Toward Domain-Neutral Human-Level Metacognition

Michael L. Anderson[1,2], **Matt Schmill**[3], **Tim Oates**[2,3], **Don Perlis**[2]

Darsana Josyula[2,4], **Dean Wright**[3] and **Shomir Wilson**[2]

(1) Department of Psychology, Franklin & Marshall College, Lancaster, PA 17604
(2)Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742
(3) Department of Computer Science, University of Maryland Baltimore County, Baltimore, MD 21250
(4) Department of Computer Science, Bowie State University, Bowie MD 20715

Abstract

We have found that implementing a *metacognitive loop* (MCL), which gives intelligent systems the ability to self-monitor their ongoing performance and make targeted changes to their various action-determining components, can play an important role in helping systems cope with the unexpected problems and events that are the inevitable result of real-world deployment. In this paper, we discuss our work with MCL-enhanced intelligent systems, and describe the ontologies that allow MCL to reason about, appropriately classify and respond to the perfomance anomalies it detects.

Introduction: The Hardest AI Problem and the Metacognitive Loop

It is widely agreed that AI has been extremely successful in the narrow sense: given any of a large variety of well-defined AI problems, there is (or, so goes the common wisdom, there could easily be) an implemented solution using reasonably well-understood techniques. Or at the very least, there are solutions to similar problems, so that there is much reason to expect solutions to the problem at hand.

But in the wide sense, AI is a disappointment. Although many AI systems are at or beyond human-level competence at the individual tasks for which they were designed (from chess playing to medical diagnosis), AI systems are nowhere near human-level competence across the board. This is not simply a matter of building a system that has numerous subsystems, one for each of hundreds or thousands of individual tasks. The combined skills of a doctor, lawyer, chess-master, linguist, etc., does not a human make. For humans have, in addition to whatever number of individual skills, the ability to do reasonably well when faced with situations for which they have *not* been specifically trained. This general competence, the ability to muddle through when faced with the unexpected or unfamiliar, we consider to be the the core of human-level common sense. Indeed, this ability may well be the key to how we are *able* to train: we recognize when we lack an ability, make a decision to gain it, and recognize when it is time to stop learning—when training is complete; or complete enough for our purposes; or when it is ineffective, too slow, too expensive, or no longer important. We also (sometimes!) recognize when we are in over our heads, and then we ask for help or wisely give up.

It is this general sort of ability that current AI systems lack, with the result that they tend to be brittle, easily breaking when confronted with situations beyond their set of narrow competencies. Why is this so? And why has the brittleness problem been so hard to solve? Our hypothesis is this: while there are well-established and sohpisticated AI technologies for dealing with change or novelty e.g., machine learning (ML) and commonsense reasoning (CSR), neither alone is adequate to the task in question, since each needs the other to realize its true potential. Thus ML needs reasoned guidance (a form of CSR) as to when, why, what, and how to learn; and CSR needs trainable modules (to which ML can be applied) so that the fruits of such reasoning can be applied to bring about better future performance. Proper melding of ML and CSR should, in our view, go a long way toward solving the brittleness problem and bringing us closer to the ultimate goal of AI.

What will it take to meld ML and CSR in such a way as to solve the brittleness problem, i.e., to build a system that (under a wide range of unforeseen circumstances) adjusts itself instead of breaking? We have isolated four capacities.

First of all, a system will need to know enough about what it is doing (its goals, its actions, and their results), so that it can detect—and then assess—when those goals are not being achieved (Brachman 2002). This is a key place where CSR comes into play, in reasoning after the fact about such goal failures. Other relevant approaches include case-based reasoning and meta-level planning (Cox 2005; Anderson & Oates 2007).

Second, it will need to be able to make targeted changes to itself when there are indications of current inadequacies in what it is doing. This is where ML can come into play: if the desired changes involve any sort of training or adapting, the relevant trainable modules and their associated training processes must be invoked.

