

# Public Announcements as Updates

Jan van Eijck

jve@cwi.nl

December 21, 2005

## Abstract

Public announcement is a means to create common knowledge: if  $\varphi$  is publicly announced to a set of agents, then every agent knows  $\varphi$ , every agent knows that every agent knows that  $\varphi$ , and so on.

We will first look at definitions, and then turn to implementation.

## Public Announcement as an Update

Public announcements  $[\varphi]$  change knowledge states, so their semantics can be given as a function from Kripke models to Kripke models:

$$M \mapsto M \mid \varphi$$

$M \mid \varphi$  given by:

$$\text{if } M = (W, V, R) \text{ then } M \mid \varphi = (W', V', R')$$

with

$$W' = \{w \in W \mid M, w \models \varphi\}$$

$$V' = V \upharpoonright W'$$

$$R' = \{w \xrightarrow{a} w' \mid w \xrightarrow{a} w' \in R, w, w' \in W'\}$$

## Effect on Actual Worlds

A pointed Kripke model is a quadruple  $M = (W, V, R, U)$  with  $(W, V, R)$  a Kripke model, and  $U \subseteq W$  a set of points.

Intention: the actual world is among  $U$ .

Extension of the definition  $M \mid \varphi$  to pointed models:

$$(W, V, R, U) \mid \varphi = (W', V', R', U')$$

where  $(W', V', R')$  is as above, and

$$U' = \{u \in U \mid (W, V, R), u \models \varphi\}$$

## Public Announcement with Falsehood

$\varphi$  is a falsehood in pointed model  $M = (W, V, R, U)$  if

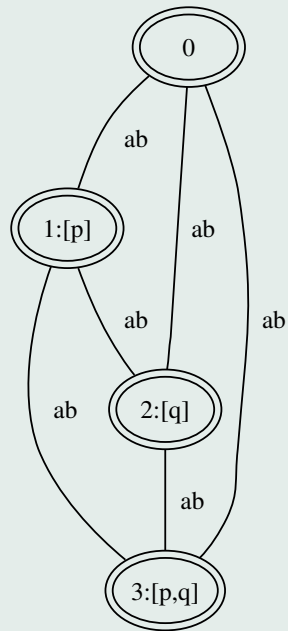
$$(W, V, R), u \not\models \varphi$$

for all  $u \in U$ .

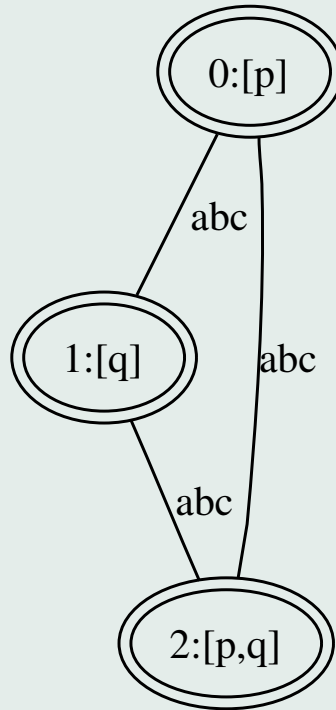
The result of updating with a falsehood is an **inconsistent** pointed model, i.e., a pointed model of the form  $(W', V', R', \emptyset)$ .

## Learning that $p \vee q$ by public announcement

Initial model:  $a$  and  $b$  ignorant about  $p$  and  $q$ , and no possibility as yet ruled out:



Result of public announcement that  $p \vee q$ :



## Module Declaration

```
module LAI10  
  
  where  
  import List  
  import Char  
  import LAI9
```

## Example Epistemic Model

```
s5example :: EpistM Integer
s5example =
  Mo [0..3]
    [a..c]
    [(0, []), (1, [P 0]), (2, [Q 0]), (3, [P 0, Q 0])]
    ([ (a,x,x) | x <- [0..3] ] ++
     [ (b,x,x) | x <- [0..3] ] ++
     [ (c,x,y) | x <- [0..3], y <- [0..3] ])
    [1]
```

## Extracting the relations from an epistemic model

```
rel :: Agent -> EpistM a -> Rel a
rel a (Mo states agents val rels actual) =
    [ (x,y) | (agent,x,y) <- rels, a == agent ]
```

This gives:

```
LAI9> rel a s5example
```

```
[(0,0),(1,1),(2,2),(3,3)]
```

```
LAI9> rel b s5example
```

```
[(0,0),(1,1),(2,2),(3,3)]
```

```
LAI9> rel c s5example
```

```
[(0,0),(0,1),(0,2),(0,3),(1,0),(1,1),(1,2),(1,3),
 (2,0),(2,1),(2,2),(2,3),(3,0),(3,1),(3,2),(3,3)]
```

