Metareasoning as an Integral Part of Commonsense and Autocognitive Reasoning

Fabrizio Morbini and **Lenhart Schubert**
University of Rochester

Abstract

In this paper we summarise our progress towards building a self-aware agent based on the definition of explicit self-awareness. An explicit self-aware agent is characterised by 1) being based on an extensive and human-like knowledge base, 2) being transparent both in its behaviour and in how the knowledge is represented and used, and 3) being able to communicate in natural language and directly display awareness through its dialogues. We first review the requirements posed by explicit self-awareness on the knowledge representation and reasoning system and then describe how these have been realized in the new version of the EPILOG system. We argue that meta-level reasoning is very important to achieve self-awareness but that it doesn't need to be on an entirely different level from object-level reasoning. In fact, in our agent meta-level reasoning and object-level reasoning cooperate seamlessly to answer each question posed to it.

Introduction

We report on progress towards building a self-aware agent based on the EPILOG inference system, an evolving implementation of the Episodic Logic (EL) knowledge representation. (Schubert 2005; Morbini & Schubert 2007) laid the foundations for our approach to self-aware agents; in summary:

- In '05 Schubert defined explicit self-awareness (as a goal in agent design) as requiring
  - self-knowledge of human-like scope, encompassing physical, mental, autobiographical, and contextual properties;
  - encoding of self-knowledge in a form that is examinable (transparent) and usable by general inference processes; and
  - overt display of self-knowledge through communication.
- In '07 we reported a first proof of concept of the basic ideas behind explicit self-awareness using EPILOG.

We have extended this preliminary work by strengthening the KR&R capacities of EPILOG in a number of ways, particularly with respect to handling of attitudes, substitutional quantification, recursive self-inquiry, and systematization of various facets of the inference process such as formula normalization, loop detection, multiple answer computation, and term evaluation. We will conclude our discussion with some subtle examples involving time-sensitive (indexical) self-knowledge and topical self-knowledge.

Metareasoning

In (Schubert 2005) Schubert lists a series of requirements on the KR&R system to enable explicit self-awareness. We first summarise these requirements, and then describe in some detail how two of these requirements have been implemented in EPILOG.

- *Logic framework*: we require an explicit, transparent representation for the agent's commonsense knowledge and metaknowledge, amenable to browsing and and inferential manipulation. We chose EL as an extension of FOL more fully adapted to the expressive devices of NL.
- *Events and situations*: the agent must be able to refer to events described by complex sentences (e.g., the event described by my (i.e. EPILOG's) failure to answer a question). This capability has always been an integral part of EL and therefore of EPILOG.
- *Generic knowledge*: much of everyday knowledge is expressed using generic or probabilistic adverbs like *usually* or *probably*. In EPILOG generics are expressed using probabilities, though this support is basic and in need of further development.
- *Attitudes and autoepistemic inference*: the ability to reason about one's own knowledge and to represent one's beliefs is fundamental to a self-aware agent. Our commitment is to Kaplan's computational notion of knowing (Kaplan 2000; Kaplan & Schubert 2000). We describe below how the basic machinery needed for this has been implemented in EPILOG.
- *Metasyntactic devices*: a self-aware agent needs to be able to refer in its logical language to the syntax of that language itself. How some of these devices have been implemented in EPILOG will be described later in this section.

We will now describe how the last two requirements have been implemented in EPILOG. As a notational alert, we should mention that EL uses infix notation for predication

and prefix notation for function application; e.g., *(EL (very expressive))* states that EL is very expressive; the predicate *(very expressive)* is infixed, while the predicate modifier *very* (a function that maps monadic predicates to monadic predicates) is prefixed.

Substitutional Quantification and Quasi-Quotation

As motivated and exemplified in (Schubert 2005; Morbini & Schubert 2007) it is important to be able to refer to syntactic elements of EL in an EL formula itself. To do so EPILOG currently supports two devices: substitutional quantifiers and quasi-quotation.

