APPROVAL SHEET

Title of Thesis: Finding a Temporal Comparison Function for the Metacognitive Loop

Name of Candidate: Dean Earl Wright III
Doctor of Philosophy, 2011

Thesis and Abstract Approved: ________________________
Dr. Tim Oates
Associate Professor
Department of Computer Science and
Electrical Engineering

Date Approved: ______________________________

Curriculum Vitae

Name: Dean Earl Wright III

Permanent Address: 422 Shannon Court, Frederick, MD 21701

Degree and date to be conferred: Doctor of Philosophy, Fall 2011.

Date of Birth: 27 November 1954.

Place of Birth: La Rochelle, France.

Previous Degrees:

Hood College, Master of Business Administration, 2005
Hood College, Master of Science (Computer Science), 2001
Hood College, Bachelor of Science (Computer Science), 1998
Frederick Community College, Associate in Arts (Business Administration), 1993

Professional publications:

M. Anderson, M. Schmill, T. Oates, D. Perlis, D. Josyula, D. Wright and S. Wilson. Toward Domain-Neutral Human-Level Metacognition. In Proceedings of the 8th International Symposium on Logical Formalizations of Commonsense Reasoning, pages 1–6, 2007.

M. Schmill, D. Josyula, M. Anderson, S. Wilson, T. Oates, D. Perlis, D. Wright and S. Fults. Ontologies for Reasoning about Failures in AI Systems. In Proceedings of the Workshop on Metareasoning in Agent-Based Systems, May 2007.

M. Anderson, S. Fults, D. Josyula, T. Oates, D. Perlis, M. Schmill, S. Wilson, and D. Wright. A Self-Help Guide For Autonomous Systems. AI Magazine. 29(2):67-76 2008.

M. Schmill, M. Anderson, S. Fults, D. Josyula, T. Oates, D. Perlis, H. Shahri, S. Wilson, and D. Wright. The Metacognitive Loop and Reasoning about Anomalies, chapter 12, pages 183-198. The MIT Press, 2011.

Professional positions held:

White Oak Technologies, Inc., October 2007–present
CRW/Logicon/Northrop Grumman, January 1985–July 2007
Scientific Time Sharing Corporation (STSC), January 1977–January 1985

ABSTRACT

Title of Thesis: Finding a Temporal Comparison Function for the Metacognitive Loop

Dean Earl Wright III, Doctor of Philosophy, 2011

Dissertation directed by: Dr. Tim Oates, Associate Professor
Department of Computer Science and
Electrical Engineering

The field of Artificial Intelligence has seen steady advances in cognitive systems. However, many of these systems perform poorly when faced with situations outside of their training. Since the real world is dynamic, this brittleness is a major problem in the field today. Adding metacognition to such systems can improve their operation in the face of perturbations found in dynamic environments.

I developed a six-level taxonomy that divided metacognitive systems according to their capabilities. This ranged from Level 0: Bereft that offered no metacognitive assistance, to Level 5: Anticipating that would make suggestions before the agent needed assistance. Using this taxonomy, I deconstructed an existing metacognitive system (MCL) so that it could operate at any of the four lower levels of the taxonomy.

I added, to the Level 3: Temporal version of MCL, a number of comparison functions designed to determine if the problem the agent is currently facing is similar to any previous problem. I created functions to both establish baseline operation performance and to try to optimize the agent's performance in a perturbed environment.

I extended an existing Mars Rover domain simulation to add more problem and recovery options. In this environment, I tested the temporal comparison functions to determine how well they were able to distinguish on type of perturbation from another and how much they helped (or hindered) the agents subjected to those perturbations. Several comparison functions were able to aid the Rover in completing its tasks, although none were perfect.

Finding a Temporal Comparison Function for the Metacognitive Loop

by

Dean Earl Wright III

Thesis submitted to the Faculty of the Graduate School
of the University of Maryland in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2011

For my three parents, two children and one wife

ACKNOWLEDGMENTS

I want to thank the past and present members of the UMBC and UMCP MCL working group who brought MCL together so that I could take it apart.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

This chapter starts by describing some of the problems associated with deploying Artificial Intelligence agents into the world. It then offers Metacognition as a domain-neutral solution to those problems. A leveled taxonomy for the metacognition support given to an agent is described. Within this setting, the dissertation's main claim is made. Then, a series of research questions are proposed which support and demonstrate the claim. The chapter ends with an outline of the remainder of the dissertation and a listing of the contributions of the research.

1.1 AI Agent Problems

The field of Artificial Intelligence has seen steady advances in cognitive systems. AI has acquitted itself in one area after another: theorem proving, game playing, machine learning, and more. While advances in computer speeds and memory sizes have certainly helped, the majority of the achievements have come from new algorithms and experience in applying them. However, many cognitive systems perform poorly when faced with situations outside of their training or in a dynamic environment. A robot trained to search out a dark blue goal may or may not detect a light blue one. A robotic car trained to drive on American roads will likely be a danger to itself and others if transported to a country with left-hand driving rules. A switch from an instrument reading in miles to one in kilometers may doom a spacecraft (Stephenson 1999). Driverless vehicles in the DARPA Grand Challenge had mechanical failures and wandered into obstacles for which their programming was not prepared (Hooper 2004).

The transition between laboratory training and successful real-world operation remains a major challenge. To cope with possible future encounters, additional rules and/or more training can be used, but this increases the cost and lengthens the time between conception and deployment. Alternatively, greater ability can be given to the agent to explore, learn, and reason about its environment letting it deal with the brittleness problem on its own (Brachman 2006).

Adding more capabilities to an agent to cope with possible perturbations increases the complexity of the agent and, except during periods of perturbation,[1] may decrease performance. Adding metacognition to such systems can improve their operation in the face of such perturbations without sacrificing performance during normal operations. Only when a problem has been noticed does the problem correction code need to be active. Metacognition can monitor an agent's performance and invoke corrective action only when needed.

1.2 Metacognition

The philosophical origins of metacognition may be traced to the dictum of "know thyself." Metacognition is studied as part of developmental and other branches of psychology. While there are several different approaches, one common model is a cognitive process which is monitored and controlled by a metacognitive process as shown in Figure 1.1. Metacognition can be studied in conjunction with metaknowledge (knowledge

[1]A perturbation is an unexpected change in the environment.

about knowledge) and metamemory (memory about memory) (Cox 2005b). Metacognition is also referred to as metareasoning (Russell & Wefald 1991).

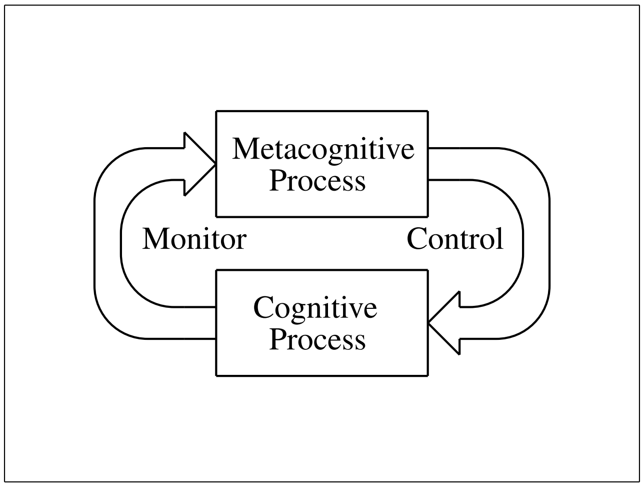


FIG. 1.1. Metacognitive monitoring and control

The canonical depiction of a software agent (Russell & Norvig 1995; Cox & Raja 2011) (Figure 1.2(a)) has sensors to perceive the environment and activators with which the agent tries to control it. Metacognition can be layered onto a software agent so that the metacognitive process monitors and controls the cognitive process of the software agent (Cox & Raja 2007; 2011), as shown in Figure 1.2(b), with metamemory and metaknowledge.

Metacognition can improve the performance of the agent in the environment by providing two control functions. The first is to inform the agent when a cognitive task (e.g., the selection of the next action to perform) has been satisfactorily achieved so that the agent can move on to another task (such as performing the selected action). For some agents, the test for task completion (or “good enough” performance) is built into the cognitive process itself. For example, the cognitive task may be limited to selecting (based on a specified

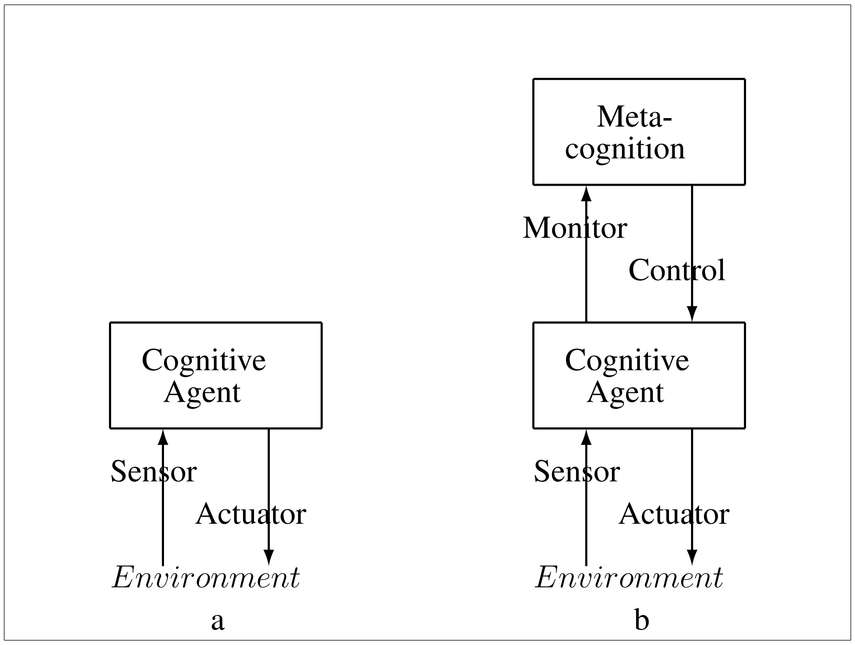


FIG. 1.2. Software agent (a) without and (b) with metacognition

estimated utility) from anong a fixed number of choices. In such cases, there is little opportunity for metacognitive intervention.

The second metacognitive control function is to reflect on the performance of the agent. Being able to detect when the agent's goals are not being achieved is the first step to being able to improve the agent's performance (Brachman 2002). Reflection can be done at the completion of a successful task, but is most often performed after a failure. Rather than wait for a complete failure, reflection can also be invoked any time an expectation of performance is not achieved. The reflective metacognitive process evaluates the agent's decisions to determine where a change would improve performance. The response may suggest a change (or repair) to the agent's current cognitive state such as invoking a learning module. Repairs can be as simple as just trying again or may require more resources of the agent to implement. It is the reflective metacognitive control function that will be examined in this dissertation.

1.3 Levels of Metacognition

The metacognition layer added to a software agent can be as simple or as complicated as the designer wishes. I defined a hierarchy to help categorize and gauge the power of different metacognition systems. This taxonomy focuses on capabilities rather than details of the implementation. There are six levels (listed in Table 1.1), moving from having no metacognitive capability to one that attempts to prevent future needs for corrective action.

0–Bereft The agent has no metacognitive system. This is the baseline case. The higher levels have to improve the performance of this agent to justify the addition of metacognition.

1–Instinctive The agent has a metacognitive system that directly maps exceptions to repairs. When there is an exception that triggers metacognition, the response is determined solely and directly from the exception.

2–Evaluative The agent has a metacognitive system that evaluates the exceptions to determine the appropriate repairs. The evaluation can be done by any number of methods (e.g., a neural network) but a Bayesian network will be used in the systems described here.

3–Temporal The agent has a metacognitive system that evaluates the current exception(s) as well as any past exceptions and repair attempts to determine the appropriate response to be used. My research centers on providing functions that compare the agent's current state to previous problem states.

4–Evolving As in the temporal level, the agent's metacognitive system evaluates the current exception(s) as well as any past exceptions and repair attempts when choosing the appropriate response. Additionally, the metacognitive system adjusts its evaluation procedure and/or parameters based on the successes and failures of the repairs.

5–Anticipating The agent's metacognitive system attempts to avoid the precursors of the conditions that lead to exceptions.

Table 1.1. Levels of Agent Metacognition

Level	*Name*	*Short Description*
0	Bereft	No metacognitive component
1	Instinctive	Exception dictates response
2	Evaluative	Reasoned response
3	Temporal	Past events included in reasoning
4	Evolving	Reasoning process adjusted by events
5	Anticipating	Guides agent to avoid exceptions

For example, suppose that an agent experiences a problem with a GPS sensor whenever the agent's battery is nearly drained and then completely recharged. A level 0 agent would receive no help in diagnosing and repairing the problem. A level 1 agent would have a hard-coded response to a problem with the GPS sensor, such as reinitializing the sensor. A level 2 agent would determine a response based on a rule base, neural network, or other reasoned approach. It would choose the response that had the highest utility (incorporating the cost to implement the response and the probability of success). A level 3 agent would take into account past responses (and their success or failure) to similar problems when determining the best repair to attempt. A level 4 agent would update its decision process (rules added/deleted from the rule base, neural connections or activation values changed, network links altered, etc.) as the result of each exception/response episode. Finally, a level 5 agent would attempt to have the agent avoid situations which prompted the exception (e.g., by not letting the battery get nearly depleted or by recharging in stages). In my research on temporal comparison functions, I created agents using metacognition levels 0 through 3 (Figure 1.3). Levels 4 and 5 are an area of possible future work (Sections 12.3 and 12.4).

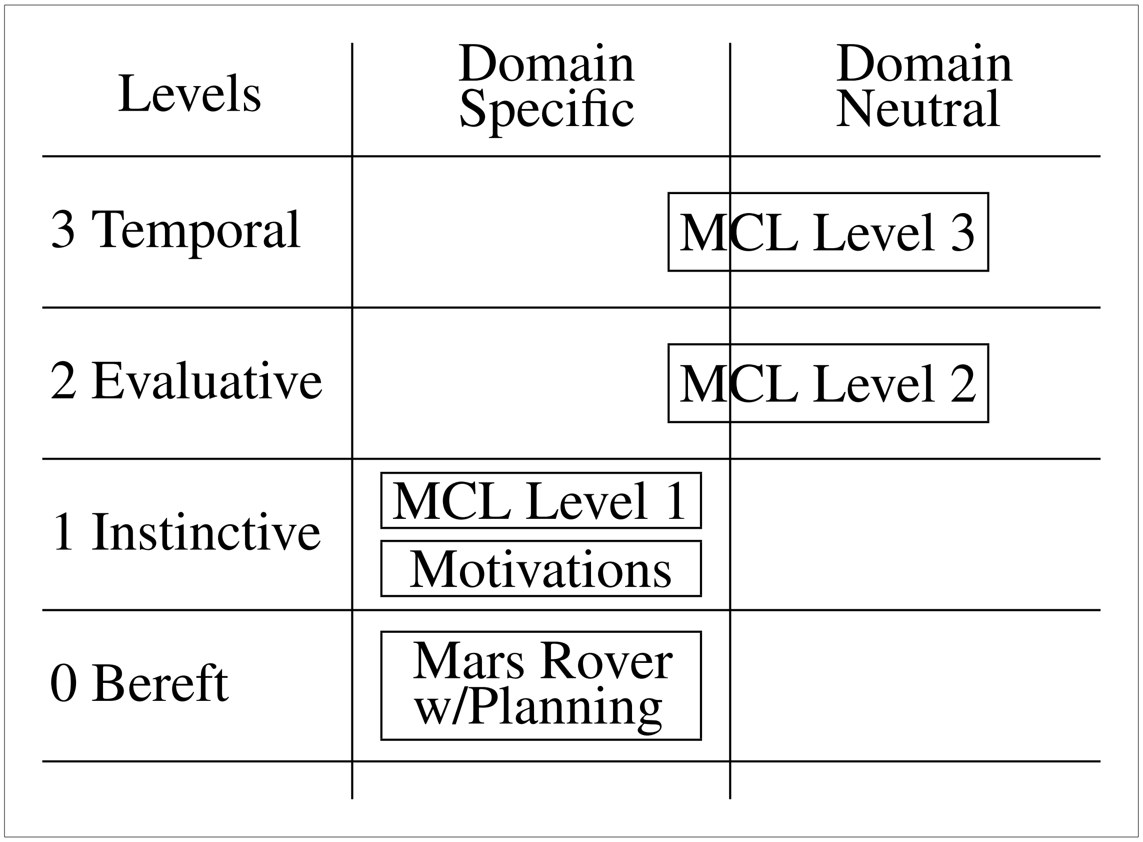


FIG. 1.3. Metacognitive Levels for the Mars Rover agent systems

1.4 Claim

Adding memory of past problems to MCL Level 3: Temporal provides better assistance to the agent than MCL Level 2: Evaluative without such memory when there are multiple perturbations. To effectively use the memory of past problems to guide the agent, MCL has to determine, using a temporal comparison function, which of the previous problems (if any) are similar to the current problem.

1.5 Questions to be Answered

During the evaluation of temporal comparison functions for a level 3 metacognitive system, I strived to answer a number of questions:

1. How does one determine that two problems are similar or different?
2. How can knowing if the current problem is the same or different from a previous problem help an agent?
3. How does this determination alter the agent's response to the problem?
4. How much does the correct determination help an agent?
5. How much does an incorrect determination hinder an agent?

1.6 Outline of Paper

This section describes the remainder of the dissertation and where each of the research questions from the last section are answered.

A Q-Learner in a small grid world is used to demonstrate the uses of hard-coded, domain-specific metacognition in Chapter 2. Agents for the Chippy grid world are defined for metacognition levels from 0 to 3. Chapter 3 introduces the more interesting Mars Rover domain with a planning agent that will be used for the remaining chapters. The next chapter adds an instinctive (level 1) metacognition to the Mars Rover.

The next three chapters, using my idea of multiple metacognitive levels, deconstruct and explain a metacognitive system that I helped develop as part of a UMBC/UMCP research team. The Motivated Mars Rover from Chapter 4 will receive assistance from the Metacognitive Loop (MCL). This will be done at metacognitive Levels 1: Instinctive (Chapter 5), 2:Evaluative (Chapter 6), and 3: Temporal (Chapter 7).

Chapter 8 describes additional temporal comparison functions that were added to the existing Level 3: Temporal MCL. That chapter also contains an evaluation of how well they would likely work in practice.

Chapter 9 describes an experiment that will test the MCL levels and comparison functions. The chapter begins with a review of the Mars Rover simulation and the scenarios that will be used in the experiments. Each experiment is run using metacognitive levels 0 through 3. The experiments are designed to show if there is an advantage for the agent to use metacognitive level 3. Running these experiments with multiple temporal comparison functions shows the effect on the agent's performance of correctly and incorrectly deciding if the current exception is related to a previous exception.

Chapter 10 provides details on the performance of the Mars Rover agent using metacognitive levels 0 through 3. The results show that (a) the performance of the agent using level 3 metacognition exceeds that of the other agents; (b) the benefit of correctly determining that the current exception is similar to a previous one; and (c) the penalty for making an incorrect determination can result in failure.

The remaining chapters describe related work (Chapter 11), future work (Chapter 12) and final conclusions (Chapter 13).

1.7 Research Contributions

My research contribution consists of three parts. First is the multi-level metacognition taxonomy that provides a framework for examining and comparing different metacognition systems. This allowed me to produce my second contribution: the deconstruction of the operations of MCL and then the construction of versions of MCL that would operate on the first four metacognitive levels (0 to 3). As my third contribution, I added, in addition to the existing temporal comparison in MCL Level 3: Temporal, several new comparison functions. Some of these functions were designed to establish baseline operational performance. Most, however, were developed to improve the performance of a simulated Mars Rover when faced with multiple perturbations.

Chapter 2

METACOGNITIVE LEVELS AND CHIPPY

This chapter shows how metacognition can be used with software agents. It uses a Q-Learner in the simple Chippy Grid World. The Q-Learner is augmented with three different levels of hard-coded, domain-specific metacognition from Level 0: Bereft to Level 3: Temporal.

2.1 The Chippy Grid World

This section examines the Chippy Grid World and a Q-Learning agent for exploring it. A method for perturbing the world is given that will be used to see how well the Q-Learner can recover from changes in the reward values. Other learnings such as SARSA (Rummery & Niranjan 1994; Sutton & Barto 1995) and Prioritized Sweeping (Moore & Atkeson 1993) could have also been used with Chippy with comparable results.

2.1.1 Q-Learning

Q-Learning (Watkins 1989; Watkins & Dayan 1992) is a reinforcement learning method that uses the time discounted reward value for each action in each state. Learning occurs continuously while the agent is operating. For each action of each state the expected value of taking that action in that state (the Q value) is accumulated using

$$Q(s, a) = Q(s, a) + \alpha(r + (\gamma \max Q(s, *)) - Q(s, a)),$$

where $Q(s, a)$ is the expected value of taking action a in state s, and α is the learning rate (typically around .5). Higher values place more emphasis on recent rewards while lower values favor long-term information. r is the reward received from taking action a in state s. τ is the discount rate for future rewards. Typical values are from .7 to .8. The higher the value the more future rewards influence the Q value. Usually, Q-learner selects the action in state s with the highest Q value. But, based on the exploration rate ϵ, the Q-learner will select a random action.

2.1.2 Chippy Grid World

The Chippy Grid World (Anderson *et al.* 2006) is an 8 by 8 square matrix as shown in Figure 2.1. An agent can move in the four cardinal directions from square to square. The agent cannot leave the board as any attempt to leave from one of the edge squares just keeps you in that square. The lower left (R1) and upper right (R2) squares provide rewards and then transport the agent to the opposite corner. The values of R1 and R2 can be any of several pairs. In the following example, R1 and R2 are initially 10 and -10. The agent starts in one of the center squares and continues to move (and occasionally transport) until the simulation is stopped.

Figure 2.2 shows the policy learned by a Q-Learner in a Chippy Grid World after 1,000 moves. Since only 2 of the 64 squares contain a reward, the Q-Learner makes many moves (average = 98) before even seeing the first reward so that learning can begin. After many more moves, a policy (direction) is learned so that most squares will direct the agent toward the positive reward. Exceptions occur in the upper left and lower right quadrants of

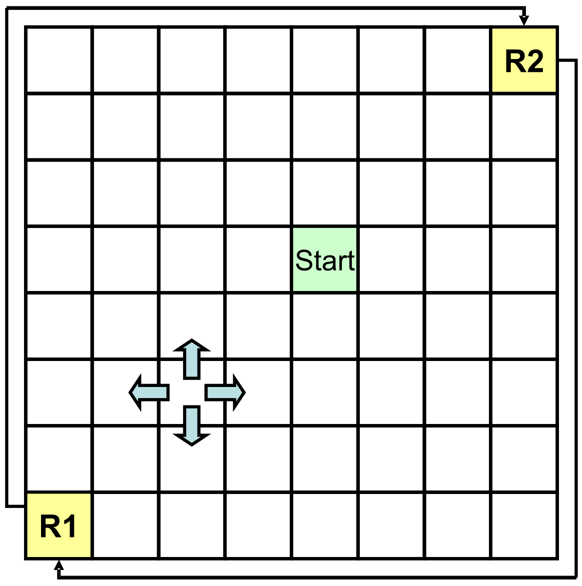


FIG. 2.1. An 8x8 grid world with two rewards

the board, that are rarely traveled to once the diagonal path between R1 and R2 is learned.

The learning rates ($\alpha = 0.5, \gamma = 0.9, \epsilon = 0.05$) produce a policy that converges in about 5000 steps. From that point onward, the agent gets a reward of 10 every 14 steps plus an occasional exploratory move.

2.1.3 Perturbing the Chippy Grid World

Perturbations are introduced into the Chippy Grid World by changing the values of the two goal squares (R1 and R2). In the Chippy experiments, the initial values for R1 and R2 are one of (10, -10), (25, 5), (35, 15), (19, 21), (15, 35), or (5, 25). After letting the agent take 10,000 steps, the values for R1 and R2 are changed to a different pair of values from one of the following: (-10, 10), (25, 5), (35, 15), (19, 21), (15, 35), or (5, 25). Additionally, if the initial configuration is (19, 21), then an addition subsequent configuration of (21, 19) is also tried, giving a total of 22 different combinations as shown in Table 2.1.

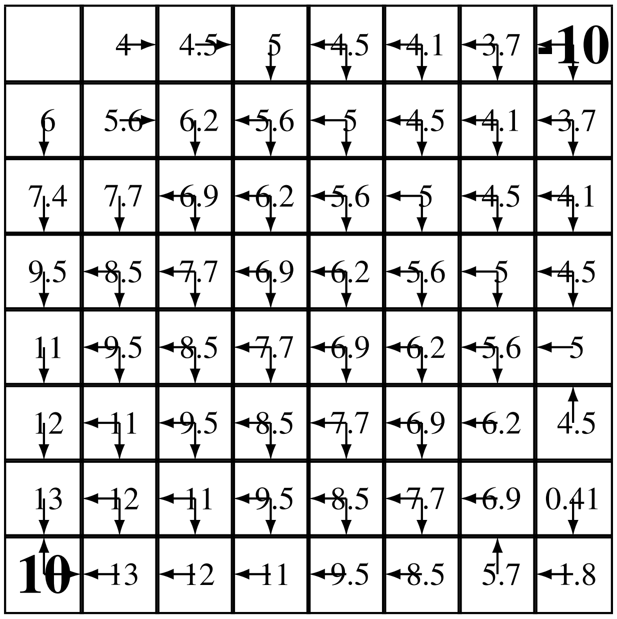


FIG. 2.2. Chippy policy after 1,000 moves for rewards (10,-10)

One advantage of Q-Learning is that it continues to update the policy for each square. Eventually, it will learn the reverse path to the new location of the positive reward. To be successful, the addition of metacognition to the agent should help the agent learn the new policy faster than it does without metacognition.

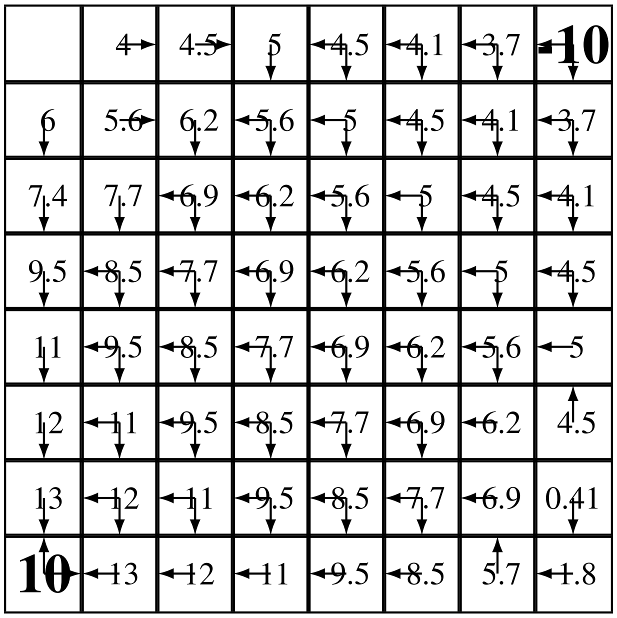


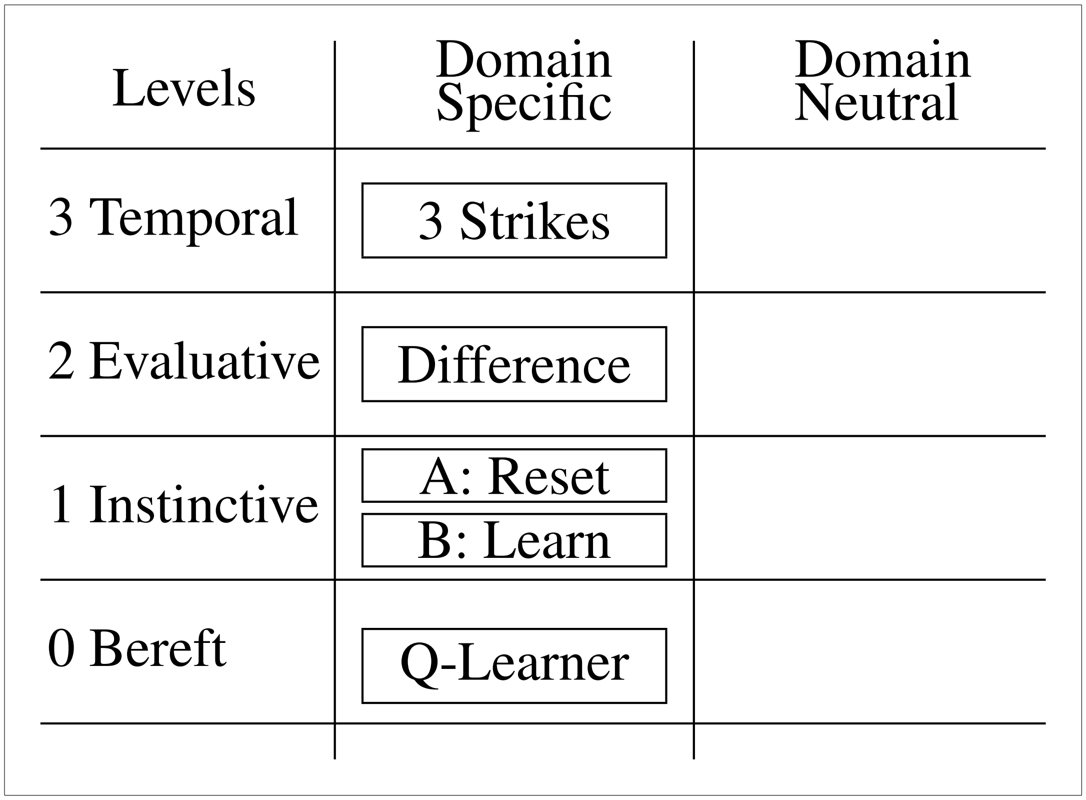
FIG. 2.3. Chippy policy after 5,000 moves for rewards (10,-10)

Table 2.1. Chippy Initial and Subsequent Reward Values

Initial	*Subsequent*					
(10, -10)	(-10, 10)	(25, 5)	(35, 15)	(19, 21)	X	X
(25, 5)	(-10, 10)	X	(35, 15)	(19, 21)	X	(5, 25)
(35, 15)	(-10, 10)	(25, 5)	X	(19, 21)	(15, 35)	X
(19, 21)	(-10, 10)	(25, 5)	(35, 15)	(21, 19)	X	X
(15, 35)	(-10, 10)	(25, 5)	X	(19, 21)	X	X
(5, 25)	(-10, 10)	X	(35, 15)	(19, 21)	X	X

2.2 Metacognition for Chippy

Metacognition can be added by the Chippy Q-Learner to help it respond to reward perturbations. Several different metacognitive agents are described in this section. They are at several different metacognitive levels but all were hardcoded to work in the Chippy domain. The agents and thier metacognition levels are shown in Figure 2.4.



Levels	Domain Specific	Domain Neutral
3 Temporal	3 Strikes	
2 Evaluative	Difference	
1 Instinctive	A: Reset B: Learn	
0 Bereft	Q-Learner	

FIG. 2.4. Metacognitive Levels for the Chippy demonstration systems

2.2.1 Interfacing to Metacognition

After each move, the agent invokes the metacognition monitor, passing the location to which the agent moved, the reward expected, and the reward received. The monitor can return one of several responses: *do nothing*, *reset the policy*, and *increase the learning rate*. If the response is *do nothing*, the agent need not take any action. If the response is *reset the policy*, the value for the square/direction policy for all squares and directions is reset to zero, effectively causing the agent to forget anything it learned. The response of *increase the learning rate* means that the epsilon value for the Q-Learner should be increased. This will allow more off-policy moves. The sensors monitored and the control suggestions are shown in Figure 2.5.

The monitoring of the sensor and the generation of the control suggestions is hard-coded and specific to the Chippy domain. A more domain-neutral metacognition system will be described in the MCL background chapters.

2.2.2 Level 0: Bereft Chippy

Chippy with metacognition level 0 always receives *do nothing* as the suggestion. This serves as the base case. Although the implementation does call the metacognition monitor function, it always gets back *do nothing* and thus acts just like a Q-Learner with no metacognition.

2.2.3 Level 1: Instinctive Chippy

There are two implementations of Chippy with metacognition 1. In both cases, if the expected and actual reward for a square are equal (or if this is the first time visiting the square), the suggestion will be *do nothing*. The first version (A) returns a suggestion of *reset the policy* if the expected and actual rewards are different. Version B returns a

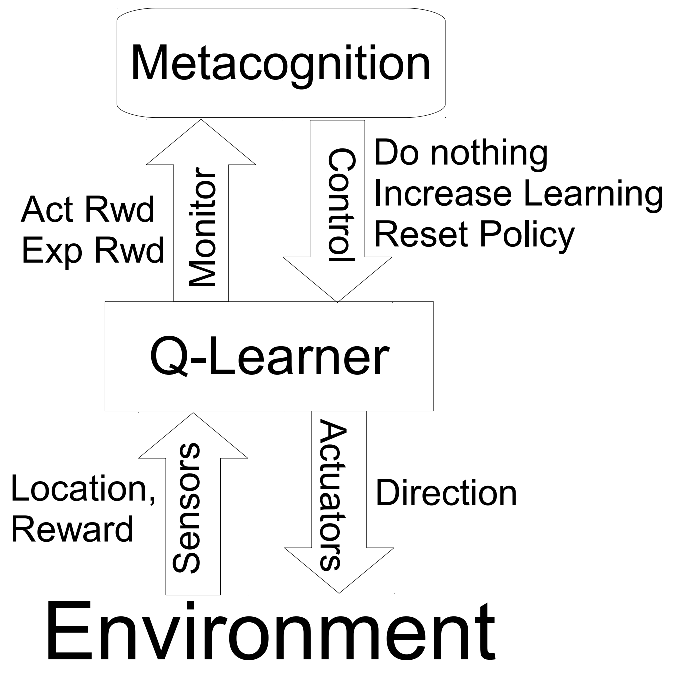


FIG. 2.5. Chippy agent with metacognition

suggestion of *increase the learning rate* if the rewards are not equal.

2.2.4 Level 2: Evaluative Chippy

A Chippy Q-Learner with metacognition level 2 will return *do nothing* if the expected and actual reward for a square are equal (or if this is the first time visiting the square). When the expected and actual rewards are different, the magnitude of the difference is determined. If the difference is small than a suggestion of *do nothing* is returned. For larger differences where the reward is less than expected, a suggestion of *increase the learning rate* is given. For even larger differences or when the reward has gone from positive to negative, the suggestion of *reset the policy* is given.

2.2.5 Level 3: Temporal Chippy

A Chippy Q-Learner with metacognition level 3 operates like level 2 in that the magnitude of the reward difference is used to determine which response of *do nothing*, *increase the learning rate* or *reset the policy* is returned. It also includes a counter that tracks the number of times an unexpected reward has been received and, when it succeeds a preset threshold (nominally 3), a response of *reset the policy* is returned and the counter is reset. The counter of number of times an unexpected reward occurred is the temporal element in the Chippy level 3 temporal metacognition.

2.3 Chippy Experimental Results

Using the 22 reward pairs given in Table 2.1, 20 experimental runs were done for each of the five metacognitive agents outlined in the previous section. Table 2.2 shows the average reward totals for the experiments. Rewards shown in *italics* indicate an average reward that is higher than the average reward for the unaided Q-Learner. Rewards shown in ***bold*** are the highest average reward for that reward pair. Thus, for experiment 1, where the rewards start as (10,-10) and then change to (25,5), the agent using metacognition level 1 that always reset the policy when an exception was detected did the best, but all the agents using metacognition did better than the agent which used none.

The last line of the table gives two numbers for each of the Chippy agents. The first number is count of times that the agent had a higher average total reward than the unaided Q-Learner. The second number is the count of times that the agent had the highest average total reward. The agent using level 3 (temporal) metacognition had the highest total of average scores that beat the unaided Q-Learner (14). The metacognitive level 1 agent that instinctively reset the learner when there was an unexpected reward had the most high scores (9).

Experiment			*Metacognitive Level*				
Num	*Begin*	*Final*	*0*	*1-Reset*	*1-Learn*	*2*	*3*
1	10,-10	25,5	63743	***74644***	*73059*	*65227*	*63774*
2	10,-10	35,15	101728	***120291***	*110561*	100744	*103465*
3	10,-10	19,21	86853	*96776*	***97116***	*88047*	85545
4	10,-10	-10,10	9893	***11058***	9223	9653	*10045*
5	25,5	35,15	187830	182295	***187893***	187646	187604
6	25,5	19,21	163371	160407	*163408*	***163959***	*163653*
7	25,5	5,25	***140128***	137917	139433	139657	139396
8	25,5	-10,10	69029	***73922***	68281	68716	*69068*
9	35,15	25,5	184943	183851	***186434***	*185474*	*185755*
10	35,15	19,21	209281	206237	*209579*	*209817*	***209899***
11	35,15	15,35	232411	228773	232135	***233696***	*232938*
12	35,15	-10,10	114381	***120489***	113644	*114513*	*114852*
13	19,21	25,5	161476	160415	*162537*	*162177*	***162568***
14	19,21	35,15	210198	206124	209781	210084	***210221***
15	19,21	21,19	186685	184423	186228	***186794***	186182
16	19,21	-10,10	91907	***97821***	*92128*	91619	*92179*
17	15,35	25,5	185505	183554	***185846***	185474	*185595*
18	15,35	19,21	***210274***	206295	209791	209684	209632
19	15,35	-10,10	115471	***120760***	113341	114796	114588
20	5,25	35,15	187526	183462	***187857***	187220	187067
21	5,25	19,21	164027	159418	162980	162970	***164058***
22	5,25	-10,10	69660	***74164***	68625	69619	69022
Better / Best			2 / 2	9 / 8	11 / 5	9 / 3	14 / 4

Table 2.2. Average rewards for each Chippy Experiment by Runner

2.4 Chippy Summary

The Chippy Q-Learner was used to demonstrate the use of metacognition to aid an agent. The metacognition component monitors the actual and expected rewards received by the Q-Learner and returns one of three suggestions when the expectation that the actual and expected rewards values are equal was violated. The Q-Learner implements the metacognition's suggestion and, hopefully, obtains a higher total rewards then an unaided Q-Learner.

Different metacognition components for Chippy were presented at several different metacognitive levels. They all shared the attribute of being hardcoded and domain-specific in that they were designed to work only in the Chippy Domain. In upcoming chapters, a metacognitive component (MCL) will be presented that, while having a domain-specific portion, uses a domain-independent element in determining the response to an expectation violation.

Chapter 3

MARS ROVER LEVEL 0: BEREFT

This chapter describes the Mars Rover domain that will be used throughout the remaining chapters for examples and experiments, as well as the first of several agents that operate in the domain. Using a simple STRIPS planner (Fikes & Nilsson 1971; 1994), as was used in the early robot Shakey (Fikes 1971), the agent in this chapter can develop and execute a plan to fulfill mission objectives given to the agent.

3.1 The Mars Rover Domain

The Mars Rover domain has been used as the experimental test bed by a number of researchers (Dearden *et al.* 2004; Estlin *et al.* 2007). Coddington (2006) describes a simplified Mars Rover Domain. The landscape consists of only eight locations (way-points), and the Rover can execute just a few commands. However, the domain is rich enough (especially with a few selected additions) to serve as a test bed for agents equipped with multiple levels of metacognition. While based on the domain described in Coddington's papers, minor changes have been made to resolve inconsistencies, fill in omissions, and expand the range of experimental scenarios.

3.1.1 The Mars Rover

The agent in the Mars Rover domain is a small multi-wheeled vehicle. Its primary purpose is to explore a limited portion of the Martian surface taking photographs. It operates under the direction of a mission specialist on Earth but is equipped with some systems (e.g., planning) that let it operate somewhat independently.

3.1.2 Sensors

The Mars Rover is equipped with several sensors. These allow the Rover to determine its present location (1-8), the current battery level, the amount of memory available for photographs, whether the movement subsystem is localized, and whether the image sensor has been calibrated. The way-point is changed as the Rover moves from location to location. Energy is used as the Rover executes commands. Energy is regained by waiting (using solar panels) or by recharging at a recharge station. Memory is used by taking photos with the Image (I) command or panoramic images with the Panoramic (P) command, and memory is regained by transmitting the photos stored in memory back to Earth using the T command. Localization is True if the movement subsystem is localized. The Rover normally starts in the localized state, but it needs to perform the Localization (L) command after moving some distance (normally 500). Calibration is True if the Calibration (C) command has been successfully performed. Calibration returns to False after successfully performing an Image (I) command. The Rover normally starts with Calibration set to False.

The Rover also has an internal clock that measures elapsed time since the start of the mission and a distance sensor that records total distance traveled. The total distance traveled since the last successful Localization is also maintained, as is the time since the last Image (I) and Panoramic (P) commands. The Zero sensor just provides a reference point for a zero value. The Speed and Sleeping sensors show the Rover's current speed

Table 3.1. The Mars Rover Sensors

Sensor	*Minimum*	*Maximum*	*Type*
Zero	0	0	Integer
Energy	0	100	Integer
Memory	0	30	Integer
Waypoint	1	8	Integer
Calibrated	False	True	Boolean
Localized	False	True	Boolean
Sleeping	False	True	Boolean
Speed	0	2	Integer
TotalDistance	0	none	Integer
TotalTime	0	none	Integer
LocalDistance	0	none	Integer
TimeSincePhoto	0	none	Integer
TimeSincePanoramic	0	none	Integer

setting (Fast=2, Medium=1, or Slow=0) and indicate whether the Rover is sleeping (True) or operating (False), respectively. Table 3.1 shows the available sensors.

The units for the sensors are idealized and only the magnitudes are important in the simulation. The distances could be either in feet or meters. The times are usually thought of as seconds but could just as easily be minutes.

3.1.3 Actions

There are several actions that the Mars Rover can perform. Each command is listed with a name, an initial, its purpose, and its pre- and post-conditions. An attempt to execute a command whose pre-conditions are not met causes an error and a Wait (W) command is executed instead. The actions in this first section are the same as those described by Coddingtion (2006).

move (n) Cause the Rover to move to the specified adjacent way-point. The Rover must have enough energy to complete the movement. The energy cost to move varies depending on the distance between the way-points and the speed of the Rover. There is also a time cost for performing a move command. As with energy cost, the time cost varies with the distance between the way-points and the speed of the Rover. At medium speed, the energy cost is 8 or 10, and the time required is 16 or 20 seconds. An attempt to move to a non-adjacent way-point or without sufficient energy will cause an error and the Rover will remain in place.

image (I) This action causes the Rover to take a detailed photo image and store it in memory. Each image takes 5 memory storage units, costs 5 energy units, and takes 20 seconds. In order to successfully take an image, the Rover must have sufficient energy and memory storage available, be calibrated, and be at one of the three way-points for imaging (1, 4, and 8).

calibrate (C) In order to take detailed photographic images, the Rover must calibrate its image sensors. This can only be done at way-point 4. Calibration uses no image memory and costs only 1 energy unit, but takes 20 seconds to complete.

recharge (R) Restores the Rover's battery to full charge (recharging can be done at way-points 1 and 5). The time to complete charging depends on the current level of the battery.

panoramic (P) Take a panoramic image and store it in memory. Unlike detailed photographic images taken with the I command, a panoramic image does not need the Rover to be calibrated and they can be take at all way-points. Like the I command, the P command has an energy cost of 5 units and uses 5 memory storage units. It takes longer (30 seconds versus 20) to take a panoramic image as the camera must rotate 360 degrees.

transmit (T) Send the photos in memory to Earth. This can only be done at way-points 1 and 2. The energy and time required depends on the amount of memory used for photo and panoramic images. When the command is complete, all of the image memory is available for new images.

localize (L) In order to move at normal speed, the Rover must reset its way-point sensor periodically. Localization can only occur at way-point 3. Executing the L command requires 5 energy units and takes 20 seconds.

The actions below are additions to the Mars Rover domain that I added to allow for additional scenarios.

wait (W) This is the zen of actions. The Rover remains in place. The energy level is increased by one (up to but not beyond the maximum energy level) and it takes five seconds of elapsed time to complete.

fast (F) Set the Rover's speed to fast. This increases the energy to move between way-points but decreases the time required. Fast speed can only be used when localized. Moving at this speed may clean the wheels of any accumulated dust. It takes three energy units and two seconds to set the Rover's speed.

medium (M) Set the Rover's speed to medium. This is the optimal speed for the Rover as it uses the least amount of energy. Medium speed can only be used when localized. After localization, the Rover's speed is set to medium. It takes three energy units and two seconds to set the Rover's speed.

slow (S) Set the Rover speed to slow. This speed uses slightly more energy than medium and takes twice as long. It is the only speed available when the Rover is not localized. The speed setting is automatically set to slow if the Rover is not localized. It takes three energy units and two seconds to set the Rover's speed.