## Displaying S5 Models

The function `list2partition` (your homework for this week) can be used to write a display function for S5 models that shows each accessibility relation as a partition, as follows.

```
showS5 :: (Ord a, Show a) => EpistM a -> [String]
showS5 m@(Mo states agents val rels actual) =
  show states :
  show val    :
  map show [ (a, (list2partition states) (rel a m))
             | a <- agents ]
++
[show actual]
```

## Example Display

```
displayS5 :: (Ord a, Show a) => EpistM a -> IO()
displayS5 = putStrLn . unlines . showS5
```

```
LAI10> displayS5 s5example
[0,1,2,3]
[(0, []), (1, [p]), (2, [q]), (3, [p, q])]
(a, [[0], [1], [2], [3]])
(b, [[0], [1], [2], [3]])
(c, [[0,1,2,3]])
[1]
```

## Blissful Ignorance

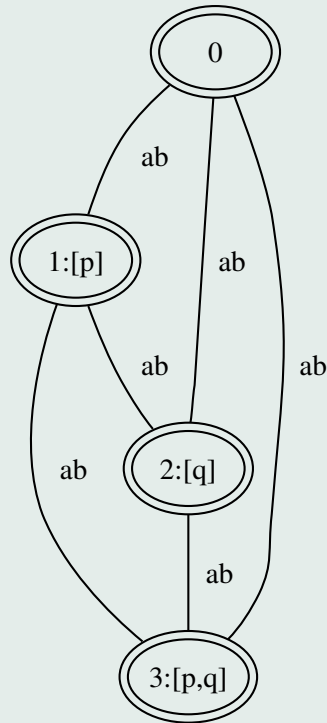
Blissful ignorance is the state where you don't know anything, but you know also that there is no reason to worry, for you know that nobody knows anything.

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:

$$\begin{aligned}M &= (W, V, R) \text{ where} \\W &= \{0, \dots, 2^k - 1\} \\V &= \text{any surjection in } W \rightarrow \mathcal{P}(P) \\R &= \{x \xrightarrow{a} y \mid x, y \in W, a \in A\}.\end{aligned}$$

Note that  $V$  is in fact a bijection, for  $|\mathcal{P}(P)| = 2^k = |W|$ .

## Blissful Ignorance – Example



## Generating Models for Blissful Ignorance

```
initM :: [Agent] -> [Prop] -> EpistM Integer
initM ags props = (Mo worlds ags val accs points)
  where
    worlds = [0..(2k-1)]
    k      = length props
    val    = zip worlds (sortL (powerList props))
    accs   = [ (ag,st1,st2) | ag  <- ags,
                                     st1 <- worlds,
                                     st2 <- worlds      ]
    points = worlds
```

sortL and powerList are computer lab exercises for you.

## zip

zip is a predefined function for zipping two lists together. Home-made version:

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

This gives:

```
LAI10> zip [0..2^3-1] (sortL (powerList [P 1,P 2, P 3]))
[(0, []), (1, [p1]), (2, [p2]), (3, [p3]), (4, [p1,p2]),
 (5, [p1,p3]), (6, [p2,p3]), (7, [p1,p2,p3])]
```

## General Knowledge

The general knowledge accessibility relation of a set of agents  $C$  is given by

$$\bigcup_{c \in C} R_c.$$

Implementation:

```
genK :: Ord state => [(Agent, state, state)]
      -> [Agent] -> Rel state
genK r ags = [ (x,y) | (a,x,y) <- r, a 'elem' ags ]
```

## Right Section of a Relation

If  $R$  is a binary relation on  $A$ , and  $a \in A$ , then  $aR$  is the set

$$\{b \in A \mid aRb\}.$$

Implementation:

```
rightS :: Ord a => Rel a -> a -> [a]
rightS r x = (sort.nub) [ z | (y,z) <- r, x == y ]
```

## General Knowledge Alternatives

```
genAlts :: Ord state => [(Agent, state, state)]
         -> [Agent] -> state -> [state]
genAlts r ags s = rightS (genK r ags) s
```

## Closures of Relations

If  $\mathcal{O}$  is a set of properties of relations on a set  $A$ , then the  $\mathcal{O}$  closure of a relation  $R$  on  $A$  is **the smallest relation  $S$  that includes  $R$  and that has all the properties in  $\mathcal{O}$ .**

The most important closures of relations:

- the reflexive closure,
- the symmetric closure,
- the transitive closure,
- the reflexive transitive closure.