Third, it will need to be able to apply the same first two capacities to monitor how a given self-change is working out (thus there is a form of recursion involved). This again involves CSR in order to assess whether the targeted change is occurring properly.

And fourth, it will need to do this in real time.

This is a tall order, but the elements are clear enough to allow progress. This paper reports on our work to date and where we see it headed. The rest of the paper is organized as follows: We outline our specific approach to these four capacities (which we call MCL: the metacognitive loop); we discuss some early pilot studies; we present three ontologies that allow MCL to reason about, and appropriately classify and respond to the perfomance anomolies it detects; we show how these ontologies have been applied to a simulated autonomous vehicle domain; and we conclude with some general considerations and a discussion of future work.

MCL

The metacognitive loop (MCL) is our hypothesized agent's means of dealing with surpise, and consists of three parts: (i) note an anomaly, (ii) assess it, and (iii) guide a response into place. This seems to be a much-used capacity by humans: we regularly notice something amiss and make decisions about it (is it important, can I fix it, etc.) and then choose a response based on those decisions (ignore the anomaly, get help, use trial-and-error, etc.).

The Note phase corresponds to the agent's "self-awareness", allowing it to detect discrepancies between its expectations and its observations, which we call anomolies. Anomolies are such things as mismatches between intended and observed consequences of an action, or slower (or faster) than expected progress on some task. Once MCL detects such an anomaly, it moves into the Assess phase, where it reasons about the seriousness and likely cause of the anomaly. Having fomulated an hypothesis, MCL moves into the Guide phase, where it selects and implements a targeted solution to the problem. Having done this, it continues to monitor its performance on its various tasks, including the task of fixing the previously noted problem. If the solution fails, MCL can try another solution, or change its hypothesis about what the original problem was. It can also decide to give up and move on to some other task, or ask for help.

MCL has been the basis for much of our recent work. Our idea is that MCL can be a domain-general solution to the brittleness problem. We initally explored the feasability of this idea by applying the MCL paradigm to several very different domains (see Pilot Studies, below). Since then, we have taken the idea a step further by developing three special ontologies—indications, failures, and responses—that are intended to provide a very general, domain-neutral framework for reasoning about the causes of and possible solutions to any problems that the agent is facing. We describe these ontologies after presenting the pilot studies.

Pilot studies

MCL-enhanced reinforcement learning

In one early demonstration of the efficacy of MCL, we built some standard reinforcement learners using Q-learning (Watkins 1989; Watkins & Dayan 1992), SARSA (Sutton & Barto 1995) and Prioritized Sweeping (Moore & Atkeson 1993). We placed these learners 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 10,000 actions in this initial world, which was enough in all cases to establish a very good albeit non-optimal policy. In turn 10,001, the values of the rewards were abruptly changed.

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—the bigger the change, the worse they did. However, even a simple (and somewhat stupid) MCL-enhancement, that did no more than generate and monitor expectations for performance (average reward per turn, average time between rewards, and amount of reward in each state) and re-learn its entire policy whenever its expectations were violated three times, significantly outperformed standard reinforcement learning in the case of high-degree perturbations. And a somewhat smarter MCL-enhancement 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-perfomed standard reinforcement learners overall, despite some underperformance in response to mid-range perturbations (Anderson *et al.* 2006).

MCL-enhanced navigation

Another agent that we developed uses a neural net for navigation; however it also has a monitoring component that notices when navigational failures (such as collisions) take place, and records these and their circumstances. It is then able to use this information to assess the failures and make targeted changes to the neural net, including starting with a different set of weights, or re-training on a specific set of inputs. The agent exhibits better behavior while training, and also learns more quickly to navigate effectively (Hennacy, Swamy, & Perlis 2003).