Table 3.2. The Mars Rover Actions

Action	*Code*	*Energy*	*Memory*	*Time*
move	n	varies	0	varies
image	I	5	5	20
calibrate	C	1	0	20
recharge	R	to max	0	varies
panoramic	P	5	5	30
transmit	T	varies	to max	varies
localize	L	5	0	20
wait	W	gain 1	0	5
slow	S	3	0	2
medium	M	3	0	2
fast	F	3	0	2
blow	B	20	0	10
diagnose	D	20	0	10
sleep	Z	gains	0	varies

blow (B) Attempt to blow the dust from the Rover's wheels and from the rotary mount for the panoramic camera. This action can only be done at way-point 1. It requires 20 energy units and takes 10 seconds.

diagnose (D) Run a sensor diagnostic. This may correct problems with the energy, memory, distance, way-point or time sensors. If there were problems with multiple sensors, zero, one, or more may be corrected. The D command can be done at any way-point. It requires 20 energy units and takes 10 seconds.

sleep (Z) Enter sleep mode. The Rover stops all actions and waits for a command from Mission Control. The battery recharges in sleep mode faster than during a wait (W) command.

Table 3.2 shows the cost in energy, memory, and time to execute each action.

Table 3.3. Actions that can only occur at particular locations

	Location							
Action	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
Image	X			X				X
Calibrate				X				
Recharge	X				X			
Transmit	X	X						
Localize			X					
Blow	X							

Certain actions can only take place at specific locations. For example, there are only two places where the Rover can recharge its batteries. Also, there are only two places where the Rover can transmit to Earth and empty the photo memory. There are only single locations for calibrating, localizing, and blowing dust from the Rover. Finally, the detailed images for science experiments can only be taken at three locations, whereas panoramic photographs can be taken anywhere. Table 3.3 details which actions can occur at which way-point locations.

Only slow movement is allowed when the Rover is not localized. Taking an image requires both available photo memory and that the image sensor be calibrated.

3.1.4 Commands

Mission Control (or the cognitive agent) can direct the Rover to execute one or more actions using the letter codes from Table 3.2. Movement commands require the number of the adjacent way-point. The Mars Rover simulator provides a point and click interface for entering these basic commands. Commands that cannot be executed (not enough energy, not at the proper way-point) are replaced by the wait (W) command. When the Rover has no more commands to process, it gives itself the sleep (Z) command and waits for more

Table 3.4. Base energy costs to move between way-points (elapsed time is twice the energy cost)

	To Way-point							
From	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
1		8						
2	8		10					
3		10			10		10	8
4					10			
5			10	10		8		
6					8			
7			10					10
8			8				10	

commands to be sent.

3.1.5 The Landscape

The portion of Mars in which the Rover operates consists of eight connected locations as shown in figure 3.1. The same information is presented in tabular form in table 3.4. The number between connected way-points in figure 3.1 (and in table 3.4) is the energy cost to travel between the two way-points when the Rover is running at medium speed and the wheels are free of dust. The travel time is usually twice the energy cost when the Rover is running at medium speed. When running at fast speed, the energy cost is doubled but the travel time is halved. When the Rover is running at slow speed, the travel time is doubled and it takes an additional 15% energy. Having dust on the wheels may add 15 or 30% to the travel time.

Table 3.5. Executing the commands "2354C538I" starting at way-point 1 and time 0

At	*WP*	*CMD*	*NRG*	*MEM*	*TIME*	*DIST*
0	1	2@1	8/92	0/30	16/16	8/8
16	2	3@2	10/82	0/30	20/36	10/18
36	3	5@3	10/72	0/30	20/56	10/28
56	5	4@5	10/62	0/30	20/76	10/38
76	4	C@4	1/61	0/30	20/96	0/38
96	4	5@4	10/51	0/30	20/116	10/48
116	5	3@5	10/41	0/30	20/136	10/58
136	3	8@3	10/31	0/30	20/156	10/68
156	8	I@8	5/26	5/25	20/176	0/68

3.1.6 Commanding the Basic Mars Rover

As a sample of the basic Mars Rover operation, the Rover is instructed to move from way-point 1 to way-point 4 for calibration and from there to way-point 8 to take a scientific image. The basic Mars Rover has to be given each instruction to move from each way-point to the next. The command sequence is "2354C538I". Table 3.5 show the Rover information at each step in the execution of these commands.

The first two columns have the time and way-point locations before the execution of the command in the third column. The fourth and fifth columns have the energy and memory used (or gained) during execution of the command along with the amounts available after the step. The final two columns have the time required for the command and the distance traveled (if any) along with accumulating totals.

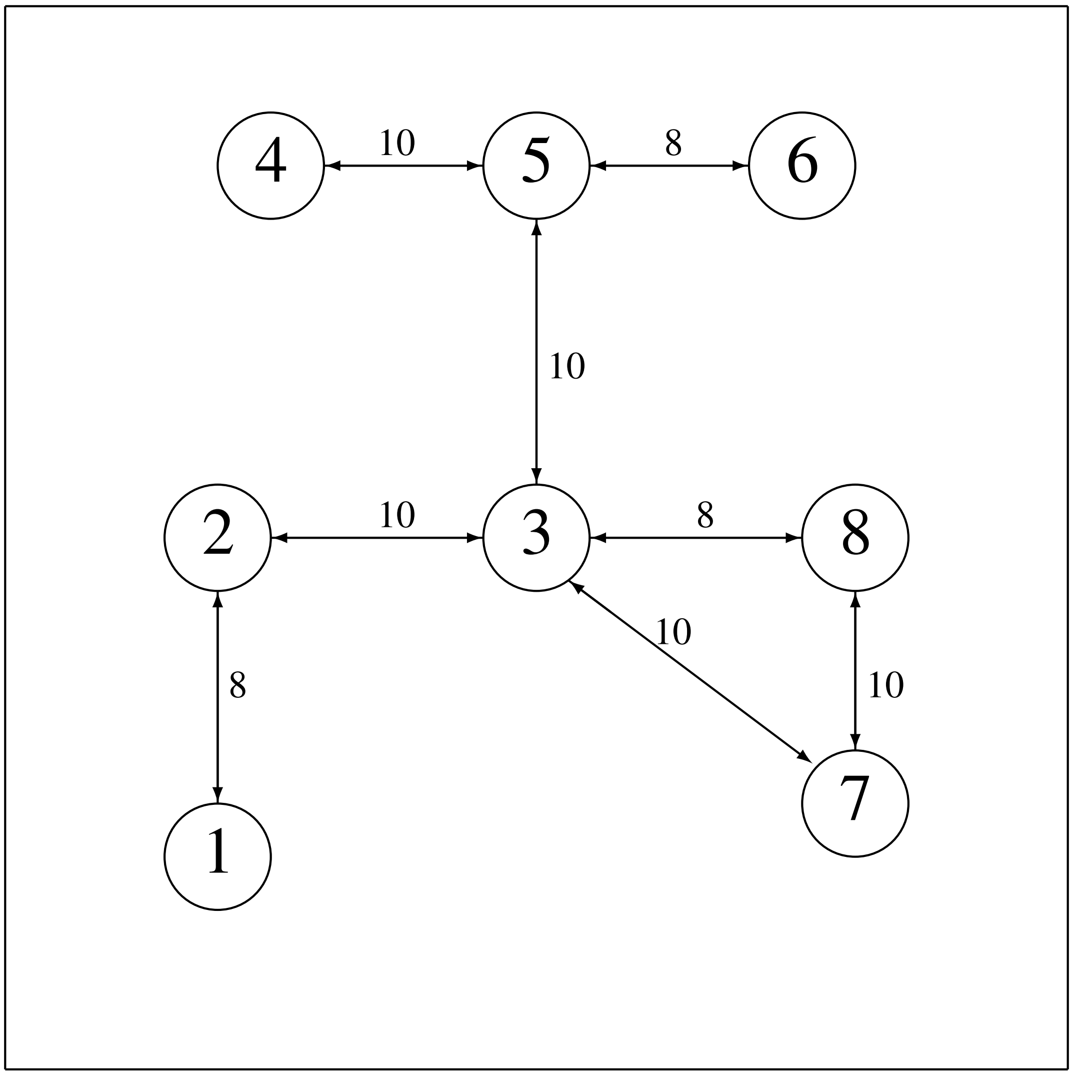


FIG. 3.1. Waypoints and connections

3.2 Mars Domain Perturbations

This section defines several perturbations in the context of software agents and gives sample perturbations in the Mars Rover domain. The basic Rover is not well equipped to handle most of the perturbations. The metacognition systems described later in this chapter and in the next two will enable the Rover to cope with the majority of them.

3.2.1 Perturbations

The agent failures described in the introduction were not caused by a failure of the designers or in the execution of that design. In all the cases, the agent encountered a change in the environment that had not been provided for in the agent.

The following are some of the problems that the Rover might encounter in the Mars Rover domain. This is not an exhaustive list of the things that could happen. A devious mind with a little time could fill several pages with possible perturbations. Perturbations used in the experiments for this dissertation are indicated with [Pn] where n is 1 to 9.

3.2.2 Recharging Problems

Loss of recharge station One (or both) of the two recharge points (way-points 1 and 5) stops functioning. The Rover can attempt to recharge at the way-point, but after 10 time units, the charging ends with no energy restored.

Longer charging time One (or both) of the two recharge points (way-points 1 and 5) start recharging at a slower rate. It takes twice as long to recharge as normal.

Partial charging [P1] One (or both) of the two recharge points (way-points 1 and 5) only charges the Rover for 30 energy units before stopping. Each subsequent attempt to recharge will charge an additional 30 energy units.

Probabilistic charging [P4] The recharge action (R) only restores the Rover to full capacity a percentage of the time (75%). The remainder of the time it only partially recharges the Rover's battery.

3.2.3 Battery Problems

Reduced capacity [P2] The battery's energy capacity is reduced to 75 from 100.[1] Charging attempts can only restore the battery to 75 units.

Leakage The battery loses power at the rate of one unit per 10 minutes in addition to the energy to implement the requested action.

3.2.4 Calibration Problems

Longer calibration time [P3] It takes twice as long to calibrate as expected.

Probabilistic calibration The calibration action (C) only sets calibration to True a percentage of the time (75%).

Recharge loses calibration [P5] Recharging the Rover causes it to reset calibration to False.

3.2.5 Localization Problems

Longer localization time It takes twice as long to localize as expected.

Probabilistic localization The localization action (L) only sets localization to True a percentage of the time (90% or 75%).

Recharge loses localization Recharging the Rover causes it to reset localization to False.

[1]Some of the examples use a Rover with maximum energy of 200.

3.2.6 Navigation Problems

Path energy change The energy required to travel between two specific way-points changes.

Path time change [P6] The time required to travel between two specific way-points changes.

Navigation energy change The energy required to travel between every two connected way-points changes.

Navigation time change The time required to travel between every two connected way-points changes.

Path blocked [P7] There is no longer a path between two specific way-points.

Dirty axle Movement takes longer and requires more energy until the axles are cleaned by running at high speed or blowing them clean.

Speed requirement A specific speed (Fast, Medium, or Slow) is required to move between two specific way-points.

3.2.7 Imaging Problems

Imaging energy change The energy required to take an image changes (generally more).

Imaging time change The time required to take an image changes (generally longer).

Probabilistic imaging change Some attempts to take an image fail to store the image in memory. The energy cost is the same for a failing image command as it is for a successful one.

3.2.8 Panoramic Problems

Panoramic energy change The energy required to take a panoramic image changes.

Panoramic time change The time required to take a panoramic image changes.

Panoramic imaging change Some attempts to take a panoramic image fail to store the image in memory. The energy cost is the same for a failing image command as it is for a successful one.

Dirty panoramic rotator [P8] The camera rotator for the panoramic images gets clogged with dirt and will not work until blown clean.

3.2.9 Sensor Problems

Noisy sensor [P9] A sensor provides a value +/- a small amount of the true value.

Probabilistic sensor A sensor provides the correct value only some of the time.

Fixed sensor A sensor provides only a single (usually incorrect) value.

3.3 Level 0 Bereft: Planning Agent Without Metacognition

Adding a goal-oriented planner to the basic Mars Rover allows Mission Control to send higher-level requests to the Rover and have them executed. This section describes the requests that the Rover can accept, how those requests are translated into actions, and gives examples of successful and unsuccessful scenarios.

3.3.1 Commands

In addition to the single letter/number commands that the basic Rover knows how to execute, the level 0 Mars Rover can also accept goals to be performed listed in the first column of the STRIPS Table 3.6. Goals that are to be achieved concurrently are separated by commas. Goals that are to be achieved serially are separated by semi-colons. For example, "TookImage2,TookImage3;Transmit" would first cause the Rover to generate and execute a plan that would take images at way-points 4 and 8.[2] After that, the Rover would generate and execute a plan that transmitted the images back to Earth.

Once a plan to satisfy a goal or goals is generated, it is executed step by step until its completion. As with the basic Rover, commands that cannot be executed (not enough energy, not at the proper way-point) are replaced by the wait (W) command. When the Rover has no more commands to process, it gives itself the sleep (Z) command and waits for more commands to be sent.

The Mars Rover simulator I developed provides a command line and a point-and-click interface for entering both the basic commands and the STRIPS goals. The command line interface is the one used to command the Rover for the experiments. The point-and-click graphic user interface is mainly for demonstrations.

3.3.2 Planner

The Rover uses a ground version of the classic STRIPS (Ghallab, Nau, & Traverso 2004). This planner is sufficient to take goals such as TakeImage1, Calibrate, etc. and turn them into a sequence of actions that the Rover can execute.

The STRIPS operator table is shown in Figure 3.7. The top half of the table deals with moving from way-point to way-point across the Martian terrain. The rest of it handles the location and calibration preconditions for the operations that require them. The final column in the table, Inverse, is used to keep plan generation from including an operation and then trying to immediately use the inverse of that operation. For example, this prevents

[2]Image location 1 is at way-point 1, Image2 is at 4 and Image3 at 8.

Table 3.6. Goals for the Mars Rover with planning

Goal	*Description*
Scientific Images	
TookImage1	Take a photo image at way-point 1
TookImage2	Take a photo image at way-point 4
TookImage3	Take a photo image at way-point 8
Panoramic Images	
TookPanoramic1	Take a panoramic image at way-point 1
TookPanoramic2	Take a panoramic image at way-point 2
TookPanoramic3	Take a panoramic image at way-point 3
TookPanoramic4	Take a panoramic image at way-point 4
TookPanoramic5	Take a panoramic image at way-point 5
TookPanoramic6	Take a panoramic image at way-point 6
TookPanoramic7	Take a panoramic image at way-point 7
TookPanoramic8	Take a panoramic image at way-point 8
Navigation	
Goto1	Move the Rover to way-point 1
Goto2	Move the Rover to way-point 2
Goto3	Move the Rover to way-point 3
Goto4	Move the Rover to way-point 4
Goto5	Move the Rover to way-point 5
Goto6	Move the Rover to way-point 6
Goto7	Move the Rover to way-point 7
Goto8	Move the Rover to way-point 8
Other	
Transmitted	Transmit images to Earth
Recharged	Recharge the Rover's battery
Localized	Ensure that the Rover is localized
Calibrated	Ensure that the Rover is calibrated
Extended	
Fast	Set movement speed to Fast
Medium	Set movement speed to Medium
Slow	Set movement speed to Slow
BlowCmplt	Blow air to movement and panoramic motors
DiagCmplt	Perform a sensor diagnosis
Wait	Perform a wait action
Sleep	Shutdown the Rover

trying a plan that has "2@1" directly following "1@2". The loops "8@3,7@8,3@7" and "7@3,8@7,3@8" are also deleted from any partial or generated plan. These little bits of heuristics greatly reduce the time to generate a plan that has to move the Rover.

Note that energy, localization, and photo memory constraints are not modeled with this simple STRIPS plan. Having enough energy is required for all commands except recharge (R), wait (W), and sleep (Z) but the STRIPS planner doesn't take this into account. Nor does the planner ensure that there is enough available photographic memory before adding a P or I command to the plan. It would be the responsibility of the Mission Controller not to send goals that generate plans that exceed the capabilities of the Rover. Metacognition will be offered later in this chapter as an alternate approach to handling the Rover's limited resources.

3.3.3 Sample Commands and Plans

If the Rover is at way-point 1, the command "TookImage2,TookImage3" could generate the plan: "2@1; 3@2; 5@3; 4@5; C@4; I@4; C@4; 5@4; 3@5; 8@3; I@8" if it chooses to satisfy the "TookImage2" goal first. If the STRIPS planner were to non-deterministically choose to satisfy "TookImage3" first, the plan would be longer as it would have to go to way-point 4 for the initial calibration and then return there after taking the image at way-point 8: "2@1; 3@2; 5@3; 4@5; C@4; 5@4; 3@5; 8@3; I@8; 3@8; 5@3; 4@5; C@4; I@4".

By default, the Rover tries several times to generate a working plan, keeping the shortest plan generated for execution. While the difference in the two plans above is only two steps, as more goals are specified, the differences in the best and worst plans generated grow as well.

3.3.4 Plan Execution

Having a Mars Rover that can translate goals into a series of commands allows Mission Control to issue more compact instructions but doesn't relieve them of ensuring that the Rover doesn't exceed its memory or energy constraints. As the Rover is given more goals, the plans generally get longer and the likelihood of draining the battery or filling memory increases. This section has the Rover generating and executing plans for grand photographic and panoramic tours which nearly or do exceed the Rover's limited resources.

The photographic tours require visiting the three way-points of scientific interest, taking photographic images there, and transmitting them back to Earth. The Rover is given the goals in two parts to ensure that the transmission of the images is done after all three images are taken: "TookPhoto1,TookPhoto2,TookPhoto3;Transmitted". Table 3.8 traces the Rover's successful execution of the twenty-one step plan. In taking three photographic images, the Rover used half of its 30 units of memory. But taking those images ran down the battery to only 5% (39 units of 200 total). Luckily the Rover had enough power to complete the photo tour by transmitting the images back to Earth.

The second example for the level 0 Rover has it going to each of the eight way-points and taking a panoramic image at each. The eight first goals (TookPanoramic1, TookPanoramic2, TookPanoramic3, TookPanoramic4, TookPanoramic5, TookPanoramic6, TookPanoramic7, TookPanoramic8) requires an eighteen step plan: P@1; 2@1; P@2; 3@2; 5@3; 4@5; P@4; 5@4; P@5; 6@5; P@6; 5@6; 3@5; P@3; 7@3; P@7; 8@7; P@8.

The first fifteen actions were executed successfully as shown in Table 3.9. The taking of the first six panoramic images exhausted the Rover's image memory and caused it to fail on the sixteenth step (P@7) with the error: "Insufficient memory: had 0 needed 5". If the Rover had been outfitted with more memory (40 units instead of 30), it could have taken the panoramic image at way-point 7, and traveled to way-point 8 for the final image. But even with the extra memory, the Rover wouldn't have enough energy to transmit the panoramic images back to Earth.

3.3.5 Resource Aware Planning

The Mars Rover operating at MCL level 0 (Bereft) can fail if it is given a string of commands that tax its limited energy or photograph memory storage. Its only recourse is to wait and receive further orders from its controllers. A more robust planner that takes into consideration the energy and memory limitations could generate plans that include sufficient recharges and transmits. Such resource-aware planners can be found in Ghallab, Nau, and Traverso (2004). The execution of a plan with additional actions to recharge the battery and transmit the images as necessary to free photographic memory is shown in Table 3.10.

3.3.6 Resource Aware Planning with Perturbations

The carefully crafted plan shown in Table 3.10 can fail in a perturbed Martian environment. For example, assume that the recharging circuit has been damaged and stops the recharging cycle prematurely. It will only replenish the battery a maximum of 30 energy units.[3] When such a perturbation is added to the environment, the panoramic tour now fails as seen in Table 3.11. The plan fails even sooner (Table 3.12) with a perturbation which limits the capacity of the battery to 75 units instead of the usual 100 energy units.[4]

[3]This is perturbation 1 (P1) of the experiments.

[4]P2 of the experiments.

3.3.7 Limitations and Improvements

With a planner that doesn't account for resources, the level 0 Mars Rover agent is limited to goals that can be quickly achieved before resources (in this case, memory and energy) are depleted. A more robust planner can overcome this limitation by inserting actions that replenish the resources. However, these augmented plans can't handle situations where the environment is perturbed beyond the environment model that governed the planning.

The next section will explore adding Instinctive Metacognition to allow the simple planner used in the Rover to handle the energy and memory resource limitations. The Instinctive Metacognition is also capable of handling some of the perturbations that cause purely plan-based (Metacognition level 0) Rovers to fail.

Table 3.7. The STRIPS table for the Mars Rover

OP	*Preconditions*	*Add*	*Delete*	*Inverse*
1@2	AtWP2	AtWP1,Goto1	AtWP2	2@1
2@1	AtWP1	AtWP2,Goto2	AtWP1	1@2
2@3	AtWP3	AtWP2,Goto2	AtWP3	3@2
3@2	AtWP2	AtWP3,Goto3	AtWP2	2@3
3@5	AtWP5	AtWP3,Goto3	AtWP5	5@3
3@7	AtWP7	AtWP3,Goto3	AtWP7	7@3
3@8	AtWP8	AtWP3,Goto3	AtWP8	8@3
4@5	AtWP5	AtWP4,Goto4	AtWP5	5@4
5@3	AtWP3	AtWP5,Goto5	AtWP3	3@5
5@4	AtWP4	AtWP5,Goto5	AtWP4	4@5
5@6	AtWP6	AtWP5,Goto5	AtWP6	6@5
6@5	AtWP5	AtWP6,Goto6	AtWP5	5@6
7@3	AtWP3	AtWP7,Goto7	AtWP3	3@7
7@8	AtWP8	AtWP7,Goto7	AtWP8	8@7
8@3	AtWP3	AtWP8,Goto8	AtWP3	3@8
8@7	AtWP7	AtWP8,Goto8	AtWP7	7@8
I@1	AtWP1,Calibrated	TookImage1	Calibrated	
I@4	AtWP4,Calibrated	TookImage2	Calibrated	
I@8	AtWP8,Calibrated	TookImage3	Calibrated	
P@1	AtWP1	TookPanoramic1		
P@2	AtWP2	TookPanoramic2		
P@3	AtWP3	TookPanoramic3		
P@4	AtWP4	TookPanoramic4		
P@5	AtWP5	TookPanoramic5		
P@6	AtWP6	TookPanoramic6		
P@7	AtWP7	TookPanoramic7		
P@8	AtWP8	TookPanoramic8		
R@1	AtWP1	Recharged		R@1
R@5	AtWP5	Recharged		R@5
T@1	AtWP1	Transmitted		T@1
T@2	AtWP2	Transmitted		T@2
C@4	AtWP4	Calibrated		C@4
L@3	AtWP3	Relocalized		L@3
F		Fast	Slow,Medium	
M		Medium	Fast,Slow	
S		Slow	Fast,Medium	
B@1	AtWP1	BlowCmplt		
D		DiagCmplt		
W		Wait		
Z		Sleep		

Table 3.8. Executing the plan for photo tour (TookPhoto1, TookPhoto2, TookPhoto3; Transmitted)

At	*WP*	*CMD*	*NRG*	*MEM*	*TIME*	*DIST*
Rover with extra energy: 200						
0	1	2@1	8/192	0/30	16/16	8/8
16	2	3@2	10/182	0/30	20/36	10/18
36	3	5@3	10/172	0/30	20/56	10/28
56	5	4@5	10/162	0/30	20/76	10/38
76	4	C@4	1/161	0/30	20/96	0/38
96	4	5@4	10/151	0/30	20/116	10/48
116	5	3@5	10/141	0/30	20/136	10/58
136	3	8@3	10/131	0/30	20/156	10/68
156	8	I@8	5/126	5/25	20/176	0/68
176	8	3@8	10/116	0/25	20/196	10/78
196	3	5@3	10/106	0/25	20/216	10/88
216	5	4@5	10/96	0/25	20/236	10/98
236	4	C@4	1/95	0/25	20/256	0/98
256	4	I@4	5/90	5/20	20/276	0/98
276	4	C@4	1/89	0/20	20/296	0/98
296	4	5@4	10/79	0/20	20/316	10/108
316	5	3@5	10/69	0/20	20/336	10/118
336	3	2@3	10/59	0/20	20/356	10/128
356	2	1@2	8/51	0/20	16/372	8/136
372	1	I@1	5/46	5/15	20/392	0/136
392	1	T@1	7/39	+15/30	15/407	0/136

Table 3.9. Executing (partially) the plan for panoramic tour (TookPanoramic1, TookPanoramic2, TookPanoramic3, TookPanoramic4, TookPanoramic5, TookPanoramic6, TookPanoramic7, TookPanoramic8; Transmitted)

At	*WP*	*CMD*	*NRG*	*MEM*	*TIME*	*DIST*
Rover with extra energy: 200						
0	1	P@1	5/195	5/25	30/30	0/0
30	1	2@1	8/187	0/25	16/46	8/8
46	2	P@2	5/182	5/20	30/76	0/8
76	2	3@2	10/172	0/20	20/96	10/18
96	3	5@3	10/162	0/20	20/116	10/28
116	5	4@5	10/152	0/20	20/136	10/38
136	4	P@4	5/147	5/15	30/166	0/38
166	4	5@4	10/137	0/15	20/186	10/48
186	5	P@5	5/132	5/10	30/216	0/48
216	5	6@5	8/124	0/10	16/232	8/56
232	6	P@6	5/119	5/5	30/262	0/56
262	6	5@6	8/111	0/5	16/278	8/64
278	5	3@5	10/101	0/5	20/298	10/74
298	3	P@3	5/96	5/0	30/328	0/74
328	3	7@3	10/86	0/0	20/348	10/84
348	7	*P@7*	+1/87	0/0	5/353	0/84
Insufficient memory: had 0 needed 5						

Table 3.10. Executing a panoramic tour plan that handles limited memory and energy.

At	*WP*	*CMD*	*NRG*	*MEM*	*TIME*	*DIST*
0	1	P@1	5/95	5/25	30/30	0/0
30	1	2@1	8/87	0/25	16/46	8/8
46	2	P@2	5/82	5/20	30/76	0/8
76	2	3@2	10/72	0/20	20/96	10/18
96	3	5@3	10/62	0/20	20/116	10/28
116	5	4@5	10/52	0/20	20/136	10/38
136	4	P@4	5/47	5/15	30/166	0/38
166	4	5@4	10/37	0/15	20/186	10/48
186	5	P@5	5/32	5/10	30/216	0/48
216	5	6@5	8/24	0/10	16/232	8/56
232	6	P@6	5/19	5/5	30/262	0/56
262	6	5@6	8/11	0/5	16/278	8/64
278	5	R@5	+89/100	0/5	89/367	0/64
367	5	3@5	10/90	0/5	20/387	10/74
387	3	P@3	5/85	5/0	30/417	0/74
417	3	2@3	10/75	0/0	20/437	10/84
437	2	1@2	8/67	0/0	16/453	8/92
453	1	R@1	+33/100	0/0	33/486	0/92
486	1	T@1	15/85	+30/30	30/516	0/92
516	1	2@1	8/77	0/30	16/532	8/100
532	2	3@2	10/67	0/30	20/552	10/110
552	3	8@3	10/57	0/30	20/572	10/120
572	8	P@8	5/52	5/25	30/602	0/120
602	8	7@8	8/44	0/25	16/618	8/128
618	7	P@7	5/39	5/20	30/648	0/128
648	7	3@7	10/29	0/20	20/668	10/138
668	3	2@3	10/19	0/20	20/688	10/148
688	2	T@2	5/14	+10/30	10/698	0/148

Table 3.11. Execution of a panoramic tour plan that handles limited memory and energy that cannot handle a damaged recharger which only adds a maximum of 30 energy units.

At	*WP*	*CMD*	*NRG*	*MEM*	*TIME*	*DIST*
0	1	P@1	5/95	5/25	30/30	0/0
30	1	2@1	8/87	0/25	16/46	8/8
46	2	P@2	5/82	5/20	30/76	0/8
76	2	3@2	10/72	0/20	20/96	10/18
96	3	5@3	10/62	0/20	20/116	10/28
116	5	4@5	10/52	0/20	20/136	10/38
136	4	P@4	5/47	5/15	30/166	0/38
166	4	5@4	10/37	0/15	20/186	10/48
186	5	P@5	5/32	5/10	30/216	0/48
216	5	6@5	8/24	0/10	16/232	8/56
232	6	P@6	5/19	5/5	30/262	0/56
262	6	5@6	8/11	0/5	16/278	8/64
278	5	R@5	+30/41	0/5	89/367	0/64
367	5	3@5	10/31	0/5	20/387	10/74
387	3	P@3	5/26	5/0	30/417	0/74
417	3	2@3	10/16	0/0	20/437	10/84
437	2	1@2	8/8	0/0	16/453	8/92
453	1	R@1	+*30/38*	0/0	92/545	0/92
P1 limits recharging to 30 per attempt						
545	1	T@1	15/23	+30/30	30/575	0/92
575	1	2@1	8/15	0/30	16/591	8/100
591	2	3@2	10/5	0/30	20/611	10/110
611	3	*8@3*	0/5	0/30	0/611	0/110
Insufficient energy: had 5 needed 10						

Table 3.12. Execution of a panoramic tour plan that handles limited memory and energy that cannot handle a weakened battery which can only store 75 energy units.

At	*WP*	*CMD*	*NRG*	*MEM*	*TIME*	*DIST*
P2 limits battery to maximum of 75						
0	1	P@1	5/75	5/25	30/30	0/0
30	1	2@1	8/67	0/25	16/46	8/8
46	2	P@2	5/62	5/20	30/76	0/8
76	2	3@2	10/52	0/20	20/96	10/18
96	3	5@3	10/42	0/20	20/116	10/28
116	5	4@5	10/32	0/20	20/136	10/38
136	4	P@4	5/27	5/15	30/166	0/38
166	4	5@4	10/17	0/15	20/186	10/48
186	5	P@5	5/12	5/10	30/216	0/48
216	5	6@5	8/4	0/10	16/232	8/56
232	6	*P@6*	0/4	0/10	0/232	0/56
Insufficient energy: had 4 needed 5						

Chapter 4

MARS ROVER LEVEL 1: INSTINCTIVE

A hard-coded, instinctive metacognition is added to the Mars Rover presented in the previous chapter. This will allow the Level 1: Instinctive Mars Rover to operate independently and to successfully handle a wider range of problems. The general technique describe herein can be applied to a variety of domains but each implementation will be domain-specific.

4.1 Level 1 Instinctive: Planning Agent with Hard-coded Metacognition

Metacognition can be added to the Level 0: Bereft Mars Rover of the last chapter to overcome the limitations of its simple planning system and to help it deal with perturbations in the environment. There are multiple ways to accomplish this. One approach would be to use a state machine and have exceptions cause changes in state. The approach described in the following pages is both simple to describe and rich enough to show both the gains in using it and its limitations.

Coddington (2007b; 2007a) describes a Mars Rover that uses metacognition to add goals to the planner when the sensors cross preset thresholds. Each of these sensor expectations are derived from ascribing motivations to the Rover: the desire to recharge when energy is low, etc. To handle conflicts between motivations, multiple goal levels are added to the planner with a plan only being generated for goals at highest non-empty level. The sensors, thresholds, goals and goal levels are all domain-specific. The next two sections describe the motivations of the Mars Rover and how they are used.

4.2 Motivations

If the Mars Rover were alive, it would likely have a desire to remain alive.[1] For the Rover, this would translate into a desire to keep its energy level above zero. Actually, the Rover would like to keep its energy level higher so that it would still have enough energy to reach a recharge way-point before the energy level was depleted to zero. The Rover would have the expectation that the energy sensor would always be above 40 or so.[2] The value threshold for this motivation (and for all of the motivations) are domain-specific.

The level 1: Instinctive Mars Rover has six motivations that strive to keep the Rover recharged, localized, and transmitting information back to Earth. The first one, *Conserve Energy*, causes the Rover to seek a recharge point when the battery is getting low. *Acquire Data 1* and *Acquire Data 2* have the Rover taking photo and panoramic images. When the photo memory is getting full, *Communicate Image Data* has the Rover transmitting the images back to Earth. The fifth motivation, *Relocalize*, keeps the Rover localized by sending the Rover back to the re-localization point after traveling too long. The sixth and last motivation, *Slow Down*, switches the Rover into slow speed if it loses localization. This is done because the Rover can still move at slow speed when not localized. Table 4.1 lists the motivations.

Each motivation has a test to determine when the associated expectation is violated and the new goal to be added if it is. If the Rover already has the goal that is to be added,

[1]Issac Asimov would formulate this motivation as the third rule of robotics.

[2]If the Rover kept track of the amount of energy needed to reach a recharge point, the expectation value could be dynamically determined. However, using a constant is simpler and works for the Mars Rover simulation given here.

Name	*Expectation Test*	*Added Goal*	*Goal Level*
Conserve Energy	energy < 40	Recharged	Emergency
Acquire Data 1	time since image > 250	TookImage$n$ n is 1 to 3	Suggestion
Acquire Data 2	time since panoramic > 100	Panoramic$n$ n is 1 to 8	Background
Communicate Image Data	memory < 6	Transmitted	Immediate
Relocalize	distance traveled > 400	Relocalized	Immediate
Slow Down	$\neg$localized $\&$ speed $\neq$ slow	Slow	Emergency

Table 4.1. Motivations for the Mars Rover

no additional goal is added.

The tests for distance traveled [since last localization], time since [last] image, and the time since [last] panoramic can be handled by adding sensors to track these values. Or the values needed by the expectation tests can be synthesized from monitoring successful command execution and the distance traveled and mission time. This can either be done in the Rover, in the metacognition system, or in the communication layer between the two.

For *Acquire Data 1* and *Acquire Data 2*, one of a set of possible goals is randomly selected. For *Acquire Data 1* it will be TookImage1, TookImage2, or TookImage3. The goal for *Acquire Data 2* will be one of Panoramic1 through Panoramic8. If the Rover already has a TookImage$n$ or Panoramic$n$ goal then an additional one isn't assigned.

The last column, Goal Level, will be discussed in the next section.

4.3 Goal Hierarchies

Intrinsically, the various motivations of the Mars Rover are in conflict with each other:

- The energy level remains high if no activity is performed.
- The Rover stays localized if it doesn't move.
- Taking a scientific image or a panoramic photo uses memory.

Importance	*Level*	*Name*
Highest	0	Emergency
	1	Immediate
Middle	2	Command
	3	Suggestion
Lowest	4	Background

Table 4.2. Goal Hierarchy for Mars Rover

- The Rover's motivations can be in conflict with the instructions received from Earth.

These conflicts can become explicit when the Rover has to execute a long-running plan involving multiple goals.

To resolve these conflicts, the Rover has multiple goal levels. The Rover only generates a plan using the goals at the highest non-empty level.

The *Conserve Energy* motivation, which attempts to keep the Rover supplied with energy, adds its goal, Recharged, at the highest level, Emergency. Also added at the Emergency level is the Slow goal of *Slow Down*. The motivations for transmitting back to Earth (*Communicate Image Data*) and Localization (*Relocalize*) add their goals at the Immediate level. Instructions from Earth are placed in the middle Command Goals level. Goals from the Rover motivations to take detailed photographic images (*Acquire Data 1*) are added at the Suggestion level. The *Acquire Data 2* adds goals for taking panoramic images at the lowest, Background level. Such goals will only be used to generate a plan for the Rover if there are no goals at the four higher levels.

4.4 Instinctive Metacognition Plan Execution

The photographic grand tour (TookPhoto1, TookPhoto2, TookPhoto3; Transmitted) is executed by the Level 1: Instinctive Rover exactly the same as the Level 0: Bereft Rover. This is by design in that, unless there is a problem, the agent's metacognition need not be invoked. However, while the Level 0 Rover was unsuccessful in completing the panoramic tour (see Table 3.9), the Level 1 Rover successfully takes all eight panoramic images and transmits them to earth.

Table 4.3 shows the execution of the panoramic tour and the addition of new goals at various levels as the test for different motivations becomes True. The motivated Rover starts out at the same as the level 0 Rover but after executing 3@8 at time 232, the *Acquire Data 1* motivation adds the TakeImage$n$ (in this case TakeImage1) goal at the Suggestion level. Since this goal is at a higher (less important) level, the Rover continues with the Panoramic Tour. After taking a panoramic image at way-point 3, the memory is less than 6, so the *Communicate Image Data* motivation adds a "Transmitted" goal at the Immediate level. This goal is at a lower (more important) level than the remaining TakePanoramic$n$ goals (4, 5, and 6) so the Rover generates and starts executing a plan to move to a transmit location and execute a T command. After the T@2 command at time 302 satisfies the Transmitted goal, the Rover resumes the tour, creating a plan to satisfy the remaining goals at the command level (TookPanoramic4, TookPanoramic5, and TookPanoramic6). The Rover is ready to take the last panoramic image at way-point 4 at time 479 but the energy level is less than 40, triggering the *Conserve Energy* motivation to add a Recharged goal at the Emergency level. Once recharged, the Rover continues on the tour, taking the last panoramic image at way-point 4 at time 698. Since all of the goals at the Command level have been satisfied, the deferred goal of Transmitted is now active. The Rover moves to way-point 2 and executes the T@2 command at time 788. At this point, the Panoramic Tour with the deferred Transmit is complete. The Rover goes on to now starts to handle the Suggestion level goal of TakeImage1 that was added back at time 252. With the maximum energy set to 100 (much less than is needed to complete a panoramic tour) the instinctive Rover is still able to complete the tour, needing an additional two recharges.

Table 4.3. Executing (partially) the plan for panoramic tour (TookPanoramic1, TookPanoramic2, TookPanoramic3, TookPanoramic4, TookPanoramic5, TookPanoramic6, TookPanoramic7, TookPanoramic8; Transmitted)

At	*WP*	*CMD*	*NRG*	*MEM*	*TIME*	*DIST*
Rover with extra energy: 200						
0	1	P@1	5/195	5/25	30/30	0/0
30	1	2@1	8/187	0/25	16/46	8/8
46	2	P@2	5/182	5/20	30/76	0/8
76	2	3@2	10/172	0/20	20/96	10/18
96	3	8@3	10/162	0/20	20/116	10/28
116	8	7@8	8/154	0/20	16/132	8/36
132	7	P@7	5/149	5/15	30/162	0/36
162	7	3@7	10/139	0/15	20/182	10/46
182	3	8@3	10/129	0/15	20/202	10/56
202	8	P@8	5/124	5/10	30/232	0/56
232	8	3@8	10/114	0/10	20/252	10/66
Time since last Image > 250, add Sug:TookImage$n$						
252	3	P@3	5/109	5/5	30/282	0/66
Memory < 6, add Imm:Transmitted						
282	3	2@3	10/99	0/5	20/302	10/76
302	2	T@2	12/87	+25/30	25/327	0/76
Transmitted, resume panoramic tour						
327	2	3@2	10/77	0/30	20/347	10/86
347	3	5@3	10/67	0/30	20/367	10/96
367	5	P@5	5/62	5/25	30/397	0/96
397	5	6@5	8/54	0/25	16/413	8/104
413	6	P@6	5/49	5/20	30/443	0/104
443	6	5@6	8/41	0/20	16/459	8/112
459	5	4@5	10/31	0/20	20/479	10/122
Energy below 40, add Imm:Recharged						
479	4	5@4	10/21	0/20	20/499	10/132
499	5	R@5	+179/200	0/20	179/678	0/132
Recharged complete, resume Tour						
678	5	4@5	10/190	0/20	20/698	10/142
698	4	P@4	5/185	5/15	30/728	0/142
Panoramic Tour complete, add cmd:Transmitted						
728	4	5@4	10/175	0/15	20/748	10/152
748	5	3@5	10/165	0/15	20/768	10/162
768	3	2@3	10/155	0/15	20/788	10/172
788	2	T@2	7/148	+15/30	15/803	0/172
Transmitted complete, continue with TakeImage$n$ ...						

4.5 Instinctive Metacognition with Perturbation

When either perturbation P1 (recharging limited to 30 units) or P2 (battery capacity limited to 75 units) were added to the environment, the Rover with no metacognitive assistance failed to complete the panoramic tour. This occurred even when a more robust, resource-cognizant, planner was used. An instinctive Mars Rover equipped with some motivation rules can complete the tour with the P2 perturbation which reduced the memory capacity to 75. Several additional recharges are needed and it takes about 20% longer than with no perturbations, but it does succeed.

The story is not as bright when P1 (partial charging) is introduced into the environment. The Rover starts out fine, but after the first of three images of the photographic tour, it finds itself in the position where it does not have sufficient energy to travel to the next image location before needing to recharge again. A trace of the Rover failure to complete a photographic tour with the P1 perturbation is shown in Table 4.4.

There are several ways to improve the instinctive Rover's performance with the P1 perturbation.

- Instead of adding a single Recharge goal, the *Conserve Energy* motivation could add two Recharge goals.
- A second *Conserve Energy* motivation could be added, that adds a Recharge goal, if you are at a recharge way-point and the energy level is less than 100.

The first one adds a needless recharge action when there is no P1 perturbation but is mostly harmless. The additional motivation would not hurt in the unperturbed case, would fix the problem with the P1 perturbation, but would cause the Rover to remain forever at a recharge way-point if the P2, reduced capacity, perturbation was active.

The cycle of deploy, fail, patch, and re-deploy is a familiar one in computer science

Table 4.4. Rover with Level 1: Instinctive metacognition failing to execute a photographic tour with perturbation P1: Partial charging

At	*WP*	*CMD*	*NRG*	*MEM*	*TIME*	*DIST*
0	1	2@1	8/92	0/30	16/16	8/8
16	2	3@2	10/82	0/30	20/36	10/18
36	3	5@3	10/72	0/30	20/56	10/28
56	5	4@5	10/62	0/30	20/76	10/38
76	4	C@4	1/61	0/30	20/96	0/38
96	4	I@4	5/56	5/25	20/116	0/38
116	4	C@4	1/55	0/25	20/136	0/38
136	4	5@4	10/45	0/25	20/156	10/48
156	5	3@5	10/35	0/25	20/176	10/58
Energy below 40, added Recharged goal						
176	3	5@3	10/25	0/25	20/196	10/68
196	5	R@5	+30/55	0/25	75/271	0/68
Partial recharge to 55						
271	5	3@5	10/45	0/25	20/291	10/78
291	3	8@3	10/35	0/25	20/311	10/88
Needs another recharge						
311	8	3@8	10/25	0/25	20/331	10/98
331	3	5@3	10/15	0/25	20/351	10/108
351	5	R@5	+30/45	0/25	85/436	0/108
436	5	3@5	10/35	0/25	20/456	10/118
456	3	5@3	10/25	0/25	20/476	10/128
476	5	R@5	+30/55	0/25	75/551	0/128
In loop recharging and moving but never moving far enough to take another image						

and robotics. Adding instinctive metacognition to an agent is a small step in improving its autonomous capabilities; however, it is limited to the pre-programmed responses that the agent's designer includes. The next chapter explores a richer form of metacognition that will, hopefully, enable the agent to function successfully in a broader range of perturbed environments.

Chapter 5

MCL LEVEL 1: INSTINCTIVE

This is the first of three chapters, that takes my idea of multiple metacognitive levels, to deconstruct and explain the existing MCL system that I will be modifying in Chapter 8. The previous chapter described how a Level 1: Instinctive metacognition can be added to an agent using motivations. Each motivation consisted of an expectation and a repair to invoke if the expectation was violated. This chapter describes a level 1 metacognition based on the Metacognitive Loop (MCL) (Schmill *et al.* 2007). As originally defined, MCL was operating Level 3: Temporal system. It can be manipulated into operating at several metacognitive levels. MCL operating at Level 1: Instinctive will be presented here. It will instinctively suggest that the Rover discard its existing action plan and create a new plan based on the current conditions. Using MCL at Level 2: Evaluative and Level 3: Temporal will be presented in the next two chapters, respectively.

5.1 Note, Access and Guide (The NAG Cycle)

MCL consists of three phases that implement its metacognitive knowledge about problem detection, fault isolation, and corrective action for cognitive agents. These three phases correspond to the process often used by humans in which we (1) notice that something is not working, (2) make decisions about it (whether the problem is important, how likely it is to get worse in the future, if it is fixable, etc.), and then (3) implement a response based on the decisions that were made (ignore the problem, ask for help, attempt to fix the problem using trial-and-error, etc.)

MCL Note Phase The MCL process starts with the Note phase that provides the host system with a “self-awareness” component. MCL monitors the host system to detect a difference between expectations and observations. An expectation is a statement about the allowable values for a sensor. Statements such as “the mixing vat temperature will not exceed 170 degrees” and “the flow in the coolant pipe will be between 80 and 90 gallons per second” are expectations made about external sensors. An anomaly occurs when an expectation is violated. Anomalies can also be about internal host processes such as “a new plan will be generated no more than 5 seconds after a new subgoal has been made the current subgoal.” When sensor information is at odds with expected values, an anomaly is noted and MCL moves to the assess phase.

For the Chippy agent, once it found a reward, it would have an expectation that it would always find a reward with the same value at that square. After the rewards were switched, the next time the agent reached the old reward square the expectation would be violated and MCL would then try to assess the problem.

MCL Assess Phase In the Assess state, MCL attempts to determine the cause of the problem that led to the anomaly and the severity of the problem. The computation done in this phase need not be excessive. Indeed, it is the philosophy of MCL that lightweight, efficient problem analysis is better than ignoring the problem, attempting to design out every conceivable problem, or to attempt to model and monitor large portions of the world. In a Level 1: Instinctive implementation of MCL, this phase is almost nonexistent with a direct connection between the exception and the corrective action.

