

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 a mechanization of epistemic logic, also called knowledge logic, we have done using COQ. This work includes a formalization in COQ of epistemic logic and a check of its adequacy on two well known puzzles. We try to draw from this exercise lessons for future works.

1 Introduction

Epistemic logic is the logic which formalizes *knowledge*, but also *belief* of agents. Among many applications it is used in verifying cryptographic protocols. For instance Howell and Kotz [4] use it to formalize the Simple Public Key Infrastructure (SPKI). In our implementation we consider in addition *common knowledge* which are assumptions that are taken as granted by a group of agents, in a rather strong sense (see the discussion below). This can serve as the foundation of authentication or of public key distribution.

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 are built by a sophisticated 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.

Shared knowledge, common knowledge. Epistemic logic is also known as the *logic of knowledge*, it deals with concepts called *modalities*, which are not part of traditional logic and which modify the meaning of a proposition. For instance such a modality is the *knowledge modality*: “agent Alice knows that ...”, written K_{Alice} . 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 a proposition p into a proposition $E_g(p)$ which means that “everyone in the group g knows p ” and a modality C_g of *common knowledge*. $C_g(p)$ would say “ p is well known from everybody in the group g ”. 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 fixed point. A typical example of common knowledge is *traffic regulation*. When, as a car driver, you enter an intersection you know that the person on your left will let you go, moreover you know that she knows that you have the right to go and you are sure (you know) that she will not go because she knows that you know that she knows that you have the right to go etc. Actually you pass an intersection with a car on your left, because there is a common knowledge between you as a driver and the driver of the other car on the rule of priority. But those who travel have experienced the variability of the common knowledge. Take a *stop sign*. In Europe it means that the person which has a stop sign will let the other to pass the intersection. In some countries, the stop sign is just a decoration of intersections. In the USA, the common knowledge is different since there are intersections of two crossing roads with four stop signs and this has puzzled more than one European.

One main goal of epistemic logic is to handle properly those concepts of *knowledge of an agent*, *shared knowledge* and *common knowledge*. For the reader who wants to know more, the two books [3,9] are two excellent introductions to epistemic logic. Rules of epistemic logic are given in the appendix.

Deduction rule and presentation à la Hilbert. 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

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

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. Indeed suppose that we have the deduction rule in epistemic logic (or in modal logic by the way), we would have $p \vdash p$ and then

$$\frac{p \vdash p}{p \vdash K_i p}$$

by the *Knowledge Generalization* rule (see appendix) and then $\vdash p \Rightarrow K_i p$ using the deduction rule which would translate into “*if p holds then agent i knows p* ” which is not what we want in formalizing epistemic logic. Indeed we want agents to know part of the truth not all the truth. 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. This kind of approach is called *deep embedding* and requires a very expressive logic. The formalization we get eventually is a higher order epistemic logic.

Related works From a mechanical certification point of view, epistemic logic is usually mechanized by model checking [6, 8]. The work presented in this paper (see also [5])

is to our knowledge the first presentation of the mechanization of the proof theory of epistemic logic. Concurrently Paulien de Wind has made her own mechanization using COQ based on an extension of natural deduction with several levels [11]. We should also mention a recent attempt done by Mehrnoosh Sadrzadeh of University of Ottawa to formalize modal linear logic [7]. When dealing with examples, we faced the well-known issue lengthily discussed in books and papers [3, 9, 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 outline 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 in COQ is available on the WEB at <http://perso.ens-lyon.fr/pierre.lescanne/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

$$(\text{theorem } Hyp_1) \rightarrow \dots (\text{theorem } Hyp_n) \rightarrow (\text{theorem } Conclusion).$$

Predicate Logic. There are two axioms for universal quantification (see for instance [10] 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 is one rule:

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

The operator (or the quantifier) `Forall` whose signature is part of the definition of proposition depends on a set `A` and builds a proposition from a predicate. In our mechanization in `COQ` a predicate is a function `(A -> proposition)` — notice the use of `->` as a constructor for a function space — and the quantification `Forall` takes a set `A` and a predicate `P` to produce a proposition `(Forall A P)`.

`Forall1` and `Forall12` are two axioms. `Forall1` is the translation in `COQ` of $\vdash (\forall x : A)P(x) \Rightarrow P(a)$ and `Forall12` is the translation in `COQ` of $\vdash [(\forall x : A)(q \Rightarrow P(x))] \Rightarrow q \Rightarrow (\forall x : A)P(x)$ provided that `x` does not occur freely in `q`. In `Forall12`, `[x:A]` `(q => (P x))` represents a predicate which depends on `x`, since `[x:A]` `(... x ... x)` is the function of body `... x ... x`. `(x:A)` is the universal meta-quantification over the `A` (i.e., the universal quantification in `COQ`). Declaring that `p` is a proposition means that `p` does not depend on any parameter, in other words neither `x` nor any other variable occur in `p`.

