Dynamic Epistemic Model Checking

Jan van Eijck
CWI & ILLC, Amsterdam

Tsinghua University, November 4, 2015

Abstract

This lecture introduces and discusses a tiny program for epistemic model checking with S5 models in Haskell. The model update operations are public announcement and publicly observable factual change. The implementation is much more efficient than the earlier implementation of DEMO (Van Eijck 2007), but less efficient than a symbolic model checker for DEL (Lecture 3). Still, because the approach is so simple, it gives a useful idea of what goes on in model checking DEL. As examples, we implement the sum and product riddle, which is solved in a few seconds, and the parametrized muddy children problems, where the cases with up to ten children run in a matter of seconds. Next, we look at PRODEMO, a program for probabilistic epistemic model checking, and use it for solving some probabilistic epistemic model checking problems.

Abstract

This lecture introduces and discusses a tiny program for epistemic model checking with S5 models in Haskell. The model update operations are public announcement and publicly observable factual change. The implementation is much more efficient than the earlier implementation of DEMO (Van Eijck 2007), but less efficient than a symbolic model checker for DEL (Lecture 3). Still, because the approach is so simple, it gives a useful idea of what goes on in model checking DEL. As examples, we implement the sum and product riddle, which is solved in a few seconds, and the parametrized muddy children problems, where the cases with up to ten children run in a matter of seconds. Next, we look at PRODEMO, a program for probabilistic epistemic model checking, and use it for solving some probabilistic epistemic model checking problems.

Update of the abstract: the slides give a full implementation of public announcement updates for S5 models and for S5 weight models.

Who in Modal and Epistemic Logic?

Who in Modal and Epistemic Logic?

Saul Kripke (born 1940)

Jaakko Hintikka (1929–2015)

```
module DEMO

where

import Data.List
```

Epistemic Logic

Equivalence relations as partitions:

A partition β of a set X is a family of subsets of X with the following properties:

1. $\bigcup \beta = X$,
2. $Y \in \beta$ implies $Y \neq \emptyset$,
3. $Y, Z \in \beta \wedge Y \neq Z$ implies $Y \cap Z = \emptyset$.

Partitions of X correspond to equivalence relations on X in the following precise sense:

- If $\sim$ is an equivalence on X, then $\{[x]_\sim \mid x \in X\}$ is the corresponding partition.
- If β is a partition of X, then $\sim_\beta$ given by $x \sim_\beta y$ iff $\exists Y \in \beta$ such that $\{x, y\} \subseteq Y$ is the corresponding equivalence relation on X.

Building epistemic models from partitions

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

The block of an element 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 :: Ord a => [a] -> Erel a -> Erel a
restrict domain =  nub . filter (/= [])
                   . map (filter (flip elem domain))
```

Agents

```
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
```

Agents

```
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

```
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
```

```
p,q,r,s :: Prp
p = P 0; q = Q 0; r = R 0; s = S 0
```

Epistemic models

```
data EpistM state = Mo
             [state]
             [Agent]
             [(state,[Prp])]
             [(Agent,Erel state)]
             [state]  deriving (Eq,Show)
```

```
example1 :: EpistM Int
example1 = Mo
 [0..3]
 [a,b,c]
 []
 [(a,[[0],[1],[2],[3]]),
  (b,[[0],[1],[2],[3]]),(c,[[0..3]])]
 [1]
```

```
example2 :: EpistM Int
example2 = Mo
 [0..3]
 [a,b,c]
 [(0,[p,q]),(1,[p]),(2,[q]),(3,[])]
 [(a,[[0..3]]),(b,[[0..3]]),(c,[[0..3]])]
 [0..3]
```

Extracting an epistemic relation from a model

```
rel :: Agent -> EpistM a -> Erel a
rel ag (Mo _ _ _ rels _) = apply rels ag
```

Extracting an epistemic relation from a model

```
rel :: Agent -> EpistM a -> Erel a
rel ag (Mo _ _ _ rels _) = apply rels ag
```

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

Epistemic Formulas

```
data Frm a = Tp
           | Info a
           | Prp Prp
           | N (Frm a)
           | C [Frm a]
           | D [Frm a]
           | Kn Agent (Frm a)
          deriving (Eq,Ord,Show)
```

A useful abbreviation:

```
impl :: Frm a -> Frm a -> Frm a
impl form1 form2 = D [N form1, form2]
```

