The Metacognitive Loop: An Architecture for Building Robust Intelligent Systems

H. Haidarian, W. Dinalankara, S. Fults, S. Wilson, Don Perlis
University of Maryland, College Park, MD
M. Schmill, T. Oates
University of Maryland Baltimore County, Baltimore, MD
D. P. Josyula
Bowie State University, Bowie, MD
M. L. Anderson
Franklin and Marshall College, Lancaster, PA

Abstract: What commonsense knowledge do intelligent systems need, in order to recover from failures or deal with unexpected situations? It is impractical to represent predetermined solutions to deal with every unanticipated situation or provide predetermined fixes for all the different ways in which systems may fail. We contend that intelligent systems require only a finite set of anomaly-handling strategies to muddle through anomalous situations. We describe a generalized metacognition module that implements such a set of anomaly-handling strategies and that in principle can be attached to any host system to improve the robustness of that system. Several implemented studies are reported, that support our contention.

1. Introduction

"When we are confronted with unexpected situations, we deal with them by falling back on our general knowledge or making analogies to other things we know. When software applications fail, on the other hand, they often do so in brittle and unfriendly ways. The sheer amount of commonsense knowledge one would need to represent makes it challenging to acquire, to represent, to reason efficiently with, and to harness in applications. This is, ultimately, the bottleneck to strong AI." Or at any rate, so says the text accompanying the workshop call.

Our view of the brittleness problem is as follows: it is not an abundance of special knowledge about the world that is needed. For no matter how abundant that may be, and no matter how efficient a system is in processing it, the world will nevertheless provide plenty of anomalies that fall outside the applicability of any given abundance of such knowledge. Rather, we contend, what is needed is a set of general-purpose anomaly-handling strategies. We contend that this is what humans have, that this is what lets us "muddle through" a vast variety of unanticipated situations.

While it is true that we (humans) do have lots of special methods for dealing with special problems, those methods tend to be picked up along the way as we encounter anomalies of a given kind. We do not – and we cannot – come equipped with such methods for all conceivable (or inconceivable) anomalies. In fact, our impressive repertoire of such methods is a work in progress, getting better and better as we go through life.

The question then should be: what underlying mechanisms make this possible for humans? How is it that when faced with an unanticipated and unfamiliar situation, rather than crashing and burning, or cluelessly blundering ahead, we typically deal with it quite effectively even in the absence of any prior methods for that specific type of situation? Furthermore, in the process, we add to our anomaly-handling repertoire so that we handle situations of that sort more quickly in the future.

To be more specific, humans have a specialized "reasoned anomaly-handling" capability, that is largely domain-independent, that involves only modest background knowledge and computation, and that thus is implementable in automated systems. We call it *generalized metacognition*. It consists of three steps: (i) monitor expectations in order to note any anomaly that might arise, (ii) assess it in terms of available response options, and (iii) guide any chosen response into place (and monitor the progress of that response for further anomalies). This requires, of course, expectations as to how things "ought" to be, responses that apply across the board to almost any type of anomaly, and the ability to re-configure expectations in light of how things go.

This may seem like a tall order; indeed it may seem that all this does is push the problem into another form ("expectations and responses that apply across the board" sounds like "falling back on our general knowledge or making analogies to other things we know" in the quoted piece, at the beginning of this section). But here is the main point: there are a few powerful general-purpose anomaly-handling strategies that apply pretty much across the board. Examples include: give up on the task, ask for help, try it again, ignore, make small random changes, confirm data, check equipment, shut down, and initiate training. Our claim is that: (i) humans do this, do it well, do it all the time, and would be utter failures without this; and (ii) this smallish set of general strategies is not at all mysterious, and in fact can be characterized algorithmically and built into systems, endowing them with the robustness and flexibility to adjust to new situations in real time. We call an algorithmic version of *note-assess-guide* the Metacognitive Loop (MCL).

We have been engaged in testing this idea over the past several years. In what follows, we first outline ways in which such generalized metacognition lends itself to various issues in commonsense reasoning. We then describe a particular MCL *architecture* and associate engineering methodology for deploying generalized metacognition toward the implementation of more robust AI systems. Our experience to date and several pilot studies show that an AI system that is built based on the MCL architecture and has a generalized metacognition module, i.e. a special-purpose anomaly-processor, can perform in a more robust fashion. In addition, by following the MCL architecture, existing AI systems can be coupled with a generalized metacognition module to improve their robustness.

