Modelling the Epistemics of Communication with Functional Programming

Jan van Eijck[1,2] and Simona Orzan[3]

[1] CWI, Amsterdam (jve@cwi.nl)
[2] Uil-OTS, Utrecht
[3] TUE, Eindhoven (s.m.orzan@tue.nl)

Abstract

Dynamic epistemic logic is the logic of the effects of epistemic actions like making public announcements, passing private messages, revealing secrets, telling lies. This paper takes its starting point from the version of dynamic epistemic logic of [4], and demonstrates a tool that can be used for showing what goes on during a series of epistemic updates: the dynamic epistemic modeling tool DEMO [10]. DEMO allows modeling epistemic updates, graphical display of update results, graphical display of action models, formula evaluation in epistemic models, and translation of dynamic epistemic formulas to PDL [22] formulas. DEMO is written in Haskell. This paper intends to demonstrate its usefulness for visualizing the model transformations that take place during epistemic updating.

Project paper, on the application of functional programming in a new area.

1 INTRODUCTION

Analysis of multi-agent communication, in the spirit of [13], consists of representing the knowledge or beliefs of the agents in a semantic model, representing the operations on the knowledge or beliefs of the agents as operations on semantic models, and do model checking to see if given formulas are true in the models that result from given updates. After the advances in dynamic epistemic logic documented in [21, 12, 3, 4], taking a model checking approach to epistemic dynamics is more attractive than ever. In this paper we introduce DEMO, a model checking tool written in Haskell and based on the streamlined version of dynamic epistemic logic taken from [4].

DEMO represents epistemic models as objects of type `EpistM` and update actions (pointed action models) on epistemic models as objects of type `PoAM`. Update operations are specified as

```
upd ::    EpistM -> PoAM -> EpistM
upds ::   EpistM -> [PoAM] -> EpistM
```

Here `upd` defines an update with a single update action and `upds` an update with a sequence of update actions. The updates generate new epistemic models. Formula checking is defined as

```
isTrue :: EpistM -> Form -> Bool
```

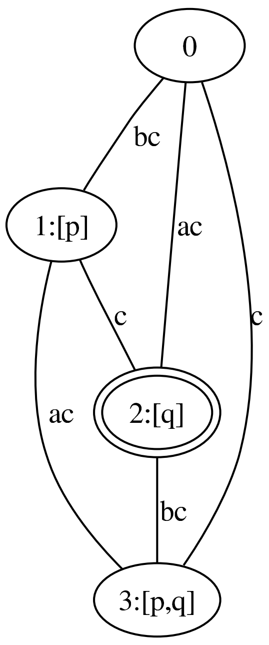
The formula evaluator `isTrue` takes an epistemic model and an epistemic formula as arguments, and returns the truth value of the formula in the model.

Here is the lay-out of the pages that follow. Section 2 provides the background on epistemic logic that is needed for understanding what goes on in the rest of the paper. Sections 3, 4 and 5 discuss examples of epistemic modeling with DEMO, and the final Section 6 concludes and lists further work.

2 EPISTEMIC MODELS, ACTION MODELS AND UPDATING

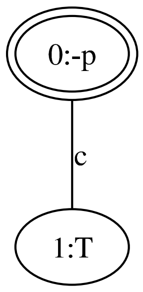
A model for representing the state of knowledge among a group of agents is a labeled transition system (LTS) with labels for the individual agents, and valuations for the states. It is common to call the states *worlds* and to refer to the LTSs as epistemic models or Kripke models [14, 6, 11].

In a situation where one of us (a) knows about p and the other (b) knows about q, while you (c) know nothing about either p or q, and while in fact p happens to be false and q true, the state of knowledge of the agents a, b, c is represented by a Kripke model where the worlds are the four different possibilities for the truth of p and q ($\emptyset$, p, q, pq), the epistemic accessibility relation $\sim_a$ is the relation that links the two worlds where p is true and the two worlds where p is false, the epistemic accessibility relation $\sim_b$ is the relation that links the two worlds where q is true and the two worlds where q is false, and the epistemic accessibility relation $\sim_c$ is the total relation on the set of worlds. This situation can be visualized as follows (this and the following pictures were generated by DEMO with the graphic visualization tool *dot* [17]):



Epistemic formula $K_a\phi$ evaluates to *true* in a world in such a model if in that world every a-accessible world makes ϕ true. In the actual world of the example picture, indicated by the double oval, $K_a\neg p$ is true, for $\neg p$ is true in worlds 0 and 2, and these are the two worlds that are a-accessible from the actual world 2 (we assume that every world is self-accessible). On the other hand, K_aq is false in world 2, for q is true in world 2 but false in world 0. K_bq is true in the actual world. More subtly, $K_a(K_bq \vee K_b\neg q)$ is true the actual world, for it happens to be the case that K_bq is true in world 2 and $K_b\neg q$ is true on world 0. The examples with embedded knowledge operators illustrate how the Kripke models encode information about what agents know about the knowledge or ignorance of other agents. In the example, a does not know about q, but a knows that b knows whether q. Also, all agents know that c is ignorant about p and q.