MCL Guide Phase The third phase of the Metacognitive Loop is Guide, where MCL attempts to guide the host system back to proper operation by offering a suggestion as to what action(s) will return the sensor values to within the limits set by the expectations. The suggestions available in this phase vary depending on the features of the host system.

A Chippy agent with MCL 1 whose reward expectation had been violated would receive (as in the hard-coded, domain-specific implementation described in Chapter 2) a predetermined suggestion such as *reset the policy*.

Once the suggestion has been made, MCL returns to the exception monitoring state. Any new exceptions will cause MCL to again enter the Note, Assess, and Guide phases of the NAG cycle.

5.2 Sensors and Observations

MCL does not observe the environment directly but has to rely on the agent to provide sensor information. The agent does not need to provide all of its sensor information to MCL. Only those sensors that are tied to expectations actually need to be defined. Even then, the agent can decided not to provide MCL with all of the observations for the defined sensors. There is little benefit of withholding information from MCL, and MCL should be given access to all of the sensor information that the agent has. In this section, the concepts of sensors and observations as used by MCL are defined.

5.2.1 Sensor Nomenclature

A “sensor” is anything that provides the agent “observations” about the agent or the environment. MCL allows the agent to define two kinds of sensors. Sensors that are native to the agent are called “self” sensors. Sensors that describe properties of objects in the world external to the agent are called “object” sensors. The main difference between these two kinds of sensors is that the observations for self sensors are always obtainable but objects may not always be observable by the agent.

As an example, consider a robot that moves from room to room. A self sensor can tell the robot which room it is in so that information is always available. But the contents (objects) of a room, observable while the robot is in the room, are not observable when the robot leaves the room and enters another room. It is likely that the objects stay in the same place and retain their size, color, etc., but this can no longer be determined by direct sensor observation.

Each sensor defined to MCL needs to be given a unique name so that the sensor and its observations can be uniquely identified.

Sensor Noise Sensors may or may not provide accurate observations for the item they are monitoring. Often, a certain leeway is allowed. For example, a weight sensor may only be accurate within two pounds.

Observations At any time, the agent can explicitly report the current observation for one or more of the sensors. Observations are also provided when closing an expectation group (defined in the next section) or invoking the NAG cycle using the MCL monitor call.

Explicit legal values or ranges of legal values may be associated with a sensor. If there is an observation for the sensor outside of the legal values or range (taking any noise profile into account) an exception is noted for the next monitor/NAG cycle.

5.3 Expectations and Exceptions

MCL only becomes active when there is a deviation between what should be and what is. "What is" is obtained from the sensor observations. "What should be" is expressed as expectations about sensor observations. Legal values and ranges for observations are one way to express expectations (see previous section).

For purposes of organizing expectations that share common characteristics (e.g., that are about the same external object) and to control when the expectations are tested for validity, expectations are assigned to expectation groups. Any number of expectation groups can be defined, each containing one or more expectations.[1] An expectation group can be related to another group in a parent:child relationship. An expectation group can be maintained for the entire life of the agent or created and closed as needed.

There are four types of expectations, based on when the expectation tests are performed.

Effect Checked only when the expectation group is closed normally. An expectation group can also be "aborted", which closes it without checking the group's expectations.

Maintenance Checked when the agent requests that MCL compare the sensor values to the expectations but not when the expectation's group is closed.

Delayed Maintenance Like Maintenance, but checking only starts after the specified time has elapsed, thus defining an expectation that should be true in the future but is not necessarily true now.

Temporal The exception is triggered if the time specified expires before the expectation's group is closed.

Expectation groups are terminated by either abortion or completion. Aborted expectation groups do not have their expectations checked. Completed expectation groups evaluate their expectations tests using the latest observations.

[1]Actually, expectation groups don't have to contain any expectations but such groups aren't very interesting or useful.

When the test in an expectation fails, it causes an exception. If the exception is the result of a failing maintenance expectation detected in monitor processing, the MCL NAG cycle is initiated. If the exception is caused by an effect expectation (e.g., one defined with EC_TAKE_VALUE) failing during the expectation group completion checks, the exception is held pending until the next monitor processing.

The Chippy agent would set an expectation that the integer value of the "REWARD_1_VALUE" would maintain its value.

5.4 MCL Implementation

The most recent implementation of MCL (MCL2) was developed for Linux in C++ by Matt Schmill and ported to Mac OS X by this author. The MCL Application Program Interface information (e.g., Tables 5.1 through 5.4) was derived from the C++ code and the API documentation in (Schmill 2009). MCL2 was designed to support metacognition level 3, Temporal. Using MCL2 as metacognition level 1, Instinctive required some minor changes to mask out the higher level MCL2 features.

5.4.1 Sensor Properties

MCL associates three properties with each sensor: data type, class, and noise profile. Optionally, legal values or ranges can also be specified.

Sensor Data Type MCL uses floating point numbers for the observation values. Integers and other discrete values will need to be mapped to floating point numbers when using MCL. The actual sensor format should be defined to MCL using the sensor data type (PROP_DT) codes given in Table 5.1. All of the Chippy sensors would be defined as DT_INTEGER as the contain location values (0-63) or reward values.

Table 5.1. Sensor data types (PROP_DT) and descriptions

Code	*Value*	*Description*
DT_INTEGER	0	Integer values (including negative)
DT_RATIONAL	1	Floating point numbers
DT_BINARY	2	Integer values (always positive)
DT_BITFIELD	3	Multiple values in one number
DT_SYMBOL	4	Discrete values

Explicit legal values or ranges of legal values may be specified for a sensor. The Chippy reward location sensors could be defined to only have a legal range of 0 to 63. Any other value would indicate a problem with the agent's location detection.

Sensor Class How the sensor is used by the agent or the origin of sensor information can be used in MCL when evaluating exceptions involving the sensors. Table 5.2 lists the values for the sensor class (PROP_SCLASS) property. The Chippy agent's reward location would be defined as SC_SPATIAL and the value sensors as SC_REWARD.

Table 5.2. Sensor classes (PROP_SCLASS) and descriptions

Code	*Value*	*Description*
SC_STATE	0	Agent condition
SC_CONTROL	1	Agent actuator
SC_SPATIAL	2	Location
SC_TEMPORAL	3	Time based
SC_RESOURCE	4	Consumable
SC_REWARD	5	Feedback
SC_AMBIENT	6	Environmental
SC_OBJECTPROP	7	Object property
SC_MESSAGE	8	Message
SC_COUNTER	9	Incrementing value
SC_UNSPEC	10	Unknown / Not specified

Sensor Noise Sensors may or may not provide accurate observations for the item they are monitoring. Often, a certain leeway is allowed. For example, a weight sensor may only be accurate within two pounds. MCL allows associating with each sensor a profile of how accurate the sensor's observations will be. Table 5.3 lists the values for the sensor class (PROP_NOISEPROFILE) property.

Table 5.3. Sensor noise profiles (PROP_NOISEPROFILE)

Code	*Value*	*Description*
MCL_NP_NO_PROFILE	0	No profile specified
MCL_NP_PERFECT	1	No error
MCL_NP_UNIFORM	2	Uniform error
MCL_NP_AUTOMATIC	0xFF	Error calculated

Expectations The list of expectation tests is given in Table 5.4.

5.5 Mars Rover Integration

To use MCL2, an agent establishes a connection either through the C++ API or using the socket/telnet interface. For demonstration purposes, the Mars Rover simulation described in the previous chapter invokes MCL2 over a TCP/IP interface.[2] An actual Mars Rover would use the direct C++ interface. The TCP/IP interface is text-based for both the requests and responses. This section shows the initialization of interface, the definition of the sensors, creating an expectation, and monitoring expectations.

[2]The default port number is 5150 (the model number of the original IBM Personal Computer).

Table 5.4. Expectations Codes and the tests

Maintenance		
Code	*Num*	*Test*
EC_STAYUNDER	0	$o < v$
EC_STAYOVER	1	$o > v$
EC_MAINTAINVALUE	2	$o = v$
EC_WITHINNORMAL	3	$v_{min} < o < v_{max}$
Temporal		
Code	*Num*	*Test*
EC_REALTIME	4	$t_{clock} < v$
EC_TICKTIME	5	$t_{tick} < v$
Effect		
Code	*Num*	*Test*
EC_GO_UP	6	$o_e > o_0$
EC_GO_DOWN	7	$o_e < o_0$
EC_NET_ZERO	8	$o_e = o_0$
EC_ANY_CHANGE	9	$o_e <> o_0$
EC_NET_RANGE	10	$v_{min} < o_e < v_{max}$
EC_TAKE_VALUE	11	$o_e = v$
EC_DONT_CARE	12	none
EC_BE_LEGAL	13	$o_e \in \{values\}$

5.5.1 Initialization

The first few interactions with MCL2, shown in Figure 5.1, establish a name for the agent (mr), the name of the ontology to be used (DW_MCL_ROVER.ont, Appendix B), and the directory for configuration files (DW_MCL_ROVER). The ontology is effectively ignored by the level 1 MCL but the interface requires that one be defined. The discussion of the ontologies is deferred to section 6.1 in the next chapter.

```
send initialize(mr)
recv ok(initialized 'mr'.)
send ontology(mr,DW_MCL_ROVER)
recv ok(ontology for mr set to DW_MCL_ROVER)
send configure(mr,DW_MCL_MARS)
recv ok(configured mr with DW_MCL_MARS.)
```

FIG. 5.1. Initializing MCL connection specifying ontology and domain

Sensor	*Default*	*MCL SC Type*	*MinMax*	*Expectation*
Zero	0	Counter	Yes	General
Energy	100	Resource	Yes	Action
Memory	30	Resource	Yes	Action
Waypoint	1	Spatial	Yes	Action
Calibrated	0	State	Yes	Action
Localized	1	State	Yes	Action
Sleeping	0	State	Yes	General
Speed	1	State	Yes	Action
TotalDistance	0	Counter	No	Action
TotalTime	0	Counter	No	Action
LocalDistance	0	Resource	No	Action
TimeSincePhoto	0	Temporal	No	General
TimeSincePanoramic	0	Temporal	No	General

Table 5.5. The Mars Rover Sensors

5.5.2 Defining Sensors

The Rover's available sensors were described in Section 3.1.2 and are listed in Table 5.5. Sensors are defined to MCL by name and given an initial value as shown in Figure 5.2. Next, the class for each sensor is defined with a setObsPropSelf command (Figure 5.3). For those sensors that have a minimum and maximum value, a setObsLegalRangeSelf command tells MCL to monitor for values outside of the acceptable range (Figure 5.4). As the last step in defining the sensors, we tell MCL the current (initial) values using updateOvservables() as shown in Figure 5.5.

5.5.3 Defining Expectations

The Mars Rover uses two expectation groups. Group 1 is for general expectations about the Rover that will be true during an entire simulation run. Expectation Group 2 is used for each action. It is created and completed as each action is performed. Figure 5.6 shows the creation of the two expectation groups.

The definition of the Group 1 expectations is shown in Figure 5.7. There is an expectation defined that should keep the time between Photographic or Panoramic Images below 1,000 seconds. In normal operation, the Rover should never go into a sleep state, nor should the Zero sensor have any value other than 0.

The action expectations are different for each action and the current state of the Rover. The costs (time, energy, and memory) for the actions are given in tables 3.2 and 3.4. Figure 5.8 shows the declaration of expectations for the initial 2@1 action. The WayPoint sensor should change to 2. The Calibration and Localization sensors should maintain their current values. The action costs 8 energy units so the Energy sensor should drop from 200 to 192. The Memory sensor should not change. The distance moved is 8 and the time required is 16 so the TotalDistance and TotalTime sensors should become 8 and 16, respectively. The last expectation is that the Rover's speed should not change.

5.5.4 Monitoring and Responses

After every action, the Rover interacts with MCL to

1. indicate that the action is done using EGComplete,
2. ask MCL to evaluate the Rover's situation, and
3. (if needed) respond to any MCL suggestions.

Figure 5.9 shows the interaction when MCL does not have a suggestion to make. The sensor values in EGComplete are the same ones for the monitor call. When MCL doesn't have anything to suggest it returns an empty response. When MCL detects an expectation violation, it can return one or more of the concrete suggestions listed in table 6.3 in the next chapter. The Instinctive Mars Rover will only be getting back a single possible concrete suggestion, CRC_NOOP. Figure 5.10 shows the interaction when MCL makes a suggestion which the Rover will use to trigger replanning.

5.5.5 Rover Response to MCL Concrete Suggestions

This section lists the responses that the Rover will make to the MCL's suggestions. For this MCL level 1 implementation, only a single *NOOP* suggestion will be returned to any and all expectation violations.

CRC_NOOP The Rover deletes the existing queue of planner actions to perform. This will cause the Rover to invoke the STRIPS planner to create a new plan using the current goals and sensor values. The Rover always returns "SuggestionImplemented" to MCL.

```
send declareObservableSelf(mr,Zero,0)
recv ok(declared 'Zero'.)
send declareObservableSelf(mr,Energy,200)
recv ok(declared 'Energy'.)
send declareObservableSelf(mr,Memory,30)
recv ok(declared 'Memory'.)
send declareObservableSelf(mr,WayPoint,1)
recv ok(declared 'WayPoint'.)
send declareObservableSelf(mr,Calibrated,0)
recv ok(declared 'Calibrated'.)
send declareObservableSelf(mr,Localized,1)
recv ok(declared 'Localized'.)
send declareObservableSelf(mr,Sleeping,0)
recv ok(declared 'Sleeping'.)
send declareObservableSelf(mr,Speed,1)
recv ok(declared 'Speed'.)
send declareObservableSelf(mr,TotalDistance,0)
recv ok(declared 'TotalDistance'.)
send declareObservableSelf(mr,TotalTime,0)
recv ok(declared 'TotalTime'.)
send declareObservableSelf(mr,LocalDistance,0)
recv ok(declared 'LocalDistance'.)
send declareObservableSelf(mr,TimeSincePhoto,0)
recv ok(declared 'TimeSincePhoto'.)
send declareObservableSelf(mr,TimeSincePanoramic,0)
recv ok(declared 'TimeSincePanoramic'.)
```

FIG. 5.2. Defining the names and initial values of the Rover's sensors to MCL

```
send setObsPropSelf(mr,Zero,prop_sclass,sc_counter)
recv ok(set prop for 'Zero'.)
send setObsPropSelf(mr,Energy,prop_sclass,sc_resource)
recv ok(set prop for 'Energy'.)
send setObsPropSelf(mr,Memory,prop_sclass,sc_resource)
recv ok(set prop for 'Memory'.)
send setObsPropSelf(mr,WayPoint,prop_sclass,sc_spatial)
recv ok(set prop for 'WayPoint'.)
send setObsPropSelf(mr,Calibrated,prop_sclass,sc_state)
recv ok(set prop for 'Calibrated'.)
send setObsPropSelf(mr,Localized,prop_sclass,sc_state)
recv ok(set prop for 'Localized'.)
send setObsPropSelf(mr,Sleeping,prop_sclass,sc_state)
recv ok(set prop for 'Sleeping'.)
send setObsPropSelf(mr,Speed,prop_sclass,sc_state)
recv ok(set prop for 'Speed'.)
send setObsPropSelf(mr,TotalDistance,prop_sclass,
  sc_counter)
recv ok(set prop for 'TotalDistance'.)
send setObsPropSelf(mr,TotalTime,prop_sclass,sc_counter)
recv ok(set prop for 'TotalTime'.)
send setObsPropSelf(mr,LocalDistance,prop_sclass,
  sc_resource)
recv ok(set prop for 'LocalDistance'.)
send setObsPropSelf(mr,TimeSincePhoto,prop_sclass,
  sc_temporal)
recv ok(set prop for 'TimeSincePhoto'.)
send setObsPropSelf(mr,TimeSincePanoramic,prop_sclass,
  sc_temporal)
recv ok(set prop for 'TimeSincePanoramic'.)
```

FIG. 5.3. Defining the property class of each of the Rover's sensors to MCL

```
send setObsLegalRangeSelf(mr,Zero,0,0)
recv ok(Added Zero range.)
send setObsLegalRangeSelf(mr,Energy,0,100)
recv ok(Added Energy range.)
send setObsLegalRangeSelf(mr,Memory,0,30)
recv ok(Added Memory range.)
send setObsLegalRangeSelf(mr,WayPoint,1,8)
recv ok(Added WayPoint range.)
send setObsLegalRangeSelf(mr,Calibrated,0,1)
recv ok(Added Calibrated range.)
send setObsLegalRangeSelf(mr,Localized,0,1)
recv ok(Added Localized range.)
send setObsLegalRangeSelf(mr,Sleeping,0,1)
recv ok(Added Sleeping range.)
send setObsLegalRangeSelf(mr,Speed,0,3)
recv ok(Added Speed range.)
```

FIG. 5.4. Defining the range of legal values of the Rover's sensors to MCL

```
send updateObservables(mr,{Zero=0,Energy=200,Memory=30,
  WayPoint=1,Calibrated=0,Localized=1,Sleeping=0,Speed=1,
  TotalDistance=0,TotalTime=0,LocalDistance=0,
  TimeSincePhoto=0,TimeSincePanoramic=0})
recv ok(update success.)
```

FIG. 5.5. Setting the initial values for the sensors

```
send declareEG(mr,1)
recv ok(expectation group declared (no parent/ref).)
send declareEG(mr,2,1,NULL)
recv ok(expectation group declared (with parent/ref).
```

FIG. 5.6. Declaring MCL expectation groups

```
send declareSelfExp(mr,1,TimeSincePhoto,ec_stayunder,1000)
recv ok(self expectation declared (1-arg).)
send declareSelfExp(mr,1,TimeSincePanoramic,
  ec_stayunder,1000)
recv ok(self expectation declared (1-arg).)
send declareSelfExp(mr,1,LocalDistance,ec_stayunder,600)
recv ok(self expectation declared (1-arg).)
send declareSelfExp(mr,1,Sleeping,ec_stayunder,1)
recv ok(self expectation declared (1-arg).)
send declareSelfExp(mr,1,Zero,ec_maintainvalue)
recv ok(self expectation declared (0-arg).)
```

FIG. 5.7. Specifying general expectations to MCL

```
send declareSelfExp(mr,2,WayPoint,ec_take_value,2)
recv ok(self expectation declared (1-arg).)
send declareSelfExp(mr,2,Calibrated,ec_maintainvalue)
recv ok(self expectation declared (0-arg).)
send declareSelfExp(mr,2,Localized,ec_maintainvalue)
recv ok(self expectation declared (0-arg).)
send declareSelfExp(mr,2,Energy,ec_take_value,192)
recv ok(self expectation declared (1-arg).)
send declareSelfExp(mr,2,Memory,ec_maintainvalue)
recv ok(self expectation declared (0-arg).)
send declareSelfExp(mr,2,TotalTime,ec_take_value,16)
recv ok(self expectation declared (1-arg).)
send declareSelfExp(mr,2,TotalDistance,ec_take_value,8)
recv ok(self expectation declared (1-arg).)
send declareSelfExp(mr,2,Speed,ec_maintainvalue)
recv ok(self expectation declared (0-arg).)
```

FIG. 5.8. Specifying expectations for 2@1 action

```
send EGcomplete(mr,2,{Zero=0,Energy=192,Memory=30,
  WayPoint=2,Calibrated=0,Localized=1,Sleeping=0,
  Speed=1,TotalDistance=8,TotalTime=16,LocalDistance=8,
  TimeSincePhoto=16,TimeSincePanoramic=16})
recv ok(EG 2 Completed.)
send monitor(mr,{Zero=0,Energy=192,Memory=30,
  WayPoint=2,Calibrated=0,Localized=1,Sleeping=0,Speed=1,
  TotalDistance=8,TotalTime=16,LocalDistance=8,
  TimeSincePhoto=16,TimeSincePanoramic=16})
recv ok([])
```

FIG. 5.9. Action expectation group completion and monitor with no suggestions

```
send monitor(mr,{Zero=0,Energy=67,Memory=15,
  WayPoint=5,Calibrated=1,Localized=1,Sleeping=0,Speed=1,
  TotalDistance=184,TotalTime=1045,LocalDistance=184,
  TimeSincePhoto=813,TimeSincePanoramic=707})
recv ok([response(type=suggestion,ref=0x00000001,
  code=crc_sensor_diag,action=true,abort=true,
  text="MCL response = (2,noop,crc_noop)")]
send suggestionImplemented(mr,1)
recv ok(Suggestion 1 Implemented.)
```

FIG. 5.10. Monitor suggestion to create a new plan (crc_noop) with the Rover responding that the suggestion was successful.

5.6 Demonstration with Perturbations

Tables 5.6 and 5.7 show a MCL Level 1 Rover coping with longer than expected calibration times (P3).

The MCL process starts when the calibration command expectation group is completed with the Rover sending:

```
send EGComplete(mr,2,{Zero=0, Energy=61, Memory=30,
  WayPoint=4, Calibrated=1, Localized=1, Sleeping=0,
  Speed=1, TotalDistance=38, TotalTime=116,
  LocalDistance=38, TimeSincePhoto=116,
  TimeSincePanoramic=116})
```

MCL compares the observations given with the expectations for the expectation group. After a calibration, the Energy decreases by 1, Memory should not increase or decrease, TotalTime should increase by 20, and Calibration should stay at 1. But since the previous TotalTime was 76, the new TotalTime should be 96, not 116. This violates the expectation declared with:

```
declareSelfExp(mr,2,TotalTime,ec_take_value,96)
```

The violation is recorded for further processing in the next MCL monitor call. It is recorded by creating a MCLFrame which contains information about the violation including a copy of the MCL Bayes network with the concrete indication of the exception inserted into the Indications ontology. A new node, *provenance:self* is linked to the *resource* and *long-of-target* nodes. This linkage was selected because the TotalTime sensor was given the resource property and the TotalTime value of 116 was larger than the target level 96 specified in the expectation. The probabilities of these three nodes are set to 1.

The Rover completes the action execution cycle by calling the MCL monitor with:

```
send monitor(mr,{Zero=0, Energy=61, Memory=30, WayPoint=4,
  Calibrated=1, Localized=1, Sleeping=0, Speed=1,
  TotalDistance=38, TotalTime=116, LocalDistance=38,
  TimeSincePhoto=116, TimeSincePanoramic=116})
```

MCL will analyze the problem, and provide guidance in the form of the “CRC_NOOP” suggestion.

For the longer calibration time perturbation (P3), replanning is sufficient to complete the tour. A MCL Level 1 Rover has less success when the perturbation is reduced recharging (P1) as shown in Table 5.8 et al..

At	*WP*	*CMD*	*NRG*	*MEM*	*TIME*	*DIST*
Initial plan of 2354CIC538I354C5321I						
0	1	2@1	8/92	0/30	16/16	8/8
16	2	3@2	10/82	0/30	20/36	10/18
36	3	5@3	10/72	0/30	20/56	10/28
56	5	4@5	10/62	0/30	20/76	10/38
76	4	C@4	1/61	0/30	40/116	0/38
Calibration took long MCL suggests 1:crc_noop Setting plan to IC538I354C5321I						
116	4	I@4	5/56	5/25	20/136	0/38
136	4	C@4	1/55	0/25	40/176	0/38
Calibration took long MCL suggests 1:crc_noop Setting plan to 538I354C5321I						
176	4	5@4	10/45	0/25	20/196	10/48
196	5	3@5	10/35	0/25	20/216	10/58
Energy below 40, added Recharged goal Setting plan to 5R						
216	3	5@3	10/25	0/25	20/236	10/68
236	5	R@5	+75/100	0/25	75/311	0/68
Recharged, resuming previous goals Setting plan to 38I354C5321I						
311	5	3@5	10/90	0/25	20/331	10/78
331	3	8@3	10/80	0/25	20/351	10/88
351	8	I@8	5/75	5/20	20/371	0/88
371	8	3@8	10/65	0/20	20/391	10/98
391	3	5@3	10/55	0/20	20/411	10/108
411	5	4@5	10/45	0/20	20/431	10/118
431	4	C@4	1/44	0/20	40/471	0/118
Calibration took too long MCL suggests 1:crc_noop Setting plan to 5321I						
471	4	5@4	10/34	0/20	20/491	10/128
Energy below 40, added Recharged goal Setting plan to R						
491	5	R@5	+66/100	0/20	66/557	0/128
Continued in Table 5.7						

Table 5.6. MCL Level 1 Rover executing a photographic tour with perturbation P3 (longer calibration time) with replanning (Part 1).

At	*WP*	*CMD*	*NRG*	*MEM*	*TIME*	*DIST*
Recharged, resuming previous goals Setting plan to 321I						
557	5	3@5	10/90	0/20	20/577	10/138
577	3	2@3	10/80	0/20	20/597	10/148
597	2	1@2	8/72	0/20	16/613	8/156
613	1	I@1	5/67	5/15	20/633	0/156
All images taken, adding Transmitted goal Setting plan to T						
633	1	T@1	7/60	+15/30	15/648	0/156
Only remaining goal is TookPanoramic8 Setting plan to 238P						
648	1	2@1	8/52	0/30	16/664	8/164
664	2	3@2	10/42	0/30	20/684	10/174
684	3	8@3	10/32	0/30	20/704	10/184
Energy below 40, added Recharged goal Setting plan to 35R						
704	8	3@8	10/22	0/30	20/724	10/194
724	3	5@3	10/12	0/30	20/744	10/204
744	5	R@5	+88/100	0/30	88/832	0/204
Recharged, resuming previous goals Setting plan to 38P						
832	5	3@5	10/90	0/30	20/852	10/214
852	3	8@3	10/80	0/30	20/872	10/224
872	8	P@8	5/75	5/25	30/902	0/224
Completed photographic tour with added panoramic goal						

Table 5.7. MCL Level 1 Rover executing a photographic tour with perturbation P3 (longer calibration time) with replanning (Part 2).

5.7 Limitations

While an instinctive MCL can handle some problems, there are situations where an instinctive MCL (that can make only a single suggestion) cannot provide adequate assistance to the host system. Tables 5.8 and 5.9 show a MCL Level 1 Rover attempting to complete a three photo tour with the reduced recharge (P1) perturbation. Each time the energy level drops below 40, the Rover adds a Recharged goal. The perturbed R action only adds 30 energy units so the Rover's energy level is below the 100 energy units specified in the MCL expectation. MCL always makes the 'NOOP' suggestion and then the Rover replans. The Rover is never able to take any of the three photos before getting into a state where it is not able to accumulate enough energy to move to the next photographic location.

Table 5.8. MCL Level 1: Instinctive Rover executing a photographic tour with perturbation P1: Partial recharge with replanning (Part 1).

At	*WP*	*CMD*	*NRG*	*MEM*	*TIME*	*DIST*
Setting initial plan to 2354C538I354CIC5321I						
0	1	2@1	8/92	0/30	16/16	8/8
16	2	3@2	10/82	0/30	20/36	10/18
36	3	5@3	10/72	0/30	20/56	10/28
56	5	4@5	10/62	0/30	20/76	10/38
76	4	C@4	1/61	0/30	20/96	0/38
96	4	5@4	10/51	0/30	20/116	10/48
116	5	3@5	10/41	0/30	20/136	10/58
136	3	8@3	10/31	0/30	20/156	10/68
Energy below 40, added Recharged goal Setting plan to 35R						
156	8	3@8	10/21	0/30	20/176	10/78
176	3	5@3	10/11	0/30	20/196	10/88
196	5	R@5	+30/41	0/30	89/285	0/88
Recharge only added 30 units MCL suggests 1:crc_noop New plan is 38I354CIC5321I						
285	5	3@5	10/31	0/30	20/305	10/98
Energy below 40, added Recharged goal Setting plan to 5R						
305	3	5@3	10/21	0/30	20/325	10/108
325	5	R@5	+30/51	0/30	79/404	0/108
Recharge only added 30 units MCL suggests 1:crc_noop Setting plan to 38I354CIC5321I						
404	5	3@5	10/41	0/30	20/424	10/118
424	3	8@3	10/31	0/30	20/444	10/128
Energy below 40, added Recharged goal Setting plan to 35R						
444	8	3@8	10/21	0/30	20/464	10/138
464	3	5@3	10/11	0/30	20/484	10/148
484	5	R@5	+30/41	0/30	89/573	0/148
Continued in Table 5.9						

Table 5.9. MCL Level 1: Instinctive Rover executing a photographic tour with perturbation P1: Partial recharge with replanning (Part 2).

At	*WP*	*CMD*	*NRG*	*MEM*	*TIME*	*DIST*
Recharge only added 30 units MCL suggests 1:crc_noop Setting plan to 38I354CIC5321I						
573	5	3@5	10/31	0/30	20/593	10/158
Energy below 40, added Recharged goal Setting plan to 5R						
593	3	5@3	10/21	0/30	20/613	10/168
613	5	R@5	+30/51	0/30	79/692	0/168
Recharge only added 30 units MCL suggests 1:crc_noop Setting plan to 38I354CIC5321I						
692	5	3@5	10/41	0/30	20/712	10/178
712	3	8@3	10/31	0/30	20/732	10/188
Energy below 40, added Recharged goal Setting plan to 35R						
732	8	3@8	10/21	0/30	20/752	10/198
752	3	5@3	10/11	0/30	20/772	10/208
772	5	R@5	+30/41	0/30	89/861	0/208
Recharge only added 30 units MCL suggests 1:crc_noop Setting plan to 38I354CIC5321I						
861	5	3@5	10/31	0/30	20/881	10/218
Energy below 40, added Recharged goal Setting plan to 5R						
881	3	5@3	10/21	0/30	20/901	10/228
901	5	R@5	+30/51	0/30	79/980	0/228
Recharge only added 30 units MCL suggests 1:crc_noop Setting plan to 38I354CIC5321I						
980	5	3@5	10/41	0/30	20/1000	10/238
1000	3	8@3	10/31	0/30	20/1020	10/248
Energy below 40, added Recharged goal Setting plan to 35R And so on and so on and so on ...						

Chapter 6

MCL LEVEL 2: EVALUATIVE

This is the second of three chapters, that takes my idea of multiple metacognitive levels, to deconstruct and explain the existing MCL system that I will be modifying in Chapter 8. This chapter describes the Level 2: Evaluative version of MCL, that uses Bayesian inference when analyzing the reasoning for an expectation failure and the appropriate suggestion to make.

6.1 Ontologies (Indication, Failure, and Response)

For MCL to serve as a general-purpose tool for the brittleness problem for cognitive systems, it should be able to perform its Note, Assess, and Guide phases without needing extensive tailoring for each domain (Schmill *et al.* 2008; 2011). MCL should be able to reason using mainly abstract, domain-neutral concepts to determine why a system is failing and how to cope with the problem. To support this, three ontologies were created. Each of the three ontologies is used by a different phase of MCL (see Table 6.1).

The Indications ontology is used in the Note phase when sensor input shows that an expectation has been violated. The Assess phase uses the Failure ontology to determine likely causes of the violated expectations. Once likely causes of the failure have been identified, the Guide phase uses the Response ontology to determine appropriate response to the failure.

Table 6.1. Ontologies used with NAG cycle

Phase	*Ontology*
Note	Indications
Assess	Failure
Guide	Response

Elements within each ontology are linked to others in the ontology to show an “is-a” relationship. For example, the “sensor not responding” node in the Failure ontology is connected to the “sensor failure” node to show that “sensor not responding” is a type of “sensor failure.” Elements in one ontology may also be linked to elements in a different ontology to show a possible “cause-and-effect” or “problem-solution” relationship. The general pattern of ontology linkage is shown in Figure 6.1. This figure also shows how expectations are linked to the Indications ontology elements and the elements of the Response ontology lead to suggestions that MCL gives to the host system.

The sensors and expectations are part of the “concrete” realm of the host system. Processing by MCL moves from the concrete expectations to the Indications, Failure, and Response ontologies, and then back to the concrete suggestions implemented by the host system. Figure 6.1 shows the division between the concrete and abstract processing. The next sections expand on this process, going into each of the three ontologies in greater detail.

6.1.1 Indications

The Indications ontology is comprised of three types of nodes (core, sensor, and divergence) arranged in concrete and abstract sections. The purely abstract Indication nodes

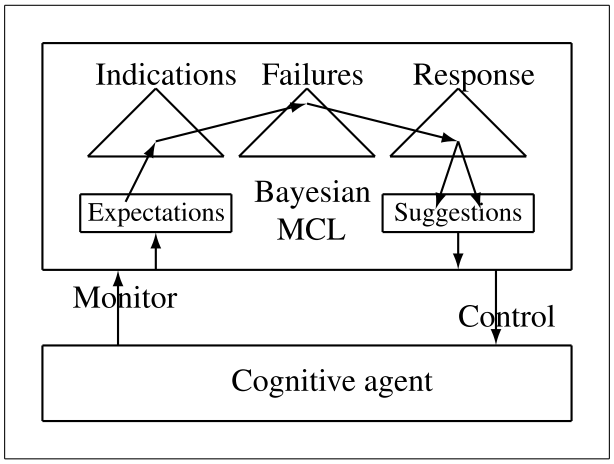


FIG. 6.1. Ontological Linkages

support concepts that cross multiple domains. These make up the core of the Indications ontology. These nodes represent concepts such as “deadline missed,” “failed to change state” and “reward not received.”

The sensor nodes of the Indications ontology model the sensors of the host system and their attributes. When the sensors of the host system are defined, sensor nodes are added to the Indications ontology.

The third set of nodes in the Indications ontology forms a linkage from the concrete sensor and expectations nodes and the abstract, core nodes of the ontology. The divergence nodes define how expectations has been violated. This part of the Indications ontology is show in Figure 6.2.

It is the violation of expectations that starts the MCL NAG cycle. The type of violation and the type of sensor are linked together to a core indications ontology node. Table 6.2 shows sensor and divergence nodes linked to core nodes.

Table 6.2. Sensor, Divergence and Core Node Indications Ontology Linkages

Core Node	*Sensor*	*Divergence Node*
Deadline missed	temporal	late
Reward not received	reward	under
Resource overflow	resource	over
Resource deficit	resource	under
Failed state change	state	missed-unchanged
Unanticipated state change	state	aberration
Asserted control unchanged	control	missed-unchanged

6.1.2 Failure

Once the violated expectations have been evaluated in the Note phase, MCL proceeds to evaluate the problem indications to determine the cause in the Assess phase. The Failure ontology is used in the problem determination. This phase is used (rather than mapping indications directly to responses) because of the ambiguous nature of failures and their indications: two different failures which need different responses might have the same initial problem indications and a single problem might manifest itself with multiple indications.

The Failure ontology (Figure 6.3) is a catalog of the various problems that befall cognitive systems. This includes problems with sensors, effectors, resources, and the domain model (or models). The links in the Failure ontology are all of the is-a variety. Thus a "Sensor Malfunction" is-a "Sensor Error" is-a "Knowledge Error" is-a "Failure". The "Failure" node is the root of the Failure ontology and all Failure ontology nodes eventually lead to it.

6.1.3 Response

As the Failure ontology was an itemized list of everything that can go wrong with a cognitive system, the Response ontology (Figure 6.4) is a list of everything that can be done about it. There are two types of nodes in the Response ontology: abstract and concrete.

The abstract nodes represent general problem-solving techniques and the concrete nodes represent specific suggestions that MCL can send to the host system. The links within the response ontology are for “is-a” relationships. For example, “Strategic Change” is-a “System Response” is-a “Internal Response” which is-a “Response.” The “Response” node is the root of the Response ontology.

6.1.4 Inter-ontology linkages

The three ontologies (Indications, Failure, and Response) are connected by inter-ontology links. Core nodes in the Indications ontology connect to nodes in the Failure ontology. Many nodes of the Failure ontology are connected to nodes in the Response ontology. The linkages form a chain of reasoning from the violated expectation to a suggestion that may correct the problem.

Figure 6.5 shows such a path through the ontology linkages that Chippy Q-learner faced when it returned to what had been the positive reward square and received a negative reward instead. Note that this is a very simplified diagram with most of the nodes and links removed. When the Q-learner moves to the grid square that no longer contains the expected reward, the maintenance expectation of getting the reward in that square is violated. This *activates* the “Reward not received” node in the Indications ontology. That node is connected (via an inter-ontology link) to the “Model error” node of the Failure ontology. The “Model error” node has two children, “Procedure model error” and “Predictive model error” that are connected with intra-ontology links. The “Predictive model error” node has an inter-ontology link to the “Modify predictive response” node of the Response ontology. The “Modify predictive response” has a child node of “Rebuild model response” that is a concrete node for generating the “Rebuild model” suggestion. This set of inter- and intra-ontology linkages allows reasoning from the failed expectation of obtaining a reward to rebuilding of the Chippy agent’s Q-table.

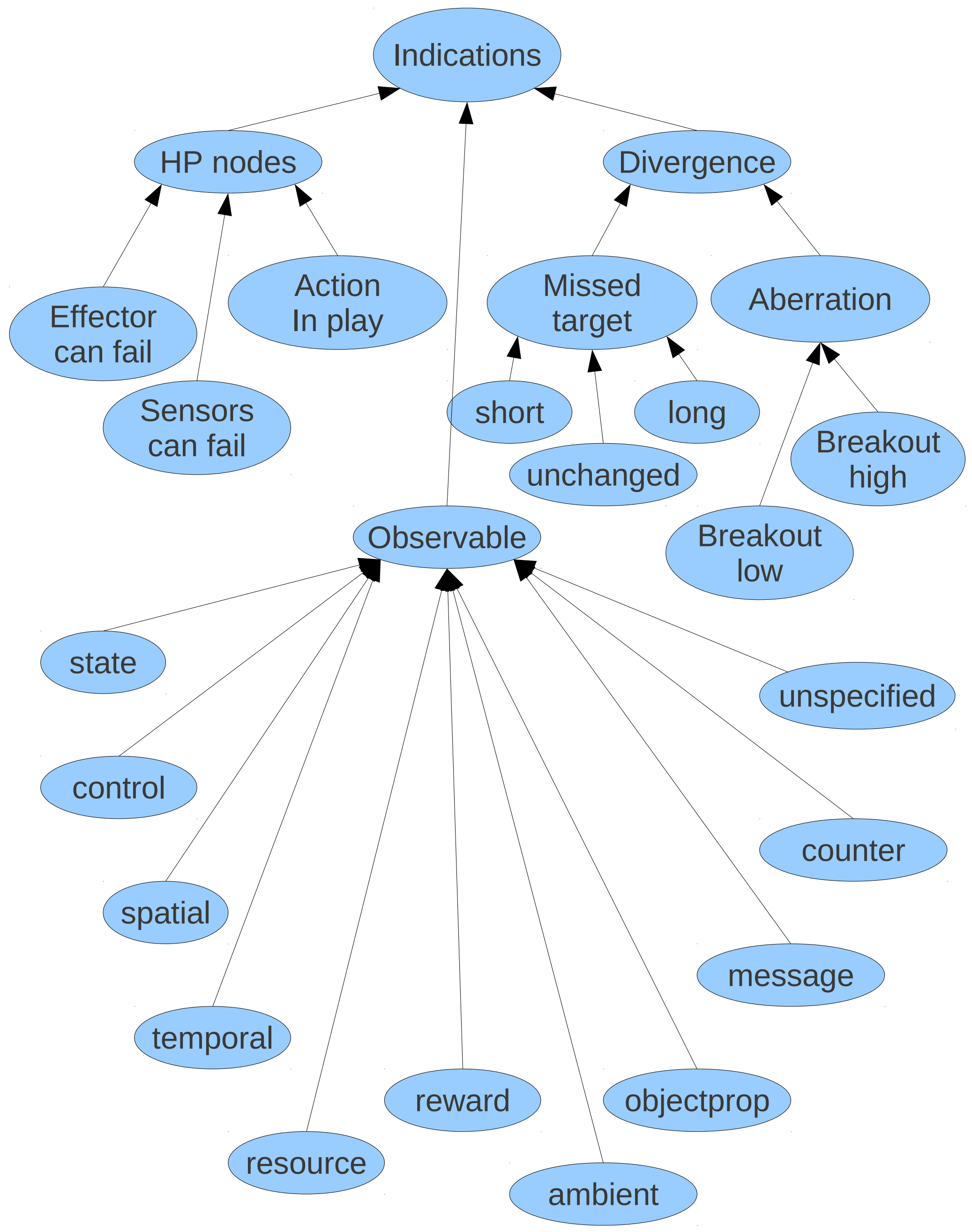


FIG. 6.2. Core Indications Ontology

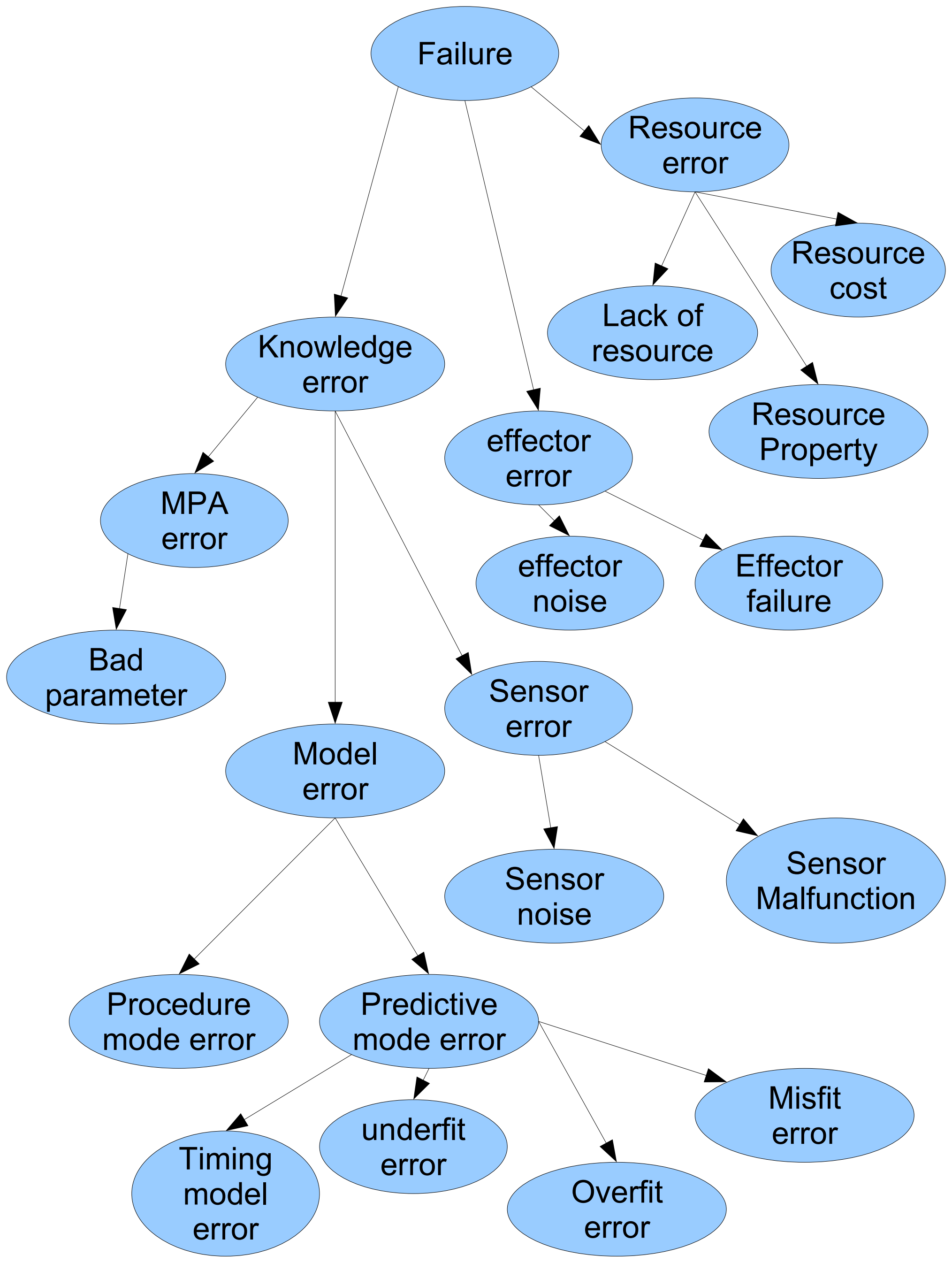


FIG. 6.3. Failure Ontology

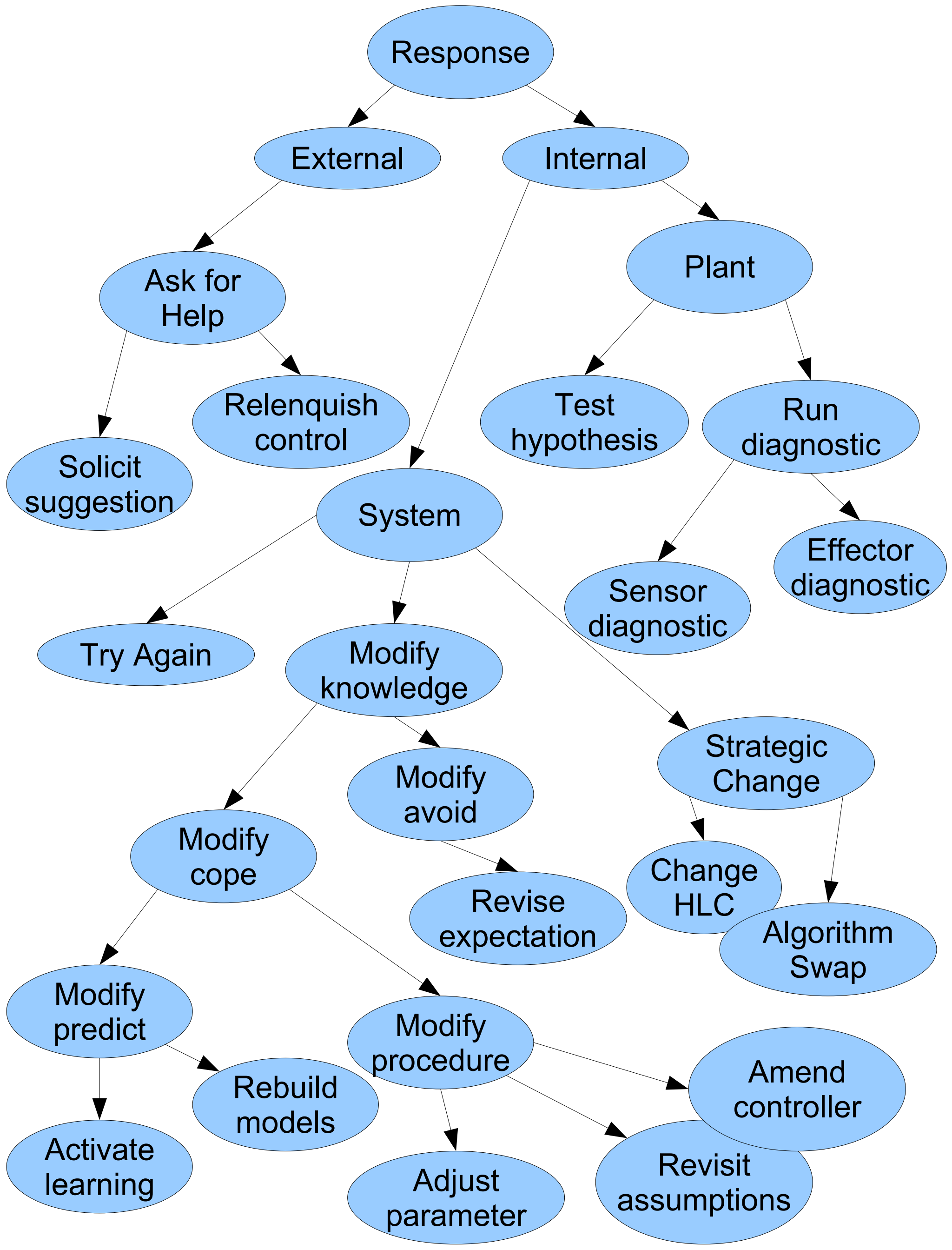


FIG. 6.4. Response Ontology

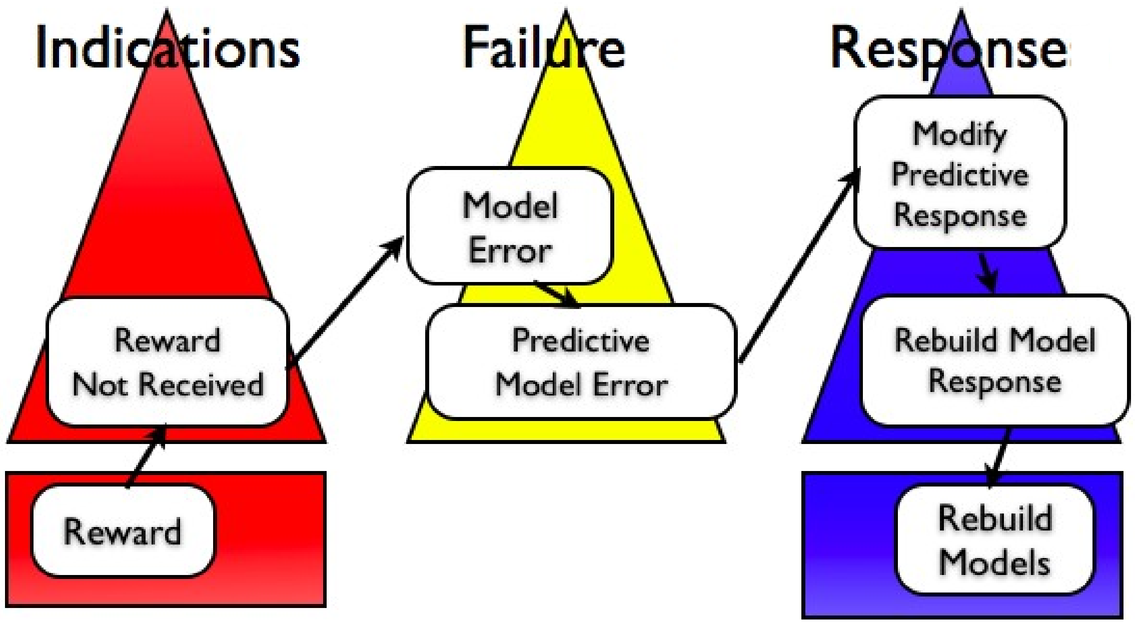


FIG. 6.5. Example Ontology Connections

6.2 Bayesian Conditional Probability Tables

For each problem indication, MCL needs to be able to determine the most likely cause or causes of the failure. For each failure, the responses that are the most likely to correct the failure need to be selected. Thus, given the sensors and the violated expectations, MCL will try to find the responses with the highest probability of working. Actually, MCL should find the response with the highest utility. For this discussion, all the costs will be the same, so the response with the highest probability will also be the response with the highest utility. The three ontologies and their inter-ontology linkages (which form a directed graph) can be viewed as a Bayes net. Direct observation can be made of the sensors. By associating conditional probability tables (CPTs) with each node in the three ontologies, MCL can use Bayesian inference to compute the needed probabilities for the responses. Figure 6.6 shows the addition of CPTs to a small section of the Response ontology.