2. Commonsense, Active Logic and Metacognition

Much work in commonsense reasoning aims to “get it right” – to set down once and for all a correct (enough) and complete (enough) description of the dynamic everyday world to afford correct (enough) conclusions about that world, of the sort that would be useful to an intelligent agent negotiating within that world. However, work to date, despite very impressive gains, continues to face serious difficulties. Among these difficulties are the following:

- Most of this work involves formalisms that specify conclusions *for* an agent, but not how an agent can *make* appropriate inferences to that end; that is to say these formalisms are about agents, but not for use by the agents
- The same formalisms do not come to terms with evolving time; what is true one moment might not be so at the next, and so what an agent should conclude at one moment might not be appropriate at the next
- These formalisms are useless unless all available beliefs are consistent – or at least all but one new incoming belief
- These formalisms are designed with a fixed language and a fixed intended semantics
- Finally, the everyday world is far too complex to capture with a reasonably sized set of axioms; there are too many exceptions, indeed too many unpredictable exceptions.

We advocate a very different approach to commonsense reasoning – an agent logic suited to the MCL jobs of *note-assess-guide*, that (not coincidentally) addresses the above difficulties. To that end we employ the active-logic engine (Elgot-Drapkin 1988, Elgot-Drapkin and Perlis 1990, Elgot-Drapkin et al. 1993, Purang 2001), which tracks inference over evolving (real) time.

In active logic, at a given time only those inferences that have actually been carried out so far can affect the present state of the agent’s knowledge. As a result, even if directly contradictory formulae P and ~P are present in the knowledgebase at time t, they do not necessarily have to be used to derive anything else in that step. By adding a rule to the logic to recognize these conflicts, contradictions can be recognized when they are about to be used for reasoning in the current step, and handled appropriately. (Handling the contradiction can be done in multiple ways, such as ignoring the contradicting formulae, or actively trying to repair it. This can be seen as a metacognitive function of the reasoning agent, since it enables the agent to reason about its own reasoning process.) Active logic systems have been developed which can reason in the presence of, and in some cases automatically resolve, contradictory information (Elgot-Drapkin and Perlis 1990, Elgot-Drapkin et al. 1993, Purang 2001), and have also been applied to such related areas as deadline-coupled planning (Miller and Perlis 1993).

We should emphasize that the use of active logic to implement MCL is *not* aimed at solving puzzles (three wise men, mutilated checkerboard, missionaries and cannibals, etc.). In our view, these problems have sidetracked the essential aspects of commonsense reasoning. Indeed, humans find these challenging, yet difficulties with such puzzles does not hinder human negotiations with the everyday world.

3. Examples of Robust Commonsense Reasoning

We will indicate how MCL can address various issues in commonsense reasoning; this will mostly be very sketchy, but in the next section, we will discuss a few empirical results.

Nonmonotonic reasoning in general tends to involve determining that there are no known conditions to preclude drawing an expected conclusion, and then actually so concluding. The tough part is knowing one does not know (precluding conditions). And in general this is undecidable, although in special cases this can be done. But a real agent must make decisions in real time, without the luxury of waiting until all the results of extended deliberation are in. The MCL approach simply uses whatever conditions are already known (not consequences of what is known) and if later a precluding consequence surfaces, the agent can always retract any conclusion based on the absence of such. Thus MCL straddles both nonmonotonicity and belief revision, by combining them into one ongoing process of drawing conclusions and then revising them as needed.

The same sanguinity with regard to not-yet-known consequences allows MCL to proceed in the presence of contradictory beliefs – until such time as the contradiction is discovered; and then a repair process can initiate. The “best” repair for a contradiction is hard to characterize in general, and surely is very context dependent. But at the very least, noticing a contradiction should make the agent wary of using the contradictands until a resolution is available. Notice that the use of evolving time is key to this approach: the agent can change its mind in light of further information.