Epistemic updates are themselves also a kind of Kripke models, with the important difference that the worlds do not carry a valuation but a precondition formula [3]. Here is an example of a model of a group message to a,b that $\neg p$ is the case:



What this expresses is that in fact $\neg p$ is communicated (indicated by the double oval), but that agent c cannot distinguish this communication from a trivial communication $\top$.

Technically, 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 with the accessibility relations holding between pairs (w,u) and (w',u') just in case they hold both between w and w' and between u and u'. Further details are in [3, 4].

This product construction causes an exponential blow-up, and in order to model the update process in a feasible way we need to minimize the update results modulo bisimulation [15]. For this, DEMO uses the following algorithm for partition refinement, in the spirit of [19]:

- Start out with a partition of the state set where all states with the same precondition function are in the same class. The equality relation to be used to evaluate the precondition function is given as a parameter to the algorithm.

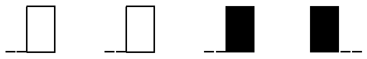
- Given a partition Π, for each block b in Π, partition b into sub-blocks such that two states s,t of b are in the same sub-block iff for all agents a it holds that s and t have $\xrightarrow{a}$ transitions to states in the same block of Π. Update Π to Π' by replacing each b in Π by the newly found set of sub-blocks for b.

- Halt as soon as $\Pi = \Pi'$.

DEMO implements epistemic formula evaluation in update results, for a wide class of epistemic logics (multimodal epistemic logic, epistemic PDL, epistemic PDL with action modalities).

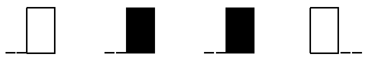
3 THE RIDDLE OF THE CAPS

Picture a situation of four people a,b,c,d standing in line, with a,b,c looking to the left, and d looking to the right. a can see no-one else; b can see a; c can see a and b, and d can see no-one else. They are all wearing caps, and they cannot see their own cap. If it is common knowledge that there are two white and two black caps, then in the following situation c knows what color cap she is wearing.



If c now announces that she knows the color of her cap (without revealing the color), b can infer from this that he is wearing a white cap, for b can reason as follows: "c knows her color, so she must see two caps of the same color. The cap I can see is white, so my own cap must be white as well." In this situation b draws a conclusion from the fact that c knows her color.

In the following situation b can draw a conclusion from the fact that c does not know her color.



In this case c announces that she does not know her color, and b can infer from this that he is wearing a black cap, for b can reason as follows: "c does not know her color, so she must see two caps of different colors in front of her. The cap I can see is white, so my own cap must be black."

To account for this kind of reasoning, we use model checking for epistemic updating, as follows (the Haskell code for this example is given in Figure 1). Proposition p_i expresses the fact that the i-th cap, counting from the left, is white. Thus, the facts of our first example situation are given by $p_1 \wedge p_2 \wedge \neg p_3 \wedge \neg p_4$, and those of our second example by $p_1 \wedge \neg p_2 \wedge \neg p_3 \wedge p_4$.

An initial situation with four agents a,b,c,d and four propositions p_1, p_2, p_3, p_4, with exactly two of these true, where no-one knows anything about the truth of the propositions, and everyone is aware of the ignorance of the others, is modeled like this:

```
Caps> showM mo0
==> [5,6,7,8,9,10]
[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]
```

```
module Caps
where
import List
import DEMO

capsInfo :: Form
capsInfo = Disj [Conj [f, g, Neg h, Neg j] |
                       f <- [p1, p2, p3, p4],
                       g <- [p1, p2, p3, p4] \\ [f],
                       h <- [p1, p2, p3, p4] \\ [f,g],
                       j <- [p1, p2, p3, p4] \\ [f,g,h],
                       f < g, h < j                     ]

awarenessFirstCap  = info [b,c] p1
awarenessSecondCap = info [c]   p2

cK =  Disj [K c p3, K c (Neg p3)]
bK =  Disj [K b p2, K b (Neg p2)]

mo0  = upd (initE [P 1, P 2, P 3, P 4]) (test capsInfo)
mo1  = upd mo0 (public capsInfo)
mo2  = upds mo1 [awarenessFirstCap, awarenessSecondCap]
mo3a = upd mo2 (public cK)
mo3b = upd mo2 (public (Neg cK))
```