MCL-enhanced human-computer dialog

One of the most important application areas for MCL has been natural language human-computer interaction (HCI). Natural language is complex and ambiguous, and communication for this reason always contains an element of uncertainty. To manage this uncertainty, human dialog partners continually monitor the conversation, their own comprehension, and the apparent comprehension of their interlocutor. Both partners elicit and provide feedback as the conversation continues, and make conversational adjustments as necessary. We contend that the ability to engage in such meta-language, and to use the results of meta-dialogic interactions to help understand otherwise problematic utterances, is the source of much of the flexibility displayed by human conversation (Perlis, Purang, & Andersen 1998), and we have demonstrated that enhancing existing HCI systems with a version of MCL allowing for such exchanges improved performance.

For instance, in one specific case tested, a user of the natural-language train-control simulation TRAINS-96 (Allen *et al.* 1996) says "Send the Boston train to New York" and then, after the system chooses and moves a train, says "No, send the *Boston* train to New York". Such an exchange might occur if there is more than one train at Boston station, and the system chose a train other than the one the user meant. Whereas the original TRAINS-96 dialog system would respond to this apparently contradictory sequence of commands by sending the very same train, our MCL-enhanced HCI system notes the contradiction, and, by assessing the problem, identifies a possible mistake in its choice of referent for ‘the Boston train’. Thus, the enhanced system will choose a different train the second time around, or if there are no other trains in Boston, it will ask the user to specify the train by name. The details of the implementation, as well as a specific account of the reasoning required for each of these steps, can be found in (Traum *et al.* 1999).

In more recent years, we have built a dialog system called ALFRED[1], which uses the MCL approach to accurately assess and resolve a broader class of dialog issues. The system deals with anomalies by setting and monitoring a set of time-related, feedback-related and content-related expectations.

For instance, if the user does not respond to a system query with in the expected time limit, then the system recognizes that there is a problem and tries repeating the query. However, continuous repetition of the query without a response from the user indicates a continuing problem (recall that it is part of MCL to monitor the progress of solutions), and causes a re-evaluation of the possible 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 will 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 ALFRED notices that it doesn’t know the word ‘Metro’, it will request specific help from the user, saying: “I don’t know the word ‘Metro’. What does ‘Metro’ mean?” Once the user tells the system that ‘Metro’ is another word for ‘Metroliner’, it is able to correctly implement the user’s request (Josyula 2005).

Ontologies

For MCL to be valuable as a general tool for addressing brittleness, it must be able to identify anomalous situations, and deal with them with minimal domain-dependent engineering. MCL must be able to leverage abstract, domain-independent reasoning to find solutions to problems without burdening the host system designer with the task of specifying how the host might fail and how to cope with those failures. To allow for this ability, we have developed three ontologies that support the required classification and reasoning abilities in each of the three MCL phases (Note, Assess, Guide). The *core* of these ontologies contains abstract and domain-neutral concepts; when an actual anomaly is detected, MCL attempts to map it onto the MCL core so that it may reason about it abstractly. Nodes in the ontologies are linked, expressing relationships between the concepts they represent. There are linkages both within and between the ontologies, which together allow MCL to perform abstraction and reasoning about the anomaly being considered.

In our current implementation, each of the three phases of MCL employs one of the ontologies to do its work. The Note phase uses an ontology of *indications*, where an indication is a sensory or contextual signifier that the system’s expectations have been violated. Processing in the indication ontology allows the Assess phase to map nodes in the indication ontology to nodes in its own *failure* ontology, which contains nodes that abstractly describe how a system might fail. Nodes in the failure ontology represent the underlying cause of expectation violations. Finally, when hypotheses about the failure type have been generated, the Guide phase maps that information to its own *response* ontology, which describes means for dealing with failures at various levels of abstraction. Through these three phases, reasoning starts at the concrete, domain-specific level of expectations, becomes more abstract as MCL moves to the concept of system failures, and then becomes more concrete again as it selects a specific system response based on the hypothesized failure.

In the following sections, we will describe in greater detail how the three ontologies are organized and how MCL gets from expectation violations to responses that can be executed by the host system, using the MCL-enhanced reinforcement learning system as an example.