A telling example is that of going to lunch. The agent has made a lunch appointment for noon at a nearby restaurant. It must then reason – say at 11:30am – that at 11:40 it will be time to start walking; and at 11:40 it must then realize that this is “now” that very time and initiate the needed action (walk). While this seems rather trivial, it is not easy to capture in the form of a reasoning process, for that process must be responsive to changing time *during* its reasoning. We have mentioned three key elements of MCL: note, assess, guide. But in fact two more are essential as well: in order to note an exception to an expectation, an agent needs expectations in the first place; and since the world is far too complex to have a full and accurate set of expectations, then these must in general be learned or modified on the fly (we give an example of this below – the Bolo domain). Once there are expectations, noting a violation can be as simple as detecting a contradiction between an expectation E and an observation ~E. Then one or more available anomaly-handling strategies is selected (possibly well-chosen, or not – this too is refined by learning); and finally any such chosen strategy is initiated and monitored.

DARPA and NASA have unintentionally provided excellent examples of the potential power of MCL, in the following “rover” and “satellite” situations: (i) a DARPA

Grand Challenge robot bumped into a chain-link fence that it could not see and then simply stayed there, spinning its wheels futilely; and (ii) a NASA satellite turned itself to look in a certain direction as instructed, but then was unable to receive further instructions (even the instruction to turn back), since its antenna was no longer in a direct line of sight. In each of these cases, a modest amount of self-modeling (I should be moving forward, or I should be receiving more instructions) and self-observation (I am not moving forward, or I am no longer receiving instructions) would have alerted the systems that something was amiss; and even a modest amount of self-repair (try small random motions) would have been better than staying stuck.

Thus in our view, much of what is loosely referred to as commonsense is captured in the aphorism: "fool me once, shame on you; fool me twice, shame on me." The world will take us unawares, perhaps badly so. But then we notice, and most of the time we are not so quickly surprised in that same way again. That is, we have expectations (beliefs about various patterns in how things behave), and when such an expectation is violated, we are able to use the information to our advantage. Put differently, we learn from mistakes; yet this considerably oversimplifies the picture.

4. Pilot Studies

Reasoning about changes in belief over time plays an important role in building a sophisticated reasoning system. (Miller 1993) demonstrates how active logic can be used to reason about such changes; an example is the Two Johns problem, where we imagine that an agent is talking to a user about a third person, whom the agent initially comes to identify as his unmarried friend John, by virtue of matching John to the user's description of the person (note that the user has not revealed the name of the person). Then, the agent is led to believe that this person's leg is broken and that his wife has to do everything for him. This confuses the agent, since John is not married. Subsequently, the user starts employing the name 'John' to refer to this person. To resolve this confusion, the agent needs to relax its usage so that the word 'John' is not firmly tied to the person identified at the beginning by the agent. Thus, the agent must realize that the word 'John' is now ambiguous and that there are two entities with the same label. While a certain amount of specialized knowledge representation is needed to carry this out, the methods are quite general and broadly applicable in dialog and indeed any circumstance in which it is important to consider errors or misunderstandings of symbolic information.

In fact, our major application area of the MCL architecture to date has been *natural language human-computer interaction*. We used an existing implementation of a natural-language train-control simulation (Allen et al. 1995), as a host. Assume that a user says "Send the Boston train to New York" and then, after the system chooses a train and moves it, the user says "No, send the Boston train to New York." Such an exchange might occur if there is more than one train at the Boston station, and the system chose a train other than the one the user intended. In the original implementation, the dialog system responds to this apparently contradictory sequence of commands by sending the very same train. However, in our MCL-enhanced version (Traum et al. 1999), the system notes a contradiction, since the user says "Do X. No, do X." Then, the system assesses the problem, which could possibly be a mistake in its choice of referent for "the Boston train." Finally, the system attempts to repair the problem by executing the command with a different choice of referent. Thus, the enhanced system chooses a different train the second time around, and if there are no other trains in Boston, it will ask the user to specify the train by name.

In more recent years, we have extended our dialog system to assess and resolve a broader class of dialog issues. The new system deals with anomalies by setting and monitoring time-, feedback-, and content-related expectations (Josyula 2005). For instance, if the user does not respond to a system query within an expected time limit, then the system recognizes that there is a problem and repeats the query. However, continuous repetition of the query and the user not responding indicates a continuing problem (recall that the generalized metacognition module monitors the progress of corrective actions), and causes a re-evaluation of the other response options. In this case, the system would ask the user whether everything is OK. If there is still no response from the user, the system may drop its expectation about getting a response from the user in the near future.