Truth Definition

...

```
isTrueAt :: Ord state =>
            EpistM state -> state -> Frm state -> Bool
isTrueAt m w Tp = True
isTrueAt m w (Info x) = w == x
isTrueAt
  m@(Mo worlds agents val acc points) w (Prp p) = let
   props = apply val w
  in
   elem p props
isTrueAt m w (N f) = not (isTrueAt m w f)
isTrueAt m w (C fs) = and (map (isTrueAt m w) fs)
isTrueAt m w (D fs) = or  (map (isTrueAt m w) fs)
isTrueAt
 m@(Mo worlds agents val acc points) w (Kn ag f) = let
    r = rel ag m
    b = bl r w
  in
    and (map (flip (isTrueAt m) f) b)
```

Public Announcement

Restriction to φ worlds:

```
upd_pa :: Ord state =>
          EpistM state -> Frm state -> EpistM state
upd_pa m@(Mo states agents val rels actual) f =
  (Mo sts' agents val' rels' actual')
   where
   sts'    = [ s | s <- states, isTrueAt m s f ]
   val'    = [ (s, ps) | (s,ps) <- val, s `elem` sts']
   rels'   = [(ag,restrict sts' r) | (ag,r) <- rels ]
   actual'= [ s | s <- actual, s `elem` sts' ]
```

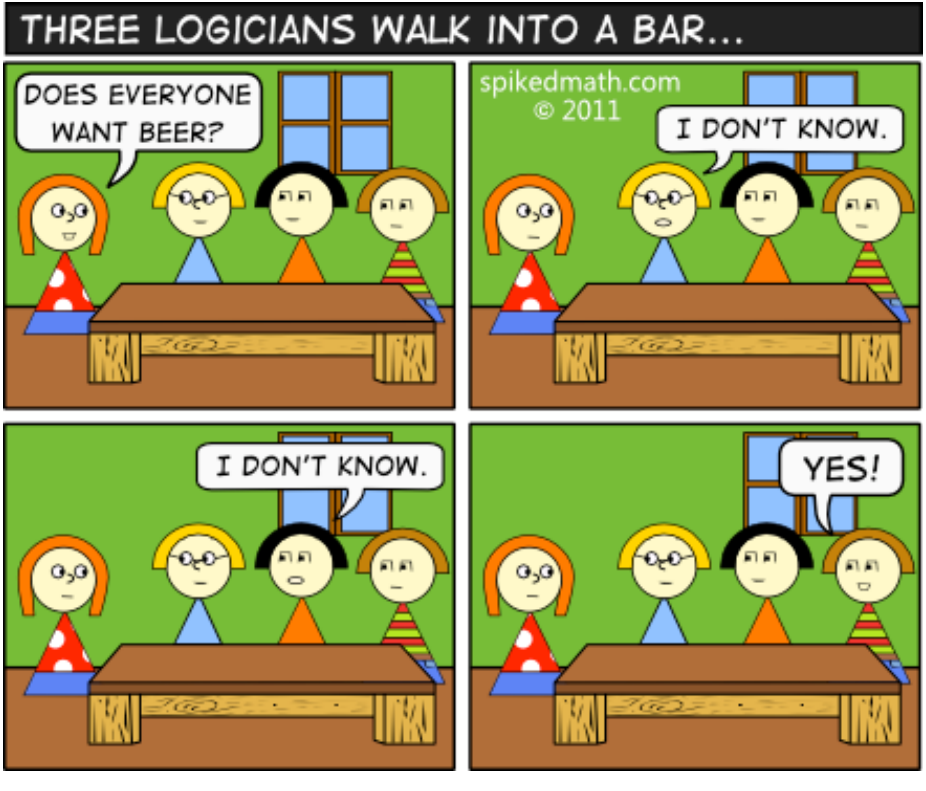
```
upds_pa ::  Ord state =>
         EpistM state -> [Frm state] -> EpistM state
upds_pa = foldl upd_pa
```

```
upds_pa ::  Ord state =>
         EpistM state -> [Frm state] -> EpistM state
upds_pa = foldl upd_pa
```

To understand this we have to understand `foldl`. Here is a home-made version:

```
foldl :: (a -> b -> a) -> a -> [b] -> a
foldl f z [] = z
foldl f z (x:xs) = foldl f (f z x) xs
```