`ForallRule` is a rule that says that if for each `x` in `A`, `(P x)` is a theorem, then `(Forall A P)` is a theorem. It translates the rule

$$\frac{\vdash P(x)}{\forall x P(x)}$$

A monadic rule is a statement of the form `(theorem foo) -> (theorem bar)`. A dyadic rule is a statement of the form `(theorem foo) -> (theorem bar) -> (theorem baz)` (see the modus ponens above). `ForallRule` sets an interesting connection between the meta-quantification and the quantification in the object theory. This presentation allows us to get rid of the machinery for handling free variables and captures.

Other connectors and quantifiers. Usually in intuitionistic logic each connector and each quantifier must be defined independently of the others, unlike classical logic where one defines usually only two connectors and one defines the other connectors for those two. In higher order logic, the situation is different. One can define all the connectors and quantifiers from two of them, even in intuitionistic logic. We use the connector `=>` and the quantifier `Forall` and we derive the other connectors and the quantifier `Exists` from these ones. Notice that `[p,q:proposition]` is equivalent to `[p:proposition][q:proposition]`.

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

for

$$p \wedge q \triangleq (\forall r : \text{proposition})(p \Rightarrow q \Rightarrow r) \Rightarrow r$$

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

for

$$p \vee q \triangleq (\forall r : \text{proposition})(p \Rightarrow r) \Rightarrow (q \Rightarrow r) \Rightarrow r$$

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

for

$$(\exists x : A)(P x) \triangleq (\forall p : \text{proposition})[(\forall a : A)(P(a) \Rightarrow p) \Rightarrow p]$$

In what follows *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 disjunction, 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).

This statement is the rule

$$\frac{\vdash p \Rightarrow q \quad \vdash q \Rightarrow r}{\vdash p \Rightarrow r}$$

which means that to prove a theorem p => r one has to prove p => q and q => r, where q is a newly introduction proposition, the *cut* proposition.

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 (\mathbb{K} i). In COQ this is an easy task. \mathbb{K} has the signature $\text{nat} \rightarrow \text{proposition} \rightarrow \text{proposition}$. From 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)) .
```

Epistemic logic requires to introduce a modality E for *shared knowledge*. This done in COQ by using the operator `Fixpoint`

```
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 fixed point on the structure of g .

$$E\ nil\ p = TRUE$$

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

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))).
```

E satisfies all the axioms of a modality, for instance $E_g(p) \Rightarrow p$.

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)).
```

i. e.,

$$\frac{}{\vdash C_G(p) \Rightarrow p \wedge E_G(C_G(p))} \quad \frac{\vdash p \Rightarrow (q \wedge E_G(p))}{\vdash p \Rightarrow C_G(q)}$$

