)
  where
    worlds  = [0..k]
    urns    = map (\ xs -> [xs]) (distMarbles k)
    val     = zip worlds urns
    accs    = [ (ag,[worlds]) | ag  <- ags   ]
    points  = worlds
    lot     = zip [0..]
     (map (\ [(m,n)] -> fromIntegral $ binom (m+n) m) urns)
    lots    = [ (a,lot) | a <- ags ]
```

This uses `binom` as an implementation of $\binom{n}{k}$:

```
binom n 0 = 1
binom n k | n < k = 0
          | otherwise =
            (n * binom (n-1) (k-1)) `div` k
```

This gives:

```
*PRODEMO> initURN [a] 2
MO [a] [0,1,2] [(0,[(0,2)]),(1,[(1,1)]),(2,[(2,0)])]
[(a,[[0,1,2]])] [0,1,2]
[(a,[(0,1 % 1),(1,2 % 1),(2,1 % 1)])]
```

We can also give a different representation, in terms of sequences of marbles.

```
data Marble = W | B deriving (Eq,Show)
```

This gives rise to a different function for marble distribution:

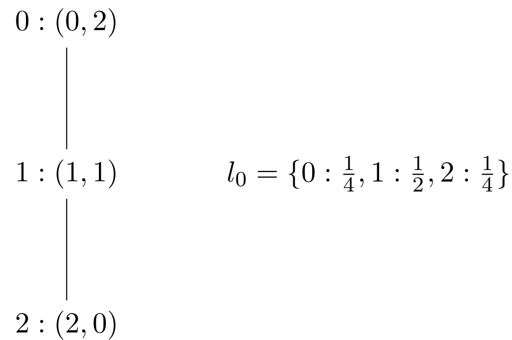


Figure 4.2: Urn model with binomial marble distribution

```
distMarbles2 :: Int -> [[Marble]]
distMarbles2 0 = [[]]
distMarbles2 k = map (W:) (distMarbles2 (k-1))
              ++ map (B:) (distMarbles2 (k-1))
```

Now in a model of type `EM Marble` a valuation assigns a list of marbles, and we get the following initialisation function:

```
initU2 :: [Agent] -> Int -> EM Marble
initU2 ags k =  (Mo ags worlds val accs points)
  where
    worlds  = [0..2^k-1]
    urns    = distMarbles2 k
    val     = zip worlds urns
    accs    = [ (ag,[worlds]) | ag  <- ags   ]
    points  = worlds
```

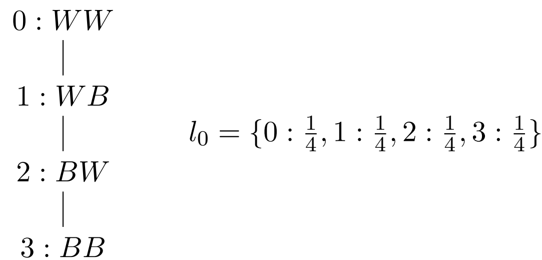
Use this for urn initialisation:

```
initURN2 :: [Agent] -> Int -> Pem Marble
initURN2 as k = liftEM (initU2 as k)
```

This gives:

```
*PRODEMO> initURN2 [a] 2
MO [a] [0,1,2,3] [(0,[W,W]),(1,[W,B]),(2,[B,W]),
(3,[B,B])] [(a,[[0,1,2,3]])] [0,1,2,3]
[(a,[(0,1 % 4),(1,1 % 4),(2,1 % 4),(3,1 % 4)])]
```

See Figure 4.3 for a picture.



$$l_0 = \{0 : \frac{1}{4}, 1 : \frac{1}{4}, 2 : \frac{1}{4}, 3 : \frac{1}{4}\}$$

Figure 4.3: Alternative urn model with indifference about marble distribution

According to this model, the probability that both marbles are black is $\frac{1}{4}$. How is this possible? This is the question John Maynard Keynes posed in his book [Key63].

The answer is that in the model of Figure 4.1 the principle of indifference was applied in the wrong way. Compare this to the puzzle about the probability that a family with two kids has two boys. This probability is $\frac{1}{4}$ rather than $\frac{1}{3}$, for the relevant cases are BB, BG, GB and GG. If we represent this as $2B$, $B + G$ and $2G$, then we must bear in mind that the pattern $B + G$ is twice as likely as the two other patterns.

It is all a matter of representation. If we represent "consider an arbitrary urn with k marbles, either black or white" in terms of (m, n), then we have to add the lottery information that the pattern (m, n) gets lottery value $\binom{m+n}{m}$.

Chapter 5

Formulas and Truth: Implementation

In the implementation we take conjunctions and disjunctions over lists, and we add a type parameter for the type of atomic information. Also, it will be convenient to declare an `Info` datatype for expressing properties of valuations tables. This will allow us to use valuations in a more versatile way than for just encoding truth of propositional atoms. Since the `Info` information is functional, the function itself cannot be displayed.

```
data Inf a = Inf ([a] -> Bool)

instance Show (Inf a) where
  show (Inf f) = "..."

instance Eq (Inf a) where
  (Inf f) == (Inf g) = True
```

```
data Form a = Top
            | Info (Inf a)
            | Prp a
            | Ng (Form a)
            | Conj [Form a]
            | Disj [Form a]
            | Kn Agent (Form a)
            | Geq (Term a) (Term a)
            | Eq  (Term a) (Term a)
            | Pr Agent (Form a) Rational
          deriving (Eq,Show)
```

For convenience we have added

`Kn Agent (Form a)` for $K_a\varphi$,

`Pr Agent (Form a) Rational` for $P_a\varphi = q$,

```
data Term a = Rat Rational
            | Prb Agent (Form a)
            | Cprb Agent (Form a) (Form a)
            | Cmpl (Term a)
            | Prod [Term a]
          deriving (Eq,Show)
```

`Cprb a f1 f2` expresses the probability that a assigns to `f1`, conditioned by `f2`. Also added for convenience.

Example formulas:

```
p,q,r, p_or_q :: Form Prp
p = Prp (P 0); q = Prp (Q 0); r = Prp (R 0)
p_or_q = Disj [p,q]
```

Example formulas of a different type, to be used later for encoding urn problems.

```
hasWhite, hasBlack :: Form (Int,Int)
hasWhite = Info (Inf (\ [(m,n)] -> m > 0))
hasBlack = Info (Inf (\ [(m,n)] -> n > 0))
```

A useful abbreviation:

```
impl :: Form a -> Form a -> Form a
impl form1 form2 = Disj [Ng form1, form2]
```

Semantic interpretation for this logical form language. We evaluate with respect to a state:

```
isTrueAt :: Eq a => Pem a -> World ->
            Form a -> Bool
```

`Info` formulas are checked against the index of a world. Atomic propositions are checked against the valuation result for the current world and the current lottery index.

```
isTrueAt m w Top = True
isTrueAt
  m@(MO agents worlds val acc points lots)
  w (Info (Inf f)) = let
    value = table2fct val w
  in
    f value
isTrueAt
  m@(MO agents worlds val acc points lots)
  w (Prp p) = let
    props = table2fct val w
  in
    elem p props
```

Treatment of Booleans as usual:

```
isTrueAt m w (Ng f)    = not (isTrueAt m w f)
isTrueAt m w (Conj fs) = and (map (isTrueAt m w) fs)
isTrueAt m w (Disj fs) = or  (map (isTrueAt m w) fs)
```

Treatment of knowledge operators:

```
isTrueAt
 m@(MO agents worlds val acc points lots)
 w (Kn ag f) = let
    r  = rel ag m
    wb = bl r w
  in
    and (map (\ x -> isTrueAt m x f) wb)
```

The truth of term semi-equalities and equalities is computed by means of an `eval` function for terms:

```
isTrueAt m w (Geq t1 t2) =
   (eval m w t1) >= (eval m w t2)
isTrueAt m w (Eq t1 t2) =
   (eval m w t1) == (eval m w t2)
```

The truth of a probability statement at a world is given in terms of the probability function P, implemented as `prob`.

$P_a\varphi = q$ is true at w iff the probability of φ, computed according to the lottery for a, equals q.

