

Computational Infrastructure for a Self-Aware Agent

by

Fabrizio Morbini

Submitted in Partial Fulfillment

of the

Requirements for the Degree

Doctor of Philosophy

Supervised by

Lenhart K. Schubert

Department of Computer Science
Arts, Sciences and Engineering
School of Arts and Sciences

University of Rochester
Rochester, New York

October 8, 2009

Curriculum Vitae

Fabrizio Morbini received a Laurea in Electronic Engineering from Università degli Studi di Brescia in 2002, with a thesis on extensions to the DISCOPLAN system supervised by Prof. Alfonso Gerevini. He began graduate studies at the University of Rochester in the fall of 2003. He pursued his research on the EPILOG reasoner, revising and extending it to support his work towards a self-aware agent, under the guidance of Prof. Lenhart Schubert. He received his MS from the University of Rochester in 2005.

Acknowledgments

Thanks to Aaron Kaplan for his effort in attempting to clean the original version of EPILOG and bringing it under a revision system for the first time.

Thanks to Alexandre Riazanov for his help with Vampire and the JJ parser.

Thanks to Larry Lefkowitz for his support on Cyc.

Thanks to the doctoral committee (Professors James Allen, David Braun, Dan Gildea and Lenhart Schubert) and the chair of the defense (Professor Michael Tanenhaus) for their helpful comments.

This material is based upon work supported by grant #IIS-053510 from the National Science Foundation and a gift from Bosch Research and Technology Center (Palo Alto) from 2007-2008. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of above named organizations.

Abstract

Self-awareness is an aspect of consciousness that is highly developed in humans in comparison with other animals. A human being unaware of his or her personal characteristics, of what he or she knows and doesn't know, can do and cannot do, wants and doesn't want, has experienced and is experiencing, etc., would surely be difficult to communicate with naturally. Therefore we believe that consciousness plays a crucial role in building artificial dialog agents with human-level abilities.

In this work we describe our effort in building the extension of the COMA-/EPILOG system, called Epi2Me/EPILOG2 aimed at explicit self-awareness. A system is explicitly self-aware if it has a complex self-model, is transparent in its internal workings and explicitly displays self-awareness through its interaction with the users.

The new system achieves a more than 100% improvement over the previous version of EPILOG in reasoning speed on traditional first-order logic problems, providing at the same time more extensive support for the requirements of explicit self-awareness, including many of the introspective and metareasoning capabilities needed for explicit self-awareness. In addition its design will provide a solid base on which to build future developments.

Table of Contents

Curriculum Vitae	ii
Acknowledgments	iii
Abstract	iv
List of Tables	vii
List of Figures	x
1 Introduction	1
2 Concepts of Consciousness and Self-Awareness	5
2.1 Definitions	5
2.2 Human consciousness	11
2.3 Machine consciousness	32
3 Previous Work	40
3.1 SHRDLU	41
3.2 Homer	48
3.3 CASSIE	61

3.4	COMA, Episodic Logic and EPILOG1	67
3.5	Comparative Evaluation	79
4	Epi2Me	84
4.1	Parser	85
4.2	Normalization	88
4.3	Interface to Knowledge Bases	90
4.4	Unifier	94
4.5	Specialist interface	95
4.6	Inference	97
4.7	Type extraction	98
4.8	Storage Framework	105
4.9	Question-answering (QA) Framework	105
4.10	Summary	112
5	Evaluation	115
5.1	Evaluation with Cyc and hand-built questions	116
5.2	Evaluation using TPTP	132
5.3	Summary	139
6	Conclusion and Future Work	142
	Bibliography	149

List of Tables

2.1	Performance obtained by human subjects classifying unidimensional inputs.	17
2.2	This table (from (Miller, 2003)) shows how recoding can be employed to reduce the number of items that have to be stored in working memory. This enables relaxation of the strong constraint on the number of items that working memory can contain.	19
3.6	Summary of the distinguishing features of the 4 systems compared in this chapter. The upper part of the table lists features generally connected with reasoning and planning agents. The bottom part instead has features more directly related to self-awareness. The green color indicates that the way in which a particular aspect has been realized by a particular system is well suited for the development of a self-aware agent. The orange color instead indicates that the particular aspect is lacking some characteristic necessary for self-aware agents.	81
5.11	Results of some of the past CASCs, from http://www.cs.miami.edu/~tptp/CASC/	133

- 5.12 Summary of the tests carried out between EPILOG1, EPILOG2 and the Vampire theorem prover versions 8.1, 9 and 10. For version 10 we report the results from the TSTP collection of solutions because we were unable to run the theorem prover on our machine. The first column contains the segment numbers (1-4) of the segments comprising the scalability subset of the CSR dataset (with 50 problems in each segment). Column 2 contain the min, max and average number of formulas contained in the problems in that specific segment. (If all problems contain the same number of formulas only the average is shown). Columns 3, 4, and 5 show the percentage of problems for which a solution was found, respectively by EPILOG1 with forward inference enabled, EPILOG1 without forward inference and EPILOG2 (which by default has no forward inference enabled). Column 6 shows the average depth of the answer found by EPILOG2. Column 7 shows the percentage of problems solved by Vampire. All systems, except Vampire 10, have been limited to a timeout of 120 seconds on a Pentium D running at 2.8GHz with 1G of RAM and Linux 2.6.28. (*) Vampire 10 results were obtained with a 10 minutes timeout on a P4 at 2.8GHz with 1G of RAM and Linux 2.6.18. Vampire 8.1 and 9 on segment 5 are unable to complete any problem because they terminate with a segmentation fault. 137
- 5.13 To give an idea of the dependency of the results provided in table 5.12 on the time allowed for reasoning, we report here the percentage of problems solved in the 5 segments by EPILOG2 and Vampire with 4, 8 and 16 minutes timeout. On segment 5 both Vampire 8.1 and 9 are unable to complete any problem because they terminate with a segmentation fault. 138

5.14	Results obtained by SInE 0.3 on the same set of problems used in table 5.12. The second column shows the percentage of problems solved in that segment when given a 10 minutes timeout. Given that we were unable to run the system on our machine, we report the results contained in the TSTP.	139
------	--	-----

List of Figures

2.1	Performance obtained by human subjects classifying multidimensional inputs. The performance grows less than linearly with the number of dimensions considered.	18
2.2	Baars' theater model of consciousness.	20
2.3	Kuipers' model of consciousness.	26
3.1	Structure of SHRDLU.	43
3.2	A representation of the world in which SHRDLU acts.	43
3.3	A representation of the virtual world in which Homer acted. . . .	49
3.4	A schematic representation of Homer's internal architecture. . . .	53
3.5	The CASSIE architecture.	62
3.6	The embodied CASSIE architecture.	63
3.7	The high level structure of EPILOG1.	69
4.1	The high level structure of EPILOG2.	86
4.2	Tree representation of an EL formula. Note the sharing of variable nodes to facilitate variable substitution.	89

4.3	State of long-term storage after the formulas $(\forall x (\exists y [x \text{ Foo } y]))$ and $(\forall x (\exists y [y \text{ Foo } x]))$ have been stored. Each node has an id associated with it. The formulas displayed in the graph (e.g., $[\text{var Foo var}]$ are not actually stored, they are reported here just for convenience). To reconstruct a stored formula, start from the corresponding leaf and follow the solid line after having traversed the dashed line (that points to the content of non-leaf nodes). . .	93
4.4	Proof found for the problem CSR034 when transitive predicates are used to build hierarchies of their arguments.	102
4.5	Proof found for the problem CSR034 when the extraction of transitive information is not enabled.	103
4.6	Hierarchy automatically built for the arguments of the transitive predicate GENLMT in the problem CSR034.	104
4.7	High level algorithm of EPILOG2's QA framework.	106
4.8	The dashed subgoal is not added if $f1_{maj} \equiv f2_{maj}$ and $key1_{maj} \equiv key2_{maj}$ and $key3_{sg}$ was a part of $f1_{maj}$. This avoids the creation of an inference loop that creates a cascade of ever growing subgoals.	108

1 Introduction

Self-awareness, considered as the ability to perceive and reason about oneself, is still largely lacking in machines even though it is one of the most distinctive features of human intelligence. In addition to being an important part of human intelligence, self-awareness has the potential to make the interaction with an artificial agent more engaging and effective because self-awareness provides the ability to explain choices, remember previous actions and reason about capabilities and limitations. For example, without self-awareness an agent would be unable to genuinely comprehend questions containing the pronoun “*you*” because the interpretation of “*you*” presupposes the presence of a self-model in the agent, which is one of the requirements for consciousness¹. Not only will self-aware systems make the interactions with them easier and more interesting but they will also require pushing forward the language abilities and knowledge representation schemes that will bring benefits to many other areas of artificial intelligence (AI).

Self-awareness is a property as yet not well understood in humans. Therefore it’s appropriate to more clearly define the term, at least for the purpose of comparison with other systems and to guide the development of our own system. In

¹One can think of very simple implementations with basic support for “*you*” that function without any self-model. What we mean here is that to allow an agent to reason about itself as required by a human-level consciousness (a notion to be clarified in the rest of the document), we need to add to the agent a non-trivial model of itself.

this work we use the definition of explicit self-awareness introduced by Schubert in (Schubert, 2005) and characterized by:

- *complex self-model*: the agent must be able to express complex features in its self-model and relate these to a substantial base of general common-sense knowledge. This is motivated by the long term goal of human-like intelligence and therefore human-like complexity.
- *explicit*: the agent’s self-awareness is evident from its behaviour (i.e., its dialogs in the case of a dialog agent). This is motivated by usability considerations given that humans prefer natural language communications. Also natural language interactions don’t require any additional hardware and therefore allow the agent to be a pure software system, significantly simplifying the development process.
- *transparent*: it is possible for an external observer to inspect and easily understand the state of the agent and its internal reasoning. This is not strictly related to self-awareness but it’s necessary to allow an easier development of the system.

There has been much work on the philosophical and theoretical implications of self-awareness in general (some of which will be described in chapter 2), but this work is focused in particular on building an artificial, explicitly self-aware agent. There have been various attempts to build agents that display various degrees of self-awareness, and chapter 3 will describe some of them. The major points of distinction for this work are:

- Our agent is based on logic, in particular we selected Episodic Logic (EL) for its expressivity and connection with natural language. This helps with all 3 requirements of explicit self-awareness and distinguishes our system from

most of the other agents, which either leave logical aspects under-defined or use languages easily convertible to first order logic (FOL).

- We focus on question/answering (QA), ignoring the planning component that any agent should have. This simplifies our work without imposing any limit on future development and without trivializing any aspect of self-awareness. We also haven't built an English-to-EL converter yet, so all communications with the systems are forced to be in EL. For the time being we also ignore applying learning to any aspect of the developed system.
- Comparison with other agents and with past/future revisions of the same agent are quite hard with respect to self-awareness and reasoning in general; even more so when the agents use different representation languages. In this work we tried to design an evaluation that should allow more easily and readily for comparison with others.
- The design is aimed at extensibility and usability. These are extremely important features for a software system, to allow its augmentation over time and its use in several applications. As mentioned in chapter 3 several systems have been abandoned in part because of bad design decisions that made extensions hard. In this work we paid particular attention to avoidance of such mistakes. On the usability front, an important factor is the amount of setup required by the system to successfully solve a given problem. We re-designed EPILOG to require minimum setup and in particular no problem specific setup.

The rest of this thesis is divided in four major parts: in chapter 2 we introduce the terminology characteristic of this field (section 2.1), some background work on modelling human consciousness (section 2.2) and some preliminary work in classifying and defining machine consciousness (section 2.3).

In chapter 3 we describe four systems featuring some aspects of self-awareness: SHRDLU (section 3.1), Homer (section 3.2), CASSIE (section 3.3) and COMA (section 3.4). We conclude this chapter with a comparison of these systems according to the criteria outlined in section 2.3.

In chapter 4 we describe the extensions to the COMA/EPILOG system that are the core of this work. We then evaluate the system (chapter 5) using a two-part evaluation. In the first part we use a selected set of demo questions to display some of the characteristics of EPILOG including the use of introspection, substitutional quantifiers and quasi-quotation. In the second part we use the scalability challenge included in the TPTP dataset. The new system displays performance similar to the best FOL theorem provers and more than doubles the performance of the original EPILOG system.

Finally, we conclude in chapter 6. In summary, with this work we produced a new version of the EPILOG system that is easier to use, more extendible and faster than the original EPILOG version, proving that reasoning with a highly expressive logic like EL is feasible and doesn't necessarily imply compromising performance. In addition, although the new version doesn't yet have all the features of the old version (like probabilistic reasoning), it better supports explicit self-awareness with a better implementation of substitutional quantifiers and introspection and a transparent interface to specialists.

2 Concepts of Consciousness and Self-Awareness

In this chapter we will define several terms often found in the relevant literature. Then we will describe a few attempts to computationally model human consciousness, preceded by a short introduction to the philosophical debate on human consciousness. We will conclude the chapter with a section describing work on defining machine consciousness and the requirements it imposes on knowledge representation and inference.

This chapter is not intended to be an extensive survey of consciousness research but only a selection of topics that we found interesting and relevant to our computational goals. In particular our work is not focused on understanding and modelling how our brain works (and how it achieves consciousness); therefore the review of the background material contained in this chapter only superficially covers those aspects.

2.1 Definitions

In this section we will list various terms commonly used in papers on consciousness and provide a brief description for each of them. Given that most of the terms listed here are hard to precisely define, the descriptions provided are intended to

be superficial, but generally uncontroversial, summaries of the meanings of these terms.

- **Consciousness:** as a philosophical concept Ned Block in (Block, 2002)¹ gives the following classification:

- *Phenomenal consciousness:* X is in a phenomenal conscious state if X is experiencing something that characterizes that state. The criterion widely used to talk about phenomenal consciousness is that “there is something it is like to be in that state”. For example, if you are phenomenally conscious of the bright blue sky then, it is because you are experiencing something that makes that mental state a phenomenally conscious state. This experiencing is the key concept of phenomenal consciousness, and according to David J. Chalmers accounting for this experiencing should be considered the hard problem of consciousness (Chalmers, 1995).
- *Access consciousness:* we are access conscious of something if we have a representation of it and this is *broadcast* to every part of the brain so that it can be used for reasoning and rational control of actions. Some question whether being access-conscious of something counts as conscious at all, but we think that access consciousness is the type of consciousness we would like to implement in a machine first. When this is achieved, it may provide a better insight in tackling the question of whether the machine is “really” experiencing something or not (where “really” is not at all well defined).

Ned Block identifies three differences between phenomenal and access consciousness:

¹Revised version of (Block, 1995).

1. Access consciousness is *totally* defined by a representation (like a clause in a logic agent that represents some concept or fact). While phenomenal consciousness can also have a representational component, what identifies a phenomenal conscious state is the experience the agent is undergoing because of being in that phenomenal conscious state.
2. Access consciousness characterizes a mental state as a conscious state because of its relations with other modules (in other words access consciousness uses a functional approach to classify a mental state as a conscious mental state). Therefore a state, to be conscious, has to be able to influence reasoning and action.
3. Phenomenal consciousness identifies types of states. For example, all the pain feelings imply the same kind of phenomenal conscious state (the pain kind). But if we consider each pain from the perspective of access consciousness, each one is a different state because it produces different reactions, memories and inferences.

To better identify this type of consciousness Block describes cases in which we have access consciousness but not phenomenal consciousness and vice versa.

A case with only access consciousness but not phenomenal consciousness is the one of blind-sight patients². In fact a blind-sight patient has the representation of the stimulus, but because she doesn't experience the stimulus, according to phenomenal consciousness she is not conscious of it. Another example is that of a robot that doesn't have experiences: it is conscious of all the factual knowledge about what is

²Blind-sight is the pathology in which the patient has blind spots in her field of view because of some damage in the visual cortex. But if simple stimuli are given in these blind spots and if the patient is asked questions about properties of the stimuli she is able to answer with surprisingly high precision.

going on in front of it but it doesn't experience anything (this is debatable in the same way as Searle's Chinese room thought experiment).

A case of phenomenal consciousness without access consciousness is for example one in which we are experiencing a sound but we are not aware of it. For example, how many times we have heard of a person from out of town coming to visit a friend living in a busy and noisy city and asking the friend how she sleeps with all that noise. The most common answer we get is: "*Oh, I don't even hear it*". But that is actually: "*Oh, I'm phenomenally conscious of it but not access conscious of it*".

- *Monitoring consciousness*: can be found in three different variants:
 1. Phenomenal consciousness of one's state (in other words, perception of one's state).
 2. Internal scanning. This is a simple functionalist view, an operating system doing its monitoring work falls in this category.
 3. Metacognitive definition: conscious state in which one is thinking about the effects of being in that state.

(Block, 2002) proposes to unify these three different definitions into one: *S is a monitoring state if and only if S is phenomenally present in a thought about S (in other words S is a monitoring state if it is conscious (i.e. has a representation) of being phenomenally conscious of S)*. Given this definition it seems that phenomenal consciousness could be seen as a type of monitoring consciousness because it seems counterintuitive to have a conscious state (in this case, a phenomenal conscious state) without having a representation of it (and having a representation of it makes it fall into the definition of monitoring consciousness). Ned Block then uses an example to show that it is still conceivable to distinguish phenomenal consciousness and monitoring consciousness (i.e. to be in a phenomenally conscious state without

having a representation of it): imagine two dogs, where Dog1 has only a perceptual state while Dog2 has a perceptual state and a representation of it. Then:

- * It is conceivable for Dog2 to be conscious and Dog1 to be not conscious, this shows that it may be the case that being conscious requires having a representation of the state (i.e. phenomenal consciousness is a subtype of monitoring consciousness).
- * However, even if being conscious of something implies being phenomenally conscious of it, it is plausible that one can be phenomenally conscious without being conscious. Therefore we can say that it is plausible for Dog1 to be phenomenally conscious but not conscious. So, it's possible that phenomenal consciousness is distinct from monitoring consciousness.

Ned Block concludes by mentioning that the presence of these multiple concepts of consciousness derive from the ambiguity of the concept itself. For some, consciousness is access-consciousness for others it's phenomenal consciousness. It could be that consciousness is a combinations of all those 3 types but it's useful to avoid confusing the concept of consciousness with any of the single types described above, and vice-versa we should not assume that any single type completely describes consciousness.

- **Mental event:** a mental event is any event inside our mind, like having a particular thought. For a materialist mental events are in a direct correspondence with physical events. Physical events are basically neuron firings.
- **Introspection:** is the ability to reason about one's own perceptions, knowledge, reasoning and in general any conscious mental event. For example, answering the question "Do you know if Clinton is sitting or standing?" re-

quires introspection about your reasoning to decide that you cannot know anything about Clinton's current posture.

- **Awareness:** often used as a synonym for consciousness. McCarthy in (McCarthy, 1995) gives the following definition for awareness and consciousness in a logic based reasoning agent: the set of sentences that are available for reasoning defines of what the agent is aware. The subset of those sentences that can be used for reasoning about themselves defines of what the agent is conscious. But after these definitions he assumes awareness and consciousness to be the same and that is what most researchers do.
- **Self-consciousness and self-awareness:** this is awareness/consciousness of the self, it is the state in which the thing that is self-conscious has an internal representation for itself that can be used for reasoning. For example a chimp or a baby (after about 1 and half years) is able to recognize itself in a mirror but a dog is not. Ned Block suggests that the mirror test may be a test of self-consciousness. Therefore a dog is not self-conscious but a chimp is. However, a dog can be phenomenally conscious and also access conscious³. Therefore, phenomenal consciousness, even though possessing a strong self flavor to it (one's experience), does not coincide with self-consciousness. The example also shows that there are many levels of awareness: a dog has a lower awareness of itself than humans because it doesn't recognize itself in the mirror. However a dog can be aware of the itch on one of its legs. Our experience with dogs suggests that dogs recognize other dogs using not only visual perception but other types of perceptions (like smell). Putting a mirror in front of a dog normally evokes little response (i.e., it is possible that the dog doesn't recognize another dog in the mirror, but

³The mirror test applied to a dog could be explained by saying that the dog is phenomenally conscious of its image reflected in the mirror but it interprets the representation of which it is access conscious as referring not to itself but to some other thing.

just ignores the moving image of itself because it doesn't possess the other sensory dimensions like, for instance, smell).

- **Qualia:** these are the subjective properties of our perceptions. For example as Daniel Dennett writes in (Dennett, 1988), how a glass of milk at sunset looks to you is a visual qualia, how it tastes to you is a gustatory qualia and so on.

2.2 Human consciousness

In this section we will describe what is known about human consciousness. Before going into more detail we will describe why researchers think consciousness has evolved in humans; Mandler in (Mandler, 2003) gives 5 reasons:

1. Consciousness can be considered as a scratch pad useful to improve the selection of actions because it allows the simulation of the outcome of different choices using more complicated and, hopefully, effective criteria. This scratch pad function also makes possible to try new actions never used before without having to test them in the real environment, which could be very dangerous.
2. Consciousness is mainly used on long span plans. With consciousness it is possible to analyze each part of a long plan and its relations with the other parts. Note that planners do exactly this and also what is said in the first point (at least to some extent) but that doesn't mean that planners are conscious. In fact here we are listing only some evolutionary advantages that may motivate the presence of consciousness in the human brain. We are not listing a series of conditions to classify a machine as conscious.

3. Consciousness helps in accessing data stored in long-term memory. The actual retrieval is unconscious but consciousness guides this unconscious process, making it more effective.
4. The consciousness machinery is used to build representations of the current activity that are used by many mental processes. Mental imaging, categorization and language are examples of mental processes that use representations generated by consciousness to accomplish their tasks effectively. This generation process is also called *binding* because it combines in structures information coming from different sources. Baddeley in (Baddeley, 2000) gives the interesting example of trying to imagine an “ice-hockey-playing elephant”. Presumably consciousness acts in conjunction with mental imaging, binding information from the sport of hockey and from the animal elephant to build this artificial image. Also for decision-making involving categorization, it seems that consciousness has to be involved. The same happens in the case of language understanding and even more apparently in the case of memory recall.
5. Consciousness seems to allow a better interaction with the environment
 - by enabling reflective reaction instead that instinctive reaction. Reflective reaction is better than pure instinctive behavior because it enables the improvement of procedural knowledge built from past experiences, increasing the likelihood of success in new situations. In (Baars, 1997) Baars gives an interesting example of the use of this capacity of consciousness: campaigns against smoking use the ability of consciousness to change procedural knowledge (i.e., unconscious mechanisms) to remove the dependency on cigarettes. Making the user conscious of the damage caused by smoking tends to bring to the conscious level the unconscious need to smoke, so that it can be changed.

- by reducing the number of sensory input through the use of a mental process called attention. The advantage of this reduction of information is of great importance (boosting computational efficiency) but how this is done is mostly unknown.

The layout of the remainder of this section is as follow: first we will report how philosophy looks at human consciousness, then we will describe interesting experiments designed to discover some properties of human consciousness. Finally we will describe some of the current attempts to build computational conscious architectures.

2.2.1 Philosophy

Human consciousness has been studied for a very long time, we can say from the first philosophers. It is considered one of the most difficult problems in the branch of philosophy called philosophy of mind. The literature in philosophy is extensive and we will give only a brief introduction to what are the main issues.

The first discussion topic is in trying to decide if consciousness is physical or not. The most important arguments against materialism⁴ are the following two:

- *The knowledge argument*: this argument described by Frank Jackson in (Jackson, 1986, 2002) images a woman, Mary, to grow up in a black and white room with no interaction with the outside world except for a black and white TV. In this black and white life she studies all about the physics and psychophysics of colors. After a while the scientists doing this experiment free her and when she sees a real color she says “*Oh, so this is what it is like to see red*”. The argument concludes that materialism is false because there is something more than physics in experiencing colors.

⁴Materialism is the view according to which every mental activity is composed and caused only by physical events and also causes only physical events.

The classic answers from the different flavors of materialism are:

- When Mary sees for the first time a color, she doesn't learn any physical fact, she learn an ability, that is the ability to distinguish colors.
 - Another way of answering is that Mary learns the same thing but with a different name. For example we could know everything about H_2O and only after, we learn the word water and that water is liquid, etc. When we learn these new facts they don't add anything to our knowledge regarding H_2O /water. We are just using another name for the same thing.
 - A third answer is that there are two types of physical facts: facts about the theory and facts about the physical objects. Mary knows everything about the first but not about the second and exactly the second is responsible of conscious experiences.
- *The modal Argument*: this is based on variations of Descartes' sixth meditation that argues that materialism is false because it is possible to imagine ourselves living without a body. More modern versions of this argument use the concept of zombies. A zombie is a being physically equal to a human but without consciousness. So if zombies could exist materialism is false i.e., consciousness is a consequence of some non-physical stuff. This is the argument taken by Chalmers against materialistic views of consciousness. The answers to this argument from materialists are similar to the H_2O -water parallel. In other words there is a difference in explaining mental and physical concepts but they are only different names for the same thing.

Similarly, instead of arguing that consciousness is physical it is possible to argue that consciousness is explainable in physical terms (using the modal argument as starting point). This hypothesis that consciousness could not be explained in physical terms is called the *explanation gap*. Some philosophers, like Chalmers,

take the direction of non-reductionism: in which consciousness becomes a fundamental property not reducible to other fundamental physical properties. Others instead accept the possibility that we will not be able to physically explain consciousness, not because it is not physical but because we have limited access to it.

According to this view, because of our limited capabilities to access our minds, understanding consciousness could be impossible even if consciousness is a physical process.

In the following section we will describe some experiments to discover some of the properties of human consciousness and some attempts to model human consciousness with computational models.

2.2.2 Computational models of human consciousness

Most of the available theories of consciousness are very similar and based on the same principles. (Baars, 1988) is probably the most complete and widely believed. This theory, based on the early work of Allen Newell in (Newell, 1990) that gave birth to the SOAR cognitive architecture (described in the same reference), takes its strength from a broad empirical base of data from years of experiments on human subjects. Other models have been proposed but typically they are less detailed because they are not so strongly based on human data; we will briefly describe some of them, in particular: Kuipers', Minsky's and McDermott's models.

Before describing the four models mentioned above, it is interesting to note one fundamental characteristic of consciousness: its limitations compared to the massive capacities of the unconscious parts of the brain. The two main limitations of consciousness are:

1. Working memory: consciousness is based on working memory, which many experiments have proven to be able to contain about 7 elements (plus or

minus 2). A very interesting discussion is by Miller in (Miller, 2003), which is reported right after this list.

2. We can be conscious of only one input stream at a time. This process is called attention or focus of attention and is easily verifiable by everyone. For example if there are two dialogues going on we can understand only one at any time. Or if we are counting a set of objects and someone starts speaking to us and we switch our attention to the conversation, we normally forget the number we were at during the counting.

(Miller, 2003) contains an interesting discussion and comparison of two limitations of our brains:

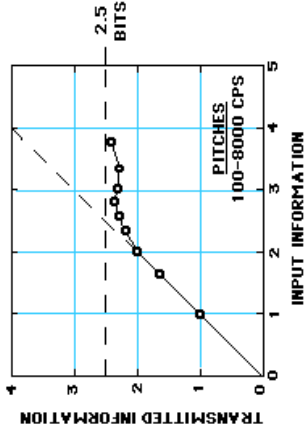
- The number of different clusters in which we can classify inputs;
- The capacity of our working memory.

Miller starts by showing data regarding the ability to classify unidimensional stimuli. These stimuli were unidimensional in the sense that the classification was asked to be done using only one property of the data (for example imagine to be asked to classify cars using only their color). In table 2.1 we reproduce the results of these experiments.

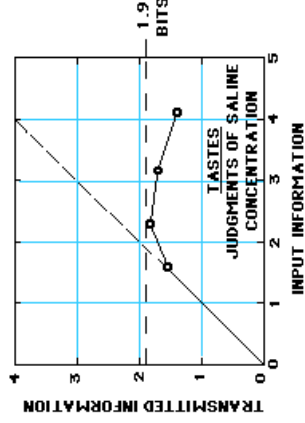
If the subjects were asked to classify multidimensional⁵ stimuli the performance increased less than linearly with the number of dimensions as shown in figure 2.1 (from (Miller, 2003)).

This means that we can exploit the presence of more dimensions (more features) in classification but our ability grows less than linearly with the number of features. Therefore adding features is not going to help us indefinitely.

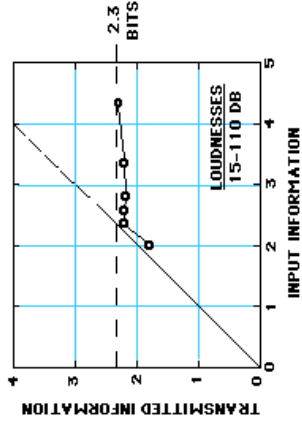
⁵multidimensional in the sense that more than one property was used to classify them. For example, imagine classifying cars using their color and transmission type (i.e., a two dimensional classification task).



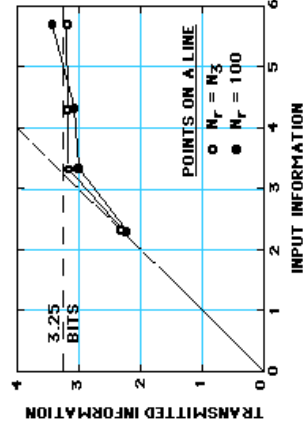
From (Pollack, 1952, 1953). The task was to classify the pitch of sound signals into an increasing number of buckets (clusters). On the abscissa is the \log_2 of the number of different input pitches. On the ordinate is the \log_2 of the number of different buckets identified by the subjects.



From (Beebe-Center et al., 1955). In this case the experiment consisted in classifying solutions based on the concentration of salt in them.