## Reflexive Transitive Closure

Let a set  $A$  be given. Let  $R$  be a binary relation on  $A$ . Let  $I = \{(x, x) \mid x \in A\}$ .

We define  $R^n$  for  $n \geq 0$ , as follows:

- $R^0 = I$ .
- $R^{n+1} = R \circ R^n$ .

Next, define  $R^*$  by means of:

$$R^* = \bigcup_{n \in \mathbb{N}} R^n.$$

In the lecture on common knowledge it was stated that  $R^*$  is the reflexive transitive closure of  $R$ . Here we are going to **prove** that fact.

## $R^*$ is the reflexive transitive closure of $R$

We have to show the following:

1.  $R \subseteq R^*$ ,
2.  $R^*$  is reflexive and transitive,
3.  $R^*$  is the **smallest** reflexive and transitive relation that includes  $R$ .

(1) To show that  $R \subseteq R^*$ , we show that  $R^1 = R$ .

$(x, y) \in R^1$  iff (definition of  $R^1$ )  $(x, y) \in R \circ I$  iff (definition of  $\circ$ )  
 $\exists z \in A$  with  $(x, z) \in R$  and  $(z, y) \in I$  iff (definition of  $I$ )  $(x, y) \in R$ .

(2) Since  $I \subseteq R^*$ ,  $R^*$  is reflexive. For transitivity, assume  $(x, y) \in R^*$ ,  $(y, z) \in R^*$ . Then there are  $k, m$  with  $(x, y) \in R^k$  and  $(y, z) \in R^m$ . It follows that  $(x, z) \in R^k \circ R^m$ , i.e.,  $(x, z) \in R^{k+m}$ . This proves that  $(x, z) \in R^*$ , so  $R^*$  is transitive.

(3) To show that  $R^*$  is the **smallest** reflexive and transitive relation that includes  $R$ , we first restate what is to be proved. Restatement: if  $S$  is any reflexive and transitive relation on  $A$  with  $R \subseteq S$ , then  $R^* \subseteq S$ . We prove this fact by induction on  $n$ , by showing that for every  $n \in \mathbb{N}$ ,  $R^n \subseteq S$ .

- Base case: If  $(x, y) \in R^0$  then  $x = y$ . It follows by reflexivity of  $S$  that  $(x, y) \in S$ .
- Induction step: Assume (ih)  $R^n \subseteq S$ . We have to show that if  $(x, y) \in R^{n+1}$  then  $(x, y) \in S$ .

So assume  $(x, y) \in R^{n+1}$ . Then  $(x, y) \in R \circ R^n$ , so for some  $z \in A$ ,  $(x, z) \in R$  and  $(z, y) \in R^n$ . By the fact that  $R \subseteq S$ ,  $(x, z) \in S$ . By induction hypothesis,  $(z, y) \in S$ . By transitivity of  $S$ ,  $(x, y) \in S$ . QED.

## Computing Reflexive Transitive Closure

If  $A$  is finite, any  $R$  on  $A$  is finite as well. In particular, there will be  $k$  with  $R^{k+1} = R^k$ .

Thus, in the finite case reflexive transitive closure can be computed by successively computing  $\bigcup_{n \in \{0, \dots, k\}} R^n$  until  $R^{k+1} = R^k$ .

In other words: the reflexive transitive closure of a relation  $R$  can be computed from  $I$  by repeated application of the operation

$$\lambda S.(S \cup (R \circ S)),$$

until the operation reaches a fixpoint.

## Least Fixpoint

A fixpoint of an operation  $f$  is an  $x$  for which  $f(x) = x$ .

Least fixpoint calculation:

```
lfp :: Eq a => (a -> a) -> a -> a
lfp f x | x == f x = x
        | otherwise = lfp f (f x)
```

Reflexive transitive closure:

```
rtc :: Ord a => [a] -> Rel a -> Rel a
rtc xs r = lfp (\ s -> (sort.nub) (s ++ (r@@s))) i
  where i = [(x,x) | x <- xs ]
```

## Computing Transitive Closure

Same as computing reflexive transitive closure, but starting out from the relation  $R$ .

```
tc :: Ord a => Rel a -> Rel a
tc r = lfp (\ s -> (sort.nub) (s ++ (r @@ s))) r
```

Examples:

```
LAI10> tc [(1,2), (2,3), (3,4)]
[(1,2), (1,3), (1,4), (2,3), (2,4), (3,4)]
LAI10> tc [(1,2), (2,3), (3,1)]
[(1,1), (1,2), (1,3), (2,1), (2,2), (2,3), (3,1), (3,2), (3,3)]
LAI10> rtc [1,2,3] [(1,2), (2,3)]
[(1,1), (1,2), (1,3), (2,2), (2,3), (3,3)]
```