FIGURE 1. Haskell code for the caps example.

```
(0,[])(1,[p1])(2,[p2])(3,[p3])(4,[p4])
(5,[p1,p2])(6,[p1,p3])(7,[p1,p4])(8,[p2,p3])(9,[p2,p4])
(10,[p3,p4])(11,[p1,p2,p3])(12,[p1,p2,p4])(13,[p1,p3,p4])(14,[p2,p3,p4])
(15,[p1,p2,p3,p4])
(a,[[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]])
(b,[[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]])
(c,[[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]])
(d,[[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]])
```

The first line indicates that worlds $5,6,7,8,9,10$ are compatible with the facts of the matter (the facts being that there are two white and two black caps). E.g., 5 is the world where *a* and *b* are wearing the white caps. The second line lists all the possible worlds; there are 2^4 of them, since every world has a different valuation. The third through sixth lines give the valuations of worlds. The last four lines represent the accessibility relations for the agents. All accessibilities are total relations, and they are represented here as the corresponding partitions on the set of worlds. Thus, the ignorance of the agents is reflected in the fact that for all of them all worlds are equivalent: none of the agents can tell any of them apart.

The information that two of the caps are white and two are black is expressed by the formula

$$\begin{aligned} & (p_1 \wedge p_2 \wedge \neg p_3 \wedge \neg p_4) \vee (p_1 \wedge p_3 \wedge \neg p_2 \wedge \neg p_4) \vee (p_1 \wedge p_4 \wedge \neg p_2 \wedge \neg p_3) \\ \vee \quad & (p_2 \wedge p_3 \wedge \neg p_1 \wedge \neg p_4) \vee (p_2 \wedge p_4 \wedge \neg p_1 \wedge \neg p_3) \vee (p_3 \wedge p_4 \wedge \neg p_1 \wedge \neg p_2). \end{aligned}$$

A public announcement with this information has the following effect:

```
Caps> showM (upd mo0 (public capsInfo))
==> [0,1,2,3,4,5]
[0,1,2,3,4,5]
(0,[p1,p2])(1,[p1,p3])(2,[p1,p4])(3,[p2,p3])(4,[p2,p4])
(5,[p3,p4])
(a,[[0,1,2,3,4,5]])
(b,[[0,1,2,3,4,5]])
(c,[[0,1,2,3,4,5]])
(d,[[0,1,2,3,4,5]])
```

Let this model be called `mo1`. The representation above gives the partitions for all the agents, showing that nobody knows anything. A perhaps more familiar representation for this multi-agent Kripke model is given in Figure 2. In this picture, all worlds are connected for all agents, all worlds are compatible with the facts of the matter (indicated by the double ovals).

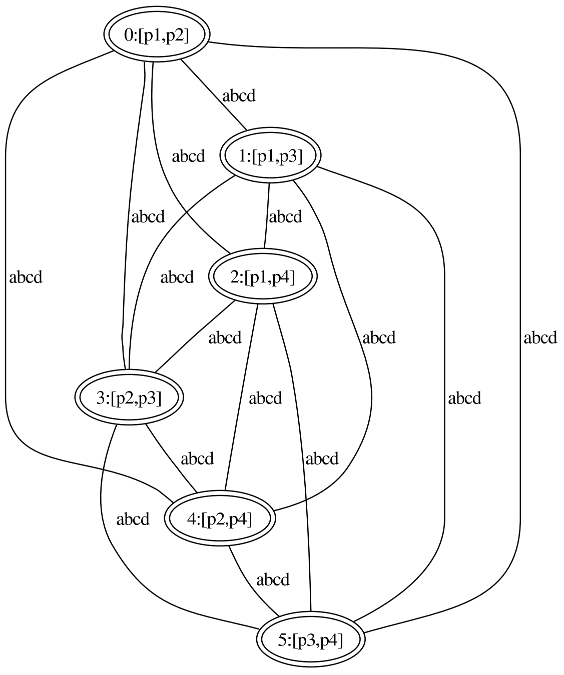


FIGURE 2. Caps situation where nobody knows anything about p_1, p_2, p_3, p_4.

Next, we model the fact that (everyone is aware that) b can see the first cap and that c can see the first and the second cap, as follows:

```
Caps> showM (upds mo1 [info [b,c] p1, info [c] p2])
==> [0,1,2,3,4,5]
[0,1,2,3,4,5]
(0,[p1,p2])(1,[p1,p3])(2,[p1,p4])(3,[p2,p3])(4,[p2,p4])
(5,[p3,p4])
(a,[[0,1,2,3,4,5]])
(b,[[0,1,2],[3,4,5]])
(c,[[0],[1,2],[3,4],[5]])
(d,[[0,1,2,3,4,5]])
```

Notice that this model reveals that in case a,b wear caps of the same color (situations 0 and 5), c knows the color of all the caps, and in case a,b wear caps of different colors, she does not (she confuses the cases $1,2$ and the cases $3,4$). Figure 3 gives a picture representation.

