Epistemic Logic in Higher Order Logic
An experiment with COQ

Pierre Lescanne

Laboratoire de l'Informatique du Parallélisme, École Normale Supérieure de Lyon
46, Allée d'Italie, 69364 Lyon 07, FRANCE
E-mail: `Pierre.Lescanne@ens-lyon.fr`

Abstract. We present an experiment on epistemic logic, also called knowledge logic, we have done using COQ. This work involves a formalization in COQ of the epistemic logic which has been checked for adequacy on two puzzles well known in the community. COQ is a proof assistant which implements a higher logic known as the calculus of inductive construction and which provides a convenient framework to embed logics like epistemic logic. We try to draw from this exercise lessons for future works. We also expect the framework we have built to be useful for people who wish to formalize cryptographic protocols.

1 Introduction

Epistemic logic is one of the formal basis of cryptographic protocols. It is the logic which formalizes the *knowledge*, but also the *belief* of agents. Therefore, at a time when formal methods is part of the requirements of the certification of cryptographic protocols [3] this contribution can be useful. Among other applications, belief logic is the foundation of the BAN [2] and it has been used also by Howell and Kotz [5] to formalize the Simple Public Key Infrastructure (SPKI). In our implementation we consider in addition *common knowledge* which are statements that are taken as "really true" by a group of agents, this can serve as the foundation of authentication.

We have chosen to embed epistemic logic into type theory as implemented in the proof assistant COQ [1]. There are many reasons for that. COQ which is based on the *calculus of inductive constructions* offers a very general tool for representing logic theories. In COQ, proofs are objects that can be built by a computer aided system and exchanged among researchers. Due to lack of space, we cannot fully introduce COQ, but we hope that we give enough information in this paper for a reader to catch much of the concepts necessary to understand the development of epistemic logic presented here.

Epistemic logic is also known as the *logic of knowledge*, it deals with concepts called *modalities*, which are not part of the classical logic and which modifies the meaning a proposition. For instance such a modality is the *knowledge modality*: "agent Alice knows that ...". There is one knowledge modality K_i by agent i, so when there are n agents, there are n knowledge modalities. From the K_i's, one can build two new modalities, namely a modality E_g of *shared knowledge*, which modifies p into "everyone knows p" when applied to p and a modality C_g of *common knowledge* which would say "p is well known from everybody". Slightly more precisely, if g is the group of agents and p is a proposition, $E_g p$ is the conjunction over the $i \in g$ of the $K_i p$ and $C_g p$ means something like "everybody knows p and everybody knows that everybody knows p and ... and everybody knows that everybody knows that everybody knows ... that everybody knows p..." This infinite conjunction is handled by making $C_g p$ a fix point. One main goal of epistemic logic is to handle properly those concepts. For the reader who wants to know more, [4, 8] are two excellent introductory books to epistemic logic.

The well-known *deduction rule* is as follows: if a statement ψ can be deduced from a set Γ of hypotheses augmented by φ, then the theorem "φ implies ψ" can be deduced from Γ

$$\frac{\Gamma, \varphi \vdash \psi}{\Gamma \vdash \varphi \Rightarrow \psi}$$

Most of the logics, noticeably the calculus of inductive constructions, fulfill the deduction rule, but modal logic and epistemic logic do not, therefore they cannot be represented directly in COQ. Consequently, for a further extension to epistemic logic one has to formalize propositional logic and predicate calculus in an approach à la Hilbert, which involves to embed the calculus as a specific theory in COQ and specifically to define the set of propositions as a `Set` in COQ. The formalization we get eventually is a higher order epistemic logic.

From a mechanical certification point of view, epistemic logic is usually mechanized by model checking [6, 7]. To our knowledge no mechanization of the proof theory of epistemic logic has been done so far. When dealing with examples, we faced the well-known issue lengthly discussed in books and papers [4, 8, 2] of finding the right modeling for the problem of interest; we discuss this about the puzzle of the muddy children in Section 5.

The paper is structured as follows. In Section 2 we describe the implementation of predicate calculus in COQ. In Section 3, we describe modal logic and epistemic logic. Section 4 and Section 5 are devoted to two examples. Section 6 summarizes what we learned from this experiment. The whole development is available on the WEB at `http://www.ens-lyon.fr/~plescann/COQ/EPISTEMIC/`.