Three Logicians



```
bools = [True,False]
```

Initialize the bar situation: they all know what they want but are ignorant about what the others want.

```
initBar :: EpistM (Bool,Bool,Bool)
initBar = Mo states [a,b,c] [] rels [(True,True,True)]
  where
  states = [ (b1,b2,b3) | b1 <- bools,
                          b2 <- bools,
                          b3 <- bools ]
  rela = (a,[[(True,x,y)  | x <- bools, y <- bools],
             [(False,x,y) | x <- bools, y <- bools]])
  relb = (b,[[(x,True,y)  | x <- bools, y <- bools],
             [(x,False,y) | x <- bools, y <- bools]])
  relc = (c,[[(x,y,True)  | x <- bools, y <- bools],
             [(x,y,False) | x <- bools, y <- bools]])
  rels = [rela,relb,relc]
```

Statements of ignorance and knowledge:

```
allBeer :: Frm (Bool,Bool,Bool)
allBeer = Info (True,True,True)

ignA, ignB, ignC :: Frm  (Bool,Bool,Bool)
ignA = C [N (Kn a allBeer), N (Kn a (N allBeer))]
ignB = C [N (Kn b allBeer), N (Kn b (N allBeer))]
ignC = C [N (Kn c allBeer), N (Kn c (N allBeer))]

knowC, knowC' :: Frm  (Bool,Bool,Bool)
knowC  = Kn c allBeer
knowC' = Kn c (N allBeer)
```

Finally, Updating

```
barModel1 = upd_pa initBar ignA
```

Result of second update:

```
barModel2 = upd_pa barModel1 ignB
```

Result of third update:

```
barModel3 = upd_pa barModel2 knowC
```

Or the third logician could have said ‘no’, for ‘I know that not all of us want beer’.

```
barModel3' = upd_pa barModel2 knowC'
```

Example: Sum and Product (Hans Freudenthal)

A says to S and P: I have chosen two integers x, y such that $1 < x < y$ and $x + y \leq 100$. In a moment, I will inform S only of $s = x + y$, and P only of $p = xy$. These announcements remain private. You are required to determine the pair (x, y). He acts as said. The following conversation now takes place:

Example: Sum and Product (Hans Freudenthal)

A says to S and P: I have chosen two integers x, y such that $1 < x < y$ and $x + y \leq 100$. In a moment, I will inform S only of $s = x + y$, and P only of $p = xy$. These announcements remain private. You are required to determine the pair (x, y). He acts as said. The following conversation now takes place:

1. P says: "I do not know the pair."

Example: Sum and Product (Hans Freudenthal)

A says to S and P: I have chosen two integers x, y such that $1 < x < y$ and $x + y \leq 100$. In a moment, I will inform S only of $s = x + y$, and P only of $p = xy$. These announcements remain private. You are required to determine the pair (x, y). He acts as said. The following conversation now takes place:

1. P says: "I do not know the pair."
2. S says: "I knew you didn't."

Example: Sum and Product (Hans Freudenthal)

A says to S and P: I have chosen two integers x, y such that $1 < x < y$ and $x + y \leq 100$. In a moment, I will inform S only of $s = x + y$, and P only of $p = xy$. These announcements remain private. You are required to determine the pair (x, y). He acts as said. The following conversation now takes place:

1. P says: "I do not know the pair."
2. S says: "I knew you didn't."
3. P says: "I now know it."

Example: Sum and Product (Hans Freudenthal)

A says to S and P: I have chosen two integers x, y such that $1 < x < y$ and $x + y \leq 100$. In a moment, I will inform S only of $s = x + y$, and P only of $p = xy$. These announcements remain private. You are required to determine the pair (x, y). He acts as said. The following conversation now takes place:

1. P says: "I do not know the pair."
2. S says: "I knew you didn't."
3. P says: "I now know it."
4. S says: "I now also know it."

Determine the pair (x, y).

A model checking solution with DEMO [vE05, vE07] (based on a DEMO program written by Ji Ruan) was presented in [DRV05]. An optimized version of that solution is in [vE13].

The list of candidate pairs:

```
pairs :: [(Int,Int)]
pairs  = [ (x,y) |  x <- [2..100], y <- [2..100],
                    x < y, x+y <= 100 ]
```

The initial epistemic model is such that a (representing S) cannot distinguish number pairs with the same sum, and b (representing P) cannot distinguish number pairs with the same product. Instead of using a valuation, we use number pairs as worlds.

```
msnp :: EpistM (Int,Int)
msnp = (Mo pairs [a,b] [] acc pairs)
  where
  acc   = [ (a, [ [ (x1,y1) | (x1,y1) <- pairs,
                              x1+y1 == x2+y2 ] |
                                (x2,y2) <- pairs ] ) ]
          ++
          [ (b, [ [ (x1,y1) | (x1,y1) <- pairs,
                              x1*y1 == x2*y2 ] |
                                (x2,y2) <- pairs ] ) ]
```