Their implementation is straightforward and posed no major problems. The two main modifications required were

- to the parser: we added substitutional quantifiers and metavariables. A metavariable is a particular type of variable that is bound by a substitutional quantifier. A substitutional quantifier is much like a normal quantifier except that it quantifies over substitutions of expressions of a particular category for the metavariable. For example: *($\forall_{wff}$ w (w $\Rightarrow$ (me Know (that w))))* quantifies over substitutions of EL well-formed formulas for the variable *w*. The truth conditions of the formula are that all instances of the formula are true under the object-level semantics, when EL well-formed formulas (not containing metavariables) are substituted for *w*.

  Quasi-quotation, written with a quote sign (apostrophe) accepts an expression as argument that may contain metavariables; substitution for such metavariables treats quasi-quotes as transparent. As an example of the use of (quasi-)quotation, we can express in EL that the "=" predicate is commutative by writing *('= (Commutative El-predicate))* where *Commutative* is a predicate modifier and *El-predicate* is a predicate.

- to the unification routines: since we have provided for metavariables, unification should be able to unify not only variables with terms but also metavariables with EL expressions of the appropriate categories. For example, we should be able to unify *(me Know (that w))* with *(me Know (that (?x Foo)))*, yielding unifier $\{(?x\ Foo)/_w\}$.

Recursive QA

In (Morbini & Schubert 2007) we described the basic properties a computational notion of knowing should have. We indicated why *knowing* is very different from *being able to infer*, and referred to Kaplan & Schubert's algorithmic ASK mechanism as a basis for *knowing*. Here we describe how that notion is supported in EPILOG. The intuitive way to implement ASK (and thus to answer questions that involve predicates *Know* or *Believe*) is to allow for question-asking within a question/answering (QA) process. Answering questions about nested beliefs thus involves further nesting of QA processes. That is the basic idea behind recursive QA.

Again implementing this is straightforward in any system with a clean and modular implementation of the QA process. What is needed is that the QA process must work only on local variables and the same must be true for all systems on which QA depends (e.g., knowledge base, unification, inference, normalization, etc).

In addition to having a modular system, one needs a way to connect inference with the QA process so that this process can be started whenever it is required by some inference. In EPILOG this is achieved by using the metasyntactic devices described in section and by providing a single special-purpose function, called "APPLY", that the QA process knows how to evaluate whenever its arguments are quoted and metavariable-free. (It executes a Lisp function of the same name for the given arguments; this linkage from reasoning to computation can also be used for many other purposes). In particular, to implement the ASK mechanism we added the following axiom to EPILOG's standard knowledge base: *($\forall_{wff}$ w ('w WithoutFreeVars) ((me Know (that w)) $\Leftrightarrow$ ((APPLY 'knownbyme? 'w) = 't)))*, where knownbyme? is a Lisp function that implements the ASK mechanism as a recursive QA-process. *WithoutFreeVars* is a predicate with one argument denoting an expression, typically specified using quotation, that is true whenever the argument doesn't contain free variables. To evaluate this predicate EPILOG will have another axiom in its knowledge base: *($\forall_{subst}$ x (('x WithoutFreeVars) $\Leftrightarrow$ ((APPLY 'withoutfreevars? 'x) = 't)))* where it withoutfreevars? is the Lisp function that detects whether or not an EL expression contains free variables.

Currently, all formulas involving APPLY are equivalences (like the two above) in which the variables are universally quantified and that may have simple conditions (like *('w WithoutFreeVars)* above) applied to those universally quantified variables.

Whenever the QA process uses any of the formulas involving the APPLY function it does the following:

1. unify the universal variables and check that all conditions on them are satisfied. This check is done by starting a QA process for each condition.
2. if the previous check is satisfied, execute the Lisp function specified as first argument of the APPLY function, with the arguments provided for it, after having applied the unifier found at the previous step.

Inference in EL

In this section we will describe other characteristics of EPILOG's inference machinery not directly related to metareasoning but important to EPILOG's functioning.

Normalization

Because EL uses non-first-order constructs, e.g., substitutional quantification, quotation, lambda abstraction, and modifiers, the standard FOL normalization to clause form cannot be used. (Besides, clause form can be exponentially larger that the original form, for example for a disjunction of conjunctions.) Normalization for EL is based on term-rewriting systems (), in particular on the following algorithm. (Think of EL expressions as trees, where the children of a node are the immediate subexpressions.) The two main parts of the algorithm are

1. a set of rewriting rules each divided into two parts:

(a) a set of preconditions. Each precondition is defined by a child identifier and by a function name. The child identifier extracts one or more descendants (subexpressions) of the EL expression currently being analysed, and the function name specifies a boolean Lisp function that takes as argument(s) the descendant(s) specified by the first part.