2 Predicate calculus à la Hilbert

If we aim to introduce modal logic, we have to present predicate calculus in a framework à la Hilbert. For this, we introduce a type `proposition` which is an inductive `Set`. Its constructors are the implication `Imp` (written `=>` as an infix), the quantifier `Forall` and two modal operators `K` and `C`. Axiomatization for `K` and `C` is given in Section 3

```
Inductive proposition: Set :=
    Imp    :  proposition -> proposition -> proposition |
    Forall :  (A:Set) (A -> proposition) -> proposition |
    K      :  nat -> proposition -> proposition         |
    C      :  (list nat) -> proposition -> proposition.
```

We introduce a predicate `theorem` in the set `proposition` which tells which propositions are theorems. For instance, `(theorem p)` says that proposition `p` is a theorem in the object theory representing epistemic logic.

Propositional Logic. First we introduce axioms for intuitionistic logic only. Classical logic will be introduced later if necessary. The axioms are therefore just:

```
Hilbert_K: (p,q:proposition) (theorem p => q => p)

Hilbert_S: (p,q,r:proposition)
    (theorem (p => q => r) => (p => q) => p => r)
```

plus the *modus ponens* as a rule:

```
MP: (p,q:proposition) (theorem p => q) -> (theorem p) -> (theorem q).
```

Here `->` is the implication in the meta-theory, namely in COQ. In the proposition-as-type approach, `->` is also the type constructor for function spaces. Rather naturally, a rule is a way to deduce a new theorem from one or more previous ones and has the form

(theorem Hyp_1) -> ...(theorem Hyp_n) -> (theorem *Conclusion*).

Predicate Logic. There are two axioms for universal quantification (see for instance [11] p. 68):

```
Forall1: (A: Set)(P:A -> proposition)(a:A)
    (theorem (Forall A P) => (P a))

Forall2: (A: Set)(P:A -> proposition)(q:proposition)
        (theorem (Forall A [x:A](q => (P x))) => q => (Forall A P))
```

and one rule:

```
ForallRule: (A: Set)(P:A->proposition)
    ((x:A)(theorem (P x))) -> (theorem (Forall A P)).
```

The predicates `Forall` whose signature is defined in the definition of `proposition` depends on a set. More precisely, the quantification `Forall` applies to a set `A` and to a predicate, i.e., a function `(A -> proposition)`.

In Forall2, [x:A] (q => (P x)) represents a predicate which depends on x. (x:A) is the universal meta-quantification over the A (i.e., the universal quantification in COQ). ForallRule says that if for each x in A, (P x) is a theorem, then (Forall A P) is a theorem. It sets an interesting connection between the meta-quantification and the quantification in the object theory. This presentation allows us to get read of the machinery for handling free variables and captures.

Other connectors and quantifiers. Usually in intuitionist logic each connector and each quantifier must be defined independently of the others. But as we embed our presentation in higher order logic, one connector (here =>) and one quantifier Forall are enough and the other connectors and quantifiers are defined from these ones:

```
Definition And := [p,q:proposition]
    (Forall proposition [r:proposition] (p => q => r) => r).

Definition Or := [p,q:proposition]
    (Forall proposition [r:proposition] (p => r) => (q => r) => r).

Definition Exist := [A:Set][P: A ->proposition]
    (Forall proposition [p:proposition]
        (Forall A [a:A]((P a) => p)) => p).
```

And is written & and *Or* is written |/.

Lemmas and derived rules. For use in later examples we prove lemmas like

```
Lemma Or_comm: (p,q: proposition) (theorem (p |/ q) => (q |/ p)).
```

which says that |/ is commutative. Often in a proof, we want to reverse the order of the component of a conjunction, for that its companion rule is more convenient:

```
Lemma rule_Or_comm: (p,q: proposition)
    (theorem (p |/ q)) -> (theorem (q |/ p)).
```

When applied it leads to prove theorem p |/ q for goal theorem q |/ p. In our experiments, we noticed that the *Cut Rule* was very handy

```
(p,q,r:proposition)
    (theorem p => q) -> (theorem q => r) -> (theorem p => r).
```

3 Modal Logic and Epistemic Logic

Since modal logic was developed as a part of epistemic logic, we decided to introduce not only one but infinitely many modalities (K i). In COQ this is an easy task. K has the signature nat -> proposition -> proposition. For now on, we restrict ourselves to the logic *T*, for which there are two axioms:

```
Axiom K_K: (i: nat) (p,q:proposition)
    (theorem (K i p) => (K i (p => q)) => (K i q)).

Axiom K_T: (i: nat) (p:proposition) (theorem (K i p) => p).
```

and a rule

```
Axiom K_rule: (i: nat) (p:proposition)
    (theorem p) -> (theorem (K i p))          .
```

