Representing Action Model Updates

Jan van Eijck

jve@cwi.nl

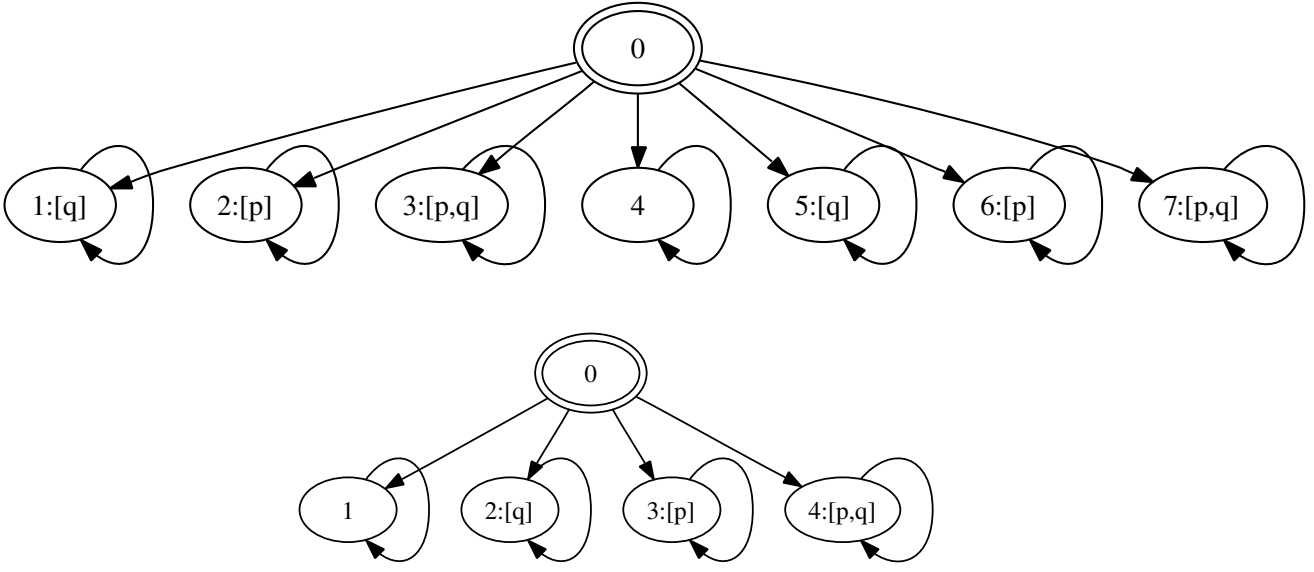
August 6, 2008

Abstract

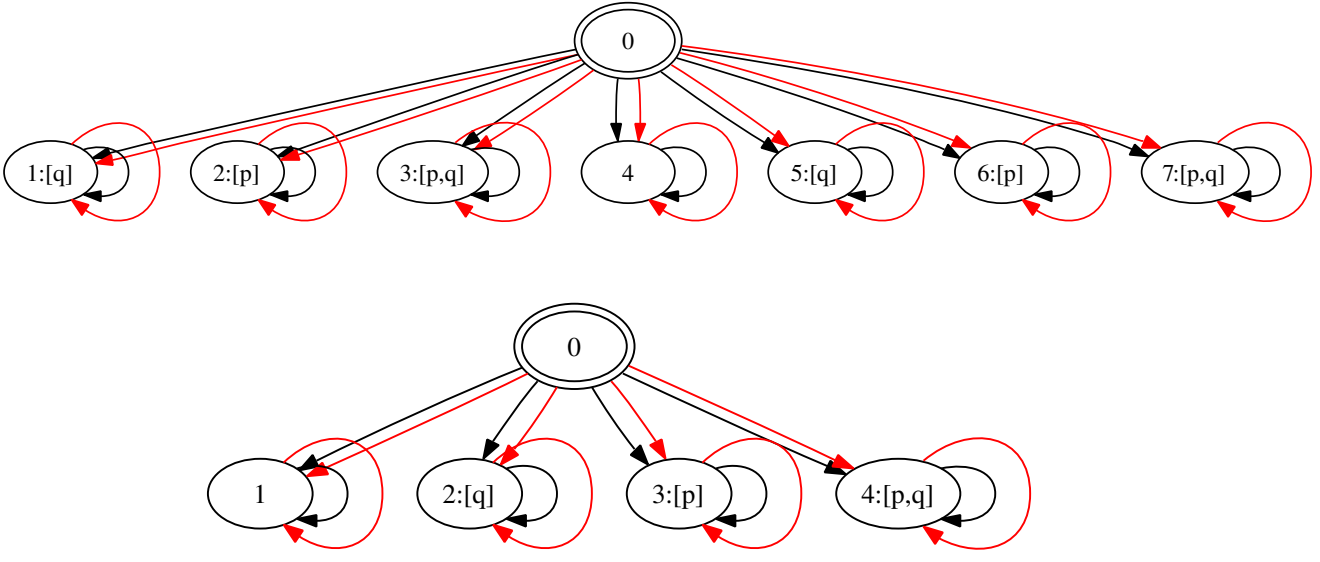
First, we will show how bisimulation can be used to simplify Kripke models by replacing each state in the model by its bisimilarity class, and we will present and implement an algorithm for carrying this out.
Action models are a special kind of Kripke models, where the valuations are replaced by precondition formulas. A wide range of communicative actions can be modelled as updates with action models.
We define a datatype for action models, and then turn to the definition and implementation of action model update.