The statement by b that he does not know the pair:

```
statement_1 =
  C [ N (Kn b (Info p)) | p <- pairs ]
```

To check this statement is expensive. A computationally cheaper equivalent statement is the following (see [DRV05]).

```
statement_1e =
  C [ Info p `impl` N (Kn b (Info p)) | p <- pairs ]
```

In Freudenthal's story, the first public announcement is the statement where b confesses his ignorance, and the second public announcement is the statement by a about her knowledge about b's state of knowledge before that confession. We can wrap the two together in a single statement to the effect that initially, a knows that b does not know the pair. This gives:

```
k_a_statement_1e = Kn a statement_1e
```

The second announcement proclaims the statement by b that now he knows:

```
statement_2 =
  D [ Kn b (Info p) | p <- pairs ]
```

Equivalently, but computationally more efficient:

```
statement_2e =
  C [ Info p `impl` Kn b (Info p) | p <- pairs ]
```

The final announcement concerns the statement by a that now she knows as well.

```
statement_3 =
  D [ Kn a (Info p) | p <- pairs ]
```

In the computationally optimized version:

```
statement_3e =
  C [ Info p `impl` Kn a (Info p) | p <- pairs ]
```

The solution:

```
solution = upds_pa msnp
     [k_a_statement_1e,statement_2e,statement_3e]
```

This is checked in a matter of seconds.

Make sure you load DEMO with `ghci -fobject-code DEMO`.

```
statement_3e =
  C [ Info p `impl` Kn a (Info p) | p <- pairs ]
```

The solution:

```
solution = upds_pa msnp
     [k_a_statement_1e,statement_2e,statement_3e]
```

This is checked in a matter of seconds.

Make sure you load DEMO with `ghci -fobject-code DEMO`.

```
*DEMO> solution
Mo [(4,13)] [a,b] [(a,[[(4,13)]]),(b,[[(4,13)]])]
   [(4,13)]
```

Epistemic Weight Models

An **Epistemic Weight Model** $\mathcal{M}$ is a tuple (W, R, V, L), where

Epistemic Weight Models

An **Epistemic Weight Model** $\mathcal{M}$ is a tuple (W, R, V, L), where

- W is a non-empty set of worlds.

Epistemic Weight Models

An **Epistemic Weight Model** $\mathcal{M}$ is a tuple (W, R, V, L), where

- W is a non-empty set of worlds.
- R is a function that assigns to every agent $i \in A$ an equivalence relation $\sim_i$ on W.

Epistemic Weight Models

An **Epistemic Weight Model** $\mathcal{M}$ is a tuple (W, R, V, L), where

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

Epistemic Weight Models

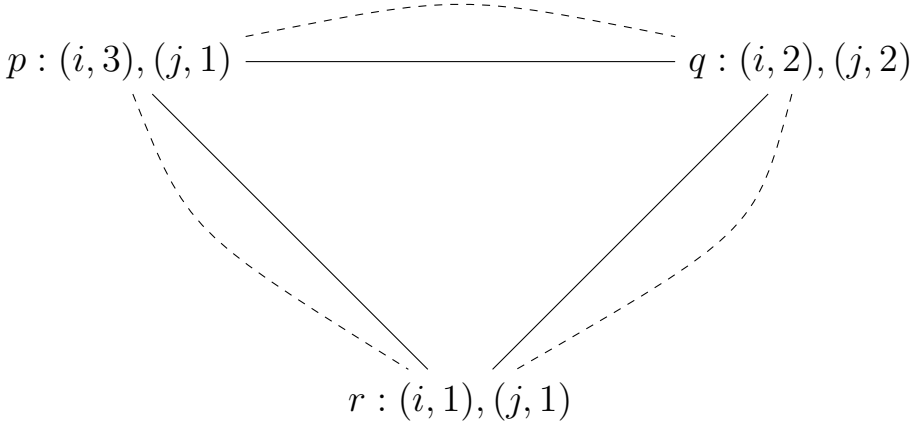
An **Epistemic Weight Model** $\mathcal{M}$ is a tuple (W, R, V, L), where

- W is a non-empty set of worlds.
- R is a function that assigns to every agent $i \in A$ an equivalence relation $\sim_i$ on W.
- V is a valuation function that assigns to every $w \in W$ a subset of $\mathbf{P}$.
- L is a function that assigns to every agent $i \in Ag$ a weight $\mathbb{L}_i$, where $\mathbb{L}_i$ is a function from W to $\mathbb{Q}^+$, the set of positive rationals, with the constraint that for each $w \in W$,
$$\mathbb{L}_i([w]_i) < \infty.$$

Example: Willingness to Bet in Investment Banking

Two bankers i, j consider buying stocks in three firms p, q, r that are involved in a takeover bid. There are three possible outcomes: p for "p wins", q for " q wins", and r for "r wins." i takes the winning chances to be $3:2:1$, j takes them to be $1:2:1$.

i: solid lines, j: dashed lines.