```
isTrueAt
  m@(MO agents worlds val acc points lots) w
  (Pr ag f q) = prob m ag w f == q
```

The evaluation function for terms maps every term to a rational in the interval $[0..1] \subseteq \mathbb{Q}$:

```
eval ::  Eq a => Pem a -> World ->
         Term a -> Prob
eval _ _ (Rat q) = q
eval m w (Prb a form)  = prob m a w form
eval m w (Cprb a f1 f2) = let
   p1 = eval m w (Prb a (Conj [f1,f2]))
   p2 = eval m w (Prb a f2)
 in
   if p2 == 0 then 0 else p1 / p2
eval m w (Cmpl t)      = 1 - (eval m w t)
eval m w (Prod ts)     = product (map (eval m w) ts)
```

This uses the function P, implemented as `prob`.

```
prob ::  Eq a =>
         Pem a ->
         Agent -> World -> Form a -> Prob
prob m@(MO agents worlds val acc points lots)
  ag w f   = let
    r      = rel ag m
    wb     = bl r w
    g      = table2fct (lot ag m)
    ps     = [ (x, g x) | x <- wb ]
    qs     = filter (\ (x,_) -> isTrueAt m x f) ps
  in
    sum (map snd qs) / sum (map snd ps)
```

Examples Recall `example0`, where agent a can distinguish p from $\neg p$, while the other two agents b, c cannot and hold these two possibilities for equally probable. We have:

```
*PRODEMO> prob example0 a 0 p
1 % 1
*PRODEMO> prob example0 a 0 (Ng p)
0 % 1
*PRODEMO> prob example0 b 0 p
1 % 2
*PRODEMO> prob example0 b 0 (Ng p)
1 % 2
*PRODEMO> prob example0 a 1 p
0 % 1
*PRODEMO> prob example0 a 1 (Ng p)
1 % 1
*PRODEMO> prob example0 b 1 p
1 % 2
```

We can also express these facts with formulas:

```
*PRODEMO> isTrueAt example0 0 (Pr b p (1/2))
True
```

```
*PRODEMO> isTrueAt example0 0 (Pr b p 1)
False
*PRODEMO> isTrueAt example0 0 (Pr a p 1)
True
*PRODEMO> isTrueAt example0 0 ((Pr a p 1))
True
*PRODEMO> isTrueAt example0 0 (Kn b(Pr a p 1))
False
*PRODEMO> isTrueAt example0 0 (Kn b (Disj [Pr a p 1, Pr a p 0]))
True
```

b will not bet with a on the value of p.

In `example2` there are two probability distributions for p. Agent c does not confuse these.

```
*PRODEMO> isTrueAt example2 0 (Pr c p (1/2))
True
```

But agent b does:

```
*PRODEMO> isTrueAt example2 0 (Pr b p (1/2))
False
```

Still, agent b does not know that c knows the probability of p:

```
*PRODEMO> isTrueAt example2 0 (Kn b (Pr c p (1/2)))
False
```

Here are the probabilities that the three agents assign to p:

```
*PRODEMO> prob example2 a 0 p
1 % 1
*PRODEMO> prob example2 b 0 p
5 % 12
*PRODEMO> prob example2 c 0 p
1 % 2
```

The coin flip that resulted in the probability of p was fair. But b does not know this.

```
PRODEMO> eval example2 0 (Cprb b p (Lot 0))
1 % 2
*PRODEMO> eval example2 0 (Prb b p)
5 % 12
```

On the other hand, c knows that the coin flip was fair:

```
*PRODEMO> eval example2 0 (Prb c p)
1 % 2
```

Here is the implementation of the urn example from [Gne75] (see Figure 2.4 above). Black is represented by p and white by $\neg p$. The five urns are $q_1, \ldots, q_5$.

```
gnedenko :: Pem Prp
gnedenko = MO
  [a] [0..9]
  [(0,[P 0,Q 1]),(1,[Q 1]),
   (2,[P 0,Q 2]),(3,[Q 2]),
   (4,[P 0,Q 3]),(5,[Q 3]),
   (6,[P 0,Q 4]),(7,[Q 4]),
   (8,[P 0,Q 5]),(9,[Q 5])]
  [(a,[[0..9]])] [0..9]
  [(a,[(0,3/5),(1,2/5),(2,3/5),(3,2/5),
       (4,4/5),(5,1/5),(6,4/5),(7,1/5),
       (8,1/5),(9,4/5)])]
```

The textbook question: what is the probability that the ball is from urn (lottery) l_4, given that the ball is white?

```
gnedenkoT :: Term Prp
gnedenkoT = Cprb a (Prp (Q 5)) (Ng p)
```

We get:

```
*PRODEMO> eval gnedenko 0 0 gnedenkoT
2 % 5
```

Chapter 6

Public Announcements

The effect of a public announcement φ on an epistemic model is that the set of worlds of that model gets limited to the worlds where φ is true, and the accessibility relations get restricted accordingly.

If $M = (W, V, R, L)$ is an epistemic probability model, then $M^\varphi = (W', V', R', L')$ is the epistemic probability model given by

- $W' = \{w \in W \mid M, w \models \varphi\}$.
- V' is the restriction of V to W'.
- R' assigns to each agent a the relation R'_a that is the restriction of R_a to W'.
- L' is the function that assigns to each agent a the restriction of the lottery L_a to W'.

Here is the implementation:

```
upd_pa :: Eq a => Pem a -> Form a -> Pem a
upd_pa
 m@(MO agents states val rels actual lots) f =
  (MO agents states1 val1 rels1 actual1 lots1)
   where
   states1 = [ s | s <- states, isTrueAt m s f ]
   val1    = [ (s,ps) | (s,ps) <- val, elem s states1  ]
   rels1   = [(ag,restrict states1 r) | (ag,r) <- rels ]
   actual1 = filter (flip elem states1) actual
```

We normalize the lotteries for the new set of worlds.

```
   g       = norm . filter (\ (w,p) -> elem w states1)
   lots1   = [(a, g lot) | (a,lot) <- lots ]
```

Example:

```
example1a = upd_pa example1 p_or_q
```

```
*PRODEMO> example1a
MO [a,b,c] [0,1,2] [(0,[p,q]),(1,[p]),(2,[q])]
[(a,[[0,1],[2]]),(b,[[0,2],[1]]),(c,[[0,1,2]])]
[0,1,2]
[(a,[(0,1 % 3),(1,1 % 3),(2,1 % 3)]),
 (b,[(0,1 % 3),(1,1 % 3),(2,1 % 3)]),
 (c,[(0,1 % 3),(1,1 % 3),(2,1 % 3)])]
```

A public announcement can give information about a distribution. Note that b and a can learn that the coin is biased by being told the probability that c assigns to p:

```
*PRODEMO> upd_pa example2 (Pr c p (1/3))
MO [a,b,c] [2,3] [(2,[p]),(3,[])]
 [(a,[[2],[3]]),(b,[[2,3]]),(c,[[2,3]])]
 [2,3]
 [(a,[(2,1 % 3),(3,2 % 3)]),
  (b,[(2,1 % 3),(3,2 % 3)]),
  (c,[(2,1 % 3),(3,2 % 3)])]
```

Renaming of Worlds We may wish to rename the worlds, so that the list again starts from 0.