When are Epistemic Models 'Equivalent' ?

Can you find a formula that is true in the actual world of one of the models, but false in the actual world of the other?



Can you find a formula that is true in the actual world of one of the models, but false in the actual world of the other?



Bisimulation — Definition

The notion of bisimulation is intended to capture state equivalences and process equivalences.

A bisimulation Z between Models $\mathbf{M}$ and $\mathbf{N}$ is a relation on $S_{\mathbf{M}} \times S_{\mathbf{N}}$ such that if sZt then the following hold:

Invariance $V_{\mathbf{M}}(s) = V_{\mathbf{N}}(t)$ (the two states have the same valuation),

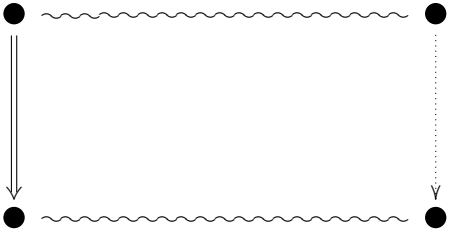
Zig if for some $s' \in S_{\mathbf{M}}$ $s \xrightarrow{a} s' \in R_{\mathbf{M}}$ then there is a $t' \in S_{\mathbf{N}}$ with $t \xrightarrow{a} t' \in R_{\mathbf{N}}$ and $s'Zt'$.

Zag same requirement in the other direction.

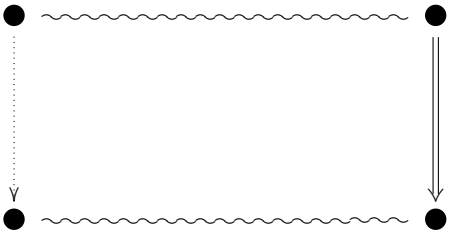
Use $Z : \mathbf{M}, s \underline{\leftrightarrow} \mathbf{N}, t$ to indicate that Z is a bisimulation that connects s and t. Use $\mathbf{M}, s \underline{\leftrightarrow} \mathbf{N}, t$ to indicate that there is a bisimulation that connects s and t. If the models are clear, use $s \underline{\leftrightarrow} t$. If $s \underline{\leftrightarrow} t$ one says that s and t are bisimilar.

Invariance

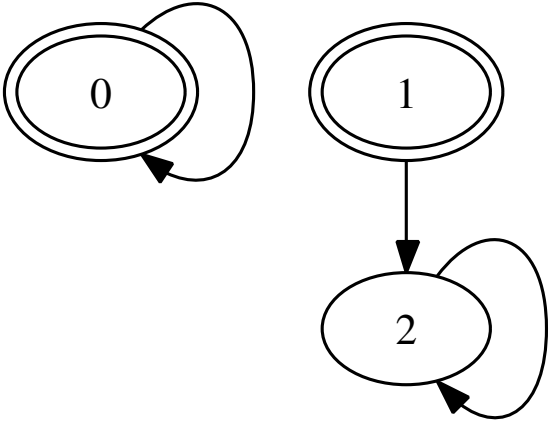
Zig



Zag

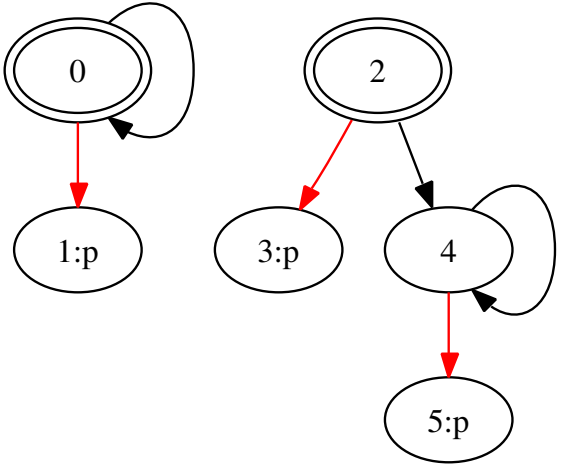


Bisimulation — Example 1



In the models of the picture, $0 \underline{\leftrightarrow} 1 \underline{\leftrightarrow} 2$.

Bisimulation — Example 2



In the models of the picture, $0 \underline{\leftrightarrow} 2 \underline{\leftrightarrow} 4$ and $1 \underline{\leftrightarrow} 3 \underline{\leftrightarrow} 5$.

Invariance for Bisimulation

A formula φ (of some logical language suitable for talking about Kripke models) is called invariant for bisimulation if the following holds:

If

$$\mathbf{M}, s \leftrightarrows \mathbf{N}, t$$

then

$$\mathbf{M}, s \models \varphi \text{ iff } \mathbf{N}, t \models \varphi.$$

Fact: Epistemic formulas are invariant for bisimulation.

Proof: Induction on the structure of formula φ.

Bisimilarity is an Equivalence

Bisimilarity within a single model $\mathbf{M}$ is an equivalence relation.