6.3 MCL Implementation

This section adds to the agent developer's view of MCL2 that was started in Section 5.4. The Bayesian inference for MCL was initially implemented using the Intel-contributed open source Probabilistic Network Library (PNL).[1] MCL2 uses Smile[2] (Druzdzel 1999) from the Decision Systems Laboratory at the University of Pittsburgh. Smile is supported on Mac OS X computers, which allowed the port of MCL2 to that platform.

Ontology Definitions The three ontologies (Indications, Failure, and Response) and the linkages between them are defined in one or more text files using a declarative language.

[1]http://www.sourceforge.net/projects/openpnl
[2]http://genie.sis.pitt.edu/wiki/SMILE_Documentation

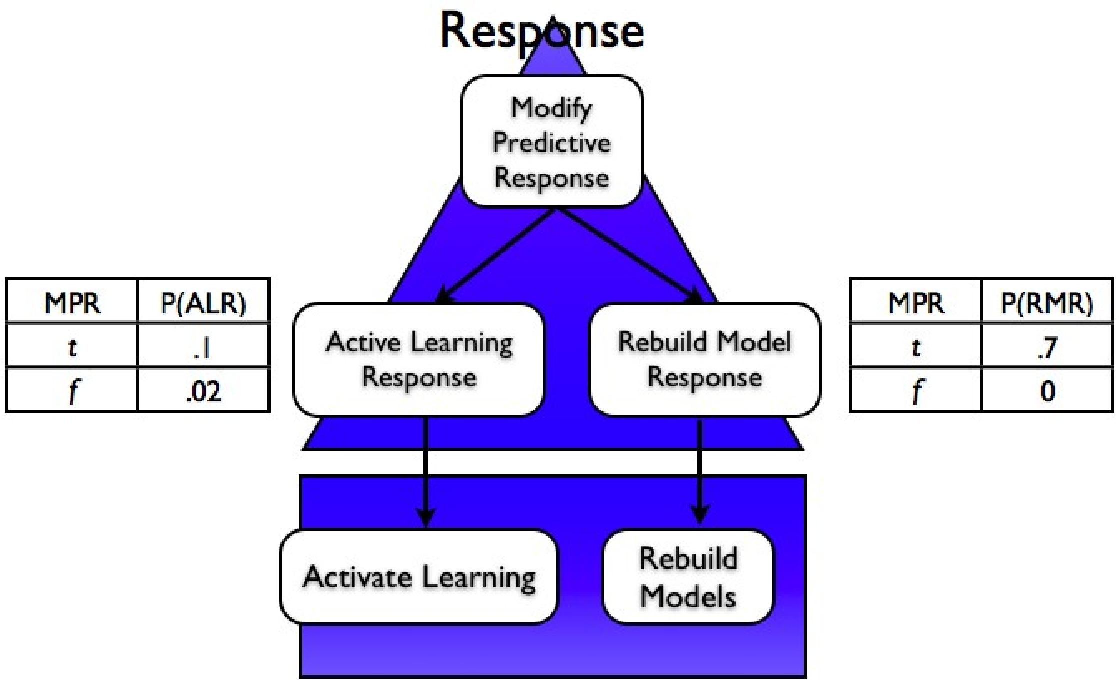


FIG. 6.6. Conditional probability tables for portion of MCL Response ontology

See Listing 6.1 for a short sample and Appendix B for a complete ontology.

Listing 6.1. Portion of MCL Failure Ontology

```
ontology failures (
  node failure(name=failure ,
                doc="the␣class␣of␣all␣failures.")
  node failure(name=knowledgeError ,
                doc="class␣of␣failures␣pertaining␣to␣internal␣↘
                    knowledge␣and␣representations.")

  node failure(name=plantError ,
                doc="class␣of␣failures␣pertaining␣to␣the␣physical↘
                    ␣agent.")
  )

linkage all (
  link abstraction(src=knowledgeError ,dst=failure)
  link abstraction(src=plantError ,dst=failure)
```

Suggestions MCL assigns a *Concrete Response Code* for all suggestions that it returns. The codes are listed in Table 6.3 below. The repairs that an agent initiates (if any) upon receiving one of these codes in the monitor response from MCL depends on the agent. The responses made by the Mars Rover are discussed in the next section.

Table 6.3. MCL Concrete Responses Code

Concrete Response Code	*Concrete Node*	Abstract Node
CRC_SOLICIT_HELP	Solicit suggestion	Ask for help
CRC_RELINQUISH_CONTROL	Relinquish control	Ask for help
CRC_SENSOR_DIAG	Run sensor diagnostic	Run diagnostic
CRC_EFFECTOR_DIAG	Run effector diagnostic	Run diagnostic
CRC_SENSOR_DIAG	Activate Learning	Modify Predictive Model
CRC_EFFECTOR_RESET	Rebuild Models	Modify Predictive Model
CRC_ACTIVATE_LEARNING	Adjust Parameters	Modify Procedure Model
CRC_REVISIT_ASSUMPTIONS	Revisit Assumptions	Modify Procedure Model
CRC_REVISE_EXPECTATIONS	Revise Expectations	Modify Avoid
CRC_ALG_SWAP	Algorithm Swap	Strategic Change
CRC_CHANGE_HLC	Change HLC	Strategic Change
CRC_TRY_AGAIN	Try Again	System

6.4 Mars Rover Integration

This section describes additions to the integration of MCL with the Mars Rover beyond what was described for the MCL Level 1: Instinctive in Section 5.5. The initialization of MCL, the definition of the sensors and expectations, and the monitor interface remain the same. The only change to the Rover is to handle the longer list of possible concrete responses shown in the table below.

CRC_IGNORE The Rover will immediately return "SuggestionIgnored." This is an internal MCL2 code that should never be suggested to an agent.

CRC_NOOP The Rover deletes the existing queue of planner actions to perform. This will cause the Rover to invoke the STRIPS planner to create a new plan using the current goals and sensor values. The Rover always returns "SuggestionImplemented" to MCL.

CRC_TRY_AGAIN The Rover will repeat the last action and then send "Suggestion-Implemented." If the Rover cannot repeat the action (for example, it doesn't have enough energy), it will send "SuggestionIgnored."

CRC_SOLICIT_HELP The Rover will add a "Sleep" goal to await instructions from ground control. Other agents might ask for human assistance or from another metacognitive resource.

CRC_RELINQUISH_CONTROL As with CRC_SOLICIT_HELP, the Rover will add a "Sleep" goal to await instructions from ground control. For other agents, this suggestion may result in releasing the auto-pilot so that a human can take over.

CRC_SENSOR_DIAG The Rover will add a "DiagCmplt" goal (to execute a Diagnostic(D) command) at the immediate level. It will respond with "Suggestion Implemented" when the goal is achieved regardless of the initial or final state of the sensors. The Rover's sensor diagnostic action is more like a sensor reset as it tries to restore the sensors to the factory-calibrated state rather than providing a report of the operating state of the sensors.

CRC_EFFECTOR_DIAG The Rover will add a "BlowCmplt" goal (to execute a Blow (B) command) at the immediate level. It will respond with "Suggestion Implemented" when the goal is achieved regardless of the initial or final state of the motors. The Rover's blow action is more like an effector reset as it tries to clean the motors so that they can work correctly rather than providing a report of the operating state of the motors.

CRC_SENSOR_RESET The Rover will add a "DiagCmplt" goal (to execute a Diagnostic (D) command) at the immediate level. It will respond with "Suggestion Implemented" when the goal is achieved regardless of the initial or final state of the sensors.

CRC_EFFECTOR_RESET The Rover will add a "BlowCmplt" goal (to execute a Blow (B) command) at the immediate level. It will respond with "Suggestion Implemented" when the goal is achieved regardless of the initial or final state of the motors.

CRC_ACTIVATE_LEARNING The Rover is currently returning "SuggestionIgnored" as it does not have a learning subsystem.

CRC_ADJ_PARAMS As the Rover does for CRC_ACTIVATE_LEARNING, it will return "SuggestionIgnored" as the agent has no learning subsystem.

CRC_REBUILD_MODELS Depending on the last action the Rover performed, it would adjust the STRIPS action table or the action cost table and return "SuggestionImplemented." If changing the tables is not possible (e.g., they had already been changed), the Rover would ignore the suggestion and return "SuggestionIgnored."

CRC_REVISIT_ASSUMPTIONS The Rover returns "SuggestionIgnored."

CRC_AMEND_CONTROLLER The Rover returns "SuggestionIgnored."

CRC_REVISE_EXPECTATIONS The Rover returns "SuggestionIgnored."

CRC_ALG_SWAP The Rover returns "SuggestionIgnored."

CRC_CHANGE_HLC The Rover returns “SuggestionIgnored.” As the Rover only has a single High Level Controller (Motivations) there is no other HLC to change to.

CRC_RESCUE As with CRC_RELINQUISH_CONTROL and others, the Rover will add a “Sleep” goal to await instructions from ground control. Other agents could use this suggestion to completly start over.

CRC_GIVE_UP If MCL offers this suggestion, it is because MCL has determined that no other suggestion could possibly provide any assistance and that the best course of action at this point is to admit defeat. The Rover will just add a “Sleep” goal to await instructions from Mission Control.

6.5 Demonstration with Perturbations

Section 4.5 has a trace (Table 4.4) of the Motivated Mars Rover attempting to do a Photo Tour but failing when the recharge action only partially recharges the Rover (Perturbation P1). Also unable to complete a Photo Tour was the Rover with MCL Level 1: Instinctive that always suggested replanning (*NOOP*) when there was an expectation violation. The addition of an Evaluative MCL to the Rover allows it to recover from this problem. In this case, the MCL response of *TRY AGAIN* is sufficient for the Rover to complete the tour, as shown in Tables 6.4 and 6.5.

Table 6.4. MCL Rover able to execute a photographic tour with perturbation P1: Partial charging by retrying the recharge action. (Part 1)

At	*WP*	*CMD*	*NRG*	*MEM*	*TIME*	*DIST*
0	1	2@1	8/92	0/30	16/16	8/8
16	2	3@2	10/82	0/30	20/36	10/18
36	3	5@3	10/72	0/30	20/56	10/28
56	5	4@5	10/62	0/30	20/76	10/38
76	4	C@4	1/61	0/30	20/96	0/38
96	4	5@4	10/51	0/30	20/116	10/48
116	5	3@5	10/41	0/30	20/136	10/58
136	3	8@3	10/31	0/30	20/156	10/68
Energy below 40, added Recharged goal						
156	8	3@8	10/21	0/30	20/176	10/78
176	3	5@3	10/11	0/30	20/196	10/88
196	5	R@5	+30/41	0/30	89/285	0/88
Partial recharge to 41 MCL suggests crc_try_again						
285	5	R@5	+30/71	0/30	59/344	0/88
344	5	3@5	10/61	0/30	20/364	10/98
364	3	8@3	10/51	0/30	20/384	10/108
384	8	I@8	5/46	5/25	20/404	0/108
404	8	3@8	10/36	0/25	20/424	10/118
Energy below 40 again						
424	3	5@3	10/26	0/25	20/444	10/128
444	5	R@5	+30/56	0/25	74/518	0/128
Partial recharge to 56 MCL suggests crc_try_again						
518	5	R@5	+30/86	0/25	44/562	0/128
562	5	4@5	10/76	0/25	20/582	10/138
582	4	C@4	1/75	0/25	20/602	0/138
602	4	I@4	5/70	5/20	20/622	0/138
Only one photo left to go Continued in Part 2						

Table 6.5. MCL Rover able to execute a photographic tour with perturbation P1: Partial charging by retrying the recharge action. (Part 2)

At	*WP*	*CMD*	*NRG*	*MEM*	*TIME*	*DIST*
Continuing with only one photo to go						
622	4	C@4	1/69	0/20	20/642	0/138
642	4	5@4	10/59	0/20	20/662	10/148
662	5	3@5	10/49	0/20	20/682	10/158
682	3	2@3	10/39	0/20	20/702	10/168
Energy below 40 once again						
702	2	1@2	8/31	0/20	16/718	8/176
718	1	R@1	+30/61	0/20	69/787	0/176
Partial recharge to 61 MCL suggests crc_try_again						
787	1	R@1	+30/91	0/20	39/826	0/176
826	1	I@1	5/86	5/15	20/846	0/176
846	1	T@1	7/79	+15/30	15/861	0/176
The Transmission completes the photographic tour. Finishing up with goal of TookPanoramic6						
861	1	2@1	8/71	0/30	16/877	8/184
877	2	3@2	10/61	0/30	20/897	10/194
897	3	5@3	10/51	0/30	20/917	10/204
917	5	6@5	8/43	0/30	16/933	8/212
933	6	P@6	5/38	5/25	30/963	0/212
Photographic tour and added goals complete						

The MCL process starts when the R command expectation group is completed with the Rover sending:

```
send EGcomplete(mr,2,{Zero=0, Energy=41, Memory=30,
  WayPoint=5,Calibrated=1, Localized=1, Sleeping=0,
  Speed=1, TotalDistance=88, TotalTime=285,
  LocalDistance=88, TimeSincePhoto=285,
  TimeSincePanoramic=285})
```

MCL compares the observations given with the expectations for the expectation group. After a recharge, the energy should be at the maximum but the energy level has climbed only to 41. This violates the expectation declared with:

```
declareSelfExp(mr,2,Energy,ec_take_value,100)
```

The violation is recorded and held pending until the next MCL monitor call. It is recorded by creating a MCLFrame that contains information about the violation including a copy of the MCL Bayes network with the concrete indication of the exception inserted into the Indications ontology. A new node, *provenance:self*, is linked to the *resource* and *short-of-target* nodes. This linkage was selected because the Energy sensor was given the resource property and the Energy level of 41 fell short of the target level 100 specified in the expectation. The probabilities of these three nodes are set to 1.

The Rover completes the action execution cycle by calling MCL monitor with:

```
send monitor(mr,{Zero=0, Energy=41, Memory=30, WayPoint=5,
  Calibrated=1, Localized=1, Sleeping=0, Speed=1,
  TotalDistance=88, TotalTime=285, LocalDistance=88,
  TimeSincePhoto=285, TimeSincePanoramic=285})
```

MCL will analyze the problem, and provide guidance in the form of the “CRC_TRY_AGAIN” suggestion.

At	*WP*	*CMD*	*NRG*	*MEM*	*TIME*	*DIST*
1091	5	3@5	10/64	0/25	20/1111	10/254
1111	3	7@3	10/54	0/25	20/1131	10/264
1131	7	P@7	5/49	5/20	30/1161	0/264
The Rover completed one panoramic tour and is starting the next but this time the path between nodes 3 and 8 is blocked.						
1161	7	3@7	10/39	0/20	20/1181	10/274
1181	3	5@3	10/29	0/20	20/1201	10/284
1201	5	R@5	+71/100	0/20	71/1272	0/284
1272	5	P@5	5/95	5/15	30/1302	0/284
1302	5	3@5	10/85	0/15	20/1322	10/294
1322	3	P@3	5/80	5/10	30/1352	0/294
1352	3	*8@3*	9/71	0/10	30/1382	10/304
The path is blocked. MCL suggests crc_try_again						
1382	3	*8@3*	9/62	0/10	30/1412	10/314
We try again but still blocked. MCL suggests crc_try_again						
1412	3	*8@3*	9/53	0/10	30/1442	10/324
We try again but still blocked. MCL suggests crc_try_again						
And so on and so on ...						

Table 6.6. Rover with MCL level 2 failing to execute a panoramic tour with perturbation P7, blocked path

6.6 Limitations and improvements

While an instinctive MCL can handle some problems and an evaluative MCL even more, there are situations where evaluative MCL cannot provide adequate assistance to the host system. As seen in Table 6.6, the “best” suggestion MCL can make, “Try Again”, fails to move the Rover around the blocked path. As the Rover is blocked again and again, MCL keeps reevaluating and keeps coming up with its “best” (but ineffective) suggestion of “Try Again”.

The evaluative MCL’s lack of a sense of time leads to an implementation where each expectation violation is evaluated independently with an initialized ontology network. The violations are added as concrete indications to the network, probabilities are adjusted throughout the network, and the concrete repair suggestions are selected based on cost and probability. When the next expectation violation occurs, the evaluation occurs anew with a fresh ontology network.

The problem with this approach is that it will give the same suggestion for the same expectation violation every time. This is not a problem if that suggestion repairs the current situation and the violation only reoccurs when the failure is reintroduced later. However, if the first suggestion doesn't fix the problem, MCL will not move on to try other suggestions. This problem can be partially corrected by supplying MCL with feedback on its suggestions. This feedback can be used to raise and lower the cost of the suggestion.

A general solution to this situation is to have MCL compare the current set of expectation violations with those that have previously occurred. MCL can then decide to resolve the current violations in the context of previous violations or to start evaluation afresh using a new ontology network to handle a new problem. Adding this sense of prior violations, the temporal element, is the subject of the next chapter.

Chapter 7

MCL LEVEL 3: TEMPORAL

This is the third of three chapters, that takes my idea of multiple metacognitive levels, to deconstruct and explain the existing MCL system that I will be modifying in the next chapter. This chapter describes how MCL works as a Level 3: Temporal metacognition component for use with agent systems. The Level 2: Evaluative MCL from the previous chapter is expanded to include remembering past violation expectations. When a new expectation violation occurs, MCL compares the current violation to previous ones to determine if it should analyze the violation as a new problem or as continuation of an older one.

As shown in the previous chapter, the Metacognitive Loop can be effective in lessening the problem of brittleness in cognitive systems when unexpected perturbations occur. Using Bayesian inference over the Indications, Failure, and Response ontologies allows MCL to better handle perturbations than agents without metacognition or those with only hard-coded responses to stimuli. Fine tuning the ontologies and the conditional probability tables could provide incremental improvement in performance. However, an extension to the evaluative MCL may be able to offer even greater benefits. Saving past exception violations and the successful and unsuccessful attempts to repair the failure(s) adds a temporal dimension to MCL. Figure 7.1 shows the addition of previous exception violation information as part of the meta-knowledge of the enhanced Metacognitive Loop.

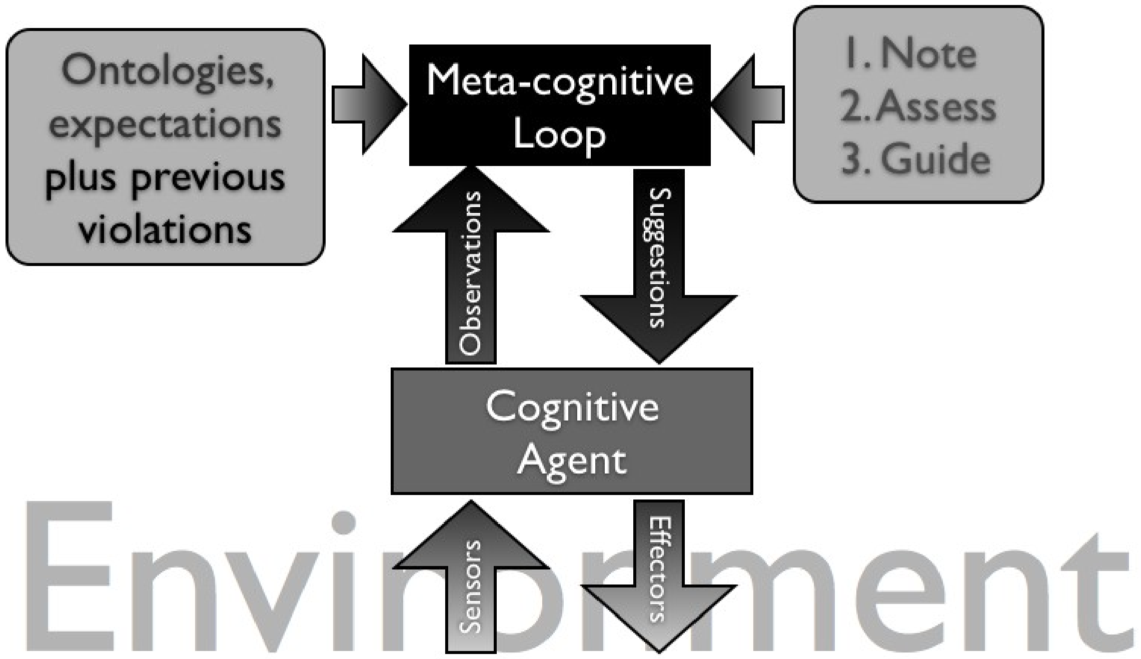


FIG. 7.1. Reentrant MCL

This chapter describes an approach to comparing the agent's MCL state at the time of the current exception with states from previous exceptions. MCL will decide if the current exception appears related to any saved previous exceptions and either base its evaluation on the current exception alone or combined with the previous exception(s). In the first section, the information available at the time of an expectation violation is enumerated. Next, the details of how MCL stores exception states is presented, describing what is stored, how it is retrieved, and when it is discarded. How MCL compares the exception states comes next, with a description of the temporal comparison function currently included with MCL.

7.1 What MCL Knows When an Exception Occurs

When an expectation is violated, MCL has available a number of pieces of information about its internal state and the state of the agent it is supporting. Whether the expectation violation is noticed when an expectation group is completed, or during a monitor call, the same information is available. This section lists and describes the information available at the time of the exception. The items are ordered from simplest to most complex. Expectation violations associated with the P1: Partial Recharge perturbation in the Mars Rover domain is used as an example but the information is generally domain-independent.

7.1.1 Expectation Group ID (EGID)

All expectations are assigned to an expectation group to control when the expectations are tested for validity. Each expectation group is assigned a unique number: the Expectation Group ID (or EGID). For agents with many expectation groups (such as having a different expectation group for each of several actions), this can be a rich source for discriminating between different problems. However, using many groups requires the agent's designer to do more work in defining the agent to MCL interface. The Expectation Group ID is independent of the evaluation technique by MCL. It is directly related to the expectation, but several different exceptions can have the same Expectation Group ID.

In the current implementation of the Mars Rover, only two EGIDs are used. Every Mars Rover expectation is associated with either the EGID of 1 (for expectations valid over the life of the Rover) or the EGID of 2 (for expectations that are valid only for the current action). The recharge (R) action has an expectation that the energy level at the completion of the action becomes 100 units and is associated with the EGID of 2.

7.1.2 Expectation Group Hierarchy

Expectation groups can contain other expectation groups as well as expectations. The Expectation Group Hierarchy is the list of Expectation Group IDs from the terminal to the root. If the agent's designer uses multiple expectation groups, then a number of hierarchies are possible. Like the Expectation Group ID, the Expectation Group Hierarchy is independent of the evaluation technique by MCL. It is directly related to the violated expectation, but several different exceptions can have the same Expectation Group Hierarchy.

The Mars Rover uses only two groups, with the permanent group (EGID=1) being the parent of the action group (EGID=2). Thus, the parent group of the recharge action energy expectation is 1 and the Expectation Group Hierarchy is <1,2>.

7.1.3 Expectation Violation Signatures (EVS)

Every expectation is described as a tuple consisting of

Expectation type The expectation types were given in Table 5.4. An expectation's expectation type determines when and how the observation from the sensor is to be compared to the parameter value of the expectation.

Sensor The sensor whose observations are being tested. The sensors for the Mars Rover are listed in Table 5.5.

Parameters The value to compare with the observation from the sensor (if needed). For example, expectation types of EC_STAYUNDER and EC_TAKE_VALUE need one parameter value, EC_WITHINNORMAL needs two parameter values, while type EC_GO_DOWN doesn't need any parameter values.

Like the Expectation Group ID and Expectation Group Hierarchy, the Expectation Violation Signature is independent of the evaluation technique by MCL. It is directly related to the violated expectation, but several different exceptions can have the same Expectation Group Signature.

Expectation Violation Signatures for the Mars Rover domain sample perturbations are given in Table 8.2. The Expectation Violation Signature for the Mars Rover recharge action (R) energy expectation is < `EC_TAKE_VALUE, Energy, 100` >. It indicates that when the expectation group completed, the *Energy* sensor had a value other than 100.

7.1.4 Initial Indications

When an expectation is violated, the expectation type and the sensor type are used to set nodes in the Indications ontology to True. These nodes are the "Initial Indications" of the exception and follow directly from the violated expectation using the mapping in Table 6.2. There are two or three indicators that describe where the expectation violation was detected, the type of sensor, and type of violation. As the Initial Indications are nodes in the Indications ontology, the Initial Indications are tied to MCL's evaluation technique of using Bayesian inference over ontologies. Just as different violated expectations can have the same Expectation Violation Signatures, different violated expectations can have the same Initial Indications.

The Expectation Violation Signature for the Mars Rover recharge action energy expectation is < `EC_TAKE_VALUE, Energy, 100` >. The Initial Indications that would result from the violation of that expectation would have three parts:

provenance:self The Energy sensor is one that observes the Mars Rover itself. A sensor that provides observations about external objects would have *provenance:object*.

resource The Energy sensor was declared as measuring a resource (as opposed to a reward, temporal, or control sensor).

short-of-target The observed value of the Energy was less than the value the expectation believed it would reach.

7.1.5 Initial Indication Signature (IIS)

Combining the Expectation Group ID (EGID) with the Initial Indications creates an "Initial Indication Signature". The IIS is tied to MCL's evaluation technique of using Bayesian inference over ontologies. Different expectation violations can have the same IIS.

Initial Indication Signatures for sample Mars Rover perturbations are given in Table 8.2. For the violated Recharge expectation, this is a tuple of (< EG=2, {provenance:self resource short-of-target} >).

7.1.6 Bayesian Network

The exception activates specific nodes in the Indications ontology. In turn, these nodes affect the likelihood of other nodes in the Indications ontology and, through inter-ontology links, nodes in the Failure and Response ontologies. By looking at the Initial Indications, you can see what the initial activations for the exceptions are; by looking at the Bayesian network, you can see the diagnosis of the problem (in the Failure ontology) and the prescription (in the Response ontology). Exceptions with different Initial Indications may have similar nodes with high probability in other parts of the network.

7.1.7 MCL Frame

All of the Meta data information above is included in an MCL data structure called the "frame". A frame is created for each exception. It also contains various miscellaneous pieces of data (mostly counts). Table 7.1 lists the data in the MCL frame. Items used only for internal housekeeping are not listed.

Table 7.1. MCL Frame Information

Item	*Description*
Ontologies	A Bayesian network formed by the Indications, Failure, and Response ontologies
State	Current state in the repair process
Active Response	The recommended concrete response node
Expectation Group	Number of expectation group with the violated expectation
Parent Group	Number of parent of the expectation group with the violated expectation
Failures	Number of failing repairs
Successes	Number of successful repairs
Signatures	List of Expectation Violation Signatures
Ontology States	List of previous ontology states

7.1.8 Frame Entry Vector

All of the information about an expectation violation is collected by MCL in a Frame Entry Vector (FEV). It either contains the information listed in the sections above or a reference to it. These items are given in Table 7.2.

The vECode (Entry Code) is set when the FEV is created or is reused. Most FEVs are created in response to an exception violation and have the ENTRY_VIOLATION code. If MCL uses the frame to generate a response, the frame may be reentered (reprocessed) if the host replies that the repair in the response worked (REENTRY_SUCCESS) or not (REENTRY_FAIL). See Table 7.3 for a complete list of Frame Entry Codes.

The major improvement my research made to MCL was designing and implementing algorithms (described in Section 8.4) to evaluate when an expectation violation was a new problem (ENTRY_VIOLATION) or when it was a reoccurrence of a previously seen problem (REENTRY_RECURRENCE). Being able to tell the difference between new problems and previously seen problems lets MCL make better repair suggestions to the agent. The Evaluative MCL in Chapter 6 doesn't make such distinctions, so it repeatedly suggested

TRY AGAIN when the Rover encountered a block path. Being able to associate the current failure to execute a command with a previous failure to do so allows MCL to suggest a different repair that may have more success.

Table 7.2. MCL Frame Entry Vector information

Item	*Section*	*Description*
vEG	7.1.1	Violation Expectation Group Key
vEVS	7.1.3	Expectation Violation Signature
vIIS	7.1.5	Violation Initial Indication Signature
vRef	7.1.7	Violation Frame Referent
vECode	7.1.8	Entry or Reentry code

Table 7.3. MCL Frame Entry Codes

Code	*Description*
ENTRY_UNKNOWN	Un-initialized frame
ENTRY_NEW	Newly created frame
ENTRY_HIA	Host initiated action
ENTRY_VIOLATION	Expectation violated
ENTRY_CLEAN	No violation
REENTRY_RECURRENCE	Recurrence of the violation
REENTRY_ALIAS_VIOL	An alias violation occurred
REENTRY_HOST_SUCCESS	Successful response from host
REENTRY_HOST_FAIL	Failed response from host
REENTRY_HOST_ABORT	Host aborted response
REENTRY_HOST_IGNORE	Host ignored response

7.2 Saving MCL Exception Violation State

To compare the current expectation violation to past violations, the past violation must be saved and accessible. This section looks at what is saved, how it is accessed and how old exception violations are discarded.

7.2.1 What Is Saved

MCL saves previous and current exception violation information in two lists of Frame Entry Vectors. The pending list contains frames for violations that occur between calls to monitor. Violations that are detected when an exception group completes are put into the pending list. The main MCL frame list is for frames from monitor calls. These can be frames created by an exception noted by monitor or frames that monitor moved from the pending list. After a call to monitor, there will be no frames in the pending list.

7.2.2 How it is Referenced

When comparing states to determine if they are similar, the list of Frame Entry Vectors is processed sequentially. The current implementation is to have of all of the FEVs in memory. Future implementation may use a database.

7.2.3 When it is Discarded

While the current implementation has provision for the removal of FEVs from the monitor list, no such actions are being performed. This is less of a problem than it might appear because

1. only exceptions are saved and exceptions by their nature should be exceptional events and
2. the experimental tasks the Rover is asked to perform are limited.

When to purge FEVs and which FEVs to purge is a subject for future research.

7.3 Comparing MCL Exception Violation States

The general description of the workings of MCL's metacognitive loop (note, access, guide) was given in Chapter 5 for MCL Level 1: Instinctive. This section delves deeper into the code and data structures to provide a framework for discussion of the changes that support effective temporal operation.

The monitor() routine has to decide if it should respond at this time or not. It could decline to provide assistance if

1. it is operating in asynchronous mode and insufficient time has passed since the last monitor call or
2. MCL is not active, having been stopped by a call to stopMCL().

Otherwise, the monitor() routine calls the nag() routine to execute the MCL Notice, Assess and Guide function. The responses returned from nag() are returned as the result from monitor().

The nag() function first checks for any new expectation violations with a call to note(). If there are any new exceptions or any pending exceptions, then access() and guide() are called for each exception. The combined responses from the guide() calls are returned as the result of nag(). If there are no new or pending exceptions, no responses are returned from nag().

The main processing of note() is to loop over each expectation group and then over each expectation within the expectation group. If an expectation is not supported by the observations, an expectation violation (an exception) is noted by creating a FrameEntryVector.

At this point, the note() routine has a list of frames from exceptions it discovered as well as any frames created from pending exceptions. These frames can be processed separately, with each generating a response, or two or more can be combined if they are deemed sufficiently similar.

note() determines which frame(s) describing expectation violations that it will send to guide() by using a frame comparison function to determine if the new violation is similar to any previous expectation violations. The temporal comparison function used by the stock version of MCL is described in the next section. Experimental temporal comparison functions are described in the next chapter.

7.4 Sample Temporal Comparison Function

The stock version of MCL Level 3: Temporal contains a temporal comparison function called "passive" that declares the current exception the same as a previous exception if both the Expectation Group ID and the Exception Violation Signature are the same.

7.5 Mars Rover Integration

There are no changes to the Mars Rover beyond those described for MCL Level 2: Evaluative in Section 6.4 as the temporal comparison function is internal to MCL.

Chapter 8

TEMPORAL COMPARISON FUNCTIONS

The "passive" temporal comparison function that has been included in MCL (and was described in the previous chapter) uses only two of the several elements that are available in its comparison. This chapter looks at the construction of several other temporal comparison functions that use different items available to MCL at the time of an expectation violation (as listed in Section 7.1).

The first section of this chapter looks at the information available when an expectation violation is identified (from Section 7.1). The items of the information are examined in terms of how they could contribute to a function that is able to separate new problems from old problems independent of the domain. The second section describes a number of temporal comparison functions created for this research. A few of the temporal comparison functions were purposely constructed to be suboptimal, but useful as baseline cases. Most, however, were built to make the best possible use of the exception state information. In the next section, using sample perturbations from the Mars Rover domain, the temporal functions are evaluated for their ability to distinguish between different problems. This is followed by a short section on how these functions were implemented within MCL. In the final section, two comparison functions are demonstrated using the Mars Rover.

8.1 Information for Temporal Comparison Functions

A Level 3: Temporal metacognition keeps a history of past exceptions, the suggestions given for those exceptions, and if the agent reported it, whether the suggestions were successful. Using this temporal history, a Level 3 MCL is in a better position to suggest the best response for the current exception then a Level 2: Evaluative MCL. A Temporal MCL will favor responses that were successful for exceptions similar to the current exception. However, in a dynamic environment the problem(s) now may not be the same as when the prior exception occured.

For MCL, all of the knowledge we have about the previous exceptions, the suggested repairs(s) and the results of those repairs, are saved within the Frame data structure. What is needed is a way to compare the prior frames with the current one that will be simple to explain, easy to implement, quick to execute, and successful in improving the performance of the host systems.

The items available in the saved MCL frames (as listed in the previous chapter) are examined here to see how well they satisfy the first three criteria. A violation of the expectations that can occur with the partial recharge perturbation (P1) in the Mars Rover domain is used as an example but the discussion is domain-independent.

Expectation Group ID (EGID) The EGIDs of two MCL frames are easy to compare with a single numeric comparison. The amount of information in the EGID will vary with the host implementation. The Mars Rover simulation has only two expectation groups so every exception is either from the permanent EGID of 1 or the action EGID of 2. For the Mars Rover, the EGID could be made more meaningful by creating an expectation group for each action. One could even create a separate expectation group for each action at each location.

Having many expectation groups does not mean that the cause of two exceptions in the same group are related. An energy sensor that reads too low could cause the same recharge expectation (`< EC_TAKE_VALUE, Energy, 100 >`) to fail as a battery that only recharges to half its maximum level or a battery that only recharges a portion on each attempt.

Expectation Group Hierarchy While the EGID is only a single number, the Expectation Group Hierarchy is a vector of numbers. The list of Expectation Groups is likely to be small in practice, allowing for very quick comparisons. Like the EGID, the problem discrimination utility of the Expectation Group Hierarchy will depend on how the host system divides its expectations into groups.

Expectation Violation Signatures (EVS) The EVS consists of the expectation type, the sensor being evaluated, and the parameter value(s). The EVS of a previous frame can be directly compared to the EVS of the current exception using three numeric or string equalities. Repeated violations of the same expectation would produce the same EVS. Exceptions with different sensors will have different EVSs.

Initial Indications The Initial Indications are the nodes activated in the Bayesian network in response to the exception. As it doesn't include specific sensor information or any expectation values, Initial Indications is more general than the Expectation Violation Signature (EVS) for the same exception. The list of initial activation nodes can be obtained as a string, allowing easy and quick comparisons.

Initial Indication Signature (IIS) IIS is the combination of the Initial Signature and the EGID. IIS shares the characteristics of both. It has the implementation-specific EGID and the more general Initial Indications. It is easy to implement and quick to execute as it is an ANDing of the results of the EGIS and Initial Indications comparisons.

Bayesian Network (BN) The heart of the MCL exception/repair evaluation is the Bayesian network that classifies the exception, determines the failure, and suggests a response. Different exceptions will produce different truth values in the nodes of the network. As the networks can be extensive, and the differences between two networks subtle, no quick, easy, fast method has been found to compare two arbitrary Bayesian networks.

While the indications portion of the network changes as initial indication nodes are linked in, the failure and response portions stay structurally static however much the node probabilities may change. Thus, a comparison of the probabilities associated with the concrete response nodes can be efficiently implemented.

Table 8.1 summarizes my subjective rating of the frame information elements according to their simplicity, implementation, and execution. The scale for the three categories is great, good, fair, and poor. Ideally, the best comparison functions would rate as great in all categories as well as (and most importantly) be an excellant discriminator between the same and different problems.

Table 8.1. Frame information elements rated

Element	*Simplicity*	*Implementation*	*Execution*
EGID	Great	Great	Great
EGH	Good	Good	Good
EVS	Good	Good	Good
II	Good	Good	Good
IIS	Good	Good	Good
BN	Fair	Fair	Fair

‘The next section discusses the frame comparison functions as they will be used within the MCL implementation.

8.2 Frame Comparison Functions

Based on the analysis in Section 8.1, I created several frame comparison functions. They should allow the Level 3: Temporal metacognition in MCL to select the correct expectation violation frame (either the new one or one from a previous violation) that would let the access phase provide the best repair suggestion.

I designed the first three (F1: First, F2: New, and F3: Random) as experimental controls and they are not expected to perform well. The frame comparison functions most likely to provide sufficient differentiation to allow MCL to give good advice to the Mars Rover (from the analysis in Section 8.1) are Expectation Violation Signature (EVS) or Initial Indication Signature (IIS), either singularly (F4 or F5) or in combination (F6).

8.2.1 F1: Always The Same (First) Frame

This function always declares that the two frames are equal. In practice, this means that any new exception frame is declared equal to the first exception frame. Thus, MCL sees each expectation violation as the result of a single perturbation. The only advantages of this algorithm are that it is easy to code and extremely quick in execution. It is expected to be a very poor performer.

8.2.2 F2: Frames Always Different (New)

This function returns the opposite of the previous as it always declares the two frames to be different. Thus, MCL will always use the new exception frame for evaluation, ignoring any previous exception frames. When operating with this frame comparison function, the Level 3: Temporal MCL is reduced to Level 2: Evaluative. Like the first algorithm, this one is easy to code and extremely quick in execution.

8.2.3 F3: Random

As with the previous two algorithms, this one does not actually compare the two frames, but delivers its verdict based on a random number generator: half of the time the frames are declared equal and half of the time the frames are declared not-equal. Like the first two algorithms, this one is easy to code and extremely quick in execution.

8.2.4 F4: EVS Equal

Unlike the first three functions, this function (and the ones that follow) do take the values in the two frames into account in its comparison. If the Exception Violation Signature (sensor, expectation type, value) of the two frames are the same then this function declares the two frames equal.

8.2.5 F5: IIS Equal

This function compares the Initial Indication Signatures (IIS) of two frames to determine if the frames represent the same exception. If the IIS of the two frames are the same then the two frames are declared equal.

8.2.6 F6: IIS and EVS

This functions tests both the IIS and EVS of two frames and doesn't judge the frames equal unless both match. The idea was that if either the F4: EVS or the F5: IIS comparisons gave false positives, using both would limit the false positives. The downside is that if either IIS or EVS has a false negative then the two frames being compared are declared not equal.

8.2.7 F7: EVS but not IIS

This function tests both the IIS and EVS of two frames and declares the frames equal if the EVS of the two frames match but the IIS do not. It was not expected that this function would perform well but the worse it performed, the more likely that an IIS comparison of the frames would have made the correct determination.

8.2.8 F8: IIS but not EVS

This function tests both the IIS and EVS of two frames and declares the frames equal if the IIS of the two frames matches but the EVS doesn't. It was not expected that this function would perform well but the worse it performed, the more likely that an EVS comparison of the frames would have made the correct determination.