On could add other axioms like Barcan formula, which rules the connection between `K` and *Forall*:

```
(i: nat) (A:Set) (P:A->proposition)
        (theorem (forall A [x:A](K i (P x))))
     -> (theorem (K i (forall A P)))
```

Epistemic logic requires to introduce a modality `E` for *shared knowledge*,

```
Fixpoint E [g: (list nat)]: proposition -> proposition :=
        [p:proposition] Cases g of
            nil        =>  TRUE
         | (cons i g1) => (K i p) & (E g1 p)
        end.
```

`E` takes a group `g` of agents and a proposition `p` and returns a proposition `(E g p)`. It is defined *inductively*, i.e., as a fix point on the structure of g.

$$E\ nil\ p = TRUE$$
$$E\ (cons\ i\ g_1\ p) = (K\ i\ p) \quad \& \quad (E\ g_1\ p)$$

`E` enjoys some nice properties we proved in COQ like

```
(g:(list nat); p1,p2:proposition)
    (theorem ((E g p1) & (E g p2)) => (E g (p1 & p2))).
```

C is the modality for *common knowledge*, it is defined by the axiom

```
(g:(list nat)) (p:proposition)
    (theorem (C g p) => (p & (E g (C g p)))).
```

and the rule

```
(g:(list nat)) (p,q:proposition)
    (theorem q => (p & (E g q))) -> (theorem q => (C g p)).
```

We can prove lemmas about `C`, for instance

```
Lemma C_T: (g:(list nat); p:proposition) (theorem (C g p) => p).
Lemma C_CE: (g:(list nat); p:proposition)
    (theorem (C g p) => (C g (E g p))).
```

or fix point properties like

```
(g:(list nat); p:proposition)
    (theorem (p & (C g (E g p))) => (C g p)).
```

4 The king, the three wise men and the five hats

Before addressing examples from the cryptographic protocol area, we decided to tackle classical examples. The first one is the puzzle of the king, the three wise men and their hats. It uses only agent knowledge modalities `(K i)`. In [4], Exercise 1.3, it is presented as *"There are three wise men. It is common knowledge that there are three red hats and two white hats. The king puts a hat on the head of each of the three wise men and asks them (sequentially) if they know the color of the hat on their head. The first wise man says that he does not know; the second wise man says that he does not know; then the third man says that he knows"*. In what follows the wise (wo)men are called agents with names Alice, Bob and Carol. Actually in COQ, Alice, Bob and Carol are taken as abbreviations for `(0)`, `(1)` and `(2)`. The puzzle is based on a function

```
Definition Kh := [i:nat] (K i (white i)) |/ (K i (red i)).
```

which says that the *"agent i knows the color of his hat"*. With a minimal set of hypotheses, we are able to prove

```
(theorem (K Bob (Not (Kh Alice))) & (Not (Kh Bob)) => (red Carol)).
```

In other words, *"If Bob knows that Alice does not know the color of her hat and if Bob himself does not know the color of his hat, the color of the hat on the head of Carol is red."* If `(red Carol)` is provable from the two premises, then Carol knows that fact; therefore if she knows that *if Bob knows that Alice does not know the color of her hat and if Bob himself does not know the color of his hat*, then she knows that the color of her hat and even more (since she knows that the color of her hat is red).

The above involved sentences are typical assertions about knowledge. As they are hard to understand for a human, one is happy to make the computer check them.

What are the assumptions we made? There are five.

- *An agent wears a white hat exclusive or a red one.*

  ```
  (i:nat) (theorem (white i) | (red i)).
  ```

- *There are only two white hats.* Actually we do not need such a general statement. We only have to state that *"If Bob and Carol wear a white hat, then Alice wears a red hat."* which translates in COQ into

  ```
  (theorem ((white Bob) & (white Carol) => (red Alice))).
  ```

  Note that we are not interested by a statement like "If Carol and Alice wear a white hat, then Bob wears a red hat." Moreover the number of red hats is irrelevant.
- *Each agent knows the color of the hat of the two other agents, in some specific events, namely Alice (Bob) knows the color of the hat of Bob and Carol (of Carol) when those (this) hats are (is) white.*

```
(theorem ((white Bob) => (K Alice (white Bob)))).
(theorem ((white Carol) => (K Alice (white Carol)))).
(theorem ((white Carol) => (K Bob (white Carol)))).
```

These hypotheses assert that the agents can be supposed to be in a row Carol, Bob, Alice and that each agent know the color of the hat of the agents before her or him. This is sometime a presentation of this puzzle(see for instance [4] Exercise 1.3 (b)). Actually, we see in our proof, that the fact that the color of a hat is red is of no interest for any agent.