Indications

The indication ontology consists of two types of nodes: domain-neutral *indication nodes* and domain-specific *expectation nodes*. Indication nodes belong to the MCL core, and represent general classes of sensory events that MCL might watch for during the operation of any host system. For example, the MCL core contains a node for the class of “spatial indicators”, which represents all sensory events pertaining to sensors operating in the spatial realm.

Expectation nodes represent concrete predictions of how the host system’s sensors and state should behave over a period of time and under foreseeable circumstances. Expectations are generated dynamically based on what the host system is doing; the expectations for an autonomous vehicle would be different depending on whether it was accelerating or decelerating, for example. Expectations may be specified by the system designer or learned by MCL. In either case, expectations must be linked into the core of indications when they are generated, since the reasoning performed by MCL takes place inside the core rather than with system-level expectations.

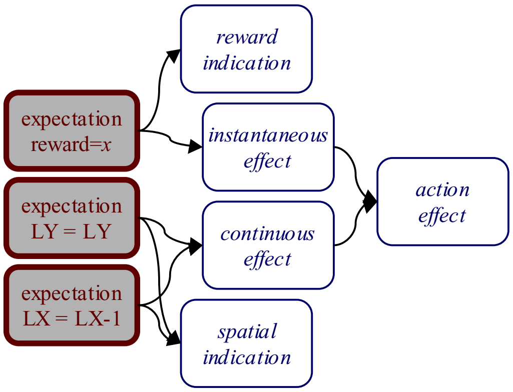

Figure 1: A fragment of the MCL indication ontology.

Consider the fragment of the indicator ontology pictured

[1] **A**ctive **L**ogic **F**or **R**eason **E**nhanced **D**ialog

in figure 1. This fragment shows three example expectations that the enhanced reinforcement learner might produce when it attempts to move into a grid cell containing a reward. First, a reward x should be experienced at the end of the movement. Second, the sensor LY (meant to signify the agent's Y location in Cartesian space) should not change. Finally, the sensor LX should decrease by one unit by the end of the action.

Indication nodes are shown in italics. Note that the concrete expectations have been linked to these nodes; linkage among nodes in the indication hierarchy generally express the *abstraction* relationship, that the source of a link is a member of the more abstract concept that the destination represents.

By the linkage of our example expectations in figure 1 to the MCL core, we see that we have two spatial expectations, with continuous feedback, and one reward expectation that will be experienced instantaneously. All three expectations belong to the abstract class "action effect", which expresses that the expectations are linked to the agent's intentional activity (and not to exogenous events).

The metacognitive reasoning process is initiated when an expectation is violated. MCL monitors all active expectations as it sits and waits in the Note phase. Once a violation occurs, the corresponding expectation node is *activated*. Reasoning with the MCL ontologies can be likened to a *spreading activation* algorithm. (Anderson 1983), and through this process, activation spreads into the MCL core along abstraction links.

Assume in our example that the agent moves, its LX and LY sensors change as expected, but no reward is received because the location of the reward has been changed. The violation of the reward expectation causes its corresponding node to become active. The "instantaneous effect", "action effect" and "reward indication" nodes are then activated by their association to the violated expectation. Once all violated expectations have been noted, and activation through the indication ontology is finished, the Note phase of MCL is complete.

Failures

Once the Note phase has been completed, MCL moves to the Assess phase, in which indications are used to hypothesize a cause for the expectation violation. The failure ontology serves as the basis for processing during the Assess phase.

It is worth explaining why MCL does not map directly from indications to responses. In fact, earlier incarnations of MCL did derive responses directly from expectation violations. The failure ontology was added because of the potentially ambiguous nature of indications. In many cases, a single indication might suggest several potential failures. Similarly, a given failure might be suspected when any of a number of indications are present. The mapping between indications and failures, then, might be one-to-many or many-to-one. This rich connectivity is lost without all three ontologies.

