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 $\beta$ 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. $Y, Z \in \beta \land Y \neq Z$ 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 $\beta$ 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

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

The block of an element in a partition:

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

The restriction of a partition to a domain:

```haskell
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

```haskell
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

```haskell
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 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 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 $\varphi$ worlds:

\[
\text{upd\_pa} :: \text{Ord} \text{ state} \Rightarrow \\
\text{EpistM} \text{ state} \Rightarrow \text{ Frm} \text{ state} \Rightarrow \text{ EpistM} \text{ state} \\
\text{upd\_pa} \text{ m@(Mo states agents val rels actual)} \text{ f} = \\
\text{(Mo sts' agents val' rels' actual')} \\
\hspace{1cm} \text{where} \\
\hspace{1cm} \text{sts'} = \{s \mid s \leftarrow \text{states}, \text{isTrueAt m s f} \} \\
\hspace{1cm} \text{val'} = \{(s, ps) \mid (s,ps) \leftarrow \text{val}, s \in \text{elem} \text{ sts'}\} \\
\hspace{1cm} \text{rels'} = \{(ag,\text{restrict} \text{ sts'} r) \mid (ag,r) \leftarrow \text{rels} \} \\
\hspace{1cm} \text{actual'} = \{s \mid s \leftarrow \text{actual}, s \in \text{elem} \text{ sts'} \}
\]
updps_pa :: Ord state =>
    EpistM state -> [Frm state] -> EpistM state
updps_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

THREE LOGICIANS WALK INTO A BAR...

DOES EVERYONE WANT BEER?

I DON'T KNOW.

I DON'T KNOW.

YES!
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:

\[
\begin{align*}
\text{allBeer} :: & \text{ Frm (Bool,Bool,Bool)} \\
\text{allBeer} = & \text{ Info (True,True,True)} \\
\text{ignA, ignB, ignC} :: & \text{ Frm (Bool,Bool,Bool)} \\
\text{ignA} = & \text{ C [N (Kn a allBeer), N (Kn a (N allBeer))]} \\
\text{ignB} = & \text{ C [N (Kn b allBeer), N (Kn b (N allBeer))]} \\
\text{ignC} = & \text{ C [N (Kn c allBeer), N (Kn c (N allBeer))]} \\
\text{knowC, knowC’} :: & \text{ Frm (Bool,Bool,Bool)} \\
\text{knowC} = & \text{ Kn c allBeer} \\
\text{knowC’} = & \text{ Kn c (N allBeer)}
\end{align*}
\]
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’
```
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]).

\[
\text{statement}_\text{le} = \\
\quad \text{C} [ \text{Info} \ p \ \text{`impl`} \ N (\text{Kn} \ b \ (\text{Info} \ p)) \mid p \leftarrow \text{pairs} ]
\]

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:

\[
\text{k}_a\_\text{statement}_\text{le} = \text{Kn} \ a \ \text{statement}_\text{le}
\]

The second announcement proclaims the statement by \( b \) that now he knows:
\[
\text{statement}_2 = \\
D \left[ \text{Kn} \ b \ (\text{Info} \ p) \mid p \leftarrow \text{pairs} \right]
\]

Equivalently, but computationally more efficient:

\[
\text{statement}_2e = \\
C \left[ \text{Info} \ p \ \text{`impl`} \ \text{Kn} \ b \ (\text{Info} \ p) \mid p \leftarrow \text{pairs} \right]
\]

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

\[
\text{statement}_3 = \\
D \left[ \text{Kn} \ a \ (\text{Info} \ p) \mid p \leftarrow \text{pairs} \right]
\]

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 $\mathcal{P}$. 
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 $P$.
- $L$ 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.

\[ p : (i, 3), (j, 1) \]
\[ q : (i, 2), (j, 2) \]
\[ r : (i, 1), (j, 1) \]
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 $r$. 

$p : (i, 3), (j, 1)$
$q : (i, 2), (j, 2)$
$r : (i, 1), (j, 1)$
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

\[
wrel :: \text{Agent} \rightarrow \text{EpistWM}\ a\rightarrow \text{Erel}\ a
\]
\[
wrel\ \text{ag}\ (\text{WMo}\ _\ _\ _\ \text{rels}\ _\ _\ )\ =\ \text{apply}\ \text{rels}\ \text{ag}
\]
Extracting an epistemic relation from a weight model

\[
\begin{align*}
\text{wrel} &:: \text{Agent} \to \text{EpistWM} \ a \to \text{Erel} \ a \\
\text{wrel} \ \text{ag} \ (\text{WMo} \ _ \ _ \ _ \ \text{rels} \ _ \ _) &= \text{apply} \ \text{rels} \ \text{ag}
\end{align*}
\]

Restriction of a table to a domain

\[
\begin{align*}
\text{restrictT} &:: \text{Eq} \ a \implies [a] \to [(a,b)] \to [(a,b)] \\
\text{restrictT} \ \text{xs} \ \text{t} &= [ \ (x,y) \mid (x,y) <\!\!\!\!\_ \ \text{t}, \ \text{elem} \ x \ \text{xs} \ ]
\end{align*}
\]

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].

\[ \varphi ::= \top | p | \neg \varphi | \varphi \land \varphi | \Phi \leq_i \Phi \]

\[ \Phi ::= \varphi | \varphi \oplus \Phi \]
Language

We use notation borrowed from [DR15].

\[
\varphi ::= \top \mid p \mid \neg \varphi \mid \varphi \land \varphi \mid \Phi \leq_i \Phi
\]
\[
\Phi ::= \varphi \mid \varphi \oplus \Phi
\]

Abbreviations:

As usual for \( \bot, \lor, \to, \leftrightarrow \).