It should be noted that we made actually less hypotheses than in the statement of the puzzle.

The proof. The proof is not too difficult when the assumptions are properly stated, it requires just eight small lemmas and needs only modal logic. The mechanization of the proof shows us that many hypotheses made in classical presentation of this puzzle are redundant. Perhaps a careful human analysis of the problem would have lead to the same hypotheses, but what is interesting in this experiment is that this comes naturally from the mechanical development of the proof. One makes the proof and then one traces the hypotheses one actually uses. For instance, in a first attempt we made much more statements about the knowledge of the agents about the color of the hat of the other agents than actually needed. Afterwards, we removed the useless hypotheses.

5 The muddy children

This problem is considered by Fagin et al. [4] as the illustration of epistemic logic, especially of common knowledge. Let us give the presentation of [8]. *A number, say n, of children are standing in a circle around their father. There are $k(1 \leq k \leq n)$ children with mud on their heads. The children can see each other but they cannot see themselves. In particular, they do not know if themselves have mud on their heads. ... Father says aloud: "There is at least one child with mud on its head. Will all children who know they have mud on their heads please step forward?"... This procedure is repeated until, after the k-th time Father has asked the same question, all muddy children miraculously step forward.* We propose a proof of the correctness of the puzzle under reasonable and acceptable hypotheses. The main question is "What does it mean to say that the children see each other and what consequences do they draw from what they see?" For us, "the children see" means that

- they know whether the other children have mud on their head,
- they notice the children stepping forward or not.

The main interest of the *muddy children* puzzle lies in the use of *common knowledge* (modality `C`).

We define two predicates depending on two naturals, namely `At_least` and `Exactly`. `(At_least n p)` is intended to mean that among the `n` children, there are at least `p` muddy children, whereas `Exactly` means that among the `n` children, there are exactly `p` muddy children. `Exactly` is defined as

```
[n,p:nat] (At_least n p) & (Not (At_least n (S p))).
```

The hypothesis. Suppose we are in the situation where

1. all the children know that there are at least p muddy children
2. by the fact that none of the children stepped forward, all the children know that there are not exactly p muddy children.

Fact 2, namely the knowledge (shared by the group `(list_of n)` which stands for $[0, 1, ...n]$) on the non exactness, comes from the absence of step forward of children. Therefore Fact 2 is known by every children, formally `(K i (E (list_of n) (Not (Exactly n p))))`. Therefore we state the axiom:

```
Axiom Knowledge_Diffusion : (n,p,i:nat)
    (theorem (E (list_of n) (At_least n p))
             => (E (list_of n) (Not (Exactly n p)))
             => (K i (E (list_of n) (Not (Exactly n p))))).
```

From it we prove two lemmas:

```
Lemma E_Awareness : (n,p,k:nat) (le k n) ->
    (theorem (E (list_of n) (At_least n p))
             => (E (list_of n) (Not (Exactly n p)))
      => (E (list_of k) (E (list_of n) (Not (Exactly n p))))).
Lemma C_Awareness : (n,p:nat)
    (theorem (C (list_of (S n)) (At_least (S n) p))
           => (E (list_of (S n)) (Not ((Exactly (S n) p))))
     => ((C (list_of (S n)) (Not (Exactly (S n) p))))).
```

We use these lemmas to prove the main result which shows how the knowledge of the children progresses.

```
(C (list_of (S n)) (At_least (S n) p))
& (E (list_of (S n)) (Not (Exactly (S n) p)))
       => (C (list_of (S n)) (At_least (S n) (S p)))).
```

In other words: *"If this is a common knowledge that there are at least p muddy children and if every child knows that there are not exactly p muddy children then this is a common knowledge that there are at least p+1 muddy children."* Therefore, if for some k, a child knows that there is at least $p+1$ muddy children and if he sees p muddy children, he steps forwards. This is the secret of the miracle.

6 What we learned

The problems we had to solve are of two kinds: those dealing with the organization of the proofs (the praxis) and those dealing with the theory (the proof system).