From (Garner, 1953). The task was of the same type as in the previous experiment except that now the subjects had to classify the loudness of sound stimuli.



From (Hake and Garner, 1951). Here the task consisted in classifying the position of a point in a segment.

Table 2.1: Performance obtained by human subjects classifying unidimensional inputs.

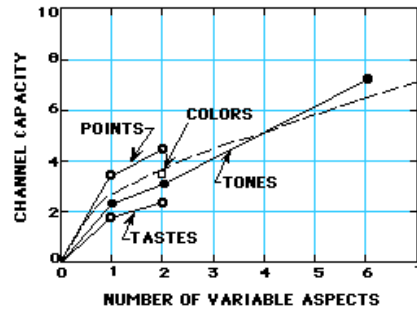


Figure 2.1: Performance obtained by human subjects classifying multidimensional inputs. The performance grows less than linearly with the number of dimensions considered.

Other experiments have shown that working memory has a limit in its capacity of about 7 elements. The fact that these two limits (capacity of working memory and ability to classify unidimensional inputs) are very similar and because the two functions are related suggested to Miller to check if the limit of working memory is also information-based as is the limit of classification. But experiments published in (Hayes, 1952) showed that working memory is not limited by the amount of information but only by the number of items.

Because working memory is not limited by the amount of information but by the number of items, our brain has developed a trick to be able to store in memory the same number of items even if each one can have a different information content: recoding. A clear visualization of this recoding property is given in table 2.2 from (Miller, 2003). Experiments show that recoding is directly correlated with the level of expertise in the particular domain. So an expert uses higher levels of encoding compared to a novice.

Baars' Global Workspace Theory

Baars' theory of consciousness, called *Global Workspace Theory*, explains consciousness as a theater where the main components are the following (visualized

binary digits to be memorized:	1	0	1	0	0	1	0	0	1	0	0	1	1	1	0	1	1	0
2:1 recoding: 9 elements	10 2	10 2	00 0	10 2	01 1	11 3	00 0	11 3	00 0	11 3	11 3	10 2						
3:1 recoding: 6 elements	101 5	000 0	100 4	111 7	001 1	110 6												
4:1 recoding: 4 elements	1010 10	0010 2	0111 7	0011 3	10 -													
5:1 recoding: 3 elements	10100 20	01001 9	11001 25	110 -														

Table 2.2: This table (from (Miller, 2003)) shows how recoding can be employed to reduce the number of items that have to be stored in working memory. This enables relaxation of the strong constraint on the number of items that working memory can contain.

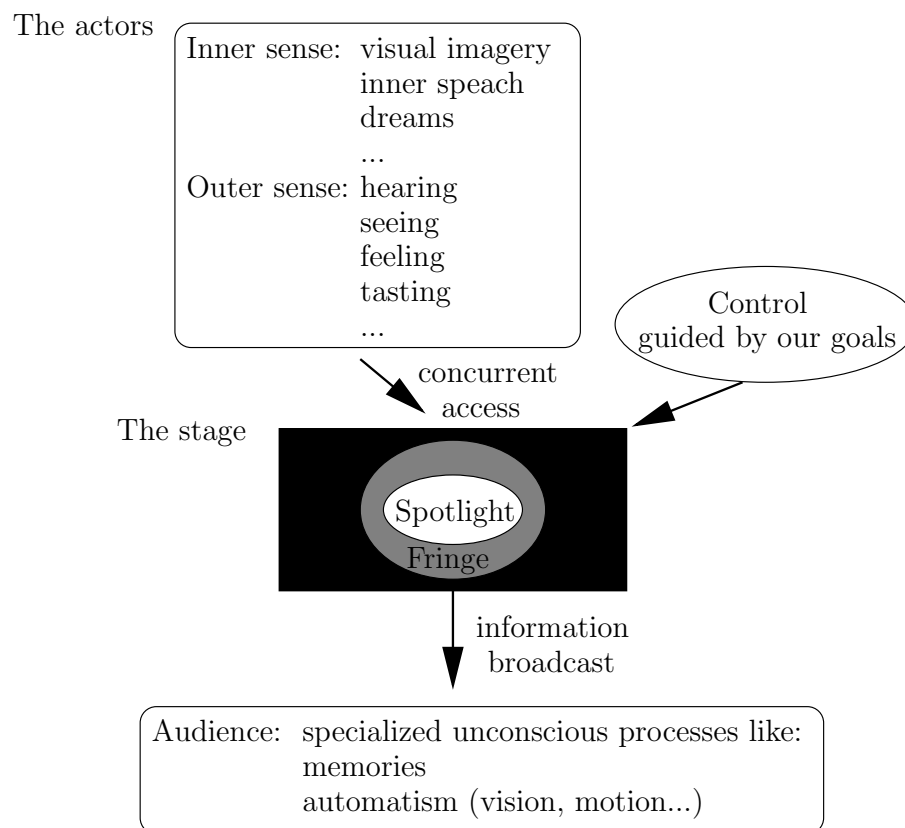


Figure 2.2: Baars' theater model of consciousness.

in figure 2.2):

- *The stage*: it is the working memory. It represents the limitation in capacity of working memory and the contention of all the actors to go on the stage (i.e., in accessing working memory). Working memory works as a serialization block and acts as a binding place where information from different sources converge and from which it can reach any other part of the brain (represented by the audience, see below).
- *Spotlight*: represents attention. Attention as previously said is fundamental to restricting sensory information that could potentially become conscious to the subset relevant to the current goals. This decision is mostly an unconscious process and is the function of the control block.
- *Fringe*: the fringe of attention represents information that is not conscious but also not unconscious. Some fringe phenomena are: feeling of knowing (FOK, the situation in which one clearly feels to know something even if unable to actually retrieve that knowledge), tip of the tongue (TOT, a particular case of FOK in which one is temporarily unable to retrieve a particular word), familiarity, like/dislike, coherence, rightness.
- *The actors*: they represent all the possible entities that can have access to consciousness. Examples are sensory information, visual imagery, inner speech, thoughts, etc.
- *The control*: as said, this block controls attention based on the current goals. Its internal functioning is mostly unknown.
- *The audience*: represents the set of all the unconscious processes that our brain provides. It could be considered similar to software libraries. Consciousness is called a gateway because it is able to connect to almost every part of the brain. This ability is well documented by the phenomenon

called *neurofeedback*. An interesting introduction to neurofeedback is given in (Evans and Abarbanel, 1999). Neurofeedback is used to cure various neurological pathologies like phobias, attention deficit, dissociation (also called multiple personality disorder), addiction (for example to alcohol), mood disorder (depression, mania, associated with unbalanced activation of the right and left frontal areas) and coma and other problems caused by open and closed head injuries. The principle of neurofeedback (also called biofeedback) was first introduced by Joe Kamiya in 1962 and basically consists in training patients to consciously modify particular characteristics of their own electroencephalogram (EEG) that are related to the pathology they are suffering. The experiment reported by Kamiya in 1962 was on a patient suffering from thanatophobia⁶. Kamiya cured him using a device that produced an auditory feedback proportional to the quality of the *alpha waves*⁷ of the patient EEG. This auditory feedback was used by patients to learn to consciously control their own level of alpha waves. The significance of neuro-

⁶Thanatophobia is the phobia of death or dying.

⁷According to frequency, brain waves (read using EEG) are classified in 6 bands:

- *Delta*: Waves with frequency ranging from 0.5 to 4 Hz. They are related to restoring/repairing functions of the brain (for example, they are present during sleep, coma and other brain injuries and pathologies).
- *Theta*: Waves with frequency ranging from 4 to 8 Hz. They are associated with pathological conditions of disconnection from reality but in normal individuals they are related to improvisation and creativity (they are an indicator of activity in the subconscious part of the brain).
- *Alpha*: Waves with frequency ranging from 8 to 12 Hz. They are associated with calm states of mind (a lower than normal power of the alpha frequency band is associated with stress, pain, fear, etc.).
- *Sensory motor rhythm (SMR)*: Waves with frequency ranging from 12 to 14 Hz in the sensory motor cortex. They are associated with muscular activity (their power is inversely proportional to the amount of muscular activity) and mind activity (thinking, learning, etc.). Energy in this band is low, for example, in epileptic patients.
- *Beta1*: Waves with frequency ranging from 12 to 16Hz. They are associated with high-level cognitive functions like thinking, learning, planning, problem-solving, etc.
- *Beta2*: Waves with frequency ranging from 16 to 24 Hz. They are normally associated with stress conditions.

feedback relative to consciousness as a gateway of the brain is prominent. In fact, neurofeedback shows how we can control very low-level characteristics of our brain, like motor neurons activity, through conscious effort.

From the empirical data collected on consciousness the following properties of it are known:

- Learning seems to occur automatically as consequence of our conscious attention, but the learning is done unconsciously. In the architecture above, this phenomenon is represented by the audience that, as in a real theater, learns from being exposed to what happens on the stage (or better, on the part of the stage on which the spotlight is shining).
- There is also implicit learning, where we learn (through an unconscious process) information related to what we are attending to. Baars gives, as example, the fact that a child when learning a language doesn't know anything about the concepts of noun, verb, etc. The child learns those concepts through unconscious processes that probably use correlations from many input examples (to which the child attended).
- Conscious events seem to automatically trigger sets of unconscious events. Learning is one of these, problem solving is another: given a conscious question we obtain a conscious answer through an unconscious mechanism. This is called the *incubation process*. Another effect in this category is called *priming*, that is: being conscious of something starts unconscious processes that later influence another conscious process; an example is disambiguating the meaning of a noun when previously given another noun as context.
- Consciousness allows us to get access to what we can control or perceive. In other words consciousness is the module that gives us the ability to control our actions.

But how do scientists discover properties of consciousness? Nagel in (Nagel, 1974) suggests that the problem of consciousness will be solved when we are able to answer the question: “*What is like to be a bat?*”. Baars in (Baars, 1994) instead gives an empirical method of analysis and responds to Nagel’s proposal by saying that at this point in time Nagel’s question is far too complex and in the future may even turn out to be unanswerable.

Baars’ proposed method is to use what he calls *contrastive analysis*. These are psychological experiments on humans with the goal of emphasizing the differences and relations between consciousness and unconsciousness. Several of these experiments are reported in (Baars, 1988). An example is the following significant experiment (see (Pani, 1982) for more details). The subjects were given a series of similar random geometric images, and they were then asked to memorize one of them (this image will be called *prototype*). Afterwards a series of classification tasks was initiated. Each task was to choose between two novel images the one more similar to the prototype. The difficulty of each task was strictly related to the differences between the two novel images to be classified: the more similar they were the more difficult the task was. The results showed that given a level of difficulty, the speed at which the subjects solved each classification task increased with time (i.e., with the number of solved tasks). As soon as the difficulty of the classification tasks increased the time needed by the subjects to solve the tasks increased, decreasing again after some time spent practicing. This behavior shows that the more a subject becomes acquainted with a particular task the less conscious effort she/he has to put into solving it. In other words, the classification *algorithm* and its data structures used by the subject “move” from consciousness to unconsciousness. The contrastive analysis here is the analysis of the relation between the conscious and unconscious representations of the prototype image. Because the prototype image can be considered to be the most important data structure of the classification algorithm used by the subject, Pani assumed that the

representation of the prototype image moves from consciousness to unconsciousness together with the classification algorithm. A hypothesis that can explain this finding is that information goes from consciousness to unconsciousness when its novelty and informativeness decreases and its predictability increases (the more the job becomes mechanical the less involvement of consciousness is required, and the more boring the job becomes).

Kuipers' Firehose Model

Kuipers in (Kuipers, 2005) proposes a model of consciousness and evaluates it against what Searle lists as the 11 features an artificial agent that wants to be called conscious must have (see (Searle, 2004)).

Figure 2.3 shows how Kuipers models an agent and its interaction with the world. In the model, $z(t)$ is the input vector coming from the agent's sensors. $x(t)$ is the world's state vector (world = environment+robot). $u(t)$ is the vector that feeds the robot's actuators. $F(\cdot)$ describes the dynamic of the world (it is unknown to the agent). $G(\cdot)$ describes the connection between the current state and what is read by the robot's sensors (also this function is unknown to the agent). m is the internal state of the robot. H_i is a *reactive control law* that computes the output vector u given the sensory input z .

Kuipers reminds the reader that the amount of data from the sensory input is impressive, impossible to completely track. To deal with this, various cognitive scientists have created the concept of a *tracker*. In Kuipers' model, a tracker can be described as the equation $m_k(t) = \tau_k(z(t))$ where τ_k is an individual tracker that produces the symbolic description $m_k(t)$ (that is part of the state m and depends on time t). Trackers can be generated top-down (from reasoning) or bottom-up (from features of the sensory input). A tracker binds the sensory stream with the corresponding symbols in the representation language (it is a dynamical system that tries to keep itself locked on its target in the sensory input). These symbols

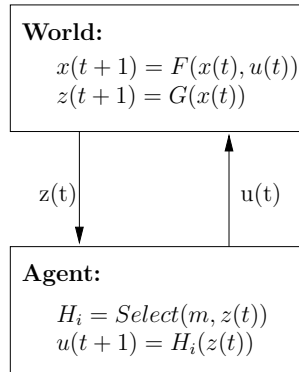


Figure 2.3: Kuipers' model of consciousness.

have time-dependent properties and these properties are set by trackers. For example, given that we want to look for a chair we (top-down) build a tracker for it that searches the sensory input for a chair-like structure and that will lock on it after having found one. The fact of being locked on the target produces the symbol of the chair and grounds it in perception and provides several attributes for it (e.g., its color, size and type).

For Kuipers a robot is conscious if:

- it has an architecture as the one in figure 2.3;
- it has a non-trivial collection of trackers m_k ;
- it has a non-trivial collection of control laws H_i ;
- it has a sufficiently realistic internal representation of F and G . This is important because the success of the agent's behavior is determined also by its ability to assess the effects of its actions. Ability that strictly depends on how accurate are the functions F and G . The ability to foresee what the effects of an action will be is important to effectively choose which actions to carry out to achieve the agent's current goals.

In the following, we summarize Kuipers' discussion of his model in relation to Searle's 11 features that any conscious thing should have. Each point is organized as follow: the name of the feature specified by Searle, then a brief explanation of it and finally Kuipers' interpretation of that concept in his architecture:

1. *Qualitativeness* (qualia): Searle identifies with this the fact that the experience of something (from the sensory input) is different from the experience of thinking about that same thing. Kuipers explains this difference with the enormous amount of information carried by the sensory input (for example, consider the number of colors that one can distinguish with the eyes versus the number of colors one can think of (i.e., for example, that one can describe with words)). Some of these qualia can be stored and can be activated by successive experiences generating this particular feeling of those qualia. An interesting comment, not strictly related to qualitativeness, is made by Kuipers regarding hallucination. Perception is not only bottom-up but also top-down, and hallucinations are perceptions created top-down (i.e., not grounded by specific sensory input). Sensory input helps in balancing top-down and bottom-up perceptions: it grounds concepts in perception so that these concepts are considered much more reliable than any other non-grounded concept.
2. *Subjectivity*: consciousness is strictly related to the feeling of self. This can be explained in this architecture by the presence of the control loop between z and u . Thanks to this, the agent can learn the effects of its actions, what is under its control and what is not. Among the sensory inputs those crucial to define the self are those regarding the robot (e.g., the sensory input regarding the position of each of the robot's arms or actuators).
3. *Unity*: the feeling of synchronisation between all the sensory inputs despite the fact that the processing time of each type of sensory input (sight, hear-

ing,...) is different. It seems that this feeling of unity is constructed from 20 to 500ms after the perception of the various inputs that compose an event. Unity could be explained by regarding this unitary conscious feeling as arising from forming a single coherent interpretation from the output of many modules. In Kuipers' model, the modules are the trackers each providing an interpretation of a specific portion of the sensory input.

4. *Intentionality*: this refers to the connection between the concepts in our mind and the real things to which they refer. Intentionality is provided by the trackers; they are the bindings between the symbols and their correspondent sensory perceptions. Given that trackers can be learnt autonomously, the robot can construct its own intentionality.
5. *Attention*: the ability to easily and quickly switch your attention to different things in your current perception stream (attention highlights portions of the background, see situatedness). This is handled by the trackers; once started they are locked on a target even if that is currently not the subject of attention. The trackers provide the set of things to which we could attend in the current input stream.
6. *Situatedness*: one is aware of one's position in time and space and of other more abstract properties that compose the situation (i.e., background) of one's current perceptions and conscious state. This could also be explained in terms of trackers, because they can also track things more abstract than objects in the sensory stream, like the current time or position.
7. *Active and passive consciousness*: distinction between things of which one is conscious but doesn't cause (e.g., some smell) and things of which one is conscious and causes (e.g., moving an arm). This could be explained in terms of the separation between input and output, things that one can

control using the output vector u and things that one simply perceives from the input vector z .

8. *The Gestalt structure*: objects appear as one thing even if overlapped or partially covered. Again, trackers have the property, once locked on something, of tracking their targets even after alterations of their initial perceptions (e.g., caused by partial occlusion).
9. *Mood*: mood, that is part of the input vector z (directly affecting portions of the state vector x), could be encoded in the functions F , G and the selection criterion for H_i .
10. *Pleasure/unpleasure*: these are associated to a reward scale learnt with experience and from evolution.
11. *The sense of self*: describes the particular feeling of perceiving oneself as a clear and demarcated entity. This can be explained by using the unitary and coherent interpretation of the input vector that is part of consciousness (see unity above), once created one can store it and reason about it, thus contributing to the identification of the self.

Summing up, Kuipers' model is less detailed and less grounded on facts than Baars' model. This model doesn't directly help to actually create a self-aware agent; most of the details are still left unexplained (for example, the use of the word "*non-trivial*" to characterize the requirements of consciousness on the models). However, this model is interesting because it tries to connect non-symbolic processes with symbolic processes (i.e., sensorimotor AI and symbolic AI) and it interprets the mind in terms of a classical control system approach to a dynamical system, providing a supporting argument in favor of the possibility of building fully human-like intelligent machines (this is the ultimate goal of what has been called "*strong AI*").

Minsky's Model

Minsky in (Minsky, 1988) proposes a very high-level architecture for the whole mind. As part of it there are chapters on consciousness, the self, introspection, etc. but the main concept that seems to emerge from the book is that too little is known to talk about how consciousness is organized and how it may be reproduced in machines. The fundamental idea that forms the basis for Minsky's model is that an artificial mind could be organized as multiple interconnected blocks each of which solving a very simple problem, but when properly connected they form the mind (this is the reason for "society" in the title).

McDermott's Model

McDermott in (McDermott, 2001) associates consciousness with systems that have a self-model and that produce beliefs. This is also called a *second-order system* because beliefs are sentences about propositions (e.g., "I believe that the tree is green", is a sentence expressing a belief about the proposition "that the tree is green".).

McDermott believes that qualia, the hard problem of consciousness, are not actually a problem. In fact, according to McDermott a quale is a primitive element of an artificial agent's mind that is not further explainable in terms of other entities. Each quale is identified by the behaviors in which it plays a role and these behaviors are defined by beliefs possessed by the agent. The key is that the external behavior of an agent must be coherent with that of entities believed to be able to have qualia. McDermott uses the example of a Robot that is trying to save humans that may be present in a burning building to describe how it can experience the quale of unpleasantness regarding high heat and how it shows it by the way in which it tries to avoid the portions of the building that are hotter while pursuing the goal of looking for trapped humans.

Another of the main problems of consciousness, free will, is explained by McDermott as the belief that the decisions taken by the agent possessing this belief are not caused only by external events (in this paragraph we refer to this axiom with the name *free-will axiom*). In other words this belief breaks the chain of causality and allows the robot to take decisions. To clarify this concept McDermott uses the following example: a robot, let's call it R, is in a room in which it perceives the presence of a bomb. Currently R is not planning to leave the room. How should R come to the conclusion to leave the room? The assumption is that the reasoning that should take place has this form: perception \rightarrow prediction of what will happen \rightarrow evaluation of the foreseen events \rightarrow generation of new actions, if necessary.

- Let's assume that R doesn't have the *free-will axiom*: given the perception of the bomb, R tries to make a prediction of what would happen. During this step, R has to make a prediction of what R, itself, would do (because R is an element of the world that R is trying to predict). Therefore R goes in an infinite loop of simulations of what R's behavior would be.
- With the *free-will axiom*: now when R tries to predict R's behavior its internal symbol has a special status compared to anything else in R's internal representation of the world: R's behavior cannot be determined (by pure simulation given its inputs, that are its perceptions). This conclusion keeps out of this prediction step the need to predict R's behavior so that the conclusion that R would be destroyed is reached (because R doesn't do anything different from what it was doing) and appropriate actions can be taken.

Additionally, McDermott answers various criticisms of the general idea of the possibility of having human-like intelligence in a machine (called strong AI). In particular, regarding the Chinese room argument, McDermott refutes it by grant-

ing that the person inside the Chinese room doesn't understand Chinese, but denying that this implies that the whole room doesn't actually understand Chinese. In other words, computationalism (i.e., the approach to AI using computation) says that understanding and consciousness can be implemented in weird ways, in this case a person that simulates a computer. This does not entail that the components used to obtain understanding or consciousness will understand or be conscious as well. So, Searle's argument refutes a claim to which computationalism actually is not committed.

From the present perspective McDermott reinforces the idea that a self-model is essential for consciousness. But as McDermott says, these are only hypotheses, and they should be tested in real systems.

2.3 Machine consciousness

In this section we will present some of the current ideas about how to make machines conscious. We will describe the work of John McCarthy (McCarthy, 1995) and Lenhart Schubert (Schubert, 2005). Additional research on conscious agents is covered in section 3, which is concerned with implemented systems.

McCarthy in (McCarthy, 1995) presents and discusses what could be the requirements, in terms of knowledge representation and reasoning, for self-aware machines:

- The machine should be able to observe its own situation in the world and the value of important parameters (e.g., the power supply status).
- It should be able to conclude that it knows or doesn't know something. McCarthy seems to equate "knowing" with "being able to infer". But then determining that one doesn't know something is equivalent to proving the

consistency of one's KB, which is impossible, according to Gödel's incompleteness theorem. To get around this problem (only in a partial way) McCarthy uses a very complicated second order formula (called *relative consistency*). We later mention a simpler proposal in the context of Schubert's work.

- A robot has to be able to reason about its abilities (for example, to improve them).
- It needs to record episodic knowledge of mental events.
- It needs to observe its goals (plans) and make inferences about them.
- It needs to be able to explain its actions.
- It needs to be able to use goals as constraints (parallel with Asimov's three laws as goals/constraints for a robot).
- It should be able to reason about its beliefs.
- It should have a way to represent the motivations for a belief.
- It should have the ability to reason about its mental state as an object. This single object is called a *context* by McCarthy (see (McCarthy, 1993)). Contexts are knowledge bases, an agent can have various contexts and it can reason in each of them. For example, a context could contain the beliefs of another agent. In each context sentences are interpreted differently according to the meaning of the literals in that particular context. McCarthy defines as the main function associated with contexts: $ist(c, p)$ that returns true if the proposition p is true in the context c .
- The robot should have a utility function that allows it to choose among different possible actions. Free will can be seen as this ability to decide which action to choose based on what maximizes your utility.

Lenhart Schubert in (Schubert, 2005) proposes a classification of different levels of machine self-awareness:

- Explicit self-awareness: human-like self-awareness that contains a full self model that can be used as any other knowledge during reasoning, is transparent (i.e., completely and easily examinable from outside) and is explicit (i.e., it can be easily seen from the interaction with the agent, this interaction should be preferably in natural language). There are no example of systems belonging to this category.
- Self-monitoring: agents that monitor and control themselves. This capability of auto-control doesn't imply the presence of a self model. Examples of self-monitoring systems are operating systems and any control device.
- Self-explanation: agents that are able to provide explanations of the reasons behind their actions. In this case as well there is no necessity of having a full self model. Examples in this category are expert systems.
- Global workspace or blackboard systems: these are systems that use a blackboard as internal data structure to mimic the architecture of human consciousness (see section 2.2.2). Examples in this category are SOAR (Newell, 1990), BB1 (Hayes-Roth, 1985) and IDA (Stan et al., 1998).
- Adaptive robust goal-directed systems: these are systems with goals and can change their behavior according to the environment to accomplish their tasks. Examples in this category are from nature: ants, spiders, viruses, etc. See (Mitchell, 2005) for an interesting discussion of this category.

After this introduction the paper goes over a series of knowledge requirements for having explicit self-awareness:

- *Logic*: the representation language plays an important role in making the goal of self-awareness not only possible but also feasible (i.e., to make it

not only theoretically possible but also practically possible). The paper proposes Episodic Logic (in short EL, see (Hwang and Schubert, 1993) for more details) as representation language because it has been designed with natural language in mind so that the actual interface with English will probably be easier to build and because it supports the requirements on the representation language that follow. In section 3.4 we will give a brief description of EL and highlight its features that allow for most of the requirements listed here.

- *Events*: events are important in the representation of historical information like biographic data and also for representing actions and their relations in time. Episodic Logic has special operators to represent events and their relations.
- *Attitudes*: it is important for a self-aware agent to be able to represent attitudes like beliefs, intentions, memories, etc., and to make inferences about what attitudes follow from some initial attitudes and/or other information. If “believing” (or “knowing”) is equated with “being able to infer” (as by McCarthy), it could be arbitrarily complex to verify a belief. But humans don’t do that; it is easy to see from personal experience that to determine whether we know something or not, we introspect briefly and if we get a positive answer we know it, otherwise we don’t. The method proposed in (Kaplan, 2000) does exactly that to avoid both the extremes of total ignorance and what is called logical omniscience. More specifically, the idea is that if we have to answer the question “*Do you know Φ ?*”, we start proving it and:

1. if we find immediately that Φ is false then we say that we know $\neg\Phi$;
2. if we find that Φ is true then we know Φ ;

3. if we are unable to conclude either of the previous cases, then we say that we don't know Φ . If we want to be more specific, we may use the amount of knowledge relevant to Φ we know to provide a feedback like: "I'll need more time to answer", or "I never heard anything about it".
- *Autoepistemic Inference*: that is inference about one's own knowledge. Consider, for example, a question like: "*Did someone call?*". In this case some possible answers could be:
 - "*Yes, you have received 3 phone calls*" if the agent's episodic memory has recorded those 3 events.
 - "*No, I haven't received any phone call*" this might be asserted not simply because the agent's episodic memory contains no record of a phone call but because it knows that all the time it was near enough to the phone to hear it and its sensory inputs were active during that period of time, and yet its episodic memory (which it believes doesn't lose information) contains no phone call events.
 - "*I don't know*" this might be asserted, for instance, if the agent knows it was sleeping all day.
 - *Generic knowledge*: this is specified in terms of probabilities and requires probabilistic inference. Because we still lack a well founded integration between inference and probabilities there are no definitive solutions. The solution implemented in EPILOG (see section 3.4) is an approximation and the computed probabilities could be inaccurate in various situations.
 - *Metasyntactic devices*: in many cases we would like a particular syntax for an answer; for example if we ask on what date the first extraterrestrial organism was found, we don't want the name of a variable that contains that value, or the sentence "*It is the date on which the first extraterrestrial*

organism was found”, as answers. We would like to be able to specify the syntax of the answer; in the case of the previous example we want a date as answer. As example, Schubert gives a summation problem. Suppose we ask to our agent, what is the sum of “*123+456*”. The agent should be able to answer with “*579*” instead of “*123+456*”, which strictly speaking is also a valid answer. Schubert proposes the following attachment axiom:

$$\begin{aligned}
 &(\forall_{term} x: \text{decimal-number-rep}('x')) \\
 &(\forall_{term} y: \text{decimal-number-rep}('y')) \\
 &(\forall_{term} z: \text{decimal-number-rep}('z')) \\
 &(x=y+z) \Leftrightarrow 'x' = \text{APPLY}('add', 'y', 'z'))))
 \end{aligned}$$

\forall_{term} is a substitutional quantifier that runs through all the terms in the logical syntax. *term* refers to all the objects that are terms accordingly to the definitions in the basic logical language. *decimal-number-rep* is a function that returns true if the argument passed is a decimal representation of a number. Quoting, *'x'*, means that we pass the name instead of the value (but the quotation is transparent to substitution for metavariables).

The previous axiom is given only as an example of the use of these meta-syntactic devices. In fact, we believe that the correct syntax of the answer should be inferred by the answering agent (using metasyntactic devices in this reasoning) using some notion of utility of the form (syntax) of the answer for the questioning agent instead that being given by an axiom. For every question there are many different answers to it but one answer may be better than another depending on the context.

- *Knowledge categorization*: we should be able to access knowledge about a certain topic. For example “*What do we know about trains?*” requires such access. Furthermore, it should be possible to pose this question in

the system’s internal representation. (Schubert suggests that the above metasyntax is suitable for doing so.)

- *Summarization*: we don’t want our agent to answer the question “*What did you do today?*” with a long list of actions. We would like the agent to summarize what it did today, taking account of how interesting something is (for the agent and for the questioner). This doesn’t simply mean selecting from a list of events the ones that are significant according to some criterion (that itself is a nontrivial task). Rather, we want a more general ability to represent a set of events as a single event that summarizes them, as in “*I did a lot of shopping and cleaning today*”.

The use of logic is motivated by the need for transparency. The requirement for a complex self-model and human-like general world knowledge and the preference for dialogues in natural language as the way to communicate with the agent motivate the remaining features listed above.