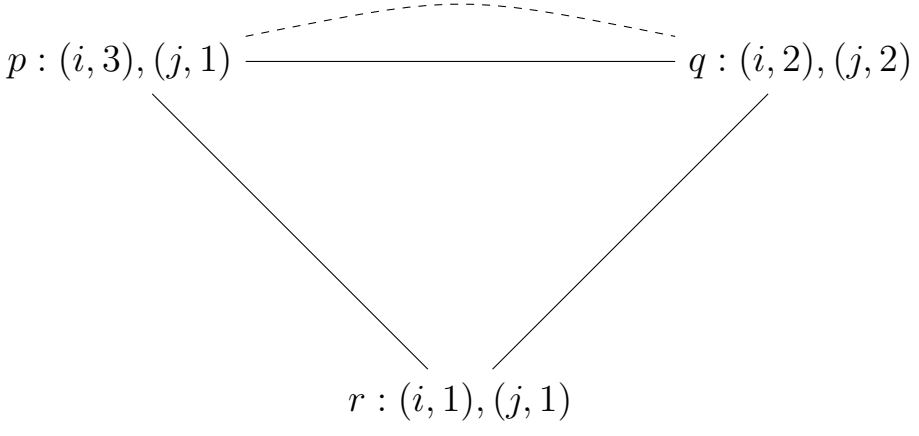
Belief as Willingness to Bet

We see that i is willing to bet $1:1$ on a, while j is willing to bet $3:1$ against p.

It follows that in this model i and j have an opportunity to gamble, for, to put it in Bayesian jargon, they do not have a common prior.

Foreknowledge in Investment Banking

Suppose j has foreknowledge about what firm r will do.



The probabilities assigned by i remain as before. The probabilities assigned by j have changed, as follows. In worlds p and q, j assigns probability $\frac{1}{3}$ to p and $\frac{2}{3}$ to q. In world r, j is sure of r.

Implementing Epistemic Weight Models

```
data EpistWM state = WMo
             [state]
             [Agent]
             [(state,[Prp])]
             [(Agent,Erel state)]
             [(Agent,[(state,Rational)])]
             [state]
      deriving (Eq,Show)
```

Extracting an epistemic relation from a weight model

```
wrel :: Agent -> EpistWM a -> Erel a
wrel ag (WMo _ _ _ rels _ _) = apply rels ag
```

Extracting an epistemic relation from a weight model

```
wrel :: Agent -> EpistWM a -> Erel a
wrel ag (WMo _ _ _ rels _ _) = apply rels ag
```

Restriction of a table to a domain

```
restrictT :: Eq a => [a] -> [(a,b)] -> [(a,b)]
restrictT xs t = [ (x,y) | (x,y) <- t, elem x xs ]
```

We will need this for adjusting weight tables later on.

Example Model

```
invest :: EpistWM Int
invest = WMo
 [0..2]
 [a,b]
 [(0,[p]),(1,[q]),(2,[r])]
 [(a,[[0,1,2]]),(b,[[0,1,2]])]
 [(a,[(0,3),(1,2),(2,1)]),
  (b,[(0,1),(1,2),(2,1)])]
 [0]
```

Language

We use notation borrowed from [DR15].

$$
\begin{aligned}
\varphi &::= \top \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \Phi \leq_i \Phi \\
\Phi &::= \varphi \mid \varphi \oplus \Phi
\end{aligned}
$$

Language

We use notation borrowed from [DR15].

$$\begin{array}{rcl} \varphi & ::= & \top \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \Phi \leq_i \Phi \\ \Phi & ::= & \varphi \mid \varphi \oplus \Phi \end{array}$$

Abbreviations:

As usual for $\bot, \vee, \rightarrow, \leftrightarrow$.

$\Phi <_i \Psi$ for $\Phi \leq_i \Psi \wedge \neg\Psi \leq_i \Phi$.