The organization

With the importance of formal development of proofs, a new discipline is emerging namely *proof engineering* which deals with problems very similar to those of software engineering. For this reason, proof engineering uses tools of software engineering. We identified two main issues.

Notations. We have to cope with abstract concepts the human brain has difficulty to handle. Among others in a real proof one has to handle formulae larger than those of the everyday life of mathematics. Especially, in epistemic logic, where one gets easily lost by the accumulation of *"knows that"*.

Thus a main issue is to adequately *name concepts* to ease the task of the person who puts them together in a large system. For instance, mathematicians prefer infix notations for binary operators and it is very convenient to have the ability to define them. A difficult task is also to assign appropriate names to lemmas, in order to invoke them later on. What we have done so far can surely be improved.

Version control. We realize that the use of *version control system* (like RCS or CVS on Unix) is very handy. In proof engineering, it eases reverting when one regrets the last changes made. In case the development is done by a group, it allows keeping track of versions and changes done by the participants and it enables the group to access the same files. Despite the use of the word "we" in this paper, the development presented here has been done by one person. One advantage we found in the methodology of version control is when we noticed that the proof we are going to perform might go in a wrong direction. We did not hesitate to throw it away in order to try another one and to compare with the previous development.

The proof

Building proofs in a system à la Hilbert is slightly more difficult than in natural deduction as we do not have the ability to introduce hypotheses in the environment, at the level of the theory. The statements one has to prove have to be handled as a whole. Fortunately the use of rules like the modus ponens, the cut rule or the rules specific to modal logic and epistemic logic allows us to organize the proof. One can postpone the proof of some statements of the form `(theorem ...)` and one can divide and conquer proofs. We foresee that some of the tasks of the proof developers can be lightened by tactics to be developed. As one difficulty in building proofs in cryptographic protocols is to state the reasonable and acceptable hypotheses, we notice that we often build proofs of properties backward from the main property we want to prove. Usually there is not so much facility offered by proof assistants, for that. A good approach is to state temporary axioms for the intermediary lemmas and see what can be proved for them and proceed backward, until an acceptable hypothesis is reached.

References

1. Bruno Barras, Samuel Boutin, Cristina Cornes, Judicaël Courant, Yann Coscoy, David Delahaye, Daniel de Rauglaudre, Jean-Christophe Filliâtre, Eduardo Giménez, Hugo Herbelin, Gérard Huet, Henri Laulhère, César Muñoz, Chetan Murthy, Catherine Parent-Vigouroux, Patrick Loiseleur, Christine Paulin-Mohring, Amokrane Saïbi, and Benjamin Werner. *The Coq Proof Assistant Reference Manual.* INRIA, version 6.3.11 edition, May 2000.
2. Michael Burrows, Martín Abadi, and Roger Needham. A logic of authentication. *Proceedings of the Royal Society*, 426:233–271, 1989.
3. The Common Criteria Project Sponsoring Organisations, editor. *Common Criteria for Information Technology Security Evaluation, Part I: Introduction and general model.* ISO/IEC, August 1999.
4. Ronald Fagin, Joseph Y. Halpern, Yoram Moses, and Moshe Y. Vardi. *Reasoning about Knowledge.* The MIT Press, 1995.
5. Jon Howell and David Kotz. A formal semantics for SPKI. In *Proceedings of the Sixth European Symposium on Research in Computer Security (ESORICS 2000)*, pages 140–158. Springer-Verlag, October 2000.
6. Gavin Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using CSP and FDR. In T. Margaria and B. Steffen, editors, *Tools and Algorithms for the Constrcution and Analysis of Systems, TACA'96*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166, 1996.
7. Will Marrero, Edmund Clarke, and Somesh Jha. Model checking for security protocols. Technical Report CMU-CS-97-139, Carnegie Mellon University, 1997.
8. John-Jules Ch. Meyer and Wiebe van der Hoek. *Epistemic Logic for Computer Science and Artificial Intelligence*, volume 41 of *Cambridge Tracts in Theoretical Computer Science.* Cambridge University Press, 1995.
9. Lawrence C. Paulson. The inductive approach to verifying cryptographic protocols. *J. Computer Security*, 6:85–128, 1998.
10. Bruce Schneier. *Applied Cryptography : Protocols, Algorithms, and Source Code in C.* John Wiley and Sons, 1995. 2nd edition.
11. Anne S. Troelstra and Dirk van Dalen. *Constructivism in mathematics*, volume 1. North Holland, 1988.