Lemmas about C , we have 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))).
```

i. e.,

$$\vdash C_G(p) \Rightarrow p \qquad \vdash C_G(p) \Rightarrow C_G(E_G(p))$$

or a fixed point property like

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

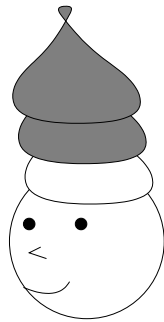
i. e.,

$$\vdash p \wedge E_G(C_G(p)) \Rightarrow C_G(p)$$

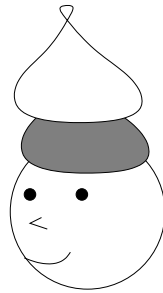
This shows that $C_G(p)$ is a solution of the equation $p \wedge E_G(\varphi) \Leftrightarrow \varphi$ with unknown φ . The rule shows that if ψ is another solution of that equation then $\psi \Rightarrow C_G(p)$.

4 The king, the three wise men and the hats

To illustrate the use of our implementation, let us tackle classical examples. The first one is the puzzle of the king, the three wise men and their hats (Figure 1). It uses only agent knowledge modalities (κ i). In [3], 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). We weakened the two middle sentences in “*It is common knowledge that there are two white hats and red hats. The king puts hats on the head of each of the three wise men and asks them (sequentially) if they know whether they wear a white hat on their head.*”



Alice



Bob



Carol

Fig. 1. The three wise (wo)men

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 whether or not she (he) wears a white 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 whether she wears a white hat and if Bob himself does not know whether he wears a white hat, Carol wears only red hats.” 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 whether she wears a white hat and if Bob himself does not know whether he wears a white hat, Carol wears only red hats, then she knows that the color of her hats and even more (since she knows that the color of all her hats 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 xor red ones. “xor” is the exclusive or written |.

(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 red hats.” 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 red hats.” Moreover the number of red hats is irrelevant and surprisingly an agent can wear more than one hat.

- Each agent knows the color of the hats of the two other agents. Actually we are even more restricted than that, namely Alice knows when Bob (resp. Carol) wears a white hat and Bob knows when Carol wears a white hat.

(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 hats of the agents before her or him. This is sometime a presentation of this puzzle(see for instance [3] 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 emphasized that we made actually less hypotheses than in the usual 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, i. e., no common knowledge. 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, in cleaning up the proof we removed the useless hypotheses.

5 The muddy children

This problem is considered by Fagin et al. [3] as the illustration of epistemic logic, especially of common knowledge. Let us give the presentation of [9]. *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 they 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

Fact 1 all the children know that there are at least p muddy children

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

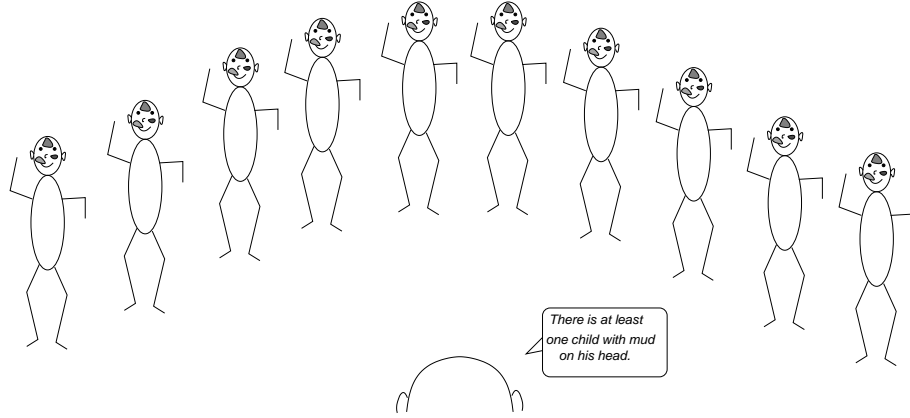


Fig. 2. The muddy children

Fact 1 is there since the children know that there is p muddy children because they see them or because they acquired that fact by deduction. Fact 2, namely the knowledge (shared by the group ($\text{list_of } n$) which stands for $[0, 1, \dots, n]$ also written Chd_n) on the non exactness of the number p of muddy children, comes from the absence of step forward of children. Therefore Fact 2 is known by every children, formally $(K \ i \ (E \ (\text{list_of } n) \ (\text{Not} \ (\text{Exactly } n \ p))))$. In other words, after no child has stepped forward, any child knows that all the children know that there are not exactly p children. 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))))).
```

which is in usually mathematical notation:

$$\vdash E_{\text{Chd}_n}(\text{At_least}(n, p)) \Rightarrow E_{\text{Chd}_n}(\neg \text{Exactly}(n, p)) \Rightarrow K_i(E_{\text{Chd}_n}(\neg \text{Exactly}(n, p))).$$

From it we prove two lemmas:

```
Lemma E_Awareness : (n,p:nat)
  (theorem (E (list_of n) (At_least n p))
    => (E (list_of n) (Not (Exactly n p)))
    => (E (list_of n) (E (list_of n) (Not (Exactly n p))))).
```

i.e.,

$$\vdash E_{\text{Chd}_n}(\text{At_least}(n, p)) \Rightarrow E_{\text{Chd}_n}(\neg \text{Exactly}(n, p)) \Rightarrow E_{\text{Chd}_n}(E_{\text{Chd}_n}(\neg \text{Exactly}(n, p)))$$

```
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))))).
```

i.e.,

$$\begin{aligned} \vdash C_{\text{Chd}_{n+1}}(\text{At_least}(n+1, p)) &\Rightarrow E_{\text{Chd}_{n+1}}(\neg \text{Exactly}(n+1, p)) \\ &\Rightarrow C_{\text{Chd}_{n+1}}(E_{\text{Chd}_{n+1}}(\neg \text{Exactly}(n+1, p))) \end{aligned}$$

Notice that the lemma *C_Awareness* can only be proved for a non empty group of children. We use these lemmas to prove the main result which shows how the knowledge of the children progresses.

$$\begin{aligned} &(\text{C}(\text{list_of}(\text{S } n))(\text{At_least}(\text{S } n) p)) \\ &\& (\text{E}(\text{list_of}(\text{S } n))(\text{Not}(\text{Exactly}(\text{S } n) p))) \\ &=> (\text{C}(\text{list_of}(\text{S } n))(\text{At_least}(\text{S } n) (\text{S } p))). \end{aligned}$$

i. e.,

$$\begin{aligned} \vdash C_{\text{Chd}_{n+1}}(\text{At_least}(n+1, p)) &\Rightarrow E_{\text{Chd}_{n+1}}(\neg \text{Exactly}(n+1, p)) \\ &\Rightarrow C_{\text{Chd}_{n+1}}(\text{At_least}(n+1, p+1)) \end{aligned}$$

In other words: “*If it 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 it 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 apparent miracle.