```
rename :: Pem a -> Pem a
rename (MO agents states val rels actual lots) =
 MO agents states1 val1 rels1 actual1 lots1
   where
    f       = enum states
    states1 = map f states
    val1    = map (\ (x,y) -> (f x,y)) val
    rels1   = map (\ (x,r) -> (x, map (map f) r)) rels
    actual1 = map f actual
    lots1   = map
     (\ (x,lot) -> (x, map (\ (x,p) -> (f x,p)) lot)) lots
```

This gives:

```
*PRODEMO> rename $ upd_pa example2 (Pr c p (1/3))
MO [a,b,c] [0,1] [(0,[p]),(1,[])]
[(a,[[0],[1]]),(b,[[0,1]]),(c,[[0,1]])] [0,1]
[(a,[(0,1 % 3),(1,2 % 3)]),
 (b,[(0,1 % 3),(1,2 % 3)]),
 (c,[(0,1 % 3),(1,2 % 3)])]
```

A series of public announcement updates:

```
upds_pa :: Eq a =>
           Pem a -> [Form a] -> Pem a
upds_pa = foldl upd_pa
```

Example:

```
*PRODEMO> upds_pa example1 [Prp (P 0),Ng (Prp (Q 0))]
MO [a,b,c] [1] [(1,[p])]
[(a,[[1]]),(b,[[1]]),(c,[[1]])] [1]
[(a,[(1,1 % 1)]),
 (b,[(1,1 % 1)]),
 (c,[(1,1 % 1)])]
```

We can now extend the language with an operator for public announcement. The language $\mathcal{L}^{\mathrm{PA}}$ is given by:

$$\varphi \quad ::= \quad \top \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid t_1 + \cdots + t_n \geq q \mid [\varphi]\varphi$$

$$t \quad ::= \quad q \mid qP_a\varphi$$

Semantics of the new language construct:

$$M, w \models [\varphi_1]\varphi_2 \quad \text{iff} \quad M, w \models \varphi_1 \text{ implies } M^{\varphi_1}, w \models \varphi_2.$$

Chapter 7

Public Change

Following [BvEK06], we represent changes in the world as substitutions. A substitution maps proposition letters to formulas. A type for a substitution as a table is `[(a,Form a)]`.

```
type Subst a = [(a,Form a)]
```

Substitutions as functions, constructed from tables:

```
sub :: Eq a => Subst a -> a -> Form a
sub subst x =
  if elem x (map fst subst) then
     table2fct subst x else (Prp x)
```

A public change update:

```
upd_pc :: Eq a => [a] -> Pem a
               -> Subst a -> Pem a
upd_pc
  ps
  m@(MO agents states val rels actual lots)
  sb =
  (MO agents states val1 rels actual lots)
   where
   val1 = [ (s, [p | p <- ps,
                      isTrueAt m s (sub sb p)])
                         | s  <- states ]
```

A series of public change updates:

```
upds_pc :: Eq a => [a] -> Pem a
                     -> [Subst a] -> Pem a
upds_pc ps = foldl (upd_pc ps)
```

A list of propositions:

```
exampleprops = [P i | i <- [0..3 ]]
                ++
               [Q i | i <- [0..3 ]]
```

An example substitution for these propositions:

```
sexample =  [(P i, Prp (P 0)) | i <- [0..3 ]]
             ++
            [(Q i, Prp (Q 0)) | i <- [0..3 ]]
```

This can be used to initialize a valuation function:

```
exmple :: Pem Prp
exmple = initPM [a,b,c] [P 0,Q 0]
```

```
*PRODEMO>  upd_pc exampleprops exmple sexample
MO [a,b,c] [0,1,2,3] [(0,[p,p1,p2,p3,q,q1,q2,q3]),
(1,[p,p1,p2,p3]),(2,[q,q1,q2,q3]),(3,[])]
[(a,[[0,1,2,3]]),(b,[[0,1,2,3]]),(c,[[0,1,2,3]])]
[0,1,2,3]
[(a,[(0,1 % 4),(1,1 % 4),(2,1 % 4),(3,1 % 4)]),
 (b,[(0,1 % 4),(1,1 % 4),(2,1 % 4),(3,1 % 4)]),
 (c,[(0,1 % 4),(1,1 % 4),(2,1 % 4),(3,1 % 4)])]
```

Chapter 8

Generic Update

In [BGK09], update models for probabilistic epistemic logic are built from sets of formulas that are mutually exclusive. We will stay a bit closer to the original update model from [BMS98, BM04].

A probabilistic update model is like a probabilistic epistemic model, but with the valuation function replaced by a function that assigns preconditions and actions (substitutions) to events.

Update models for S5 are like epistemic S5 models, but with their valuations replaced by precondition functions. A PA function assigns a Precondition formula and an Action (i.e., a substitution) to each state/lottery-index pair.

Let $\mathcal{L}$ be our epistemic language, and let $\mathbf{S}$ be the set of substitutions (represented as lists of bindings) for that language. It is assumed that the sets $\mathbf{P}$ of basic propositions and A of agents are fixed.

An update model for probabilistic epistemic logic is a tuple

$$(W, P, R, L)$$

where

- W is a non-empty set of events
- P is a function $W \times L \to \mathcal{L} \times \mathbf{S}$ that assigns a pair (φ, S) consisting of a $\mathcal{L}$ formula φ (the precondition) and a substitution S (the action) to each world w.
- R is a function that maps each agent a to an equivalence R_a on W.

- L is a function from agents to W-lotteries.

Again it is convenient to single out a subset $U \subseteq W$ as the actual events of the model.

Note how close this is to the definition of an epistemic model. The only change is the replacement of the valuation by an precondition/action function.

For clarity we introduce a type synonym for events.

```
type Event = Int
```

```
data UM a = UM
            [Agent]
            [Event]
            [(Event,(Form a, Subst a))]
            [(Agent,Erel Event)]
            [Event]
            [(Agent, Lottery Event)]
 deriving (Eq,Show)
```

Extracting a suspicion from an event model:

```
ssp :: Agent -> UM a -> Erel Event
ssp ag (UM _ _ _ susps _ _) = table2fct susps ag
```

Extracting a lottery from an event model:

```
elot :: Agent -> UM a -> Lottery Event
elot ag (UM _ _ _ _ _ lots) = table2fct lots ag
```

The list of agents in an update model should match that of the input epistemic model. This can be achieved by means of the following type:

```
type FUM a = [Agent] -> UM a
```

An example FUM:

```
fum1 :: FUM Prp
fum1 = \ ags -> UM
  ags
  [0,1]
  [(0,(p_or_q,[])),(1,(Top,[]))]
  ((a,[[0],[1]]) : [ (x,[[0,1]]) | x <- ags \\ [a] ])
  [0]
  [(x, [(0,1/2),(1,1/2)]) | x <- ags ]
```

In this example, a learns that $p \vee q$, while all other agents remain unaware of this. All probability distributions are uniform.

Another example FUM:

```
fum2 :: FUM Prp
fum2 = \ ags -> UM
  ags
  [0,1,2,3]
  [(0,(Top,[(P 0,Top)])),(1,(Top,[(P 0,Ng Top)])),
   (2,(Top,[(P 0,Top)])),(3,(Top,[(P 0,Ng Top)]))]
  ((a,[[0,1],[2,3]]) : [ (x,[[0,1,2,3]]) | x <- ags\\[a]])
  [0,1]
  [(x, [(0,1/4),(1,1/4),(2,1/3),(3,1/6)]) | x <- ags ]
```

In this example, p gets value $\top$ with probability $\frac{1}{2}$. a learns that the probability that p is true is $\frac{1}{2}$, while all other agents think it is possible that this probability is $\frac{2}{3}$.

Getting the precondition from a PA table:

```
precond :: Eq a => [(Event,(Form a, Subst a))]
                -> Event -> Form a
precond table = fst . table2fct table
```

Getting the action from a PA table:

```
action :: Eq a => [(Event,(Form a, Subst a))]
               -> Event -> Subst a
action table = snd . table2fct table
```

Cartesian product of two lists:

```
cP :: [a] -> [b] -> [(a,b)]
cP xs ys = [(x,y) | x <- xs, y <- ys ]
```

Product of two partitions:

```
prod :: (Eq a,Eq b) => Erel a -> Erel b -> Erel (a,b)
prod r s =
  [ [ (x,y) | x <- b, y <- c ] | b <- r, c <- s ]
```

Constrained product of two partitions. The constraint is given by a domain of pairs.

```
prodD :: (Eq a,Eq b) =>
              [(a,b)] -> Erel a -> Erel b -> Erel (a,b)
prodD domain r s =
  [ [ (x,y) | x <- b, y <- c, elem (x,y) domain ] |
              b <- r, c <- s ]
```

Generic update is now defined as:

```
upd :: Eq a => [a] ->
       Pem a -> FUM a -> Pem a
upd ps
    m@(MO agents states val rels actual lots)
    fum =
  MO agents1 states1 val1 rels1 actual1 lots1
  where
```

Now we get a long list of auxiliary definitions that are all part of the definition of `upd`. First we specify the update model `um`, which is the result of applying `fum` to the agent list of `m`.