$\Phi =_i \Psi$ for $\Phi \leq_i \Psi \wedge \Psi \leq_i \Phi$.

$B_i\varphi$ for $(\neg\varphi) <_i \varphi$, $\check{B}_i\varphi$ for $(\neg\varphi) \leq_i \varphi$. "Belief as willingness to bet"

$K_i\varphi$ for $\top \leq_i \varphi$, $\check{K}_i\varphi$ for $\bot <_i \varphi$. "Knowledge as certainty"

Semantics for this language

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

$$
\begin{aligned}
[\![\varphi]\!]_{\mathcal{M}} &:= \{w \in W \mid \mathcal{M}, w \models \varphi\} \\
[\![\varphi]\!]_{\mathcal{M}}^{w,i} &:= [\![\varphi]\!]_{\mathcal{M}} \cap [w]_i \\
\mathbb{L}_{w,i}\varphi &:= \sum_{u \in [\![\varphi]\!]_{\mathcal{M}}^{w,i}} \mathbb{L}_i(u)
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{M}, w \models \top & \quad \text{always} \\
\mathcal{M}, w \models p & \quad \text{iff} \quad p \in V(w) \\
\mathcal{M}, w \models \neg\varphi & \quad \text{iff} \quad \text{not } \mathcal{M}, w \models \varphi \\
\mathcal{M}, w \models \varphi_1 \wedge \varphi_2 & \quad \text{iff} \quad \mathcal{M}, w \models \varphi_1 \text{ and } \mathcal{M}, w \models \varphi_2 \\
\mathcal{M}, w \models \Phi \leq_i \Psi & \quad \text{iff} \quad \sum_{\varphi \in \Phi} \mathbb{L}_{w,i}\varphi \leq \sum_{\psi \in \Psi} \mathbb{L}_{w,i}\psi
\end{aligned}
$$

Language Implemented

```
data Form a = Top
            | INFO a
            | Pr Prp
            | Ng (Form a)
            | Cj [Form a]
            | Dj [Form a]
            | Less Agent [Form a] [Form a]
            | Leq  Agent [Form a] [Form a]
           deriving (Eq,Ord,Show)
```

Abbreviations

Knowledge as certainty.

```
kb, kd :: Agent -> Form a -> Form a
kb i f = Leq  i [Top] [f]
kd i f = Less i [Ng Top] [f]
```

Abbreviations

Knowledge as certainty.

```
kb, kd :: Agent -> Form a -> Form a
kb i f = Leq  i [Top] [f]
kd i f = Less i [Ng Top] [f]
```

Belief as willingness to bet.

```
bb, bd ::  Agent -> Form a -> Form a
bb i f = Less i [Ng f] [f]
bd i f = Leq i [Ng f] [f]
```

Extracting a probability function from a model

The probability of w for i in $\mathcal{M}$ is given by the weight that i assigns to w, divided by the sum of the weights assigned by i to the members of $[w]_i$ (the i-accessible block). That is:

$$P_i(w) = \frac{\mathbb{L}_i w}{\mathbb{L}_i [w]_i}.$$

```
prob :: Eq a => Agent -> EpistWM a -> a -> Rational
prob ag (WMo _ _ _ rels weight _) w = let
   r     = apply rels ag
   block = bl r w
   table = apply weight ag
   f     = apply table
 in
   f w / sum (map f block)
```

Truth

```
isTrue :: Ord state =>
          EpistWM state -> state -> Form state -> Bool
isTrue m w Top = True
isTrue m w (INFO x) = w == x
isTrue m@(WMo worlds agents val acc weight points)
  w (Pr p) =
  let
   props = apply val w
  in
   elem p props
isTrue m w (Ng f) = not (isTrue m w f)
isTrue m w (Cj fs) = and (map (isTrue m w) fs)
isTrue m w (Dj fs) = or  (map (isTrue m w) fs)
```

```
isTrue
   m@(WMo worlds agents val acc weight points)
   w (Less i fs1 fs2) =
 let
   r = wrel i m
   b = bl r w
   s1 = sum [ prob i m u | u <- b, f <- fs1,
              isTrue m u f ]
   s2 = sum [ prob i m u | u <- b, f <- fs2,
              isTrue m u f ]
 in
   s1 < s2
```

```
isTrue
   m@(WMo worlds agents val acc weight points)
   w (Leq i fs1 fs2) =
 let
   r = wrel i m
   b = bl r w
   s1 = sum [ prob i m u | u <- b, f <- fs1,
                           isTrue m u f ]
   s2 = sum [ prob i m u | u <- b, f <- fs2,
                           isTrue m u f ]
 in
   s1 <= s2
```