Summing up, Both McCarthy’s and Schubert’s requirements are intuitive and reasonable. Both go into detail about specific requirements for the representation language, like the need for representing events, quotation, substitutional quantification, context, etc. However, McCarthy’s approach to introspection using second order logic doesn’t seem feasible. Moreover McCarthy doesn’t talk about the issue of probabilistic knowledge and inference that seems to be necessary to reproduce human knowledge and reasoning. Objectively, we cannot judge the two approaches and produce a definite list of necessary and/or sufficient requirements for machine consciousness. We believe that these requirements will become more and more clear with the progress in the construction of conscious agents that are increasingly complex and hopefully increasingly similar to us. If this progress turns out to be slower than the advance of cognitive science in understanding the human brain, cognitive science will give additional insights into what could be the

requirements for consciousness.

3 Previous Work

In this chapter we will describe some historical and current systems that come closest to showing explicit self-awareness (including a capacity for natural language interactions). These systems are: SHRDLU, Homer, CASSIE and COMA. The first two are not longer under development, while the last two are more recent and under active development.

We will not talk about the numerous computational models concerned more broadly with modeling human cognition such as¹: SOAR², IDA³ and ACT-R⁴.

These architectures are not very relevant to our themes. Even though they are concerned with trying to simulate human cognition on a machine, they focus on low level details (like real-time processing) rather than on building an explicitly self-aware agent.

Section 3.1 is concerned with Winograd's SHRDLU. Section 3.2 describes Vere and Bickmore's Homer agent. Section 3.3 describes Shapiro et al's CASSIE project. Section 3.4 describes Schubert et al.'s COMA architecture. The last section, 3.5, focuses on comparisons and discussions of the four systems in relation

¹See <http://ai.eecs.umich.edu/cogarch0/> for a more complete listing of cognitive architectures and references.

²<http://sitemaker.umich.edu/soar>

³http://csrg.cs.memphis.edu/csrg/html/IDA/ida_home.html

⁴<http://act-r.psy.cmu.edu/>

to the requirements for consciousness as discussed in section 2.3.

3.1 SHRDLU

SHRDLU⁵ was the centerpiece of Terry Winograd’s Ph.D. thesis at MIT and was finished in 1972 (see (Winograd, 1972) for the complete thesis). Considering that it was built more than 30 years ago and the power of the machines available at that time (PDP 10 with 100KB of memory), the results achieved were impressive. It is in many respects comparable with the current state of the art. This is probably an indicator that we are still missing good ideas instead of machine power.

The structure of the program is sketched in figure 3.1. The functions of the blocks are:

- Monitor: this is a coordination program which starts and ends the processing.
- Input: processes the English input strings. It provides the input for the next block: the grammar.
- Grammar: does syntax analysis. It is written in “*programmar*”, one of the blocks of the system. For each English part of speech there is an associated function written in *programmar*.
- Semantics: gives the semantic interpretation of a sentence. This module helps the previous one in parsing a sentence. This is achieved by letting grammar fuctions call functions associated with the basic semantic units and vice versa.

⁵The name comes from the keyboard used before computer typewriting was introduced. Most of the modern keyboards are called QWERTY because those are the first few keys in the top row. Before computers the same position was occupied by ETAOIN and the second row by SHRDLU because the keyboard was built using the frequency of letters in English.

- Answer: produces the English output and stores the previous answers for reference.
- Programmer: is the parser core, implementing the specific language in which the grammar is written. It is a top-down left-right parser.
- Dictionary: each word has two parts: the syntactic part used by Grammar and a semantic part used by Semantics.
- Semantic features: this contains the list of features used for the semantic analysis of the input.
- Blocks: is the knowledge that describes the domain in which SHRDLU works, i.e., the Blocks domain (see figure 3.2 for a visualization of the world). The language used for expressing this knowledge is similar to first order logic.
- Mover: contains the graphic routines to simulate the actions of SHRDLU in the virtual world.
- Planner: this is the inference system and it is used not only for achieving goals but also during parsing and semantic interpretation. Again, the particular choice of implementing actions as functions makes it easy to call (and be called by) the two other main modules (the parser and the semantic analyzer).
- Data: contains the data of the current state of the world like color, size, shape and position of the blocks. To keep track of the events SHRDLU uses a special episodic-like memory with a very basic management of time relations. This special episodic memory was built specifically to be able to answer questions about the reasons for SHRDLU's actions.

The high-level capabilities of SHRDLU included the ability to:

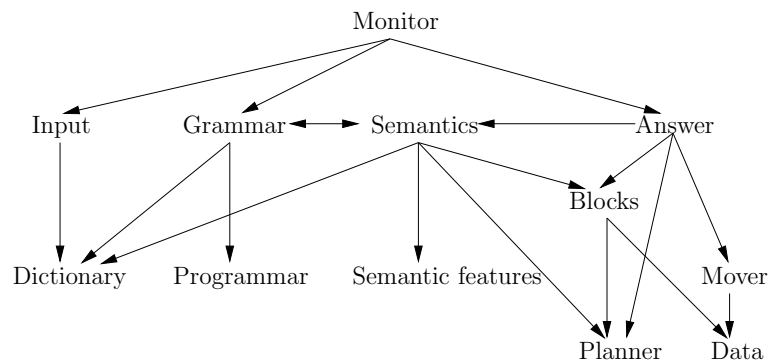


Figure 3.1: Structure of SHRDLU.

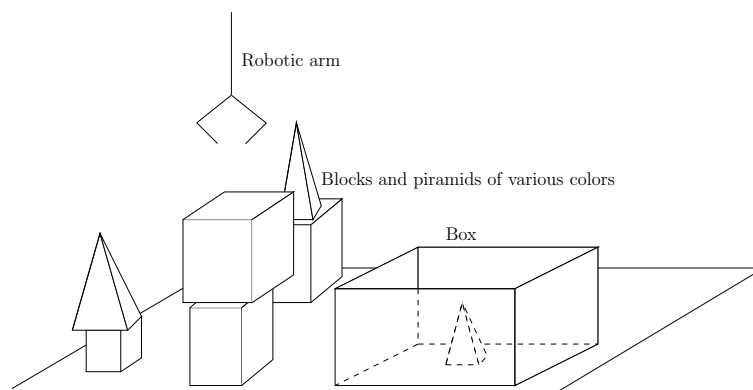


Figure 3.2: A representation of the world in which SHRDLU acts.

- understand dialogs about the block domain of which it had knowledge;
- resolve anaphora;
- distinguish objects according to relative position (above, on top of, etc.), color and shape;
- name objects and sets of objects defined in geometric terms (like: “*A steeple is a stack which contains two green cubes and a pyramid*”) and after being told definitions, be able to execute commands involving them (e.g., “*Build a steeple*”).
- explain its actions. For example it was able to answer questions like: “*How did you do it?*” (where “*it*” refers to a particular goal) or “*Why did you do that?*” (where “*that*” refers to a particular action).
- remember its actions to answer questions like “*When did you pick it up?*”.

Next we provide parts of the example dialogue contained in Winograd’s thesis.

User:	Pick up a big red block.
SHRDLU:	OK.
User:	Grasp the pyramids.
SHRDLU:	I don’t understand which pyramid you mean.
User:	Find a block which is taller than the one you are holding and put it into the box.
SHRDLU:	OK.
User:	Can the table pick up blocks?
SHRDLU:	NO. <i>This answer is obtained because the semantic interpreter failed in interpreting the command. Winograd makes the assumption that everything that SHRDLU cannot interpreter is false. This assumption is clearly wrong.</i>

Continued on next page

User: Can a pyramid be supported by a block?

SHRDLU: Yes. *This answer is obtained because SHRDLU finds an example of a pyramid that is supported by a block. From how SHRDLU is built the knowledge contained into the programs defining the allowed actions is not accessible for reasoning.*

User: Can a pyramid support a pyramid?

SHRDLU: I don't know. *This answer is obtained because the semantic interpreter doesn't find anything wrong but SHRDLU cannot find any instance of that relation.*

User: Stack up two pyramids.

SHRDLU: I can't. *This confirms that SHRDLU is not able to reason about its capabilities.*

User: How many things are on top of green cubes?

SHRDLU: I'm not sure what you mean by "on top of" in the phrase "on top of green cubes". Do you mean: 1) directly on the surface. 2) anywhere on top of?

User: 2

SHRDLU: Three of them.

User: Have you touched any pyramid before you put the green one on the little cube?

SHRDLU: Yes, the green one.

User: When did you pick it up?

SHRDLU: While I was stacking up the red cube, a large red block and a large green one.

User: Why?

SHRDLU: To get rid of it.

Continued on next page

User:	What did the red cube support before you started to clean it off?
SHRDLU:	The green pyramids.
User:	Call the biggest block “superblock”.
SHRDLU:	OK.
User:	Have you picked up superblock since we began?
SHRDLU:	Yes.

In the remainder of this section we will go into more detail about how inference works and its integration with planning. Inferences are made inside the planner module. Winograd critiques other theorem provers because they typically try to solve a problem by a blind search instead of attempting to guess which strategy will give a result with less effort. They don't resemble a human solving the same problem, as example he gave the following: How does a human prove that a Ph.D. thesis is acceptable? Assuming that the criterion used is that a thesis is acceptable if it is long or if contains a persuasive argument, he observed that a traditional theorem prover will simply run a standard inference procedure on its knowledge till it proves the answer. A human instead will use heuristics, for example, he will first check the index, if present, to see how many pages the thesis has. If this fails he will count the number of pages. As last resort he will read the thesis evaluating its arguments.

Winograd proposes to include these heuristics directly in the thesis evaluation procedure. In a normal theorem prover we might express the evaluation of a thesis in this way:

```
(FORALL (X) (IMPLIES
              (AND (#THESIS X)
                   (OR (#LONG X)
                       (EXISTS (Y)
                              (AND (#PERSUASIVE Y))
```

```

                                (#ARGUMENT Y)
                                (#CONTAINS X Y))))))
    (#ACCEPTABLE X)))

```

The theorem prover given the goal `(#ACCEPTABLE :SAM-THESIS)` would try all its knowledge in search of a way to obtain that formula (or if it uses resolution it will assert the negation and search for a way to find a contradiction). Note also that there is no straightforward way of naming this formula with something like `EVALUATION-OF-THESIS`. Winograd instead expresses the evaluation of a thesis looking for its acceptability as a procedure (executed by the planning module) in the following way:

```

(DEF THEOREM EVALUATE
  (THCONSE (X Y) (#ACCEPTABLE $X)
    (THGOAL (#THESIS $X))
    (THOR
      (THGOAL (#LONG $X) (THUSE CONTENTS-CHECK COUNTPAGES))
      (THAND
        (THGOAL (#CONTAINS $X $Y))
        (THGOAL (#ARGUMENT $Y))
        (THGOAL (#PERSUASIVE $Y) (THTBF THTRUE))))))

```

where `THOR`, `THAND` are the respective versions of `OR` and `AND` that allow for failure driven backtracking. `THGOAL` identifies a goal for the planner to achieve. The theorem says that to evaluate if `X` is `#ACCEPTABLE` it should first decide if `X` is `#LONG` and to do so it should try to use `(THUSE)` two other theorems: `CONTENTS-CHECK` or (if the previous fails) `COUNTPAGES`. If that fails it should try to prove that `X` contains a `#PERSUASIVE` argument. `(THTBF THTRUE)` specifies that in that case the planner should work like a theorem prover in finding if `Y` is `#PERSUASIVE`, i.e., it should try everything it can (no `THUSEs` are given).

Another advantage of this structure is that it supports changes to the state of the world by allowing the removal and addition of facts to it. As in `STRIPS` or `PDDL` there are positive and negative effects where positive effects are added facts and negative effects are deleted facts; `SHRDLU`'s language can represent this by using the special commands `THASSERT` and `THERASE`. Whenever some assertion is made (using `THASSERT`) or something is removed (using `THERASE`) other theorems are called to add the consequences of these changes (these theorems are called `ANTECEDENT` in the case of assertions and `ERASING` in the other case).

To sum up, SHRDLU has various interesting features like a strong connection between syntax, semantics and inference (planning). Its organization is probably not easily extensible. Our knowledge of it doesn't allow us to give a definitive answer on this topic, but presumably Winograd or others would have undertaken major extensions in the decades that followed, if these were straightforward. There have been various attempts to try to resuscitate SHRDLU but none has been completely successful. In later writing, Winograd takes a quite negative view of the feasibility of human-like understanding by machines (see (Winograd, 1990)).

Looking at SHRDLU dialog examples, the most impressive characteristic is the efficacy of the parsing and semantic interpretation. But it is hard to judge whether this quality could be maintained with larger grammars and lexicons. The problem of scalability is a common one, as will be seen in the case of Homer (section 3.2) as well. In addition, its approach to English generation is quite simple and based on patterns and simple transformations of these patterns.

On the self-awareness side, the most striking feature of SHRDLU is its ability to give explanations for its actions. But, as said in the comments on the reported demo dialog, this is achieved by a mechanical process of unwinding the goal state, and by reference to a very simple episodic memory, instead of by inference. We will have more to say in section 3.5 about the extent to which SHRDLU, as well as the other three systems to be discussed here, meet the requirements for self-awareness.

3.2 Homer

Homer (Vere and Bickmore, 1990) was implemented by Steven Vere and Timothy Bickmore over the time from 1987 to 1990.

Their initial goal was to build an agent with the following abilities:

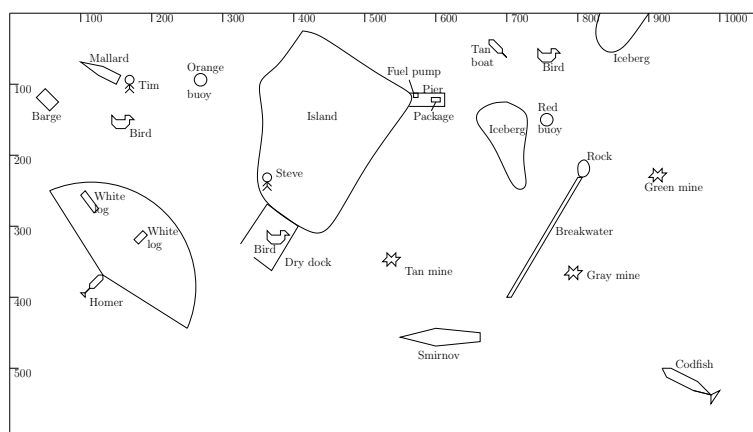


Figure 3.3: A representation of the virtual world in which Homer acted.

1. Communicate in natural language;
2. Plan;
3. Reason;
4. Act and perceive in a virtual environment;
5. Reflect on its own experiences.

Homer was an autonomous submarine able to communicate with humans using simple natural language. The virtual world in which Homer operated is depicted in figure 3.3.

As exemplification of Homer's capabilities we provide several fragments of various dialogs included in (Vere and Bickmore, 1990). In the first of these fragments Homer shows: its ability to understand basic English sentences and to execute commands, a basic comprehension of geometrical relations and the ability to answer questions regarding its knowledge. In addition, the dialogue demonstrates one of Homer's reflective processes in action: memory of its perceptions. This reflective process allows Homer to spontaneously communicate relevant things it has seen.

Steve: What is in front of you?

Homer: A log.

Steve: Do you own the log?

Homer: No I don't.

Steve: The log belongs to you.

Homer: Oh⁶.

Steve: Cows eat grass.

Homer: I know.

Steve: Lose the log.

Homer: OK. Sorry⁷.

Steve: Turn away from the log.

Homer: OK. I'm turning 34 to 191 [degrees]. (after this the authors, without being seen by Homer, drag away the log⁸.)

Steve: Do you see the log?

Homer: No, I don't

Steve: Turn around.

Homer: OK. I'm turning from 191 to 18.

Homer: I've lost the log. It has disappeared.

Steve: Where is your log?

Homer: I don't know.

Steve: Go to the rock.

Homer: OK. I'm going to the rock.

Homer: I've found the log.

Continued on next page

⁶Homer answers with "*Oh*" when it receives new information.

⁷"*OK*" is answered by Homer to a command given by a known human. If the command turns out to be not executable then Homer answers with "*Sorry*".

⁸Observers of Homer's virtual world can interact with the world through the mouse.

Homer: I've reached the rock.

In the next dialog Homer shows its ability to reason with future plans and the future consequences of these plans.

Tim: Drop the package at the barge next Saturday at 9pm.
Homer: OK.
Tim: Are you going to be at the pier next Saturday?
Homer: Yes.
Tim: Why are you going to go to the pier?
Homer: I'll go to the pier in order to pick up the package.
Tim: What time are you going to pick it up?
Homer: 8:56pm.
Tim: Where will the package be next Sunday?
Homer: At the barge.
Tim: Will the package be at the pier a week from now?
Homer: No.

The next dialog shows Homer's reaction to contrasting commands.

Steve: Go to the island.
Homer: OK.
Steve: Don't leave the island today.

Continued on next page

Homer: OK.

Tim: This is Tim.

Homer: Oh.

Tim: Take a picture of the Codfish.

Homer: OK. Sorry. *After realizing that to take the picture it should disobey the previous command, Homer answers with “sorry”.*

Moreover, Homer is able to understand commands in the form: “*If something then do this*”, such as “*If you see an animal tomorrow, photograph it*”. In (Vere and Bickmore, 1990) nothing is mentioned about mechanisms to distinguish between different levels of importance of various goals. For example, given the goal to drop a package at 9pm at the barge and the previous conditional command, we don’t know what the reaction of the agent will be if it sees an animal when it is swimming toward the barge with the package. Because taking the picture of the animal could take too long, what should Homer do? Take the picture and drop the package late, or not take the picture?

A schematic overview of the agent architecture is given in figure 3.4. In the following sections we will describe in more detail each block depicted in that figure.

Text interpretation and generation

First we describe the text interpreter module. The ability to communicate in natural language was chosen because of its convenience and naturalness for human users. To make the natural language understanding module feasible the authors made some simplifying assumptions:

- Vocabulary: there were about 800 words in the lexicon implemented at the time the paper (Vere and Bickmore, 1990) was written. The authors started

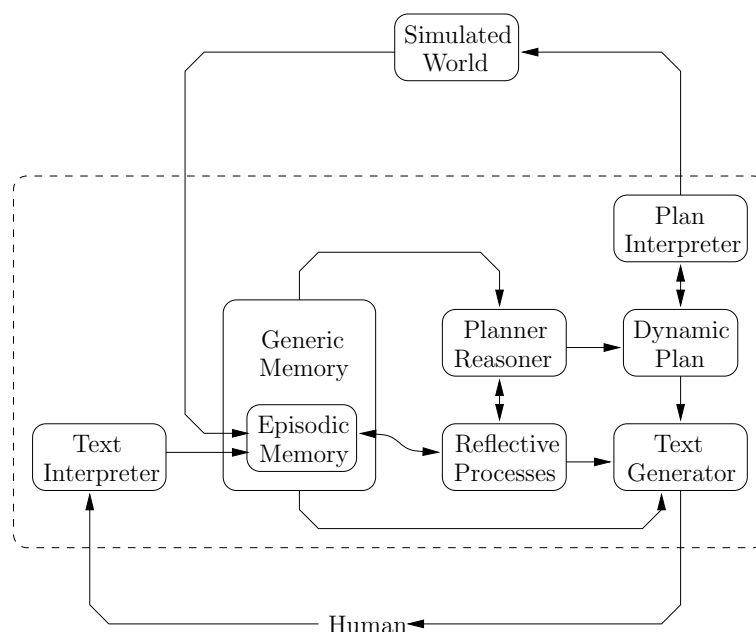


Figure 3.4: A schematic representation of Homer’s internal architecture.

with the idea of inserting into Homer’s lexicon roughly 1900 words obtained from Ogden’s Basic English vocabulary (see (Ogden, 1932)) and the 1000 most frequent words of English. The main goal of Basic English was to allow a person to express almost any meaning using a very small number of words. However, Basic English is awkward to use, so by adding the most commonly used English words to Basic English the authors would have allowed Homer to understand a broad class of commonly used sentences (in principle).

- Syntax: the authors allow for a quite broad class of sentences (listed in the paper), so the syntax is not the main limitation of the text interpreter module.
- Use: to remove some of the ambiguities the authors made some usage assumptions. In the paper only one is directly mentioned and concerns prepositional phrase attachment: in the case of noun attachment the prepositional phrase has to be preceded by “which”. In case of verb attachment, the al-

lowed prepositions are directly specified in the grammar.

Another usage assumption, not mentioned directly in the paper, is the use of the sentence “*This is* $\langle Name \rangle$ ” to start a conversation with Homer. In this way they achieve the advantages of clearly identifying the beginning of a conversation and immediately obtaining the name of the human user.

The parser used by Homer has the interesting feature of providing the set of possible semantic interpretations after every word. In other words the parsing is an on-line process running in parallel with the typing. First, this has the intrinsic advantage of masking some of the time required for parsing and the construction of the semantic interpretation with the time required by a human to type the sentence. Second, this is the way in which our mental lexicon seems to work, as several experiments have shown (Abney, 1989; Konieczny, 1996). However this parser still misses a feedback mechanism from higher-level layers (general world knowledge and current context) that has been shown to strongly improve parser performance, see (Stoness et al., 2004; Stoness, 2004).

The virtual world

The simulated world module created the virtual environment for Homer. The environment provided Homer’s perceptions and executed its actions. Because the virtual environment and Homer’s interaction with it wasn’t the main goal of this work, the environment was designed to be very spartan and with some simplifications that made Homer’s sensory perception fairly trivial. For example, the field of view of Homer was simulated as an arc of circumference 120° wide and with a fixed radius. No three-dimensional view was considered. An object is perceived by Homer as soon as one point in the set of key points characterizing the object entered the region defining Homer’s field of view (these key points were predefined for each objects, e.g., if the object was a square, the key points could be

its corners). The perception routine was executed every two seconds; no attempt was made to generalize/summarize the information coming from the perception process. Therefore the position of an object entering the agent's field of view was repeatedly stored every two seconds until its exit from the field of view.

The world (see figure 3.3) was composed of a set of fixed objects and a set of movable objects. Homer had complete knowledge of the position of the fixed objects. For movable objects, only the positions of the ones in Homer's field of view were known.

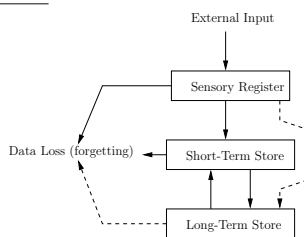
Homer's actions were perfect, meaning that if Homer issued a command to move forward of 1 meter, it moved forward, in the virtual world, by exactly that amount. But in this simple world actions could also fail because of external interactions. In fact it was possible to directly influence the virtual world while Homer was "living" in it. In particular it was possible to change the position of movable objects.

Memory

The memory module was logically divided into two sections: a general memory and an episodic memory. The general memory was used to store general world knowledge. The episodic memory was used to store all the perceptions and events (like hearing a sentence or executing an action), in other words all the knowledge in which the agent was a subject. Vere and Bickmore emphasize the importance of episodic memory in relation to consciousness using the example of subjects affected by Korsakoff's syndrome. These persons (as described in (Jacoby, 1982)) have an impaired episodic memory and a normal short and semantic memory⁹ and show a reduced level of consciousness. For example, if asked to recognize if a given word belongs to a previously seen list of words, a Korsakoff patient answers

⁹One of the first and most influential model of human memory is Atkinson-Shiffrin's *modal model* of memory. It was first published in 1968 in (Atkinson and Shiffrin, 1968). It organizes human memory as in the following figure:

correctly with about the same accuracy of a normal person but is not aware of the correctness of the answer. For him/her, it feels like guessing. But at the time Jacoby wrote this article, the existence of a distinct episodic memory was still only a theory and many researchers were skeptical about it. Even more vague was the reference to consciousness. A more recent survey on episodic memory by Tulving (see (Tulving, 2002)) gives an historical overview of the changes in



The *sensory register* stores information from the sensory inputs for a brief period of time (around 1 second). If the information is not used within that time it is lost. The content of the sensory register that is under the focus of attention reaches the *short-term store*. Here data last around 30 seconds if not rehearsed by some of the short-term store control processes. These control processes also regulate the transfer and retrieval of information from short-term store to long-term store and vice versa. A dotted line means that the presence of the communication channel represented by that dotted line is not clear. For example it is believed that long-term storage has unlimited capacity and loss of information actually corresponds to failure in the search process.

In the following years various other models have been put forward but the basic organization remains the same. What is changed in the new models is a more detailed characterization of what is stored in the memory and how the information flows. For a summary of the evolution of the modal model see (Izawa, 1999).

Baddeley and Hitch in (Baddeley and Hitch, 1974) subdivide the short-term storage into three blocks: a *phonological loop* that contains auditory information, a *visuospatial sketchpad* with similar function but on visual content and a *central executive* control system. The first two are actually slave systems of the central executive system.

Long term memory as described in (Izawa, 1999; Endel Tulving, 2000) is divided into implicit and explicit memory. The first is a procedural memory whose content is very difficult to translate into words. A typical example of implicit memory is a skill (like the skill of recognizing a musical note when it is played). Explicit memory instead is everything we can put into words, like general knowledge and events. Tulving in (Tulving, 1972) introduces the term episodic memory, dividing the content of explicit memory into episodic and non-episodic, or semantic, memories. Two important entities associated with episodic knowledge are the feelings undergone by the beholder and the time relations of that episodic knowledge with other episodic knowledge. In describing this, Tulving compares episodic memory to a “mind time-machine” that allows us to go back in time and almost fully re-live an experience. On the other hand, semantic memory contains knowledge like that necessary to answer the question “*What is the capital of France?*”. Another terminology often found in the literature (see (Gardiner, 2001)) is the parallel between *remembering* and *knowing*, the first associated with retrieval from episodic memory, the second associated with retrieval from semantic memory.

the understanding of episodic memory in the past 30 years and reports experiences from experiments on rare subjects with brain injuries that caused a loss of episodic memory while leaving all other brain functions almost intact. Currently the majority of researchers in cognitive science believe that episodic memory is a fundamental and dichotomized part of long term memory. Consciousness has also been divided into *noetic* and *autonoetic* consciousness. The first is awareness of knowing and it is associated with the part of long term memory called semantic memory. The second, autonoetic, is consciousness of previous events and it is associated with episodic memory. In addition to this long term episodic memory, Baddeley in (Baddeley, 2000) hypothesizes the presence of an *episodic buffer* in short term memory, with the very important function of binding information of different types present in short term memory and long term memory.

Coming back to Homer's memory, the general memory could be updated simply by telling the agent new knowledge. The agent assumed that everything said by the human speaker was true. No reasoning was attempted to understand whether an input was meaningful. For example, if the human speaker told the agent an incorrect position for a particular movable object and subsequently Homer saw that particular object in a different position, Homer would have simply stored the new position without even checking if the velocity at which the object would have had to move to reach the observed position starting from the communicated position in the given amount of time was feasible or not.

The episodic memory, as previously mentioned, was automatically updated through perceptions, events and reflective processes. Reflective processes are routines running in the background that are not in control of the agent's "consciousness" (they are the equivalent of unconscious processes in our brain). Examples of possible reflective processes are:

- Recognition of repeated information: this is motivated by the fact that an agent should become aware of repetitions in its own behavior or that of

related objects, to reason about those repetitions and optimize, if needed, its actions (for example to reduce useless repetitions) or recognize patterns. At the time the paper was written the state of implementation of this feature didn't allow clustering of the repetitions into semantically different groups. That is, the count of a particular type of event done by a particular object was like a global counter never reset to 0. Clearly this is not the behavior we would ultimately want. For example, the same sequence of “moving forward of 1 meter” events could mean “moving toward the fuel pump” or “moving away from the fuel pump” at different times. In general one would like Homer to cluster repetitions into meaningful clusters (like the two sentences given above) instead of just one cluster with the trivial meaning of “sequence of forward moves”.

- Elimination of useless information: like garbage collection this is an undecidable problem¹⁰ and wasn't implemented in Homer. The idea was motivated by the fact that humans tend to forget irrelevant facts (or facts that after some time become irrelevant). However, we are inclined to think that a machine would need this functionality only when the supplied memory becomes nearly full. Otherwise a machine should be able to use its “photographic” memory as a feature without trying to emulate the “defects” of human memory. Nevertheless, relevance is an important concept that is useful (for example for consciousness) independently on the presence of this reflective process. In other words, machines should be able, if necessary, to recollect the details related to an event, but these details shouldn't interfere with the speed of the reasoning process. This goal could be achieved by limiting retrieval of facts for reasoning to relevant ones as far as possible.

¹⁰We would like to point out here that being undecidable doesn't mean insoluble. In fact, many undecidable problem have practical approximative, but satisfactory, solutions. Ultimately question-answering is also undecidable.

- Summarization: as already mentioned, episodic memory has the tendency to become crowded with lots of events that are closely related and that could be efficiently represented in more concise ways. In Vere and Bickmore’s paper summarization is mentioned as the reflective process in charge of the production of high-level information from low-level facts. For example, if an object in the agent’s field of view moved from outside a box to inside it and then again went out of the same box, the episodic memory would contain the position of the movable object every two seconds. The summarization process, given this sequence of positions, would add high-level information like “*The object is **outside** the box*”, “*The object is **inside** the box*” and “*The object **left** the box*”. The possible inferences made by this reflective process were hard-coded and so could not be dynamically changed. Examples of the high level concepts produced¹¹ are: *reach*, *return*, *leave*, *enter*, *pass*, *find*, *lose* and *disappear*. To each of these events a precise pattern was associated. In general summarization is a very hard problem, reducible to language understanding, or using the words of Martha Palmer, it is an AI-complete problem. In this case it was reduced to a much more tractable problem by restricting the domain to changes in object position and to a fixed set of high-level events.