(b) a function that defines the effects of the rule. This function is executed when all preconditions are satisfied.

2. an application procedure that traverses the EL tree checking at each node if a rule applies. If a rule applies, it is executed, and if this modifies the node, control backs up N levels from the current node and the traversal is restarted from there. N is the maximum depth of the descendants used in the preconditions of the normalization rule. For example, if the rule uses only children then $N = 1$, while if some use a grandchild then $N = 2$, etc.

Currently the normalization process employs a total of 14 rules. They perform such transformations as moving negations inward, Skolemizing top-level existentials, ordering the arguments of ANDs and ORs, perform basic tautology elimination, move inward quantifiers, etc.

Inference Graph Handling

EPILOG's QA is in its simplest form a natural deduction back-chaining system. It starts with the initial question, then generates a proof subgoal and a disproof subgoal (i.e. the negation of the initial question). The QA process maintains an agenda of subgoals based on an AVL tree that decides which subgoal to process first. We have not finalised how the subgoals are sorted in the agenda; some criteria may be: subgoal complexity, their probability, their level, etc. More testing is required to decide which combination of criteria is advantageous in the majority of situations.

Each subgoal is first checked to see if it can be simplified:

- conjunctions are split into their conjuncts, and each conjunct is handled independently of its siblings until it is solved. If a conjunct appears in the current knowledge base, it is trivially solved. When a solution to a conjunct is obtained, it is properly combined with the solutions of the siblings.
- for disjunctions, if a disjunct appears in the current knowledge base, then the disjunction is trivially solved. Otherwise, disjunctions are split for each disjunct by assuming the negation of the remaining disjuncts. Here we make use of another feature, namely knowledge base inheritance. Each question is associated with a knowledge base that defines what knowledge can be used to answer the question. When assumptions are made, they are loaded into a new knowledge base (to be discarded at the end of the QA process) that inherits the contents of the knowledge base used before the addition of the assumptions. Currently the consistency of the assumptions is not checked, but problems will be detected from the contradictory answers produced.
- implications, $A \Rightarrow B$, are converted into two subgoals, $(notA)$ and B assuming A.
- equivalences are split into conjunctions.
- universally quantified goals are simplified by generating a new constant and unifying the universal variable with it. If the universal quantifier has a restrictor, that is assumed.

When no more simplifications can be applied, goal-chaining inference is attempted. From each subgoal, one or more keys are extracted and used to retrieve knowledge. These keys are the minimal well-formed formulas embedded entirely by extensional operators such as quantifiers and truth-functional connectives. Each subgoal maintains another agenda that decides which key to use first for retrieval. As in the case of subgoal ordering, we have not yet finalised the sorting criterion. Possibilities are preferring keys that contain more variables, or ones with the least associated knowledge.

Goal-chaining inferences can be thought of as being resolution-like, except that the literals being resolved may be arbitrarily embedded by extensional operators. (For details, see, e.g., (Schubert & Hwang 2000).)

For each successful inference performed for a subgoal together with a retrieved formula, a child subgoal is attached to this subgoal and the process is repeated (with termination if the derived subgoal is truth).

The two processes just described (i.e., simplification and inference) construct an inference tree. However loops and repetitions can happen, worsening performance or preventing success altogether (in case the subgoal selection behaves as a depth-first run-away). Therefore we added two optimizations, the second of which transforms the inference tree into an inference graph:

1. loop detection: a loop is created when the same subgoal appears twice along an inference branch. In saying that a new subgoal is the "same" as a previous one, we mean that it has the same EL formula and is associated with the same knowledge base, or with a knowledge base that inherits that of the previous subgoal.

2. to avoid doing the same reasoning multiple times we detect when the same node (where "same" is defined as above) is present on a different branch (therefore not forming a loop). In this case, we connect the two nodes and in case the already present node uses the exact same knowledge base as the new one, we completely stop further processing of the new node. If instead the new node uses a knowledge base that inherits from that of an old node we continue to process the new node as if the old didn't exist (except for adding the connection between the two nodes).