Subjective Probabilities

```
sprob :: Ord state =>
  EpistWM state -> state -> Agent -> Form state -> Rational
sprob m w i f = let
   r = wrel i m
   b = bl r w
   s1 = sum [ prob i m u | u <- b, isTrue m u f ]
   s2 = sum [ prob i m u | u <- b ]
 in
   s1 / s2
```

Public Announcement for Weight Models

Restriction to φ worlds:

```
upd_wpa :: Ord state =>
   EpistWM state -> Form state -> EpistWM state
upd_wpa m@(WMo sts ags val rels wght actual) f =
  (WMo sts' ags val' rels' wght' actual')
   where
   sts'    = [ s | s <- sts, isTrue m s f ]
   val'    = [ (s, ps) | (s,ps) <- val, s `elem` sts']
   rels'   = [(ag,restrict sts' r) | (ag,r) <- rels ]
   wght' = [(ag,restrictT sts' t) | (ag,t) <- wght ]
   actual'= [ s | s <- actual, s `elem` sts' ]
```

```
upds_wpa ::  Ord state =>
  EpistWM state -> [Form state] -> EpistWM state
upds_wpa = foldl upd_wpa
```

Example: Public Announcement in Investment Banking

In model `invest`, neither of a, b believe that firm p will win:

```
*DEMO> isTrue invest 0 (bb a (Pr p))
False
*DEMO> isTrue invest 0 (bb b (Pr p))
False
```

Example: Public Announcement in Investment Banking

In model `invest`, neither of a, b believe that firm p will win:

```
*DEMO> isTrue invest 0 (bb a (Pr p))
False
*DEMO> isTrue invest 0 (bb b (Pr p))
False
```

Now let `invest'` be the result of the public announcement that not r.

```
invest' = upd_wpa invest (Ng (Pr r))
```

This changes the belief of a about p:

```
*DEMO> isTrue invest' 0 (bb a (Pr p))
True
*DEMO> isTrue invest' 0 (bb b (Pr p))
False
```

Example: Reasoning about Disease

You are from a population with a statistical chance of 1 in 100 of having disease D. The initial screening test for this has a false positive rate of 0.2 and a false negative rate of 0.1. You tested positive (T).

Should you believe you have disease D?

A Weight Model for the Disease Problem

Let's use p for the outcome of the test and q for having the disease:

```
test = Pr p
disease = Pr q
```

```
dmodel :: EpistWM Int
dmodel = WMo
 [0..3]
 [a,b]
 [(0,[p,q]),(1,[q]),(2,[p]), (3,[])]
 [(a,[[0,1,2,3]]),(b,[[0,1,2,3]])]
 [(a,[(0,0.9),(1,0.1),(2,0.2*99),(3,0.8*99)]),
  (b,[(0,0.9),(1,0.1),(2,0.2),(3,0.8)])]
 [0..3]
```

Probability of having the disease

According to a:

```
*DEMO> sprob dmodel 0 a disease
1 % 100
```

According to b:

```
*DEMO> sprob dmodel 0 b disease
1 % 2
*DEMO> isTrue dmodel 0 (bb b disease)
False
*DEMO> isTrue dmodel 0 (bd b disease)
True
```

Update with the test result

```
dmodel' = upd_wpa dmodel test
```

```
*DEMO> sprob dmodel' 0 a disease
1 % 23
*DEMO> isTrue dmodel' 0 (bb a disease)
False
*DEMO> sprob dmodel' 0 b disease
9 % 11
*DEMO> isTrue dmodel' 0 (bb b disease)
True
```

Compare with Applying Bayes' Rule

Compare with Applying Bayes' Rule

$$P(D|T) = \frac{P(T|D)P(D)}{P(T)} = \frac{P(T|D)P(D)}{P(T|D)P(D) + P(T|\neg D)P(\neg D)}$$

Filling in $P(T|D) = 0.9, P(D) = 0.01, P(\neg D) = 0.99, P(T|\neg D) = 0.2$ gives $P(D|T) = \frac{1}{23}$.

Conclusion

Conclusion

- Representation of probabibility information by means of weight functions was designed with implementation of model checking in mind.

Conclusion

- Representation of probabibility information by means of weight functions was designed with implementation of model checking in mind.
- Just extend the epistemic models with a weight table for each agent.