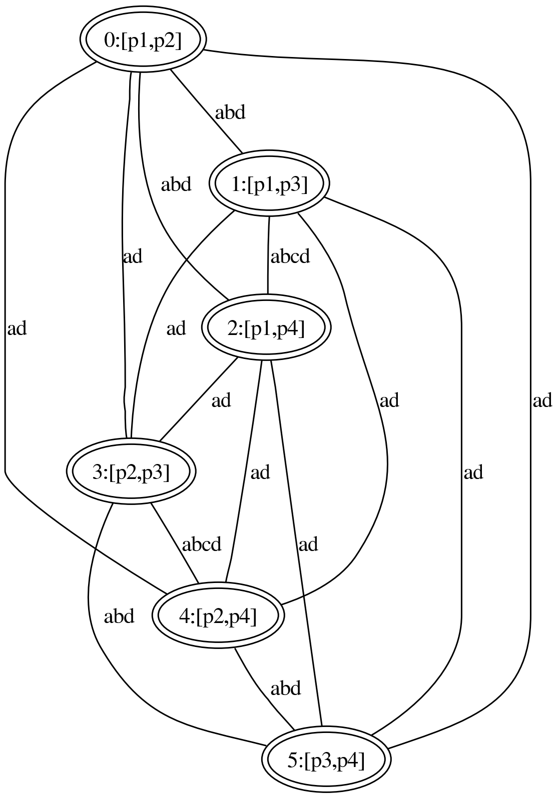


FIGURE 3. Caps situation after updating with awareness of what *b* and *c* can see.

Let this model be called `mo2`. Knowledge of c about her situation is expressed by the epistemic formula $K_c p_3 \vee K_c \neg p_3$, ignorance of c about her situation by the negation of this formula. Knowledge of b about his situation is expressed by $K_b p_2 \vee K_b \neg p_2$. Let bK, cK express that b, c know about their situation. Then updating with public announcement of cK and with public announcement of the negation of this have different effects:

```
Caps> showM (upd mo2 (public cK))
==> [0,1]
[0,1]
(0,[p1,p2])(1,[p3,p4])
(a,[[0,1]])
(b,[[0],[1]])
(c,[[0],[1]])
(d,[[0,1]])

Caps> showM (upd mo2 (public (Neg cK)))
==> [0,1,2,3]
[0,1,2,3]
(0,[p1,p3])(1,[p1,p4])(2,[p2,p3])(3,[p2,p4])
(a,[[0,1,2,3]])
(b,[[0,1],[2,3]])
(c,[[0,1],[2,3]])
(d,[[0,1,2,3]])
```

In both results, b knows about his situation, though:

```
Caps> isTrue (upd mo2 (public cK)) bK
True
Caps> isTrue (upd mo2 (public (Neg cK))) bK
True
```

4 ELEMENTS OF SECURE COMMUNICATION

Since the main concerns in security protocols are keeping, communicating and discovering secrets, security protocol analysis is a new very promising application area for epistemic logic. Attempts in this direction have been made [1, 2, 16], but the field is only now truly emerging and DEMO can offer serious support. Examples of security protocols can be found in [18, 20]; an example of verification by model checking, but without taking reasoning about knowledge into account is [23]. In this section, we will discuss the DEMO modeling of a few basic elements occurring in security protocols.

Keys and nonces as propositional variables The first difficulty encountered when trying to model security situations as epistemic models is that the data exchanged in security protocols are not propositional variables, but essentially very large numbers. However, we argue that they can be captured by propositional variables, under the restriction that guessing is excluded. For look at the kind of statements that we want to make about a key K. What we need to express are statements like "agent A knows K", "agent A doesn't know K" and "agent A sends key K to agent B".

This boils down to expressing doubt and certainty about key K. Let us adopt the convention that an agent can never guess anything about K, that is she can never make a statement about K unless she knows K. Then let a propositional variable p express the truth value of a hashing statement, like for instance "K is even". Then, provided that guessing is excluded, we have a two-way correspondence between the epistemic statement "agent A knows the value of p" and the security statement "agent A knows K".

In modeling a protocol, the convention allows us to represent every key or nonce (parameter varying with time, such as a time stamp) involved in the protocol by its propositional hash.

Public/secret key cryptography Many security protocols assume a public/secret key infrastructure (PKI), meaning that each agent owns a pair of keys, a *public* one, available to everybody, and a *secret* one, only known to the agent herself. These keys are employed in encryption/decryption algorithms that match each other, i.e. a content encrypted with a public key can only be decrypted with the corresponding secret key, and the other way around. Therefore, when an agent a wants to send a message that only an agent b should be able to read, all she has to do is encrypt the message using b's public key.