   In case the old node is answered, the answer is propagated to the new node as well.

Multiple answers

In many cases it is necessary to be able to handle multiple answers for a given subgoal, for example, when the question is not a yes/no question (i.e. wh questions).

The main capabilities required to support multiple answers are:

- the ability to propagate the answers found from the leaves of the inference graph up to the initial question. This includes taking care of merging, or waiting for[1], the answers of siblings nodes in case they where part of a conjunction or disjunction; properly handling renaming of variables and the merging of unifiers.

- the ability to avoid the propagation of duplicated answers. We consider two answers the same if they use the same knowledge and produce the same unifiers.

Term Evaluation

Sometimes the answer may contain complex terms, for example functions, instead of their result as expected by whomever asked the question. For example, to the question "*How old are you?*" the system could answer with "*The difference in years between 1st January 1993 and now*" instead of actually computing the difference.

Currently we employ a term evaluation procedure based on the same QA process. Given a complex ground term *T* the question *(∃ x (x = T))* is posed and the unifications for *x* are collected only if these are simpler than *T* itself. The process is recursive, i.e., the unifications collected for *x* can be evaluated if they are complex terms. Because the system uses the same QA process it can detect loops and avoid duplication in the evaluation process as described in section .

However this is not ideal because the evaluation process is preprogrammed and fixed. Instead, as indicated in (Schubert 2005), we would like the QA process to automatically look for the correct type of answer as specified by *syntactic* constraints that are part of the question itself; if we ask the age of an individual, the question should probably constrain the answer to be a decimal integer providing the age in years.

Examples

In this section we describe some of the examples used to test the features of this system.

In (Morbini & Schubert 2007) we included a preliminary discussion of the questions "*Do pigs have wings?*" and "*Did the phone ring (during some particular episode E1)?*", now these questions can be solved as described in that paper.

One of the most interesting questions we have tried so far is the question "*How old are you?*". Though one can easily imagine simple short-cut methods for answering such a question, doing so in a principled, knowledge-based fashion can be non-trivial. The following table shows the knowledge used for this question:

EPILOG's birth date is 12 o'clock on the 1^{st} of January 1993 *((date 1993 1 1 12 0 0) BirthDateOf Epilog)*
If x is the birth date of y then for every event e the age in years of y in e is the difference in years between x and the time of e *(∀ y (∀ x (x (be (BirthDateOf y))) (∀ e ((y HasAgeInYear (DiffInYear x (TimeOf e))) * e))))*
Time density axiom *(∀ y (∃ x (x AtAbout y)))*
Approximation of the meaning of AtAbout *(∀ x (∀ y (x AtAbout y) ((TimeOf x) = (TimeOf y))))*
Symmetry of the predicate AtAbout *(∀ x (∀ y ((x AtAbout y) ⇔ (y AtAbout x))))*
Axiom that says how to evaluate the predicate DiffInYear *(∀ x (∀ y (y AbsoluteTimePoint) (∀ z (z AbsoluteTimePoint) ((x = (DiffInYear y z)) ⇔ (x = (APPLY 'diff-in-years? 'y 'z))))))*
Axiom that says how to evaluate the predicate AbsoluteTimePoint *(∀ x ((x AbsoluteTimePoint) ⇔ ((APPLY 'abs-time-point? 'x) = 't)))*

The question in EL becomes *(∃ x (∃ e (e AtAbout Now) ((Epilog HasAgeInYear x) ** e)))*

The answer found is that in the event *(FNSK-449 Now)* the age of EPILOG is *(DiffInYear (date 1993 1 1 12 0 0) (TimeOf (FNSK-449 Now)))* where *FNSK-449* is the Skolem function derived from the density axiom; so *(FNSK-449 Now)* identifies an event temporally near the event Now.

Given this answer, the evaluation of *(TimeOf (FNSK-449 Now))* using the knowledge that *((FNSK-449 Now) AtAbout Now)* and that *(∀ x (∀ y (x AtAbout y) ((TimeOf x) = (TimeOf y))))* produces *(TimeOf Now)*, which can be evaluated to the current time.

After that *(DiffInYear (date 1993 1 1 12 0 0) (TimeOf Now))* can be directly evaluated using a dedicated function to compute the difference of two time points.