```
  um@(UM agents1 events pat susp aevents elots)
          = fum agents
```

The new state set is given by a constrained product:

```
  states0 =
    [ (s,e) | s <- states, e <- events,
              isTrueAt m s (precond pat e) ]
  f       = enum states0
  states1 = map f states0
```

The new valuation is given by taking the effect of the substitution actions into account:

```
val1    = [ (f (s,e),[ p | p <- ps,
                          isTrueAt m s
                          (sub (action pat e) p)]) |
                      s <- states,
                      e <- events,
                      elem (s,e) states0  ]
```

The new relations are given as a constrained product:

```
rels1   = [ (a,map (map f)
                (prodD states0 (rel a m) (ssp a um))) |
                                           a <- agents1 ]
```

A pair (s, e) is actual if its components were actual in their input models:

```
actual1 = map f [ (s,e) | (s,e) <- states0,
                           elem s actual,
                           elem e aevents ]
```

The new lottery table is construed as a product from the old lottery tables, using the function `newLot` defined below.

```
lots1     = [ (x, newLot states0 lot elot) |
                 (x, lot) <- lots,
                 (y,elot) <- elots, x == y ]
```

And this ends the definition of `upd`.

Combining a world lottery and an event lottery to a new world lottery:

```
newLot :: [(World,Event)] -> Lottery World
          -> Lottery Event -> Lottery World
newLot domain lot1 lot2 = let
   ws = map fst lot1
   es = map fst lot2
   f  = enum domain
   g  = \ (w,e) ->
          (table2fct lot1 w) *  (table2fct lot2 e)
 in
   norm $ map (\ (w,e) -> (f (w,e), g (w,e))) domain
```

This gives:

```
*PRODEMO> upd [P 0, Q 0] example1 fum1
MO [a,b,c] [0,1,2,3,4,5,6] [(0,[p,q]),(1,[p,q]),(2,[p]),(3,[p]),
 (4,[q]),(5,[q]),(6,[])]
[(a,[[0,2],[1,3],[4],[5,6]]),
 (b,[[0,1,4,5],[2,3,6]]),
 (c,[[0,1,2,3,4,5,6]])]
[0,2,4]
[(a,[(0,1 % 7),(1,1 % 7),(2,1 % 7),(3,1 % 7),
     (4,1 % 7),(5,1 % 7),(6,1 % 7)]),
 (b,[(0,1 % 7),(1,1 % 7),(2,1 % 7),(3,1 % 7),
     (4,1 % 7),(5,1 % 7),(6,1 % 7)]),
 (c,[(0,1 % 7),(1,1 % 7),(2,1 % 7),(3,1 % 7),
     (4,1 % 7),(5,1 % 7),(6,1 % 7)])]
```

The actual worlds of this model are the worlds where $p \vee q$ is true and where agent a knows this. The other agents do not know whether a knows $p \vee q$. Also, a still does not know whether q, and b still does not know whether p.

Next, let us start with a model of complete ignorance, with no propositions.

```
m0 :: Pem Prp
m0 = initPM [a,b] []
```

This gives:

```
*PRODEMO> m0
MO [a,b] [0] [(0,[])] [(a,[[0]]),(b,[[0]])] [0]
[(a,[(0,1 % 1)]),(b,[(0,1 % 1)])]
```

Result of updating this with `fum2` is that a distinction between p and $\neg p$ gets introduced:

```
*PRODEMO> upd [P 0] m0 fum2
MO [a,b] [0,1,2,3] [(0,[p]),(1,[]),(2,[p]),(3,[])]
[(a,[[0,1],[2,3]]),(b,[[0,1,2,3]])] [0,1]
[(a,[(0,1 % 4),(1,1 % 4),(2,1 % 3),(3,1 % 6)]),
 (b,[(0,1 % 4),(1,1 % 4),(2,1 % 3),(3,1 % 6)])]
```

In this model, p is true with probability $\frac{1}{2}$, and a knows this, but b confuses this with the possibility that p is true with probability $\frac{2}{3}$.

Finally, make a public announcement that $P_a p = \frac{1}{2}$:

```
*PRODEMO> upd_pa (upd [P 0] m0 fum2) (Pr a p (1/2))
MO [a,b] [0,1] [(0,[p]),(1,[])]
[(a,[[0,1]]),(b,[[0,1]])] [0,1]
[(a,[(0,1 % 2),(1,1 % 2)]),
 (b,[(0,1 % 2),(1,1 % 2)])]
```

The result is that now everybody knows that p is true with probability $\frac{1}{2}$.

Chapter 9

A Puzzle of Lewis Carroll

An urn contains a single marble, either white or black. Mr A puts another marble in the urn, a white one. The urn now contains two marbles. Next, Mrs B draws one of the two marbles from the urn. It turns out to be white. What is the probability that the other marble is also white [Gar81]?

Call the first white marble p and the second one q. Mrs B does not know whether she is drawing from $\neg p + q$ or from $p + q$.

Let's start with a model of complete ignorance about p, for two agents a, b:

```
m1 :: Pem Prp
m1 = initPM [a,b] [P 0]
```

This gives:

```
*PRODEMO> m1
MO [a,b] [0,1] [(0,[p]),(1,[])]
[(a,[[0,1]]),(b,[[0,1]])] [0,1]
[(a,[(0,1 % 2),(1,1 % 2)]),(b,[(0,1 % 2),(1,1 % 2)])]
```

An update model for telling a the value of p, while b does not learn this fact.

```
um1 :: FUM Prp
um1 = \ ags -> UM
   ags
   [0,1]
   [(0,(p,[])),(1,(Ng p,[]))]
   ((a,[[0],[1]]) : [ (x,[[0,1]]) | x <- ags \\ [a] ])
   [0,1]
   [(x,[(0,1/2),(1,1/2)]) | x <- ags ]
```

The result of updating with this:

```
m2 :: Pem Prp
m2 = upd [P 0] m1 um1
```

This gives:

```
*PRODEMO> m2
MO [a,b] [0,1] [(0,[p]),(1,[])]
[(a,[[0],[1]]),(b,[[0,1]])] [0,1]
[(a,[(0,1 % 2),(1,1 % 2)]),(b,[(0,1 % 2),(1,1 % 2)])]
```

Putting a second white marble in the urn. This can be implemented as a public change that makes q true:

```
m3 :: Pem Prp
m3 = upd_pc [P 0,Q 0] m2 [(Q 0,Top)]
```

The result:

```
*PRODEMO> m3
MO [a,b] [0,1] [(0,[p,q]),(1,[q])]
[(a,[[0],[1]]),(b,[[0,1]])] [0,1]
[(a,[(0,1 % 2),(1,1 % 2)]),(b,[(0,1 % 2),(1,1 % 2)])]
```

An update model for removing either p or q from the urn. Nobody knows which of these two takes place. Note that removing p from the urn has as precondition that p is true, and similarly for q.

```
um2 :: FUM Prp
um2 = \ ags -> UM
   ags
   [0,1]
   [(0,(p,[(P 0,Ng Top)])),(1,(q,[(Q 0,Ng Top)]))]
   [ (x,[[0,1]]) | x <- ags ]
   [0,1]
   [(x, [(0,1/2),(1,1/2)]) | x <- ags ]
```

The result of updating with this.

```
m4 :: Pem Prp
m4 = upd [P 0,Q 0] m3 um2
```

Here is what this model looks like:

```
*PRODEMO> m4
MO [a,b] [0,1,2] [(0,[q]),(1,[p]),(2,[])]
[(a,[[0,1],[2]]),(b,[[0,1,2]])] [0,1,2]
[(a,[(0,1 % 3),(1,1 % 3),(2,1 % 3)]),
 (b,[(0,1 % 3),(1,1 % 3),(2,1 % 3)])]
```

What is the probability that the other marble is also white? In our setting: what is the probability of $p \vee q$?

Well, it is different for a and b. Here is the whole story:

```
*PRODEMO> prob m4 a 0 p_or_q
1 % 1
*PRODEMO> prob m4 a 1 p_or_q
1 % 1
```

```
*PRODEMO> prob m4 a 2 p_or_q
0 % 1
*PRODEMO> prob m4 b 0 p_or_q
2 % 3
*PRODEMO> prob m4 b 1 p_or_q
2 % 3
*PRODEMO> prob m4 b 2 p_or_q
2 % 3
```

Chapter 10

The Hypochondriac

From [BGK09]: A hypochondriac knows that his chances of having a certain disease are 1 against 99.999. The hypochondriac learns that if he has the disease, chances that a certain gland is swollen are 97 against 3. If he does not have the disease, it is 100% sure that the gland is not swollen. Next he feels the gland, but not being a doctor he is not sure whether the gland is swollen or not. Say 50 against 50. What is the probability he has the disease?

Let p represent that he has the disease. Then the initial model looks like this:

```
disease0 :: Pem Prp
disease0 = MO
 [a]
 [0,1]
 [(0,[P 0]),(1,[])]
 [(a,[[0,1]])]
 [0,1]
 [(a, [(0,(1/100000)),(1,(99999/100000))])]
```

The information about the meaning of swelling of the gland, represented as q.