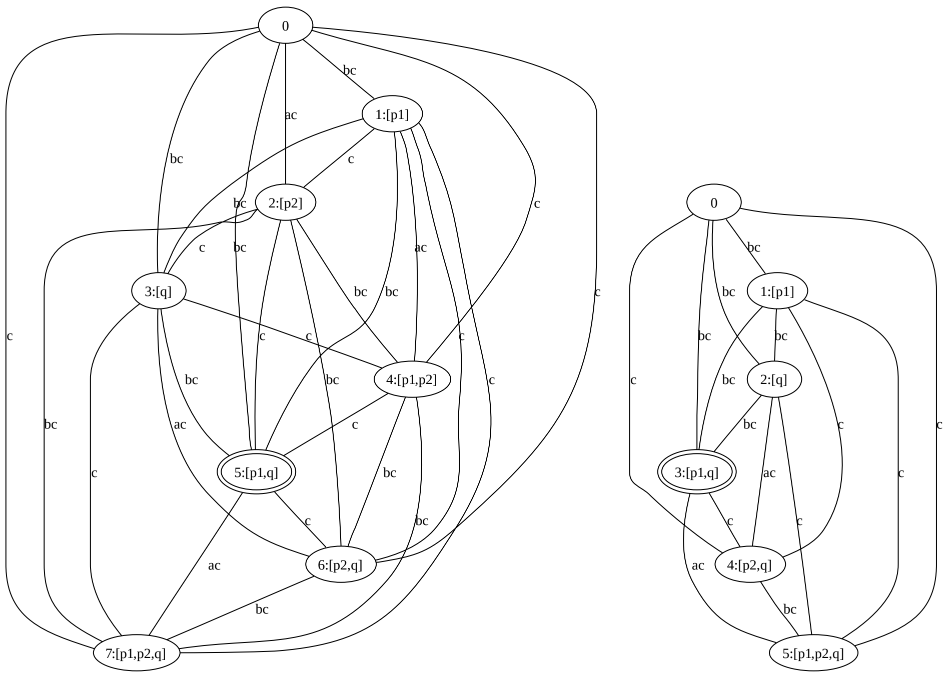


FIGURE 4. The epistemic state of agents a,b,c before and after a has sent the message q encrypted with the public key of b.

To see how this basic mechanism can be modeled in DEMO, we consider two agents a,b trying to communicate as described above, and an agent c, the eavesdropper, who intercepts all the messages. Let p_1,p_2 represent the secret keys of a and b, respectively, and let us fix their truth values to $p_1 = \top$, $p_2 = \bot$. It is not convenient to model the corresponding public keys explicitly; instead, we model the encryption algorithms for the two parties as the implications $p_1 \Rightarrow x$ and $\neg p_2 \Rightarrow x$ (where x is the content to be encrypted). Finally, let q be the message that a wishes to send to b. The situation when a is the only one who knows p_1 and q, b the only one knowing p_2 and all agents are aware of who knows what is described in DEMO as follows:

```
ms1 = upds (initE [P 1, P 2, Q 0])
 [test (Conj [p1,(Neg p2),q]),info [b] p2,info [a] p1,info [a] q]
```

Sending the encrypted message is modeled by the update step

```
ms2 = upd ms1 (public ((Neg p2) ‘impl‘ q))
```

$\neg p_2 \Rightarrow q$ is the result of applying b's public encryption algorithm to the content q. The public communication expresses the fact that c eavedrops. The generated visual representations of `ms1` and `ms2` are shown in Figure 4.

In `ms2`, it can be checked that b has learned q, while c hasn't:

```
SecCom> isTrue ms2 (Disj [K b q, K b (Neg q)])
True
SecCom> isTrue ms2 (Disj [K c q, K c (Neg q)])
False
```

It can also be verified in `ms2` that the encryption algorithm works as a blackbox, that is that a and b do not learn each other's secret key while encrypting/decrypting:

```
SecCom> isTrue ms2 (Disj [K a p2, K a (Neg p2)])
False
SecCom> isTrue ms2 (Disj [K b p1, K b (Neg p1)])
False
```

By replacing `info` with `secret` in the definition of `ms1`, it is also possible to model the fact that a doesn't actually know whether b indeed has b's secret key. This is a relevant subtlety in, for instance, authentication protocols.

Secret communication over insecure channels Another important element of secure communication is the ability to pass information from a to b along an insecure channel in such a way that an eavesdropper c cannot find out what b has learned. A particular case of that is the Russian Card Problem [7, 9], but here we will take a more abstract look at the problem.

If agent a and b share a link between propositions p and q and a and b are the only ones with this link, then a can inform b in secret about q by means of a public communication about p.