In the current solution, as already said in section , we explicitly call the term evaluation routine for each complex ground term returned as answer to a question.

The next question considered here is "*What is your name (now)?*". the knowledge used is displayed in the following table:

Epilog-name is a name *(Epilog-name Name)*
A name is a thing *(∀ x (x Name) (x Thing))*
Now is during event E2 *(Now during E2)*
The event E2 is characterised by EPILOG having name Epilog-name *((Epilog Have Epilog-name) ** E2)*
Have is a continuous property: if x have y in e then x has y in all events during e. *(∀ x (∀ y (∀ e ((x Have y) ** e) (∀ z (z During e) ((x Have y) ** z)))*

The question in EL is represented as *(∃ e0 (e0 AtAbout Now) ((∃ z ((z Name) and (Epilog Have z)) (∃ y (y Thing)*

[1] In case the siblings are part of a conjunction, the propagation of the answer of one of them must wait for a positive answer from all the siblings.

*(y (BE (L x (x = z)))))) ** e0))*. The apparently convoluted form is due to the fact that this question is automatically generated from the NL input.

After normalization we obtain the simpler question *(∃ e0 (e0 AtAbout Now) (∃ z ((z Name) and (z Thing)) ((Epilog Have z) ** e0)))*. The normalization procedure moves inward the "**" operator using the knowledge that "*Name*" and "*Thing*" are atemporal predicates. This knowledge is stated in EL and is used by the normalization procedure.

In the current reasoning we manually add the fact that EPILOG's name is valid in an interval of time that includes the Now point. However, in future we would like this property to be automatically generated by a module in charge of maintaining the self-model of the system.

The last example shows how the metasyntactic devices could be used to answer topical questions. The question is "*What do you know about the appearance of pigs?*". The following table contains the knowledge used:

Pigs have a relatively big diameter/length ratio. *((K (Plur Pig)) ThickBodied)*
ThickBodied is a predicate about the appearance of something. *('ThickBodied AppearancePred)*
Every formula with structure (x p) in which p is an appearance predicate is a fact about the appearance of x. *($\forall_{pred}$ p ('p AppearancePred)* *(∀ x (x p) ((that (x p)) AppearanceFactAbout x)))*

The question in EL becomes *(∃ x (x Appearance-FactAbout (K (Plur Pig))))*. The answer found is "*(that ((K (Plur Pig)) ThickBodied))*".

However, to retrieve more complex knowledge about pigs, for example that pigs have curly tails, more complex knowledge would have to be used.

Discussion and Further Work

The examples given show the ability of the current system to handle basic forms of metareasoning. However there is still much work to be done; some of the more pressing items are the following:

- Currently we have implemented only an exhaustive retrieval mechanism to be able to test the system. Given that our goal is to build an inference agent able to deal with the large knowledge base required by commonsense applications, having an efficient knowledge retrieval mechanism is crucial. Without an efficient retrieval mechanism the inference engine will only be able to answer questions using a small knowledge base.

  EPILOG already had an indexing scheme designed to be scalable but some extensions are required, in particular in these directions: 1) retrieval of knowledge that uses any of the metasyntactic devices described in this paper, 2) closing some retrieval gaps, in particular in goal chaining, and 3) automatically building the type information needed to efficiently index a formula based on the most restrictive type that can be inferred for the variables in it.

  After this efficient indexing schema is finalised and implemented, we plan to test its scalability using some large knowledge base (for example, the FOL conversion of OpenCyc (Ramachandran, Reagan, & Goolsbey 2005)).

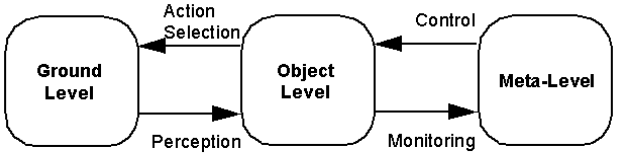
  In another test we plan to assess the completeness and efficiency of retrieval and goal chaining, by running forward inference and checking how many of the inferences produced (and in how much time) can be proved by goal chaining.

- For solving many commonsense problems it seems necessary to provide the general inference engine with a set of specialized inference methods. In EPILOG these specialised reasoning routines are called specialists. In the previous version of EPILOG these specialists were "unconsciously" invoked by the general inference engine. We would like to add knowledge, based on the APPLY function, to make the inference engine aware of its specialists and their capabilities.