Besides the episodic memory described above, Homer also had an additional episodic memory used by the planner. This memory contained the current planning problem. At every invocation of the planner process this memory was reset. This planning memory was unconscious, so it was impossible to ask questions about its contents, such as ordering relations between actions or the reasons behind the choice of a particular action.

¹¹The behavior of this reflective process can be easily seen as equivalent to the operation of systems based on production rules, like (Newell, 1990).

Planner

Finally we describe some of the most important modules of the system, the ones that provide its planning capability and that integrate this capability with the natural language understanding modules. The authors encountered problems in the direct conversion of English into FOL, called “*Logicalized English*”¹², because the latter was not expressive enough to represent the meaning of many verbs. They considered verbs in particular because planning is based on actions and usually actions are expressed in English with verbs. For example the sentence *A man gives a coin to John* could be converted into $\exists x \exists y. \text{man}(x) \wedge \text{coin}(y) \wedge \text{give}(x, y, \text{John})$. But this conversion doesn’t say anything deep about the semantics of the verb *give*. In particular for the purpose of planning it doesn’t specify what the preconditions and the effects of the “*give*” action are. In other words it doesn’t define how the action modifies the world.

Another possibility for representing verbs considered by the authors was the frame formalism. Frames are a classic representation formalism in AI but they don’t remedy the weaknesses of the previous method. In fact a typical frame for the example above could be the following:

```
GIVE1 isa GIVE with
      agent MAN1
      thing COIN1
      recipient JOHN}
```

Clearly this doesn’t provide a deeper representation than the Logicalized English method.

Vere and Bickmore’s deliberations about action representation were very much influenced by Roger Schank and Yorick Wilks, as well by STRIPS (Fikes and Nilsson, 1971). Though Vere and Bickmore don’t explicitly mention STRIPS their ideas about action and planning were certainly grounded in STRIPS, since Vere’s earlier work had been based on his DEVISER system (Vere, 1983), a generalization

¹²For lack of details in Vere and Bickmore’s paper we assume that by Logicalized English they meant some FOL approximation of the content of English sentences.

of STRIPS allowing for temporal constraints. They also mention (Cercone and Schubert, 1975) (on state based action representation) as relevant work, and we note that (Allen, 1981) makes a close link between language and STRIPS-like action representation.

Despite the shortcoming of frames pointed out by Vere and Bickmore, they ended up using a frame-like approach. However, they incorporated ways of representing state transitions and temporal relations to adequately serve their purposes. We believe the same would be true for Logicalized English. If opportunely extended or augmented with axioms about the effects of actions, Logicalized English could solve the representation problem for action verbs (at least to the same extent as Vere and Bickmore’s solution, keeping in mind that there are as yet no definitive solutions to knowledge representation for natural language understanding).

In conclusion, Homer’s most interesting features are the planning capability (the planner also has a re-planning capability that is essential considering that Homer acts in a dynamic environment), the episodic memory and its reflexive processes (even if currently they are at a very basic level).

Homer’s main problems are scalability issues, both on the lexicon/semantic side and memory access side. Another problem is the separation between planner memory and the general/episodic memories that make impossible the explanation of Homer’s actions.

3.3 CASSIE

CASSIE (the Cognitive Agent of the SNePS System, an Intelligent Entity) is the name for the architecture designed by Stuart C. Shapiro and aimed at the long term goal of achieving natural language competence.

The architecture of the system is shown in figure 3.5. Its main modules are:

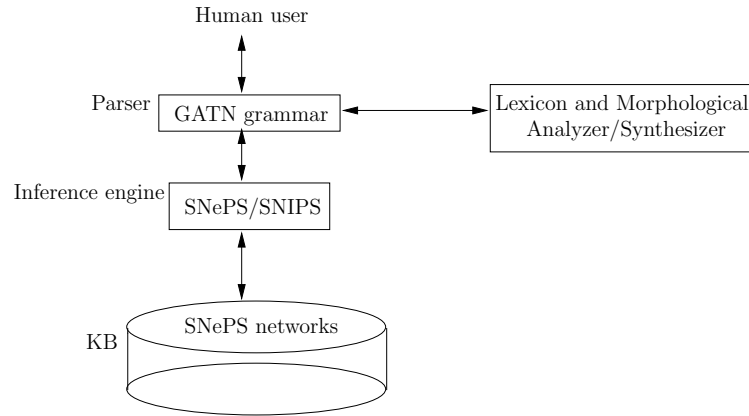


Figure 3.5: The CASSIE architecture.

- GATN, a Generalized Augmented Transition Network that takes care of translating English into Semantic Networks that can be used by the inference and knowledge representation module (see (Shapiro, 1979)).
- SNePS/SNIP (Semantic Network Processing System/SNePS Inference Package) is the knowledge representation and inference engine (see (Shapiro, 2000)).

This architecture has been used in various tasks to study different aspects of natural language understanding, generation and inference (see (Shapiro, 1989)). The project most relevant to our topic is the one in which CASSIE was embodied in a simulated/real¹³ robot (see (Shapiro, 1998)).

The architecture of the embodied CASSIE is depicted in figure 3.6. The functions of each level are as follows:

- The knowledge level represents the conscious reasoning and is implemented using the CASSIE architecture illustrated above;
- The perceptual-motor level takes care of unconscious reflexes.

¹³They tried several different simulation environments with different levels of complexity and a real robot.

KL (Knowledge Level)
PML (Perceptual-Motor Level) PMLa - high level (LISP) PMLw - interface level PMLc - low level (C)
SAL (Sensory-Actuator Level)

Figure 3.6: The embodied CASSIE architecture.

- The sensory-actuator level is the lowest level where hardware details are implemented.

Only the lowest levels (PMLw, PMLc and SAL) must be changed when CASSIE’s “body” is changed.

In this research Shapiro and his team studied various aspects of consciousness:

- *Symbols grounded by perception:* Shapiro talks about two ways of showing understanding of a symbol (called also a concept): using that concept in language, or using it in action. With the embodied CASSIE the author focuses on the second way to show understanding. In the simplified world in which CASSIE operates, objects can be uniquely distinguished using just a pair of attributes, `<shape, color>`. To ground a symbol, such as **B6**, to its corresponding object in the real world, a green round thing, CASSIE associates to **B6** a description that can be used by low level functions to recognize that object. In this case the low level description `<shape, color>` is not available to the KL level and can be used only by the functions in the two lower levels. Accordingly to Shapiro, the reason that these pairs are not accessible to the KL level is that they are only links between the concepts at the KL level and their counterparts in the real world.

- *Symbols grounded by action*: not only concepts of material things can be grounded in the real world, but also concepts of actions. The embodied CASSIE described in (Shapiro, 1989) had only 5 actions: **talkto**, **find**, **goto**, **follow** and **stop**. To ground an action the author simply associates the action to a function that executes a simplified version of the expected behavior. The system has the ability to reason about more complex actions. In fact, it is possible to specify in the knowledge base what the preconditions of an action are and what goals an action achieves (similarly to what is done in planning). It is also possible to specify how to achieve a complex action using a set of basic actions (a plan).
- *First-person privileged knowledge*: this refers to the knowledge that one has because of one's privileged access to one's internal state and decision processes. For example, consider the level of charge of the robot's battery. An external observer has to reason or examine an indicator to estimate the amount of charge left, but the robot possessing that battery knows about this directly, if properly equipped (i.e., if conscious of the battery status). In the case of CASSIE, for example, every action is stated internally so that CASSIE knows that it is doing that action.
- *Indexical terms*¹⁴: CASSIE represents indexicals using the three variables:
 - **I**: refers to the speaker;
 - **YOU**: refers to the hearer;
 - **NOW**: refers to the current time.

¹⁴Indexical (or deictic) is called a word whose meaning depends on when, where, and who said the word. Anaphora resolution is different from the resolution of indexical terms because in case of anaphora the referred entity is in a previous or successive sentence. In other words, the meaning of an anaphoric expression doesn't depend on the time, space or speaker, but on another expression in the context of the discourse (see (Levinson, 2001)).

HERE is represented by the position of the speaker at the current time. Indexicals are resolved by the input grammar and re-inserted by the generation grammar. The knowledge base doesn't contain them.

- *Sense of time*: CASSIE keeps track of time using the variable NOW. Actions update this value at the PMLa level.
- *Low-level bodily awareness*: Shapiro uses this term to denote the consciousness the machine should have of its low-level functions to make the whole system more nearly like a real conscious entity. The current CASSIE doesn't have much low-level awareness; for example, high level functions that implement a durative action, such as going to a particular place, don't know about the time needed to realize them in the real world. Basically, as the author notes, CASSIE is missing feedback from the low-level layer to the KL layer.

As an example of a typical dialogue we reproduce one of the dialogues in (Shapiro, 1989). In this dialogue one can see some features of consciousness and the use of indexicals.

Stu:	Come here.
CASSIE:	I came to you.
CASSIE:	I am near you. ¹⁵
Stu:	Who am I?
CASSIE:	You are a person and your name is 'Stu'.
Stu:	Who have you talked to?
CASSIE:	I am talking to you.
Stu:	Talk to Bill. ¹⁶
CASSIE:	I am talking to you.

Continued on next page

Bill:	Come here.
CASSIE:	I found you.
CASSIE:	I came to you.
CASSIE:	I am near you.
Bill:	Who am I?
CASSIE:	You are a person and your name is 'Bill'.
Bill:	Who are you?
CASSIE:	I am the FEVAHR and my name is 'Cassie'.
Bill:	Who have you talked to?
CASSIE:	I talked to Stu and I am talking to you.

Summing up, CASSIE doesn't have any particular feature that distinguishes it from the other agents presented in this chapter (more on this in section 3.5 where all the agents will be analyzed and compared in more detail). We found the fact that CASSIE was built from a general purpose reasoning architecture without any particular modification to be the most interesting characteristic of this agent. This should be another indicator that consciousness doesn't seem to require extra modules or particular architectures, only extra knowledge and a knowledge representation language with enough expressivity to be able to represent at least beliefs¹⁷.

¹⁵This and the previous are reflexive assertions that are automatically generated by CASSIE to tell the action that it is doing.

¹⁶Canned sentence to tell to the machine to whom it is talking.

¹⁷This is only a necessary but not sufficient condition for having self-awareness.

3.4 COMA, Episodic Logic and EPILOG1

The objective of the whimsically named COMA (COncscious MACHine) project started by Lenhart Schubert in 1999 is to build a conscious agent using as background infrastructure the EPILOG question answering system (see (Schubert et al., 1993)).

The type of consciousness this agent should realize is explicit self-awareness as defined in section 2.3. We now describe to what extent the requirements listed in section 2.3 are realized in COMA.

Most of the requirements will be directly satisfied by some features of the representation language used in COMA, Episodic Logic (EL). Therefore the following paragraphs, in addition of serving the purpose of illustrating how COMA tackles some of the requirements of explicit self-awareness, are also a summary of some of the unique features of EL.

EL allows for the representation of events and their relations, including reference to complex events. For example, given the sentence (from a website) “The driver-side front tire blew out, causing the driver to lose control of the vehicle, resulting in accident/injury,” the representation language should be able to capture the causal connections between the events characterized by the first two clauses and the accident/injury event in the last clause.

In EL there are 3 operators to represent events and their relations: $*$, $**$ and $@$. All these operators take 2 arguments, a sentence (well-formed formula, also called wff) and an event term. In particular, for a sentence P and an event e , the meaning of these operators is as follows:

- $[P * e]$ means that P is true somewhere in e , partially describing e ;
- $[P ** e]$ means that e is characterized by P . $[[P ** e] \Rightarrow [P * e]]$.
- $[P @ e]$ is a short form of $(\exists e1 [e1 \text{ same-time } e] [P * e1])$.

Generic knowledge is supported in EL in several ways: the availability of generalized quantifiers, reification operators and the possibility of expressing probabilistic statements. Examples of generalized quantifiers are *most* and *many*. In addition there are operators to express kinds and make statements about them. For example $[(K \text{ (plur Dog)}) \text{ Friendly}]$ says that dogs as a kind are friendly, and would be equivalent to $(\text{Gen } x [x \text{ Dog}] [x \text{ Friendly}])$, in which “Gen” is a generic quantifier. A well-formed formula quantified by “Gen” is true when almost all instances of its restrictor satisfy its nuclear scope (the restrictor in the example above is $[x \text{ Dog}]$ and the nuclear scope is $[x \text{ Friendly}]$). Note that in EL predications are typically in infix form while functions and predicate modifications are in prefixed form. Infix form is often highlighted in the written version of EL by the use of square brackets.

Attitudes, autoepistemic inference, metasyntactic devices and knowledge categorization are supported in EL by the availability of substitutional quantifiers and quasi-quotation. Substitutional quantifiers allow quantification over entities other than terms without making the logic a second-order logic by quantifying only over the syntactic expressions of the appropriate type. Quasi-quotation allows for writing formulas that represent properties of the syntax itself. For example, the formula $['= \text{EL-Predicate}]$ expresses that ‘=’ is a predicate in EL.

The *quasi* in quasi-quotation comes from the fact that quotation is transparent to substitutional quantifiers. That means that variables substitutionally quantified will be substituted for even in quoted contexts. For example, the formula $(\forall t1 (\forall t2 (\forall_{pred} p ['p \text{ Commutative}] [[t1 \text{ } p \text{ } t2] \Leftrightarrow [t2 \text{ } p \text{ } t1]])))$ expresses that the order of the arguments for any binary commutative predicate is irrelevant. Note the use of the quoted metavariable (i.e. substitutionally quantified variable) p in $['p \text{ Commutative}]$.

Regarding autoepistemic inference and attitudes, EL provides operators to reify a sentence into a term so that it can be used in argument position. An

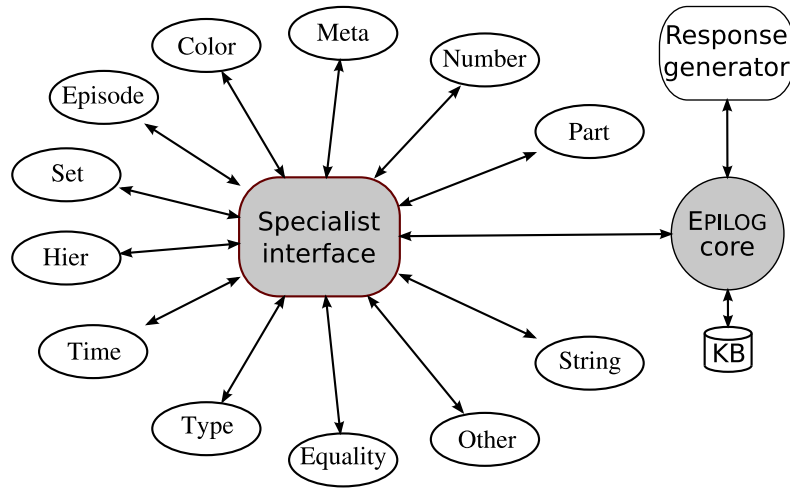


Figure 3.7: The high level structure of EPILOG1.

example of such an operator is “that”, for example, the formula

$$(\forall_{pred} p [p \text{ appearance-pred}] \\ (\forall x [x p] [(that [x p]) \text{ appearance-fact-about } x]))$$

states that the proposition (that [x p]) is a fact about the appearance of x if p is a predicate that describes the appearance of something.

Concerning the need for summarization abilities that describe complex events (especially its own experiences) at various levels of abstraction, we note in passing that there is no particular support for summarization in COMA. (Nor have we tackled this issue in our extensions.)

Regarding the need for large knowledge bases, the inference engine must be scalable in particular in the process of knowledge retrieval. That means that there should be methods to limit the amount of knowledge retrieved (on the basis of its relevance) at each step of the inference process. EPILOG provide an initial version of such a schema. However it still has problems as we will indicate towards the end of this section.

Another feature used both for autoepistemic reasoning and for increasing inference performance is the availability of a library of specialized inference engines.

Figure 3.7 shows the high-level structure of EPILOG. EPILOG's core contains the inference routines, the parser, the normalizer and the storage and access schemas to retrieve and add knowledge from/to the knowledge base. Around it there is a set of specialists, connected to the core inference engine through an interface module. These specialists help the general inference routines to carry out special inferences quickly (see (Miller and Schubert, 1988)) (e.g., type inference to conclude whether [Car1 Artifact] is true given that Car1 is a coupe and coupes are a type of car, cars are vehicles, and vehicles are artifacts). The specialist interface consists of a series of flags associated with some key predicates/functions that automatically activate predefined functions in a particular specialist.

For a more detailed introduction to EPILOG refer to (Schubert et al., 1993) and (Schaeffer et al., 1993).

After describing the features of COMA/EPILOG and before moving to a discussion of the problems of this system we will show it in action with a demo dialogue presented at the AAAI 2005 Spring Symposium. This demo is an extension of the demo realized by Aaron Kaplan in 2000.

The original demo was extended to include questions about

- the agent's self model (e.g., its name, its abilities, its interactions with the user, etc.);
- a simple way to demonstrate how the reasoning system could interact with a planner to put intentions into actions;
- interaction with specialists to get numeric answers to quantitative questions (e.g., about the cardinality of a set).

Three questions in this demo were present also in Kaplan's demo, they will be highlighted to distinguish them from the newer material.

The demo is reported as a list of assertions/questions followed by the system's answer and a discussion of it with in some cases the relevant knowledge used to obtain the answer.

```
Assertion: ((ME GREET YOU))
WFF180: (EPILOG-SYSTEM INTEND-TO (QQUOTE GREET) C146)
        Rank: 2
        Parents: (WFF141 WFF133 WFF174)
        Probability: 1
```

Here we asserted to the system that I greet EPILOG. ME and YOU are directly translated into the Skolem constant associated with the current user and with EPILOG-SYSTEM respectively (that is the internal name of EPILOG).

This is only an assertion so EPILOG doesn't produce any answer but it is interesting to note that it is possible to infer intentions. In this case we wanted to show a potential way to connect EPILOG with a planner, which would take care of the actual execution of intentions. In the inferred intention the constant C146 is the constant corresponding to ME. To make that inference the following knowledge was used:

```
(A g_pred (g %greeting-pred)
  (A u_term ((qq (u g epilog-system)) true)
    (epilog-system intend-to (qq g) u)))
```

A and E correspond respectively to \forall and \exists .

In the next exchange, we are asking the system for its name. WH indicates a question¹⁸ and the question is expressed as, what X is such that X is a name and you have X in some episode Y? An episode is an event, or a situation that persists for some time.

```
Question: (WH X (X NAME) (E Y ((YOU HAVE X) * Y)))
Answer(s):
          (EPILOG)    with probability 1
          (ECONET)    with probability 1
```

¹⁸Questions can be of two types: *yes/no* or *wh*. A yes/no question is answered with a yes or a no (the system can answer with UNKNOWN in case it doesn't find an answer). A wh question requires to find an answer that fills in the gap explicit in the question.

The system answers with two items, because during its development its name changed. The text `with probability 1` is the probability that the corresponding formula is true. EPILOG allows for probabilistic inference but the computed probabilities may not be accurate in all cases.

In the question we are asking the system for its name before 1990. EPILOG's answer is simply that name.

```
Question: (WH X (X NAME)
           (E Y (Y BEFORE (DATE 1990 1 1 0 0 0))
            ((YOU HAVE X) * Y)))
Answer(s):
           (ECONET)    with probability 1
```

Then we assert my name, and then ask, what my name is. EPILOG answers with the name just provided.

```
Assertion: (((FABRIZIO-MORBINI NAME-OF ME) ** SOME-TIME-INTERVAL))
Question: (WH X (X NAME) (E Y ((ME HAVE X) * Y)))
Answer(s):
           (FABRIZIO-MORBINI)    with probability 1
```

The next question, already present in the original demo, asks what EPILOG's abilities are. The answer lists the abilities stored in the knowledge base. The answer is not provided because it inferred this ability, but just because it was explicitly stated in its knowledge base.

```
Question: (WH X (YOU ABLE X))
Answer(s):
           ((KA
            (L VAR6
             (E VAR5 (VAR5 HUMAN)
              (E VAR3 (VAR3 TRANSPORTATION-PROBLEM)
               (VAR6 HELP VAR5
                (KA (L VAR4 (VAR4 SOLVE VAR3)))
                )))))) with probability 1
            ((KA (L VAR2 (E VAR1 (VAR1 HUMAN)
              (VAR2 CONVERSE-WITH VAR1))))))
            with probability 1
```

In the following we are asserting that a particular train, `TRAIN1`, is a train. Then we ask, what EPILOG knows about `TRAIN1`. Here we use the feature that the knowledge is indexed by concepts and topics, and we can extract the knowledge about a concept under a particular topic (here `NIL` indicates lack of topical

constraints). The answers are “*That train1 is a train*” and “*That you told me that train1 is a train*”.

SYSTEM-KNOWLEDGE-ABOUT is a special function that will be automatically evaluated to a particular Lisp function to extract the desired knowledge.

```
Assertion: ((TRAIN1 TRAIN))
Question: (WH X (X = (SYSTEM-KNOWLEDGE-ABOUT TRAIN1 NIL)))
Answer(s): ((($ 'SET (THAT (TRAIN1 TRAIN))
                (THAT
                 ((YOU TELL EPILOG-SYSTEM (THAT (TRAIN1 TRAIN)))
                  ** EP-C288)))) with probability 1
```

In the next question, already included in Kaplan’s demo, we ask whether EPILOG is human. EPILOG uses the knowledge in the type hierarchy and the fact that the system is a computer system to negatively answer the question.

```
Question: (YOU HUMAN)
Answer: NO with probability 1
```

Next we ask about EPILOG’s beliefs (also this question was already included in the original demo). This actually doesn’t use the belief specialist described in (Kaplan, 2000) because not all of the mechanisms for correct introspective inference are in place as yet. Currently the question is answered thanks to the explicit (but faulty) knowledge that everything that is provable is believed by the system.

```
Question: (YOU BELIEVE (THAT (NOT YOU HUMAN)))
Answer: YES with probability 1
```

The following example tests consciousness of past events, called autonotic consciousness (see description on page 57), by inquiring about the things we asked EPILOG. EPILOG answers with all the questions we have asked till now, in most-recent-first order (note that variables are automatically renamed; and ME and YOU are interchanged for print-out purposes).

```
Question: (WH A-THING
           (E AN-EPISODE_EP ((ME ASK EPILOG-SYSTEM A-THING)
                               ** AN-EPISODE)))
```

```

Answer(s):
  ((WH VAR18 (E VAR17 ((YOU ASK EPILOG-SYSTEM VAR18)
    ** VAR17)))) with probability 1
  ((EPILOG-SYSTEM BELIEVE (THAT (NOT EPILOG-SYSTEM
    HUMAN)))) with probability 1
  ((EPILOG-SYSTEM HUMAN)) with probability 1
  ((WH VAR16 (VAR16 EQUAL (SYSTEM-KNOWLEDGE-ABOUT TRAIN1
    NIL)))) with probability 1
  ((WH VAR15 (EPILOG-SYSTEM ABLE VAR15)))
    with probability 1
  ((WH VAR14 (VAR14 NAME) (E VAR13 ((YOU HAVE VAR14)
    * VAR13)))) with probability 1
  ((WH VAR12 (VAR12 NAME)
    (E VAR11 (VAR11 BEFORE (DATE 1990 1 1 0 0 0))
      ((EPILOG-SYSTEM HAVE VAR12) * VAR11))))
    with probability 1
  ((WH VAR10 (VAR10 NAME)
    (E VAR9 ((EPILOG-SYSTEM HAVE VAR10) * VAR9))))
    with probability 1

```

Next we ask, in an approximate way, when EPILOG's development started. More exactly we are asking, what D is the starting point of some event characterized by the being the event of EPILOG's development. The answer is the name of the variable that contains that value. This symbolic answer is unsatisfactory, illustrating the necessity of understanding what syntactic form the answer should take, as noted by Schubert in (Schubert, 2005).

```

Question: (WH D
  (E X
    ((D = (START-OF X)) AND
      ((EPILOG-SYSTEM-DEVELOPERS DEVELOP YOU) ** X))))
Answer(s): (EPILOG-SYSTEM-DEVELOPMENTSTART) with probability 1

```

With the following question we ask for the duration of the development episode. The answer in this case is roughly what we want but it is obtained not because the system realizes that the user would prefer a numerical answer to a symbolic one, but because of the particular implementation of the time specialist. Note that the answer is a list of two numbers (($\$$ NUM ...) is the particular syntax to express general data; we can define any data type using the ($\$$ type content) syntax): the first is the lower bound on the duration and the second the upper bound. Because we didn't assert anything about the end of EPILOG's development the upper bound is `+infinity`.

```

Question: (WH D
           (E X
            ((D = (DURATION-OF X)) AND
              ((EPILOG-SYSTEM-DEVELOPERS DEVELOP YOU) ** X))))
Answer(s): (($ NUM 480946905 +INFINITY))    with probability 1

```

Then we ask who are EPILOG’s developers. The answer is the set of the six persons involved at different times in EPILOG’s development.

```

Question: (WH X (E Y (E Z (((Y DEVELOP YOU) ** Z) AND
                           (X = (SET-MEMBERS Y))))))
Answer(s): (($ 'SET STEPHANIE-SCHAEFFER CHUNG-HEE-HWANG
                JOHN-DE-HAAN LENHART-K-SCHUBERT AARON-KAPLAN
                FABRIZIO-MORBINI))    with probability 1
                (EPILOG-SYSTEM-DEVELOPERS)    with probability 1

```

Last we ask what is the number of EPILOG’s developers. To answer this question we are using a function provided by the set specialist: `has-cardinality`. The problem of mapping from natural language to a representation that appropriately references functions that the specialists can evaluate is not handled at present.

```

Question: (WH X (E Y (E Z (((Y DEVELOP YOU) ** Z) AND
                           (Y HAS-CARDINALITY X))))
Answer(s): (6)    with probability 1

```

In summary, this extended demo shows some preliminary capabilities with respect to self-awareness, like access to self-model and episodic memory, in addition to more general features like temporal and specialist inference.

In the COMA effort and the successive work described in this thesis, we focus on question/answering systems avoiding planning not because it’s not important with respect to self-awareness but because both (planning and question/answering) require significant improvements before being usable for the construction of a fully utility-oriented continuously running explicit self-aware agent. Therefore in COMA and in this thesis we focus on improvements to the question/answering aspect of an explicit self-aware agent. Efforts to integrate reasoning with planning in a utility driven continual planning agent based on EL are

currently being made in a different project, see for example (Liu and Schubert, 2009).

So, architecturally there is nothing particular about COMA itself; the features it has are the features that EPILOG itself has. However, EPILOG provides a flexible and extensible architecture that we believe can be used to meet many of the requirements for explicit self-awareness. In particular, in this extended demo we stressed the interaction with the specialists and the topicalized access to memory. On the consciousness side EPILOG has the advantage of using Episodic Logic, which allows for probabilities and temporally related episodes, and this seems necessary for emulating the abilities of human episodic memory. Both these features are very important for pushing EPILOG's inferences to a more human level. But much work has to be done to improve COMA/EPILOG, we list here some of the more pressing problems we found:

- A fundamental problem lies in the low level structure of EPILOG. It was not designed with modularity and evolution in mind, therefore any change to it is very hard and would likely make it even more convoluted. There is a widespread use of global variables as a mean to communicate information across functions in different modules. The code lacks documentation and the functions are not written in a way to make the job of a programmer easier. The extensive documentation in the form of manuals that comes with the code is in several places out of date.
- Related to the previous point is a general lack of structure of the inference module. We detected bugs both in the unification module (e.g., lack of even basic occurs check) and in the techniques used to approach some inferences, unfortunately the code in its current form is very hard to fix.
- Parsing of EL formulas is accomplished with a combination of if-then rules and global variables that make every modification (e.g., extensions to how

substitutional quantifiers and quasi-quotation are currently handled to be able to realize some of the features of explicit self-awareness) to the grammar very cumbersome and prone to errors.

- The interface between EPILOG and its specialists needs revision to allow a higher level of transparency as dictated by explicit self-awareness. In particular, how can we make EPILOG aware of the capabilities of its specialists? For example, to answer the question: “*How many developers developed you?*” we added specific knowledge to access the right function of the set specialist that computes the cardinality of a set. But what we would like is that EPILOG infers that it has to use that particular function of that particular specialist because that is what solves its goal of answering the user question. The approach suggested in (Schubert, 2005) involves the use of the meta-syntax already illustrated in section 2.3, to codify knowledge about what conclusions can be drawn from the values produced by certain executable functions. (In the example in (Schubert, 2005) the function was ‘*add*’ but it could equally well have been ‘*find-cardinality*’ or ‘*find-super-concepts*’.)

A similar problem concerns the type hierarchy, of which EPILOG has no direct knowledge. In fact, even if the type hierarchy contains the knowledge that a flower is a plant, EPILOG will not be able to answer a general question like, “*What do you know about flowers?*” with, “*Flowers are plants*”.

An additional concern of both type hierarchies and specialist interface is the manual work they require to be setup. In fact neither of them are automatically constructed from explicit knowledge stored in the main knowledge base. This is also one of the reasons for the lack of transparency.

- Probabilistic inference is unreliable in some cases and will need revision.
- The mechanism to handle the storage of temporary knowledge, used in particular during assumption making, can be heavy and therefore inappropriate

for a module used often during inference.