Assume there are three agents a, b, c. A situation where all agents are ignorant about *p* and *q*, and are aware of their common ignorance looks like this:

```
SecCom> showM mo0
==> [0,1,2,3]
[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]])
```

Suppose *a* has information about *p*, and *a* and *b* either have common knowledge that *p* and *q* are equivalent or they have common knowledge that *p* and ¬*q* are:

```
SecCom> showM (upds mo0 [info [a] p,link_ab_pq])
==> [0,5,8,9]
[0,1,2,3,4,5,6,7,8,9,10,11]
(0,[])(1,[])(2,[])(3,[p])(4,[p])
(5,[p])(6,[q])(7,[q])(8,[q])(9,[p,q])
(10,[p,q])(11,[p,q])
(a,[[0],[1,6],[2,7],[3,10],[4,11],[5],[8],[9]])
(b,[[0,9],[1,3,6,10],[2,4,7,11],[5,8]])
(c,[[0,1,3,6,9,10],[2,4,5,7,8,11]])
```

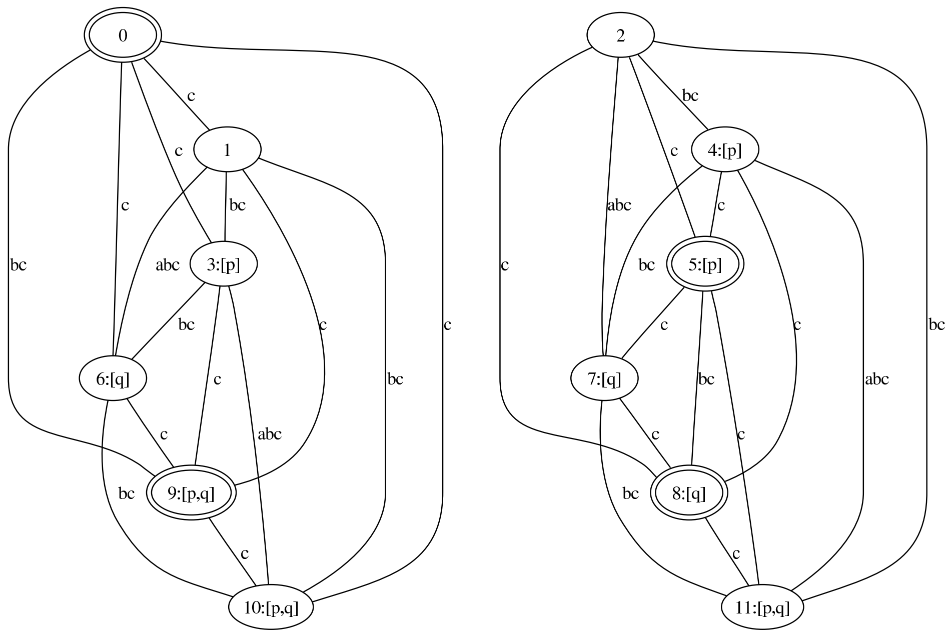


FIGURE 5. Situation where a knows whether p and where a, b have common knowledge about a link between p and q.

Call this model `mo1` (see Figure 5 for an alternative representation). In this model, c still knows nothing about q:

```
SecCom> isTrue mo1 (Disj [K c q,K c (Neg q)])
False
```

Also, in this model it is common knowledge among a, b that b knows about the link between p and q. Thus, the result of public announcement of p in this situation is that b knows whether q, while c is still left in the dark about q:

```
SecCom> showM (upd mo1 (public p))
==> [2,3]
[0,1,2,3,4,5]
(0,[p])(1,[p])(2,[p])(3,[p,q])(4,[p,q])
(5,[p,q])
(a,[[0,4],[1,5],[2],[3]])
(b,[[0,4],[1,5],[2],[3]])
(c,[[0,3,4],[1,2,5]])

SecCom> isTrue (upd mo1 (public p)) (Disj [K b q, K b (Neg q)])
True
SecCom> isTrue (upd mo1 (public p)) (Disj [K c q, K c (Neg q)])
False
```

5 THE PROTOCOL OF THE DINING CRYPTOGRAPHERS

The setting of Chaum's dining cryptographers protocol [5] is a situation where three cryptographers are eating out. At the end of the dinner, they are informed that the bill has been paid, either by one of them, or by NSA (the National Security Agency). Respecting each others rights to privacy, they want to find out whether NSA paid or not, in such a way that in case one of them has paid the bill, the identity of the one who paid is not revealed to the two others.

They decide on the following protocol. Each crypographer tosses a coin with his righthand neighbour, with the result of the toss remaining hidden from the third person. Each cryptographer then has a choice between two public announcements: that the coins that she has observed agree or that they disagree. If she has not paid the bill she will say that they agree if the coins are the same and that they disagree otherwise; if she has paid the bill she will say the opposite: she will say that they agree if in fact they are different and she will say that they disagree if in fact they are the same. Clearly, if everyone is speaking the truth, the number of 'disagree' announcements will be even. This reveals that NSA has picked up the bill. If one person is lying, the number of 'disagree' announcements will be odd, indicating that one among them is paying.