```
gland :: FUM Prp
gland = \ ags -> UM
 ags
 [0,1,2]
 [(0,(p,[(Q 0,Top)])),(1,(p,[(Q 0,Ng Top)])),
  (2,(Ng p,[(Q 0,Ng Top)]))]
 [ (x,[[0,1,2]]) | x <- ags ]
 [0,1,2]
 [ (x, [(0,97/100),(1,3/100),(2,1)]) | x <- ags ]
```

Updating with this:

```
disease1 = upd [P 0, Q 0] disease0 gland
```

The hypochondriac's uncertain observation about q:

```
observation :: FUM Prp
observation = \ ags -> UM
 ags
 [0,1]
 [(0,(q,[])),(1,(Ng q,[]))]
 [ (x,[[0,1]]) | x <- ags ]
 [0,1]
 [ (x,[(0,1/2),(1,1/2)]) | x <- ags ]
```

Updating with this:

```
disease2 = upd [P 0, Q 0] disease1 observation
```

The probability of a having the disease:

```
*PRODEMO> prob disease2 a 0 p
1 % 100000
```

Now suppose a physician comes in, and feels the gland. She is 99% sure that the gland is swollen:

```
expert :: FUM Prp
expert = \ ags -> UM
 ags
 [0,1]
 [(0,(q,[])),(1,(Ng q,[]))]
 [ (x,[[0,1]]) | x <- ags ]
 [0,1]
 [ (x, [(0,99/100),(1,1/100)]) | x <- ags ]
```

This increases the probability that the hypochondriac has the disease:

```
disease3 = upd [P 0, Q 0] disease1 expert
```

```
*PRODEMO> prob disease3 a 0 p
1601 % 1668251
```

Still, the chance that the hypochondriac has the disease is less than 1 in 1000.

Chapter 11

The Puzzle of Monty Hall

In the well-known Monty Hall puzzle [VS90], the quizmaster Monty lets you choose between three doors. Behind one of these there is a prize (in the original show, a car), behind the other two doors there is nothing (or, in the original show, a goat).

You make your choice, and then the quizmaster teases you by opening one of the other doors, showing that the prize is not there. He then gives you the option of revising your choice. The puzzle is this: What is the most rational course of action, change to the other door or not?

Let's implement this. Let p_1, p_2, p_3 represent that the prize is behind door $1, 2, 3$. Let q_1, q_2, q_3 represent that your choice is door $1, 2, 3$.

Initially, one of p_1, p_2, p_3 is true and one of q_1, q_2, q_3 is true.

Let's declare the formulas:

```
p1,p2,p3,q1,q2,q3 :: Form Prp
p1 = Prp (P 1); p2 = Prp (P 2); p3 = Prp (P 3)
q1 = Prp (Q 1); q2 = Prp (Q 2); q3 = Prp (Q 3)
```

The quizmaster acts as follows.

- If $p_i \wedge q_i$ is true then he randomly chooses $j \neq i$ and announces $\neg p_j$.
- If $p_i \wedge q_j$ is true for $i \neq j$ then he announces $\neg p_k$ for $k \neq i$, $k \neq j$.

This can be represented as an update, as follows.

```
montyFum :: FUM Prp
montyFum = \ags -> UM
  ags
  [0..11]
  [(0,(Conj [p1,q1,Ng p2],[])),
   (1,(Conj [p1,q3,Ng p2],[])),
   (2,(Conj [p3,q1,Ng p2],[])),
   (3,(Conj [p3,q3,Ng p2],[])),
   (4,(Conj [p1,q1,Ng p3],[])),
   (5,(Conj [p1,q2,Ng p3],[])),
   (6,(Conj [p2,q1,Ng p3],[])),
   (7,(Conj [p2,q2,Ng p3],[])),
   (8,(Conj [p2,q2,Ng p1],[])),
   (9,(Conj [p2,q3,Ng p1],[])),
   (10,(Conj [p3,q2,Ng p1],[])),
   (11,(Conj [p3,q3,Ng p1],[]))]
  [ (a,[[0..3],[4..7],[8..11]]) | a <- ags ]
  [0..11]
  [ (x, [(0,(1/2)),(1,1),(2,1),(3,(1/2)),(4,(1/2)),(5,1),
    (6,1),(7,(1/2)),(8,(1/2)),(9,1),(10,1),(11,(1/2))])
    | x <- ags ]
```

Here the choice of lottery is perhaps in need of explanation. The lottery expresses that in cases where the prize is behind the door that is chosen, Monty is indifferent between opening either of the other two doors, while in cases where the prize is not behind the chosen door, Monty has no choice which door to open: the door that does *not* hide the prize.

The partition for a expresses that a can see which door Monty opens (thereby showing that it does not hide the prize). I.e., a confuses each of the $\neg p_i$ cases, but can distinguish between a $\neg p_i$ and a $\neg p_j$ case for $i \neq j$.

Initial model: one of the p_i is true, but a does not know which one:

```
montyInit :: Pem Prp
montyInit =
  MO
  [a]
  [0..2]
  [(0,[P 1]),(1,[P 2]),(2,[P 3])]
  [(a,[[0..2]])]
  [0..2]
  [(a , indif [0..2])]
```

Now a makes her choice:

```
aChoice :: FUM Prp
aChoice = \ags -> UM
  ags
  [0..2]
  [(0,(Top,[(Q 1,Top)])),
   (1,(Top,[(Q 2,Top)])),
   (2,(Top,[(Q 3,Top)]))]
  [(x,[[0],[1],[2]]) | x <- ags ]
  [0..2]
  [(x, indif [0..2]) | x <- ags ]
```

The result of updating the initial model with the choice:

```
monty1 =
  upd [P 1, P 2, P 3, Q 1, Q 2, Q 3] montyInit aChoice
```

The result of Monty's action of opening a door with no prize behind it:

```
monty2 =
  upd [P 1, P 2, P 3, Q 1, Q 2, Q 3] monty1 montyFum
```

The statement that a's initial choice is the correct choice:

$$(p_1 \wedge q_1) \vee (p_2 \wedge q_2) \vee (p_3 \wedge q_3).$$

Implementation:

```
correct :: Form Prp
correct = Disj [Conj [p1,q1],Conj [p2,q2], Conj [p3,q3]]
```

This has probability $\frac{1}{3}$, whatever the situation:

```
*PRODEMO> [ prob monty2 a w correct | w <- [0..11] ]
[1 % 3,1 % 3,1 % 3,1 % 3,1 % 3,1 % 3,1 % 3,1 % 3,1 % 3,1 % 3,1 % 3,1 % 3]
```

So the rational course of action is to switch doors.

Chapter 12

Model Transformations

Functions of type $\alpha \to \beta$ can be used to map a type α model into a type β model, as follows:

```
mapPEM ::  ([a] -> [b]) -> Pem a -> Pem b
mapPEM
  f
  m@(MO agents states val rels actual lots) =
  MO agents states val1 rels actual lots
  where
    val1 = map (\ (x,ps) -> (x, f ps)) val
```

This can be made more versatile by replacing the type `[a] -> [b]` by `[a] -> Lottery [b]`.

```
gmapPEM :: (Eq a,Eq b)
        => ([a] -> Lottery [b]) -> Pem a -> Pem b
gmapPEM
  lotf
  m@(MO agents states val rels actual lots) =
  MO agents states1 val1 rels1 actual1 lots1
  where
```

Construct the table of states and action results. The indices are used to keep track of multiple copies of old states.