- Introspection, that is at the base of any self-aware agent, is currently partially handled using a specialist written by Aaron Kaplan. The author of this specialist had been forced to select this suboptimal strategy because of the intricacy of the question answering (QA) framework heavily based on global variables that didn't allow for additional questions to be asked within another question (we will refer to this in the coming sections as recursive QA).
- The efficient retrieval of knowledge is based on the presence of a type hierarchy and is currently geared toward effective forward inference more than toward goal chaining. Therefore when approaching a new problem, time has to be allocated to building the appropriate type hierarchy and in some cases a few inferences necessary to solve a particular question may not be executed.
- The topical organization of the memory sometime fails because of the particular form of the input (for example in the case of certain disjunctions) or because the user has forgotten to specify the types of the variables in the formula. The variable types are important for topical access to the memory because they are used to decide under which access keys a formula will be indexed. Currently EPILOG1 doesn't automatically assign a root-entity-like type to untyped variables.
- Another problem that can explain some of the previous points is the lack of a extensive set of problems to be used both to periodically check the correctness of the routines and to provide a performance measure to be used in comparison with future revisions of EPILOG and with other theorem provers.

3.5 Comparative Evaluation

In this section we will compare the four systems just described. After comparing their characteristics we will use the criterion of explicit self-awareness defined in section 2.3 to classify them.

SHRDLU and Homer have in common their planning capability. Homer's planner is also able to re-plan whenever it realizes that a previously computed plan fails because, for example, the world changed from what the agent knew. The re-planning capability is fundamental for acting in a dynamic environment. (One of the simplifying assumptions in SHRDLU was that it had complete knowledge of the state of the world and that SHRDLU was the only actor in that world.) CASSIE has a basic ability to plan but we don't have information on how scalable or effective it is. Currently, COMA has no planning capability but its ability to express goals and actions sets the stage for planning. Adding a planning capability as a specialist or as an external system or directly as part of the inference engine is in our plans for future extensions that will be merged in the EPI2ME agent (see chapter 4).

Homer and COMA have an explicit ability to represent episodes and their time relations. SHRDLU instead has a very limited ability both on the representation side (i.e., in representing time relations) and on the access side (i.e., the accessibility of episodic knowledge to reasoning). Also CASSIE's ability to handle time relations is limited; currently CASSIE doesn't handle any reference to a future time. Episodic memory as pointed out in section 3.2 is an important part of our memory system and it is intimately related to consciousness.

SHRDLU is the only system able to explain its own actions. This ability, as said, is more a product of an ad-hoc mechanism than a product of understanding and reasoning. In Homer the capability to explain its own actions is limited by the implementation choice of keeping the planner's episodic memory separated from

the main memory. Because of this choice, it is impossible to ask questions about the internal behavior of the planner (e.g., “*Why did you do this action before that one?*”), but it is possible to ask questions regarding the reasons for an action because actions (and their relations) are accessible to the reasoner. In COMA this capability has not been implemented and would require reason-maintenance mechanisms in addition to a planning capability. Also CASSIE doesn’t seem to be able to explain its own actions.

COMA lacks the ability to directly accept English as input, while the other three have fairly good parsers and semantic interpreters but both defined only for a limited subset of English. Homer’s authors discuss the difficulties encountered in extending Homer’s lexicon and semantic interpreter. Winograd doesn’t address this topic. The close connection between episodic logic and language promises to facilitate building this translation process for COMA.

Only COMA allows for probabilistic inferences. However, there are cases in which the computation of these probabilities is not accurate.

Using the classification of self-aware systems given by Schubert in (Schubert, 2005) the four systems presented here could be classified as follows.

- SHRDLU: a goal oriented system with self-explanation capability;
- Homer: the nearest category is the goal-oriented one, even though Homer also has features (like English communication, episodic memory, reflexive processes) that characterize explicit self-awareness;
- CASSIE: it doesn’t belong to any of the listed categories but it has features of various of those classes, like: planning, communication in English, basic reflexive processes, inference machinery. However, most of them are at a basic level.

	SHRDLU	Homer	CASSIE	COMA
Domain	Blocks World (single agent)	Submarine (multi-agent)	Not limited	Not limited
Representation language	Procedures (PLANNER)	Frame-like	Semantic nets	EL
English I/O	Yes	Yes	Yes	Preliminary
Planning	Yes	Yes (re-planning)	Yes (basic)	No
Special reasoning	Time/Geometric	Time/Geometric	Time	Yes
Reflexive processes	No	Repetition Perception	Perception	No
Action explanation	Yes	Partial	No	No
Others	Naming of set of objects			Probabilistic reasoning

Table 3.6: Summary of the distinguishing features of the 4 systems compared in this chapter. The upper part of the table lists features generally connected with reasoning and planning agents. The bottom part instead has features more directly related to self-awareness. The green color indicates that the way in which a particular aspect has been realized by a particular system is well suited for the development of a self-aware agent. The orange color instead indicates that the particular aspect is lacking some characteristic necessary for self-aware agents.

- COMA: currently it doesn't belong to any of the given classes. Since it doesn't plan, it is not fully goal-oriented (even if its answering ability could be considered as partially goal oriented) and it is not able to explain its answers.¹⁹ But it has certain features (like inference infrastructure and episodic memory) necessary for explicit self-awareness, which we set out to exploit in building COMA extensions.

Considering the knowledge requirements listed in (Schubert, 2005) and (McCarthy, 1995), we make the following observations.

- Logic: All four systems are based on logical representation languages.
- Events: Homer and COMA, as already noted, have good support for expressing time relations and episodic memory.
- Attitudes and autoepistemic reasoning: SHRDLU does not explicitly reason about beliefs, perceptions, etc., but it has thorough knowledge of its closed world. Homer currently shows basic abilities in assessing what it knows. COMA includes a belief specialist based on (Kaplan, 2000) but it is not fully functional. For example, there are problems with positive and negative introspection. We don't have any basis for assessing the ability of CASSIE to reason about what it knows, but in principle it should be able to do it, at least at a basic level.
- Generic knowledge: all four systems have very limited knowledge bases highly optimized for the task/demo for which they were built. But COMA is certainly the one that makes the job of adding generic knowledge easiest because of the use of Episodic Logic as representation language.
- Metasyntactic devices: None of the systems is currently able to reason about the syntax an answer should have, although EL does have a syntax for

¹⁹It does, however, record what premises led to the conclusions it reaches.

quotation and substitutional quantification that is used to encode axiom schemas.

- Knowledge categorization: among the four, only COMA has topicalized access to its knowledge.
- Summarization: only Homer includes a primitive capacity for summarization as one of its reflexive processes.
- Actions explanation: only SHRDLU can explain its actions, but only because of an ad-hoc process and not because of its general inference abilities.
- Utility function: none has a utility-driven behavior.
- Knowledge of capabilities and limitations: all four systems are very poor in this respect. In CASSIE this feature has not been investigated. The other three have capabilities of which their reasoning system is unaware: SHRDLU and Homer don't know about their planning capabilities and COMA is not explicitly aware of its specialists.
- System transparency: CASSIE seems to be very transparent: one reason could be that currently it is simpler than the other three systems. Each of the other three systems performs steps that are not accessible to reasoning: the planner's episodic memory in Homer, SHRDLU's action representation and COMA's specialists are a few examples.

Table 3.6 is a summary of the comparison described in this section. A green color means that the feature is considered a step in the direction of explicit self-awareness. Orange means that significant extensions/changes must be done.

4 Epi2Me

As said in section 3.4, COMA was the name used to identify the first application of the EPILOG system to study the requirements imposed on inference and representation by explicit self-awareness. Epi2Me is the name we use for the partially developed self-aware reasoner based on the newly redesigned version of EPILOG. The initial work on extending the original COMA demo underscored the need for improvements and extensions to EPILOG (especially in goal-directed inference), so that the goal of extending the COMA demo into a more articulate demonstration of explicit self-awareness evolved into the goal of improving, and extending the foundation of the EPILOG system, in particular regarding (see section 3.4 for details):

- the usability of the system, in particular in terms of minimizing the setup required to be able to solve a given problem (setup includes things like, adjusting global parameters, preparing specialist knowledge such as type hierarchies, and marking of predicates and functions to connect them with specialists); usability also refers to supporting features for the developer such as visual display of the inference graph;
- the extensibility of the system, both in term of easing the addition of new functionalities and supporting modifications in those modules prone

to changes, such as the EL grammar, the KB access schemas and the normalization procedure;

- the transparency of both the knowledge embedded in the specialists and the method used to access them by the general inference engine;
- remedying the lack of a methodology for tracking the performance and correctness of the system while under active development. This is helpful both to ensure that an extension of a functionality doesn't compromise other systems and to provide a performance comparison with future versions and also with other inference systems.

In general, we will refer to the newer version of the EPILOG system as EPILOG. In those cases where a distinction with the old version is necessary for clarity, we will use EPILOG1 for the old version and EPILOG2 for the newer one.

Figure 4.1 shows the main modules found in EPILOG2 divided into 2 categories: the functions commonly used by the user and the library functions required to execute the user level functionalities. The modules displayed here correspond roughly to the grayed boxes of figure 3.7.

In the remainder of this chapter we will describe each of the modules in detail.

4.1 Parser

The EL parser in EPILOG is now based on a standard chart parser with a few optimizations to augment its performance:

- Arcs are added to the chart only if they can extend some of the uncompleted edges already present in it.
- Arcs are added only if they pass a semantic test associated with the correspondent grammar rule.

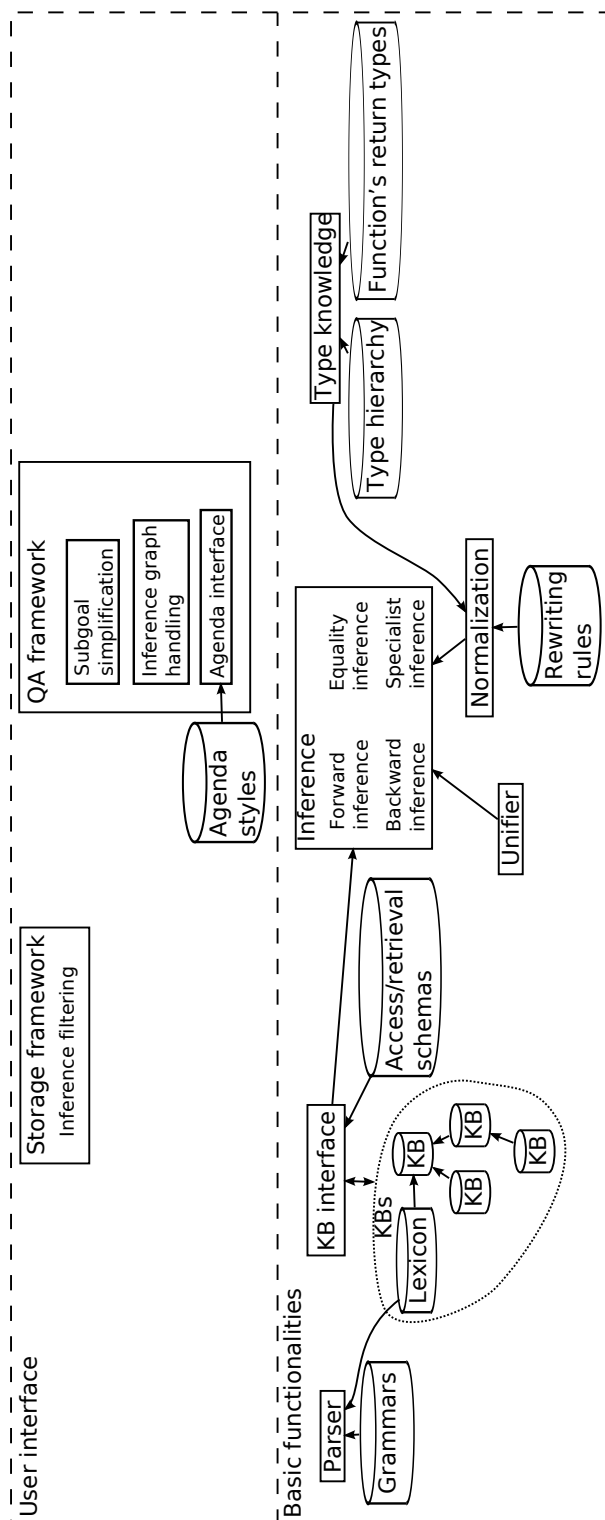


Figure 4.1: The high level structure of EPILOG2.

- In the case of arcs corresponding to variables, these must already have been introduced by a quantifier or lambda. Dynamically bound variables are handled by this optimization, but currently the rest of the system does not support them.
- A token can instantiate multiple rules (e.g., a token could be a predicate or a function). To lower the number of arcs in the chart, the parser accesses the knowledge base looking for knowledge about the current token to determine if something is known about its EL type. For example, the knowledge [*Wolf* EL-Predicate] will be used by the parser to activate only the grammar rule that forms a predicate from the token “Wolf”.

An input formula can sometime produce multiple parses because of ambiguities in the EL grammar. One source of ambiguity in the EL grammar is the support for both nested (curried) predication, e.g., $((Pred1\ Arg1)\ Arg2)\ Arg3$ and flat, infix predication, e.g., $(Arg3\ Pred1\ Arg1\ Arg2)$ (written $[Arg3\ Pred1\ Arg1\ Arg2]$ for readability). Currently these ambiguities are resolved by detecting the ambiguous tokens in the set of parses and asking the user to disambiguate them.

Our parsing algorithm can be used with multiple grammars. The grammar used to parse a given input can be changed at run-time on the fly. Currently two grammars are defined: **1)** the EL grammar and **2)** a first order logic (FOL) grammar. The FOL grammar is used to parse the problems contained in the TPTP¹ dataset. Even though EL is a superset of FOL, the FOL grammar is useful because it is not ambiguous and therefore allows for quicker parsing.

The structure produce by the parser for each formula is a tree. For example, the formula:

$(\forall x_0$

¹See <http://www.cs.miami.edu/~tptp/>

$$(\forall x_1 (\forall x_2 [[x_0 ((\text{to-degree } x_1) \text{ old})] ** x_2] (\forall x_3 [x_3 \text{ during } x_2] [[x_0 ((\text{to-degree } x_1) \text{ old})] ** x_3]]))))$$

will generate the tree represented in figure 4.2.

4.2 Normalization

After a formula has been parsed it will usually be normalized. Normalization is crucial for detecting equivalent formulas efficiently. However, the presence of generalized quantifiers, modifiers, reifiers and episodic operators make it impossible to convert an EL formula to the classic conjunction of disjunctions often used in FOL (also, even if possible, it may be undesirable to convert a formula to a conjunction of disjunctions because it's known that a formula can be exponentially larger when converted to that format).

The normalizer in EPILOG2 is based on a rewrite engine and on a set of rewrite rules (currently 23), including operations like:

- Simplification of ground type formulas.
- Simplification of ground formulas concerning transitive predicates.
- Conversion of the quantifier “No” into the universal quantifier “All”.
- Removal of restrictors from universally and existentially quantified formulas.
- Moving negation inward.
- Removing the modifier “Be”.
- Removal of trivial tautologies, simplification of trivial equalities and trivial formulas (e.g., the true/false formula).
- Conversion of curried form to infix form.

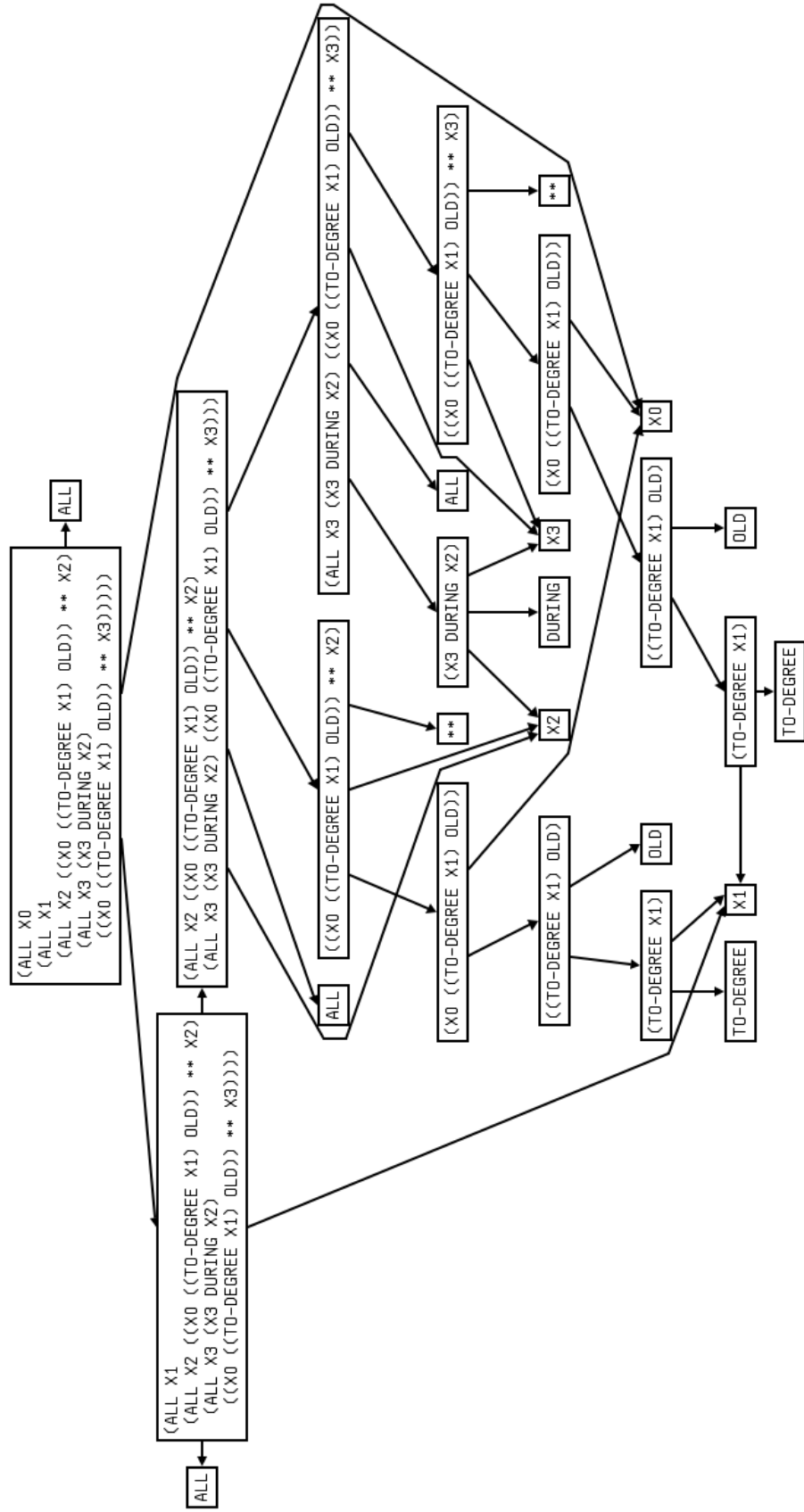


Figure 4.2: Tree representation of an EL formula. Note the sharing of variable nodes to facilitate variable substitution.

- Removal of trivial lambda abstractions.
- Clustering conjunctions and disjunctions.
- Moving quantifiers inward.
- Skolemization.
- Extraction of atemporal formulas from the scope of episodic operators.
- Breaking a characterization of a conjunction into a conjunction of characterizations (see (Schubert, 2000)).
- Lambda-conversion, e.g., converting $[\text{Arg1 } (\text{L x } [[\text{x Pred1}] \wedge [\text{x Pred2}]])]$ into the reduced form $[[\text{Arg1 Pred1}] \wedge [\text{Arg1 Pred2}]]$.
- Clustering of embedded conjunctions and/or disjunctions.
- Ordering of the arguments of commutative operations (e.g. conjunctions).

Note that the rules available to the normalizer depend on where normalization is called. For example, if the normalizer is called on a formula to be stored then Skolemization is used, whereas Skolemization is not used for goal formulas produced in question-answering by using the backward chaining inference rule.

4.3 Interface to Knowledge Bases

The way in which knowledge is accessed and stored is fundamental for the performance of any theorem prover, in particular as the amount of the knowledge stored increases to the quantity required for human-like intelligence.

We designed the new interface to knowledge to facilitate the following operations:

- Addition and retraction of temporary knowledge to/from the knowledge base. The addition of temporary knowledge is used frequently by the question/answering process, in particular by the assumption making feature used to approach goals with certain particular forms (see section 4.9).
- Development and testing of new schemas of knowledge indexing and retrieval (these schemas will be referred to as access schemas).

Concerning the first point (i.e. facilitating the management of temporary modifications to the knowledge base), our solution is to arrange the knowledge into various KBs organized in a hierarchy (similarly to what Cyc does with its hierarchy of micro-theories). When a temporary addition to a KB is required, an empty new KB is created and the additions (and, if forward inference is allowed, the consequences of those additions) are added to the new KB. If those changes must be retracted (the most common case) it suffices to discard the newly created KB. The hierarchical organization of KBs, in addition to supporting assumption-making, encourages systematic development of knowledge.

Regarding the second point above (i.e. facilitating the development and testing of new access schemas), we decided to build a uniform interface to KBs that defines the possible operations on a KB. Each KB will be associated with a particular access schema that implements the operations defined in the interface in a specific way. This framework allows each KB to have a different access schema that can be optimized to the type of knowledge contained in it; in addition it forces a modular interface to the development of new access schemas.

Currently two schemas are defined:

- a lookup schema that supports storage to a KB and lookup of each formula stored in it. This schema only allows for retrieval of the full formula and therefore is not suited for inference (because inference requires the ability

to retrieve a formula based on subformulas, called keys). This schema is currently used for the KB that contains the lexical knowledge;

- a simple exhaustive schema for testing inference. Storage of a formula w consists of first extracting the keys of w (the keys of a given formula are the smallest extensionally² embedded wffs contained in the given formula), and then for each key k of w , storing the pair $\langle w, k \rangle$ in the KB indexed by k . To support formula lookup, this schema also contains the previous schema: storing the formula w indexed by a unique id that characterizes the formula (see 4.3.1 for an explanation of how this id is obtained for a given formula).

Retrieval for the purpose of doing inference with a formula w consists of similar steps except that in the case of inference the input is not just a formula w but also a particular key, k , derived from it. Next, we try to match (unify) k with each key used to index the formulas in the KB, producing a new inference every time the unification step succeeds.

This is a trivial access schema and was designed solely for the purpose of allowing testing of the system while a more efficient schema based on a variant of the ideas used in EPILOG1 is in the making.

4.3.1 Long-term storage

To reduce the space requirement for stored formulas, after being normalized, formulas are converted to a form more suitable for long term storage (in short LTS form). The LTS form is characterized by:

- sharing all common subparts of previously stored formulas;
- storage of formula-specific information (like variable dependencies and variable bindings) in the root node of each formula.

²Extensional contexts are induced by all but the following elements of EL: episodic operators, modifiers and reifiers.

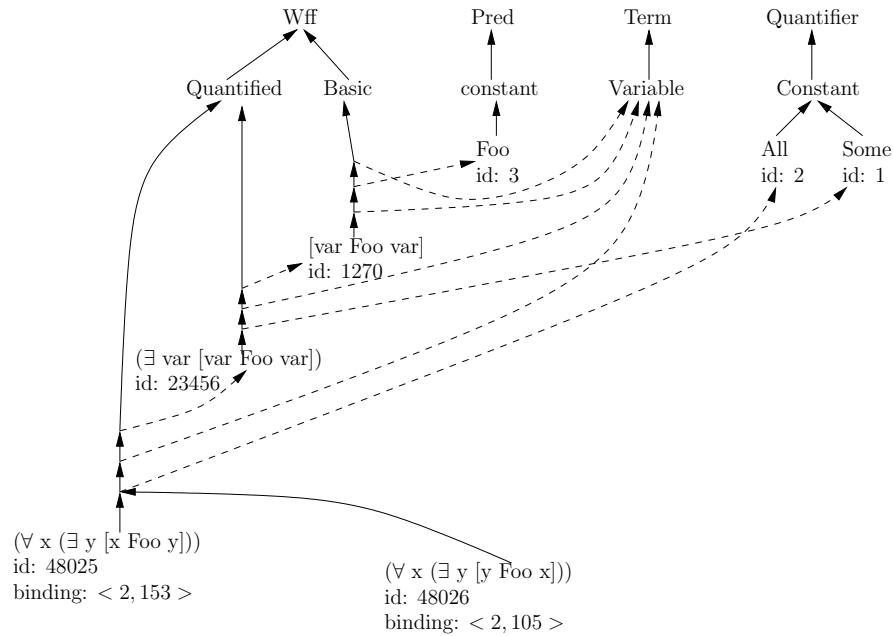


Figure 4.3: State of long-term storage after the formulas $(\forall x (\exists y [x \text{ Foo } y]))$ and $(\forall x (\exists y [y \text{ Foo } x]))$ have been stored. Each node has an id associated with it. The formulas displayed in the graph (e.g., $[var \text{ Foo } var]$) are not actually stored, they are reported here just for convenience). To reconstruct a stored formula, start from the corresponding leaf and follow the solid line after having traversed the dashed line (that points to the content of non-leaf nodes).

That means that the formulas $(\forall x (\exists y [x \text{ Foo } y]))$ and $(\forall x (\exists y [y \text{ Foo } x]))$ will share all nodes except the root in which different binding informations will be stored. Figure 4.3 show the LTS structures created after storing those two formulas.

Binding information is represented with two integers. The first specifies how many variables are used in the formula. The second is an integer that defines to which quantifier each variable is bound. The quantifiers are mapped to positive integers so that the number 1 indicates the first quantifier and so on. For example, the formula above: $(\forall x (\exists y [x \text{ Foo } y]))$ will have binding information represented by the pair $\langle 2, 153 \rangle$ because it has 2 variables, and 153, when read in portions of $\text{floor}(\log_2(2) + 1)$ bits each, becomes 4 numbers: 1, 2, 1 and 2, representing the

binding of the variables in the formula when its LTS tree is traversed depth-first.

A unique id is associated to each input formula during the conversion to LTS form. In figure 4.3 one can see such ids associated to the 2 formulas used in that example. These ids are used to recognize if two input formulas are the same (i.e. if they map to the same LTS form). Some logically equivalent formulas may still receive different ids if they are normalized to different forms.

4.4 Unifier

The unifier in EPILOG2 is basically a tree matcher that tries to unify two input subtrees, each representing a formula (or better a key of a formula) as produced by the parser module.

The 2 input trees are traversed in parallel in a depth-first fashion. While the 2 nodes (one coming from the first tree, the other coming from the second tree) can be unified, the traversal continues and the unifier is appropriately updated. A pair of nodes can fail unification because:

- they are incompatible (e.g., a predicate node with a function node, or two different constants).
- the occurs-check fails;
- the unification leads to free variables. Free variables can result when there is a use of a variable outside its definition (i.e. a quantifier or a lambda expression). This is not a hard failure because in some cases it can be recovered, for example, by a subsequent substitution that replaces the variable that otherwise would have become a free variable with a constant. Therefore, even if this check fails, the unifier will keep trying to unify the rest of the trees, and it will repeat the check for free variables before returning the complete unifier.

Substitutions are not done during unification but are left implicit. To make this possible, during traversal, if a node to be unified is a variable node, its most current substitution is used in place of the variable itself. Variables are mapped to positive integers to make this indexing (i.e. from a variable to its most recent substitution) more efficient.

Given the presence of substitutional quantifiers, the unifier deals with variables that can match not only terms but also wffs, predicates, functions, etc. This adds small complications. In particular, in the case of variables that match predicates, it is sometimes necessary to convert the other formula to be matched to curried form. For example, if we have the formulas $(\forall_{pred} p [c1 p c4])$ and $[c1 Pred c2 c3 c4]$ (both in infix form with the subject of the predicates in first position), we can match $[c1 p c4]$ with $[c1 Pred c2 c3 c4]$ by replacing the variable p with the partial application of $Pred$ to the arguments $c2$ and $c3$.

Another extension to the unification algorithm allows it to find the biggest matchable portions of the two given formulas starting from their respective keys. Basically unification first tries to unify the two keys, then it progressively moves up the ancestry of each key to find the biggest matchable subtrees. This is another way to reduce the chance of forming free variables. In addition, it may cut inference steps by matching multiple keys in a single step.

4.5 Specialist interface

As mentioned earlier, EPILOG1 required that predefined flags be attached to those predicates and functions that are connected to specialists. Specialists are modules that carry out inferences on a restricted domain much faster than if the same inferences were carried out by the general inference engine. See figure 3.7 for a representation and listing of the specialists present in EPILOG1.

This type of interface poses some problems with respect to transparency. One

aspect, from the point of view of the programmer/user, is that it's hard to follow how and when a particular specialist is called. However, the transparency problem is particularly evident with respect to making the agent self-aware. In fact, the inference engine has no control over which specialist to call or when to call it, as specialist execution is automatic, “unconscious”.

In EPILOG2, we redesigned the interface to specialists with the goal of making it more transparent. The basic idea was introduced in (Schubert, 2005) and is based on the use of a single special function, called *Apply*, known to the inference engine as having a special meaning. If a key contains the *Apply* function, the inference engine evaluates it. The evaluation consists in executing the first argument of the *Apply* function, which must be a valid Lisp function, applied to the remainder of the arguments.

To make the inference engine “aware” of a specialist functionality, represented by a particular Lisp function, it's sufficient to assert “attachment” axioms that specify when it is appropriate to use that specialist function. This knowledge will be used, if required, by the inference engine, like any other knowledge, to solve the given questions.

For example, the following is the attachment axiom that connects the introspection specialist with the general inference engine:

$$\begin{aligned}
 &(\forall_{wff} w \text{ [}'w \text{ without-free-vars]}) \\
 &\quad [[(Apply \text{ 'apply-fn-knownbyme? 'w}) = \text{'yes}] \Rightarrow \\
 &\quad \text{[Epi2Me know (that w)}}]]
 \end{aligned}$$

Note the condition [’w without-free-vars] that specifies when the specialist can be called. Basically, the formula says that to evaluate whether Epi2Me knows *w*, if *w* contains no free variables, the Lisp function *apply-fn-knownbyme?* should be used on the argument *w*.