Conclusion

- Representation of probabibility information by means of weight functions was designed with implementation of model checking in mind.
- Just extend the epistemic models with a weight table for each agent.
- Implementations of model checkers for these logics can be found in [Eij13] and in [San14] ...

Conclusion

- Representation of probabibility information by means of weight functions was designed with implementation of model checking in mind.
- Just extend the epistemic models with a weight table for each agent.
- Implementations of model checkers for these logics can be found in [Eij13] and in [San14] ...
- The implementations can deal with Monty Hall style puzzles, urn puzzles, Bayesian updating by drawing from urns or tossing (possibly biased) coins, and 'paradoxes' such as the puzzle of the three prisoners.

Conclusion

- Representation of probabibility information by means of weight functions was designed with implementation of model checking in mind.
- Just extend the epistemic models with a weight table for each agent.
- Implementations of model checkers for these logics can be found in [Eij13] and in [San14] ...
- The implementations can deal with Monty Hall style puzzles, urn puzzles, Bayesian updating by drawing from urns or tossing (possibly biased) coins, and 'paradoxes' such as the puzzle of the three prisoners.
- Efficiency was not a goal, but these implementation can be made very efficient using the techniques that Malvin will explain in the final lecture.

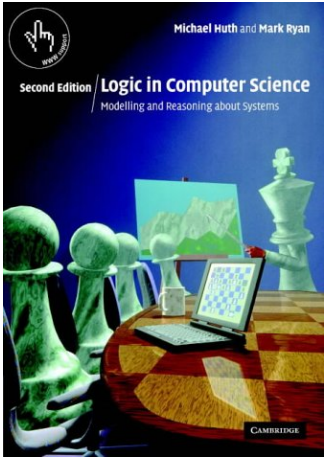
Links and Books for Further Study

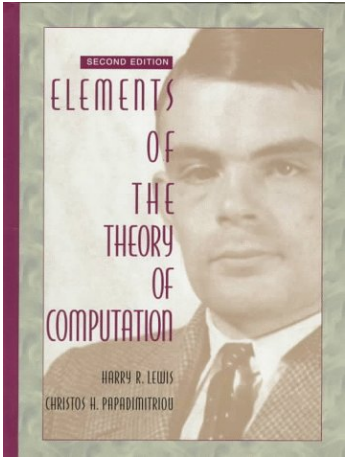
http://www.logicinaction.org

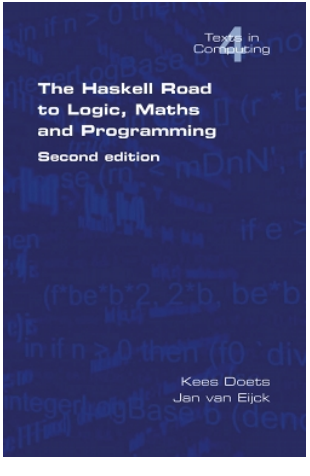
http://www.cwi.nl/~jve/HR

http://projecteuler.net

http://www.cwi.nl/~jve/software/demo_s5







References

[DR15] James P. Delgrande and Bryan Renne. The logic of qualitative probability. In Proceedings of the Twenty-Fourth International Conference on Artificial Intelligence (IJCAI 2015), pages 2904–2910, Buenos Aires, 2015.

[DRV05] Hans van Ditmarsch, Ji Ruan, and Rineke Verbrugge. Model checking sum and product. In Shichao Zhang and Ray Jarvis, editors, AI 2005: Advances in Artificial Intelligence: 18th Australian Joint Conference on Artificial Intelligence, volume 3809 of Lecture Notes in Computer Science, pages 790–795. Springer-Verlag GmbH, 2005.

[Eij13] Jan van Eijck. Learning about probability. available from `homepages.cwi.nl:~/jve/software/prodemo`, 2013.

[San14] Thomas Santoli. Haskell project epistemic logic. Technical report, ILLC, Summer 2014.

[vE05] Jan van Eijck. DEMO program and documentation, 2005. Available from `http://www.cwi.nl/~jve/demo/`.

[vE07] Jan van Eijck. DEMO — a demo of epistemic modelling. In Johan van Benthem, Dov Gabbay, and Benedikt Löwe, editors, Interactive Logic — Proceedings of the 7th Augustus de Morgan Workshop, number 1 in Texts in Logic and Games, pages 305–363. Amsterdam University Press, 2007.

[vE13] Jan van Eijck. DEMO-S5. Technical report, CWI, Amsterdam, 2013.