8.3 Static Evaluation of Comparison Functions

The nine sample perturbations used in the Rover simulation generate expectation violations with the EVS and IIS values given in Table 8.2. Using these values (and having the expectation group hierarchy always being 2/1) it is possible to determine the results of the frame comparison functions for the Rover simulation.

The comparison tables (Tables 8.3 through 8.13) have one row and column for each of the nine numbered Mars Rover perturbations from section 3.2.1 and Table 8.2. A check mark ($\surd$) in a square on the diagonal, where the row and column perturbations are the same, indicates that the comparison function can correctly tell when there are two instances of the same perturbation. A check mark in the other squares indicates that the comparison function can correctly detect that these two perturbations are different. A perfect comparison function would have check marks in all of the squares.

The percentage of correct numbers reported for the comparison functions assumes that each of the perturbations are equally probable. The only time this is likely to be true is in the experiments conducted for this dissertation. This makes these numbers highly theoretical and highly suspect.

Static Evaluation of Expectation Group ID Comparison In the Rover simulation, using EGID correctly predicts when problems are caused by the same perturbation but it is always incorrect when the problems are caused by different perturbations. This is shown in Table 8.3. Overall it is correct only 11% of the time.

Static Evaluation of Expectation Group Hierarchy Comparison Like the EGID, the Expectation Group Hierarchy currently predicts which failures are caused by the same perturbations but is always wrong when the causes are different. Thus, Table 8.4 has check-marks only on the diagonal.

Static Evaluation of F1: First Comparison This comparison function is purely for reference and statistical purposes. It always matches the first (oldest) frame on the frame list. Doing so gives it a perfect score when the perturbations are the same but always wrong when they are different as show in Table 8.5 for a combined static score of 11%.

Static Evaluation of F2: New Comparison This comparison function is also purely for reference and statistical purposes. It never matches an existing frame on the frame list. This makes it always wrong when the perturbations are the same and always correct when they are different (Table 8.6). As there are nine different perturbations, this gives a combined static score of 88%.

Static Evaluation of F3: Random Comparison This is the third and final comparison function purely for reference and statistical purposes. It randomly (50-50) matches the first frame in the frame list otherwise it creates a new frame. It should have a 50% correct rate. Anything else is experimental error (as shown in Table 8.7).

Static Evaluation of F4: EVS Comparison Like EGID and Expectation Group Hierarchy, EVS correctly determines when the failure is caused by the same perturbations. Unlike them, however, it also succeeds in most cases where the perturbations are different which is why Table 8.8 has so many more check-marks than Tables 8.3 and 8.4. It fails in cases such as P1 and P2 where both perturbations can cause unexpected values on the same sensors. Its total static correct rate of 85% outperforms EGID and Expectation Group Hierarchy.

Static Evaluation of Initial Indications Comparison The static performance of Initial Indications (Table 8.9) is close to that of EVS. With the perturbations of the Mars Rover simulation it is slightly better, achieving 90% correct evaluations.

Static Evaluation of F5: IIS Comparison Since the Exception Group ID is always the same in the Mars Rover simulation, it is not unexpected that the static performance of IIS (Table 8.10) should be equal to that of the Initial Indications.

Static Evaluation of F6: EVS and IIS Comparison This is the first of three comparison functions combining EVS and IIS. When taking their results together (Table 8.11), IIS dominates and it gets the same overall score of 90% as all of the perturbation pairs are evaluated the same as with just IIS alone (Table 8.10).

Static Evaluation of F7: EVS and not IIS Comparison Using a negated IIS as a check on EVS improves on EVS different perturbations score at the cost of missing all of the same perturbations on the diagonal as shown in Table 8.12. The total performance, however, is quite close (83% vs. 85% for EVS).

Static Evaluation of F8: IIS but not EVS Comparison When taking their results together (Table 8.11), IIS dominates and it gets the same overall score of 90% as all of the perturbation pairs are evaluated the same as with just IIS alone (Table 8.10).

Summary of static comparison evaluation Table 8.14 summarizes the static evaluation of the various comparison functions. Of the functions that directly compare portions of the expectation violation information, Initial Indications and IIS are the strongest with both having 90% correct. EVS is second at 85%. The EVS and IIS combined methods appear no better than IIS alone. Of the statistical baseline methods, New compares favorably to IIS in the multiple perturbation test.

Based on the static evaluation, IIS comparison should provide the best performance as the frame comparison function for MCL Level 3: Temporal assisting a Mars Rover.

Table 8.2. Sample Exception Information

P1: Partial charging	
EVS	< EC_TAKE_VALUE, Energy, 100 >
IIS	< EG=2, {provenance:self, resource, short-of-target} >
P2: Reduced capacity	
EVS	< EC_TAKE_VALUE, Energy, 100 >
IIS	< EG=2, {provenance:self, resource, short-of-target} >
P3: Longer calibration time	
EVS	< EC_TAKE_VALUE, TotalTime, 20 >
IIS	< EG=2, {provenance:self, temporal, long-of-target} >
P4: Probabilistic calibration	
EVS	< EC_TAKE_VALUE, Calibrated 1 >
IIS	< EG=2, {provenance:self, state, short-of-target, missed-unchanged} >
P5: Recharge loses calibration	
EVS	< EC_MAINTAIN_VALUE, Calibration >
IIS	< EG=2, {provenance:self, state, cwa-decrease} >
P6: Path time change	
EVS	< EC_TAKE_VALUE, TotalTime 20 >
IIS	< EG=2, {provenance:self, temporal, short-of-target} >
P7: Blocked path	
EVS	< EC_TAKE_VALUE, WayPoint 8 >
IIS	< EG=2, {provenance:self, spatial, short-of-target, missed-unchanged} >
P8: Dirty panoramic rotator	
EVS	< EC_TAKE_VALUE, TotalTime, 20 >
IIS	< EG=2, {provenance:self, temporal, short-of-target} >
P9: Noisy sensor	
Item	*Value*
EVS	< EC_TAKE_VALUE, Energy, 100 >
IIS	< EG=2, {provenance:self, resource, short-of-target} >

Table 8.3. Static Evaluation of comparing frames by Expectation Group ID

1/2	*P1*	*P2*	*P3*	*P4*	*P5*	*P6*	*P7*	*P8*	*P9*
P1	√								
P2		√							
P3			√						
P4				√					
P5					√				
P6						√			
P7							√		
P8								√	
P9									√

Table 8.4. Static Evaluation of comparing frames by Expectation Group Hierarchy

1/2	*P1*	*P2*	*P3*	*P4*	*P5*	*P6*	*P7*	*P8*	*P9*
P1	√								
P2		√							
P3			√						
P4				√					
P5					√				
P6						√			
P7							√		
P8								√	
P9									√

Table 8.5. Static Evaluation of comparing frames by F1: First

1/2	*P1*	*P2*	*P3*	*P4*	*P5*	*P6*	*P7*	*P8*	*P9*
P1	√								
P2		√							
P3			√						
P4				√					
P5					√				
P6						√			
P7							√		
P8								√	
P9									√

Table 8.6. Static Evaluation of comparing frames by F2: New

1/2	*P1*	*P2*	*P3*	*P4*	*P5*	*P6*	*P7*	*P8*	*P9*
P1		√	√	√	√	√	√	√	√
P2	√		√	√	√	√	√	√	√
P3	√	√		√	√	√	√	√	√
P4	√	√	√		√	√	√	√	√
P5	√	√	√	√		√	√	√	√
P6	√	√	√	√	√		√	√	√
P7	√	√	√	√	√	√		√	√
P8	√	√	√	√	√	√	√		√
P9	√	√	√	√	√	√	√	√	

Table 8.7. Static Evaluation of comparing frames by F3: Random

1/2	*P1*	*P2*	*P3*	*P4*	*P5*	*P6*	*P7*	*P8*	*P9*
P1		√	√		√		√		√
P2		√		√	√				√
P3	√	√	√	√		√	√	√	
P4	√		√			√	√		
P5	√			√	√	√	√		
P6	√		√	√			√	√	
P7	√	√		√	√				
P8	√	√	√	√	√			√	√
P9			√	√		√	√		√

Table 8.8. Static Evaluation of comparing frames by F4: EVS

1/2	*P1*	*P2*	*P3*	*P4*	*P5*	*P6*	*P7*	*P8*	*P9*
P1	√		√	√	√	√	√	√	
P2		√	√	√	√	√	√	√	
P3	√	√	√	√	√		√		√
P4	√	√	√	√	√	√	√	√	√
P5	√	√	√	√	√	√	√	√	√
P6	√	√		√	√	√	√		√
P7	√	√	√	√	√	√	√	√	√
P8	√	√		√	√		√	√	√
P9			√	√	√	√	√	√	√

Table 8.9. Static Evaluation of comparing frames by Initial Indications

1/2	*P1*	*P2*	*P3*	*P4*	*P5*	*P6*	*P7*	*P8*	*P9*
P1	√		√	√	√	√	√	√	
P2		√	√	√	√	√	√	√	
P3	√	√	√	√	√	√	√	√	√
P4	√	√	√	√	√	√	√	√	√
P5	√	√	√	√	√	√	√	√	√
P6	√	√	√	√	√	√	√		√
P7	√	√	√	√	√	√	√	√	√
P8	√	√	√	√	√		√	√	√
P9			√	√	√	√	√	√	√

Table 8.10. Static Evaluation of comparing frames by F5: IIS

1/2	*P1*	*P2*	*P3*	*P4*	*P5*	*P6*	*P7*	*P8*	*P9*
P1	√		√	√	√	√	√	√	
P2		√	√	√	√	√	√	√	
P3	√	√	√	√	√	√	√	√	√
P4	√	√	√	√	√	√	√	√	√
P5	√	√	√	√	√	√	√	√	√
P6	√	√	√	√	√	√	√		√
P7	√	√	√	√	√	√	√	√	√
P8	√	√	√	√	√		√	√	√
P9			√	√	√	√	√	√	√

Table 8.11. Static Evaluation of comparing frames by F6: EVS and IIS

1/2	*P1*	*P2*	*P3*	*P4*	*P5*	*P6*	*P7*	*P8*	*P9*
P1	√		√	√	√	√	√	√	
P2		√	√	√	√	√	√	√	
P3	√	√	√	√	√	√	√	√	√
P4	√	√	√	√	√	√	√	√	√
P5	√	√	√	√	√	√	√	√	√
P6	√	√	√	√	√	√	√		√
P7	√	√	√	√	√	√	√	√	√
P8	√	√	√	√	√		√	√	√
P9			√	√	√	√	√	√	√

Table 8.12. Static Evaluation of comparing frames by F7: EVS but not IIS

1/2	*P1*	*P2*	*P3*	*P4*	*P5*	*P6*	*P7*	*P8*	*P9*
P1		√	√	√	√	√	√	√	√
P2	√		√	√	√	√	√	√	√
P3	√	√		√	√		√		√
P4	√	√	√		√	√	√	√	√
P5	√	√	√	√		√	√	√	√
P6	√	√		√	√		√	√	√
P7	√	√	√	√	√	√		√	√
P8	√	√		√	√	√	√		√
P9	√	√	√	√	√	√	√	√	

Table 8.13. Static Evaluation of comparing frames by F8: IIS but not EVS

1/2	*P1*	*P2*	*P3*	*P4*	*P5*	*P6*	*P7*	*P8*	*P9*
P1	√		√	√	√	√	√	√	
P2		√	√	√	√	√	√	√	
P3	√	√	√	√	√	√	√	√	√
P4	√	√	√	√	√	√	√	√	√
P5	√	√	√	√	√	√	√	√	√
P6	√	√	√	√	√	√	√		√
P7	√	√	√	√	√	√	√	√	√
P8	√	√	√	√	√		√	√	√
P9			√	√	√	√	√	√	√

Table 8.14. Static Evaluation of Frame Comparison Functions

Comparison	*Percent Correct*		
Method	*Total*	*Same*	*Different*
Expectation Group ID	11	100	0
Expectation Group Hierarchy	11	100	0
F1: First	11	100	0
F2: New	88	0	100
F3: Random	56	55	56
F4: EVS	85	100	83
Initial Indications	90	100	88
F5: IIS	90	100	88
F6: EVS and IIS	90	100	88
F7: EVS but not IIS	83	0	94
F8: IIS but not EVS	90	100	88

8.4 Implementation

I implemented (in C++) several frame comparison algorithms as described in Section 8.2 above. See Appendix C for the code listings.

MCL would normally use one and only one frame comparison function, I implemented several to determine which one worked best in the Mars Rover simulation. For testing and evaluative purposes, the frame comparison function is easily changed using the MCL API.

I also implemented a new MCL API command, setREB(), which allowed the Mars Rover to change the frame comparison function at the start of each experiment.

8.5 Mars Rover Integration

There are no changes to the Mars Rover beyond those described for the Rover using MCL (evaluative) in Section 6.4 as the temporal comparison function(s) are internal to MCL. For the purpose of experimentation, MCL API functions were added to allow the agent to select the temporal comparison. Figure 8.1 show the conversation between the agent and MCL to set the temporal comparison function to F4: IIS. The codes to select the various temporal comparison functions are listed in Table 8.15.

```
send setREB(mr,four)
recv ok(REB set to 'four'.)
```

FIG. 8.1. Setting the Temporal Comparison Function to F4: IIS for a agent initialized with the key 'mr'

Code	*Fn*	*Name*
default		Passive
one	F1	First
two	F2	New
three	F3	Random
four	F4	EVS
five	F5	IIS
six	F6	EVS and IIS
seven	F7	EVS but not IIS
eight	F8	IIS but not EVS
nine		Passive

Table 8.15. Codes for Static Evaluation of Frame Comparison Functions

8.6 Examples

This section looks at two frame comparison functions in action. Two motivated Mars Rovers augmented by MCL Level 3: Temporal, one using F1: First and one using F5: IIS, attempt to complete a triple panoramic tour that is first perturbed by P7: Blocked path and then by P1: Partial charging[1]. The P1 and P7 perturbations are very different problems and should be treated differently by MCL. Table 8.9 shows that the F5: IIS comparison function should be able to correctly distinguish between the perturbations while the F1: First comparison function (Table 8.5) does not.

When the Rover encounters the blocked path, the first suggestion made by MCL will be to retry the movement action. This will not succeed and MCL will then suggest that the Rover rebuild its models. After taking the blocked path out of its STRIPS table, the Rover is able to generate a plan that moves it around the blocked path. The Rover continues on and completes the second panoramic tour where the perturbation is changed to P1, reduced recharge. For this perturbation the effective repair is to try again. But the *TRY AGAIN* repair failed for the blocked path perturbation so if MCL cannot distinguish between the

[1]See Section 9.1 for details of the triple panoramic tour.

two perturbations, MCL will not suggest *TRY AGAIN* when the recharge action fails to completely recharge the Rover. This is the Rover execution trace shown in Table 8.17. If the comparison function can distinguish that the perturbations are different, MCL will suggest *TRY AGAIN* (as in Table 8.16) and the Rover can complete the tour.

Table 8.16. Rover with MCL level 3 and IIS comparison successfully executing a photographic tour with perturbation P7: Blocked Path and then with P1: Partial Charging

At	*WP*	*CMD*	*NRG*	*MEM*	*TIME*	*DIST*
0	1	P@1	5/95	5/25	30/30	0/0
The Rover starts and completes an unperturbed Panoramic Tour. For the second tour the path between 3 and 8 is blocked.						
1146	8	*3@8*	9/61	0/20	30/1176	10/262
The movement fails because the path is blocked. MCL suggests TRY AGAIN and the Rover tries.						
1176	8	*3@8*	9/52	0/20	30/1206	10/272
But not very successfully.						
1206	8	*3@8*	9/43	0/20	30/1236	10/282
This time MCL suggests REBUILD MODELS. The 3@8 action is removed from the STRIPS tables, and a new plan created.						
1236	8	P@8	5/38	5/15	30/1266	0/282
1266	8	7@8	8/30	0/15	16/1282	8/290
1282	7	3@7	10/20	0/15	20/1302	10/300
This time a different route is chosen and the Rover goes on to complete the tour.						
2618	1	P@1	5/36	5/15	30/2648	0/590
2648	1	R@1	+30/66	0/15	64/2712	0/590
When the recharge doesn't live up to expectations, MCL suggests TRY AGAIN.						
2712	1	R@1	+30/96	0/15	34/2746	0/590
The Rover goes on to complete the third tour.						
4978	8	P@8	5/55	5/15	30/5008	0/1030
Photographic tour with added goal complete						

Table 8.17. Rover with MCL level 3 and F1: First comparison unsuccessfully executing a photographic tour with perturbation P7: Blocked path and then with perturbation P1: Partial charging

At	*WP*	*CMD*	*NRG*	*MEM*	*TIME*	*DIST*
0	1	P@1	5/95	5/25	30/30	0/0
30	1	2@1	8/87	0/25	16/46	8/8
The Rover starts and completes an unperturbed Panoramic Tour. For the second tour the path between 3 and 8 is blocked.						
1439	7	P@7	5/70	5/15	30/1469	0/348
1469	7	8@7	8/62	0/15	16/1485	8/356
1485	8	P@8	5/57	5/10	30/1515	0/356
1515	8	*3@8*	9/48	0/10	30/1545	10/366
The movement fails because the path is blocked. MCL suggests TRY AGAIN and the Rover tries.						
1545	8	*3@8*	9/39	0/10	30/1575	10/376
But not very successfully.						
1575	8	*3@8*	9/30	0/10	30/1605	10/386
This time MCL suggests REBUILD MODELS. The action 3@8 is removed from the STRIPS table. The Rover re-plans, finds another route and complete the tour. For the third tour, the perturbation changes to reduced recharge						
3126	5	R@5	+30/59	0/10	71/3197	0/730
MCL with the First temporal comparison function can't tell that this is a different problem. So MCL suggests SENSOR DIAG instead of TRY AGAIN						
3197	5	D@5	20/39	0/10	10/3207	0/730
which uses enough power so that the Rover need to recharges.						
4185	5	R@5	+30/55	0/20	75/4260	0/882
And then MCL suggests SENSOR DIAG again						
4260	5	D@5	20/35	0/20	10/4270	0/882
And this repeats self over and over again.						
5225	5	R@5	+30/45	0/10	85/5310	0/1042
5310	5	D@5	20/25	0/10	10/5320	0/1042
5320	5	R@5	+30/55	0/10	75/5395	0/1042
5395	5	3@5	10/45	0/10	20/5415	10/1052
5415	3	7@3	10/35	0/10	20/5435	10/1062
5435	7	3@7	10/25	0/10	20/5455	10/1072
5455	3	5@3	10/15	0/10	20/5475	10/1082
The Rover keeps looping through these same instructions as MCL keeps suggesting SENSOR DIAG.						

Chapter 9

METHODOLOGY

The claim made in this dissertation is that adding memory of past problems to the MCL advisor provides better assistance to the agent than a MCL advisor without such memory. The previous two chapters described how such temporal knowledge is added to MCL with several proposed temporal comparison functions. The two chapters addressed the first three of the five research questions. This chapter describes how the claim will be demonstrated and provides the metrics that will be used to answer the final two research questions.

The primary measure for the success of the proposed research is how well the proposed MCL temporal comparison function improves the performance of the host system in a perturbed environment. For baselines, experiments will also be run using non-temporal forms of MCL: Bereft, Instinctive and Evaluative. Only if the agent using temporal MCL with one of the proposed MCL temporal comparison functions performs better than agents using non-temporal MCL will the proposed function be judged as successful.

The remainder of this chapter provides details about the experiments to be run and the evaluation of the results. The next section describes the task that each agent will be required to perform and gives an example of an agent completing it. This chapter's second second will show how the environment will be perturbed for the various experiments. The configurations that will be used for each of the twelve Mars Rovers is also given there. The final two sections of this chapter describe the metrics that will be collected from each experiment and how those metrics will be analyzed to determine the effectiveness of the assistance provided to the Rover by the various MCL configurations.

9.1 Evaluation Domain

The Mars Rover domain described in Section 3.1 will be used to evaluate the MCL Frame comparison function. The basic task used will be a panoramic tour in which the Rover will have to travel to all eight waypoints and at each take a panoramic image. The Rover has neither energy nor storage to complete the task directly. It will have to recharge its battery a few times during the process, as well as transmitting the collected panoramic images, before being able to finish the tour. Additionally, due to the length of the panoramic tour, the motivation to collect a photographic image will be triggered causing the Rover to pursue a *TakeImage*n goal.

An example execution of such a tour using the Motivated Mars Rover from Chapter 4 (MCL Level 0: Bereft) is given in Tables 9.1 and 9.2. In the first part of the tour, the Rover visits all eight locations, takes the panoramic images and transmits them. These actions took 32 steps using a plan from the STRIPS planner that is less than optimal but reasonable given the planner's limitations. However, while satisfying the goals of the tour, the Rover generated additional goals due to its Take Photo and Take Panoramic Image motivations. To satisfy these additional goals took another 10 steps. The faster the Rover can complete its assigned tasks, the fewer additional Photo and Panoramic tasks will be added by the motivations.

In the experiments, the Rover is required to perform three panoramic tours in succession. Between each tour, the environment is perturbed. Each experiment is controlled by a

At	*WP*	*CMD*	*NRG*	*MEM*	*TIME*	*DIST*
Initial plan of P238P35P4P56P537P83P2P						
0	1	P@1	5/95	5/25	30/30	0/0
30	1	2@1	8/87	0/25	16/46	8/8
46	2	3@2	10/77	0/25	20/66	10/18
66	3	8@3	10/67	0/25	20/86	10/28
86	8	P@8	5/62	5/20	30/116	0/28
116	8	3@8	10/52	0/20	20/136	10/38
136	3	5@3	10/42	0/20	20/156	10/48
156	5	P@5	5/37	5/15	30/186	0/48
Energy low, Preempting with Recharged goal Setting plan to just R						
186	5	R@5	+63/100	0/15	63/249	0/48
Resuming tour, new plan of 6P54P53P7P32P						
249	5	6@5	8/92	0/15	16/265	8/56
265	6	P@6	5/87	5/10	30/295	0/56
Too long since last photo, Added TookImage3						
295	6	5@6	8/79	0/10	16/311	8/64
311	5	4@5	10/69	0/10	20/331	10/74
331	4	P@4	5/64	5/5	30/361	0/74
Memory low, Preempting with Transmitted Setting plan to 532T						
361	4	5@4	10/54	0/5	20/381	10/84
381	5	3@5	10/44	0/5	20/401	10/94
401	3	2@3	10/34	0/5	20/421	10/104
Energy low, Preempting with Recharged goal Setting plan to 1R						
421	2	1@2	8/26	0/5	16/437	8/112
437	1	R@1	+74/100	0/5	74/511	0/112
Resuming Transmitted plan with T						
511	1	T@1	12/88	+25/30	25/536	0/112
Resuming tour, new plan of 2P3P7P						
536	1	2@1	8/80	0/30	16/552	8/120
552	2	P@2	5/75	5/25	30/582	0/120
582	2	3@2	10/65	0/25	20/602	10/130
Continued in Table 9.2						

Table 9.1. Sample Panoramic tour by a Motivated Mars Rover with MCL Level 0 (Part 1).

At	*WP*	*CMD*	*NRG*	*MEM*	*TIME*	*DIST*
602	3	P@3	5/60	5/20	30/632	0/130
632	3	7@3	10/50	0/20	20/652	10/140
652	7	P@7	5/45	5/15	30/682	0/140
Finished Panoramic Image taking part of tour New goal: Transmitted, plan of 32T						
682	7	3@7	10/35	0/15	20/702	10/150
Energy low, Preempting with Recharged goal Setting plan to 5R						
702	3	5@3	10/25	0/15	20/722	10/160
722	5	R@5	+75/100	0/15	75/797	0/160
Resuming Transmitted goal Setting plan to 32T						
797	5	3@5	10/90	0/15	20/817	10/170
Too long since last Panoramic Image Adding TookPanormaic8 as low level goal						
817	3	2@3	10/80	0/15	20/837	10/180
837	2	T@2	7/73	+15/30	15/852	0/180
Finished photo tour, Goals now TookImage3 and TookPanoramic8 Setting plan to 354C538I						
852	2	3@2	10/63	0/30	20/872	10/190
872	3	5@3	10/53	0/30	20/892	10/200
892	5	4@5	10/43	0/30	20/912	10/210
912	4	C@4	1/42	0/30	20/932	0/210
932	4	5@4	10/32	0/30	20/952	10/220
Energy low, setting plan to R						
952	5	R@5	+68/100	0/30	68/1020	0/220
Resuming TookImage3 goal, plan is38I						
1020	5	3@5	10/90	0/30	20/1040	10/230
1040	3	8@3	10/80	0/30	20/1060	10/240
1060	8	I@8	5/75	5/25	20/1080	0/240
Setting plan to P for TakePanoramic8						
1080	8	P@8	5/70	5/20	30/1110	0/240
Panoramic tour with added goals complete						

Table 9.2. Sample Panoramic tour by a Motivated Mars Rover with MCL Level 0 (Part 2).

script that sets the goals for the Rover, perturbs the environment, and waits for the Rover to complete each set of goals. A sample script is shown in Figure 9.1. When the script starts, the goals for the Rover are set so that it will execute a panoramic tour and run until all goals are satisfied. After that, a perturbation is added to the environment, the panoramic tour goals set and the simulation run until there are no more goals to be satisfied. When (if) the Rover completes the second tour, the first perturbation is removed and another perturbation (possibly the same one) is added. The goals for the panoramic tour are again set and the Rover set off to finish the third tour. At this point, the controlling script is complete and the experiment ends.

9.2 Experiments

Several experiments were performed using various MCL levels, perturbations and frame comparison functions.

9.2.1 Experiment MCL Levels

Four of the six MCL levels will be used in the experiments. The Motivated Mars Rover (which includes a Level 1: Instinctive component) will be augmented with

MCL Level 0: Bereft that gives the Rover no additional benefit,

MCL Level 1: Instinctive that will re-plan if there is an action expectation violation,

MCL Level 2: Evaluative and

MCL Level 3: Temporal which will suggest a variety of repairs.

MCL Levels 4 and 5 are not included as they are still hypothetical.

9.2.2 Experiment Perturbations

For each of the four MCL levels, several experiments are run varying the perturbations introduced between the first and second, and second and third panoramic tours. Seven different perturbations selected from Section 3.2 are used along with the null perturbation, P0. The perturbations used are listed in Table 9.3. As two perturbations are used per experiment, there are fifty-six different scenarios ($8 \times 8 - 8$ duplicate pairs) but all sixty-four pairings are run.

Table 9.3. Eight perturbations used in the experiments

Num	*Description*	*Section*
0	No perturbation	
1	Partial charging	3.2.2
2	Reduced capacity	3.2.3
3	Longer calibration time	3.2.4
4	Probabilistic calibration	3.2.2
5	Recharge loses localization	3.2.4
6	Path time change	3.2.6
7	Blocked path	3.2.6

9.2.3 Experimental Temporal Comparison Functions

For MCL Level 3: Temporal, experiments will be run with each of the eight frame algorithms described in Section 8.2. The frame comparison algorithms are enumerated in Table 9.4. An additional temporal comparison function, F9: Passive, is included in the experiments. This is the default temporal comparison function included with MCL and was described in Section 7.4.

The total number of experiments is determined by the number of MCL levels (4), the number of perturbation combinations (64), and the number of frame comparison algorithms

Table 9.4. Nine frame comparison algorithms used in the experiments

Num	*Description*	*Static Evaluation*
1	First (Same)	11
2	New (Different)	88
3	Random (50/50)	50
4	EVS	85
5	IIS	90
6	EVS and IIS	90
7	EVS but not IIS	83
8	IIS but not EVS	90
9	Passive	–

(9). As the algorithms are only used at MCL Level 3: Temporal, the base number of experiments to be performed is: $64+64+64+64\times 9 = 768$. To reduce the variability and to perform ANOVA (Judd, McClelland, & Ryan 2009) calculations, each set of experiments needs to be performed twenty times, for a grand total of at least 15,360 experiments.

9.3 Metrics Collection

Each experimental run collects several pieces of information to confirm the completion (or not) of the three panoramic tours as well as to evaluate the performance of MCL and the frame comparison functions under the various perturbation conditions.

9.3.1 CSV files

The experiment software records data in comma separated variable (CSV) files to allow easy incorporation into most spreadsheet, graphing, and data analysis programs. The first line of each file is a header with a short name for each column. The remaining lines contain the data from one experimental variation. The data columns are detailed in Tables 9.5 and 9.6. A short sample results file in shown in Figure 9.2.

9.3.2 SQL files

A utility program takes one or more CSV files generated from running the experiments and combines them into an sqlite3 database. From this database, a series of SQL queries are run to produce reports on the experiment results. In addition to the standard queries, ad-hoc SQL requests can be made to delve deeper into the data.

The database consists of two tables: one to record the CSV file names used, and another to record the experimental data itself. The description of the fields is the same as those given in Tables 9.5 and 9.6. The SQL to create the database tables is given in Figure 9.3.

9.4 Evaluation Criteria

The purpose of the experiments is to support the claim that an agent using metacognition with a temporal component (level 3) and the proper frame comparison function will perform better in a perturbed environment than an agent using instinctive or evaluative metacognition. The number of steps required to complete a triple panoramic tour will be used as the primary method to compare the Rover using different MCL levels and frame comparison functions.

ANOVA analysis (NIST/SEMATECH 2003) is done to determine if the Mars Rover performance using one MCL level (and frame comparison function) is statistically significant from another. The null hypothesis is that there is no difference between the number of steps needed to complete the triple panoramic tour when using the different MCL components. This analysis is done using the Python statlib.anova package.[1]

[1]http://code.google.com/p/python-statlib

The success of the MCL Frame comparison function will be evaluated by the number of tours completed and the average number of actions (steps) required. A single metric, a letter grade from A to F, is used to combine both number of steps and number of tour failures. As usual, the lower letter grades are better (A is best).

Computing the grade score starts by grading each trial using the criteria in Table 9.7. Next, the number of trials for each letter score is determined. Each count is then multiplied by the score for the grade given in Table 9.7. These are summed and then divided by the number of experimental trials to get a numeric score between 0 and 100. A score greater than 90 is assigned an A, greater than 80 a B, greater than 70 a C, greater that 60 a D. Scores of 60 or less are given an F.

```
on start: call rover.set_goal_at_level('TookPanoramic1,
                                        TookPanoramic2,
                                        TookPanoramic3,
                                        TookPanoramic4,
                                        TookPanoramic5,
                                        TookPanoramic6,
                                        TookPanoramic7,
                                        TookPanoramic8;
                                        Transmitted'),
          run until nogoal;
on sleep: stop;
and then
on start: call rover.set_ezp(P3),
          call rover.set_goal_at_level('TookPanoramic1,
                                        TookPanoramic2,
                                        TookPanoramic3,
                                        TookPanoramic4,
                                        TookPanoramic5,
                                        TookPanoramic6,
                                        TookPanoramic7,
                                        TookPanoramic8;
                                        Transmitted'),
          run until nogoal;
on sleep: stop;
and then
on start: call rover.set_ezp(P6),
          call rover.set_goal_at_level('TookPanoramic1,
                                        TookPanoramic2,
                                        TookPanoramic3,
                                        TookPanoramic4,
                                        TookPanoramic5,
                                        TookPanoramic6,
                                        TookPanoramic7,
                                        TookPanoramic8;
                                        Transmitted'),
          run until nogoal;
on sleep: stop;
```

FIG. 9.1. Sample experiment script for three panoramic tours with perturbations.

Table 9.5. Description of the fields in the experiment result CSV file (Part 1).

Column	*Values*	*Description*
MCL	$0-3$	MCL Level (see Section 9.2.1)
FRM	$1-9$	Frame comparison algorithm (see Table 9.4)
P1	$0-7$	Perturbation after first tour (see Table 9.3)
P2	$0-7$	Perturbation after second tour (see Table 9.3)
STEPS	≥ 0	Total number of actions performed
TTIME	≥ 0	Total time to complete the script
TDIST	≥ 0	Total distance traveled
CMPLT	$0-100$	Percent of three tours completed
1	≥ 0	Number of Goto WayPoint 1 actions
2	≥ 0	Number of Goto WayPoint 2 actions
3	≥ 0	Number of Goto WayPoint 3 actions
4	≥ 0	Number of Goto WayPoint 4 actions
5	≥ 0	Number of Goto WayPoint 5 actions
6	≥ 0	Number of Goto WayPoint 6 actions
7	≥ 0	Number of Goto WayPoint 7 actions
8	≥ 0	Number of Goto WayPoint 8 actions
B	≥ 0	Number of Blow actions
C	≥ 0	Number of Calibrate actions
D	≥ 0	Number of Diagnose actions
F	≥ 0	Number of Fast Speed actions
I	≥ 0	Number of Take Image actions
L	≥ 0	Number of Localize actions
M	≥ 0	Number of Medium Speed actions
P	≥ 0	Number of Take Panoramic actions
R	≥ 0	Number of Recharge actions
S	≥ 0	Number of Slow Speed actions
T	≥ 0	Number of Transmit actions
W	≥ 0	Number of Wait actions
Z	≥ 0	Number of Sleep actions

Table 9.6. Description of the fields in the experiment result CSV filea (Part 2).

Column	*Values*	*Description*
knt_zero	≥ 0	Monitor give no suggestions
knt_one	≥ 0	Monitor give one suggestion
knt_two	≥ 0	Monitor give two suggestions
knt_three	≥ 0	Monitor give three suggestions
knt_four	≥ 0	Monitor give four or more suggestions
UNKNOWN	≥ 0	Times an Unknown CRC suggested
IGNORE	≥ 0	Times Ignore suggested
NOOP	≥ 0	Times No Operation suggested
TRY_AGAIN	≥ 0	Times Try Again suggested
SOLICIT_HELP	≥ 0	Times Solicit Help suggested
RELINQUISH_CONTROL	≥ 0	Times Relinquish Control suggested
SENSOR_DIAG	≥ 0	Times Sensor Diagnostic suggested
EFFECTOR_DIAG	≥ 0	Times Effector Diagnostic suggested
SENSOR_RESET	≥ 0	Times Sensor Reset suggested
EFFECTOR_RESET	≥ 0	Times Effector Reset suggested
ACTIVATE_LEARNING	≥ 0	Times Activate Learning suggested
ADJ_PARAMS	≥ 0	Times Adjust Learning Parameters suggested
REBUILD_MODELS	≥ 0	Times Rebuild Models suggested
REVISIT_ASSUMPTIONS	≥ 0	Times Revisit Assumptions suggested
AMEND_CONTROLLER	≥ 0	Times Amend Controller suggestion
REVISE_EXPECTATIONS	≥ 0	Times Revise Expectations suggestion
ALG_SWAP	≥ 0	Times Algorithm Swap suggested
CHANGE_HLC	≥ 0	Times Change HLC suggested
RESCUE	≥ 0	Times Rescue suggested
GIVE_UP	≥ 0	Times Give Up suggested
optionsExhausted	≥ 0	Times response Options Exhausted
otherResponse	≥ 0	Times an unlisted response was given

```
MCL,FRM,P1,P2,STEPS,TTIME,TDIST,CMPLT,"1","2","3","4","5","6","7","8","B","C","D","F","I","L","M","P","R","S","T","W","Z",⇊
⇉ knt_zero,knt_one,knt_two,knt_three,knt_four,UNKNOWN,IGNORE,NOOP,TRY_AGAIN,SOLICIT_HELP,RELINQUISH_CONTROL,SENSOR_DIAG,⇊
⇉ EFFECTOR_DIAG,SENSOR_RESET,EFFECTOR_RESET,ACTIVATE_LEARNING,ADJ_PARAMS,REBUILD_MODELS,REVISIT_ASSUMPTIONS,⇊
⇉ AMEND_CONTROLLER,REVISE_EXPECTATIONS,ALG_SWAP,CHANGE_HLC,RESCUE,GIVE_UP,optionsExhausted,otherResponse
1,0,0,0,143,3835,840,100,47,4,13,26,8,23,4,3,7,0,3,0,0,3,2,0,27,14,0,6,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
1,0,0,1,166,4777,946,100,69,7,16,29,8,26,4,6,4,0,3,0,0,3,2,0,27,25,0,6,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
1,0,0,2,154,4195,916,100,54,5,14,31,6,24,5,4,7,0,3,0,0,3,2,0,27,17,0,6,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
1,0,0,3,126,3388,704,100,42,4,12,21,7,20,3,4,3,0,3,0,0,3,1,0,27,12,0,6,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
1,0,0,4,300,5459,890,90,51,8,17,27,8,23,3,5,3,0,3,0,0,2,2,0,26,15,0,6,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
1,0,0,5,145,3890,860,100,47,5,13,29,6,24,3,3,7,0,3,0,0,3,2,0,27,14,0,6,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
1,0,0,6,147,3887,860,100,48,8,17,27,6,21,3,3,6,0,3,0,0,3,2,0,27,15,0,6,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
1,0,0,7,300,4712,556,90,34,5,12,17,4,14,2,2,3,0,2,0,0,2,1,0,23,10,0,5,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
```

688 more lines here

```
3,9,7,0,300,4178,428,90,25,4,9,12,3,11,3,2,2,0,1,0,0,1,1,0,15,7,0,3,0,0,298,1,0,1,0,0,0,0,4,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
3,9,7,1,300,4232,420,90,24,2,6,14,3,11,2,3,3,0,1,0,0,1,1,0,16,7,0,3,0,0,298,1,0,1,0,0,0,0,4,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
3,9,7,2,300,4106,384,90,21,2,7,14,2,8,1,2,4,0,1,0,0,1,0,0,15,6,0,3,0,0,298,1,0,1,0,0,0,0,4,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
3,9,7,3,300,4075,366,90,21,1,6,13,2,9,2,2,3,0,1,0,0,1,0,0,14,6,0,3,0,0,298,1,0,1,0,0,0,0,4,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
3,9,7,4,300,4018,368,90,20,2,7,13,2,8,2,2,3,0,1,0,0,1,0,0,14,6,0,3,0,0,298,1,0,1,0,0,0,0,4,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
3,9,7,5,300,4252,458,90,26,2,7,16,3,12,2,2,4,0,1,0,0,1,1,0,16,7,0,3,0,0,298,1,0,1,0,0,0,0,4,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
3,9,7,6,300,3743,268,90,14,1,5,9,2,6,1,1,3,0,1,0,0,1,0,0,10,4,0,2,0,0,298,1,0,1,0,0,0,0,4,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
3,9,7,7,300,3865,330,90,17,3,7,11,2,7,1,1,3,0,1,0,0,1,0,0,10,5,0,2,0,0,298,1,0,1,0,0,0,0,4,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
```

FIG. 9.2. Sample CSV experiment file showing the header row (split to fit on page) and the first and last several data rows

```
CREATE TABLE filenames (
   fn_id integer primary key autoincrement,
   filename text);

CREATE TABLE experiments (
   exp_id integer primary key autoincrement,
   file_id integer,  'MCL'   integer,  'FRM'   integer,
   'P1'    integer,  'P2'    integer,  'STEPS' integer,
   'TTIME' integer,  'TDIST' integer,  'CMPLT' integer,
   '1' integer, '2' integer, '3' integer, '4' integer,
   '5' integer, '6' integer, '7' integer, '8' integer,
   'B' integer, 'C' integer, 'D' integer, 'F' integer,
   'I' integer, 'L' integer, 'M' integer, 'P' integer,
   'R' integer, 'S' integer, 'T' integer, 'W' integer,
   'Z' integer,
   'knt_zero'      integer,  'knt_one'              integer,
   'knt_two'       integer,  'knt_three'            integer,
   'knt_four'      integer,
   'UNKNOWN'       integer,  'IGNORE'               integer,
   'NOOP'          integer,  'TRY_AGAIN'            integer,
   'SOLICIT_HELP'  integer,  'RELINQUISH_CONTROL'   integer,
   'SENSOR_DIAG'   integer,  'EFFECTOR_DIAG'        integer,
   'SENSOR_RESET'  integer,  'EFFECTOR_RESET'       integer,
   'ACTIVATE_LEARNING'     integer,
   'ADJ_PARAMS'    integer,  'REBUILD_MODELS'       integer,
   'REVISIT_ASSUMPTIONS'   integer,
   'AMEND_CONTROLLER'      integer,
   'REVISE_EXPECTATIONS'   integer,
   'ALG_SWAP'      integer,   'CHANGE_HLC'          integer,
   'RESCUE'        integer,   'GIVE_UP'             integer,
   'optionsExhausted'      integer,
   'otherResponse'         integer);
```

FIG. 9.3. SQL CREATE TABLE statements for the MCL/Mars Rover experiments

Table 9.7. Grading Criteria for a Single Experimental Trial

Grade	*Points*	*Criteria*
A	95	Number of steps less than average number of steps for unperturbed trials
B	85	Number of steps less than average number of steps plus one standard deviation of the unperturbed trials
C	75	Number of steps less than average number of steps plus two standard deviations of the unperturbed trials
D	65	Number of steps more than average number of steps plus two standard deviations of the unperturbed trials but less than the maximum number of steps allowed for experimental trials
F	0	Number of steps equal to maximum number of steps allowed for experimental trials

Chapter 10

RESULTS

The triple-panoramic tours using the MCL metacognition Levels 0: Bereft, 1: Instinctive, 2: Evaluative, and 3: Temporal were run as prescribed in the preceding methodology chapter. The averaged tabular results are included in Appendix A. The figures and tables in this chapter provide summary information or explore specific experimental results. There were

- 100 experimental results CSV files analyzed,
- 38400 experimental trials,
- and a maximum of 500 steps allowed in a trial.

The experimental results will be presented in three phases. First are the trials with no perturbations as these form a baseline against which the other trials will be judged. Next are trials where the same perturbation is used in the second and third parts of the triple panoramic tour. After that are trials using two different perturbations. In the last section, the frame comparison function with the best performance is determined along with some general observations.

10.1 No Perturbation Results

The basic information about the unperturbed experimental trials is given in Table 10.1. Of the 38400 experimental trials, 600 were done without any perturbations. In an ideal world, with a perfect planner, all 600 trials should have the same number of steps. But the Rover has only a simple, non-deterministic, STRIPS planner which introduces some variability in the quality of plans generated.

Table 10.1. Unperturbed Trials

What	*Number*
P0 experiments	600
P0 minimum number of steps	123
P0 average number of steps	147
P0 maximum number of steps	183
P0 minimum total time	3293
P0 average total time	3901
P0 maximum total time	4797
P0 average step time	26.54

This variability is shown in the histograms of the steps required to complete the unperturbed plans shown in Figure 10.1. The best trial had 123 steps and the worst 183. The unperturbed Mars Rover experiment trials were completed in an average of 147.1 steps. The executed steps averaged 26.54 seconds per step. Most steps take 16 or 20 seconds but recharge (R) and transmit (T) may take longer.

Table 10.2 shows the 600 trials divided by the MCL level and Frame comparison method used. An ANOVA test (bottom of Table 10.2) shows that there is no statistical difference between the trials when considering the MCL level / Frame comparison functions. This is the expected result as, when there are no perturbations, MCL is not invoked.

The average of 147.1 steps and the standard deviation of 9.0 are used to fill in the step

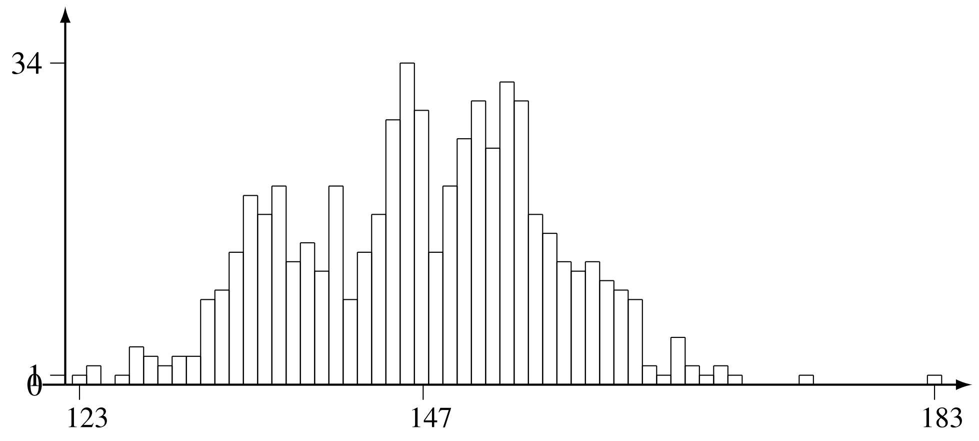


FIG. 10.1. P0 Histogram

limits for the letter scores defined in Table 9.7. Table 10.3 shows the step limits for the five grades.

Table 10.2. P0 Statistics by MCL / Frame comparison function

MCL Level	*Min*	*Mean*	*Max*	σ
M0: Bereft	127	145.9	168	9.78
M1: Instinctive	127	147.4	168	9.06
M2: Evaluative	127	145.6	163	9.11
M3: Temporal	134	148.3	160	7.5
F1: First	128	147.1	165	8.51
F2: New	130	147.7	163	7.92
F3: Random	124	147.4	167	10.25
F4: EVS	123	145.8	162	8.12
F5: IIS	124	147.5	164	9.68
F6: EVS and IIS	132	146.6	169	8.94
F7: EVS but not IIS	129	148.6	183	10.9
F8: IIS but not EVS	132	147.5	162	8.07
ANOVA judged all methods equal				

Table 10.3. Grading Limits for a Single Experimental Trial

Grade	*Points*	*Criteria*
A	95	Number of steps $\leq$ 147
B	85	148 $<$ Number of steps $\leq$ 156
C	75	157 $<$ Number of steps $\leq$ 165
D	65	166 $<$ Number of steps $\leq$ 499
F	0	Number of steps $=$ 500

10.2 Single Perturbation Results

In this section, Mars Rover experimental trials with a single perturbation are analyzed. For each perturbation, a histogram and statistics table are shown. Table 10.4 has a list of figures and tables for the seven single perturbation trials.