Currently we are not doing any extra processing of the attachment axioms to make the call to specialists more efficient. One possibility for increasing the efficiency without decreasing transparency is to automatically attach special information to these axioms, so as to limit the keys usable for goal chaining to those in the consequent of the implication³ (e.g., [Epi2Me know (that w)]) and to explicitly indicate the preconditions for the execution of the specialist (e.g., [w without-free-vars]).

4.6 Inference

The inference module implements the rules of inference designed for EL and described in (Schubert and Hwang, 2000). We made the following extensions to the rules given in that paper:

- If a formula contains equivalences, before it can be used in inference the equivalences are oriented (i.e. transformed into normal implications) according to the polarity of the key to be matched in the other formula. For example, given the two formulas $(\forall x [[x \text{ Foo}] \Leftrightarrow [x \text{ Bar}]])$ and $[C1 \text{ Bar}]$ to be used for goal chaining, the first formula would be changed into $(\forall x [[x \text{ Foo}] \Rightarrow [x \text{ Bar}]])$ to generate the new goal $[C1 \text{ Foo}]$. If instead, the second formula is $(\text{not } [C1 \text{ Bar}])$ the first would be changed into $(\forall x [[x \text{ Bar}] \Rightarrow [x \text{ Foo}]])$ to generate the goal $(\text{not } [C1 \text{ Foo}])$.
- The soundness conditions originally given were sufficient conditions to guarantee that the result didn't contain any free variable. However they are not necessary conditions. Now the inference will be avoided only if it actually produces free variables.

³For those axioms that have a form similar to the one in the given example. Currently, all interface axioms have this form. But there are no imposed constraints on the form of these axioms.

In addition to forward and backward inference this module also provides two special inference methods, one for equality and one to support the specialist interface based on the *Apply* function. If the question-answering procedure (described later in section 4.9) selects a key that is an equality predication (i.e. a formula of type $[t1 = t2]$ where $t1$ and $t2$ are two terms) as the next key to be used for inference for a particular goal formula, then the equality inference is attempted. It first unifies the two arguments of the equality predicate. If the unification succeeds, then the key is replaced with the trivially true key and the unification is applied to the whole goal formula. If the selected key contains the function *Apply*, then specialist inference is used. In that case, the function is evaluated as described in section 4.5.

4.7 Type extraction

Type information is important in speeding up inference, for example by:

- allowing direct unification of type-compatible predicates. For example, if $[Fido\ Dog]$ is asserted and if it is known that $(\forall x [x\ Mammal] [x\ Furry])$ then we could unify $[Fido\ Dog]$ with $[x\ Mammal]$ to conclude that $[Fido\ Furry]$. Also in goal chaining one can use a similar procedure except that a predicate can be matched to its sub-types (rather than its super-types as in the forward inference case above);
- filtering out unifications that unify type-incompatible terms;
- computing the truth value of type keys, e.g., a key $[Fido\ Cat]$ could be immediately simplified to *False* if it is known that Fido is a dog and that dogs are disjoint from cats.

All of these shortcuts require multiple inference steps in a theorem prover that doesn't use type information.

Types already played a special role in EPILOG1, but the type hierarchy had to be built manually. In the current version we try to build this hierarchy automatically by using the stored world knowledge. Currently we handle 2 cases:

Type subsumption/disjointness: Detect all formulas that imply a type relation between two predicates (e.g., one is a super-type of the other) or between a constant and a type predicate (e.g., [Fido Dog]). The formulas that can currently be used by this step are of the form:

- $(\forall x [[x \text{ Type1}] \Rightarrow [x \text{ Type2}]])$: this adds a link between Type1 and Type2.
- $(\forall x [[x \text{ Type1}] \Rightarrow (\text{not } [x \text{ Type2}])])$: this adds a disjointness link between Type1 and Type2.
- $(\forall x [[x \text{ Type1}] \Leftrightarrow [x \text{ Type2}]])$: this merges the two nodes in the hierarchy corresponding to Type1 and to Type2.
- $[\text{Constant1 Type1}]$: this adds a link between an instance, Constant1, and its type, Type1.

The hierarchy supports multiple inheritance (implying that the type hierarchy is actually a graph rather than a tree) and the representation of disjoint types. Also, consistency checking is done when adding an edge. In addition, each edge is linked to the formula used to build that edge, so that each relation can be justified with the formula used to create it. The meaning of an edge is represented by a number from -1 to 1 where 1 represents perfect subsumption and -1 represent perfect disjointness. Even though currently not used, this hierarchy would support partial inclusion as indicated, for example, by a formula like $(\text{Most } x [[x \text{ Type1}] \Rightarrow [x \text{ Type2}]])$.

Checking of subtype and disjointness relations is currently done by graph traversal (where disjointness is checked by looking for disjoint ancestors). EPILOG1 used a more efficient scheme based on numeric preorder labeling, but it is

unclear whether the gains offset the programming complexity and potential need for occasional renumbering.

Function typing: Detect and store information about the return type of a function based on axioms supplying such type information. From formulas matching the schema $(\forall \text{Arg}_1 [\text{Arg}_1 \text{Type}_1] \dots (\forall \text{Arg}_n [\text{Arg}_n \text{Type}_n] [(\text{Func1 } \text{Arg}_1 \dots \text{Arg}_n) \text{Type}_{n+1}]) \dots)$, it is inferred that for arguments of type $\text{Type}_1 \dots \text{Type}_n$ the return type of the function Func1 is Type_{n+1} . The known argument type signatures and corresponding return types are stored for each function. This stored information is used to determine the possible return types of given functional expressions, by checking whether the given argument types are subsumed by one of the known function's type signatures.

Also in this case a consistency check is needed to ensure that no contradictory information is stored (e.g., given a function F that takes one argument, if it's known that when the argument is of type Type1 the return type is of type Type2 then a contradiction is present if we try to store a formula that says that the return type is disjoint from Type2 when the argument is a sub-type of Type1).

Given a type signature (i.e. the type of the arguments) of a given function, the following procedure is executed to find what type(s) are allowed for the return value of the function: for each argument type, its ancestors in the hierarchy graph are traversed and all traversed nodes are marked. After this step is finished for all arguments, the return types for the input function that have been marked in all the argument positions are returned as possible types of the return value of the given function with the given inputs. For example, if it is known that the return type of $(\text{FatherOf } x)$ when x is a Mammal is also a Mammal , if we query about the return type of the function FatherOf when the argument is of type Dog , Mammal will be returned because it is an ancestor of Dog . This operation can be carried out in time proportional to $n \cdot D \cdot tf$ where n is the number of arguments, D is the depth of the type hierarchy and tf is the number of type signatures stored for

the function we used in the query.

Since type information is derived from a particular knowledge base, and since our reasoning system makes use of a hierarchy of knowledge bases, type information should be inherited downwards in the knowledge hierarchy just like other knowledge. However, currently inheritance of type hierarchies is not supported and only one type hierarchy is created.

Note that for this module to create only type hierarchies, it should be able to identify type predicates. This is essentially a matter of identifying predicates corresponding to English nominals, which can be done with a lexicon, or by attaching special suffixes to predicates, e.g., *dog.n*.

The machinery just illustrated to build optimized data structures from knowledge subsets with particular structural characteristics can be used for purposes other than extracting type information. Currently we also use it to build hierarchies of the arguments of transitive predicates so that we can speed up inferences that would normally require several steps or the calculation of transitive closure on the knowledge base. The first step is to identify transitive predicates. That is achieved by detecting formulas of form $(\forall x (\forall y (\forall z [[[x P y] \wedge [y P z]] \Rightarrow [x P z]])))$ (and slight variants of it, e.g., one in which the two conjuncts are inverted). As soon as a formula like the one above is entered in some knowledge base, the content of the knowledge base is re-scanned looking for ground instances of the predicate P to populate the transitive hierarchy for P . In addition a matcher will be created to match any ground instance of the predicate P so that future asserted knowledge will be used to update the transitive hierarchy.

The information stored in the transitive hierarchy is used during normalization to simplify keys involving transitive predicates in the same way as the type hierarchy is used to simplify type keys (to truth or falsity). To exemplify this simplification procedure and its impact on the speed with which a solution can be found, let's use one of the problems used in the evaluation of the system (see

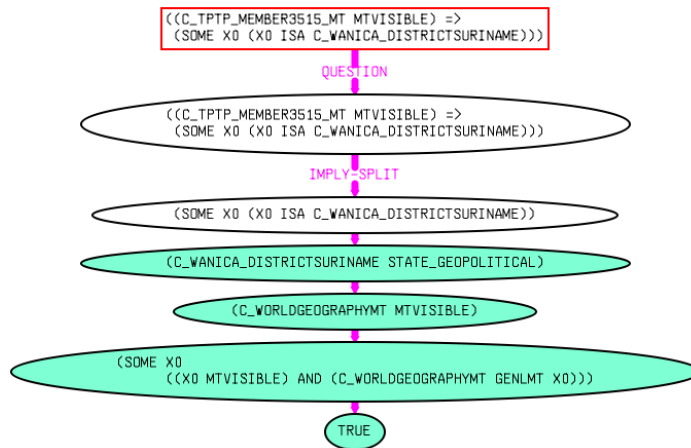


Figure 4.4: Proof found for the problem CSR034 when transitive predicates are used to build hierarchies of their arguments.

section 5): CSR034. Figure 4.4 displays the inference graph of the proof found for the problem CSR034 when hierarchies of the arguments of transitive predicates are constructed and used. In particular, the hierarchy used by this example is the one for the transitive predicate GENLMT that is shown in figure 4.6. The last step of the proof is simplified to TRUE because the system unifies the given knowledge (C_TPTP_MEMBER3515_MT MTVISIBLE) with the subformula (X0 MTVISIBLE) in the subgoal (SOME X0 ((X0 MTVISIBLE) AND (C_WORLDGEOGRAPHYMT GENLMT X0))) to obtain the new subgoal (C_WORLDGEOGRAPHYMT GENLMT C_TPTP_MEMBER-3515_MT), which is simplified to TRUE using the information in the relevant hierarchy (figure 4.6).

If this information had been ignored then the proof tree would have looked like the one in figure 4.5 in which a similar subgoal (C_WORLDGEOGRAPHYMT GENLMT C_WORLDGEOGRAPHYDUALISTMT) is proved using multiple general inference steps instead of one specialist inference step.

The transitive hierarchy is currently used uniformly for all knowledge bases, but in principle we should allow different transitive hierarchies in different knowledge bases, with inheritance to descendant KBs.

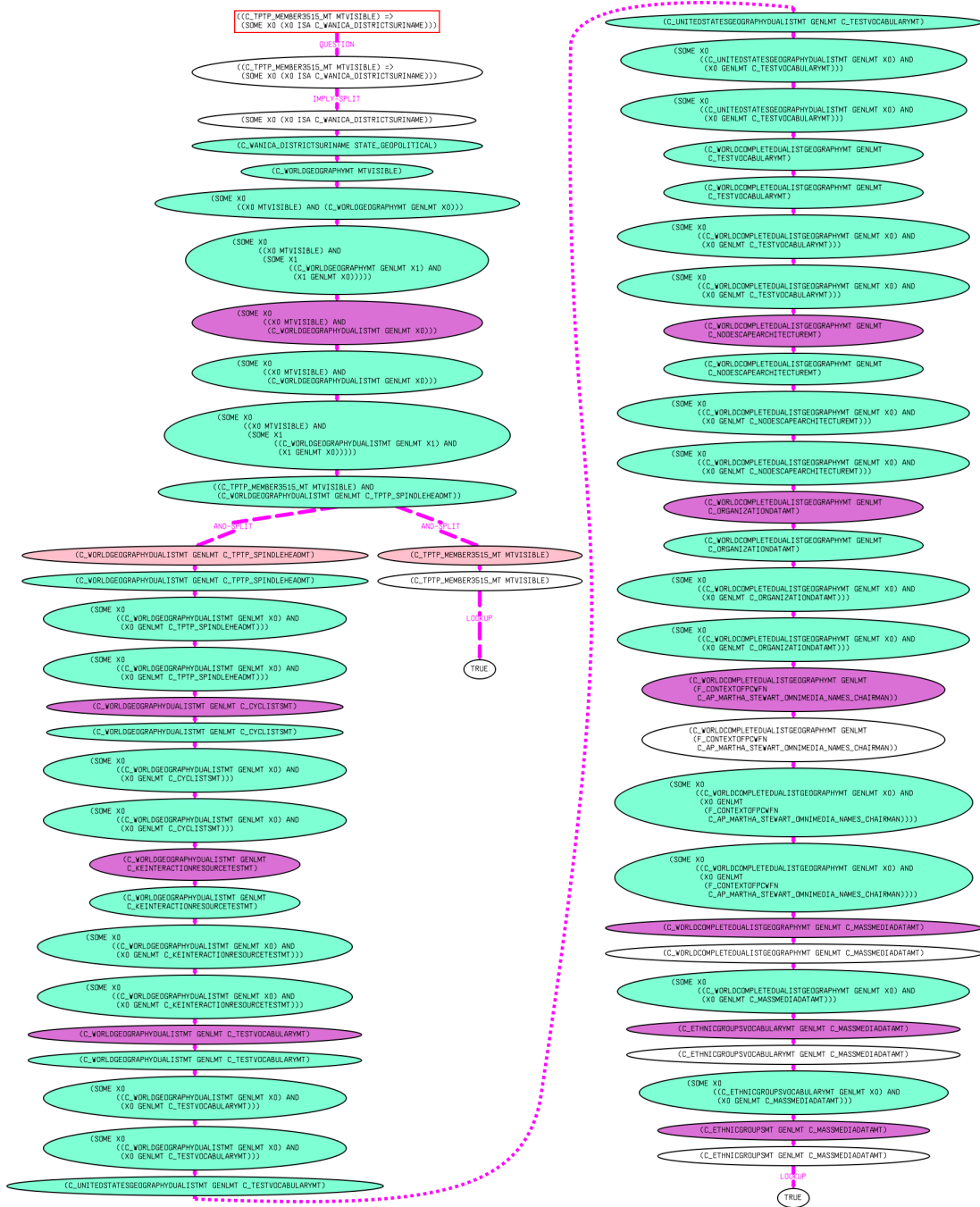


Figure 4.5: Proof found for the problem CSR034 when the extraction of transitive information is not enabled.

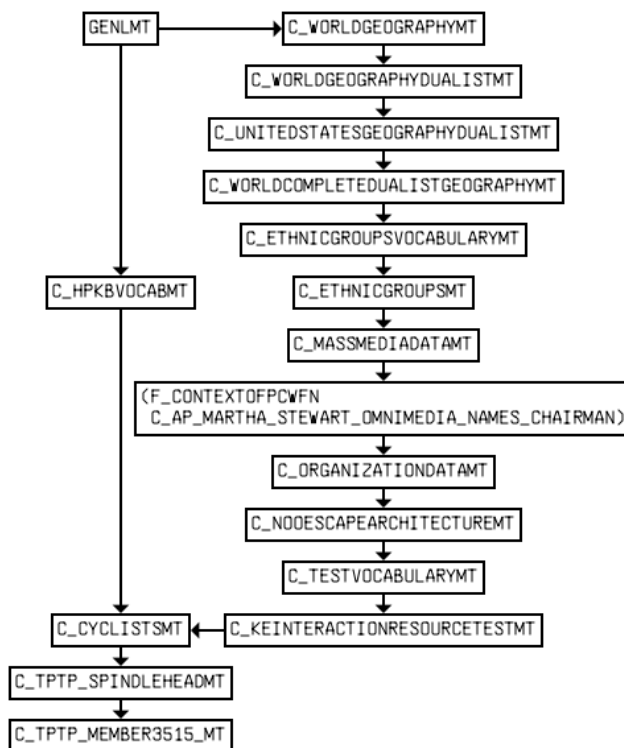


Figure 4.6: Hierarchy automatically built for the arguments of the transitive predicate GENLMT in the problem CSR034.

4.8 Storage Framework

The storage framework takes care of storing a formula in a KB and of producing relevant forward inferences with it. We have so far focused in our work on EPILOG2 on rewriting the QA framework, leaving the storage framework in a rather preliminary state.

Currently the only forward inferences made are ones based on formulas that could potentially expand type knowledge. For example, if [Fido Barks] is asserted the forward inference with the knowledge ($\forall x$ [x Barks] [x Dog]) would be attempted because the above knowledge is marked as relevant to type inference (in fact it'll generate the formula [Fido Dog] that will be used to update the type hierarchy).

4.9 Question-answering (QA) Framework

EPILOG2's QA algorithm (see figure 4.7) creates a graph of goals from the given question, starting with the asserted and negated forms of the question. A loop is entered in which a *seemingly best subgoal* g is selected, a *seemingly best retrieval key* for subgoal g is used to retrieve a set of formulas and their keys (*seemingly relevant to* g), and for each retrieved formula and each of its keys, an attempt is made to generate a new subgoal from g to be added as a subgoal of g . The subgoal is added only if it creates no loops, and if possible it is reduced to one or more simpler subgoals.

The 3 italicized phrases above require an evaluation metric to assign a utility score to each candidate. We will mention which metrics we are currently using when we discuss them below.

The generation of subgoals involves both natural deduction steps and *goal-chaining* operations based on embedded, resolution-like steps that unify subfor-

```

QA( $f, kb$ ) begin
   $q \leftarrow$  create question for  $f$  in  $kb$ ;
   $proof\text{-}sg \leftarrow$  create subgoal for  $f$ ;
   $disproof\text{-}sg \leftarrow$  create subgoal for (not  $f$ );
  add( $proof\text{-}sg, q$ );
  add( $disproof\text{-}sg, q$ );
  while  $sg \leftarrow$  get best subgoal from subgoal agenda of  $q$  do
     $\langle f_{sg}, key_{sg} \rangle \leftarrow$  get best retrieval action of  $sg$ ;
    foreach  $\langle f_r, key_r \rangle \leftarrow$  retrieve from  $kb$  using  $\langle f_{sg}, key_{sg} \rangle$  do
       $child\text{-}f \leftarrow$  attempt goal chaining between  $\langle f_{sg}, key_{sg} \rangle$  and
       $\langle f_r, key_r \rangle$ ;
      if  $child\text{-}f$  then
         $child\text{-}sg \leftarrow$  create subgoal for  $child\text{-}f$ ;
        add( $child\text{-}sg, sg, q$ );
      end if
    end foreach
  end while
  return  $AnswersOf(q)$ ;
end

add( $child\text{-}sg, parent\text{-}sg, q$ ) begin
  if adding  $child\text{-}sg$  under  $parent\text{-}sg$  doesn't form a loop then
    if  $simple\text{-}sgs \leftarrow simplify(child\text{-}sg)$  then
      foreach  $simple\text{-}sg$  in  $simple\text{-}sgs$  do
        add( $simple\text{-}sg, child\text{-}sg$ );
      end foreach
    else
      add  $child\text{-}sg$  to the subgoal agenda of  $q$ ;
      create and add the retrieval actions for  $child\text{-}sg$  to the retrieval
      agenda of  $child\text{-}sg$ ;
    end if
  end if
end

```

Figure 4.7: High level algorithm of EPILOG2's QA framework.

mulas, with due regard for their polarity (upward- or down-ward entailing environments). The most important aspects of the natural deduction framework are:

Subgoal simplification/splitting:⁴ the strategies currently implemented are:

1. splitting apart conjunctive goals;
2. splitting disjunctions and implications into individual subgoals, while assuming the negation of the other disjuncts;
3. reducing a universally quantified subgoal to the goal of proving an “arbitrary instance”, i.e., one based on a new constant. This can be considered the dual of what Skolemization does for existential formulas during storage;
4. splitting equivalences into two implicative goals.

Inference graph handling: the routines in this module provide two major functions:

1. *Work duplication avoidance:* Every time a subgoal is added it may already be present in other branches of the inference tree⁵. When this condition is detected the new subgoal is blocked, and a *copy link* is added to connect it with the equivalent subgoal already present in the inference tree (thus turning the inference tree into a graph). Copy links allow answers to be propagated to all the equivalent subgoals when the original subgoal is solved.

⁴See function *simplify* in figure 4.7.

⁵Because each subgoal is associated with a knowledge base admissible for solving it, the detection of equivalent subgoals must consider not only the subgoal formulas but also the associated KBs. In case of subgoals with different KBs, a partial equivalence can be established if the KB of the previously created subgoal is contained in that associated with the new subgoal.

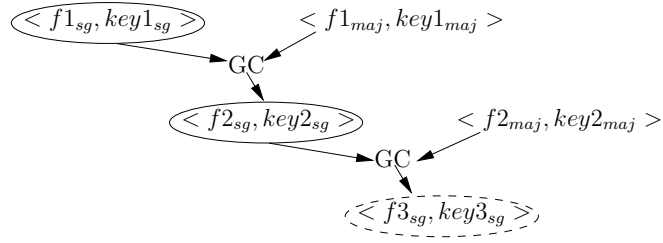


Figure 4.8: The dashed subgoal is **not** added if $f1_{maj} \equiv f2_{maj}$ and $key1_{maj} \equiv key2_{maj}$ and $key3_{sg}$ was a part of $f1_{maj}$. This avoids the creation of an inference loop that creates a cascade of ever growing subgoals.

2. *Loop avoidance*: Subgoals that would create loops in the inference graph should not be added, since they can impair the performance of the theorem prover.

Two types of loop creation are currently avoided:

- A subgoal is not added if it would duplicate an ancestor. Note that because of assumption-making one subgoal is considered equivalent to another only if their formulas as well as their associated KBs are the same.
- Cascades of ever-growing subgoals are avoided by blocking the inferences that would create a subgoal g by goal chaining between: **1)** the same retrieved wff and key used to produce g 's parent, gp , and **2)** a key in gp that comes from the retrieved wff used to create gp . Figure 4.8 contains a representation of this case.

Agendas: two agendas are used in the QA framework, one to order the subgoals pertaining to a question and one to order the retrieval actions within a subgoal (for each subgoal, there is a retrieval action for each key contained in the subgoal's formula). The agenda system defines an interface (like the knowledge module) and currently provides 3 agenda schemas, one list-based, one AVL tree-based, and a third that is hierarchy-based.

Each agenda (that is an instantiation of one of the agenda schemas provided) has a particular evaluation function associated with it that estimates the value/priority of each element in the agenda. This makes it possible to order the agenda's content. Some evaluation measure may require an update of the value given to previously inserted elements (and therefore a resorting) every time a new element is inserted. This can be inefficient. The hierarchical agenda overcomes this problem by associating to the elements in it a hierarchical structure. To find the best element in it one has to traverse this structure deciding at each element whether to stop or to proceed along one of its children based on the evaluation measure associated with the current node and its children. When an element is added to the agenda, it is added as a child of a preexisting element and, if necessary, a change in the evaluation measure is propagated to its ancestors.

In selecting which subgoal to use for the next inference the hierarchical agenda structure is used. But in that case the hierarchical structure is already present – it is the inference graph. The heuristic used to evaluate the importance of each subgoal, g , is critical in the performance of the theorem prover; it is currently based on two counteracting quantities:

Costs **1)** the size of the formula associated with g relative to the size of the biggest formula among the siblings of g ; **2)** the % of times a descendant of g or g itself was selected for inference but no improvement was obtained.⁶

Gains **1)** the % of g that is solved (this is greater than 0 only for a subgoal that at some point can be split, e.g., a conjunction); **2)** the % difference between the size of g 's formula and the size of the smallest formula among the descendants of g whose solution would imply a solution of g ; for conjunction of subgoals, their average size is considered.

⁶An improvement is measured either by a decrease in size of the resulting subgoal, or solution of the subgoal.

Optimizations: A couple of additional optimizations are used to reduce the number of possible inferences. The first concerns conjunctive subgoals split by the simplification rule described above. Consider a conjunctive subgoal g that is split into its conjuncts $g_1 \dots g_n$. It's inefficient to leave all n conjuncts viable alternatives to solve g at the same time, as that would just dilute the inference effort on the n children. It's more efficient to enable only one of the conjuncts as usable for inference till it's solved. When that conjunct is solved the second will be enabled till all of them are solved or till one of them cannot be solved.

Exceptions to the one-conjunct policy can be made, for example, when one of the conjuncts is connected with a copy link to another node. In that case it may be necessary to enable the conjunct for inference because otherwise it may be impossible to solve the node connected via the copy link.

The second optimization reduces the number of keys available for inference in a given goal formula. Given a goal g that was generated by goal chaining between a parent goal g_p and a formula maj from a knowledge base, the keys in g that may be used for inference are only those coming from maj . When those keys are exhausted then the keys coming from the maj used to produce g_p will be used and so on till only keys coming from the original goal (question) remain.

A solution is still guaranteed to be found if it exists, as the following argument shows. If the process above doesn't find a solution, then the only possibility that remains is that at some point, where we're tackling some goal g , we need to select a key from the original question, say key_q , in order to find a solution, instead of continuing to use keys from each goal's respective maj . But that could work only if the modifications made to key_q by the inference steps leading up to g made key_q provable, whereas the version of key_q found in the original question was unprovable. But this is impossible because the modifications a key can undergo during inference are:

- a matchable variable is replaced by a constant or complex compatible syn-

tactic entity (e.g., a function for a term variable);

- a matchable variable is replaced by a matchable variable;
- a matchable variable is replaced by an unmatchable variable.

All these transformations make the new key harder to prove than the original, or equivalent.

Recursive QA: among the requirements of explicit self-awareness described in section 2.3 there were the ability to reason about beliefs (and any other attitude in general) and the ability to reason about one’s own knowledge (called autoepistemic inference and referred to in humans as noetic consciousness, see description on page 57). We also mentioned that we draw from the work of Kaplan (see (Kaplan, 2000)) to overcome the problems of the traditional techniques used to tackle attitudes and autoepistemic knowledge. In particular our guiding principles are:

- commitment to a computational notion of knowing that resembles the characteristics of human introspection and distinguishes it from the notion of being able to infer something (lest we make an introspective agent omniscient and make knowing, etc., undecidable). In addition to making knowing a computational notion, we also want it to satisfy several natural properties, for example, that if we know that ϕ then we also know that $(\phi \vee \psi)$, or that if we know $(\phi \wedge \psi)$ we also know ϕ (see (Morbini and Schubert, 2007) for a more detailed listing of the desirable properties of a computational notion of knowing).
- given that we are committed to a representation language with the same expressiveness as natural language, we reject attempts to simplify the representation language to make the ability to infer a proposition tractable.
- following our transparency requirement, we also avoid methods that make hidden assumptions (for example, implicit assumptions included in the infer-

ence rules) in our approach to autoepistemic inference. An example would be assuming that something is false just because we cannot infer it from a given knowledge base, in effect making a hidden knowledge completeness assumption (see chapter 5).

Our approach to autoepistemic inference is based on two concepts:

- explicit knowledge describing which portions of our knowledge are complete (based on a concept of familiarity/relevancy/saliency).
- a computational notion of knowing that is guaranteed to quickly decide whether something is known (i.e. true), unknown or false.

The second point is the relevant one with respect to this paragraph (we will see examples of the completeness axioms in chapter 5). We redesigned the QA framework so that it supports initiating questions within a question-answering process (called recursive QA). Technically this doesn't require any special technique, but by designing each component on which the QA framework depends (basically every other subsystem, from normalization to knowledge access) to be self-contained and modular, we ensured support for recursive QA and therefore for our notion of autoepistemic inference.

4.10 Summary

In summary, in this work we re-wrote/designed most of the modules in EPILOG1 to solve the problems highlighted in section 3.4 that seriously compromised the progress of the COMA demo along the lines of explicit self-awareness.

Although EPILOG2 has functionalities that were not present in EPILOG1 (e.g., parser, recursive question/answering, hierarchical knowledge bases, hierarchical agenda, testing framework, etc.) the redesign process didn't cover all systems and

functionalities of EPILOG1. In particular, the following functionalities present in EPILOG1 have not yet being covered by this effort:

- EPILOG1 is capable of basic probabilistic reasoning. We decided to postpone introducing this functionality in EPILOG2, until more complete foundations for probabilistic reasoning were developed.
- Scalable access schema. Although no scalability testing has been done on EPILOG1, it has an access schema that in principle guarantees a more efficient access to knowledge. However, this schema had problems in particular with respect to its usability and its goal-chaining completeness. We partially developed a revision of EPILOG1's access schema but further changes were required and therefore we put this project on hold and focused on a simple access schema that allowed extensive testing of EPILOG's other functionalities.
- Some specialists have not been implemented: among these one of the most important is the time specialist.
- Forward inference is given particular emphasis in EPILOG1. However, because of our focus on question-answering (i.e. a process focused on goal chaining) and the completeness problems with the access schema in EPILOG1's goal chaining, we focused on the redesign of goal chaining in EPILOG2.

We overcome the lack of a testing mechanism and thorough evaluation of EPILOG functionalities by

- adding a set of test problems to probe the major functionalities of EPILOG, where these can be run periodically to check whether a new revision of the system introduces bugs or other malfunctions.

- a performance evaluation based both on a publicly available set of commonsense questions (in particular focused on self-awareness), and on a more standardized set of problems for FOL theorem provers. This evaluation will be described in detail in chapter 5.

5 Evaluation

The natural way to test COMA’s capabilities, both on the reasoning front and on the representation front, is to use a publicly available set of commonsense problems (selecting a subset of questions oriented towards self-awareness).