Nodes in the failure ontology are initially activated based on activations in the indication ontology. In addition to the internal linkage between nodes within the core indication ontology, there exist links between the ontologies. These connections, which we call *interontological links*, allow MCL to infer which nodes in an ontology should be activated at initialize time based on which nodes are activated in the ontology that precede it. In the case of the failure ontology, *diagnostic links* originating in the indication ontology express which class of failure is plausible given the indications.

Figure 2 shows a fragment of the MCL core failure ontology. Note that there are no concrete nodes in the failure ontology. It is a purely abstract, situation-independent ontology of system failures. Dashed arrows indicate incoming diagnostic links leading to the "sensor failure" and "model error" nodes, which are shaded and bold. These nodes represent the initial activation in the failure ontology in our enhanced reinforcement learning example; a reward indication can be associated with either of these types of failures.

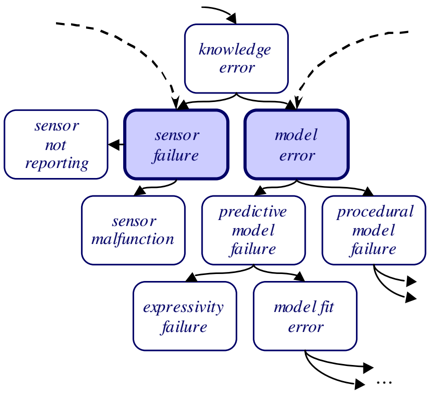

Figure 2: A fragment of the MCL failure ontology.

The remaining links in the figure are *intra*ontological, and express *specification*. A sensor may fail in two ways: it may fail to report anything, or it may report faulty data. Either of these is a refinement of the "sensor failure" node. As such, "sensor not reporting" and "sensor malfunction" are connected to "sensor failure" with specification links in the ontology to express this relationship.

As in the Note phase, reasoning occurs on the model of spreading activation, whereby inference along specification links activates more specific nodes based on which of the more abstract nodes are relevant (activated). Of particular interest in our RL example is the "predictive model failure" node, which follows from the "model error" hypothesis. The basis for action in Q-learning is the predictive model (the action-value function), and the failure to achieve a reward often indicates that the model no longer fits the domain.

Responses

Once activation in the failure ontology is complete, outgoing interontological links from active failure nodes allow MCL to move into the Guide phase. In the Guide phase, potential responses to hypothesized failures are activated, evaluated, and implemented in order of their expected utility. Interontological links connecting failures to responses are called *prescriptive links* because given a particular failure, such a link prescribes a class of responses expected to correct the failure.

Figure 3 shows a fragment of the MCL response ontology. Pictured are both MCL core responses (in italics) and host-level responses (in bold). The latter are concrete actions that can be implemented by the host system. Host system designers specify the concrete ways in which MCL can affect changes, such as by changing Q-learning parameters as seen in figure 3.

In the portion of the response ontology shown, prescriptive links from the failure ontology are pictured as dashed arrows. These links cause initial activation in the nodes “modify predictive models” and “modify procedural models”. Like the failure ontology, internal links in the response ontology are primarily *specification links*. They allow MCL to move from general response classes to more specific ones, eventually arriving at concrete responses. In our example, concrete nodes correspond to either parameter tweaks in Q-learning, or resetting the action-value function altogether.

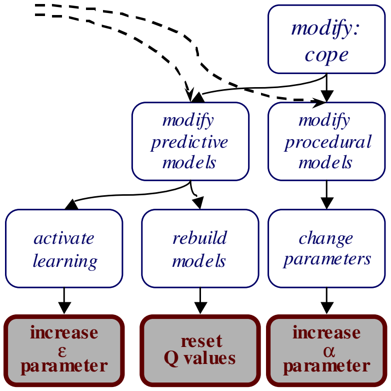

Figure 3: A fragment of the MCL response ontology.