6 What we learned

This section will appear straightforward to most of the practitioners of theorem prover environment, but we feel they are worth to be said.

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 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.

Acceptable hypotheses A challenge in building proofs in cryptographic protocols is to state the reasonable and acceptable hypotheses and the acceptable hypotheses are not know a priori. We noticed that we often built 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.

A didactic tool

As said, handling proofs à la Hilbert is a very tedious task when done by hand. This is particularly true in teaching where it is not possible to display huge proof trees in front of a class of students. Therefore we decided to use COQ in lectures dealing with Hilbert presentation of propositional logic. Actually since we start with this topics, we give the students (of Ecole normale supérieure de Lyon) the first contact with formal logic through the COQ mechanization of propositional logic à la Hilbert. This way, it is easy to tell them the difference between the language et the metalanguage. Propositional logic is the language and COQ is the metalanguage. The students make clearly the difference. Apparently they like this kind of approach and they are quickly able to handle simple proofs in propositional logic in COQ.

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. Ronald Fagin, Joseph Y. Halpern, Yoram Moses, and Moshe Y. Vardi. *Reasoning about Knowledge*. The MIT Press, 1995.
 4. 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.
 5. Pierre Lescanne. Epistemic logic in higher order logic an experiment with COQ. Technical Report RR2001-12, LIP-ENS de Lyon, 2001.
 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 Construction and Analysis of Systems, TACA'96*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166, 1996.
 7. Mathieu Marion and Mehrnoucha Sadrzadeh. Reasoning about knowledge in linear logic: Modalities and complexity. In Dov Gabbay, Shahid Rahman, John Symons, and Jean-Paul Van Bendegem, editors, *Logic, Epistemology and the Unity of Science*. Kluwer Academic Publishers, 2003.
 8. Will Marrero, Edmund Clarke, and Somesh Jha. Model checking for security protocols. Technical Report CMU-CS-97-139, Carnegie Mellon University, 1997.
 9. 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.
 10. Anne S. Troelstra and Dirk van Dalen. *Constructivism in mathematics*, volume 1. North Holland, 1988.
 11. Paulien de Wind. Modal logic in Coq. Master's thesis, Vrije Universiteit Amsterdam, 2002. available at <http://www.cs.vu.nl/~pdwind/thesis/thesis.pdf>.

A The rule of epistemic logic

The epistemic logic has the following axioms and rules. Notice that $\vdash_K \phi$ means that ϕ is a classical tautology.

K is sometime called *Distribution Axiom* and **T** is sometime called *Knowledge Axiom*. In our our experiments, we do not use *Introspection* axioms. We give them for information.

Notice that the axiom and the rule given for *C* are not the axiom and the rule given in the reference textbooks [3, 9]. We have chosen those ones since they appeared more convenient in the COQ formulation. Obviously they are equivalent to the others.

One could have added other axioms like *Barcan formula*, which rules the connection between **K** and *Forall* and which can be written readily in COQ:

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

$$\begin{array}{c}
\frac{\frac{\frac{}{\vdash_K \varphi} \text{Tautologies}}{\vdash \varphi}}{\vdash \varphi \Rightarrow \psi} \text{Modus ponens}}{\vdash \psi} \\
\frac{}{\vdash K_i \varphi \Rightarrow \varphi} \mathbf{T} \\
\frac{}{\vdash K_i \varphi \Rightarrow K_i K_i \varphi} \text{Positive Introspection} \quad \frac{}{\vdash \neg K_i \varphi \Rightarrow K_i \neg K_i \varphi} \text{Negative Introspection} \\
\frac{}{\vdash (K_i \varphi \wedge K_i(\varphi \Rightarrow \psi)) \Rightarrow K_i \psi} \mathbf{K} \\
\frac{\frac{}{\vdash \varphi}}{\vdash K_i \varphi} \text{Knowledge Generalization}
\end{array}$$

Fig. 3. The basic rules of epistemic logic

$$\begin{array}{c}
\frac{}{\vdash E_G(\varphi) \Leftrightarrow \bigwedge_{i \in G} K_i \varphi} \text{Definition of } E \quad \frac{}{\vdash C_G(\varphi) \Rightarrow \varphi \wedge E_G(C_G(\varphi))} \text{Fixed Point} \\
\frac{\vdash \varphi \Rightarrow \psi \wedge E_G(\varphi)}{\vdash \varphi \Rightarrow C_G(\psi)} \text{Least Fixed Point}
\end{array}$$

Fig. 4. The rules for common knowledge

which is

$$\frac{(\forall x : A)(K_i(P(x)))}{K_i((\forall x : A)P(x))}$$

and which assumes a strong property on A . Actually we do not need it in any of our experiments.