Model checking of this protocol in terms of process theory is described in [24]. In this approach, every aspect of the situation is modelled as a process: the process for coins is defined in terms of processes for heads and for tails, the process for cryptographers following the protocol is defined in terms of their behaviour, and finally the process for the meal is compoosed from the processes for coins and for cryptographers. The correctness specification is captured in a process that outputs ‘crypt’ if a cryptographer pays and ‘nfa’ if NFA pays. After encoding these processes, a model checker confirms that the process for the whole system is indeed a refinement of the specification, and, thus, that it meets the specification.

An epistemic model checking approach is much more straightforward. One starts with an epistemic situation where the diners have common knowledge of the fact that either NSA or one of them has paid. Next, one updates with the result of the coin tosses, and with communicative acts representing the sharing of information between a cryptographer and his neighbour about these results. Assume *b* has in fact picked up the bill.

```
module DC
where
import DEMO

zero_or_one_payer = Disj [
  Conj [Neg(p1), Neg(p2), Neg(p3)], Conj [Neg(p1), Neg(p2), p3],
  Conj [Neg(p1), p2, Neg(p3)], Conj [p1, Neg(p2), Neg(p3)] ]

xor :: Form -> Form -> Form
xor x y = Neg (equiv x y)

-- at most one cryptographer pays, and this is public info
dc1 = upds (initE [P 1, P 2, P3, Q 1, Q 2, Q 3])
          [test zero_or_one_payer, public zero_or_one_payer]

-- let’s say that crypt. b is paying (p2 true)
dc2 = upd dc1 (test ( Conj [Neg(p1), p2, Neg(p3)] ))
dc3 = upds dc2 [info [a] p1, info [b] p2, info [c] p3 ]

-- now the coins get flipped (scenario T,T,F)
flip_coins = Conj [q1, q2, Neg (q3) ]
dc4 = upd dc3 (test flip_coins)
dc5 = upds dc4 [info [a,b] q1, info [b,c] q2, info [a,c] q3 ]

a_says = (xor (xor q1 q3) p1)
b_says = (xor (xor q1 q2) p2)
c_says = (xor (xor q2 q3) p3)

-- and the results get announced
dc6 = upds dc5 [public a_says, public b_says, public c_says]
```

FIGURE 6. Haskell code for the Dining Cryptographers example.

For $i \in \{1,2,3\}$, let p_i be the proposition "cryptographer i is the payer". The aim of the protocol is that everybody learns whether the formula $p_1 \lor p_2 \lor p_3$ is true or not, but if the formula is true, nobody (except the payer herself) learns which of the three propositions was true. To model the protocol, we need three more propositions q_1,q_2,q_3 representing the result of flipping the coins shared by p_1 and p_2, p_2 and p_3, p_3 and p_1, respectively.

The three phases of the protocol — initialization, flipping the coins, announcing the results — are captured by the epistemic models `dc1` up to `dc6`, defined in the listing from Figure 6. The restriction that one or none of the cryptographers pays is modeled by the public announcement `zero_or_one_payer`. For simplicity of exposition, we fix a paying scenario (*b* pays, i.e. $p_2 = \top$) and a coin flipping situation $q_1 = \top, q_2 = \top, q_3 = \bot$.

Figure 7 shows the final epistemic state, `dc6`. Space does not permit us to list or display the (rather large) intermediate states. The textual representation of `dc6` is:

```
DC> showM dc6
==> [4]
[0,1,2,3,4,5]
(0,[p1,q2])(1,[p2,q3])(2,[p3,q1])(3,[p1,q1,q3])(4,[p2,q1,q2])
(5,[p3,q2,q3])
(a,[[0],[1,5],[2,4],[3]])
(b,[[0,5],[1],[2,3],[4]])
(c,[[0,4],[1,3],[2],[5]])
```

Here, world 4 is the actual world. This is the world where *b* has paid and where q_1 and q_2 have value $\top$. As the accessibility relations show, *a* cannot distinguish the actual world from world 2 (a world where *c* has paid), and *c* cannot distinguish the actual world from world 0 (a world where *a* has paid).

The most important properties to be checked in the final state are the fact that everybody learned $p_1 \lor p_2 \lor p_3$ and that *a* and *c* don't know that *b* was the payer:

```
DC> isTrue dc6 (CK [a,b,c] (Disj [p1,p2,p3]))
True
DC> isTrue dc6 (Conj [Neg(K a p2), Neg(K c p2)])
True
```

6 FURTHER WORK

As the program listings demonstrate, the DEMO representations of epistemic situations are very concise. A comparison between DEMO and two other model checking tools for epistemic update logic can be found in [9]: "The fastest goal to success was implementing the Russian Cards problem in DEMO."