Reflexive Surely $I : s \underline{\leftrightarrow} s$ for every state s in $\mathbf{M}$ (the identity relation is a bisimulation). Thus $s \underline{\leftrightarrow} s$.

Symmetric Let $s \underline{\leftrightarrow} t$. Then there is a Z with $Z : s \underline{\leftrightarrow} t$. Note that the invariance, zig, and zag conditions are symmetric. Thus, $Z : s \underline{\leftrightarrow} t$ implies $Z^{\smile} : t \underline{\leftrightarrow} s$. Thus, $t \underline{\leftrightarrow} s$.

Transitive Assume $s \underline{\leftrightarrow} t$ and $t \underline{\leftrightarrow} u$. Then there are $Z : s \underline{\leftrightarrow} t$ and $Z' : t \underline{\leftrightarrow} u$. But then $Z \circ Z' : s \underline{\leftrightarrow} u$, and therefore $s \underline{\leftrightarrow} u$.

It follows that we can simplify a Kripke model by replacing each state s by its bisimilarity-class $|s|_{\underline{\leftrightarrow}}$.

By the fact that epistemic formulas are invariant for bisimulation this does not affect the truth of any epistemic formulas.

Partition Refinement

Given: A Kripke model $\mathbf{M}$.

Problem: find the Kripke model that results from replacing each state s in $\mathbf{M}$ by its bisimilarity class $|s|_{\underline{\leftrightarrow}}$.

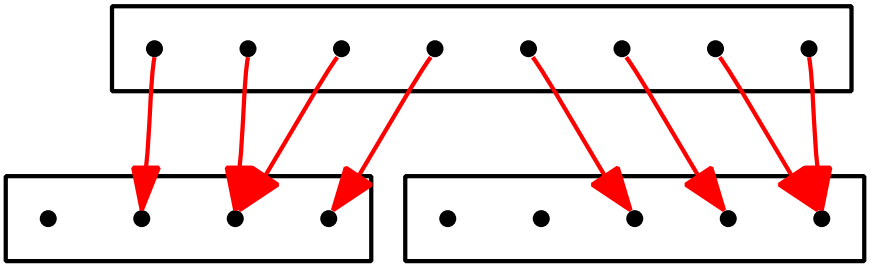
The problem of finding the smallest Kripke model modulo bisimulation is similar to the problem of minimizing the number of states in a finite automaton [5].

We will use partition refinement, in the spirit of [6].

Partition Refinement Algorithm

- Start out with a partition of the state set where all states with the same valuation are in the same class.
- Given a partition Π, for each block b in Π, partition b into sub-blocks such that two states s, t of b are in the same sub-block iff for all agents a it holds that s and t have $\xrightarrow{a}$ transitions to states in the same block of Π. Update Π to Π' by replacing each b in Π by the newly found set of sub-blocks for b.
- Halt as soon as $\Pi = \Pi'$.

Splitting a Block



Module Declaration

```
module RAMU where

import List
import HFKR
import RPAU

type State = Integer
```

Valuation Comparison

```
sameVal :: (Eq a,Eq b) => [(a,b)] -> a -> a -> Bool
sameVal val w1 w2 = apply val w1 == apply val w2
```

From Equivalence Relations to Partitions

Relations as characteristic functions.

```
cf2part :: (Eq a) =>
             [a] -> (a -> a -> Bool) -> [[a]]
cf2part [] r = []
cf2part (x:xs) r = xblock : cf2part rest r
  where
  (xblock,rest) = (x : filter (r x) xs,
                   filter (not . (r x)) xs)
```

Initial Partition

We start with the partition based on the relation 'having the same valuation':

```
initPartition :: Eq a => EpistM a -> [[a]]
initPartition (Mo states agents val rel actual) =
  cf2part states (\ x y -> sameVal val x y)
```

The block of an object in a partition

The block of `x` in a partition is the block that has `x` as an element.

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

Accessible Blocks

For an agent from a given state, given a model and a partition:

```
accBlocks :: Eq a =>
             EpistM a -> [[a]] -> a -> Agent -> [[a]]
accBlocks m@(Mo _ _ _ rel _) part s ag =
    nub [ bl part y | (ag',x,y) <- rel,
                      ag' == ag, x == s ]
```

Having the same accessible blocks under a partition

```
sameAB :: Eq a =>
          EpistM a -> [[a]] -> a -> a -> Bool
sameAB m@(Mo states ags val rel actual) part s t =
    and [ accBlocks m part s ag
          == accBlocks m part t  ag | ag <- ags ]
```

Refinement Step of Partition by Block Splitting

Splitting the blocks `bl` of `p`:

```
refineStep :: Eq a => EpistM a -> [[a]] -> [[a]]
refineStep m p = refineP m p p
  where
  refineP :: Eq a =>
             EpistM a -> [[a]] -> [[a]] -> [[a]]
  refineP m part [] = []
  refineP m part (bl:blocks) =
     newblocks ++ (refineP m part blocks)
       where
         newblocks =
          cf2part bl (\ x y -> sameAB m part x y)
```

Refining a Partition

The refining process can be implemented as a least fixpoint computation on the operation of taking refinement steps.

```
refine :: Eq a => EpistM a -> [[a]] -> [[a]]
refine m = lfp (refineStep m)
```

Remark: least fixpoint computation is an element of many refinement processes.