In another scenario, if the user says "Send the Metro to Boston" and the system doesn't know the word "Metro," the expectation that it should find the word in its dictionary is violated. This triggers the generalized metacognition module to guide the system into asking the user for specific help. Asking for help is a very reasonable way of solving a problem, and one that humans use quite often when confronted with an anomaly. The system would say "I don't know the word 'Metro.' What does 'Metro' mean?" Once the user tells the system that 'Metro' is another word for 'Metroliner' (a train in the application domain, which the system does know), the system is able to correctly implement the user's request.

In a very different pilot study (Anderson et al. 2006), we built *standard reinforcement learners* using Q-learning, SARSA and Prioritized Sweeping. These were placed in an 8x8 world with two rewards – one in square (1, 1) and the other in square (8, 8). The learner was allowed to take 10000 actions in this initial world, which was enough in all cases to establish a very good albeit non-optimal policy. In turn 10001, the values of the rewards were abruptly changed. Unsurprisingly, we found that the perturbation tolerance (i.e., the post-perturbation performance) of standard reinforcement learners was negatively correlated to the degree of the perturbation.

Our experiments showed that even a very simple generalized metacognition module that did no more than generate and monitor expectations for performance and re-learn its entire policy (whenever its expectations were violated a fixed number of times), significantly outperformed standard reinforcement learning, in the case of high-degree perturbations. Also, a somewhat smarter module that in light of its assessment of the anomalies, chose between doing nothing, making an on-line adjustment to learning parameters, or re-learning its policy, out-performed standard reinforcement learners overall, despite some under-performance in response to mid-range perturbations.

Our last pilot study involves Bolo, a *multiplayer tank game* that can be played by either humans or other Bolo-playing programs. Many programs that play Bolo perform quite poorly and are easily fooled when unexpected complications arise. The game takes place in a world that contains various terrain types (roads, swamps, walls, etc.), refueling bases, and pillboxes. There are neutral pillboxes that fire on all tanks, friendly pillboxes that fire only on other players' tanks, and dead pillboxes that pose no threat which can be captured to make them friendly. An important strategy in Bolo is to capture pillboxes, make them friendly, and then use them either offensively or defensively.

We used a Bolo player controlled by a hierarchical task network (HTN) planner with primitive actions that ground out in controllers. The main expectation maintained by the generalized metacognition module is that the tank it controls will not be destroyed. A suite of actions were maintained, including means-ends analysis and operator refinement (Gil 1994, Wang 1995). The initial HTN did not have a plan to deal with hostile pillboxes, which fire on the tank, and so it was destroyed in its first such encounter. However, our MCL-enhanced Bolo player (Anderson et al. 2008) was able to discover that firing on pillboxes offered a solution to the problem, even though it had no previous knowledge of the effect of that particular action.

In analyzing its past experience, the generalized metacognition module located the salient differences in the conditions under which it succeeded in taking pillboxes, and those in which it failed. Since only pillboxes with intact armor destroyed the tank, the next step was to see if it had any actions that could reduce the armor of a pillbox. If it had known about an action that would do that, it would have tried the action immediately. In the case we tested, it had no such knowledge. Thus, it used a heuristic to rank all its actions according to how likely they were to have the desired effect, and then tested them until it found one that worked. This is an example of how the MCL architecture turns failure into opportunity: in each case the system learned more about what effects its actions did and did not have, and in a way organized to support its ongoing mission.

5. A Metacognitive Loop Architecture

An intelligent system based on the Metacognitive Loop (MCL) architecture, consists of two components: (i) a host, and (ii) a generalized metacognition module. The generalized metacognition module is attached to a given host, to monitor and control the performance of the host. It endows the host with metacognitive skills, providing the system (as a whole) with the ability to reason about itself, i.e., its knowledge, its observations and its actions. The generalized metacognition module acts as a *commonsense* reasoning engine, whose purpose is to: (i) note when host behavior diverges from host expectations, (ii) assess the anomaly and the options the host has for dealing with the difficulty, and (iii) guide the host toward putting one or more options into action.