Among several collections that are available, we opted for the set of problems contained in the Research Cyc knowledge base. They are a set of more than 1600 problems¹ that provide both the English formulation of a question and its translation into CycL². In addition to the abundance of interesting and challenging questions, another advantage of using this dataset is that it allows comparison between our and Cyc’s interpretation of each question.

The last point highlights the problem of comparison for systems that use a dataset based on English for their evaluation. Because a question expressed in English can be formalized in many ways and at various levels of detail, it is very difficult to use the results obtained to compare different systems. This lack of a dataset expressed in logic to facilitate comparisons is not easily solved given the lack of agreement on a single logical language well-suited for NL; and even if such a language existed each English sentence can still be interpreted in many ways

¹This dataset also contains most of the relevant questions in Vaughan Pratt’s list available at <http://boole.stanford.edu/pub/cyc.report>.

²<http://www.cyc.com/cycdoc/ref/cycl-syntax.html>

and at different levels of detail.

Therefore, to give a more complete picture of the performance of the EPILOG system and to facilitate comparisons with other systems, we decided to evaluate it as well against the widely used TPTP dataset for FOL theorem provers. This puts the basic performance of the reasoner in perspective with respect to the state of the art in FOL theorem provers. The evaluation on Cyc’s commonsense test cases instead tests the features that distinguish EPILOG from a traditional FOL theorem prover.

The chapter is organized as follows: first we describe the evaluation using a small selection of questions that are either hand-built or selected from the Cyc data set mentioned above; the evaluation using the TPTP dataset follows in section 5.2. We conclude with a summary of the results.

5.1 Evaluation with Cyc and hand-built questions

In this section we will describe the set of questions used to test Epi2Me’s self-awareness and commonsense reasoning. The questions have been manually encoded in EL because the general-purpose English to EL translator is not yet robust enough to handle these questions. However care has been taken not to simplify the EL form of those questions to make the job of the reasoner easier; instead we made an effort to produce EL versions that would likely be produced by an automatic, compositional English-to-EL translator. This is why some questions may appear more complex than one might expect, based on traditional “intuited” formalizations of English sentences.

In the formulas used in the following examples, we use Epi2Me as the internal constant that refers to the system itself.

About 81% of all commonsense test cases contained in Cyc can be answered by Cyc, for the remaining 19% the required knowledge has not yet been added. Of the 5 questions from Cyc found in this section, 2 are answered by Cyc. For those the knowledge used by Cyc will be reported.

We start with a couple of questions designed to show autoepistemic reasoning based on explicit assertions about the completeness of portions of our knowledge and a computational notion of *knowing*.

The first question is “Do pigs have wings?”. Expressed in EL with:

$[(K \text{ Pig}) \text{ HaveAsPart } (K \text{ Wing})]$.

K is a reification operator that maps a predicate (here, Pig) to a *kind* (here, the kind whose realizations satisfy the predicate Pig).

The relevant knowledge used by this question is the following:

Pigs are a natural kind. $[(K \text{ Pig}) \text{ NaturalKind}]$
Wings are a major kind of body part. $[(K \text{ Wing}) (\text{Major } (\text{KindOf } \text{BodyPart}))]$
I’m familiar with pigs. $[\text{Epi2Me } \text{FamiliarWith } (K \text{ Pig})]$
Closure axiom that asserts the completeness of the agent’s own knowledge with respects to major body parts of natural kinds with which it is familiar. $(\forall y \ [[y \text{ NaturalKind}] \wedge [\text{Epi2Me } \text{FamiliarWith } y]]$ $(\forall x \ [x (\text{Major } (\text{KindOf } \text{BodyPart}))]$ $\quad [[y \text{ HaveAsPart } x] \Rightarrow$ $\quad \quad [\text{Epi2Me } \text{Know } (\text{that } [y \text{ HaveAsPart } x])]])$
Continued on next page

Positive introspection. $(\forall_{wff} w [w \text{ without-free-vars}]$ $[[Apply \text{ 'introspect? 'w} = \text{'yes}] \Leftrightarrow$ $[Epi2Me \text{ Know (that w)}}]]$
Negative introspection. $(\forall_{wff} w [w \text{ without-free-vars}]$ $[[Apply \text{ introspect? 'w} = \text{'no}] \Leftrightarrow$ $(\text{not } [Epi2Me \text{ Know (that w)}}]]]$

This question is answered negatively because the facts that the agent is familiar with pigs and that wings are major body parts enable it to use the closure axiom and negative introspection.

Before moving to the next question it is opportune to highlight some of the differences of our approach to the above question (and similar ones) from traditional nonmonotonic reasoning (NMR) approaches (such as Default Logic (Reiter, 1980), Autoepistemic Logic (Moore, 1985; Lakemeyer and Levesque, 2005) and Circumscription (McCarthy, 1980)). These traditional NMR approaches are characterized by making an implicit assumption about the completeness of the available knowledge, so that if something cannot be derived it can be assumed to be false. This implicit assumption is hard to motivate in real-world systems because it is very difficult to build knowledge bases that are logically complete, except for very special reasoners working on very restricted domains.

In addition, proving that something is not entailed by the knowledge base is an undecidable problem and therefore potentially a non-terminating process. This fact is a further indication of the lack of realism of these approaches to coping with incomplete knowledge. Humans, in fact, can quickly answer whether they know something or not (as opposed to answering whether something is true or

false, which may require a long time or might “never” terminate in the case of humans as well).

It is this lack of realism of traditional NMR techniques, and clues from thinking about how we answer questions similar to the example just examined, that justify our approach, characterized by an explicit completeness assumption that applies only to a limited portion of the agent’s knowledge (i.e. in this case knowledge regarding major parts of things with which the agent is familiar) together with a computational notion of *knowing*.

The next question uses axioms on how the agent acquires knowledge from perception to complete its knowledge about phone ringing events. In this case the agent answers negatively to the question “Did the phone ring during E1?” (where E1 is a particular event that stands for example for “in the last 4 hours”) because it has no memory of phone ringing events during E1 and it knows that if its hearing is working properly and the phone is near enough to be heard then it’ll perceive and remember phone ringing events.

The question in EL becomes:

$(\exists e [e \text{ During } E1] [[P1 \text{ Ring}] ** e])$

P1 is a phone. [P1 Phone]
During P1, I was near P1. $(\forall e [e \text{ During } E1] [[Epi2Me \text{ WithinEarshotOf } P1] @ e])$
During E1, I was conscious. $(\forall e [e \text{ During } E1] [[Epi2Me \text{ Conscious}] @ e])$
During E1, my hearing ability was normal. $(\forall e [e \text{ During } E1] [[(\text{HearingAbilityOf } Epi2Me) \text{ Normal}] @ e])$
Continued on next page

If I'm conscious, within earshot of a phone and my hearing is normal, then I'll hear the phone ringing.

$$\begin{aligned}
 &(\forall x [x \text{ Phone}] \\
 &\quad (\forall e [[x \text{ Ring}] ** e] \\
 &\quad \quad [[[[\text{Epi2Me WithinEarshotOf } x] @ e] \wedge \\
 &\quad \quad \quad [[\text{Epi2Me Conscious}] @ e] \\
 &\quad \quad \quad [[(\text{HearingAbilityOf Epi2Me}) \text{ Normal}] @ e]] \Rightarrow \\
 &\quad \quad [[\text{Epi2Me Hear } e] @ e]])))
 \end{aligned}$$

If I hear something, I'll know about it.

$$(\forall e [[\text{Epi2Me Hear } e] @ e] [\text{Epi2Me Know (that } [[\text{Epi2Me Hear } e] @ e]])])$$

Note that currently we manually assert whether Epi2Me is conscious during E1 and whether the hearing ability of Epi2Me is normal, but these facts should be automatically asserted by an unconscious demon in charge of updating knowledge about the agent's current cognitive state.

Also this question is an example in which we explicitly state under what conditions the agent's knowledge is complete regarding a perceptual event (i.e. hearing) with a known cause (i.e. a phone ringing).

The third question tried is “How old are you?”, which in EL becomes

$$\begin{aligned}
 &(\text{wh}_{term} x (\exists_{term} y ['x \text{ rounds-down } 'y] \\
 &\quad (\exists z ['y \text{ expresses } z (K (\text{plur year}))] \\
 &\quad \quad (\exists e [e \text{ at-about Now}] \\
 &\quad \quad \quad [[z \text{ age-of Epi2Me}] ** e]]])))
 \end{aligned}$$

We have assumed that the representation of the question would be expanded pragmatically to include conventional restrictions on the form of the answer expected, i.e., an answer in rounded-down years rather than, say, seconds. These

pragmatic constraints depend on the question itself; for example they would be different for a question like “How old is this bagel/star/rock/etc.?”. In the future we would like to automatically include such constraints by means of “cooperative conversation axioms”. We might have an axiom saying something like: *If X informs Y about a quantitative attribute F (such as weight, age, temperature, etc.) of some entity Z, then X is conversationally obligated to express F(Z) in units that are conventional for entities of the type(s) instantiated by Z.* In addition we would need various axioms about the conventional units for expressing weight, age, etc., of various types of entities. These axioms would then be used to refine the raw logical form of a question to include the pragmatic constraints. However, here we just focused on solving the question, manually adding the necessary pragmatic constraints.

Some of the key knowledge used to answer this question is the following.

<p>This axiom defines the age of an entity during a particular event, when the entity’s birth date is known.</p> $(\forall y (\forall x [x \text{ (be (birth-date-of } y))]$ $(\forall e$ $[[(\text{time-elapsed-between (date-of } e) \text{ } x) \text{ age-of } y]$ $@ e]]))$
<p>Axiom defining the relation between the ** and @ operators.</p> $(\forall_{wff} w (\text{all } e [[w @ e] \Leftrightarrow$ $(\exists e1 [e1 \text{ same-time } e] [w ** e1]]))$
<p>Axiom that describes which specialist function to call to express the function time-elapsed-between in a particular type.</p>
<p>Continued on next page</p>

$$\begin{aligned}
&(\forall x [x \text{ is-date}] (\forall y [y \text{ is-date}] \\
&\quad (\forall_{pred} \text{ type } ['\text{type el-time-pred}] \\
&\quad\quad (\forall r ['r = (\text{Apply } 'diff-in-dates? 'x 'y 'type)] \\
&\quad\quad\quad ['r \text{ expresses (time-elapsed-between } x \ y) \\
&\quad\quad\quad\quad (K (\text{plur type}))])))
\end{aligned}$$

The most interesting part of this example is the use of a set of axioms based on the *Apply* function to make the reasoning system “aware” of a set of procedures useful in computing mathematical operations and in doing type conversions. In this way EPILOG2 is able to return the answer to the question expressed as an integer that is the floor of the amount of time in years that has elapsed between the date of birth of EPILOG and now (the moment of speech). In EL the unifier found for the variable *x* of the initial question is: (amt 18 (K (plur year))).

The fourth question is “What’s your name?”, which expressed in EL is

$$\begin{aligned}
&(\exists e0 [e0 \text{ at-about Now0}] \\
&\quad [(\text{wh } z [[z \text{ name}] \wedge [\text{Epi2Me have } z]] \\
&\quad\quad (\exists y [y \text{ thing}] [y (\text{Be } (L \ x [x = z]))])) ** e0])
\end{aligned}$$

Some of the key knowledge used to answer this question is the following.

The event Now0 is during the event e2.

[Now0 during e2]

The event e2 is characterized by EPILOG having the name 'epilog-name'.

[[Epi2Me have 'epilog-name'] ** e2]

If one event is characterized by something possessing something else, then that will also be true for any event during the first event.

$$\begin{aligned}
&(\forall x (\forall y (\forall z [[x \text{ have } y] ** z] \\
&\quad\quad (\forall zz [zz \text{ during } z] [[x \text{ have } y] @ zz]])))
\end{aligned}$$

Of interest here is the last axiom because it ascribes “inward persistence” (homogeneity) to predicate *have*, a property it shares with other *atelic* predicates.

The two other formulas are hand-additions to the current knowledge base, but they should be automatically inserted, the first by the English to EL generator, the second by a self-awareness demon that is in charge of maintaining basic information about an individual, for instance, its name, its state (e.g., sleeping, awake, etc.) and its “state of health” (e.g., cpu consumption, free memory, garbage collection status, etc.).

To correctly answer this question the reasoner also uses lexical knowledge that states which predicates are atemporal and therefore can be moved out of the scope of the ****** operator (see (Schubert, 2000)). This knowledge is expressed in EL and it is used by the normalizer. An example is (*'thing* EL-type-pred), stating that *'thing'* is a type predicate and therefore atemporal.

The fifth question tried shows how EPILOG could answer questions about its own knowledge. The question is “What do you know about the appearance of pigs?”, which in EL we expressed as:

(wh x [x appearance-fact-about (K (plur pig))])

Some of the relevant knowledge involved in this example is:

Pigs are thick-bodied. [(K (plur pig)) thick-bodied]
The predicate ‘thick-bodied’ is an appearance predicate. ['thick-bodied appearance-pred]
Every wff that uses an appearance predicate is a fact about the appearance of its subject. ($\forall_{pred} p$ ['p appearance-pred] ($\forall x$ [x p] [(that [x p]) appearance-fact-about x]))

This is a relatively simple example. One could construct much more complex formulas pertaining to the appearance of something, e.g., that the appearance of a person’s hair – say, color and style – constitutes appearance information about the person.

The sixth question is the first obtained directly from the commonsense test cases in Research Cyc. The question is “Can gasoline be used to put out a fire?”. In Cyc this is the test case named `#$CST-CanYouUseGasToPutOutAFire`, and the question is expressed as: `((TypeCapableFn behaviorCapable) GasolineFuel ExtinguishingAFire instrument-Generic). (TypeCapableFn behaviorCapable)` returns a predicate that describes the capacity for a certain behavior of a certain type of thing in a certain role position, i.e. the predicate has 3 arguments. In effect the question becomes, “Is gasoline-fuel behaviorally-capable of being a generic-instrument in fire-extinguishing?”

We also interpret the question generically, but we adhere more closely to the English phrasing and take it as asking more specifically if a *person* can use gasoline to put out a fire:

$$\begin{aligned}
 &(\exists e \\
 &(\exists x [x \text{ person}] \\
 &(\exists y [y ((nn \text{ gasoline}) \text{ fuel})] \\
 &(\exists z [z \text{ fire}] [[x (\text{able-to } ((\text{in-order-to } (\text{put-out } z)) (\text{use } y))))] @ e))))))
 \end{aligned}$$

Some of the knowledge relevant to this question is:

If some person is able to use some stuff to put-out a fire then s/he must be at the same location as the fire, must have at hand that stuff and that stuff must be flame-suppressant.

Continued on next page

$ \begin{aligned} &(\forall e \\ &\quad (\forall x [x \text{ person}] (\forall y [y \text{ stuff}] (\forall z [z \text{ fire}] \\ &\quad\quad ([x \text{ (able-to ((in-order-to (put-out } z)) \text{ (use } y))]) @ e] \\ &\quad\quad \Rightarrow ([x \text{ has-at-hand } y] @ e] \wedge [x \text{ loc-at } z] @ e] \\ &\quad\quad [y \text{ flame-suppressant}])))))) \end{aligned} $
<p>Gasoline is flammable stuff.</p> $(\forall x [x ((\text{nn gasoline}) \text{ fuel})] [[x \text{ flammable}] \wedge [x \text{ stuff}]])$
<p>Flammable things are not flame-suppressant.</p> $(\forall x [x \text{ flammable}] (\text{not } [x \text{ flame-suppressant}]))$

The question is answered negatively by using the knowledge that to be able to put-out a fire one must use a flame-suppressant material, and gasoline is not a flame-suppressant material.

The seventh question is Cyc's question named `#$CST-DoesCycHaveABiological-Father`, which in English is "Do you (Cyc) have a biological father?". In Cyc the question is represented as `(thereExists ?F (biologicalFather Cyc ?F))`.

We expressed the question in EL as follows:

$$\begin{aligned}
&(\exists e0 [e0 \text{ at-about Now}] \\
&\quad (\exists y [[\text{Epi2Me (have-as ((attr biological) father)) } y] ** e0]))
\end{aligned}$$

In this question, `have-as` is a subject adding operator that takes a n -ary predicate as argument and returns a $n+1$ -ary predicate. In this case `((attr biological) father)` is the monadic predicate true for all individuals that are biological fathers. `(have-as ((attr biological) father))` is the binary predicate that is true for all pairs of individuals in which the object of the predicate is the father of its subject.

The relevant knowledge for this example is:

EPILOG is an artifact. [Epi2Me artifact]
No artifact is a natural object. ($\forall x$ [x artifact] (not [x natural-obj]))
A creature is a natural object. ($\forall x$ [x creature] [x natural-obj])
All creatures, and only creatures, have a biological father. ($\forall x$ ([x creature] \Leftrightarrow <div style="margin-left: 40px;">($\exists y$ ($\exists e$ <div style="margin-left: 40px;">[[x (have-as ((attr biological) father)) y] ** e]])) </div> </div>

The question is answered negatively by using the knowledge that EPILOG is an artificial thing and therefore not a natural object. Further it's known that only creatures can have a biological father and that creatures are a subtype of natural objects.

The eighth question corresponds to the Cyc question named `#$CST-Animals-DontHaveFruitAsAnatomicalParts-HypothesizedQueryTest`. In Cyc the question is expressed as `(implies (isa ?ANIMAL Animal) (not (relationInstanceExists anatomicalParts ?ANIMAL Fruit)))`.

In EL we expressed the question (more naturally, we claim) as:

($\forall e0$ (No x [x animal]

[[x (have-as anatomical-part) (K fruit)] ** e0]]))

The relevant knowledge for this example is:

Plant stuff is not animal stuff. ($\forall x$ [x plant-stuff] (not [x animal-stuff]))
Continued on next page

<p>Fruits are made of plant stuff.</p> $[(K \text{ fruit}) \text{ made-of } (K \text{ plant-stuff})]$
<p>Animals are made of animal stuff.</p> $[(K \text{ animal}) \text{ made-of } (K \text{ animal-stuff})]$
<p>If an individual x is made of (kind of stuff) p and if (kind of stuff) q is a supertype of p then x is made of q.</p> $(\forall x (\forall_{pred} p [x \text{ made-of } (K p)] \\ (\forall_{pred} q (\forall y [y p] [y q]) \\ [x \text{ made-of } (K q)]))))$
<p>If an individual x is made of (kind of stuff) p and if (kind of stuff) q is disjoint from p then x is not made of q.</p> $(\forall x (\forall_{pred} p [x \text{ made-of } (K p)] \\ (\forall_{pred} q (\forall y [y p] (\text{not } [y q])) \\ (\text{not } [x \text{ made-of } (K q)]))))$
<p>If a type p is made of (kind of stuff) q then all individuals of type p are made of q.</p> $(\forall_{pred} p (\forall_{pred} q [(K p) \text{ made-of } (K q)] \Leftrightarrow \\ (\forall y [y p] [y \text{ made-of } (K q)])))$
<p>Every part is made of the material of the whole.</p> $(\forall w (\forall e (\forall p \\ ([w \text{ (have-as anatomical-part) } p] ** e] \Rightarrow \\ (\forall wm [w \text{ made-of } wm] [p \text{ made-of } wm])))))$

We decided to answer the question by saying that all parts are made of the same substance of which the whole is made. However the case of artificial parts/organs is not captured by this knowledge. One could improve it by saying that parts must be made of materials compatible with the material of which the whole is made. This would work for parts made of titanium, or other durable inert materials. However

it'll not cover the case of external body parts that can be made of incompatible materials, for example a wooden leg.

The ninth question corresponds to the Cyc question named `#$CST-DoAgents-BelieveWhatTheyKnow`. The English version of the question reads out as “If you know that something is the case, do you believe that it is the case?”. In Cyc the question is represented as: `(implies (knows ?AGT ?PROP) (beliefs ?AGT ?PROP))`. In EL we provide the following representation as a direct reflection of English surface form³:

$$\begin{aligned}
 &(\forall e0 [e0 \text{ at-about Now}] \\
 &(\forall x [x \text{ thing}] \\
 &[[[Epi2Me \text{ know (that } (\exists e1 [e1 \text{ at-about } e0] \\
 &\quad [[x \text{ (be the-case)}] ** e1]]]) ** e0] \\
 &\Rightarrow \\
 &(\exists e2 [[e2 \text{ at-about Now}] \wedge [e0 \text{ same-time } e2]] \\
 &\quad [[Epi2Me \text{ (believe (that } (\exists e3 [e3 \text{ at-about } e2] \\
 &\quad \quad [[x \text{ (be the-case)}] ** e3]])) ** e2]]))
 \end{aligned}$$

The key knowledge to answer this question is the following:

If an event is characterized by somebody knowing something then it is also characterized by the same agent believing that thing.

$$(\forall e (\forall x (\text{all } p [[x \text{ know } p] ** e] \Rightarrow [[x \text{ believe } p] ** e]])))$$

The version of this question found in Cyc is similarly answered using the following knowledge:

```
(knows HYP-IntelligentAgent-163328 HYP-Thing-163329)

(genlPreds knows beliefs)
```

³elaborated with the events and event relations introduced by the temporal deindexing, which follows logical form computation (Schubert and Hwang, 2000).

Note that Cyc uses a technique similar to that used by EPILOG to answer an implication containing universally quantified variables. In particular, the universal variables are instantiated with new constants (this is the dual of Skolemization as already mentioned before), then the antecedent is assumed and the system attempts to prove the consequent (making use of the assumption if necessary). However, the Cyc encoding treats knowing and believing as time-independent.

The tenth and last example corresponds to the Cyc commonsense test case named `#$CST-CanYouAttackSomeoneWithAGolfClub`. In English the question is “Can you attack someone with a golf club?”. Cyc expresses it in the same way as question 4: `((TypeCapableFn behaviorCapable) GolfClub PhysicallyAttacking-AnAgent deviceUsedAsWeapon)`. In EL we represent the question as:

$$\begin{aligned}
 &(\exists x [x \text{ golf-club}] \\
 &(\exists y [y \text{ person}] \\
 &(\exists z [z \text{ person}] \\
 &(\exists e [[y ((adv-a (with-instr x)) (attack z))] ** e]])))
 \end{aligned}$$

The knowledge relevant to this question is:

If an object can be swung by hand, and is solid, and weighs at least two pounds, it can be used as a striking weapon.

$$\begin{aligned}
 &(\forall x [x \text{ phys-obj}] \\
 &[[[(\exists e [[x (\text{pasv } ((adv-a (\text{by } (K \text{ hand})))) \text{swing}]] ** e]] \wedge \\
 &[x \text{ solid}] \\
 &(\exists w [[x \text{ weighs } w] \wedge [w \geq (K ((\text{num } 2) \text{pound}))]]]] \\
 &\Rightarrow (\exists e [[x (\text{pasv } (\text{use-as } ((nn \text{ striking}) \text{weapon}))) ** e]]))
 \end{aligned}$$

A golf club can be swung by hand, is solid, and weighs at least two pounds.

Continued on next page

$ \begin{aligned} &(\forall x [x \text{ golf-club}] \\ &\quad [(\text{some } e [[x (\text{pasv } ((\text{adv-a } (\text{by } (K \text{ hand}))) \text{ swing}))) ** e]] \wedge \\ &\quad [x \text{ solid}] [x \text{ phys-obj}] (\exists w [[x \text{ weighs } w] \wedge \\ &\quad \quad [w \geq (K ((\text{num } 2) \text{ pound}))]])]) \end{aligned} $
<p>For any striking weapon, one person can attack another with the weapon, by striking him or her with it.</p> $ \begin{aligned} &(\forall x [x ((\text{nn striking}) \text{ weapon})] (\exists y [y \text{ person}] (\exists z [z \text{ person}] \\ &\quad (\exists e [[y ((\text{adv-a } (\text{by-means } (Ka ((\text{adv-a } (\text{with-instr } x)) \\ &\quad \quad (\text{strike } z)))))) \\ &\quad \quad ((\text{adv-a } (\text{with-instr } x)) (\text{attack } z))]] ** e]]))) \end{aligned} $
<p>There is a golf-club.</p> $(\exists x [x \text{ golf-club}])$
<p>("by-means" modification is monotone) If an agent does some action by means of another action, then he does the first action.</p> $ \begin{aligned} &(\forall_{pred} p (\forall x (\forall y \\ &\quad (\forall e [[x ((\text{adv-a } (\text{by-means } y)) p)] ** e] [[x p] ** e]]))) \end{aligned} $

This question is answered positively by using the knowledge that golf-clubs are heavy and solid and can be swung by a person and that objects with those properties can be used to attack another person.

In Cyc, this test case is solved by using the following knowledge:

```

(implies
  (and (genls ?SPEC ?TYPE)
    ((TypeCapableFn behaviorCapable) ?TYPE ?ACT-TYPE ?ROLE))
  ((TypeCapableFn behaviorCapable) ?SPEC ?ACT-TYPE ?ROLE))

```

```
((TypeCapableFn behaviorCapable) SportsPoundingDrivingImplement
  PhysicallyAttackingAnAgent deviceUsedAsWeapon)
```

```
(genls GolfClub SportsPoundingDrivingImplement)
```

Note that the Cyc axioms used are difficult to express in English, say nothing about the type of agent involved, and collapse complex English terms into logically primitive concepts.

A variant of this last question is obtained when “you” is interpreted literally to mean Epi2Me itself. In this case the question becomes:

```
(∃ e0 [e0 at-about Now]
  (∃ x [x person]
    (∃ y [y golf-club]
      [[Epi2Me (able (Ka ((adv-a (with-instr y)) (attack x))))] ** e0]]))
```

The knowledge relevant to this question is:

To be able to attack somebody one has to able to do a physical activity.

```
(∀ e (∀ x (∀ y (∀ z
  [[[x able (Ka ((adv-a (with-instr y)) (attack z))))] ** e] ⇒
  [[x able (Ka physical-activity)] @ e]]))
```

If Epi2Me is able to do what Epi2Me considers a major activity, then Epi2Me will know that it’s able to do it.

```
(∀pred x [(Ka x) (major activity)]
  (∀ e [[[Epi2Me able (Ka x)] @ e] ⇔
    [(that [[Epi2Me able (Ka x)] @ e] knownbyme]])
```

Physical activities are a major kind of activity.

```
[(Ka physical-activity) (major activity)]
```

In this case the question is answered negatively by using introspection and the closure axiom (second row in the previous table). The axiom states that Epi2Me’s knowledge of what Epi2Me is able to do is complete with respect to major abilities. Furthermore a physical activity is a major kind of activity. But the introspective QA will not be able to confirm that Epi2Me can do a physical activity and that means that it is not able to do a physical activity, thus ruling out being able to attack somebody.

5.2 Evaluation using TPTP

The second part of the evaluation puts into perspective the performance of the reasoner with respect to standard FOL theorem provers on the classic TPTP⁴ dataset. According to a prominent tradition in the knowledge representation community, the expressiveness of KR languages needs to be carefully constrained in applications so as to ensure tractability of inference. Even full FOL is to be avoided, since it is not only intractable in the worst case but undecidable. EL and EPILOG are instead founded on the view that human-like reasoning can only be achieved with representations that are sufficiently rich to capture whatever concepts and ideas people are able to entertain, and express in language. We do not subscribe to the view that there is an “expressivity-tractability tradeoff”: whatever techniques are employed in a less expressive representation can perfectly well be adapted to the more expressive representation – e.g., think of these techniques as subroutines employed for a syntactically limited subset of the more general representation language. In a sense, our evaluation here concretely illustrates this point.

The TPTP dataset is a collection of thousands of problems written in FOL designed to assess the performance of automated theorem proving (ATP) algo-

⁴See <http://www.cs.miami.edu/~tptp/>

	FOF	FNT	CNF	SAT	EPR	UEQ	LTB
CASC-J4 (2008)	Vampire 10.0	MetaProver 1.0 Paradox 2.2	Vampire 10.0	MetaProver 1.0	iProver 0.5	Waldmeister 806	SInE 0.3
CASC-21 (2007)	Vampire 9.0	Paradox 2.2	Vampire 8.1	Paradox 1.3	Darwin 1.3	Waldmeister 806	-
CASC-J3 (2006)	Vampire 8.1	-	Vampire 8.1	Paradox 1.3	Darwin 1.3	Waldmeister 806	-
CASC-20 (2005)	Vampire 8.0	-	Vampire 8.0	Paradox 1.3	DC ² TP 10.21p	Waldmeister 704	-
CASC-J2 (2004)	Vampire 7.0	-	Vampire 7.0	Gandalf c-2.6-SAT Paradox 1.0-casc	DC ² TP 10.21p	Waldmeister 704	-
CASC-19 (2003)	Vampire 6.0	-	Vampire 6.0	Gandalf c-2.6-SAT Paradox 1.0-casc	DC ² TP 1.3- SAT	Waldmeister 702	-
CASC-18 (2002)	Vampire 5.0	-	Vampire 5.0	Gandalf c- 2.5-SAT	E-SETHEO csp02	Waldmeister 702	-

Table 5.11: Results of some of the past CASCs, from <http://www.cs.miami.edu/~tptp/CASC/>

rithms. This collection of problems is used yearly in the CASC competition in which ATP systems compete on a selected subset of problems from the TPTP dataset to solve the highest number of them. Table 5.11 shows the results of this competition in the past years.