It is an example of what is called a design pattern in Software Engineering [4].

Construction of Minimal Model

```
minimalModel :: (Eq a, Ord a) =>
                  EpistM a -> EpistM [a]
minimalModel m@(Mo states agents val rel actual) =
  (Mo states' agents val' rel' actual')
     where
     states'   = refine m (initPartition m)
     f         = bl states'
     val'      = (nub . sort)
        (map (\ (x,y)   -> (f x, y))      val)
     rel'      = (nub . sort)
        (map (\ (x,y,z) -> (x, f y, f z)) rel)
     actual'   = map f actual
```

Map to Bisimulation Minimal Model

Map the states to their bisimilarity classes.

Next, convert the bisimilarity classes back into integers:

```
bisim ::  (Eq a, Ord a) =>
           EpistM a -> EpistM State
bisim = convert . minimalModel . gsm
```

Examples

```
lai0 :: EpistM State
lai0 = Mo
         [0..7]
         [a,b,c]
         (zip [0..]
          ((powerList [P 1, P 2])
            ++ (powerList [P 1, P 2])))
         [(ag,x,x) | ag <- [a,b,c], x <- [0..7] ]
         [2]
```

```
RAMU> displayS5 lai0
[0,1,2,3,4,5,6,7]
[(0,[]),(1,[p2]),(2,[p1]),(3,[p1,p2]),(4,[]),
 (5,[p2]),(6,[p1]),(7,[p1,p2])]
(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]])
[2]

RAMU> displayS5 (bisim lai0)
[0,1,2,3]
[(0,[]),(1,[p2]),(2,[p1]),(3,[p1,p2])]
(a,[[0],[1],[2],[3]])
(b,[[0],[1],[2],[3]])
(c,[[0],[1],[2],[3]])
[2]
```

```
lai1 :: EpistM State
lai1 = let worlds = [0..10] in
       Mo
        worlds
        [a,b,c]
        (zip worlds (repeat [P 0]))
        [(ag,x,y) | ag <- [a,b,c],
                    x  <- worlds, y <- worlds ]
        [10]
```

```
RAMU> displayS5 lai1
[0,1,2,3,4,5,6,7,8,9,10]
[(0,[p]),(1,[p]),(2,[p]),(3,[p]),(4,[p]),(5,[p]),
 (6,[p]),(7,[p]),(8,[p]),(9,[p]),(10,[p])]
(a,[[0,1,2,3,4,5,6,7,8,9,10]])
(b,[[0,1,2,3,4,5,6,7,8,9,10]])
(c,[[0,1,2,3,4,5,6,7,8,9,10]])
[10]

RAMU> displayS5 (bisim lai1)
[0]
[(0,[p])]
(a,[[0]])
(b,[[0]])
(c,[[0]])
[0]
```

```
lai2 :: EpistM State
lai2 = let worlds = [0..10] in
       Mo
        worlds
        [a,b,c]
        ((0,[Q 0]): (zip [1..10] (repeat [P 0])))
        [(ag,x,y) | ag <- [a,b,c],
                    x  <- worlds, y <- worlds ]
        [10]
```

```
RAMU> displayS5 lai2
[0,1,2,3,4,5,6,7,8,9,10]
[(0,[q]),(1,[p]),(2,[p]),(3,[p]),(4,[p]),(5,[p]),
 (6,[p]),(7,[p]),(8,[p]),(9,[p]),(10,[p])]
(a,[[0,1,2,3,4,5,6,7,8,9,10]])
(b,[[0,1,2,3,4,5,6,7,8,9,10]])
(c,[[0,1,2,3,4,5,6,7,8,9,10]])
[10]

RAMU> displayS5 (bisim lai2)
[0,1]
[(0,[q]),(1,[p])]
(a,[[0,1]])
(b,[[0,1]])
(c,[[0,1]])
[1]
```

Action Models

Action models are like Kripke models, but with the valuation function replaced by a precondition function.

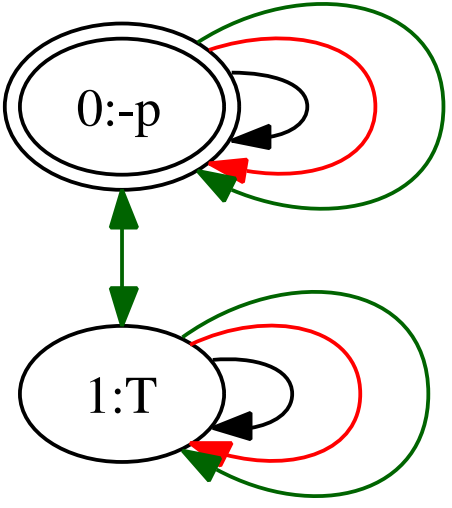
Their components are called actions.

If an action has precondition φ then the action can be applied to worlds where φ holds.

See [1, 3].

Representing Updates as Action Models

Suppose there are three agents, **a**, b, c.



Which update is this?