A schematic overview of such an intelligent system, designed based on the MCL architecture, is shown in Figure 1. The figure depicts the host (shaded in yellow), and the generalized metacognition module (MCL, shaded in blue). MCL interfaces with the host through a set of inputs, consisting of environmental and operational *expectations* (monitoring information); and a set of outputs, consisting of *corrections* (corrective actions) that the host can perform. The expectations are initially provided by the host's designer. During operation, the system can also adjust (or learn new) expectations based on its ongoing experience.

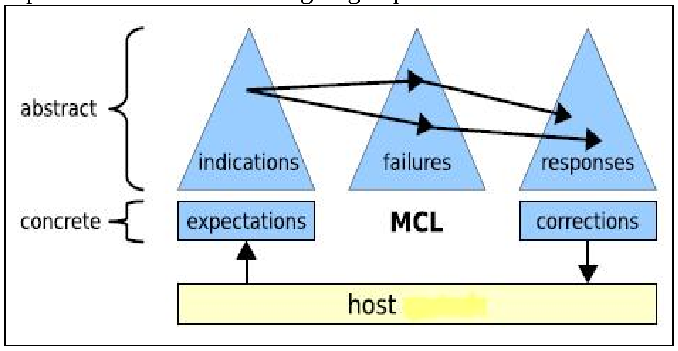

Fig. 1: The overview of an intelligent system, designed based on the MCL architecture. The host is shaded in yellow, and the generalized metacognition module is shaded in blue.

What makes an MCL-based intelligent system more robust and powerful is the emphasis on self-knowledge (expectations, correction, etc.), as opposed to world-knowledge. We have found that although specific anomalies differ in terms of number and detailed characteristics between domains, they fall into a relatively small number of general types. We capture these types and similarities using ontologies. The generalized metacognition module implements three special sets of ontologies: an *indications* ontology for anomaly types, a *failures* ontology for assessment, and a *responses* ontology for repairs, as shown in Figure 1. In essence, the role of the generalized metacognition module is to *notice* anomalies (using indications), *assess* their importance and cause (using failures), and *guide* a repair into place (using responses).

The core of each ontology is currently implemented as a Bayesian network. The core nodes of each ontology are concepts; which represent abstract and domain-independent anomalies, and the responses to those anomalies. These nodes are linked within each ontology, to express the relationships between the concepts. The nodes are also linked between ontologies, allowing the generalized metacognition module to employ a number of Bayesian algorithms for reasoning over ontologies.

The generalized metacognition module is linked to the host through two interfaces, as shown in Figure 1. At the input interface, expectations are directly linked to the indications ontology through its fringe nodes. Fringe nodes are the nodes at the bottom of the indications and response ontologies. They represent concrete, domain-specific information about the anomaly or correction. At the output interface, the set of possible corrections that the host can employ are linked to the response ontology through its fringe nodes.

When an expectation is violated, the generalized metacognition module uses the features of the violation and the current host sensor and effector states, to produce an "initial activation" in the indication fringe nodes. Those indications that are observed directly during the violation are taken as evidence, and the Bayesian update algorithm is used to propagate that evidence through the ontology so that inference can occur at a more abstract level. The fringe nodes of the indications ontology can encode information such as what type of sensor is being monitored (internal state, time, or reward). The core nodes synthesize the information that is provided by the fringe nodes and translate it into abstract, specific indications such as "reward-not-received."

Once the *note* phase is complete, the generalized metacognition module moves to the *assess* stage, in which indications are used to hypothesize the cause of an expectation violation. Probabilities of nodes in the failures ontology are updated based on values propagated through the indication ontology, through a particular kind of link called a *diagnostic link*. These links express which classes of failures are plausible, given the active indication events.

Links emanating from likely failure nodes allow the generalized metacognition module to move into the *guide* phase. In the guide phase, the Bayesian update occurs over *prescriptive links* between suspected failures that are potentially useful responses in the response ontology. Responses can be evaluated according to their cost and inferred probability of correcting the failure. Here, the system designer can specify concrete ways in which abstract responses can be mapped to domain-specific ones.

Once the generalized metacognition module has arrived at a specific suggestion in the *guide* phase, the host can implement that response. After an appropriate response has been chosen, the state of the three ontologies is stored, while the corrective action is performed. The generalized metacognition module then waits to receive feedback on the corrective action. If no new expectation violations are received, then the change that was chosen during the repair is made permanent, and the expectation violation is considered to be resolved. If the violation persists, or a new one occurs, then the generalized metacognition module adds evidence that the response does not work to the Bayesian networks, re-executes the update algorithm, and revisits its options for recovery.