```
table0 =
  [ ((s,i),(x,p)) | s     <- states,
                    value <- [table2fct val s],
                    ((x,p),i) <- zip (lotf value) [0..] ]
```

Construct a map from old states to lists of states and action results.

```
k = \w -> [ (w,i) |
            ((v,i),(x,p)) <- table0, v == w ]
l = concat . map k
```

Construct the new states:

```
states0 = l states
f       = enum states0
states1 = map f states0
```

Construct the new actual states:

```
actual1 = map f $ l actual
```

Construct the new relations:

```
h = map (map f) . (map l)
rels1 = [(ag, h r) | (ag,r) <- rels ]
```

Construct the new valuation:

```
  val1 = zip [0..] (map (\ ((s,i),(x,p)) -> x) table0)
```

Construct the new lottery list:

```
  lots1 = [ (ag, norm [ (f (s,j), p * q) |
              ((s,j),(x,p)) <- table0,
               q            <- [table2fct (lot ag m) s] ]) |
               ag           <- agents                      ]
```

To demonstrate this, we introduce a type for results of coin tosses.

```
data Coin = H | T | F | U deriving (Eq,Ord,Show)
```

The two values F and U will not concern us here (see page 85).

The toss of a fair coin:

```
fairToss = \ xs -> [(H:xs, 1/2),(T:xs, 1/2)]
```

Tossing a fair coin for the first time:

```
toss1 = gmapPEM fairToss (initPM [a] [])
```

This gives:

```
PRODEMO> toss1
MO [a] [0,1] [(0,[H]),(1,[T])] [(a,[[0,1]])] [0,1]
[(a,[(0,1 % 2),(1,1 % 2)])]
*PRODEMO> gmapPEM fairToss toss1
MO [a] [0,1,2,3] [(0,[H,H]),(1,[T,H]),(2,[H,T]),(3,[T,T])]
[(a,[[0,1,2,3]])] [0,1,2,3]
[(a,[(0,1 % 4),(1,1 % 4),(2,1 % 4),(3,1 % 4)])]
```

In Chapter 13 we will use generic updating to represent urn problems.

Chapter 13

Urn Problems

We give a different representation of the Lewis Carroll puzzle, and develop a general framework for dealing with urn problems. Here is a function for creating an epistemic representation of an urn with m white marbles and n black marbles, with two agents a and b.

```
urn :: (Int,Int) -> Pem (Int,Int)
urn (m,n) = liftEM u where
  u :: EM (Int,Int)
  u = Mo
        [a,b]
        [0]
        [(0,[(m,n)])]
        [(a,[[0]]),(b,[[0]])]
        [0]
```

An urn (m, n) contains m white marbles and n black marbles.

Putting a white marble into an urn is an update from (m, n) to $(m + 1, n)$. Similarly, putting a black marble into an urn (m, n) results in an urn $(m, n+1)$.

Use `mapPEM` for putting a white marble into an urn.

```
putWhite :: Pem (Int,Int) -> Pem (Int,Int)
putWhite = mapPEM (\ [(m,n)] -> [(m+1,n)])
```

And for putting a black marble into an urn.

```
putBlack :: Pem (Int,Int) -> Pem (Int,Int)
putBlack = mapPEM (\ [(m,n)] -> [(m,n+1)])
```

Putting a marble in an urn, with indifference between white and black.

```
putMarble :: Pem (Int,Int) -> Pem (Int,Int)
putMarble = gmapPEM putM
```

```
putM ::  [(Int,Int)] -> Lottery [(Int,Int)]
putM  [(m,n)] = [([(m+1,n)],1),([(m,n+1)],1)]
```

Likewise, the update action of drawing a marble can be modelled using a lottery action.

```
drawMarble ::  Pem (Int,Int) -> Pem (Int,Int)
drawMarble = gmapPEM drawM
```

The lottery action uses a lottery that keeps track of the number of possibities.

```
drawM :: [(Int,Int)] -> Lottery [(Int,Int)]
drawM [(m,n)] = let
          m1 = fromIntegral m
          n1 = fromIntegral n
  in
    if (m+n) <= 0  then []
    else if m == 0 then [([(0,n-1)],n1)]
    else if n == 0 then [([(m-1,0)],m1)]
    else [([(m-1,n)],m1),([(m,n-1)],n1)]
```

Drawing a white marble, with the proportion between white and black marbles determining the distribution.

```
drawWhite :: Pem (Int,Int) -> Pem (Int,Int)
drawWhite = gmapPEM drawW
```

```
drawW :: [(Int,Int)] -> Lottery [(Int,Int)]
drawW [(m,n)] = let  m1 = fromIntegral m
  in
    if (m+n) <= 0  then []
    else if m == 0 then []
    else [([(m-1,n)],m1)]
```

Drawing a black marble, with the proportion between white and black marbles determining the distribution.

```
drawBlack :: Pem (Int,Int) -> Pem (Int,Int)
drawBlack =  gmapPEM drawB
```

```
drawB :: [(Int,Int)] -> Lottery [(Int,Int)]
drawB [(m,n)] = let  n1 = fromIntegral n
  in
    if (m+n) <= 0  then []
    else if n == 0 then []
    else [([(m,n-1)],n1)]
```

Here we have the Lewis Carroll puzzle again.

```
carroll = drawWhite $ putWhite $ putMarble $ urn (0,0)
```

This gives:

```
*PRODEMO> carroll
MO [a,b] [0,1] [(0,[(1,0)]),(1,[(0,1)])]
[(a,[[0,1]]),(b,[[0,1]])] [0,1]
[(a,[(0,2 % 3),(1,1 % 3)]),(b,[(0,2 % 3),(1,1 % 3)])]
```

The probability of the final marble being white: let's define a formula for that:

```
carrollF :: Form (Int,Int)
carrollF = Info
         (Inf (\ [(m,n)] -> (m,n) == (1,0)))
```

```
*PRODEMO> prob carroll b 0 carrollF
2 % 3
```

The following function computes the probability (as an average over actual worlds and actual lotteries) of drawing a white marble from an urn.

```
probWhite :: Pem (Int,Int) -> Prob
probWhite
  m@(MO _ states val _ actual lots) = let
    f     = table2fct val
    g     = table2fct (lot a m)
    h     = \ p -> (f p, g p)
    k     = \ ([(m,n)],prb) -> let
                 m1 = fromIntegral m
                 n1 = fromIntegral n
             in prb * (m1/(m1+n1))
    fs    = map (k.h) actual
  in
    sum fs
```

Let us try this on a problem from [Usp37].

> An urn contains 3 white and 5 black balls. One ball is drawn, and *its colour unnoted*, put aside. Then another ball is drawn and we are required to find the probability that it is black or white.

This is what the urn looks like, in our current representation:

```
*PRODEMO> urn (3,5)
MO [a,b] [0] [(0,[(3,5)])] [(a,[[0]]),(b,[[0]])] [0]
[(a,[(0,1 % 1)]),(b,[(0,1 % 1)])]
```

This is the result of randomly selecting a marble and removing it:

```
*PRODEMO> drawMarble $ urn (3,5)
MO [a,b] [0,1] [(0,[(2,5)]),(1,[(3,4)])]
[(a,[[0,1]]),(b,[[0,1]])] [0,1]
[(a,[(0,3 % 8),(1,5 % 8)]),(b,[(0,3 % 8),(1,5 % 8)])]
```

And here are the probabilities that the *next* marble is white:

```
*PRODEMO> probWhite (drawMarble $ urn (3,5))
3 % 8
*PRODEMO> probWhite (urn (3,5))
3 % 8
```

Thus, we find that the fact that a ball was withdrawn does not alter the probabilities.

Next, Uspensky asks us to suppose that the first ball that was withdrawn was white. What is the probability that the second ball that is withdrawn is also white? Here is the answer:

```
*PRODEMO> probWhite (drawMarble $ drawWhite $ urn (3,5))
2 % 7
```

As mentioned in Chapter 4, urns can also be represented as stacks of marbles. Let's redo the implementations for this alternative representation.

There are $\binom{m+n}{m}$ different marbles sequences with m white and n black marbles.

Here is a generator for lists containing m white and n black marbles:

```
distMarbles3 :: Int -> Int -> [[Marble]]
distMarbles3 0 0 = [[]]
distMarbles3 m 0 = [take m (repeat W)]
distMarbles3 0 n = [take n (repeat B)]
distMarbles3 m n = map (W:) (distMarbles3 (m-1) n)
                 ++ map (B:) (distMarbles3 m (n-1))
```

Use this for an initialisation of urn models of type `EM Marble`.