Closing the loop

Once MCL has arrived at a concrete response in the Guide phase, the host system can implement the response. In our enhanced RL example, either clearing the Q values and starting over, or boosting the ϵ parameter to increase exploration will start the agent on the path to recovery. The third possibility pictured in figure 3, to increase α, may not (α modulates the effect of new experience on the current action-value function). This is why all the activated failures and responses are only considered *candidate* responses, and why MCL must verify that a response is working before it considers an expectation violation addressed.

After a response has been chosen, the state of the three ontologies is stored while the necessary action is taken. MCL re-enters the Note phase, waiting to receive feedback from the effected repair. If no new expectation violations are received, then the changes effected during the repair are made permanent, and the violation is considered addressed. If the violation persists, or a new one occurs, then MCL deactivates the invalidated candidate response, and revisits its options for recovery.

Using the MCL ontologies with WinBolo

WinBolo [www.winbolo.com] is a networked, multiplayer game where players command simulated tanks on a virtual battle field. Figure 4 shows a screen shot of WinBolo with a tank approaching a pillbox. The tanks can be contolled by human players or be under the control of programs (called “brains” in the parlance of WinBolo.)

Using WinBolo, we have implemented a tank programmed to seek out stationary pillboxes and capture them. The tank uses the following plan: (1) locate the nearest pillbox, (2) drive to and over it, (3) repeat until all pillboxes are captured.

This three step plan serves the robot tank well so long as the pillboxes are unarmored. However, when the world is modified to include armored pillboxes the brittleness of the tank’s programming shows itself.

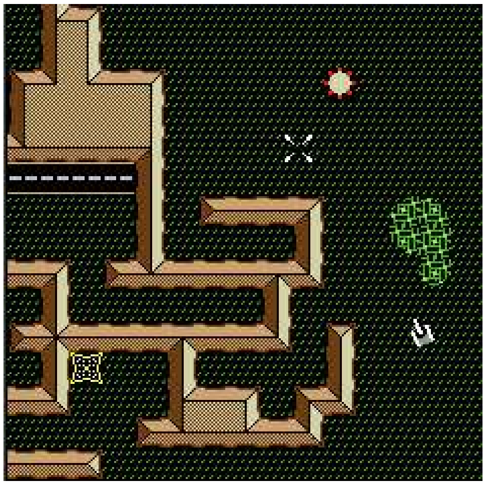

Figure 4: A screenshot of Bolo.

An armored pillbox cannot be driven over and thus cannot be captured. Also, armored pillboxes will shoot at nearby tanks, damaging and eventually destroying them. Thus, a tank equipped with only the three-step plan will drive near the pillbox, find itself unable to drive *over* it, yet keep blindly trying. Meanwhile, it will be repeatedly shot by the pillbox until the tank is destroyed. Without the ability to notice this problem and change its behavior, it will repeat this cycle forever (as in most video games, players are given a new tank after the old one is destroyed).

However, a tank equipped with the same rudimentary plan, but enhanced with MCL and the linked indication, failure, and response ontologies performs much better. It starts with having and monitoring expectations. The enhanced tank’s MOVE_TO action includes the expectation that moving the tank will not destroy it (a reasonable expectation for most actions). The concrete expection TANK_LIVES is based on the “newtank” flag of the WinBolo API. If newtank is TRUE then the TANK_LIVES expectation has been violated. The TANK_LIVES node is linked to the abstract ExistenceViolation/Death node of the indications ontology which in turn has interontological links to the ModelError and Resources/Cost nodes of the failure ontology. These links represent the assumptions that death could come as the result of executing a plan that was imperfect or whose cost for execution was (much) too high. The ontologies for WinBolo are shown in outline in figure 5.

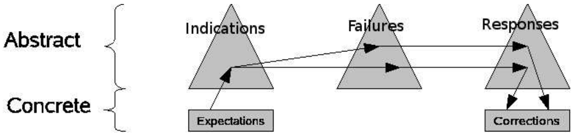

Figure 5: Overview of the MCL ontologies for Bolo.