At present DEMO is being used in various places around the world for model checking of problems in epistemic update logic [8, 9]. Example code from these

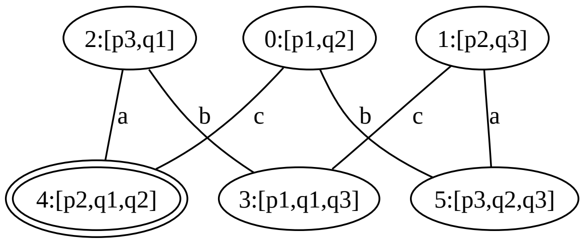


FIGURE 7. The final epistemic state of the DC protocol.

analyses is available as part of the DEMO documentation. Further applications in the area of analysis of security protocols are in preparation.

The current implementation of DEMO does not use monads, but a new implementation in terms of state monads is in the making. This will allow dynamic updating, with the current result of all updates so far held in the state, and it will hopefully also increase the efficiency of the tool, since some operations, like reduction under bisimulation, have efficient imperative implementations.

Acknowledgments Thanks to Hayco de Jong for illuminating comments on an earlier draft, and to Thijs van der Storm for advice on the use of *dot*.

Code availability The Haskell code for DEMO and for the DEMO examples is available from http://www.cwi.nl/ jve/demo/.

REFERENCES

[1] A.Bleeker and J. van Eijck. Epistemic action and change. In *LOFT-4 Proceedings*, 2000.

[2] A. Baltag. Logics for insecure communication. In *TARK '01: Proceedings of the 8th conference on Theoretical aspects of rationality and knowledge*, pages 111–121, 2001.

[3] 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.

[4] 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/.

[5] D. Chaum. The dining cryptographers problem: unconditional sender and receiver untraceability. *Journal of Cryptology*, 1:65–75, 1988.

[6] B.F. Chellas. *Modal Logic: An Introduction*. Cambridge University Press, 1980.

[7] Hans van Ditmarsch. The Russian card problem. *Studia Logica*, 75:31–62, 2003.

[8] Hans van Ditmarsch, Ji Ruan, and Rineke Verbrugge. Model checking sum and product. Technical report, Computer Science Department, University of Otago, New Zealand, 2005.

[9] Hans van Ditmarsch, Wiebe van der Hoek, Ron van der Meyden, and Ji Ruan. Model checking Russian cards. To appear in Proceedings of MoChArt 05, 2005.

[10] Jan van Eijck. Dynamic epistemic modelling. Technical Report SEN-E0424, CWI, Amsterdam, December 2004. Available from http://db.cwi.nl/rapporten/.

[11] R. Fagin, J.Y. Halpern, Y. Moses, and M.Y. Vardi. *Reasoning about Knowledge*. MIT Press, 1995.

[12] J. Gerbrandy. *Bisimulations on planet Kripke*. PhD thesis, ILLC, 1999.

[13] Joseph Y. Halpern and Moshe Y. Vardi. Model checking vs. theorem proving: A manifesto. In J. Allen, R. E. Fikes, and E. Sandewall, editors, *Proceedings 2nd Int. Conf. on Principles of Knowledge Representation and Reasoning, KR'91*, pages 325–334. Morgan Kaufmann Publishers, San Mateo, CA, 1991.

[14] J. Hintikka. *Knowledge and Belief: An Introduction to the Logic of the Two Notions*. Cornell University Press, Ithaca N.Y., 1962.

[15] M. Hollenberg. *Logic and Bisimulation*. PhD thesis, Utrecht University, 1998.

[16] A. Hommersom, J.-J. Meyer, and E.P. de Vink. Update semantics of security protocols. *Synthese*, 142:229–267, 2004. Knowledge, Rationality and Action subseries.

[17] E. Koutsofios and S. North. Drawing graphs with *dot*. Available from http://www.research.att.com/ north/graphviz/.

[18] Gavin Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 1055, pages 147–166. Springer-Verlag, Berlin Germany, 1996.

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

[20] O. Pereira. Modelling and security analysis of authenticated group key agreement protocols, 2003.

[21] J. A. Plaza. Logics of public communications. In M. L. Emrich, M. S. Pfeifer, M. Hadzikadic, and Z. W. Ras, editors, *Proceedings of the 4th International Symposium on Methodologies for Intelligent Systems*, pages 201–216, 1989.

[22] V. Pratt. Application of modal logic to programming. *Studia Logica*, 39:257–274, 1980.

[23] A. W. Roscoe. Modelling and verifying key-exchange protocols using CSP and FDR. In *Proc. 8th IEEE Computer Security Foundations Workshop*, pages 98–107, 1995.

[24] S. Schneider and A. Sidiropoulos. CSP and anonymity. In *Proc. ESORICS'96*, pages 198–218. LNCS 1146, 1996.