The TPTP library is divided in several domains⁵:

- Logic
 - Combinatory logic (COL)
 - Logic calculi (LCL)
 - Henkin models (HEN)
- Mathematics
 - Set theory (SET and SEU)
 - Graph theory (GRA)
 - Algebra
 - * Relation algebra (REL)
 - * Boolean algebra (BOO)
 - * Robbins algebra (ROB)
 - * Left distributive (LDA)
 - * Lattices (LAT)
 - * Kleene algebra (KLE)
 - * Groups (GRP)
 - * Rings (RNG)
 - * Fields (FLD)
 - * Homological algebra (HAL)
 - * General algebra (ALG)
 - Number theory (NUM)
 - Topology (TOP)
 - Analysis (ANA)
 - Geometry (GEO)
 - Category theory (CAT)
- Computer science
 - Computing theory (COM)
 - Knowledge representation (KRS)
 - Natural Language Processing (NLP)
 - Planning (PLA)
 - Agents (AGT)
 - Commonsense Reasoning (CSR)
 - Software creation (SWC)
 - Software verification (SWV)
- Science and Engineering
 - Hardware creation (HWC)
 - Hardware verification (HWV)
 - Medicine (MED)
- Social sciences
 - Management (MGT)
- Other
 - Syntactic (SYN and SYO)
 - Puzzles (PUZ)
 - Miscellaneous (MSC)

The three-letter abbreviations are the identifiers used in the TPTP library to name the respective domains. Each competition is based on a subset of problems from the TPTP library and it's divided in several divisions, From <http://www.-cs.miami.edu/~tptp/CASC/22/Design.html>:

- FOF: First-order form non-propositional theorems (axioms with a provable conjecture).
- FNT: First-order form non-propositional non-theorems (axioms with a counter-satisfiable conjecture, and satisfiable axiom sets).

⁵From <http://www.cs.miami.edu/~tptp/TPTP/TR/TPTPTR.shtml>

- CNF: Mixed clause normal form really non-propositional theorems (unsatisfiable clause sets). “Mixed” means Horn and non-Horn problems, with or without equality, but not unit equality problems. Really non-propositional means with an infinite Herbrand universe.
- SAT: Clause normal form really non-propositional non-theorems (satisfiable clause sets).
- EPR: Effectively propositional clause normal form theorems and non-theorems (clause sets).
- UEQ: Unit equality clause normal form really non-propositional theorems (unsatisfiable clause sets).
- LTB: First-order form non-propositional theorems (axioms with a provable conjecture) from large theories.

Each theorem prover that participates in a CASC can decide in which divisions to compete.

In our evaluation we used the CSR⁶ problems derived from the conversion into FOL of the OpenCyc ontology (Ramachandran et al., 2005). In particular, we used the subset of CSR problems that was designed to test the scalability of a theorem prover; these were the problems designated as CSR025 through CSR074 in segments 1 to 5. Even though the access schema of EPILOG2 is a simple exhaustive one and therefore not scalable, the results will provide a good bottom-line comparison with future improvements of EPILOG. Note that there are 6 segments in the scalability task, but we didn’t use the 6th segment because the current implementation is not efficient enough to handle the size of that segment (3.3M formulas). In EPILOG2 1G of memory is required to work on the 5th

⁶See http://www.opencyc.org/doc/tptp_challenge_problem_set in particular the section *The Scaling Challenge Problem Set*.

segment. Much more compact structures and probably a switch to a lower-level language like C would be required at least for the low-level storage routines to tackle problems of those sizes. However, this conversion will probably have to wait until the code reaches a relatively stable state, to justify investing time in switching to a language (like C) in which quick prototyping and debugging is harder to do.

We adapted the JJ⁷ parser written by Andrei Tchaltsv and Alexandre Riazanov to convert the TPTP format to an EL format that can be quickly parsed using the FOL grammar in EPILOG2.

All formulas in the CSR problems from 25 to 74 are Horn clauses with functional terms. The problems are easy if the set of axioms contained in each problem is exactly that required for a solution to be found. The difficulty comes from the amount of irrelevant extra knowledge added. Segment 1 contains the smallest number of irrelevant extra axioms. Each problem in segment 2 already contains about 95% irrelevant axioms.

Table 5.12 summarizes the results. The systems compared are EPILOG1⁸, EPILOG2, and Vampire versions 8.1 and 9, which is representative of state-of-the-art FOL theorem provers⁹. All systems were run in the same conditions and were subjected to a 2 minute limit for problem. All tests were executed on a Pentium D at 2.8GHz with 1G of RAM and running Linux 2.6.28. We also included in the table the results of the latest version of Vampire, 10, but because we were unable to run it on our machine, we report the results from the online TSTP¹⁰ dataset. Those were obtained with a different timeout (10 minutes) but on a similar machine, a P4 at 2.8GHz with 1G of RAM and running Linux 2.6.18.

Table 5.13 reports the results of EPILOG2 and Vampire 8.1 and 9 on the same

⁷http://www.freewebs.com/andrei_ch/

⁸In particular it is the version of June 22nd2005.

⁹Download available at <http://www.cs.miami.edu/~tptp/CASC/J4/Systems.tgz>

¹⁰<http://www.cs.miami.edu/~tptp/cgi-bin/SeeTPTP?Category=Solutions&Domain=CSR>

Segment	Size (min/avg/max)	EPILOG1 FI	EPILOG1 no FI	EPILOG2	Depth	Vampire 8.1	Vampire 9	Vampire 10*
1	(22/59/163)	46	46	100	5.9	100	100	100
2	(-/1101/-)	46	44	92	5.6	100	100	100
3	(-/7294/-)	0	0	54	4.5	92	82	98
4	(-/42981/-)	0	0	48	4.3	30	32	42
5	(-/534435/-)	0	0	12	1.3	0	0	0

Table 5.12: Summary of the tests carried out between EPILOG1, EPILOG2 and the Vampire theorem prover versions 8.1, 9 and 10. For version 10 we report the results from the TSTP collection of solutions because we were unable to run the theorem prover on our machine. The first column contains the segment numbers (1-4) of the segments comprising the scalability subset of the CSR dataset (with 50 problems in each segment). Column 2 contains the min, max and average number of formulas contained in the problems in that specific segment. (If all problems contain the same number of formulas only the average is shown). Columns 3, 4, and 5 show the percentage of problems for which a solution was found, respectively by EPILOG1 with forward inference enabled, EPILOG1 without forward inference and EPILOG2 (which by default has no forward inference enabled). Column 6 shows the average depth of the answer found by EPILOG2. Column 7 shows the percentage of problems solved by Vampire. All systems, except Vampire 10, have been limited to a timeout of 120 seconds on a Pentium D running at 2.8GHz with 1G of RAM and Linux 2.6.28. (*) Vampire 10 results were obtained with a 10 minutes timeout on a P4 at 2.8GHz with 1G of RAM and Linux 2.6.18. Vampire 8.1 and 9 on segment 5 are unable to complete any problem because they terminate with a segmentation fault.

EPILOG2			
Segment	4 min	8 min	16 min
1	100	100	100
2	92	94	94
3	62	74	86
4	52	52	52
5	12	28	28
Vampire 8.1			
1	100	100	100
2	100	100	100
3	92	92	94
4	30	32	34
5	0	0	0
Vampire 9			
1	100	100	100
2	100	100	100
3	82	82	82
4	32	52	52
5	0	0	0

Table 5.13: To give an idea of the dependency of the results provided in table 5.12 on the time allowed for reasoning, we report here the percentage of problems solved in the 5 segments by EPILOG2 and Vampire with 4, 8 and 16 minutes timeout. On segment 5 both Vampire 8.1 and 9 are unable to complete any problem because they terminate with a segmentation fault.

Segment	10 min
1	100
2	56
3	56
4	60
5	40

Table 5.14: Results obtained by SInE 0.3 on the same set of problems used in table 5.12. The second column shows the percentage of problems solved in that segment when given a 10 minutes timeout. Given that we were unable to run the system on our machine, we report the results contained in the TSTP.

subset of problems used to compile table 5.12 but with timeouts of 4, 8 and 16 minutes. This provides an indications of the effect of the timeout on the number of problems solved by each theorem prover.

Because the subset of problems used in this part of our evaluation belongs to the division LTB in CASC, we report in table 5.14 the results obtained by the winner of that division SInE 0.3¹¹. In this case as well we were unable to run this theorem prover on our machine, therefore we report the results from the TSTP dataset. Also in this case the timeout was 10 minutes and the results were obtained on a P4 at 2.8GHz with 1G of RAM and running Linux 2.6.18. The SInE system is a metareasoner: it first selects a subset of the input axioms and then runs a normal theorem prover (the E theorem prover¹²) on that subset. The subset is selected based on the constants used in the question to be proved.

5.3 Summary

In summary, from the first type of evaluation the main points to highlight are:

- The use of a computational notion of *knowing* to implement introspection

¹¹<http://www.cs.man.ac.uk/~hoderk/sine/>

¹²<http://www4.informatik.tu-muenchen.de/~schulz/WORK/eprover.html>

and autoepistemic inference.

- The use of knowledge closure axioms to explicitly define portions of the agent's knowledge that it considers complete.
- The use of a specialist interface based on explicit axioms that define under which conditions a particular specialist functionality can be used and what it provides.
- The use of quasi-quotation and substitutional quantification to, for example, express schematic meaning postulates, axioms about attitudes, interfacing to specialists, etc.

From the second evaluation, the main highlights are:

- The major improvement in performance obtained by EPILOG2 from EPILOG1 achieved by the reimplementation of all modules and by the introduction of new features like the hierarchical agenda, inference graph optimizations (in particular the various forms of loop and duplication detection and the particular handling of conjunctions) and automatic extraction of hierarchical information (such as transitivity relations).
- Increase in usability of EPILOG given that it achieves the results reported without any special configuration or change across the entire range of problems tested.
- Respectable performance when compared with the state of the art in FOL theorem proving, which in the case of Vampire has a long history of continuous and methodical development and therefore is based on highly refined and optimized C code. This suggests that the greater expressivity of EL is not a serious handicap in its reasoning with syntactically and semantically more restricted formulas.

- Reiteration of the importance of limiting the number of useless¹³ inferences, particularly in large problems. This can be extrapolated from the performance of EPILOG2 and Vampire, which become comparable when the size of the problems grows over a certain threshold, even though Vampire is a couple of order of magnitude faster¹⁴ than EPILOG2 in basic operations.

¹³Useless with respect to finding a solution to the problem under consideration.

¹⁴This is an estimate as we haven't instrumented the source code of Vampire to obtain a more accurate measure.

6 Conclusion and Future Work

Given that self-awareness is one of the most important aspects of human intelligence together with language, this work has tried to combine the two by improving the current state of the art in self-aware and commonsense reasoning in a highly expressive logical language geared towards natural language. This work can be considered as contributing to the long-standing effort to build an intelligent artificial agent with human-like abilities, which is the goal of what's called strong AI.

Because self-awareness is a phenomenon still mostly unexplored and hard to define (see chapter 2), Schubert attempted a classification of various types of awareness an artificial agent may display (see section 2.3). In this classification, explicit self-awareness is the most human-like self-awareness, and is the type of awareness to which we are committed. In particular, the guiding features of explicit self-awareness are: **1)** a complex self-model relating the agent to a substantial world model, **2)** transparency with respect to the agent's internal behavior, and **3)** explicit display of self-awareness focused in particular on natural language. All 3 points impose several requirements on the representation language chosen, in particular it must be capable of expressing the complexity of natural language. The natural choice is Episodic Logic because it was developed for natural language applications, already supports several of the requirements for autoepistemic rea-

soning and representing complex knowledge, and because it already had a reasoner for it: EPILOG.

A preliminary demo involving self-awareness in EPILOG preceded this definition and was implemented by Kaplan (see section 3.4) as a demonstration of his belief specialists. Belief reasoning is in fact relevant to self-awareness because introspection is a particular case of inference about beliefs (i.e. beliefs about oneself).

Using that experience, the definition of explicit self-awareness and our subsequent extension of the COMA demo, we identified several points that needed revision to enable further development towards our final goal of a human-like explicitly self-aware intelligent agent. These observations motivated the redesign of EPILOG, an effort that represents the core of this work. Chapter 4 describes the major modules redesigned during this work.

A fundamental part of any system is its evaluation, for at least two reasons: evaluation of progress toward a defined goal, and formation of a basis for comparison with other systems and future development of the same system. Evaluation for reasoners based on languages other than first-order logic is quite hard for two reasons: lack of standard and publicly available testing sets, and the difficulty in comparing with other systems using different representation languages. Therefore we designed a two-stage evaluation. In the first stage we used questions to test abilities related to self-awareness, such as introspection and self-modelling. For this first stage we used both our own examples and examples from the extensive collection of commonsense test cases contained in ResearchCyc (in that case enabling, as a side effect, some comparisons with Cyc).

The second stage of evaluation was motivated by the desire to facilitate comparison with other inference systems and providing a bottom-line performance measure for future development of EPILOG, in particular with respect to scalability to large problems (crucial to achieve human-like reasoning abilities). For that

we used the widely used TPTP dataset for FOL theorem provers. In particular we selected the subset of problems that form the scalability challenge automatically constructed from the OpenCyc ontology.

In summary, the following are the achievements of our work:

- A new revision of the EPILOG system designed to facilitate and support future development. This was achieved by following good software engineering techniques, by providing a detailed evaluation for comparison (of both performance and functionalities) and by providing several tools to support debugging of some of the complex structures used throughout the system. The re-implementation of the system has also been very useful in improving several aspects of EL, in particular its grammar and the soundness conditions of its inference rules.
- A marked increase in the performance of the EPILOG system and at the same time an improvement in its usability and functionalities (except for the basic probabilistic reasoning, for which we decided to await design of a better foundation instead of replicating EPILOG1's method, already known to have problems). One of the novelties is the automatic creation of type hierarchies, and the exploitation of predicates recognized as being transitive, for accelerating inference. The performance achieved also strengthens the case for the use of highly expressive logics by showing that high expressiveness and good performance are not mutually exclusive goals.
- On the explicit self-awareness front, we feel that we just started tackling some of the issues and that much of the work remains to be done (see the future work section below). The examples used in the first part of the evaluation and the extended COMA demo motivate the need for highly expressive logics that feature syntactic quantification, modifiers, episodic operators and reification operators (including quotation). These are required by an agent

both to express knowledge that refers to its syntax (useful for meaning postulates and introspection) and to express some of the complexity of natural language (like kinds, actions, time relations of complex events, modifications, etc.). The examples also highlight the importance of having a computational notion of knowing (that implements introspection) in conjunction with axioms about the completeness of portions of an agent's own knowledge (self-knowledge) to tackle problems till now unsatisfactorily handled by nonmonotonic techniques (see, for example, our treatment of the questions "Do pigs have wings?" and "did the phone ring?"). The examples also show the extent of the agent's self-awareness, currently based on the use of self-knowledge and introspection. An interesting augmentation of the level of self-awareness displayed by the agent could come from the introduction of the ability to give reasons for failures in answering questions. An example would be to answer the question "Do thrips have wings?" with "I don't know because I'm unfamiliar with thrips." A possible way to start handling this could be by detecting the simplest subgoal(s) that have not yet been answered but that if answered would make the initial question answerable.

The automatic extraction of type knowledge and other forms of hierarchical knowledge, in addition to improving the performance of the theorem prover by providing shortcuts for the general inference engine (without compromising its usability by requiring these special structures to be manually specified), also keeps the knowledge stored within these special structures in a transparent form. In fact, now, to the question "What do you know about flowers?" the system can answer "that they are plants" because that knowledge is explicitly stored in the general knowledge base in addition to having been used to build the type hierarchy.

The example featuring the simple question "How old are you?" highlights the importance of avoiding casual simplifications of commonsense questions

that miss important details, such as the need to specify the desired form of an answer. It also highlights how the complication of expressing and handling pragmatic constraints can be overcome by using specialists and explicit axioms describing their functionalities (the use of axioms as interface to the specialists also avoids the transparency problems that characterized the old interface).

Among the future developments, the following are some of the most important:

- Support for probabilistic reasoning in a better-founded way as compared with the methods employed in EPILOG1.
- Support for generalized quantifiers. This will also require probabilistic reasoning.
- An efficient and scalable storage and access schema.
- Implementation of other specialists, in particular the time specialist.
- Support of inheritance for specialist knowledge, such as type knowledge, in the hierarchy of knowledge bases. Given that most of the specialist knowledge built so far is in the form of graphs, this will require implementing graph inheritance.
- Further improve the level of self-awareness displayed by the agent, for example by demonstrating the ability to give reasons also in the cases where the agent is unable to answer a question.
- Improve the English-to-EL translator to increase its coverage and usability to allow I/O in natural language with EPILOG.
- Further extension of the unification routines to allow for moving quantifiers outward as an additional method for avoiding the generation of free variables that cause some of the possible inferences to be discarded.

- More efficient routines in particular with respect to memory allocation.
- Improvements to the efficiency of the specialist interface by automatically tagging with special information the axioms that define this interface. This special information should highlight which ones are the preconditions for the applicability of a given specialist so that these preconditions can be verified before calling the specialist. Because this is currently not done, it can happen that some of the preconditions are checked twice.
- Further optimization of the handling of the inference graph to improve efficiency, by reducing the repetitions that now occur when several assumptions are made. Each assumption creates a new knowledge base and therefore the reasoning tree stemming from it cannot be shared with the remainder of the tree. However this can be avoided by keeping track of the knowledge bases used by the inferences already made. For example, let's consider an already present subgoal, *sg*, in a particular inference tree and associated with the knowledge base *kb-123*; if a new subgoal, *sg-new*, is added (with the same formula but associated with a different knowledge base, *kb-234*), the two subgoals can be joined and only the missing inferences need to be added to the retrieval agenda of the already present subgoal (i.e. *sg*). These missing inferences are those using the keys of *sg* already used, but this time applied to all the knowledge bases that are in the difference between the inheritance sets of *kb-234* and *kb-123*. The inheritance set of a knowledge base *kb* is the set of knowledge bases including *kb* that are inherited by *kb*.
- Generalization of the second optimization to the QA-framework described on page 110, to remember the keys coming from all the ancestors *maj*s instead of just considering the keys coming from the *maj* used by the parent subgoal.

- Taking advantage of multithreading, given that the future of CPU development seems to lie in multicore designs.

Bibliography

- Abney, Steven. 1989. A computational model of human parsing. *Journal of Psycholinguistic Research*, 18(1).
- Allen, James F. 1981. What's necessary to hide?: modeling action verbs. In *Proceedings of the 19th conference on Association for Computational Linguistics*, pages 77–81. Association for Computational Linguistics, Morristown, NJ, USA.
- Atkinson, R.C. and R. M. Shiffrin. 1968. Human memory: A proposed system and its control processes. In K. W. spence and J. T. Spence, editors, *The Psychology of Learning and Motivation: Advances in Research and Theory*., volume 2, pages 89–195. Academic Press.
- Baars, Bernard J. 1988. *A Cognitive Theory of Consciousness*. Cambridge University Press.
- Baars, Bernard J. 1994. A thoroughly empirical approach to consciousness. *PSYCHE*, 1(6).
- Baars, Bernard J. 1997. In the theater of consciousness. global workspace theory, a rigorous scientific theory of consciousness. *Journal of Consciousness Studies*, 4(4):292–309.
- Baddeley, A.D. and G. Hitch. 1974. Working memory. In G.A. Bower, editor, *In The Psychology of Learning and Motivation*., pages 48–79. Academic Press.

- Baddeley, Alan. 2000. The episodic buffer: a new component of working memory? *Trends in Cognitive Sciences*, 4:417–423.
- Beebe-Center, J. G., M. S. Rogers, and D. N. O’Connell. 1955. Transmission of information about sucrose and saline solutions through the sense of taste. *Journal of Psychology*, 39:157–160.
- Block, Ned. 1995. On a confusion about a function of consciousness. *Behavioral and Brain Sciences*, 18:227–247.
- Block, Ned. 2002. Concepts of consciousness. In David J. Chalmers, editor, *Philosophy of Mind: Classical and Contemporary Readings*, chapter 24, pages 206–218. Oxford University Press.
- Cercone, N. and L. Schubert. 1975. Toward a state-based conceptual representation. In *Proceeding of the 4rd International Joint Conference on AI*, pages 83–90.
- Chalmers, David J. 1995. Facing up to the problem of consciousness. *Journal of Consciousness Studies*, 2(3):200–219.
- Dennett, Daniel. 1988. Quining qualia. In A. Marcel and E. Bisiach, editors, *Consciousness in Modern Science*. Oxford University Press.
- Endel Tulving, Fergus I. M. Craik, editor. 2000. *The Oxford Handbook of Memory*. Oxford University Press.
- Evans, James R. and Andrew Abarbanel, editors. 1999. *Introduction to Quantitative EEG and Neurofeedback*. Academic Press.
- Fikes, R. and N. Nilsson. 1971. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208.

- Gardiner, John M. 2001. Episodic memory and autonoetic consciousness: a first-person approach. *Philosophical Transactions of the Royal Society of London B*, 356:1351–1361.
- Garner, W. R. 1953. An informational analysis of absolute judgments of loudness. *Journal of Experimental Psychology*, 46:373–380.
- Hake, H. W. and W. R. Garner. 1951. The effect of presenting various numbers of discrete steps on scale reading accuracy. *Journal of Experimental Psychology*, 42:358–366.
- Hayes, J. R. M. 1952. Memory span for several vocabularies as a function of vocabulary size. Technical report, Acoustic Laboratory, MIT.
- Hayes-Roth, Barbara. 1985. A blackboard architecture for control. *Artif. Intell.*, 26(3):251–321.
- Hwang, C.H. and L.K. Schubert. 1993. Episodic logic: A situational logic for natural language processing. In Y. Katagiri P. Aczel, D. Israel and S. Peters, editors, *Situation Theory and its Applications*, volume 3, pages 303–338. Center for the Study of Language and Information, Stanford, CA.
- Izawa, Chizuko, editor. 1999. *On Human Memory: Evolution, Progress, and Reflections on the 30th anniversary of the Atkinson-Shiffrin Model*. Lawrence Erlbaum Associates.
- Jackson, Frank. 1986. What Mary didn't know. *The Journal of Philosophy*, 83(5):291–295.
- Jackson, Frank. 2002. Epiphenomenal qualia. In David J. Chalmers, editor, *Philosophy of Mind: Classical and Contemporary Readings*, chapter 28, pages 273–280. Oxford University Press.

- Jacoby, Larry L. 1982. Knowing and remembering: Some parallels in the behavior of korsakoff patients and normals. In Cermak Laird S, editor, *Human Memory and Amnesia*, pages 97–122. Lawrence Erlbaum Associates.
- Kaplan, Aaron. 2000. *A Computational Model of Belief*. Ph.D. thesis, University of Rochester.
- Konieczny, Lars. 1996. *Human Sentence Processing: A Semantics-Oriented Parsing Approach*. Ph.D. thesis, University of Freiburg.
- Kuipers, Benjamin. 2005. Consciousness: Drinking from the firehose of experience. In (Velooso and Kambhampati, 2005), pages 1298–1305.
- Lakemeyer, Gerhard and Hector J. Levesque. 2005. Only-knowing: Taking it beyond autoepistemic reasoning. In (Velooso and Kambhampati, 2005), pages 633–638.
- Levinson, Stephen C. 2001. *International Encyclopedia of Social and Behavioral Sciences.*, volume 17, pages 11948–11954. Elsevier Science.
- Liu, Daphne and Lenhart Schubert. 2009. Incorporating planning and reasoning into a self-motivated, communicative agent. In *Second Conference on Artificial General Intelligence (AGI 2009)*.
- Mandler, George. 2003. Consciousness: Respectable, useful, and probably necessary. In Bernard J. Baars, William P. Banks, and James B. Newman, editors, *Essential Sources in the Scientific Study of Consciousness*, chapter 2, pages 15–33. MIT Press.
- McCarthy, John. 1980. Circumscription—a form of non-monotonic reasoning. *Artificial Intelligence*, 13:27–39.
- McCarthy, John. 1993. Notes on formalizing context. In *In Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*.

- McCarthy, John. 1995. Making robots conscious of their mental states. In *Machine Intelligence 15*, pages 3–17.
- McDermott, Drew. 2001. *Mind and Mechanism*. The MIT Press.
- Miller, George A. 2003. The magical number seven, plus or minus two: Some limits on our capacity for processing information. In Bernard J. Baars, aWilliam P. Banks, and James B. Newman, editors, *Essential Sources in the SCientific Study of Consciousness*, chapter 23, pages 357–372. MIT Press.
- Miller, S. A. and L. K. Schubert. 1988. Using specialists to accelerate general reasoning. In *Proc. of AAAI-88*, pages 161–165. St. Paul, MN.
- Minsky, Marvin. 1988. *The Society of Mind*. Simon & Schuster.
- Mitchell, Melanie. 2005. Self-awareness and control in decentralized systems. In *Metacognition in Computation*, pages 80–85.
- Moore, Robert C. 1985. Semantical considerations on nonmonotonic logic. *Artif. Intell.*, 25(1):75–94.
- Morbini, Fabrizio and Lenhart K. Schubert. 2007. Towards realistic autocognitive inference. In *Logical Formalizations of Commonsense Reasoning*, pages 114–118.
- Nagel, Thomas. 1974. What is it like to be a bat? *Philosophical Review*, 83(October):435–50.
- Newell, Allen. 1990. *Unified Theories of Cognition*. William James lectures; 1987. Harvard University Press.
- Ogden, Charles Kay. 1932. *Basic English, a general introduction with rules and grammar*. Paul, Trench, Trubner & co., ltd.

- Pani, J. R. 1982. A functionalist approach to mental imagery. In *23rd Annual Meeting of the Psychonomic Society*. Baltimore, MD.
- Pollack, I. 1952. The information of elementary auditory displays. *Acoust. Soc. Amer.*, 24:745–749.
- Pollack, I. 1953. The information of elementary auditory displays ii. *Acoust. Soc. Amer.*, 25:765–769.
- Ramachandran, Deepak, Pace Reagan, and Keith Goolsbey. 2005. First-orderized researchcyc: Expressivity and efficiency in a common-sense ontology.
- Reiter, R. 1980. A logic for default reasoning. *Artificial Intelligence*, 13:81–132.
- Schaeffer, S.A., C.H. Hwang, J. de Haan, and L.K. Schubert. 1993. EPILOG, the computational system for episodic logic: User’s guide. Technical report, Dept. of Computing Science, Univ. of Alberta.
- Schubert, Lenhart K. 2000. The situations we talk about. In *Logic-Based Artificial Intelligence*, pages 407–439. Kluwer Academic Publishers.
- Schubert, Lenhart K. 2005. Some knowledge representation and reasoning requirements for self-awareness. In *Metacognition in Computation*, pages 106–113.
- Schubert, Lenhart K., Stephanie Schaeffer, Chung Hee Hwang, and Johannes de Haan. 1993. *EPILOG: The Computational System for Episodic Logic. USER GUIDE*.
- Schubert, L.K. and C.H. Hwang. 2000. Episodic Logic meets Little Red Riding Hood: A comprehensive, natural representation for language understanding. In L. Iwanska and S.C. Shapiro, editors, *Natural Language Processing and Knowledge Representation: Language for Knowledge and Knowledge for Language*, pages 111–174. MIT/AAAI Press, Menlo Park, CA.

- Searle, John R. 2004. *Mind : A Brief Introduction*. Oxford University Press.
- Shapiro, S. C. 1979. Generalized augmented transition network grammars for generation from semantic networks. In *Proceedings of the Seventeenth Meeting of the Association for Computational Linguistics*, pages 25–29.
- Shapiro, Stuart C. 1989. The cassie projects: An approach to natural language competence. In João P. Martins and Ernesto M. Morgado, editors, *EPIA*, volume 390 of *Lecture Notes in Computer Science*, pages 362–380. Springer.
- Shapiro, Stuart C. 1998. Embodied Cassie. In *Cognitive Robotics: Papers from the 1998 AAAI Fall Symposium, Technical Report FS-98-02*, pages 136–143. AAAI Press, Menlo Park, California.
- Shapiro, Stuart C. 2000. SNePS: A logic for natural language understanding and commonsense reasoning. In Lucja Iwańska and Stuart C. Shapiro, editors, *Natural Language Processing and Knowledge Representation: Language for Knowledge and Knowledge for Language*, pages 175–195. AAAI Press/The MIT Press, Menlo Park, CA.
- Stan, F., K. Arpad, and M. Lee. 1998. Ida: A cognitive agent architecture.
- Stoness, S. C. 2004. A general architecture for incremental parsing. Technical Report TR 838, University of Rochester.
- Stoness, Scott, Joel Tetreault, and James Allen. 2004. Incremental parsing with reference interaction. In *ACL2004, Workshop on Incremental Parsing: Bringing Engineering and Cognition Together*.
- Tulving, Endel. 1972. Episodic and semantic memory. In E. Tulving and W. Donaldson, editors, *Organization of Memory.*, pages 381–403. Academic Press.
- Tulving, Endel. 2002. Episodic memory: from mind to brain. *annu. Rev. Psychol.*, 53:1–25.

- Veloso, Manuela M. and Subbarao Kambhampati, editors. 2005. *Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, July 9-13, 2005, Pittsburgh, Pennsylvania, USA*. AAAI Press AAAI Press / The MIT Press.
- Vere, Steven. 1983. Planning in time: Windows and durations for activities and goals. *IEEE Transactions on Pattern Analysis and Machine Intelligence*., 5(3):246–267.
- Vere, Steven and Timothy Bickmore. 1990. A basic agent. *Comput. Intell.*, 6(1):41–60.
- Winograd, Terry. 1972. Understanding natural language. *Cognitive Psychology*, 3(1):1–191.
- Winograd, Terry. 1990. Thinking machines: Can there be? are we? In Derek Partridge and Yorick Wilkis, editors, *The Foundations of Artificial Intelligence: A Sourcebook*, pages 167–189. Cambridge University Press.