Table 10.4. Single Perturbation Histograms and Statistics

Perturbation	*Histogram*	*Statistics*
P1: Partial charging	Figure 10.2	Table 10.7
P2: Reduced capacity	Figure 10.3	Table 10.8
P3: Longer calibration time	Figure 10.4	Table 10.9
P4: Probabilistic calibration	Figure 10.5	Table 10.10
P5: Recharge loses calibration	Figure 10.6	Table 10.11
P6: Path time change	Figure 10.7	Table 10.12
P7: Blocked path	Figure 10.8	Table 10.13

Table 10.5 shows the number of completed single perturbation trials for each of the four MCL levels with level 3 broken out for various frame comparison functions. Of the seven different perturbations, P7: Blocked path caused the most failures with P2: Reduced capacity and P1: Partial charging a distant second and third.

MCL Level 0: Bereft, failed to complete any trial with P1: Partial charging or P7: Blocked path. MCL Level 1: Instinctive, which re-planned on any expectation failure fared only slightly better being able to complete one trial of fifty with P1: Partial charging. MCL Level 2: Evaluative, failed with P2: Reduced capacity and P7: Blocked path. The MCL level 1 and 2 failures are consistent with the failure examples give in Tables 4.4 and 6.6 at the end of Chapters 3 and 6.

Table 10.6 shows the average number of steps for the trials with a single perturbation. The lower the average number of steps, the better the performance. Where there were failures in Table 10.5, Table 10.6 will have an average number of steps near or equal to 500.

MCL Level 2, Evaluative, and MCL Level 3, Temporal, comparison routine 2: New, have the same failures in Table 10.5 and comparable steps in Table 10.6. This was expected as the MCL Level 2 treats every exception as a new event as does the New comparison function.

Table 10.5. Tours Completed by MCL / Frame comparison function when hindered by a single perturbation

Perturbation	*M0*	*M1*	*M2*	*M3*	*F1*	*F2*	*F3*	*F4*	*F5*	*F6*	*F7*	*F8*
P0 No perturbation	50	50	50	50	50	50	50	50	50	50	50	50
P1 Partial charging	0	1	50	50	50	50	50	50	50	50	50	50
P2 Reduced capacity	50	50	0	50	50	0	50	50	50	50	50	0
P3 Longer calibration time	50	50	50	50	50	50	50	50	50	50	50	50
P4 Probabilistic calibration	50	50	50	50	50	50	50	50	50	50	50	50
P5 Recharge loses calibration	50	50	50	50	50	50	50	50	50	50	50	50
P6 Path time change	50	50	50	50	50	50	50	50	50	50	50	50
P7 Blocked path	0	0	0	50	23	0	50	50	50	50	18	0

Table 10.6. Average Steps by MCL / Frame comparison function when hindered by a singe perturbation

Perturbation	*M0*	*M1*	*M2*	*M3*	*F1*	*F2*	*F3*	*F4*	*F5*	*F6*	*F7*	*F8*
P0 No perturbation	145	147	145	148	147	147	147	145	147	146	148	147
P1 Partial charging	500	495	170	168	170	168	185	170	172	168	174	168
P2 Reduced capacity	159	163	500	177	177	500	168	175	163	160	177	500
P3 Longer calibration time	149	147	319	147	148	326	149	148	146	149	320	317
P4 Probabilistic calibration	151	145	149	147	145	148	147	145	150	148	150	147
P5 Recharge loses calibration	156	158	157	158	156	161	157	160	158	165	157	161
P6 Path time change	147	147	147	148	147	147	147	147	146	150	146	146
P7 Blocked path	500	500	500	157	347	500	164	158	159	157	389	500

10.2.1 P1: Partial Charging

The inability of the MCL Level 0: Bereft and MCL Level 1: Instinctive, to complete panoramic tours with the P1: Partial charging perturbation skews the P1 histogram, Figure 10.2. It is shown with a broken horizontal axis to allow more of the information to seen. For any histogram with failure trials, two averages are give: one including the 500 step trials, and one without. In this case, the average is 226.13 when the failing cases are included and 172.02 when they are not.

The ANOVA analysis given at the bottom of Table 10.7 shows puts MCL levels 0 and 1 in the same poorly performing group and separates into three groups with temporal comparison functions F2: New and F6: EVS and IIS doing the best.

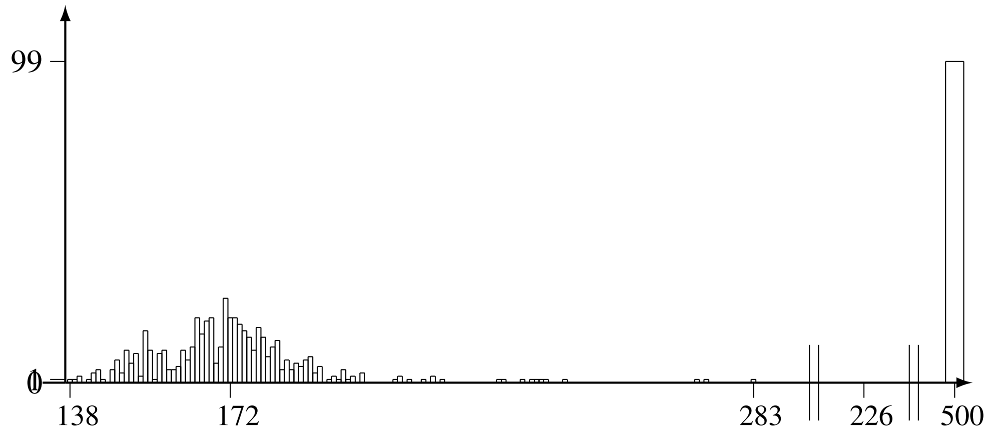


FIG. 10.2. P1 Histogram

Table 10.7. P1 Statistics by MCL / Frame comparison function

MCL Level	*Min*	*Mean*	*Max*	σ	*A*	*B*	*C*	*D*	*F*	*G*
M0: Bereft	500	500.0	500	0.0	0	0	0	0	50	F
M1: Instinctive	273	495.5	500	32.1	0	0	0	1	49	F
M2: Evaluative	143	170.7	243	17.06	2	9	9	30	0	C
M3: Temporal	140	168.9	215	13.12	2	6	10	32	0	C
F1: First	139	170.4	210	14.02	2	7	8	33	0	C
F2: New	143	168.1	189	12.52	2	8	12	28	0	C
F3: Random	145	185.6	283	28.55	1	3	2	44	0	D
F4: EVS	144	170.5	207	11.12	1	4	11	34	0	D
F5: IIS	148	172.9	239	16.78	0	7	5	38	0	D
F6: EVS and IIS	140	168.1	198	11.64	3	5	10	32	0	C
F7: EVS but not IIS	142	174.1	236	16.7	2	5	4	39	0	D
F8: IIS but not EVS	138	168.8	196	14.28	2	10	9	29	0	C

ANOVA ordered grouped methods
C F2: New and F6: EVS and IIS
C-D M2, M3, F1, F4, F5, F7 and F8
D F3: Random
F M0: Bereft and M1: Instinctive

10.2.2 P2: Reduced Capacity

Reducing the battery of the Rover proved more challenging than reducing the amount of recharge. MCL Level 2: Evaluative, failed completely as did its twin MCL Level 3: Temporal with comparison function F2: New. Also failing was MCL Level 3: Temporal with comparison function F8: IIS but not EVS. So while a Rover with MCL Level 1: Instinctive, can handle this perturbation, Rovers with what should be a “better” metacognitive assistance do not. All of the failing MCLs keep suggesting *TRY AGAIN* when the recharge action brought the Rover’s battery only up to 80 energy units and not the expected 100. MCL continued repeating the advice until the trial was stopped at 500 steps. While *TRY AGAIN* is good advice in some cases (e.g., P1: Partial charging), it does not solve the problem of reduced battery capacity.

The ANOVA analysis reflects the difference between the failing and non-failing trials.

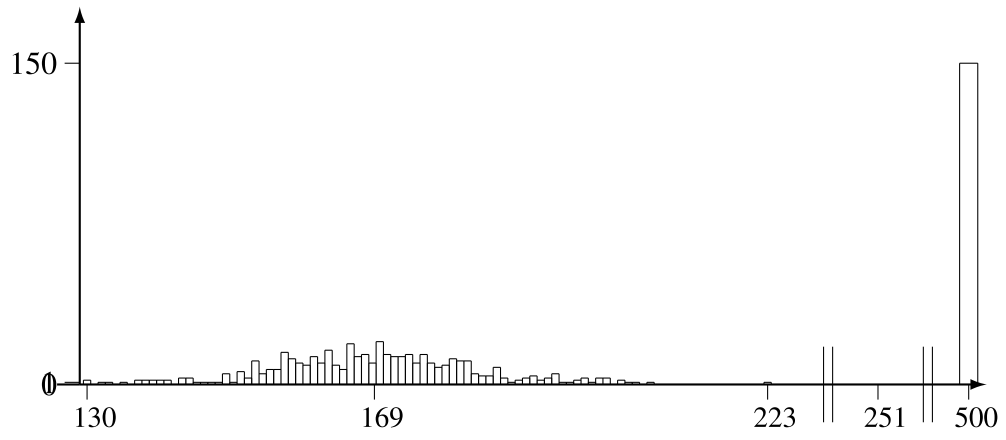


FIG. 10.3. P2 Histogram

Table 10.8. P2 Statistics by MCL / Frame comparison function

MCL Level	*Min*	*Mean*	*Max*	*σ*	*A*	*B*	*C*	*D*	*F*	*G*
M0: Bereft	132	159.7	186	12.04	6	10	18	16	0	C
M1: Instinctive	140	163.3	200	11.7	2	13	17	18	0	C
M2: Evaluative	500	500.0	500	0.0	0	0	0	0	50	F
M3: Temporal	150	177.0	223	15.09	0	3	4	43	0	D
F1: First	153	177.5	204	11.18	0	1	4	45	0	D
F2: New	500	500.0	500	0.0	0	0	0	0	50	F
F3: Random	139	168.7	194	12.68	3	2	16	29	0	C
F4: EVS	149	175.8	205	10.58	0	1	6	43	0	D
F5: IIS	130	163.3	194	12.59	6	8	16	20	0	C
F6: EVS and IIS	130	160.9	182	12.03	7	8	15	20	0	C
F7: EVS but not IIS	161	177.3	203	10.55	0	0	5	45	0	D
F8: IIS but not EVS	500	500.0	500	0.0	0	0	0	0	50	F

ANOVA ordered grouped methods
C M0, M1, F5 and F6
C F3: Random
D M3, F1, F4 and F7
F M2: Evaluative, F2: New and F8: IIS but not EVS

10.2.3 P3: Longer Calibration Time

With longer calibration time, the Rovers were divided into those whose average number of steps were around 149 (about the same as unperturbed) and whose average steps were near 320. So while there were no complete failures, MCL Level 2: Evaluative and MCL Level 3: Temporal with comparison functions 2, 7, and 8 provided poor advice which doubled the number of steps. This difference between the two groups of trials is statistically significant as seen in the ANOVA analysis in Table 10.9.

Rovers using MCL Level 0: Bereft, didn't notice the longer calibration time. The Rovers using MCL Level 1: Instinctive, responded to the expectation failure by re-planning. This didn't change the calibration times but may have resulted in slightly better plans. The other Rovers were divided into those whose metacognitive assistant offered *TRY AGAIN* once and those that continued to offer it again and again. Repeated attempts to Calibrate eventually drained the battery until recharging became imperative. Once the Rover was recharged, it moved on to take the photographic image. This time, the assistant's bad advice didn't stop the Rover from completing the panoramic tour. The bad advice did cause a lengthy delay.

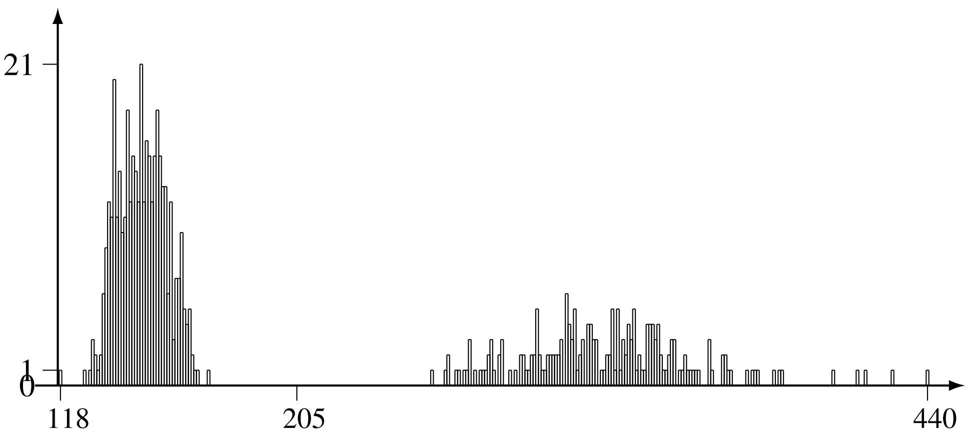


FIG. 10.4. P3 Histogram

Table 10.9. P3 Statistics by MCL / Frame comparison function

MCL Level	*Min*	*Mean*	*Max*	σ	*A*	*B*	*C*	*D*	*F*	*G*
M0: Bereft	127	149.0	169	8.96	20	22	5	3	0	B
M1: Instinctive	118	147.3	166	9.71	25	16	8	1	0	B
M2: Evaluative	269	319.2	414	28.54	0	0	0	50	0	D
M3: Temporal	133	147.4	164	9.51	25	14	11	0	0	B
F1: First	133	148.7	173	9.42	23	15	11	1	0	B
F2: New	261	326.1	440	35.76	0	0	0	50	0	D
F3: Random	132	149.4	165	8.96	21	16	13	0	0	B
F4: EVS	131	148.7	167	8.85	20	22	6	2	0	B
F5: IIS	130	146.8	166	8.92	27	16	5	2	0	B
F6: EVS and IIS	136	149.0	168	8.38	25	16	8	1	0	B
F7: EVS but not IIS	256	320.1	417	29.13	0	0	0	50	0	D
F8: IIS but not EVS	262	317.7	385	33.84	0	0	0	50	0	D

ANOVA ordered grouped methods
B M0, M1, M3, F1, F3, F4, F5 and F6
D M2, F2, F7 and F8

10.2.4 P4: Probabilistic Calibration

Generally only three Photo Image actions are needed during a triple panoramic tour so only three calibrations will need to be done. The repair, *TRY AGAIN*, should be sufficient to overcome this perturbation. The calibration action would be repeated until it eventually succeeds (usually on the next attempt) and then *TRY AGAIN* would not be suggested. The ANOVA analysis (bottom of Table 10.10) does separate the experimental trials into three distinct sets despite having very close means.

Rovers using MCL Level 0: Bereft, whose metacognitive agent would not suggest *TRY AGAIN*, would get a failure upon attempting to take the image while not calibrated. The failure would cause a new plan to be generated which would include a calibration. Rovers using MCL Level 1: Instinctive, would re-plan when the calibration expectation. This plan would start with a calibration action so they were effectively doing the *TRY AGAIN* repair.

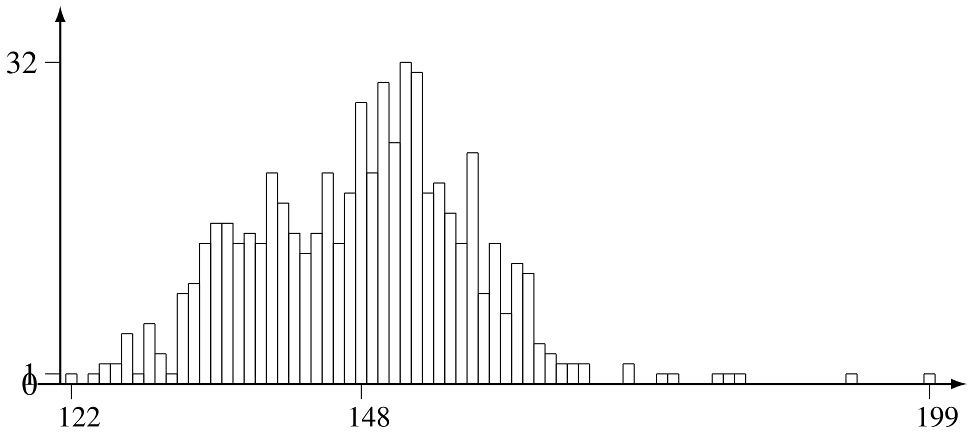


FIG. 10.5. P4 Histogram

Table 10.10. P4 Statistics by MCL / Frame comparison function

MCL Level	*Min*	*Mean*	*Max*	σ	*A*	*B*	*C*	*D*	*F*	*G*
M0: Bereft	132	151.8	181	10.31	16	20	10	4	0	B
M1: Instinctive	125	145.6	172	9.22	24	23	2	1	0	B
M2: Evaluative	125	149.3	165	9.3	19	20	11	0	0	B
M3: Temporal	122	147.5	176	10.1	21	19	9	1	0	B
F1: First	126	145.1	192	11.56	30	13	6	1	0	B
F2: New	126	148.0	164	9.36	24	17	9	0	0	B
F3: Random	124	147.1	162	9.05	19	24	7	0	0	B
F4: EVS	127	145.8	164	8.62	29	15	6	0	0	B
F5: IIS	133	150.1	167	8.62	21	17	10	2	0	B
F6: EVS and IIS	129	148.2	182	10.6	22	19	7	2	0	B
F7: EVS but not IIS	132	150.2	199	12.23	19	18	10	3	0	B
F8: IIS but not EVS	127	147.6	167	10.24	22	17	10	1	0	B

ANOVA ordered grouped methods
B M1: Instinctive, F1: First and F4: EVS
B M2, M3, F2, F3, F5, F6, F7 and F8
B M0: Bereft

10.2.5 P5: Recharge Loses Calibration

In order to take an Image (I), the camera has to be calibrated (C). Since these are done at different nodes, a movement command must be executed between the I and C commands. If the movement causes the power level to drop too low, then a recharge (R) must be done. With the P5 perturbation, a recharge after doing a calibration requires another calibration to be done. Since only three Images are likely added to a triple panoramic tour, only three calibrations should be needed.

If the Rover does a calibration and then a recharge, MCL levels 1 and above would notice the loss of calibration. A second recharge (such as after a *TRY AGAIN*) would not lose calibration because the Rover was no longer calibrated. The new plan would now direct the Rover to calibrate. With a full battery, the Rover would be able to calibrate and take the photographic image without needing to charge again. The ANOVA analysis shows that there were no significant performance differences between the trials.

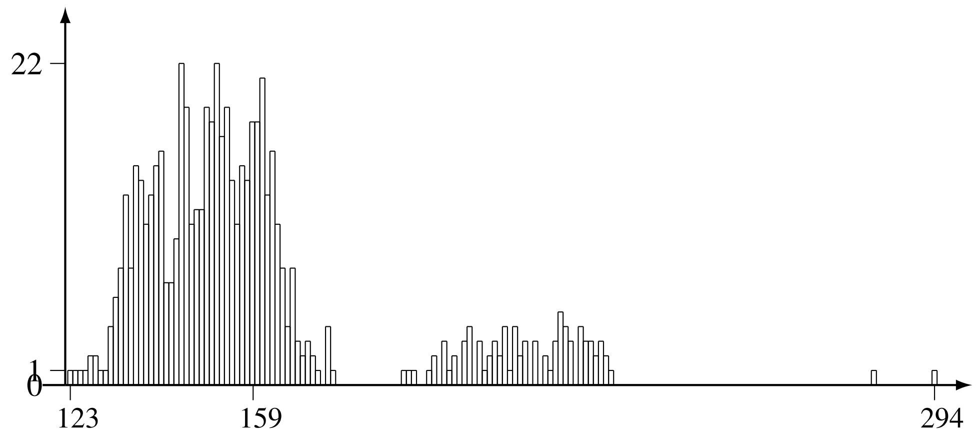


FIG. 10.6. P5 Histogram

Table 10.11. P5 Statistics by MCL / Frame comparison function

MCL Level	*Min*	*Mean*	*Max*	σ	*A*	*B*	*C*	*D*	*F*	*G*
M0: Bereft	132	156.5	219	19.98	15	16	10	9	0	B
M1: Instinctive	131	158.1	221	21.27	13	16	13	8	0	B
M2: Evaluative	134	157.4	282	25.81	19	13	12	6	0	B
M3: Temporal	130	158.3	227	21.47	16	10	18	6	0	B
F1: First	127	156.2	224	22.12	20	14	8	8	0	B
F2: New	123	161.5	294	32.53	20	12	9	9	0	B
F3: Random	126	157.3	226	23.24	20	7	12	11	0	B
F4: EVS	124	161.0	224	26.85	16	12	9	13	0	B
F5: IIS	128	158.8	230	25.07	18	12	10	10	0	B
F6: EVS and IIS	128	165.7	228	32.14	20	8	7	15	0	B
F7: EVS but not IIS	131	157.5	225	24.5	20	9	13	8	0	B
F8: IIS but not EVS	131	161.8	225	25.18	12	15	13	10	0	B
ANOVA judged all methods equal										

10.2.6 P6: Path Time Change

Like changing the time to perform re-calibration, changing the time to move from one node to another did little to the performance of the Rover. The times for completing the triple panoramic tour with this perturbation were not much different from those without a perturbation. The ANOVA analysis isolated F6: EVS and IIS which had the highest mean into its own group otherwise there were no significant performance differences.

While every Rover's triple panoramic tour had to include three calibrations and thus the Rover had to encounter the changed calibration time at least twice, not every tour needed to move between nodes 3 and 8. Also, when movement was made between those two nodes, MCL would see the longer transit time and make a TRY AGAIN suggestion. But as the Rover would have moved, it could no longer re-execute the 3@8 (or 8@3) command so the TRY AGAIN suggestion would be ignored.

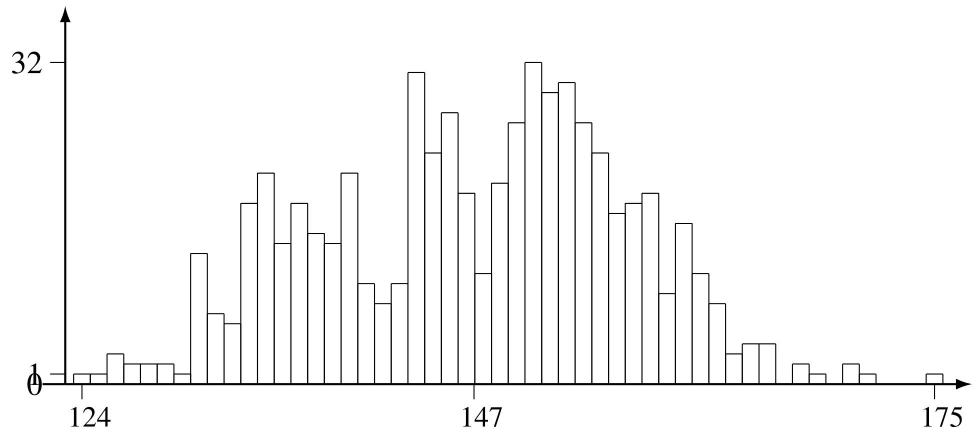


FIG. 10.7. P6 Histogram

Table 10.12. P6 Statistics by MCL / Frame comparison function

MCL Level	*Min*	*Mean*	*Max*	*σ*	*A*	*B*	*C*	*D*	*F*	*G*
M0: Bereft	129	147.3	175	9.65	25	16	8	1	0	B
M1: Instinctive	126	147.2	165	9.48	26	18	6	0	0	B
M2: Evaluative	126	147.5	170	9.67	27	14	8	1	0	B
M3: Temporal	131	148.1	167	8.93	24	16	9	1	0	B
F1: First	132	147.3	161	8.46	21	23	6	0	0	B
F2: New	133	147.7	162	7.61	27	16	7	0	0	B
F3: Random	126	147.5	170	9.15	21	20	8	1	0	B
F4: EVS	132	147.1	162	8.67	24	18	8	0	0	B
F5: IIS	125	146.6	171	9.92	22	22	5	1	0	B
F6: EVS and IIS	131	150.5	164	8.28	19	15	16	0	0	B
F7: EVS but not IIS	124	146.6	168	9.67	25	20	4	1	0	B
F8: IIS but not EVS	128	146.2	167	10.38	26	16	7	1	0	B

ANOVA ordered grouped methods
B M0, M1, M2, M3, F1, F2, F3, F4, F5, F7 and F8
B F6: EVS and IIS

10.2.7 P7: Blocked Path

Rovers using MCL Levels 0: Bereft, 1: Instinctive, and 2: Evaluative, were unable to complete any tours with the path blocked between nodes 3 and 8. Some MCL Level 3: Temporal, Rovers failed completely (comparison functions 2 and 8). Other Rovers (with comparison functions 1 and 7) had some successes and failures. While the rest (3, 4, 5, and 9) had all successes.

The failures were caused by *TRY AGAIN* being offered as the suggestion again and again. The success occurred when the *TRY AGAIN* was judged to have failed and *REBUILD MODELS* suggested or when a low battery allowed the Rover to ignore the *TRY AGAIN* suggestion, recharge, and then have a plan that avoided the broken node 3 to node 8 link.

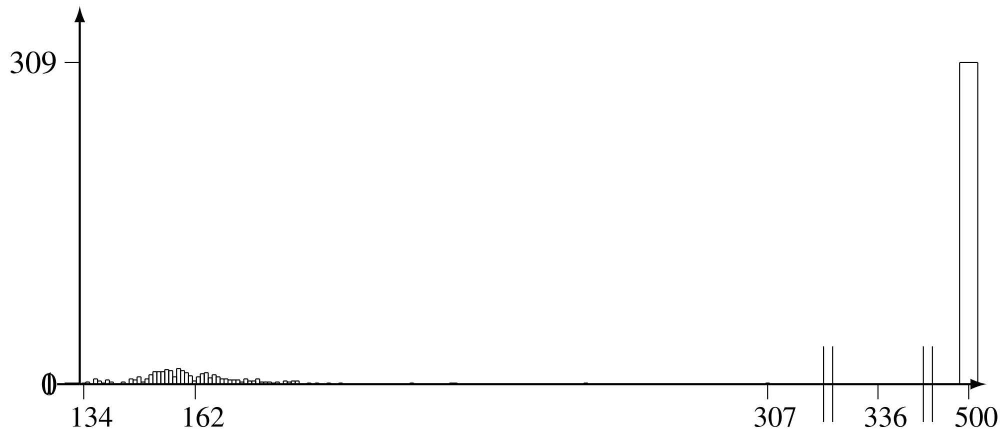


FIG. 10.8. P7 Histogram

Table 10.13. P7 Statistics by MCL / Frame comparison function

MCL Level	*Min*	*Mean*	*Max*	σ	*A*	*B*	*C*	*D*	*F*	*G*
M0: Bereft	500	500.0	500	0.0	0	0	0	0	50	F
M1: Instinctive	500	500.0	500	0.0	0	0	0	0	50	F
M2: Evaluative	500	500.0	500	0.0	0	0	0	0	50	F
M3: Temporal	135	157.3	186	11.4	6	23	12	9	0	B
F1: First	144	347.0	500	167.62	1	3	7	12	27	F
F2: New	500	500.0	500	0.0	0	0	0	0	50	F
F3: Random	140	164.3	196	12.45	3	10	15	22	0	C
F4: EVS	134	158.9	185	9.64	4	17	14	15	0	C
F5: IIS	138	159.2	187	11.18	7	16	17	10	0	C
F6: EVS and IIS	137	157.8	183	10.98	8	15	16	11	0	C
F7: EVS but not IIS	152	389.7	500	150.5	0	2	4	12	32	F
F8: IIS but not EVS	500	500.0	500	0.0	0	0	0	0	50	F

ANOVA ordered grouped methods
B-C M3, F4, F5 and F6
C F3: Random
F F1: First and F7: EVS but not IIS
F M0, M1, M2, F2 and F8

10.2.8 Analysis of Single Perturbation

Table 10.14 shows the steps taken, total time, and counts of action performed for the best and worst MCL level / frame comparison function for each single perturbation. By looking at the best cases, the triple panoramic tour is likely to require 3 photographic images (I) and therefore 3 calibrations (C), 12 recharges (R) (unless the perturbation involves recharging in which case you need 24), 24 to 27 panoramic images (P), 6 transmissions (T) and 1 location (L).

For P0: No Perturbation, the best (MCL Level 3: Temporal, with comparison function F4: EVS) and the worst (MCL Level 3: Temporal, with comparison function F7: EVS but not IIS) owe their places to variations in the plans generated by the STRIPS planner. Rovers who get longer plans end up having to take extra images and panoramic views which forces additional recharges, transmissions, calibrations, and localizations.

With P1: Partial charging, the important counts to observe are for 5 and R. A Rover with MCL Level 3: Temporal, and comparison function 7: EVS but not IIS, went to the recharge location 18 times (about the same as the best P0 time) but, because of the *TRY AGAIN* suggestion, recharged 28 times. A Rover using MCL Level 1: Instinctive, without such *TRY AGAIN* guidance, returned to node 5 115 times for 115 recharges and would still be doing it if the trial hadn't been stopped at 500 steps.

An excessive number of recharges, 447, was also done by the MCL Level 2: Evaluative Rover with the most steps for P2: Reduced capacity. In this case, MCL noticed that the recharge did not bring the battery level to the expected 100 level and suggested that the recharge be done again. As the reduced battery capacity will never advance the battery level past 80, when the repeated recharge did not reach the anticipated 100 level, *TRY AGAIN* was again suggested. This bad advice repeated to the end of the trial at 500 steps.

More calibrations than normal were done with P3: Longer calibration time, P4: Probabilistic calibration, and P5: Recharge loses calibration.

Table 10.15 shows the MCL repair suggestions made for the single perturbation trials. Naturally, the unperturbed (P0) trials have no suggestions as no exceptions were raised. *TRY AGAIN* was the most suggested repair and it was suggested for all perturbations. *REBUILD MODELS* was suggested by some when it was seen that *TRY AGAIN* wasn't working. A distant third was *SENSOR DIAGNOSTIC* which was only suggested by the very desperate coping with the blocked path.

Table 10.14. Best and Worst Performance by Perturbations for Single Perturbations

	Which	*Steps*	*Time*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*B*	*C*	*D*	*F*	*I*	*L*	*M*	*P*	*R*	*S*	*T*
P0: No perturbation																						
Best	F4	123	3293	3	10	21	6	19	3	3	5	0	3	0	0	3	1	0	27	12	0	6
Worst	F7	183	4797	9	21	37	9	29	4	5	6	0	4	0	0	4	2	0	27	18	0	7
P1: Partial charging																						
Best	F8	138	3681	4	11	20	6	18	3	3	4	0	3	0	0	3	1	0	27	28	0	6
Worst	M0	500	16811	5	12	168	3	115	2	23	29	0	1	0	0	1	8	0	15	115	0	3
P2: Reduced capacity																						
Best	F6	130	3557	5	13	19	6	19	4	3	3	0	3	0	0	3	1	0	27	17	0	6
Worst	M2	500	10404	3	7	11	2	9	2	1	1	0	1	0	0	1	0	0	12	447	0	3
P3: Longer calibration time																						
Best	M1	118	3193	4	11	19	5	16	4	3	4	0	3	0	0	3	1	0	27	11	0	6
Worst	F2	440	14624	7	17	30	8	29	6	5	3	0	246	0	0	4	2	0	28	19	0	6
P4: Probabilistic calibration																						
Best	M3	122	3307	3	10	20	6	18	3	4	6	0	3	0	0	3	1	0	27	12	0	6
Worst	F7	199	5135	11	25	39	9	30	4	6	6	0	6	0	0	5	2	0	27	20	0	8
P5: Recharge loses calibration																						
Best	F2	123	3291	3	10	22	5	19	4	4	3	0	3	0	0	3	1	0	27	12	0	6
Worst	F2	294	7183	20	41	49	22	51	5	4	4	0	16	0	0	3	4	0	27	41	0	6
P6: Path time change																						
Best	F7	124	3285	3	10	22	6	19	3	3	5	0	3	0	0	3	1	0	27	12	0	6
Worst	M0	175	4535	9	20	34	8	27	5	6	4	0	4	0	0	4	2	0	27	17	0	7
P7: Blocked path																						
Best	F4	134	3564	7	15	21	5	16	3	7	6	0	3	0	0	3	1	0	27	13	0	6
Worst	M0	500	7273	1	5	16	3	13	3	4	49	0	1	0	0	1	1	0	16	8	0	3

Table 10.15. Repair suggestions made by MCL / Perturbation

Repair Suggestion	*P0*	*P1*	*P2*	*P3*	*P4*	*P5*	*P6*	*P7*
Unknown	0	0	0	0	0	0	0	0
Ignore	0	0	0	0	0	0	0	0
noop	0	4268	765	102	30	65	250	1644
Try again	0	8046	70772	32025	371	708	1378	54301
Solicit help	0	0	0	0	0	0	0	0
Relinquish control	0	0	0	0	0	0	0	0
Sensor diagnostic	0	0	0	0	0	0	0	117
Effector diagnostic	0	0	0	0	0	0	0	0
Sensor reset	0	0	0	0	0	0	0	0
Effector reset	0	0	0	0	0	0	0	0
Activate learning	0	0	0	0	0	0	0	0
Adjust params	0	0	0	0	0	0	0	0
Rebuild models	0	150	250	50	47	2	0	1114
Revisit assumptions	0	0	0	0	0	0	0	0
Amend controller	0	0	0	0	0	0	0	0
Revise expectation	0	0	0	0	0	0	0	0
Algorithm swap	0	0	0	0	0	0	0	0
Change HLC	0	0	0	0	0	0	0	0
Rescue	0	0	0	0	0	0	0	0
Give up	0	0	0	0	0	0	0	0
Options exhausted	0	0	0	0	0	0	0	0
Other response	0	0	0	0	0	0	0	0

Each trial with a single perturbation was graded and tabulated with the results shown in Table 10.16. When averaged and a final grade assigned, it is clear that there is no single standout. MCL Levels 0: Bereft, 1: Instinctive, 2: Evaluative, and the comparison functions F2: New and F8: IIS but not EVS all score badly due to the number of failures. The best, MCL Level 3: Temporal with comparison functions (4, 5, 6, and passive), are clustered together with F5: IIS a slight favorite.

Table 10.16. Single Perturbation Grades by MCL / Frame comparison function

MCL Level	*Min*	*Mean*	*Max*	*σ*	*A*	*B*	*C*	*D*	*F*	*G*
M0: Bereft	127	252	500	157	82	84	51	33	100	F
M1: Instinctive	118	250	500	157	90	86	46	29	99	D
M2: Evaluative	125	277	500	152	67	56	40	87	100	F
M3: Temporal	122	157	227	17	94	91	73	92	0	B
F1: First	126	184	500	92	97	76	50	100	27	C
F2: New	123	278	500	152	73	53	37	87	100	F
F3: Random	124	159	283	20	88	82	73	107	0	C
F4: EVS	124	158	224	17	94	89	60	107	0	C
F5: IIS	125	156	239	16	101	98	68	83	0	B
F6: EVS and IIS	128	157	228	17	104	86	79	81	0	B
F7: EVS but not IIS	124	216	500	107	66	54	40	158	32	D
F8: IIS but not EVS	127	277	500	152	62	58	39	91	100	F

ANOVA ordered grouped methods
B-C M3, F4, F5 and F6
C F3: Random
C F1: First
D F7: EVS but not IIS
D-F M0: Bereft and M1: Instinctive
F M2: Evaluative, F2: New and F8: IIS but not EVS

10.3 Dual Perturbation Results

Experimental trials with two different perturbations make up the majority of the Mars Rover experiments. The average number of steps from the PxPy trials by MCL level and temporal comparison function are shown in Appendix A. The summary information (min, mean, max steps and standard deviation) with grades and ANOVA groupings is below in Table 10.17. The catastrophic failures for single perturbations (e.g., P1: Reduced recharging for MCL Levels 0: Bereft, 1: Instinctive and P2: Reduced capacity for MCL Level 2: Evaluative) also caused failure when these perturbations were part of a dual perturbation trial. Even when each perturbation of a dual trial could be conquered singularly, having two perturbations could cause a trial to reach the 500 step limit. The only Rover that did not have a complete failure was the one using the random frame comparison function.

Table 10.17. PxPy Grades by MCL / Frame comparison function

MCL Level	*Min*	*Mean*	*Max*	*σ*	*A*	*B*	*C*	*D*	*F*	*G*
M0: Bereft	122	288	500	169	680	652	382	234	1252	F
M1: Instinctive	118	292	500	171	711	663	355	181	1290	F
M2: Evaluative	124	318	500	161	459	371	264	733	1373	F
M3: Temporal	122	162	500	40	756	819	811	792	22	C
F1: First	120	199	500	112	771	672	642	727	388	C
F2: New	123	318	500	161	431	368	276	752	1373	F
F3: Random	117	161	411	21	696	738	757	1009	0	C
F4: EVS	123	161	500	38	812	790	781	800	17	C
F5: IIS	123	161	500	42	835	919	774	637	35	B
F6: EVS and IIS	121	161	500	40	836	849	828	664	23	B
F7: EVS but not IIS	124	222	500	113	480	498	469	1346	407	D
F8: IIS but not EVS	123	316	500	162	453	371	287	723	1366	F

ANOVA ordered grouped methods
B-C M3, F3, F4, F5 and F6
C F1: First
D F7: EVS but not IIS
F M0: Bereft and M1: Instinctive
F M2: Evaluative, F2: New and F8: IIS but not EVS

Table 10.18 shows the repair suggestions made by the MCL levels 0 through 3. There are no suggestions for MCL Level 0: Bereft, as that Rover only uses its own, pre-programmed motivations. MCL Level 1: Instinctive was set to always return *NOOP* which the agent interprets as a suggestion to create a new plan from the current goals based on the current status. As it was in Table 10.15, *TRY AGAIN* was the top suggestion by far, followed by *REBUILD MODELS* and *SENSOR DIAGNOSTIC*. F3, the random comparison function, also tried several other suggestions and, at times, ran out of alternate repairs to suggest.

Table 10.18. Repair suggestions made by MCL / Frame comparison function

Repair Suggestion	*M0*	*M1*	*M2*	*M3*	*F1*	*F2*	*F3*	*F4*	*F5*	*F6*	*F7*	*F8*
Unknown	0	0	0	0	0	0	0	0	0	0	0	0
Ignore	0	0	0	0	0	0	0	0	0	0	0	0
noop	0	83944	0	0	0	0	0	0	0	0	0	0
Try again	0	0	553455	14830	12484	544795	14641	14712	11148	11174	229989	538940
Solicit help	0	0	0	0	0	0	0	0	0	0	0	0
Relinquish control	0	0	0	0	0	0	0	0	0	0	0	0
Sensor diagnostic	0	0	0	0	5277	0	1124	0	0	0	0	0
Effector diagnostic	0	0	0	0	0	0	165	0	0	0	0	0
Sensor reset	0	0	0	0	0	0	205	0	0	0	0	0
Effector reset	0	0	0	0	0	0	139	0	0	0	0	0
Activate learning	0	0	0	0	0	0	0	0	0	0	0	0
Adjust params	0	0	0	0	0	0	0	0	0	0	0	0
Rebuild models	0	0	0	2130	708	0	4907	2079	2939	2966	1358	2136
Revisit assumptions	0	0	0	0	0	0	0	0	0	0	0	0
Amend controller	0	0	0	0	0	0	0	0	0	0	0	0
Revise expectation	0	0	0	0	0	0	0	0	0	0	0	0
Algorithm swap	0	0	0	0	0	0	0	0	0	0	0	0
Change HLC	0	0	0	0	0	0	0	0	0	0	0	0
Rescue	0	0	0	0	0	0	0	0	0	0	0	0
Give up	0	0	0	0	0	0	0	0	0	0	0	0
Options exhausted	0	0	0	0	0	0	89	0	0	0	0	0
Other response	0	0	0	0	0	0	0	0	0	0	0	0

10.4 The Best Frame Comparison Function

Measured by average number of actions (Table 10.19), several MCL Level 3 frame comparison functions were tied. The same functions were closely packed for average elapsed time (Table 10.20), and percent of tasks completed (Table 10.21) with frame function F3: Random, with a slight lead.

Table 10.19. Average number of actions

Best	*Counts*	*MCL/Frame*
	288	M0: Bereft
	292	M1: Instinctive
	318	M2: Evaluative
	162	M3: Temporal
	199	F1: First
	318	F2: New
$\Longrightarrow$	161	F3: Random
$\Longrightarrow$	161	F4: EVS
$\Longrightarrow$	161	F5: IIS
$\Longrightarrow$	161	F6: EVS and IIS
	222	F7: EVS but not IIS
	316	F8: IIS but not EVS

Looking at the grades of all the trials (Table 10.17), the same comparison functions are the top rated: F3: Random, F4: EVS, F5: IIS, F6: EVS and IIS. F6: EVS and IIS had the most As, barely beating F5: IIS which had the most As and Bs, while only F3: Random had no failures. The four functions are not statistically different from each other nor from MCL Level 3: Temporal using the default passive comparison function, according to the ANOVA analysis in Table 10.22.

Table 10.20. Average elapsed time

Best	*Time*	*MCL/Frame*
	7135	M0: Bereft
	7806	M1: Instinctive
	7175	M2: Evaluative
	4384	M3: Temporal
	5129	F1: First
	7169	F2: New
$\Longrightarrow$	4339	F3: Random
	4361	F4: EVS
	4373	F5: IIS
	4363	F6: EVS and IIS
	5646	F7: EVS but not IIS
	7132	F8: IIS but not EVS

Table 10.21. Percent of tasks completed

Best	*Counts*	*MCL/Frame*
	60	M0: Bereft
	59	M1: Instinctive
	57	M2: Evaluative
	99	M3: Temporal
	87	F1: First
	57	F2: New
$\Longrightarrow$	100	F3: Random
	99	F4: EVS
	98	F5: IIS
	99	F6: EVS and IIS
	87	F7: EVS but not IIS
	57	F8: IIS but not EVS

Table 10.22. ANOVA Crosstab of MCL level / Frame comparison function

	M0	*M1*	*M2*	*M3*	*F1*	*F2*	*F3*	*F4*	*F5*	*F6*	*F7*	*F8*
M0	——	0.438	*0.000*	*0.000*	*0.000*	*0.000*	*0.000*	*0.000*	*0.000*	*0.000*	*0.000*	*0.000*
M1	0.438	——	*0.000*	*0.000*	*0.000*	*0.000*	*0.000*	*0.000*	*0.000*	*0.000*	*0.000*	*0.000*
M2	*0.000*	*0.000*	——	*0.000*	*0.000*	0.874	*0.000*	*0.000*	*0.000*	*0.000*	*0.000*	0.786
M3	*0.000*	*0.000*	*0.000*	——	*0.000*	*0.000*	0.687	0.402	0.398	0.237	*0.000*	*0.000*
F1	*0.000*	*0.000*	*0.000*	*0.000*	——	*0.000*	*0.000*	*0.000*	*0.000*	*0.000*	*0.000*	*0.000*
F2	*0.000*	*0.000*	0.874	*0.000*	*0.000*	——	*0.000*	*0.000*	*0.000*	*0.000*	*0.000*	0.667
F3	*0.000*	*0.000*	*0.000*	0.687	*0.000*	*0.000*	——	0.525	0.517	0.287	*0.000*	*0.000*
F4	*0.000*	*0.000*	*0.000*	0.402	*0.000*	*0.000*	0.525	——	0.957	0.703	*0.000*	*0.000*
F5	*0.000*	*0.000*	*0.000*	0.398	*0.000*	*0.000*	0.517	0.957	——	0.758	*0.000*	*0.000*
F6	*0.000*	*0.000*	*0.000*	0.237	*0.000*	*0.000*	0.287	0.703	0.758	——	*0.000*	*0.000*
F7	*0.000*	*0.000*	*0.000*	*0.000*	*0.000*	*0.000*	*0.000*	*0.000*	*0.000*	*0.000*	——	*0.000*
F8	*0.000*	*0.000*	0.786	*0.000*	*0.000*	0.667	*0.000*	*0.000*	*0.000*	*0.000*	*0.000*	——

10.5 Comparison to Predicted Performance

Section 8.3 showed the results of performing a static evaluation of MCL Level 3: Temporal comparison functions. Table 8.14 showed the total expected correct comparisons for the experimental perturbations.

The top three comparison functions in the static evaluation were F5: IIS, F6: EVS and IIS, and F8: IIS but not EVS with 90% correct each. F2: New (88%), F4: EVS (85%) and F7: EVS but not IIS (83%) followed closely. The worst two were F1: First (11%) and F3: Random (56%).