Updating with an Action Model

The result of updating with an action model is defined as the product of the epistemic model and the action model, restricted to the pairs (w, u) where

- w satisfies the precondition of action u, and
- the accessibility relations hold between pairs (w, u) and (w', u') just in case they hold both between w and w' and between u and u'.

The valuation of (v, u) in the new model is that of v in the old epistemic model.

See [1, 2].

Update Execution: Formal Definition

Given an epistemic model

$$\mathbf{M} = (W, V, R, U)$$

where $U \subseteq W$ is the set of actual worlds, and an action model

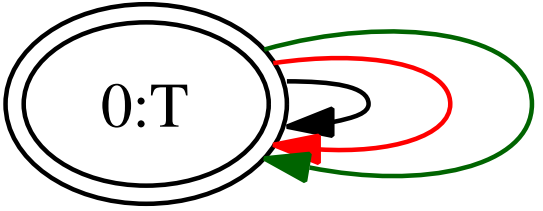
$$A = (\mathsf{E}, \mathsf{pre}, \mathsf{R}, \mathsf{F})$$

where $\mathsf{F} \subseteq \mathsf{E}$ is the set of actual actions, we say that the result of executing A, s in M, w is the model $M \circ A = (W', V', R', U')$ where

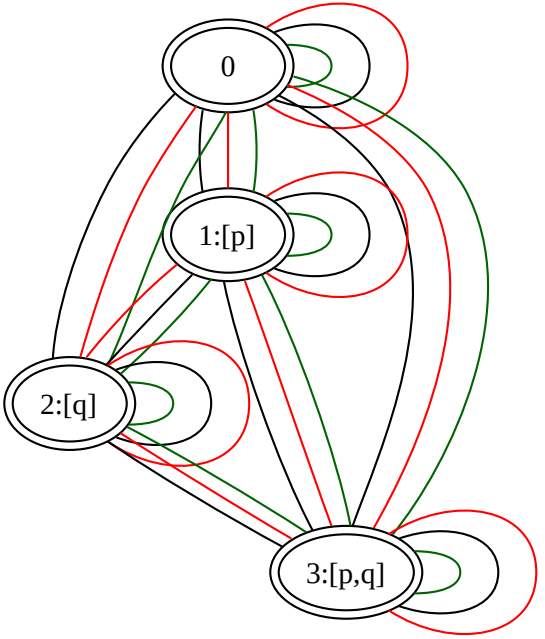
- $W' = \{(v, \mathsf{t}) \mid M, v \models \mathsf{pre}(\mathsf{t})\}$,
- $V'(v, b) = V(v)$,
- $R'(a) = \{((v, \mathsf{t}), (u, \mathsf{u})) \mid (v, u) \in R(a) \text{ and } (\mathsf{t}, \mathsf{u}) \in \mathsf{R}(a)\}$,
- $U' = \{(v, \mathsf{t}) \in W' \mid v \in U, \mathsf{t} \in \mathsf{F}\}$.

The Action that Changes Nothing

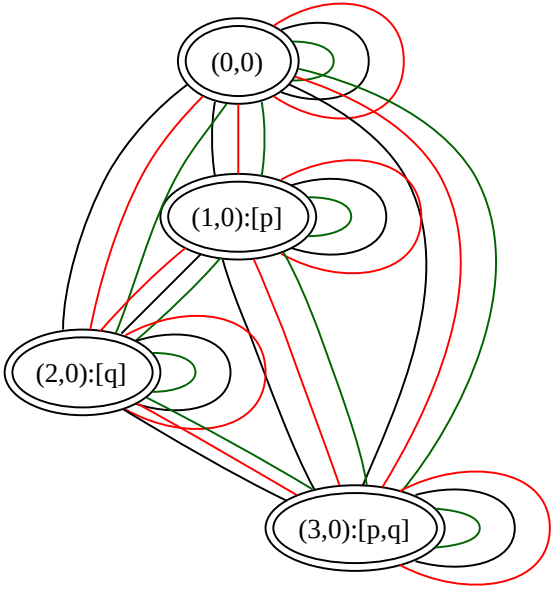
Suppose there are three agents, **a**, b, c.



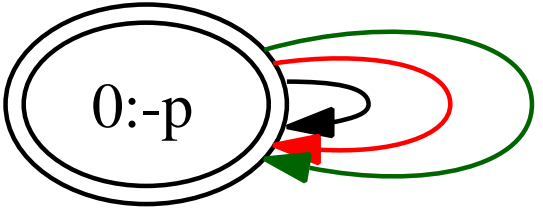
Blissful Ignorance



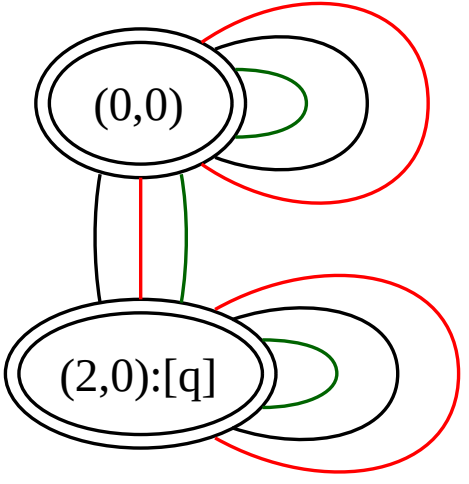
After Update with ‘The Action That Changes Nothing’