In the Assess phase of MCL, it follows the link from the ModelError node to the ProceduralModelError node. This brings on the Guide phase. The ProceduralModelError failure node is linked to the ModifyProceduralModels node in the response ontology. Once in the Guide phase, MCL follows a link from ModifyProceduralModels to AmendController. From there it selects the concrete response Hypothesis-Driven Operator Refinement (HDOR), which works as follows: MCL compares the conditions (sensor values) on occasions on which the task failed with those on which it succeeded. If there is more than one difference, it uses a heuristic ranking method to order them. In the current case, it notices that the pillbox armor level is different. The next step is to see if the system knows a way to change armor level. If it does not, it looks for actions where the effect on armor level is *unknown*, and tests them until it finds one that changes armor level. So, either the system knows, or eventually discovers, that the FIRE_ON action lowers an object's armor level, and inserts the FIRE_ON action into the brain's CAPTURE_PILLBOX plan.

With this modified plan, the resurrected tank will shoot at armored pillboxes until the pillbox armor level is zero, thereby allowing the MOVE_TO action to complete successfully and the pillbox to be captured. If this response had not worked—if, for instance the tank still failed even when armor level was zero—it would revisit its options and try something different.

Conclusions and Future Work

Although we have had demonstrable success in decreasing the brittleness of different systems with MCL, there is much more work to do, and many open questions remain. For instance, the current interontological links are very sparse, and many more need to be established before truly general reasoning about anomalies and their causes and remedies can be implemented. Moreover, we have yet to address the questions of how to set the weights of the various internodal links, whether there is likely to be a great deal of uniformity in weight assignment across domains, or whether these weights need to be set (e.g. learned) for each individual system. More information on this project can be found by going to http://www.ucl.ac.uk/commonsense07/papers/notes/anderson-et-al/

References

Allen, J. F.; Miller, B. W.; Ringger, E. K.; and Sikorski, T. 1996. Robust understanding in a dialogue system. In *Proceedings of the 1996 Annual Meeting of the Association for Computational Linguistics (ACL'96)*, 62–70.

Anderson, M. L., and Oates, T. 2007. A review of recent research in metareasoning and metalearning. *AI Magazine* 28(1).

Anderson, M. L.; Oates, T.; Chong, W.; and rlis, D. P. 2006. Enhancing reinforcement learning with metacognitive monitoring and control for improved perturbation tolerance. *Journal of Theoretical and Experimental Artificial Intelligence* 18:387–411.

Anderson, J. 1983. A spreading activation theory of memory. *Journal of Verbal Learning and Verbal Behavior* 22:261–295.

Brachman, R. J. 2002. Systems that know what they're doing. *IEEE Intelligent Systems* 17(6):67–71.

Cox, M. T. 2005. Metacognition in computation: A selected research review. *Artificial Intelligence* 169(2):104–41.

Hennacy, K.; Swamy, N.; and Perlis, D. 2003. RGL study in a hybrid real-time system. In *Proceedings of the IASTED NCI*.

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

Moore, A. W., and Atkeson, C. G. 1993. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning* 13:103–130.

Perlis, D.; Purang, K.; and Andersen, C. 1998. Conversational adequacy: mistakes are the essence. *Int. J. Human-Computer Studies* 48:553–575.

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

Traum, D. R.; Andersen, C. F.; Chong, W.; sana Josyula, D.; Okamoto, Y.; Purang, K.; O'Donovan-Anderson, M.; and Perlis, D. 1999. Representations of dialogue state for domain and task independent met a-dialogue. *Electronic Transactions on Artificial Intelligence* 3:125–152.

Watkins, C. J. C. H., and Dayan, P. 1992. Q-learning. *Machine Learning* 8:279–292.

Watkins, C. J. C. H. 1989. *Learning from Delayed Rewards*. Ph.D. Dissertation, Cambridge University, Cambridge, England.