It is worth explaining why the generalized metacognition module does not directly map from indications to responses. The failure ontology enables the module to handle the ambiguous nature of indications. In many cases, a single indication might suggest several potential failures. Similarly, a single failure might only be suspected when several indications are present at the same time. The mapping between indications to failures, then, might be one-to-many or many-to-one. This subtlety is lost without all three ontologies.

Note that nowadays, intelligent systems in different domains are built based on various existing architectures, e.g. BDI (Rao et al. 1995). One advantage of the modular nature of the MCL architecture is its compatibility with existing agent architectures. In other words, the MCL architecture enables system designers to use this architecture, regardless of the architecture of the host. The MCL architecture can be used in the following two scenarios. (1) If an implementation of the host does not exist, and is being built from scratch, the designer can keep in mind the MCL architecture and the appropriate expectations and violation responses, while designing the host (based on any architecture). Then, the host will more easily connect to a generalized metacognition (MCL) module, using the expectations and corrections as an interface. (2) If an implementation of the host already exists, the designer need only extract the necessary expectations and corrections from the host (this generally does not require any extensive modification of the host's implementation). Then again, the host is easily connected to the generalized metacognition module, using the extracted expectations and corrections.

Preliminary results from our pilot studies demonstrate that:

1) The MCL architecture provides a general software model, which can be used for building robust intelligent systems in various domains (domain generality).
2) The generalized metacognition module of the MCL architecture provides a powerful mechanism for representing and handling various types of anomalies that a system may encounter (anomaly generality).

These two issues are somewhat related, but designate different benefits, so we list them separately.

The domain generality of the MCL architecture entails that by following the MCL architecture, a system designer is able to easily and "out-of-the-box" couple a generalized metacognition module with any of a wide variety of AI hosts. In other words, the host can be from any domain, e.g. planner, mobile robot, dialog agent, etc. The connection between the host and the MCL module is basically "plug-and-play." The system designer would only need to know the expectations (monitoring information) and the corrections (corrective actions), for the host.

The anomaly generality of the generalized metacognition module entails that the module contains a sufficiently high-level typology of anomalies that virtually all specific anomalies would fall into one or another type. This in turn empowers the system to deal flexibly with whatever comes its way. The high-level typology of anomalies is represented using the ontologies, within the module.

Crucial to our approach is that the generalized metacognition module does *not* monitor and respond to specific expectation failures based on intricate knowledge of how the world works; that would require building pre-determined fixes for anticipated failures, which is exactly what we want to enable designers to avoid. Instead, we have observed that, at some useful level of abstraction, the various ways in which systems might fail form a finite core of modest size. Therefore, we can develop abstract ontologies of possible failures, their causes and solutions. These ontologies are general enough to accommodate nearly every anomaly that a system might face.

6. Related Work and Discussion

Fault Detection, Isolation, and Recovery (FDIR) is an approach that on the outset is similar to the MCL architecture in its phases of detection, isolation, and recovery (Stroulia and Goel 1997). However unlike MCL, FDIR takes a model-based or expert-systems approach, which significantly limits the applicability of the technique to different problems and domains. The fixes employed in the FDIR approach tend to be quite specific, unlike the very general approach of MCL. The reason why the MCL architecture does not suffer from the same setback is the possibility of novel discoveries in the MCL approach; this in turn is a result of our view that to increase autonomy we need to provide greater freedom in action to the system and allow it to make its own discoveries using the tools we've provided.

Certain reinforcement learning based techniques have been used for acting in uncertain situations (Sutton and Barto 1995, Bertsekas and Tsitsiklis 1996). While it may seem at first glance that the goal of such systems is similar to that of the MCL architecture, these techniques are aimed solely at action selection, rather than monitoring one's own performance and reasoning about it. These techniques require continuous exploration (i.e. behavior that is different from what is presently known to be optimal), as opposed to an MCL-based system, which can continue its usual performance until something goes awry.