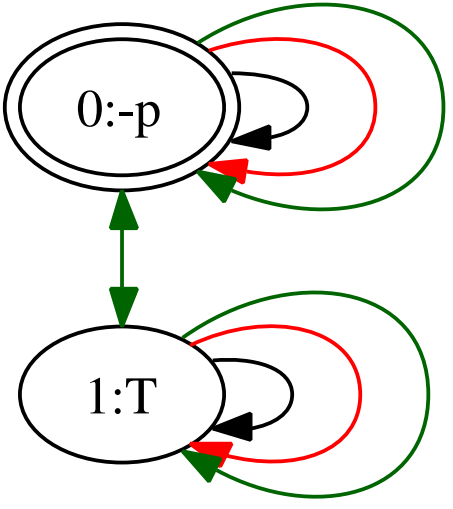
Public Announcement of $\neg p$



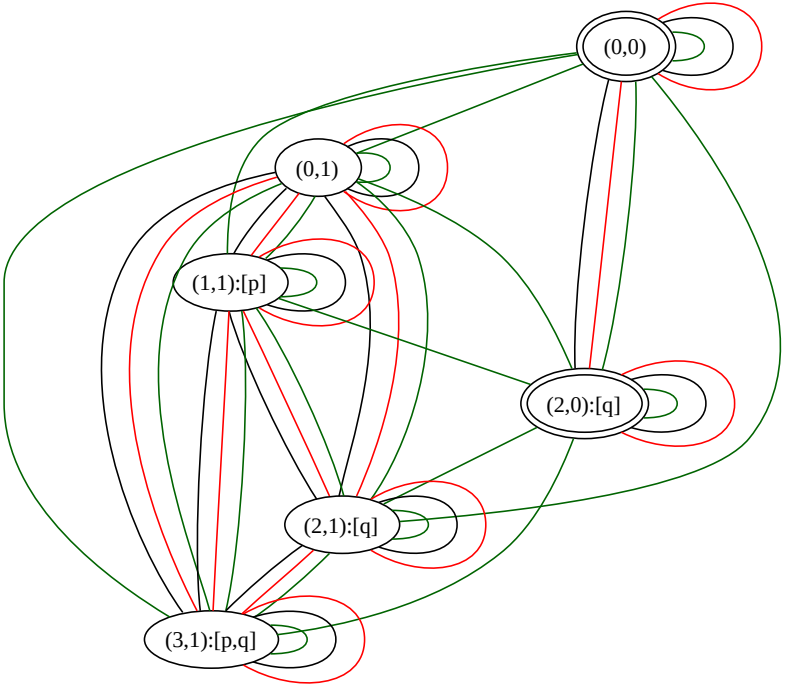
After Update with Public Announcement of $\neg p$



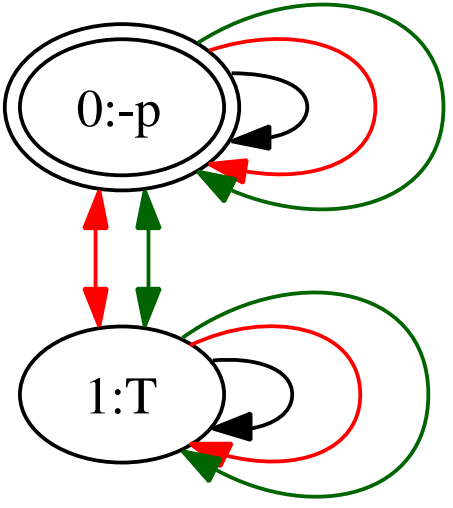
Group Message to a, b that $\neg p$



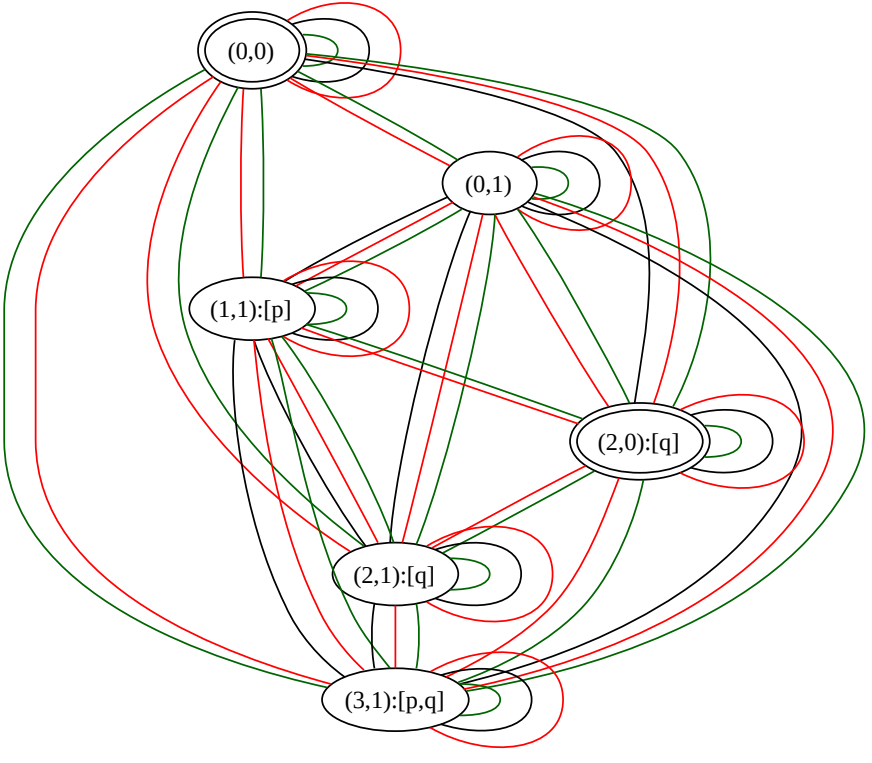
After Update with Group Message to a, b that $\neg p$