```
initU3 :: [Agent] -> (Int,Int) -> EM Marble
initU3 ags (m,n) =  (Mo ags worlds val accs points)
  where
    k       = binom (m+n) m
    worlds  = [0..k-1]
    urns    = distMarbles3 m n
    val     = zip worlds urns
    accs    = [ (ag,[worlds]) | ag  <- ags   ]
    points  = worlds
```

Lift this to a probabilistic model:

```
urn2 :: (Int,Int) -> Pem Marble
urn2 (m,n) = liftEM (initU3 [a,b] (m,n))
```

Use `mapPEM` for putting a white marble into an urn.

```
putWhite2 :: Pem Marble -> Pem Marble
putWhite2 = mapPEM (\ xs -> W:xs)
```

Use `mapPEM` for putting a black marble into an urn.

```
putBlack2 :: Pem Marble -> Pem Marble
putBlack2 = mapPEM (\ xs -> B:xs)
```

Putting a marble in an urn, with indifference between white and black.

```
putMarble2 :: Pem Marble -> Pem Marble
putMarble2 = gmapPEM putM2
```

```
putM2 ::  [Marble] -> Lottery [Marble]
putM2  xs = [(W:xs,1),(B:xs,1)]
```

Removing an object from a list, in all possible ways:

```
remove :: Eq a => a -> [a] -> [[a]]
remove x [] = []
remove x (y:ys) = if x == y then
                    ys : map (y:) (remove x ys)
                  else map (y:) (remove x ys)
```

Use this for drawing a marble.

```
drawMarble2 ::  Pem Marble -> Pem Marble
drawMarble2 = gmapPEM (\xs ->
  zip (remove W xs ++ remove B xs) (repeat 1))
```

Drawing a white marble:

```
drawWhite2 :: Pem Marble -> Pem Marble
drawWhite2 = gmapPEM
  (\ xs ->  zip (remove W xs) (repeat 1))
```

Drawing a black marble:

```
drawBlack2 :: Pem Marble -> Pem Marble
drawBlack2 = gmapPEM
    (\ xs ->  zip (remove B xs) (repeat 1))
```

The Lewis Carroll puzzle once more:

```
carroll2 = drawWhite2 $ putWhite2 $ putMarble2 $ urn2 (0,0)
```

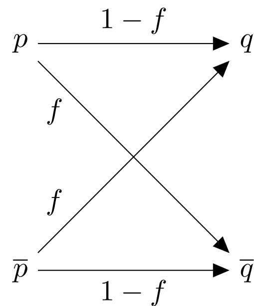
This gives:

```
*PRODEMO> carroll2
MO [a,b] [0,1,2] [(0,[W]),(1,[W]),(2,[B])]
[(a,[[0,1,2]]),(b,[[0,1,2]])] [0,1,2]
[(a,[(0,1 % 3),(1,1 % 3),(2,1 % 3)]),
 (b,[(0,1 % 3),(1,1 % 3),(2,1 % 3)])]
```

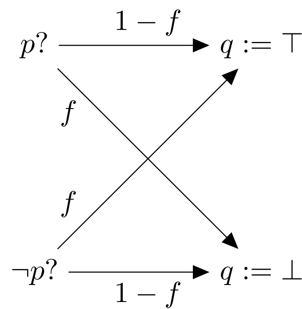
Chapter 14

Noisy Data Transfer

Noisy data transfer can be viewed as an update that transfers a bit $p = 1$ with probability $1 - f$ to $q = 1$ and with probability f to $q = 0$, and vice versa. In a picture:



This bit transfer update with noise f can be implemented as follows:



Here is the implementation:

```
bitTransfer :: Prob -> FUM Prp
bitTransfer = \ noise  ags -> UM
 ags
 [0,1,2,3]
 [(0,(p,[(Q 0,Top)])),(1,(p,[(Q 0,Ng Top)])),
  (2,(Ng p,[(Q 0,Ng Top)])),(3,(Ng p,[(Q 0,Top)]))]
 [ (x, [[0,1,2,3]]) | x <- ags ]
 [0,1,2,3]
 [ (x, [(0,1-noise), (1,noise), (2,1-noise), (3,noise)])
    | x <- ags ]
```

If we view the bit as a boolean, we can use `gmapPEM`, as follows:

```
bitTransf :: Prob -> Pem ([Bool],[Bool])
                  -> Pem ([Bool],[Bool])
bitTransf= \ noise  -> gmapPEM
  (\ [(x:xs,ys)] ->
     [ ([(xs,x:ys)], 1 - noise), ([(xs,not x:ys)], noise)])
```

Use this to transfer a sequence of bits. First initialize a bit sequence model without lotteries:

```
bitm :: [Agent] -> [Bool] -> EM ([Bool],[Bool])
bitm ags bits = Mo ags [0] val accs [0]
 where
 val = [(0,[(bits,[])])]
 accs = [ (a,[[0]]) | a <- ags ]
```

Introduce a lottery by means of an indifference lift:

```
bitM :: [Agent] -> [Bool] -> Pem ([Bool],[Bool])
bitM ags bits = liftEM (bitm ags bits)
```

This gives:

```
*PRODEMO> bitM [a] [True,True,False]
MO [a] [0] [(0,[([True,True,False],[])])]
[(a,[[0]])] [0] [(a,[(0,1 % 1)])]
```

Transfer these bits with a given amount of noise:

```
transfer :: Prob -> Pem ([Bool],[Bool])
                 -> Pem ([Bool],[Bool])
transfer noise m@(MO _ _ ((_,[([],_)]):_) _ _ _) = m
transfer noise m = let m1 = bitTransf noise m in
   transfer noise m1
```

Transfer, starting from an initial model with a single agent:

```
tr :: [Bool] -> Prob -> Pem ([Bool],[Bool])
tr bits noise = transfer noise (bitM [a] bits)
```

For transferring three bits, with noise level $\frac{1}{10}$, this gives the following result (bear in mind that the transferred list is in reverse order):

```
*PRODEMO> tr [True,True,False] (1/10)
MO [a] [0,1,2,3,4,5,6,7] [(0,[([],[False,True,True])]),
(1,[([],[True,True,True])]),(2,[([],[False,False,True])]),
(3,[([],[True,False,True])]),(4,[([],[False,True,False])]),
(5,[([],[True,True,False])]),(6,[([],[False,False,False])]),
(7,[([],[True,False,False])])] [(a,[[0,1,2,3,4,5,6,7]])]
[0,1,2,3,4,5,6,7]
```

```
[(a,[(0,729 % 1000),(1,81 % 1000),(2,81 % 1000),(3,9 % 1000),
     (4,81 % 1000),(5,9 % 1000),(6,9 % 1000),(7,1 % 1000)])]
```

Chapter 15

Bayesian Learning as Knowledge Growth

Suppose there are two coins, one of them fair, and the other with a bias of $\frac{2}{3}$ for heads. The agent doesn't know which coin she is holding, and tosses the coin a number of times. How can we represent this action?

We can use a recursive function that computes an epistemic model after n coin tosses from the epistemic model after $n-1$ tosses. The type of the epistemic model is `Pem Coin`, which means that its valuation assigns a coin list.

We will use the values `F` and `B` of datatype `Coin` to distinguish between the fair and the biased coin.

The function `repeatToss` creates the model that results from n coin tosses, with a coin that is either fair or not, so an appropriate type declaration is:

```
repeatToss :: Int -> Pem Coin
```

Initially (after 0 tosses) there are already two possibilities: the agent is tossing with the fair coin or with the biased coin, and she does not know which is which. Initially, let us assume she holds these possibilities for equally likely.