- Another front that needs work is the refinement of the ASK mechanism. Currently the ASK mechanism is made time-bounded simply by means of a hard limit on the depth of reasoning. (However, the limit can be computed so that several desirable properties of knowing are maintained – e.g., given that EPILOG knows *A* it also knows *A or B*).

Relationship to traditional conceptions of metareasoning

The traditional conception of the role of metareasoning in an intelligent agent is diagrammed in the following figure.



The emphasis in this conception is on control and monitoring of object-level reasoning by higher-level reasoning processes (e.g., (Cox 2005)). This is certainly one of the most important roles that metareasoning can play, but not the one we have been focusing on. Instead, we have focused on metareasoning as an integral part of commonsense and autocognitive reasoning.

This leaves the question of how we might add metalevel control to an explicitly self-aware agent of the sort we have been striving to define and implement. We have no fully worked-out answer to this question, but can make the following remarks.

We ultimately conceive of a purposive agent as continually modifying, evaluating, and partially executing a "lifelong" plan. This plan contains hierarchically structured goals and actions (and various annotations concerning the purpose of steps, their prerequisites, effects, timing, etc.) It is perfectly possible for goals or steps to be reasoning goals or steps (e.g., to confirm or disconfirm certain propositions, to identify tuples of entities standing in certain specified relationships, to find a hypothesis accounting for given facts, etc.) As soon as we admit this possibility, planning becomes

metareasoning, in the sense that the agent is controlling its own reasoning actions via its planning/execution activity.

However, such a view of metalevel control of reasoning may not fit well with the above figure because it blurs both the distinction between Doing and Reasoning (the two modules on the left) and that between Reasoning and Meta-Level control (the two modules on the right). Concerning the first distinction, planning and execution of reasoning steps in our conception are handled by the same processes as planning and execution of physical (or perceptual, or communicative) actions; i.e., the continual planning/execution module allows for mingling of these types of steps. Concerning the second distinction, while the planning/execution process involves processes unique to it (we envisage a fixed "greedy" process that continually tries to improve upon the net expected cumulative utility of the agent's "life"), it also involves some of the same reasoning processes required for commonsense question-answering. For example, one important aspect of planning is the reasoned elaboration of a step of type "achieve subgoal G" into more explicit actions to achieve G. This may well be done with the help of axioms such as one stating that "doing action(s) A brings about G". Now, is this elaboration step an instance of object-level or meta-level reasoning? The answer depends on the syntax of A and G. If A is an action described in the object-level language of the agent, and G is a goal also described in that language, then the elaboration step could be regarded as object-level reasoning; but if A describes a reasoning action (using attitudinal predicates such as "I try to prove that ...", or perhaps using explicit quotation to refer to an executable inference procedure with known effects), then the elaboration step could be regarded as an instance of meta-level reasoning. So in our conception, the distinction between meta-level and object-level reasoning is not an explicit architectural one, but rather is "hidden" in the syntax of the knowledge involved.

References

Cox, M. 2005. Metacognition in computation: A selected research review. *Artificial Intelligence* 169(2):104–141.

Kaplan, A. N., and Schubert, L. K. 2000. A computational model of belief. *Artif. Intell.* 120(1):119–160.

Kaplan, A. 2000. *A Computational Model of Belief*. Ph.D. Dissertation, University of Rochester.

Morbini, F., and Schubert, L. K. 2007. Towards realistic autocognitive inference. In *Logical Formalizations of Commonsense Reasoning*, 114–118.

Ramachandran, D.; Reagan, P.; and Goolsbey, K. 2005. First-orderized researchcyc: Expressivity and efficiency in a common-sense ontology.

Schubert, L., and Hwang, C. 2000. Episodic logic meets little red riding hood: A comprehensive, natural representation for language understanding. In Iwanska, L., and Shapiro, S., eds., *Natural Language Processing and Knowledge Representation: Language for Knowledge and Knowledge for Language*. Menlo Park, CA: MIT/AAAI Press. 111–174.

Schubert, L. K. 2005. Some knowledge representation and reasoning requirements for self-awareness. In *Metacognition in Computation*, 106–113.