\( \Phi <_i \Psi \) for \( \Phi \leq_i \Psi \land \neg \Psi \leq_i \Phi \).

\( \Phi =_i \Psi \) for \( \Phi \leq_i \Psi \land \Psi \leq_i \Phi \).

\( B_i \varphi \) for \( \neg \varphi <_i \varphi \), \( \tilde{B}_i \varphi \) for \( \neg \varphi \leq_i \varphi \). “Belief as willingness to bet”

\( K_i \varphi \) for \( \top \leq_i \varphi \), \( \tilde{K}_i \varphi \) for \( \bot <_i \varphi \). “Knowledge as certainty”
Semantics for this language

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

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

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

\[
\begin{align*}
kb, \ kd :: Agent \rightarrow Form\ a \rightarrow Form\ a \\
kb\ i\ f &= \text{Leq}\ i\ [\text{Top}]\ [f] \\
kd\ i\ f &= \text{Less}\ i\ [\text{Ng}\ \text{Top}]\ [f]
\end{align*}
\]
Abbreviations

**Knowledge as certainty.**

\[
\begin{align*}
kb, \ kd & \in Agent \rightarrow Form\ a \rightarrow Form\ a \\
kb\ i\ f & = \text{Leq}\ i\ [\text{Top}]\ [f] \\
kd\ i\ f & = \text{Less}\ i\ [\text{Ng}\ Top]\ [f]
\end{align*}
\]

**Belief as willingness to bet.**

\[
\begin{align*}
bb, \ bd & \in Agent \rightarrow Form\ a \rightarrow Form\ a \\
bb\ i\ f & = \text{Less}\ i\ [\text{Ng}\ f]\ [f] \\
bd\ i\ f & = \text{Leq}\ i\ [\text{Ng}\ f]\ [f]
\end{align*}
\]
Extracting a probability function from a model

The probability of \( w \) for \( i \) in \( M \) 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:

\[
P_i(w) = \frac{L_i w}{L_i[w]_i}.
\]

\[
\text{prob :: Eq a => Agent -> EpistWM a -> a -> Rational}
\]

\[
\text{prob ag (WMo _ _ _ rels weight _) w = let}
\]

\[
\text{r = apply rels ag}
\]

\[
\text{block = bl r w}
\]

\[
\text{table = apply weight ag}
\]

\[
\text{f = apply table}
\]

\[
\text{in}
\]

\[
\text{f w / sum (map f block)}
\]
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

\[
\begin{align*}
&\quad \text{m@}(\text{WMo worlds agents val acc weight points}) \\
&\quad \text{w (Less i fs1 fs2) =} \\
&\quad \text{let} \\
&\quad \quad r = \text{wrel i m} \\
&\quad \quad b = \text{bl r w} \\
&\quad \quad s1 = \text{sum } \{ \text{prob i m u | u } \leftarrow b, f \leftarrow \text{fs1}, \\
&\quad \quad \quad \quad \text{isTrue m u f } \} \\
&\quad \quad s2 = \text{sum } \{ \text{prob i m u | u } \leftarrow b, f \leftarrow \text{fs2}, \\
&\quad \quad \quad \quad \text{isTrue m u f } \} \\
&\quad \text{in} \\
&\quad s1 < s2
\end{align*}
\]
isTrue

\[ m@((W\text{Mo worlds agents val acc weight points}) \quad w \ (\text{Leq} \ i \ fs1 \ fs2) = \]

let

\[ r = \text{wrel} \ i \ m \]

\[ b = \text{bl} \ r \ w \]

\[ s1 = \text{sum} \ [ \text{prob} \ i \ m \ u \mid u <- b, f <- fs1, \]

\[ \quad \text{isTrue} \ m \ u \ f ] \]

\[ s2 = \text{sum} \ [ \text{prob} \ i \ m \ u \mid u <- b, f <- fs2, \]

\[ \quad \text{isTrue} \ m \ u \ f ] \]

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 $\varphi$ worlds:

\begin{verbatim}
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' ]
\end{verbatim}
upsds_wpa :: Ord state =>
    EpistWM state -> [Form state] -> EpistWM state
upsds_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 $\text{invest}'$ be the result of the public announcement that not $r$.

\[
\text{invest}' = \text{upd_wpa inspect} \ (\text{Ng} \ (\text{Pr} \ r))
\]

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:

\[
\begin{align*}
test &= Pr \, p \\
disease &= Pr \, q
\end{align*}
\]

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

\[
\text{dmodel'} = \text{upd_wpa dmodel test}
\]

*DEMO> \text{sprob dmodel'} 0 \text{ a disease}
1 \% 23
*DEMO> \text{isTrue dmodel'} 0 (bb \text{ a disease})
False
*DEMO> \text{sprob dmodel'} 0 \text{ b disease}
9 \% 11
*DEMO> \text{isTrue dmodel'} 0 (bb \text{ 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 probability information by means of weight functions was designed with implementation of model checking in mind.
Conclusion

- Representation of probability 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 probability 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 probability 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 probability 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