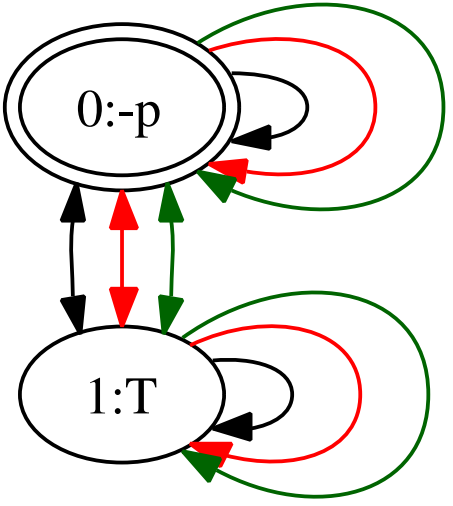
Private Message to a that $\neg p$



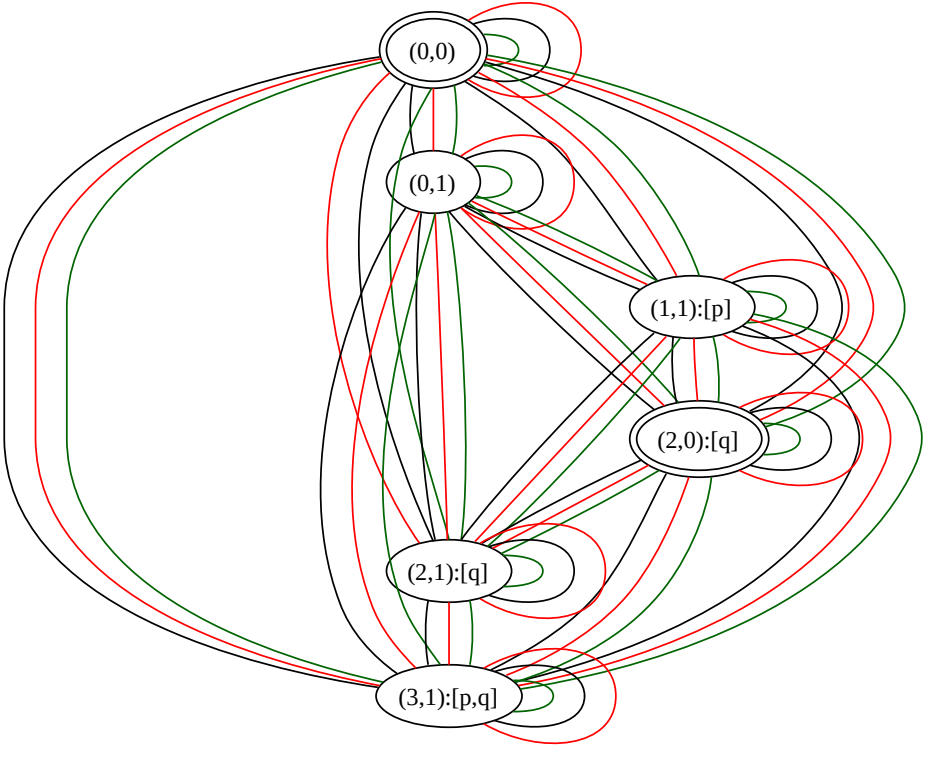
After Update with Private Message to a that $\neg p$