## Common Knowledge

The common knowledge relation for group of agents  $C$  is the relation

$$\left(\bigcup_{c \in C} R_c\right)^*.$$

Given that the  $R_c$  are represented as a list of triples

$[(Agent, state, state)]$

we can define a function that extracts the common knowledge relation:

```
commonK :: Ord state => [(Agent, state, state)]  
         -> [Agent] -> [state] -> Rel state  
commonK r ags xs = rtc xs (genK r ags)
```

## Common Knowledge Alternatives

```
commonAlts :: Ord state => [(Agent,state,state)]  
            -> [Agent] -> [state] -> state -> [state]  
commonAlts r ags xs s = rightS (commonK r ags xs) s
```

## Representing Formulas

```
data Form = Top
          | Prop Prop
          | Neg Form
          | Conj [Form]
          | Disj [Form]
          | K Agent Form
          | EK [Agent] Form
          | CK [Agent] Form
  deriving (Eq,Ord)
```

EK is the operator for general knowledge.

CK is the operator for common knowledge.

## Example formulas

```
p = Prop (P 0)
q = Prop (Q 0)
```

Note the following type difference:

```
LAI10> :t (P 0)
P 0 :: Prop
LAI10> :t p
p :: Form
```

```

instance Show Form where
  show Top          = "T"
  show (Prop p)    = show p
  show (Neg f)     = '-' : (show f)
  show (Conj fs)   = '&' : show fs
  show (Disj fs)   = 'v' : show fs
  show (K agent f) = '[' : show agent ++ "]" ++ show f
  show (EK agents f) = 'E' : show agents ++ show f
  show (CK agents f) = 'C' : show agents ++ show f

```

This gives:

```

LAI10> CK [a..c] (Disj[p,K a (Neg p)])
C[a,b,c]v[p,[a]-p]

```

## Valuation Lookup

```
apply :: Eq a => [(a,b)] -> a -> b
```

Computer lab exercise for you.

This can be used to look up the valuation for a world in a model.

## Evaluation

```
isTrueAt :: Ord state =>  
  EpistM state -> state -> Form -> Bool
```

Your homework for the second week.

## Evaluating the State of Bliss

```
test1 = isTrueAt  
  (initM [a..c] [P 0]) 0  
  (CK [a..c] (Neg (K a p)))
```

## Truth in a Model

Use the function `isTrueAt` to implement a function that checks for truth at **all** the designated states of an epistemic model:

```
isTrue :: Ord state => EpistM state -> Form -> Bool
isTrue m@(Mo worlds agents val acc points) f =
  and [ isTrueAt m s f | s <- points ]
```

Another test of `initM`

```
test2 = isTrue
      (initM [a..c] [P 0])
      (CK [a..c] (Neg (K a p)))
```

## Public Announcement Update

```
upd_pa :: Ord state =>
    EpistM state -> Form -> EpistM state
upd_pa m@(Mo states agents val rels actual) f =
    (Mo states' agents val' rels' actual')
  where
    states' = [ s | s <- states, isTrueAt m s f ]
    val'    = [(s,p) | (s,p) <- val,
                    s 'elem' states' ]
    rels'   = [(a,x,y) | (a,x,y) <- rels,
                       x 'elem' states',
                       y 'elem' states' ]
    actual' = [ s | s <- actual, isTrueAt m s f ]
```

## Examples

```
m0 = initM [a..c] [P 0,Q 0]
```

```
LAI10> displayS5 m0
```

```
[0,1,2,3]
```

```
[(0, []), (1, [p]), (2, [q]), (3, [p, q])]
```

```
(a, [[0,1,2,3]])
```

```
(b, [[0,1,2,3]])
```

```
(c, [[0,1,2,3]])
```

```
[0,1,2,3]
```

```
LAI10> displayS5 (upd_pa m0 (Disj [p,q]))
[1,2,3]
[(1, [p]), (2, [q]), (3, [p,q])]
(a, [[1,2,3]])
(b, [[1,2,3]])
(c, [[1,2,3]])
[1,2,3]
```

## Conversion of States to Integers

Convert **any type of state list** to [0..]:

```
convert :: Eq state =>  
        EpistM state -> EpistM Integer
```

Implementation left for you as a computer lab exercise.

```
LAI10> (displayS5.convert) (upd_pa m0 (Disj [p,q]))  
[0,1,2]  
[(0, [p]), (1, [q]), (2, [p,q])]  
(a, [[0,1,2]])  
(b, [[0,1,2]])  
(c, [[0,1,2]])  
[0,1,2]
```

## Next Time

Bisimulations