Of the top comparison functions from the static evaluation, only F5: IIS and F6: EVS and IIS remained in the top tier of the experimental results. The two worst from the static evaluation did well in the experiments with F3: Random rising to the top tier and F1: First to the second. F8: IIS but not EVS, which was tied for first place in the static evaluations, dropped to the lowest tier in the experiments. The static evaluations were a poor predictor of experimental success. Table 10.23 compares the static and experimental tiering of the comparison functions.

Table 10.23. Static versus Experimental Comparison Function Evaluation

Comparison	*Static*		*Experimental*		*Prediction*
Method	*Percent*	*Tier*	*Grade*	*Tier*	*Success*
F1: First	11	4	C	2	Poor
F2: New	88	2	F	5	Very Poor
F3: Random	56	3	C	1	Poor
F4: EVS	85	2	B	1	Good
F5: IIS	90	1	B	1	Great
F6: EVS and IIS	90	1	B	1	Great
F7: EVS but not IIS	83	2	D	3	Poor
F8: IIS but not EVS	90	1	F	5	Very Poor

10.6 Answers to Research Questions

The search for the temporal comparison function was also to determine the answers to five questions:

How to determine that the current and previous symptoms are related? For MCL, symptom states are captured in an data object called a frame. It is easy to come up with comparison functions (e.g., New) that do not correctly identify related and non-related symptoms. There are also several comparison functions that can (usually) make the correct determination (e.g., IIS, EVS, and IIS AND EVS).

How sure are we that we have made the correct determination? A static analysis of the frame comparison functions for the Mars Rover context were at best around 90% correct. In practice, the functions IIS, EVS, and IIS AND EVS almost always assisted the Rover in completing its task in the face of perturbations.

How does the correct determination affect the response? The frame state sets nodes in the concrete indications portion of the Bayesian network of MCL. They influence which repair suggestion (if any) that MCL will make.

How much does the correct determination improve performance? When the temporal comparison function made the correct determination there was often a definite performance gain. With the P3: Longer calibration time perturbation, the average number of steps needed to complete the triple panoramic tour was halved for those Rovers whose comparison functions correctly advised, versus those whose didn't. At other times, the improvement (if any) was not statistically significant.

How much does the incorrect determination degrade performance? Making an incorrect determination is not usually fatal unless the same bad advice is repeated over and over again. This was the downfall of a couple of the poor frame comparison functions (e.g., First, and EVS BUT NOT IIS). Even the good frame comparison functions (with the notable exception of Random) were sometimes not able to guide a Rover to a successful completion of the task.

Chapter 11

RELATED WORK

Cox (2005b) provides a survey of selected AI metacognition research areas through 2000 (and a little beyond). Newer research is surveyed by Anderson and Oates (2007). This chapter starts by looking at pre-ontology and early-ontology MCL work. Next, Case-based Reasoning and Model Based Reflection are examined as another approach to assisting agents. It ends with a section on a topic not covered in the survey papers, monitoring multi-agent systems.

11.1 Pre-ontology Metacognitive Loop

An early version of MCL was part of a hybrid system to assist Learning agents using both Neural Networks and a symbolic logic reasoner (Hennacy, Swamy, & Perlis 2003). The Metacognitive Loop (Anderson & Perlis 2005), with its notice, assess, and guide phases, is offered as a solution to the problem of brittleness in AI systems as due to the lack of perturbation tolerance. A trio of problem domains (reinforcement learning, navigation, and human-computer dialogue) is shown to benefit from adding MCL. Three research areas were proposed corresponding to the three MCL phases:

1. How should expectations be formulated to best track the performance of the systems?
2. How should the reasoning about the exceptions be organized?

3. What are the best strategies for guiding a system back to proper operation?

The first and third questions remain open issues. Bayesian inference over three sets of ontologies is thought to be the answer to the second question and this dissertation has shown it to be an effective approach (at least in the Mars Rover domain).

Three alternatives to the incorporation of MCL to deal with perturbations have been suggested (Anderson *et al.* 2006):

1. Do nothing,
2. Incorporate a recovery strategy for every possible problem, and
3. Create an extensive world model and continually compare the actual and predicted performance.

The first of these approaches offers nothing except ease of implementation while the last two are too expensive to use. MCL is offered as a cost-effective alternative as it has only a moderate cost and can greatly improve a system's tolerance to perturbation. This is demonstrated with the Chippy grid world. Perturbation in Chippy was used to explore different expectations (average reward and steps between rewards), different assessment techniques (immediate and cumulative), and different recovery strategies (increasing the exploration rate and resetting the Q values). All of this was tailored for Q-learning and would not be applicable to other types of cognitive systems. The approach outlined in this dissertation can produce the same perturbation tolerance as observed in the MCL-enhanced Chippy, but is applicable to more types of systems.

Another domain used in early MCL research was TRAINS (Allen *et al.* 1994; 1996; Ferguson, Allen, & Miller 1996) where a human and a computer carry out a natural-language dialog about controlling trains. MCL was used in the detection and resolving of ambiguities (Perlis, Purang, & Andersen 1998; Traum *et al.* 1999). An MCL component was also built into ALFRED (Active Logic for Reason-Enhanced Dialog) (Anderson *et al.* 2004; Josyula, Anderson, & Perlis 2003). Active Logic (Elgot-Drapkin 1988; Elgot-Drapkin & Perlis 1990; Elgot-drapkin *et al.* 1993; Purang 2001; Miller 1993) combines inference rules with a time-tagged knowledge base. ALFRED also uses use-mention distinction (Saka 1998) when processing utterances(Anderson *et al.* 2002). ALFRED is able to ask the user for help when it doesn't know a term and then to use that knowledge later to implement the user's request (Josyula 2005).

11.2 Early Ontology Metacognitive Loop

In 2007, MCL went through a metamorphous to replace many of its domain-specific elements by using three ontologies linked together (Anderson *et al.* 2007). However, instead of using Bayesian inference, MCL would reason from expectations to response by spreading activation (Anderson 1983; Anderson & Pirolli 1984; Cohen & Kjeldsen 1987). Additional domains were also explored including human-computer dialogue (Schmill *et al.* 2007) and the Chippy reinforcement learner (Anderson *et al.* 2008) that is also used as an example in this dissertation.

11.3 Case-based Reasoning

Case-based reasoning (CBR) uses past experiences to solve new problems (Hammond 1986; 1989; 1990; Kolodner 1993). Humans are thought to make use of CBR as we reason from stories (previous experience) (Schank & Leake 1989; Schank 1990) using reasoning by analogy (Leake 1996a) when presented with a new situation. That is, we internalize the basic elements of our experiences as stories. Then, when faced with a new problem, we retrieve a previous experience that is similar to the current one. We use the similar story to provide the outline of a plan for dealing with the current situation. The assumptions underlying the computer use of CBR (Leake 1996b) are that similar problems have similar solutions and that an agent will often encounter problems that it has seen in the past. A temporal MCL can be viewed as using the same general techniques as it uses knowledge about past expectation violations in attempting to resolve new violations.

CBR is generally shown as having four steps (Aamodt & Plaza 1994):

Retrieve Find a past case similar to the current problem.

Reuse Use the similar past case in the solution to the current problem.

Revise Update the past case by noting how well it solved the current problem.

Retain Save the updated case information for successful adapted solutions.

A Level 3: Temporal MCL will also perform these steps, using a frame comparison function to find a similar past exception. It then modifies the Bayesian network from the past exception with the indications of the new exception. When the agent responds that the repair suggestion was successful (or not), MCL saves the updated exception.

The main research in this dissertation was to find a "good" function with which to compare past expectation violations with the current violation. Similarity metrics provide the same function for case-based reasoning (Bento & Costa 1994) and (Veloso & Carbonell 1994). Constructive similarity assessment (Leake 1992; 1995; Leake, Kinley, & Wilson 1997) looked at making the judgement dynamic rather that using a fixed set of criteria.

11.4 Model Based Reflection

Model-based reflection (MBR) (Stroulia 1994; Stroulia & Goel 1995; 1996) also uses a three-phase approach to provide metacognition. The "monitor" phase checks expectations, the "assign blame" phase determines the cause of the failure, and the "redesign" phase makes the necessary corrections. Rather than using a general model of cognitive systems and their failures, MBR uses a detailed model of the problem solver using structure-behavior-function (SBF) models. Having these models allows the expectations to be automatically generated.

MBR uses a model of the agent and the environment when deciding the cause of the problem. This model could become quite large with a large search space and may only be partially observable (Ulam, Goel, & Jones 2004). Rather than searching the complete space, large systems could be broken down into parts. Blame then could be assigned at the top level (Jones & Goel 2005). If the agent has several tasks, each of which is controlled by reinforcement learning, MBR could aid the agent in deciding which task's learning needed modification based on the assignment of failure to that part (Ulam *et al.* 2005).

As systems become more complicated, there is a greater chance that system faults will damage more components so being able to detect and isolate faults becomes more important (Tinós, Terra, & Bergerman 2002; Tinós & Terra 2002). The NASA Deep Space I spacecraft[1] is equipped with a MDR system that monitors its systems and performs failure analysis and correction (Williams & Nayak 1996; Williams & Nyak 1999). Nuclear power systems also can benefit from systems that detect and isolate faults automatically (Hardy, Miller, & Hajek 1992; Hines & Hajek 1995; Hines, Miller, & Hajek 1996).

Instead of using a model of the agent and the domain, MCL uses a model of how agent fail and then repair themselves as captured in the three ontologies. This should allow MCL to assist agents even when many aspects of the of the agent and/or domain are not known.

[1]http://nmp.jpl.nasa.gov/ds1/DS1_Extended_Mission.pdf

11.5 Multi-agent Metacognition

One approach to handling problems (or perturbations) in multi-agent systems is to task an agent with monitoring and controlling the other agents (Hägg 2000). The sentinel agents act as a metacognitive control on the collection of task-solving agents (Figure 11.1). The sentinel monitors the communication between agents. If an agent is acting outside the model of the application-specific interaction plan, the sentinel can take action to correct the situation such as killing an agent, or informing other agents to ignore it.

Sentinels can also be part of a general purpose three-phase monitoring scheme (Dellarocas & Klein 2000). Rather than an application-specific model of the agent interactions, these sentinels use a knowledge base of error conditions, causes, and responses. Table 11.1 contrasts this approach with the MCL phases and ontologies.

Table 11.1. Monitor and control in MCL and Sentinels

MCL		Sentinels	
Phase	Ontology	Phase	KB
Notice	Indications	Instrumentation	Failure
Assess	Failure	Diagnosis	Exception
Guide	Response	Resolution	Resolution

The similarities between metacognition for single agents and sentinels for multi-agents systems allows the transfer of ideas between the two. This is particularly apparent when the fault-handling approach is abstracted using ontologies (or knowledge bases) to hold domain-specific information.

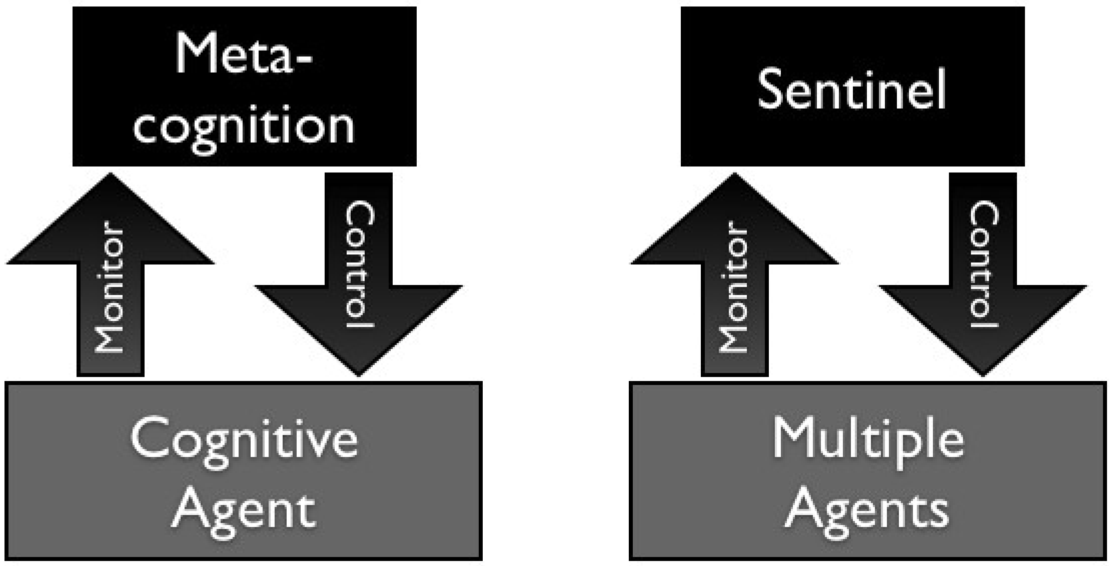


FIG. 11.1. Monitoring a multi-agent system with a sentinel is isomorphic to using metacognition with a single cognitive agent.

Chapter 12

FUTURE WORK

Adding the Metacognitive Loop with temporal knowledge has been shown to improve the operation of the Mars Rover. However, more can be done at this level of metacognition as well as developing further enhancements. Also, MCL can be applied to different domains to show that the improvement in performance for the Mars Rover was not a fluke. This chapter examines several different areas of exploration.

12.1 Revisiting the Current System

Experimental trials using the Mars Rover simulation were able to show that MCL Level 3: Temporal is statistically better at assisting the agent during perturbations than MCL Level 2 : Evaluative. Will this improvement continue to be shown if the experiments were changed? This section looks at a several changes that could be done within the current testing framework to expand the breadth and intensity of testing.

Expand the number of perturbations The seven perturbations used were chosen in the belief that they would exercise both the Rover and MCL. But a different (or larger) mix of perturbations might show different results than the seven chosen. In Section 3.2 many more perturbations were described than those used. Rather than repeating the same seven perturbations in each experimental run, a random set of seven could be chosen.

Improve the frame comparison functions While most of the frame comparison functions performed better than level 0 and 1 MCL agents, there was no one perfect function. Given the available frame information, it should be possible to construct frame comparison functions using decision tables (or other learning technique) that would perform better than the simple ones that were used. A learned comparison function could be trained using the seven perturbations used in this dissertation and then tested against a different set (such as the random perturbation in the previous paragraph).

Changing the task A triple panoramic tour was chosen as it was easy to explain, had fixed divisions for changing the perturbations, and forced the Rover to perform actions multiple times. A triple photographic tour could easily be substituted or a mixed tour consisting of both panoramic and photo tours. Rather than changing the perturbations at tour boundaries, the perturbations could be switched during the execution of the tours. Experiments could be run at low and high volatility of switching to see how well MCL was able to improve performance as a function of the degree of perturbation.

More repair options A larger MCL ontology with more repair nodes to activate would give MCL more possible repair suggestions. Having more possible repairs to suggest could give MCL a greater ability to suggest repairs that improve the Rover's performance. This might allow MCL to improve the agent's performance across a greater range of perturbations.

Improve Conditional Probability Tables Many of the nodes in the MCL Ontology have links whose conditional probabilities are 0.5/0.5. Having more realistic probabilities should improve the MCL repair suggestions. However, at this time, the setting of the probabilities is more art than science.

Multiple Perturbations An agent with Evaluative MCL was able to recover from all of the single perturbations. It was unable to complete the tour when two perturbations were presented serially. An agent with Temporal MCL has been shown to handle the problem of two serial perturbations, but would it do as well if both perturbations were presented simultaneously? To make the performance measures comparable to the experiments in this dissertation, there should be an unperturbed panoramic tour, a second panoramic tour with both perturbations introduced at the start, and then a final unperturbed tour.

12.2 Improving the Level 3 System

This section looks at improvements to the Level 3 MCL to make it more usable to the designer of robotic agents.

12.2.1 Automatic Expectation Generation

The MCL NAG cycle starts when an exception has occurred. It is required that the designer of the system specify the exception conditions. For many maintenance conditions, these are fairly obvious and easy to create: internal temperature will not exceed 150 degrees, battery power will not drop below 5 percent.

Perhaps a better solution would be to specify a minimal set of absolute expectations and then have MCL learn and incorporate additional expectations. This has several advantages:

Minimizing human effort Since only a limited set of expectations would have to be given to MCL, the domain programmer's task would be limited to specifying that limited set and not a broad range of expectations.

Minimizing the exception-checking overhead A system can have a great many maintenance expectations - most of which will never be violated. Such expectations degrade the system as they need to be checked against the sensor values just as often as expectations that may be violated. By generating expectations through learning, only expectations that have the potential for violation would be created.

Improve problem detection As problems are identified, expectations would be created to detect them earlier. This could allow identification of a reoccurrence of a problem early enough to avoid or lessen the consequences.

The automatic generation of expectations would have to be tempered with commonsense reasoning or other heuristics to prevent generating expectations that do not improve the efficiency of the host system.

12.2.2 Automatic Ontology Expansion/Linking

The structure (node and linkages) of the MCL ontologies were created based on experience in the initial problem domains and modified as additional domains and analysis was undertaken. It is, however, a static model and is certainly not optimal for all situations. As with the automatic generation of expectations, the automatic generation of ontology nodes and linkages has the potential to improve the operation of MCL, particularly in new domains. The same caveat applies that any changes must improve the efficiency of the host system and may require extensive heuristics to implement.

12.2.3 Application to Multi-agent Systems

All of the problem domains discussed and evaluated in this proposal are with single agents. MCL can be directly applied to individual agents in a multi-agent system. These agent can be either normal agents or sentinel agents.

When using MCL to monitor and control multiple agents, additional concrete responses would be needed along with the corresponding augmentations to the Response ontology. The Failure ontology would need to be expanded to incorporate nodes for agent communication and coordination failures. Agents can be treated as both sensors and effectors of the sentinel agents.

12.2.4 Transferring Learning with MCL Networks

As MCL works with a host system, the conditional probabilities on the intra-ontological and inter-ontological links change to reflect the experience of what suggestions were effective strategies in coping with the failed expectations. Each host system is different, with different expectations and available concrete responses, but there should be a way to apply a tuned set of conditional probabilities from one system to another.

12.2.5 Modeling Dynamic Environments

A basic tenet of this paper is that MCL provides agents with a mechanism for coping with dynamic environments so that such mechanisms do not need to be crafted into the agents themselves. What exactly is a dynamic environment and how is it quantified? Is it possible to create quantitative or predictive models of dynamic environments and how could these models be used to improve the operation of the NAG cycle within MCL?

12.3 Developing Level 4: Evolving Systems

As in the temporal level, the agent's metacognitive system evaluates the current exception(s) as well as any past exceptions and repair attempts when choosing the appropriate response. Additionally, the metacognitive system adjusts its evaluation procedure/paramaters based on the success and failures of the repairs. While the goal of MCL is to make the correct suggestion as often as possible, it is also important to make as few bad suggestions as it can. Agents can more easily tolerate getting the second best suggestion than they can getting the second worst one.

There are several items within MCL that could be changed if a learning mechanism is in place that would examine the exceptions and the repair actions.

Repair costs The repair costs for the Mars Rover were defined in the repair nodes of the Response Ontology (see Appendix B). These were set before the experiments were run. They were loosely based on the energy cost for the Rover to implement them. MCL could monitor the change in the agent from the time of the suggestion to the time that the repair is acknowledged and adjust the repair cost accordingly. A simple learning method would be to have repairs that succeed have their costs adjusted downward while ones that don't would be moved upwards.

Revise expectations While it wasn't used in the Mars Rover ontologies, MCL has a repair that instructs an agent to "Revise Expectations." The idea being that the agent may be giving MCL too low a threshold in the expectation so that exceptions are occurring where there really isn't a problem. Expectation thresholds may need to be lowered, as well, if the exception isn't signaled until the problem grows large enough that it might have been better handled at an earlier stage.

Update conditional probabilities The conditional probabilities of the MCL Ontologies are set from configuration files when MCL is initialized. As MCL makes suggestions and the agent replies that they were successful or not, MCL could adjust the conditional probabilities tables to favor successful repair.

Update ontology links A more difficult change to the ontologies than adjusting the probabilities on existing ontology links is the addition and removal of ontology links. If, however, experience shows that certain indications are best resolved with a specific repair, then adding a link to positively reinforce that dependency would improve MCL's response to the same problem in the future.

All of the approaches above may benefit from having a meta-MCL monitor and guide the agent's MCL as it alters itself to better serve the agent. The meta-MCL could suggest when a change should be made and what part of MCL (expectations, ontology, or costs) might be best changed.

12.4 Developing Level 5: Anticipating Systems

MCL reacts when the agent's expectations are violated. The point at which the exception occurs may not be the point when the perturbation first caused an unnoticed problem. If MCL could reason backward from the exception to the time of the initial problem, MCL could then help the agent avoid the problem.

For example, if, when traveling between two specific points, the Rover always picked up dust that would later cause a problem with the motors, MCL might be able to examine the operational traces to determine that the cause of the motor performance exception was traveling along a certain path and suggest that the agent choose a different path from now on. Rather than just treating the symptom of the problem (slow motor performance correctable by blowing away the dust), MCL would be helping the agent avoid the cause of the problem (a dusty path).

12.5 Alternative Domains

All of the above research can be done within the Mars Rover domain used in this dissertation. Even given the limited locations and actions of the Rover, there is still enough depth that much can be done with it. However, many domains have been used in AI planning, problem solving, and learning. It would be interesting to apply MCL to those domains as well.

Chippy The Chippy grid world was used as an example in the beginning of this dissertation. It was the subject of much of the early pre-ontology MCL work (Anderson *et al.* 2006). Now that a temporal, ontology-based MCL is available, it would be interesting to revisit Chippy and see if the latest MCL approaches meet or exceed the earlier ones.

WinBolo A domain used in MCL research as the transition was being made from hard coded metacognition to ontologies was the game Bolo (Anderson *et al.* 2007). The MCL Bolo player used a hierarchical task network (HTN) planner (Ghallab, Nau, & Traverso 2004; Lekavỳ & Návrat 2007; Nau *et al.* 2003). When MCL detected an expectation exception, it could use means-end analysis and operator refinement in its repairs (Gil 1994; Wang 1995).

WinBolo (a Windows implementation of the game) has multiple players (or AIs) driving tanks to explore the landscape and destroy other tanks, individually or in teams. This domain is much like the Mars Rover domain in that there are multiple goals in play, planning needs to be done to achieve those goals, and a dynamic environment (mainly the other tanks and the building and destruction of buildings, roads, and stationary defenses.) With team play, WinBolo can be a platform for research in the use of MCL with multiple agents.

Wumpus The game of "Hunt the Wumpus" has been around since the early days of interactive computing. Russell and Norvig (1995) describe simplified variation that on a rectangular grid. A metacognitive agent call INTRO has already been developed to explore that version (Cox 2005a; 2007). As with the Chippy grid world, the challenge will be to see how well the latest MCL compares to previous work.

FreeCiv The open source close of Sid Meier's Civilization$^{\text{TM}}$, FreeCiv[1] has been used for AI learning by researchers at Georgia Tech (Jones & Goel 2005) and others (Hinrichs & Forbus 2007). It is a multiplayer game implemented with a server and multiple clients. It comes bundled with a set of AI players and is supported on a number of platforms. Unlike the Mars Rover, Chippy, and Wumpus domains where the agent is directed to move through and react with the environment, FreeCiv casts the agent in the role of a supervisor who directs many others to build cities, explore, and develop resources, all the while competing with other agents attempting to do the same things. The task will be to learn and evolve strategies for playing the game with MCL suggesting when current strategies need revising or when the focus should shift to a different set of tasks.

[1]freeciv.wikia.com

Chapter 13

CONCLUSIONS

The six-level taxonomy (from Bereft to Anticipating), that I developed to divide metacognitive systems according to their capabilities, provided me with a way to deconstruct MCL in order to analyze MCL's ability to assist agents. In the Mars Rover domain, the limitations of the Level 0: Bereft, Level 1: Instinctive, and Level 2: Evaluative were easily demonstratable. A Motivated Mars Rover with no MCL assistance (Level 0: Bereft) was unable to complete its task when either the P1: Partial charging or P7: Block path perturbations were present. A Rover with Level 1: Instinctive assistance also failed with either of those two perturbations. And, while a Rover with Level 2: Evaluative assistance was able to successfully overcome the P1: Partial charging perturbation, it failed with P2: Reduced capacity and P7: Blocked path. It was only when the MCL level was raised to Level 3: Temporal that the Mars Rover was able to complete its task with any single perturbation.

The research presented here focused on identifying a temporal comparison function that would allow a Level 3: Temporal MCL to successfully assist a Mars Rover in the face of multiple perturbations presented serially. Instead, four such functions were found. One function compared domain-specific items directly related to the exceptions while a second compared the translation of the exception onto the domain-independent Indications ontology of MCL. A third function combined these two tests. These three functions also performed well when there was only a single perturbation. A fourth function randomly announced that two exceptions were the same or different. This function did not perform as well with a single perturbation, however, it was the only function that assisted the Mars Rover to complete all tasks given any combination of two perturbations.

For the Mars Rover simulations with multiple perturbations, MCL Level 1: Instinctive (replanning only) is incapable of completing about 41% of the trials. MCL Level 2: Evaluative is slightly worse with 43% of the trials failing. This is due to bad advice being given in some cases and repeated until the end of the experiment. Such problems have been suggested (Wilson & Schooler 1991) as a reason to avoid too much introspection. MCL Level 3: Temporal using any one of the several top temporal comparison functions, fails less than 1% as they only rarely continuously offered the same bad advice. Hopefully, future research will determine a temporal comparison function that will always offer good advice or, at least, never repeats the same bad advice.

Appendix A

TABLES WITH EXPERIMENT RESULTS

The Mars Rover experiments were run in two parts of 50 repetitions. The main set was started on the May 8, 2010 and were completed May 31, 2010. A second set of experiments using just MCL level 1, that only issued a re-planning suggestion, were started on October 26, 2010 and were completed November 3, 2010. The tables in this appendix and in the Results chapter were all created directly from all 100 experiment CSV files.

To rerun the experiments and recreate the latex figures and tables issue the command below. When running on OSX (instead of Linux) change /home to /Users).

```
MCL_CONFIG_PATH=/home/dean3/lib/mcl
export MCL_CONFIG_PATH
rm *.csv
rm *.zip
python Dissertation.py -x -o5150 -3 -r50
python Results.py -t current.tex -s ../current *.csv
```

1/2	*P0*	*P1*	*P2*	*P3*	*P4*	*P5*	*P6*	*P7*	*Avg*
P0	145	473	157	144	148	153	147	458	228
P1	448	500	454	440	446	446	446	498	459
P2	154	480	159	154	158	163	153	493	239
P3	147	454	155	149	149	146	146	429	221
P4	147	474	155	148	151	162	147	472	232
P5	155	447	158	151	156	156	147	458	228
P6	145	433	153	147	147	156	147	457	223
P7	466	493	465	478	472	454	473	500	475
Avg	226.0	469.0	232.0	226.0	228.0	230.0	226.0	471.0	288.5

Table A.1. Average steps for perturbed Panoramic Tour with M0: Bereft

1/2	*P0*	*P1*	*P2*	*P3*	*P4*	*P5*	*P6*	*P7*	*Avg*
P0	147	474	151	147	148	153	146	450	227
P1	481	495	467	440	479	458	486	500	475
P2	155	453	163	152	154	158	156	492	235
P3	146	467	153	147	145	154	146	478	229
P4	146	459	154	146	145	161	145	493	231
P5	148	466	155	151	154	158	149	493	234
P6	146	446	153	150	146	153	147	471	226
P7	450	493	466	471	478	466	471	500	474
Avg	227.0	469.0	233.0	226.0	231.0	233.0	231.0	485.0	291.875

Table A.2. Average steps for perturbed Panoramic Tour with M1: Instinctive

1/2	*P0*	*P1*	*P2*	*P3*	*P4*	*P5*	*P6*	*P7*	*Avg*
P0	145	159	500	234	146	153	146	457	242
P1	155	170	500	244	157	163	160	500	256
P2	500	500	500	500	500	500	500	500	500
P3	232	244	500	319	229	238	232	477	308
P4	148	156	500	237	149	153	146	500	248
P5	150	163	500	240	147	157	151	452	245
P6	147	159	500	237	145	154	147	493	247
P7	493	500	500	484	500	473	493	500	492
Avg	246.0	256.0	500.0	312.0	247.0	249.0	247.0	485.0	317.75

Table A.3. Average steps for perturbed Panoramic Tour with M2: Evaluative

1/2	*P0*	*P1*	*P2*	*P3*	*P4*	*P5*	*P6*	*P7*	*Avg*
P0	148	159	163	148	149	154	146	158	153
P1	156	168	171	158	160	166	156	170	163
P2	161	182	177	160	164	170	163	171	168
P3	151	157	163	147	151	163	147	156	154
P4	149	169	161	148	147	158	149	155	154
P5	151	160	164	148	152	158	150	162	155
P6	148	159	163	145	147	153	148	156	152
P7	158	423	168	156	160	165	159	157	193
Avg	153.0	197.0	166.0	151.0	154.0	161.0	152.0	161.0	161.875

Table A.4. Average steps for perturbed Panoramic Tour with M3: Temporal

1/2	*P0*	*P1*	*P2*	*P3*	*P4*	*P5*	*P6*	*P7*	*Avg*
P0	147	158	160	148	147	156	147	316	172
P1	156	170	173	158	157	159	156	199	166
P2	159	179	177	159	162	169	161	212	172
P3	149	158	165	148	149	154	149	281	169
P4	147	164	164	147	145	153	148	275	167
P5	149	164	168	151	150	156	148	281	170
P6	146	425	155	146	149	156	147	465	223
P7	309	463	473	332	288	342	257	347	351
Avg	170.0	235.0	204.0	174.0	168.0	181.0	164.0	297.0	199.125

Table A.5. Average steps for perturbed Panoramic Tour with F1: First

1/2	*P0*	*P1*	*P2*	*P3*	*P4*	*P5*	*P6*	*P7*	*Avg*
P0	147	160	500	230	148	154	148	486	246
P1	157	168	500	241	156	167	157	479	253
P2	500	500	500	500	500	500	500	500	500
P3	231	250	500	326	235	241	227	478	311
P4	149	162	500	243	148	153	144	500	249
P5	154	163	500	244	153	161	155	485	251
P6	147	160	500	227	149	154	147	471	244
P7	486	474	500	494	493	480	493	500	490
Avg	246.0	255.0	500.0	313.0	248.0	251.0	246.0	487.0	318.25

Table A.6. Average steps for perturbed Panoramic Tour with F2: New

1/2	*P0*	*P1*	*P2*	*P3*	*P4*	*P5*	*P6*	*P7*	*Avg*
P0	147	164	161	147	148	153	147	166	154
P1	164	185	172	161	165	174	160	181	170
P2	158	180	168	160	158	164	158	184	166
P3	147	171	160	149	152	150	148	157	154
P4	145	167	159	149	147	153	147	162	153
P5	155	171	167	156	150	157	153	164	159
P6	146	166	160	146	148	150	147	167	153
P7	167	224	202	172	172	172	169	164	180
Avg	154.0	179.0	169.0	155.0	155.0	159.0	154.0	168.0	161.625

Table A.7. Average steps for perturbed Panoramic Tour with F3: Random

1/2	*P0*	*P1*	*P2*	*P3*	*P4*	*P5*	*P6*	*P7*	*Avg*
P0	145	159	161	147	148	153	145	157	151
P1	159	170	172	156	158	170	159	166	163
P2	160	187	175	162	163	164	160	171	167
P3	148	158	162	148	147	148	147	159	152
P4	147	163	159	147	145	155	149	155	152
P5	149	157	163	158	148	160	149	164	156
P6	148	160	160	147	146	155	147	157	152
P7	157	401	167	159	159	167	159	158	190
Avg	152.0	194.0	165.0	153.0	152.0	159.0	152.0	161.0	161.0

Table A.8. Average steps for perturbed Panoramic Tour with F4: EVS

1/2	*P0*	*P1*	*P2*	*P3*	*P4*	*P5*	*P6*	*P7*	*Avg*
P0	147	156	157	147	148	155	146	156	151
P1	160	172	168	159	155	160	154	168	162
P2	157	470	163	158	155	163	157	169	199
P3	146	160	157	146	147	157	147	158	152
P4	148	159	157	148	150	156	146	158	152
P5	152	161	160	157	153	158	151	161	156
P6	147	156	157	148	148	153	146	155	151
P7	157	175	166	156	162	161	158	159	161
Avg	152.0	201.0	161.0	152.0	152.0	158.0	151.0	161.0	161.0

Table A.9. Average steps for perturbed Panoramic Tour with F5: IIS

1/2	*P0*	*P1*	*P2*	*P3*	*P4*	*P5*	*P6*	*P7*	*Avg*
P0	146	164	154	149	147	151	146	159	152
P1	157	168	167	156	160	162	156	168	161
P2	158	455	160	156	155	164	158	166	196
P3	147	160	156	149	147	153	144	157	151
P4	146	162	156	147	148	154	148	159	152
P5	154	163	160	150	153	165	153	162	157
P6	146	159	156	149	144	147	150	154	150
P7	157	170	169	156	159	170	158	157	162
Avg	151.0	200.0	160.0	152.0	152.0	158.0	152.0	160.0	160.625

Table A.10. Average steps for perturbed Panoramic Tour with F6: EVS and IIS

1/2	*P0*	*P1*	*P2*	*P3*	*P4*	*P5*	*P6*	*P7*	*Avg*
P0	148	161	162	235	147	155	149	387	193
P1	158	174	177	244	157	170	157	177	176
P2	160	172	177	251	162	169	163	379	204
P3	232	247	248	320	229	238	230	386	266
P4	146	155	161	235	150	154	148	351	187
P5	155	167	168	230	153	157	153	370	194
P6	146	158	160	151	148	158	146	359	178
P7	353	420	358	417	358	356	347	389	374
Avg	187.0	207.0	201.0	260.0	188.0	195.0	187.0	350.0	221.875

Table A.11. Average steps for perturbed Panoramic Tour with F7: EVS but not IIS

1/2	*P0*	*P1*	*P2*	*P3*	*P4*	*P5*	*P6*	*P7*	*Avg*
P0	147	156	500	235	146	150	147	479	245
P1	158	168	500	241	158	160	156	480	252
P2	500	500	500	500	500	500	500	500	500
P3	231	249	500	317	228	238	225	478	308
P4	146	156	500	236	147	153	147	450	241
P5	147	164	500	241	153	161	150	466	247
P6	146	156	500	231	146	153	146	486	245
P7	486	479	500	483	492	493	500	500	491
Avg	245.0	254.0	500.0	311.0	246.0	251.0	246.0	480.0	316.625

Table A.12. Average steps for perturbed Panoramic Tour with F8: IIS but not EVS

Appendix B

MCL ONTOLOGY USED FOR EXPERIMENTS

The following two listings comprise the MCL ontologies used for the Mars Rover simulation. They were created by Matt Schmill of the UMBC/UMCP MCL working group. They have been slightly modified from the version included with the MCL code. The main changes are in the cost model which reflect the cost of implementing the repairs by the Mars Rover.

Listing B.1. Basic MCL Ontology

```
# BASIC.ONT .... basic MCL ontology description file
# modified for mars rover simulation

# node types:
#
# hostProp - host properties
# genInd - general purpose indication node
# concInd - concrete (fringe) indication directly activatable by↘
   MCL
# iCore - indication core node
# HII - Host Initiated Indication
# failure - general purpose failure node
# genResponse - general purpose response node
# interactive - boolean interactive response node
# concResponse - concrete (implementable) response node

# link types:
#
```

```
# > intraontological
# link abstraction(src=,dst=)
#      - from specific (src) node to more general (dst)
# link IFC(src=,dst=)
#      - Indication Fringe to Core
# link specification(src=,dst=)
#      - from abstract to specific (response ontology base type)
#
# > interontological
# link diagnostic(src=,dst=)
#      - link from indication to failure
# link inhibitory(src=,dst=)
#      - link from indication to response (inhibiting response)
# link support(src=,dst=)
#      - link from indication to response (supporting response)

include CORE_indicationFringe

ontology indications (

  # much of this ontology is currently in CORE_iF
  # if it is in dynamicILinks, it should be in iF

  node iCore(name=resourceUnchanged,
           doc="a␣resource␣level␣apparently␣did␣not␣change.")
  node iCore(name=stuck,
           doc="a␣spatial␣sensor/expectation␣indicates␣an␣↘
              expected␣movement␣did␣not␣occur.")

  node HII(name=sensorVerifiedBroken,
         doc="the␣host␣is␣verifying␣that␣a␣sensor␣appears␣↘
            nonfunctional.")
  node HII(name=sensorVerifiedWorking,
         doc="the␣host␣is␣verifying␣that␣a␣sensor␣of␣interest␣↘
            appears␣to␣be␣working.")

)

ontology failures (
  node failure(name=failure,
             doc="the␣class␣of␣all␣failures.")
  node failure(name=knowledgeError,
                 doc="class␣of␣failures␣pertaining␣to␣internal␣↘
                    knowledge␣and␣representations.")

    node failure(name=plantError,
                 doc="class␣of␣failures␣pertaining␣to␣the␣physical↘
                    ␣agent.")
    node failure(name=modelError,
                 doc="class␣of␣failures␣pertaining␣to␣internal␣↘
                    representations.")
    node failure(name=predictiveModelError,
                 doc="class␣of␣failures␣in␣which␣inaccurate␣↘
                    predictions␣made␣by␣models␣are␣causing␣↘
                    anomalies.")
    node failure(name=proceduralModelError,
                 doc="class␣of␣failures␣in␣which␣inadequacies␣in␣a↘
                    ␣procedural␣model␣is␣causing␣anomalies.")
    node failure(name=effectorError,
                 doc="class␣of␣failures␣in␣which␣an␣effector␣is␣↘
                    not␣operating␣to␣spec.")
    node failure(name=sensorError,
                 doc="class␣of␣failures␣in␣which␣a␣sensor␣is␣not␣↘
                    operating␣to␣spec.")
    node failure(name=sensorNoise,
                 doc="anomaly␣is␣caused␣by␣noise␣outside␣of␣↘
                    specified␣parameters.")
    node failure(name=sensorMiscalibrated,
                 doc="anomaly␣is␣caused␣by␣misconfiguration␣or␣↘
                    calibration␣of␣a␣sensor.")
    node failure(name=sensorMalfunction,
                 doc="anomaly␣is␣caused␣by␣sensor␣fault␣other␣than↘
                    ␣noise.")
    node failure(name=sensorStuck,
                 doc="anomaly␣is␣caused␣by␣a␣sensor␣that␣is␣no␣↘
                    longer␣changing␣according␣to␣specification.")


)

ontology responses (
    node genResponse(name=response,
                     doc="root␣of␣all␣responses.")
    node genResponse(name=internalResponse,
                     doc="response␣taken␣internally␣to␣the␣host.")
```

```
node genResponse(name=externalResponse,
                 doc="response␣taken␣with␣external␣help.")
node genResponse(name=plantResponse,
                 doc="response␣pertaining␣to␣the␣physical␣↘
                     agent.")

node genResponse(name=systemResponse,
                 doc="response␣pertaining␣to␣the␣soft␣agent.")
node genResponse(name=runDiagnostic,
                 doc="perform␣a␣diagnostic␣of␣the␣agent␣↘
                     aparatus.")

node genResponse(name=amendKnowledgeBase,
                 doc="general␣changes␣to␣the␣host's␣KB.")
node genResponse(name=amendPredictiveModels,
                 doc="modify/revise␣predictive␣models.")
node genResponse(name=amendProceduralModels,
                 doc="modify/revise␣procedural␣models.")

node interactive(name=runSensorDiagnostic, cost=100,
                 doc="instructs␣the␣agent␣to␣check␣for␣sensor␣↘
                     faults.",
                 code=crc_sensor_diag,
                 runOnce=true,
                 yes=sensorVerifiedBroken,
                 no=sensorVerifiedWorking)
node concResponse(name=resetSensor, cost=100,
                  code=crc_sensor_reset,
                  doc="physical␣restart␣of␣sensor.")
node concResponse(name=runEffectorDiagnostic, cost=100,
                  doc="instructs␣the␣agent␣to␣check␣for␣↘
                      nonfunctional␣effectors.",
                  code=crc_effector_diag)
node concResponse(name=resetEffector, cost=100,
                  doc="physical␣restart␣of␣an␣effector/↘
                      effector␣group.",
                  code=crc_effector_reset)
node concResponse(name=rebuildPredictiveModels, cost=50,
                  doc="rerun␣batch-mode␣predictive␣model␣↘
                      generators.",
                  code=crc_rebuild_models)
node concResponse(name=tryAgain, cost=5,
                    doc="instruct␣the␣host␣to␣retry␣the␣failed␣↘
                        activity.",
                    code=crc_try_again)

)

linkage all (

  # I->F I links

  link IFC(src=resource, dst=resourceUnchanged)
  link IFC(src=missed-unchanged, dst=resourceUnchanged)
  link IFC(src=spatial, dst=stuck)
  link IFC(src=missed-unchanged, dst=stuck)

  # intra F links

  link abstraction(src=knowledgeError, dst=failure)
  link abstraction(src=plantError, dst=failure)
  link abstraction(src=modelError, dst=knowledgeError)
  link abstraction(src=predictiveModelError, dst=modelError)
  link abstraction(src=proceduralModelError, dst=modelError)
  link abstraction(src=sensorError, dst=plantError)
  link abstraction(src=sensorNoise, dst=sensorError)
  link abstraction(src=sensorStuck, dst=sensorError)
  link abstraction(src=sensorMiscalibrated, dst=sensorError)
  link abstraction(src=sensorMalfunction, dst=sensorError)
  link abstraction(src=effectorError, dst=plantError)

  # intra R links

  link specification(src=response, dst=internalResponse)
  link specification(src=internalResponse, dst=plantResponse)
  link specification(src=internalResponse, dst=systemResponse)
  link specification(src=plantResponse, dst=runDiagnostic)
  link specification(src=plantResponse, dst=resetEffector)
  link specification(src=plantResponse, dst=resetSensor)
  link specification(src=runDiagnostic, dst=runSensorDiagnostic)
  link specification(src=runDiagnostic, dst=runEffectorDiagnostic↘
     )
  link specification(src=systemResponse, dst=amendKnowledgeBase)
  link specification(src=amendKnowledgeBase, dst=↘
     amendPredictiveModels)
  link specification(src=amendKnowledgeBase, dst=↘
     amendProceduralModels)
  link specification(src=amendKnowledgeBase, dst=↘
     rebuildPredictiveModels)
  link specification(src=response, dst=externalResponse)
  link specification(src=systemResponse, dst=tryAgain)

  # inter links

  link diagnostic(src=indication, dst=failure)
  link diagnostic(src=sensorsCanFail, dst=sensorMalfunction)
  link inhibitory(src=sensorVerifiedBroken, dst=↘
     runSensorDiagnostic)
  link inhibitory(src=sensorVerifiedWorking, dst=↘
     runSensorDiagnostic)
  # link support(src=sensorVerifiedBroken, dst=fixSensor)
  # link inhibitory(src=sensorVerifiedWorking, dst=fixSensor)
  link diagnostic(src=resourceUnchanged, dst=predictiveModelError↘
     )
  link diagnostic(src=stuck, dst=sensorStuck)
  link diagnostic(src=stuck, dst=effectorError)

  link prescriptive(src=sensorMalfunction, dst=↘
     runSensorDiagnostic)
  link prescriptive(src=predictiveModelError, dst=↘
     amendPredictiveModels)

)
```

Listing B.2. Core MCL Ontology

```
# CORE.ONT .... MCL ontology fringeless description file

# node types:
#
# hostProp - host properties
# genInd - general purpose indication node
# concInd - concrete (fringe) indication directly activatable by↘
   MCL
# iCore - indication core node
# HII - Host Initiated Indication
# failure - general purpose failure node
# genResponse - general purpose response node
# interactive - boolean interactive response node
# concResponse - concrete (implementable) response node

# link types:
#
# > intraontological
# link abstraction(src=,dst=)
#      - from specific (src) node to more general (dst)
# link IFC(src=,dst=)
#      - Indication Fringe to Core
# link specification(src=,dst=)
#      - from abstract to specific (response ontology base type)
#
# > interontological
# link diagnostic(src=,dst=)
#      - link from indication to failure
# link inhibitory(src=,dst=)
#      - link from indication to response (inhibiting response)
# link support(src=,dst=)
#      - link from indication to response (supporting response)

ontology indications (
  # host properties...

  node hostProp(name=sensorsCanFail,
               prop=pci_sensors_can_fail)
  node hostProp(name=effectorsCanFail,
               prop=pci_effectors_can_fail)
  node hostProp(name=actionInPlay,
               prop=pci_action_in_play)
  node genInd(name=hostProp,
             doc="supernode␣for␣host␣properties␣to␣ensure␣full␣↘
                linkage.")

  # provenance (what the sensor is attached to)
  node concInd(name=provenance:object)
  node concInd(name=provenance:self)

  # sensor classes

  node concInd(name=state)
  node concInd(name=control)
  node concInd(name=spatial)
  node concInd(name=temporal)
  node concInd(name=resource)
  node concInd(name=reward)
  node concInd(name=ambient)
  node concInd(name=objectprop)
  node concInd(name=message)
  node concInd(name=counter)
  node concInd(name=unspecified_sc)

  node iCore(name=observable)

  node iCore(name=indication,
          doc="root␣indication␣for␣temporarily␣orphaned␣↘
             fringe␣nodes.")

  # violation type

  node genInd(name=divergence,
             doc="observable␣did␣not␣do␣what␣was␣expected␣(this↘
                ␣is␣general)")
  node genInd(name=aberration,
             doc="observable␣changed␣when␣it␣was␣not␣supposed␣↘
                to.")
  node genInd(name=breakout-low,
             doc="observable␣fell␣through␣a␣floor␣expectation."↘
                )
  node genInd(name=breakout-high,
             doc="observable␣exceeded␣a␣ceiling␣expectation.")
  node genInd(name=missed-target,
             doc="observable␣was␣supposed␣to␣change␣but␣missed␣↘
                the␣target.")
  node genInd(name=short-of-target,
             doc="observable␣lower␣than␣expected␣target.")
  node genInd(name=long-of-target,
             doc="observable␣higher␣than␣expected␣target.")
  node genInd(name=missed-unchanged,
             doc="was␣supposed␣to␣change␣to␣target␣but␣didn't␣↘
                change␣at␣all.")

  node concInd(name=outOfRange,
               doc="value␣out␣of␣specified␣range␣of␣acceptable␣↘
                  values.")
```

```
  node concInd(name=notInSet,
               doc="value␣not␣in␣specified␣set␣of␣acceptable␣↘
                  values.")

  node concInd(name=illegalValue,
               doc="value␣was␣not␣in␣the␣specified␣legal␣range/↘
                  set.")

  node concInd(name=unreachableMCLstate,
               doc="a␣supposedly␣unreachable␣MCL␣state␣was␣↘
                  reached.")

)

linkage all (
  # intra I links

  link abstraction(src=state,dst=observable)
  link abstraction(src=control,dst=observable)
  link abstraction(src=spatial,dst=observable)
  link abstraction(src=temporal,dst=observable)
  link abstraction(src=resource,dst=observable)
  link abstraction(src=reward,dst=observable)
  link abstraction(src=ambient,dst=observable)
  link abstraction(src=objectprop,dst=observable)
  link abstraction(src=message,dst=observable)
  link abstraction(src=counter,dst=observable)
  link abstraction(src=unspecified_sc,dst=observable)

  link abstraction(src=observable,dst=indication)
  link abstraction(src=divergence,dst=indication)
  link abstraction(src=hostProp,dst=indication)

  link abstraction(src=sensorsCanFail,dst=hostProp)
  link abstraction(src=effectorsCanFail,dst=hostProp)
  link abstraction(src=actionInPlay,dst=hostProp)

  link abstraction(src=aberration,dst=divergence)
  link abstraction(src=breakout-low,dst=aberration)
  link abstraction(src=breakout-high,dst=aberration)

  link abstraction(src=missed-target,dst=divergence)
  link abstraction(src=missed-unchanged,dst=missed-target)
  link  abstraction(src=short-of-target, dst=missed-target)
  link  abstraction(src=long-of-target, dst=missed-target)

)
```

Appendix C

MCL TEMPORAL COMPARISON FUNCTIONS

The two listings in this appendix contain the major pieces of the C++ code I wrote to implement the MCL Temporal comparison functions for the experiments. They are the header file (DWtestREB.h) and the implementation file (DWtestREB.cc).