7. Conclusion

We have presented an approach for building robust intelligent systems, along with several examples and implemented pilot studies. Our MCL approach is not based on representing vast amounts of intricate knowledge about various anomalies and methods to fix them. Instead, our approach is based on representing a finite set of general causes of, types of and responses for system failures at a high level of abstraction, using three special ontologies, namely indications, failures and responses. These ontologies create the generalized metacognition module, which can be coupled to any host system by specifying the set of *observables* and *actions* of the host system.

It is important to emphasize that the generalized metacognition module is not a kind of magic-bullet intelligence that figures out and then performs perfect actions for every situation. Rather, the MCL architecture enables designers to build systems that can step back and assess a difficulty, when it arises. The system could quite possibly decide that it does not have the wherewithal to resolve the difficulty, and so avail itself of options such as asking for help, giving up, using trial-and-error, web-searching for relevant information, or deciding that it needs to initiate training for that difficulty (if learning capacity exists). This kind of commonsense approach to handling anomalies serves humans very well, and should do the same for automated systems. For example, a stuck robot, instead of literally spinning its wheels, should at least stop needless waste of effort, and at best should have capabilities to successfully negotiate a problem.

References

Allen, J.F., et al., The TRAINS project: a case study in building a conversational planning agent. Journal of Experimental and Theoretical AI, 1995.

Anderson, M., Oates, T., Chong, W., Perlis, D., The Metacognitive Loop I: Enhancing Reinforcement Learning with Metacognitive Monitoring and Control for Improved Perturbation Tolerance. Journal of Experimental and Theoretical Artificial Intelligence, 18(3), pp. 387-411, 2006.

Anderson, M., Fults, S., Josyula, D., Oates, T., Perlis, D., Schmill, M., Wilson, S., and Wright, D. A Self-Help Guide for Autonomous Systems. AI Magazine, Volume 29 (2), 2008

Bertsekas, D.P., Tsitsiklis, J.N., Neuro-Dynamic Programming, Nashua, NH: Athena Scientific, 1996.

Elgot-Drapkin, J., Kraus, S., Miller, M., Nirkhe, M., Perlis, D., Active logics: A unified formal approach to episodic reasoning, Technical Report CS-TR 4072, University of Maryland, College Park, 1993.

Elgot-Drapkin, J., Perlis, D., Reasoning situated in time I: Basic concepts, J. Exp. Theor. Artif. Int., 2, pp. 75–98, 1990.

Elgot-Drapkin, J., Step-logic: Reasoning Situated in Time. PhD thesis, Department of Computer Science, University of Maryland, College Park, 1988.

Gil, Y., Learning by Experimentation: Incremental Refinement of Incomplete Planning Domains. In Proceedings of the Eleventh International Conference on Machine Learning, 87–95, 1994. San Francisco: Morgan Kaufmann Publishers.

Josyula, D., A Unified Theory of Acting and Agency for a Universal Interfacing Agent. PhD thesis, Department of Computer Science, University of Maryland, College Park, 2005.

Miller, M., Perlis, D., Presentations and this and that: Logic in action, in Proceedings of the 15th Annual Conference of the Cognitive Science Society, 1993.

Miller, M.J., A View of One's Past and Other Aspects of Reasoned Change in Belief. Doctoral Thesis, University of Maryland at College Park. 1993.

Perlis, D., Purang, K., Andersen, C., Conversational adequacy: mistakes are the essence, Int. Jour. Human-Computer Studies, 48, 1998.

Purang, K., Systems that detect and repair their own mistakes, PhD thesis, Department of Computer Science, University of Maryland, College Park, 2001.

Rao, A.S., Georgeff, M.P., BDI-agents: From Theory to Practice, In Proceedings of the First International Conference on Multiagent Systems (ICMAS'95), San Francisco, 1995.

Stroulia, E., Goel, A.K., Redesigning a Problem-Solver's Operations to Improve Solution Quality. In Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, 1997.

Sutton, R.S., Barto, A.G., Reinforcement Learning: An Introduction, MIT Press, 1995.

Traum, D., Andersen, C., Chong, W., Josyula, D., Okamoto, Y., Purang, K., Anderson, M., Perlis., D., Representations of Dialogue State for Domain and Task Independent Meta-dialogue. In Electronic Transactions on Artificial Intelligence, 3:125-152, 1999.

Wang, X., Learning by Observation and Practice: An Incremental Approach for Planning Operator Acquisition. In Proceedings of the Twelfth International Conference on Machine Learning, 1995.