```
repeatToss 0 =
  MO [a] [0,1] [(0,[F]),(1,[U])] [(a,[[0,1]])] [0,1]
     [(a,[(0,1/2),(1,1/2)])]
```

The model that results after n tosses is computed from the model after $n-1$ tosses. The number of worlds doubles, and the lottery values get updated.

```
repeatToss n = let
    m@(MO [a] ws val rel points lots) =
       repeatToss (n-1)
    k = length ws
    wsh = ws
    wst = map (+k) ws
    ws1 = wsh ++ wst
    val1 = map (\ (w,x:xs) -> (w,x:H:xs)) val
            ++
           map (\ (w,x:xs) -> (w+k,x:T:xs)) val
    rel1 = [(a,[ws1])]
    points1 = ws1
    f       = \ (w,p) -> (w,1/2 * p)
    g       = \ (w,p) -> (w,2/3 * p)
    h       = \ (w,p) -> (w+k,1/2 * p)
    j       = \ (w,p) -> (w+k,1/3 * p)
    lots1 = [(a,
             amap f g (lot a m)
             ++
             amap h j (lot a m))]
  in
    MO [a] ws1 val1 rel1 points1 lots1
```

This uses an auxiliary function for alternate mapping of f and g over a list:

```
amap :: (a -> b) -> (a -> b) -> [a] -> [b]
amap f g [] = []
amap f g (x:xs) = f x : amap g f xs
```

This gives, e.g.:

```
*PRODEMO> repeatToss 2
MO [a] [0,1,2,3,4,5,6,7] [(0,[F,H,H]),(1,[U,H,H]),(2,[F,H,T]),
(3,[U,H,T]),(4,[F,T,H]),(5,[U,T,H]),(6,[F,T,T]),(7,[U,T,T])]
[(a,[[0,1,2,3,4,5,6,7]])] [0,1,2,3,4,5,6,7]
[(a,[(0,1 % 8),(1,2 % 9),(2,1 % 8),(3,1 % 9),(4,1 % 8),
     (5,1 % 9),(6,1 % 8),(7,1 % 18)])]
```

To talk about coin lists, we can use info formulas.

```
fair :: Form Coin
fair = Info (Inf (\xs -> head xs == F))

value :: [Coin] -> Form Coin
value xs = Info (Inf (\ys -> tail ys == xs))
```

```
*PRODEMO> eval (repeatToss 0) 0 (Cprb a fair (value []))
1 % 2
*PRODEMO> eval (repeatToss 2) 0 (Cprb a fair (value [H,H]))
9 % 25
```

Thus, after two tosses of heads, the probability of the hypothesis that the coin was fair has gone down from $\frac{1}{2}$ to $\frac{9}{25}$.

More subtly:

```
*PRODEMO> eval (repeatToss 6) 0 (Cprb a fair
    (Info (Inf (\ xs -> length (filter (\ x -> x == H) xs) > 3))))
8019 % 23891
```

```
*PRODEMO> eval (repeatToss 6) 0 (Cprb a (Ng fair)
  (Info (Inf (\ xs -> length (filter (\x -> x == H) xs) > 3))))
15872 % 23891
```

Thus, if after six tosses more than half of the outcomes are heads, then it is almost twice as probable that the biased coin rather than the fair coin was used.

These examples can be viewed as instances of Bayesian learning, where the lotteries appear as the hypotheses, and where the computed probabilities represent knowledge rather than belief.

It is also easy to construct models for a given sequence of tosses. An empty sequence has two worlds, for the distinction between the fair and the biased coin. Again, we assume that these possibilities are initially equally likely. Bayesians call this the prior.

```
genSequence :: [Coin] -> Pem Coin
genSequence [] =
  MO [a] [0,1] [(0,[F]),(1,[U])] [(a,[[0,1]])] [0,1]
     [(a,[(0,1/2),(1,1/2)])]
```

If the toss is heads, then compute the new probabilities for the two cases.

```
genSequence (H:xs) = let
   m@(MO [a] ws val rel points lots) = genSequence xs
   val1 =  [(0,F:H:xs),(1,U:H:xs)]
   p    =  table2fct (lot a m)
   lots1 = [(a, [(0,p 0 * 1/2),(1, p 1 * 2/3)])]
 in
   MO [a] ws val1 rel points lots1
```

If the toss is tails, then compute the new probabilities for the two cases.

```
genSequence (T:xs) = let
   m@(MO [a] ws val rel points lots) = genSequence xs
   val1 =  [(0,F:T:xs),(1,U:T:xs)]
   p    =  table2fct (lot a m)
   lots1 = [(a, [(0,p 0 * 1/2),(1, p 1 * 1/3)])]
 in
   MO [a] ws val1 rel points lots1
```

This gives, e.g.:

```
*PRODEMO> genSequence [H,H,T]
MO [a] [0,1] [(0,[F,H,H,T]),(1,[U,H,H,T])]
[(a,[[0,1]])] [0,1] [(a,[(0,1 % 16),(1,2 % 27)])]
```

Two heads and one tails increases the probability that the coin has a bias for heads:

```
*PRODEMO> prob (genSequence [H,H,T]) a 0 fair
27 % 59
*PRODEMO> prob (genSequence [H,H,T]) a 0 (Ng fair)
32 % 59
```

Thus, the system allows us to compute the probability of a hypothesis.

Bibliography

[BGK09] J. van Benthem, J. Gerbrandy, and B. Kooi. Dynamic update with probabilities. *Studia Logica*, 93:67–96, 2009.

[BM04] A. Baltag and L.S. Moss. Logics for epistemic programs. *Synthese*, 139(2):165–224, 2004.

[BMS98] A. Baltag, L.S. Moss, and S. Solecki. The logic of public announcements, common knowledge, and private suspicions. In I. Bilboa, editor, *Proceedings of TARK'98*, pages 43–56, 1998.

[BvEK06] J. van Benthem, J. van Eijck, and B. Kooi. Logics of communication and change. *Information and Computation*, 204(11):1620–1662, 2006.

[Ell61] Daniel Ellsberg. Risk, ambiguity, and the Savage axioms. *Quarterly Journal of Economics*, 75(4):643–669, 1961.

[FH94] R. Fagin and J.Y. Halpern. Reasoning about knowledge and probability. *Journal of the ACM*, pages 340–367, 1994.

[Gar81] Martin Gartner. *Mathematical Circus*. Vintage, 1981.

[Gne75] B.V Gnedenko. *The Theory of Probability*. Mir Publishers, Moscow, 1975.

[Hac75] Ian Hacking. *The Emergence of Probability — A Philosophical Study of Early Ideas about Probability, Induction and Statistical Inference*. Cambridge University Press, 1975.

[Hal03] J. Halpern. *Reasoning About Uncertainty*. MIT Press, 2003.

[Jay03] E.T. Jaynes. *Probability Theory: The Logic of Science*. Cambridge University Press, 2003.

[Jon03] S. Peyton Jones, editor. *Haskell 98 Language and Libraries; The Revised Report*. Cambridge University Press, 2003.

[Key63] J.M. Keynes. *A Treatise on Probability*. Macmillan, London, 1963.

[Knu92] D.E. Knuth. *Literate Programming*. CSLI Lecture Notes, no. 27. CSLI, Stanford, 1992.

[Koo03a] Barteld P. Kooi. *Knowledge, Chance, and Change*. PhD thesis, Groningen University, 2003.

[Koo03b] Barteld P. Kooi. Probabilistic dynamic epistemic logic. *Journal of Logic, Language and Information*, 12(4):381–408, 2003.

[Kör08] T.W. Körner. *Naive Decision Making: Mathematics Applied to the Social World*. Cambridge University Press, 2008.

[Lap14] M. le Comte Laplace. *Essai Philosophique sur les Probabilités*. Courcier, Paris, 1814.

[Par94] Jeff B. Paris. *The uncertain reasoner's companion — a mathematical perspective*, volume 39 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1994.

[Sav72] Leonard J. Savage. *The Foundations of Statistics — Second Revised Edition*. Dover, New York, 1972.

[Usp37] J.V. Uspensky. *Introduction to Mathematical Probability*. McGraw-Hill, 1937.

[VS90] M. Vos Savant. Ask Marilyn. *Parade Magazine*, page 15, 1990.