Listing C.1. Header File for MCL Frame Comparison Functions

```
#ifndef MCL_DW_TEST_REB_H
#define MCL_DW_TEST_REB_H

#include "reentrancy.h"

namespace metacog {

  using namespace std;

  /**
     Multiple REBs being tested by Dean Wright for PhD ↘
        dissertaion
  */
  class DWtestREB : public SelectBestREB {
  public:
    DWtestREB() : SelectBestREB() {method_number = 0;};
    DWtestREB(int num) : SelectBestREB() {method_number = num;};
    virtual bool selectFramesForReEntry(mclFrameEntryVector& fev↘
       ,
                                        frameVec& allFrames,
                                        frameVec& selectonVector↘
                                           );
    virtual bool selectCandidateFramesForReEntry(↘
       mclFrameEntryVector& fev,
                                        frameVec& allFrames,
                                        frameVec& selectonVector↘
                                           ,
                                        bool single);
    virtual bool selectForSuccess(mclFrameEntryVector& fev,↘
       mclFrame* frame,
                                    double* score);
    virtual bool selectForFailure(mclFrameEntryVector& fev,↘
       mclFrame* frame,
                                    double* score);
    virtual string describe();
    virtual string name();
  private:
    int method_number;
    int random_frame(mclFrameEntryVector& fev,
                     frameVec& candidates,
                     bool addNone=false);
    int first_frame(mclFrameEntryVector& fev,
                    frameVec& candidates);
   };

};

#endif
```

Listing C.2. Implementation File for MCL Frame Comparison Functions

```
/* ************************************************
 * Implement Renetrant Behavior for Dissertation
 *
 * Dean Earl Wright
 * Fall 2010
 *
 ************************************************

F0: No MCL (Passive)
F1: Always the same frame
F2: Always use a new frame
F3: Random
F4: reuse frame if EVS equal
F5: IIS equal
F6: both IIS and EVS equal
F7: EVS equal but IIS different
F8: IIS equal but EVS different
F9: Current algorithm (Passive)

 ************************************************ */

#include "reentrancy.h"
#include "mclFrame.h"
#include "DWtestREB.h"
#include "stdlib.h"
#include "mclLogging.h"
#include "umbc/text_utils.h"

using namespace std;
using namespace umbc;
using namespace metacog;

bool DWtestREB::selectFramesForReEntry(mclFrameEntryVector& fev,
                                        frameVec& allFrames,
                                                frameVec& selectionVector↘
                                                   ) {

  // 1. Start with no frame selected for reentry
  mclFrame* bestFrame  = NULL;
  frameVec candidates;
  frameVec::iterator fvi;
  int i;

  // 2. Log the request
#ifndef NO_DEBUG

  // 2a. Output header with method number
  char char_num[31];
  //snprintf(char_num, 30, "%d", method_number);
  uLog::annotate(ULAT_NORMAL, "[DWtestREB]::selectFramesForRetry(↘
     "+describe()+")");

  // 2b. Output Frame Entry Vectory
  string desc_fev = fev.describe();
  uLog::annotate(ULAT_NORMAL, "[DWtestREB]::"+desc_fev);

  // 2c. Output number of frames in allFrames
  snprintf(char_num, 30, "%d", (int)allFrames.size());
  uLog::annotate(ULAT_NORMAL, "[DWtestREB]::allFrames("+((string)↘
     char_num)+")");

  // 2d. Loop for all of the Frames
  for (fvi = allFrames.begin();
       fvi != allFrames.end();
       fvi++) {

       // 2e. Output the frame
       string desc_frame  = (*fvi)->describe();
       uLog::annotate(ULAT_NORMAL, "[DWtestREB]::"+desc_frame);
  } // end for
#endif

  // 3. Select canidate frames
  bool single = false;
  if (method_number == 0 || method_number == 9) single = true;
  selectCandidateFramesForReEntry(fev, allFrames, candidates, ↘
     single);
```

```
// 4. Select frame (if any) based on behavior
switch (method_number) {

  // F0: No MCL (Passive)
  // F9: Current algorithm (Passive)
  case 0:
  case 9:
    for (fvi = candidates.begin();
         fvi != candidates.end();
         fvi++) {
      selectionVector.push_back(*fvi);
    } // end for
    return selectionVector.empty();

  // F1: Always the same frame
  case 1:
    if (!candidates.empty())
      bestFrame=candidates[0];
    break;

  // F2: Always use a new frame
  case 2:
    break;

  // F3: Random frame
  case 3:
    i = random_frame(fev, candidates, true);
    if (i >= 0)
      bestFrame = candidates[i];
    break;

  // F4: reuse frame if EVS equal
  case 4:
    i = random_frame(fev, candidates);
    if (i >= 0)
      bestFrame = candidates[i];
    break;

  // F5: IIS equal
  case 5:
    i = random_frame(fev, candidates);
    if (i >= 0)
```

```
        bestFrame = candidates[i];
      break;

    // F6: both IIS and EVS equal
    case 6:
      i = random_frame(fev, candidates);
      if (i >= 0)
        bestFrame = candidates[i];
      break;

    // F7: EVS equal but IIS different
    case 7:
      i = random_frame(fev, candidates);
      if (i >= 0)
        bestFrame = candidates[i];
      break;

    // F8: IIS equal but EVS different
    case 8:
      i = random_frame(fev, candidates);
      if (i >= 0)
        bestFrame = candidates[i];
      break;

    default: //Invalid method_number
      break;

  } // end switch

  // 5. Return the frame and true if we found one
  if (bestFrame) {
    selectionVector.push_back(bestFrame);
#ifndef NO_DEBUG
    string desc_frame = bestFrame->describe();
    uLog::annotate(ULAT_NORMAL, "[DWtestREB]::selected␣"+↘
       desc_frame);
#endif
    return true;
  }

  // 6. Return False if no frame for re-entry
#ifndef NO_DEBUG
    uLog::annotate(ULAT_NORMAL, "[DWtestREB]::selected␣none");
```

```
#endif
  return false;
}

bool DWtestREB::selectCandidateFramesForReEntry(↘
    mclFrameEntryVector& fev,
                                                             frameVec& ↘
                                                                allFrames,
                                                             frameVec& ↘
                                                                selectionVector↘
                                                                ,
                                                             bool single) {
  // 1. Start with no best canidate
  double best_score = −1;
  mclFrame* bestFrame  = NULL;

  // 2. Determine if we are look for success or failure frames
  bool success = fev.isSuccessfulEntry();

  // 3. Loop for all of the frames
  for (frameVec::iterator afi = allFrames.begin();
       afi != allFrames.end();
       afi++) {

    // 4. Get the score for this frame
    double current = −1;
    bool   possible;
    if (success) {
      possible = selectForSuccess(fev,(*afi),&current);
    } else {
      possible = selectForFailure(fev,(*afi),&current);
    }
    if (possible) {
      if (single) {
        if (current > best_score) {
          best_score=current;
          bestFrame=(*afi);
        }
      } else {
       selectionVector.push_back(*afi);
      }  // end else if (single)
    } // end if (possible)
  }
  if (single && bestFrame) {
    selectionVector.push_back(bestFrame);
  }
  return !selectionVector.empty();
}


bool DWtestREB::selectForSuccess(mclFrameEntryVector& fev,
                                 mclFrame* frame,
                                 double* score) {

  uLog::annotate(ULAT_NORMAL, "DWtestREB::selectForSuccess("+
                 textFunctions::num2string(method_number)+")");

  // 1. An referent match is always acceptable
  if (frame->matchesReferent(fev.vRef)) {
    if (score) *score=1.0;
    uLog::annotate(ULAT_NORMAL, "referent␣match");
    return true;
  }

  // 2. Defermine acceptability based on behavior
  switch (method_number) {

    // F0: No MCL - flag for FrameRover.py
    // F9: Current algorithm (Passive)
    case 0:
    case 9:
      if ((fev.vEG == frame->get_vegKey()) &&
          (frame->isMostRecentFrame()) &&
          (frame->in_advice_state())) {
            if (score) *score=1.0;
            uLog::annotate(ULAT_NORMAL, "passive␣match");
            return true;
          }
      break;

    // F1: Always the same frame
    // F2: Always use a new frame
    // F3: Random
    // F4: Reuse frame if EVS equal
    // F5: IIS equal
    // F6: both IIS and EVS equal
    // F7: EVS equal but IIS different
    // F8: IIS equal but EVS different
    case 1:
    case 2:
    case 3:
    case 4:
    case 5:
    case 6:
    case 7:
    case 8:
      if ((fev.vEG == frame->get_vegKey()) &&
          (frame->isMostRecentFrame()) &&
          (frame->in_advice_state())) {
            if (score) *score=1.0;
            uLog::annotate(ULAT_NORMAL,"reb␣match");
            return true;
          }
      break;

    default:
      uLog::annotate(ULAT_NORMAL,"invalid␣number");
      break;
  }

  // 3. Didn't like this frame
  uLog::annotate(ULAT_NORMAL,"no␣match");
  return false;
}

bool DWtestREB::selectForFailure(mclFrameEntryVector& fev,↘
   mclFrame* frame,
                                  double* score) {

  uLog::annotate(ULAT_NORMAL,"DWtestREB::selectForFailure("+
             textFunctions::num2string(method_number)+")");


  // 1. An referent match is always acceptable
  if (frame->matchesReferent(fev.vRef)) {
    if (score) *score=1.0;
    uLog::annotate(ULAT_NORMAL,"referent␣match");
    return true;
  }
```

```
// 2. Defermine acceptability based on behavior
switch (method_number) {

  // F0: No MCL - flag for FrameRover.py
  // F9: Current algorithm (Passive)
  case 0:
  case 9:
    if ((fev.vEG == frame->get_vegKey()) &&
        (frame->evSignatureExists(fev.vEVS))) {
      if (score) *score=1.0;
      fev.vECode = REENTRY_RECURRENCE;
      uLog::annotate(ULAT_NORMAL, "passive␣match");
      return true;
    }
    break;

  // F1: Always the same frame
  case 1:
    if (score) *score=1.0;
    fev.vECode = REENTRY_RECURRENCE;
    uLog::annotate(ULAT_NORMAL, "always␣match");
    return true;

  // F2: Always use a new frame
  case 2:
    uLog::annotate(ULAT_NORMAL, "never␣match");
    return false;

  // F3: Random
  case 3:
    if (score) *score=1.0;
    fev.vECode = REENTRY_RECURRENCE;
    uLog::annotate(ULAT_NORMAL, "should␣be␣random␣but␣always␣↘
        matches");
    return true;

  // F4: reuse frame if EVS equal
  case 4:
    if (frame->evSignatureExists(fev.vEVS)) {
      if (score) *score=1.0;
      fev.vECode = REENTRY_RECURRENCE;
      uLog::annotate(ULAT_NORMAL, "EVS␣match");
    return true;
  }
  break;

  // F5: IIS equal
  case 5:
    if (frame->isSignatureExists(fev.string_iis())) {
      if (score) *score=1.0;
      fev.vECode = REENTRY_RECURRENCE;
      uLog::annotate(ULAT_NORMAL, "IIS␣match");
      return true;
  }
  break;

  // F6: both IIS and EVS equal
  case 6:
    if ((frame->isSignatureExists(fev.string_iis())) &&
        (frame->evSignatureExists(fev.vEVS)) ) {
      if (score) *score=1.0;
      fev.vECode = REENTRY_RECURRENCE;
      uLog::annotate(ULAT_NORMAL, "IIS␣and␣EVS␣match");
      return true;
  }
  break;

  // F7: EVS equal but IIS different
  case 7:
    if ((!frame->isSignatureExists(fev.string_iis())) &&
        (frame->evSignatureExists(fev.vEVS)) ) {
      if (score) *score=1.0;
      fev.vECode = REENTRY_RECURRENCE;
      uLog::annotate(ULAT_NORMAL, "!IIS␣and␣EVS␣match");
      return true;
  }
  break;

  // F8: IIS equal but EVS different
  case 8:
    if ((frame->isSignatureExists(fev.string_iis())) &&
        (!frame->evSignatureExists(fev.vEVS)) ) {
      if (score) *score=1.0;
      fev.vECode = REENTRY_RECURRENCE;
      uLog::annotate(ULAT_NORMAL, "!IIS␣and␣!EVS␣match");
```

```
        return true;
      }
      break;

      default: //Invalid method_number
        uLog::annotate(ULAT_NORMAL, "invalid␣number");
        break;

    } // end switch(method_number)

  // 3. Didn't like this frame
  uLog::annotate(ULAT_NORMAL, "not␣a␣match");
  return false;
}


// ↘
   ----------------------------------------------------------------------------↘

// Algorithms to select the best frame from a set of canidates ↘
   or not.
// ↘
   ----------------------------------------------------------------------------↘

int DWtestREB::random_frame(mclFrameEntryVector& fev,
                            frameVec& candidates,
                            bool addNone) {

  // 1. No canidates, return new frame
  if (candidates.empty()) {
    return -1;
  }

  // 2. Get method_number of choices (size maybe plus one)
  long size = candidates.size();
  long choices = size;
  if (addNone) ++choices;

  // 3. Get random choice
  ldiv_t m = ldiv(random(), choices);

  // 4. Return new frame if not selecting a frame
  if ((unsigned int)m.rem >= candidates.size())
```

```
    return -1;

  // 5. Return method_number of canidate frame to use
  return m.rem;
}

int DWtestREB::first_frame(mclFrameEntryVector& fev,
                           frameVec& candidates) {
  // 1. No canidates, return new frame
  if (candidates.empty()) {
    return -1;
  }

  // 2. Else return the first frame
  return 0;
}


string DWtestREB::describe() {
  string what = "???";
  switch (method_number) {
    case 0: what = "0:␣Passive"; break;
    case 1: what = "1:␣First"; break;
    case 2: what = "2:␣New"; break;
    case 3: what = "3:␣Random"; break;
    case 4: what = "4:␣EVS"; break;
    case 5: what = "5:␣IIS"; break;
    case 6: what = "6:␣EVS␣and␣IIS"; break;
    case 7: what = "7:␣EVS␣but␣not␣IIS"; break;
    case 8: what = "8:␣IIS␣but␣not␣EVS"; break;
    case 9: what = "9:␣Passive"; break;
    default: what ="?:␣Invalid␣method_number"; break;
  }
  return "DW_REB=" + what;
}

string DWtestREB::name() {
  switch (method_number) {
    case 0: return "zero";
    case 1: return "one";
    case 2: return "two";
    case 3: return "three";
    case 4: return "four";
    case 5: return "five";
    case 6: return "six";
    case 7: return "seven";
    case 8: return "eight";
    case 9: return "nine";
  }
  return "???";
}
```

Appendix D

PYTHON CODE USED IN EXPERIMENTS

I wrote the Mars Rover domain, experiment scaffolding, and various utilities in Python.[1] There are four sets of python files.

Mars Rover Agent The files in Table D.1 implement a series of Mars Rovers. Each one builds on the one before to implement additional features. R0, R1, and R2 implement the basic hardware of the Mars Rover. R3 and R4 add perturbations. R5 adds the STRIPS planner creating a Level 0: Bereft agent with no metacognitive supervision. R6 and R7 implement the multi-level, goal-oriented, motivated Rover acting as a Level 1: Instinctive metacognitive agent. R8 and R9 provide scripting to allow for complicated experiments. RA provides a visualization of the Rover. RB interfaces with MCL to get a Level 2: Evaluative Rover and RC, with selectable frame comparison functions, provides a Level 3: Temporal Rover. Finally, RD implements a Rover dedicated to running the triple panoramic tour used in the dissertation experiments.

Mars Rover Agent Utility Files These files are used by the Mars Rover agent. This set (Table D.2) includes files that just enumerate constants as well as those that implement important sub-systems.

[1]www.python.org

File	*Lines*	*Description*
BasicRover.py	972	R0: Implements the Coddington commands and environment
ExtendedRover.py	681	R1: Adds additional commands
SensorRover.py	345	R2: Provides sensor for use in motivations and expectations
NoisyRover.py	827	R3: Allow sensors to be less than perfect
PerturbRover.py	602	R4: Adds many possible perturbations to the Rover and the environment
PlanRover.py	1062	R5: Adds STRIPS planner
MultiLevelRover.py	862	R6: Provides multi-level goals
MotivateRover.py	583	R7: Adds selectable sets of motivations (stimulus –> response)
ScriptRover.py	1243	R8: Experiment setup and takedown
ChapterRover.py	564	R9: And then support for multiple scripts
VisualRover.py	1058	RA: GUI display and interface
MCLRover.py	1056	RB: Communicates to MCL via sockets
FrameRover.py	559	RC: Allows selecting frame comparison function to use for experiment
DisseminationRover.py	774	RD: Runs triple panoramic tour experiment for dissertation

Table D.1. Mars Rover Domain Experiments Python Files

Dissertation Related Programs The files listed in Table D.3 are not part of the Mars Rover. They relate to running experiments, generating content for the dissertation, and backing up the experiment and dissertation files.

Chippy Demonstration Programs The files in Table D.4 are for the Chippy grid world used to demonstrate the metacognition levels in Chapter2.

Figure D.1 shows the import relationships between the Python code files for the Mars Rover simulation. And Figure D.2 for the Python files for the Chippy demonstration.

File	*Lines*	*Description*
Levels.py	32	Multilevel Rover goal hierarchy
MCLLevels	82	Metacognitive levels
MCLResponse.py	265	Decode and encapsulate suggestions from MCL
MCLServer.py	554	Manages socket connection to MCL
Motivation.py	166	Generic motivation: sensor, test, urgency and action
RoverAction.py	548	Describes a single Rover action
RoverMotivation.py	400	Rover motivation groups
RoverScript.py	1095	Implements mini-language to define Rover experiments
Sensors.py	135	Defines single and multiple sensors
perturbations.py	468	Creates anomalies in Rover or environment
rstrips.py	1024	STRIPS planner
tokenizer.py	222	Simple parser for action scrips

Table D.2. Mars Rover Agent Utility Python Files

File	*Lines*	*Description*
ApproachTables.py	726	Generate LaTeX tables for static evaluation of frame comparison functions
Backupumbc.py	100	Tar, ZIP, and copy files from laptop to dated directories on network file server
Dissertation.py	805	Master script to run one or multiple sets of experiments
Results.py	3179	Read experiment CSV files and generate LaTeX tables for results chapter
csv2tex.py	345	Create a LaTeX table from an experiment CSV file
trace2tex.py	149	Convert Rover to MCL communication trace to a LaTeX tabular figure

Table D.3. Dissertation Related Python Programs

File	*Lines*	*Description*
Dissertation.py	150	Executes experiments for dissertation
Experiments.py	158	Executes 22 Chippy experiments with a single runner
Experiment.py	134	Executes a single Chippy experiment
Runners.py	205	Run routines for Metacognition 0, 1, 2, and 3
Runner.py	124	Abstract Q-Learner that invokes metacognition
QLearner.py	213	A grid world walker that learns
Walker.py	168	Agent that walks a grid world
Grid.py	276	A x/y collection of Squares
Square.py	255	A single square of the grid
Constants.py	151	A ".h" file for the Chippy python files
Results.py	1001	Creates latex tables from Chippy experiment csv file

Table D.4. Chippy Demonstration Python Programs

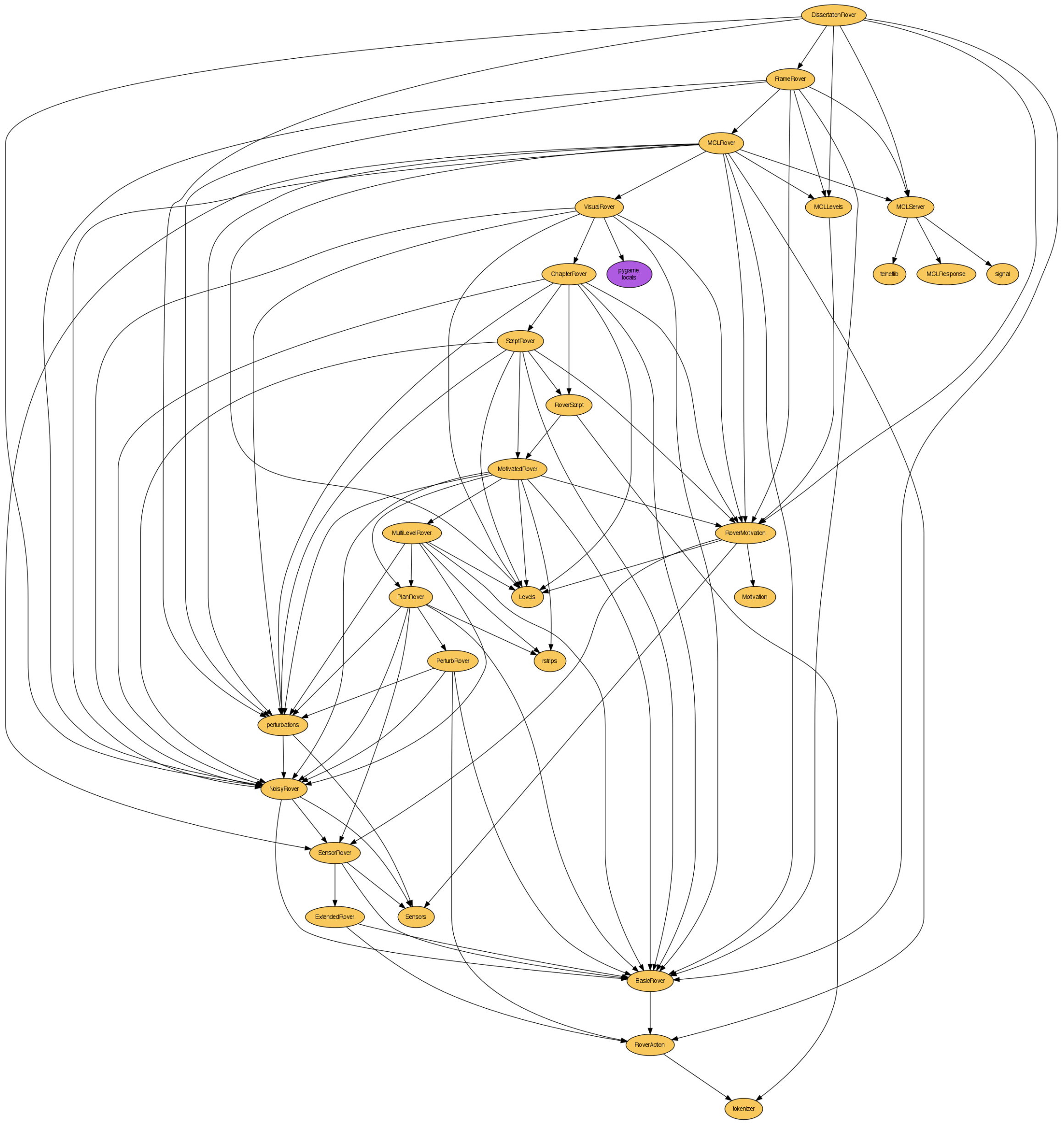


FIG. D.1. Dependency graph for Python implementation of the Mars Rover simulation.

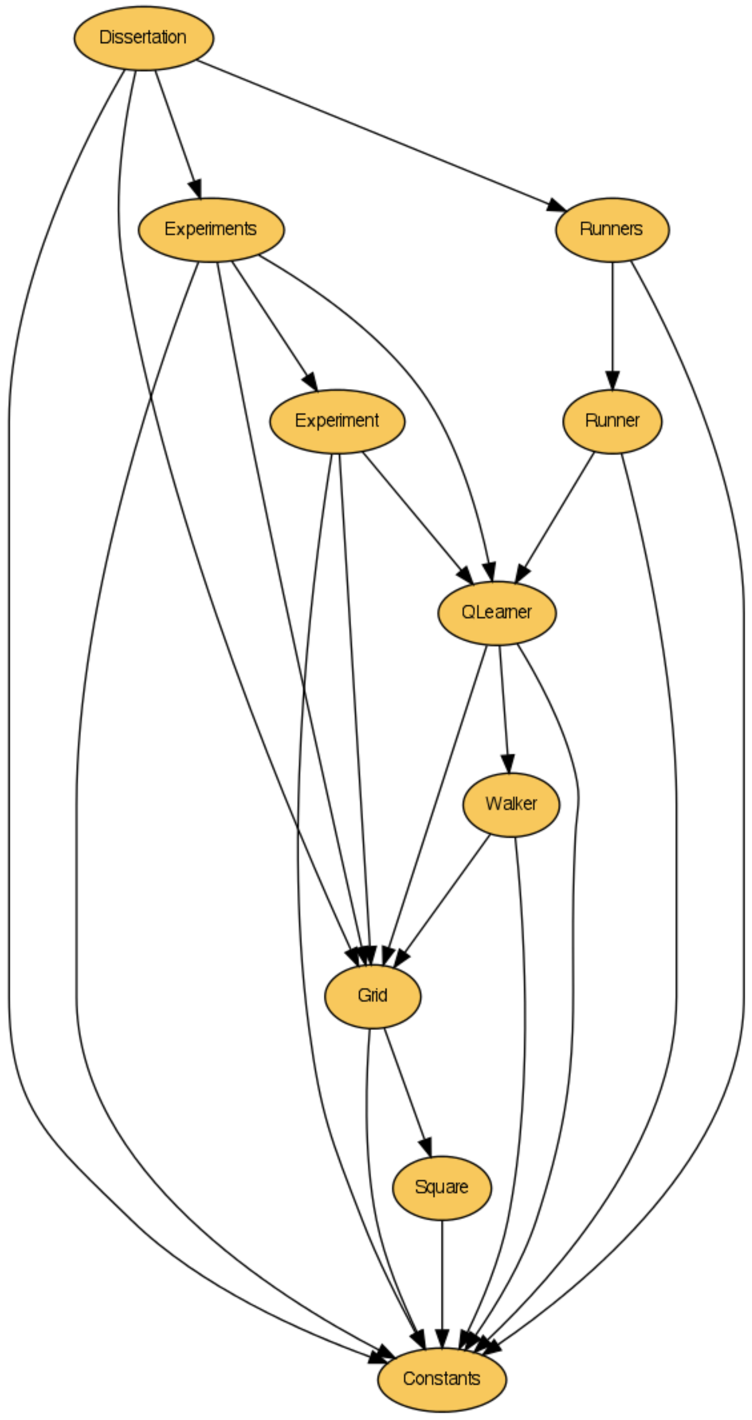


FIG. D.2. Dependency graph for Python implementation of the Chippy Q-Learner with multiple metacognition levels.

REFERENCES

[1] Aamodt, A., and Plaza, E. 1994. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI Communications* 7(1):39–59.

[2] Allen, J. F.; Schubert, L. K.; Ferguson, G.; Heeman, P.; Hwang, C. H.; Kato, T.; Light, M.; Martin, N. G.; Miller, B. W.; Poesio, M.; and Traum, D. R. 1994. The trains project: A case study in defining a conversational planning agent. Technical report, University of Rochester, Rochester, NY, USA.

[3] 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.

[4] Anderson, M. L., and Oates, T. 2007. A review of recent research in metareasoning and metalearning. *AI Magazine* 28(1):7–16.

[5] Anderson, M., and Perlis, D. 2005. Logic, Self-Awareness, and Self-Improvement: The Metacognitive Loop and the Problem of Brittleness. *Journal of Logic and Computation* 15(1):21–40.

[6] Anderson, J., and Pirolli, P. 1984. Spread of activation. *Journal of Experimental Psychology: Learning, Memory, and Cognition* 10(4):791.

[7] Anderson, M. L.; Okamoto, Y. A.; Josyula, D.; and Perlis, D. 2002. The use-mention distinction and its importance to hci. In *TheProceesings of the Sixth Workshop on the Semantics and Pragmatics of Dialog*, 21–28.

[8] Anderson, M. L.; Josyula, D.; Perlis, D.; and Purang, K. 2004. Active logic for more effective human-computer interaction and other commonsense applications. In *Proceedings of the Workshop Empirically Successful First-Order Reasoning, International Joint Conference on Automated Reasoning*.

[9] Anderson, M. L.; Oates, T.; Chong, W.; and Perlis, D. 2006. 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):387–411.

[10] Anderson, M. L.; Schmill, M.; Oates, T.; Perlis, D.; Josyula, D.; Wright, D.; and Wilson, S. 2007. Toward domain-neutral human-level metacognition. In *8th International Symposium on Logical Formalizations of Commonsense Reasoning*, 1–6.

[11] Anderson, M. L.; Fults, S.; Josyula, D. P.; Oates, T.; Perlis, D.; Schmill, M. D.; Wilson, S.; and Wright, D. 2008. A self-help guide for autonomous systems. *AI Magazine* 29(2):67–76.

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

[13] Bento, C., and Costa, E. 1994. A similarity metric for retrieval of cases imperfectly explained. In Wess, S.; Althoff, K.; and Richter, M., eds., *Topics in Case-Based Reasoning*. Berlin: Springer Verlag. 92–105.

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

[15] Brachman, R. J. 2006. (AA)AI, More Than the Sum of its Parts. *AI Magazine* 27(4):AAAI Presidential Address.

[16] Coddington, A. 2006. Motivations for MADbot: a motivated and goal directed robot. In *Proceedings of the 25th Workshop of the UK Planning and Scheduling Special Interest Group (PlanSIG2006)*, 39–46.

[17] Coddington, A. 2007a. Motivations as a meta-level component for constraining goal generation. In *Proceedings of the First International Workshop on Metareasoning in Agent-Based Systems*, 16–30.

[18] Coddington, A. 2007b. Integrating motivations with planning. In *AAMAS '07: Proceedings of the 6th international joint conference on autonomous agents and multi-agent systems*, 855–857. Honolulu, Hawaii: ACM. 978-81-904262-7-5.

[19] Cohen, P., and Kjeldsen, R. 1987. Information retrieval by constrained spreading activation in semantic networks. *Information processing & management* 23(4):255–268.

[20] Cox, M. T., and Raja, A. 2007. Metareasoning: A manifesto. Technical Report BBN TM-2028, BBN Technologies.

[21] Cox, M., and Raja, A. 2011. *Metareasoning: Thinking about thinking*. The MIT Press.

[22] Cox, M. T. 2005a. Perpetual self-aware cognitive agents. In Anderson, M., and Oates, T., eds., *Metacognition in Computation: Papers from 2005 AAAI Spring Symposium*. Menlo Park, CA: AAAI Press. 42–48. Technical Report SS-05-04.

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

[24] Cox, M. 2007. Perpetual self-aware cognitive agents. *AI Magazine* 28(1):32.

[25] Dearden, R.; Wileke, T.; Simmons, R.; Verma, V.; Hunter, F.; and Thrun, S. 2004. Real-time Fault Detection and Situational Awareness for Rovers: Report on the Mars Technology Program Task. In *In Proceedings of IEEE Aerospace Conference*, 826–840. IEEE Press.

[26] Dellarocas, C., and Klein, M. 2000. An experimental evaluation of domain-independent fault handing services in open multi-agent systems. In *Proceedings of the International Conference on Multi-agent Systems (ICMAS)*.

[27] Druzdzel, M. 1999. SMILE: Structural Modeling, Inference, and Learning Engine and GeNIe: a development environment for graphical decision-theoretic models. In *Proceedings of the national conference on artificial intelligence*, 902–903. JOHN WILEY & SONS LTD.

[28] Elgot-Drapkin, J., and Perlis, D. 1990. Reasoning situated in time I: Basic concepts. *Journal of Experimental and Theoretical Artificial Intelligence* 2:75–98.

[29] Elgot-drapkin, J.; Kraus, S.; Miller, M.; Nirkhe, M.; and Perlis, D. 1993. Active logics: A unified formal approach to episodic reasoning. Technical Report UMIACS TR # 99-65m CS-TR # 4072, University of Maryland, UMIACS and CSC.

[30] Elgot-Drapkin, J. J. 1988. *Step-logic: reasoning situated in time*. Ph.D. Dissertation, Department of Computer Science, University of Maryland, College Park, MD.

[31] Estlin, T.; Gaines, D.; Chouinard, C.; Castano, R.; Bornstein, B.; Judd, M.; Nesnas, I.; and Anderson, R. 2007. Increased Mars Rover Autonomy using AI Planning, Scheduling and Execution. In *2007 IEEE International Conference on Robotics and Automation, International Conference on Robotics and Automation*, 4911–4918. Rome, Italy: IEEE.

[32] Ferguson, G.; Allen, J.; and Miller, B. 1996. Trains-95: Towards a mixed-initiative planning assistant. In *Proceedings of the Third Conference on Artificial Intelligence Planning Systems (AIPS-96)*, 70–77.

[33] Fikes, R. E., and Nilsson, N. J. 1971. STRIPS: a new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2:189–208.

[34] Fikes, R., and Nilsson, N. 1994. Strips, a retrospective. *Artificial intelligence in perspective* 227.

[35] Fikes, R. 1971. Monitored execution of robot plans produced by strips. Technical Report Technical Report 55, AI Center, SRI International, Menlo Park, CA.

[36] Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning; Theory and Practice*. San Francisco, CA: Morgan Kaufmann.

[37] Gil, Y. 1994. Learning by experimentation: Incremental refinement of incomplete planning domains. In *Proceedings of the Eleventh International Conference on Machine Learning*, 87–95.

[38] Hägg, S. 2000. A sentinel approach to fault handling in multi-agent systems. In *Proceedings of the 2nd Australian Workshop on Distributed AI*.

[39] Hammond, K. J. 1986. Learning to anticipate and avoid planning problems through the explanation of failures. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, 556–560. Menlo Park, CA: AAAI Press.

[40] Hammond, K. J. 1989. Case-based planning: Viewing planning as a memory task. In *Perspectives in Artificial Intelligence*, volume 1. San Diego, CA: Academic Press.

[41] Hammond, K. J. 1990. Explaining and repairing plans that fail. *Artificial Intelligence* 45:173–228.

[42] Hardy, C. R.; Miller, D. W.; and Hajek, B. K. 1992. A model-based approach to malfunction isolation in interacting systems. In *Proceedings of the 8th Power Plant Control & Testing Symposium*, 37.1–37.13.

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

[44] Hines, J. W., and Hajek, D. W. M. B. K. 1995. Fault detection and isolation: A hybrid approach. In *Proceedings of the Topical Meeting on Computer-Based Human Support Systems: Technology, Methods, and Future, Philadelphia, Pennsylvania, June 25-29, 1995*.

[45] Hines, J. W.; Miller, D. W.; and Hajek, B. K. 1996. Hybrid approach for detecting and isolating faults in nuclear power plant interacting systems. *Nuclear technology* 115(3):342–358.

[46] Hinrichs, T., and Forbus, K. 2007. Analogical learning in a turn-based strategy game. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence*, 853–858.

[47] Hooper, J. 2004. Darpa's debacle in the desert an inside look at what went wrong at the grand challenge. plus: Who's ready for next year's race? *POPULAR SCIENCE* 264(6):84–93.

[48] Jones, J., and Goel, A. 2005. Knowledge organization and structural credit assignment. *IJCAI-05 Workshop on Reasoning, Representation, and Learning in Computer Games*.

[49] Josyula, D.; Anderson, M. L.; and Perlis, D. 2003. Towards domain-independent, task-oriented, conversational adequacy. In *Proceedings of the Eighteenth international Joint Conference on Artificial Intelligence (IJCAI-03)*, 1637–8.

[50] 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.

[51] Judd, C.; McClelland, G.; and Ryan, C. 2009. *Data analysis: A model comparison approach (2nd ed.)*. New York, NY, US: Routledge/Taylor & Francis Group.

[52] Kolodner, J. L. 1993. *Case-Based Reasoning*. San Mateo, CA: Morgan Kaufmann Publishers.

[53] Leake, D.; Kinley, A.; and Wilson, D. 1997. Case-based similarity assessment: Estimating adaptability from experience. In *Proceedings of the national conference on artificial intelligence*, 674–679.

[54] Leake, D. 1992. *Evaluating Explanations: A Contect Theory*. Hillsdale, NJ: Lawrence Erlbaum Associates.

[55] Leake, D. 1995. Adaptive similarity assessment for case-based explanation. *International Journal of Expert Systems* 8(2):165–194.

[56] Leake, D. B. 1996a. Experience, introspection, and expertise: Learning to refine the case-based reasoning process. *Journal of Experimental and Theoretical Artificial Intelligence* 8(3-4):319–339.

[57] Leake, D. B. 1996b. CBR in context: The present and future. In Leake, D. B., ed., *Case-based Reasoning: Experiences, Lessons, and Future Directions*. AAAI Press / The MIT Press. chapter 1, 3–30.

[58] Lekavỳ, M., and Návrat, P. 2007. Expressivity of strips-like and htn-like planning. *Agent and Multi-Agent Systems: Technologies and Applications* 121–130.

[59] Miller, M. 1993. *A view of one's past and other aspects of reasoned change in belief*. Ph.D. Dissertation, University of Maryland, College Park, Maryland.

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

[61] Nau, D.; Au, T.; Ilghami, O.; Kuter, U.; Murdock, J.; Wu, D.; and Yaman, F. 2003. Shop2: An htn planning system. *Journal of Artificial Intelligence Research* 20(1):379–404.

[62] NIST/SEMATECH. 2003. e-handbook of statistical methods. http://www.itl.nist.gov/div898/handbook/prc/section4/prc43.htm, May 2011.

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

[64] Purang, K. 2001. *Systems that Detect and Repair Their Own Mistakes*. Ph.D. Dissertation, Dept. of Computer Science, University of Maryland, College Park, MD.

[65] Rummery, G. A., and Niranjan, M. 1994. On-line q-learning using connectionist systems. Technical Report CUED/F-INGENG/TR 166, Cambridge University Engineering Department.

[66] Russell, S., and Norvig, P. 1995. *Artificial intelligence: a modern approach*. Prentice Hall Englewood Cliffs, NJ.

[67] Russell, S. J., and Wefald, E. 1991. Principles of metareasoning. *Artificial Intelligence* 49(1-3):361–395.

[68] Saka, P. 1998. Quotation and the use-mention distinction. *Mind* 107(425):113.

[69] Schank, R. C., and Leake, D. B. 1989. Creativity and learning in a case-based explainer. *Artificial Intelligence* 40(1-3):353–385.

[70] Schank, R. 1990. *Tell me a story: A new look at real and artificial memory.* Charles Scribner's Sons.

[71] Schmill, M.; Josyula, D.; Anderson, M. L.; Wilson, S.; Oates, T.; Perlis, D.; Wright, D.; and Fults, S. 2007. Ontologies for reasoning about failures in AI systems. In *Workshop on Metareasoning in Agent-Based Systems*.

[72] Schmill, M.; Oates, T.; Anderson, M.; Fults, S.; Josyula, D.; Perlis, D.; and Wilson, S. 2008. The role of metacognition in robust AI systems. In *AAAI-08 Workshop on Metareasoning,(Chicago, IL)*.

[73] Schmill, M.; Anderson, M.; Fults, S.; Josyula, D.; Oates, T.; Perlis, D.; Shahri, H.; Wilson, S.; and Wright, D. 2011. The metacognitive loop and reasoning about anomalies. In Cox, M. T., and Raja, A., eds., *Metareasoning: Thinking about thinking*. The MIT Press. chapter 12, 183–198.

[74] Schmill, M. 2009. Overview of the MCL API. Included with MCL2.

[75] Stephenson, A. 1999. Mars climate orbiter: Mishap investigation board report. *NASA, November* 10.

[76] Stroulia, E., and Goel, A. K. 1995. Functional representation and reasoning for reflective systems. *Journal of Applied Intelligence* Special Issue on Functional Reasoning 9(1):101–124.

[77] Stroulia, E., and Goel, A. K. 1996. A model-based approach to blame assignment: Revising the reasoning steps of problem solvers. In *Thirteenth Annual Conference on Artificial Intelligence*, 959–965. Portland, Oregon: AAAI Press.

[78] Stroulia, E. 1994. *Failure-Driven Learning as Model-Based Self-Redesign*. Ph.D. dissertation, College of Computing, Georgia Institute of Technology, Atlanta.

[79] Sutton, R. S., and Barto, A. G. 1995. *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press.

[80] Tinós, R., and Terra, M. H. 2002. Fault detection and isolation for multiple manipulators. In *the Proceedings of the 2002 IFAC Work Congress (IFAC 2002)*.

[81] Tinós, R.; Terra, M. H.; and Bergerman, M. 2002. Fault tolerance in cooperative manipulators. In *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, volume 1, 470–475. IEEE.

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

[83] Ulam, P.; Goel, A.; Jones, J.; and Murdock, W. 2005. Using model-based reflection to guide reinforcement learning. In *IJCAI Workshop on Reasoning, Representation, and Learning in Computer Games*.

[84] Ulam, P.; Goel, A.; and Jones, J. 2004. Reflection in action: Model-based self-adaptation in game playing agents. In *AAAI Challenges in Game AI Workshop*.

[85] Veloso, M., and Carbonell, J. G. 1994. Case-based reasoning in PRODIGY. In Michalski, R. S., and Tecuci, G., eds., *Machine Learning IV: A Multistrategy Approach*. San Francisco: Morgan Kaufmann. 523–548.

[86] Wang, X. 1995. Learning by observation and practice: An incremental approach for planning operator acquisition. In *Proceedings of the Twelfth International Conference on Machine Learning*.

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

[88] Watkins, C. 1989. *Learning from delayed rewards*. Ph.D. Dissertation, King's College, Cambridge University, Cambridge, England.

[89] Williams, B. C., and Nayak, P. P. 1996. A model-based approach to reactive self-configuring systems. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, 971–978.

[90] Williams, B., and Nyak, P. P. 1999. A model-based approach to reactive self-configuring systems. In Minker, J., ed., *Workshop on Logic-Basid Artificial Intelligence (LBAI)*. College Park, Maryland: Americian Science Foundation and AAAI.

[91] Wilson, T. D., and Schooler, J. W. 1991. Thinking too much: Introspection can reduce the quality of preferences and decisions. *Journal of Personality and Social Psychology* 60(2):181–192.