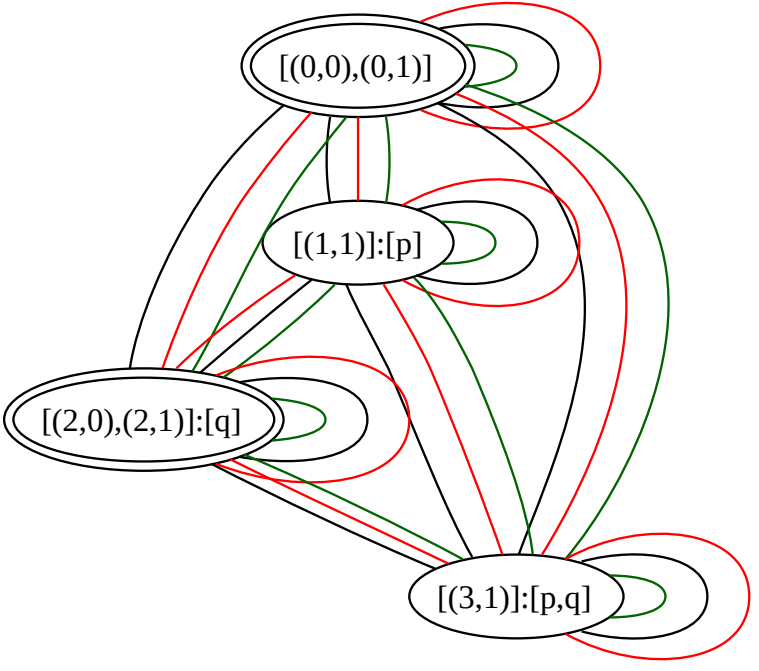
Test that $\neg p$



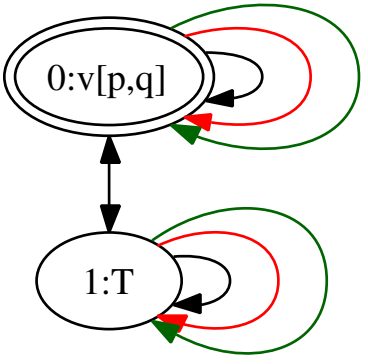
After Update with Test that $\neg p$



Bisimulation Minimal Version



Group Message to b, c that $p \vee q$



Module for Action Models

```
module AM where

import List
import HFKR
import RPAU
import RAMU
```

Datatype for Action Models

```
data AM state = Am
              [state]
              [Agent]
              [(state,Form)]
              [(Agent,state,state)]
              [state]  deriving (Eq,Show)
```

Action Models

In updating with an action model, we will have to make sure that the set of agents of the action model is the same as that of the epistemic model that gets updated.

Therefore, it makes sense to update with functions from agents to action models:

```
type FAM state = [Agent] -> AM state
```

Updating with an Action Model

```
up :: (Eq state, Ord state) =>
       EpistM state -> FAM state
                    -> EpistM (state,state)
```

```
up m@(Mo worlds ags val rel points) fam =
   Mo worlds' ags' val' rel' points'
   where
   Am states ags' pre susp actuals = fam ags
   worlds' = [ (w,s) | w <- worlds, s <- states,
                       isTrueAt m w (apply pre s)    ]
   val'    = [ ((w,s),props) | (w,props) <- val,
                               s         <- states,
                               elem (w,s) worlds']
   rel'    = [ (ag1,(w1,s1),(w2,s2)) |
                 (ag1,w1,w2) <- rel,
                 (ag2,s1,s2) <- susp,
                  ag1 == ag2,
                  elem (w1,s1) worlds',
                  elem (w2,s2) worlds'   ]
   points' = [ (p,a) | p <- points, a <- actuals,
                       elem (p,a) worlds'          ]
```

Update and simplify

```
upd :: (Eq state, Ord state) =>
       EpistM state -> FAM state
                    -> EpistM State
upd m a = bisim (up m a)
```

Public Announcement Again

Update model consists of a single action, with reflexive arrows for all agents.

Precondition is the formula that expresses the content of the announcement.

```
public :: Form -> FAM State
public form ags = Am [0] ags [(0,form)]
                        [(a,0,0)| a <- ags ] [0]
```

Example

```
AM>  displayS5 (up lai2  (public p))
[(1,0),(2,0),(3,0),(4,0),(5,0),(6,0),(7,0),(8,0),(9,0),(10,0
[((1,0),[p]),((2,0),[p]),((3,0),[p]),((4,0),[p]),((5,0),[p])
(a,[[(1,0),(2,0),(3,0),(4,0),(5,0),(6,0),(7,0),(8,0),(9,0),(
(b,[[(1,0),(2,0),(3,0),(4,0),(5,0),(6,0),(7,0),(8,0),(9,0),(
(c,[[(1,0),(2,0),(3,0),(4,0),(5,0),(6,0),(7,0),(8,0),(9,0),(
[(10,0)]
```

```
AM>  displayS5 (upd lai2  (public p))
[0]
[(0,[p])]
(a,[[0]])
(b,[[0]])
(c,[[0]])
[0]
```

Public Announcement, Group Announcement, Private Message, Test

- All special cases of Group Announcements.
- Public Announcement: Group Announcement to All Agents
- Private Message: 'Group' Announcement to a Single Agent
- Test: Group Announcement to the Empty Group

Homework . . .

Implement group announcement by means of a suitable action model.

Tomorrow

- Adding Factual Change
- Wise Men Puzzle
- Muddy Children Puzzle
- ...

References

[1] A. Baltag, L.S. Moss, and S. Solecki. The logic of public announcements, common knowledge, and private suspicions. Technical report, Dept of Cognitive Science, Indiana University and Dept of Computing, Oxford University, 2003.

[2] J. van Benthem, J. van Eijck, and B. Kooi. Logics of communication and change. Under submission, 2005. Available from www.cwi.nl/~jve/papers/05/lcc/.

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

[4] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison Wesley Professional, 1995.

[5] J.E.Hopcroft. An n log n algorithm for minimizing states in a finite automaton. In Zvi Kohavi and Azaria Paz, editors, Theory of Machines and Computations. Academic Press, 1971.

[6] Robert Paige and Robert E. Tarjan. Three partition refinement algorithms. SIAM J. Comput., 16(6):973–989, 1987.