Goal-Driven Learning in the GILA Integrated Intelligence Architecture

Jainarayan Radhakrishnan and **Santiago Ontañón** and **Ashwin Ram**
CCL, Cognitive Computing Lab
Georgia Institute of Technology
Atlanta, GA 303322/0280
{jai,santi,ashwin}@cc.gatech.edu

Abstract

Goal Driven Learning (GDL) focuses on systems that determine by themselves what has to be learnt and how to learn it. Typically GDL systems use *meta-reasoning* capabilities over a base *reasoner*, identifying learning goals and devising strategies. In this paper we present a novel GDL technique to deal with complex AI systems where the meta-reasoning module has to analyze the reasoning trace of multiple components with potentially different learning paradigms. Our approach works by distributing the generation of learning strategies among the different modules instead of centralizing it in the meta-reasoner. We implemented our technique in the GILA system, that works in the airspace task orders domain, showing an increase in performance.

1 Introduction

Traditional Machine Learning systems are "canned": a human decides the target concept or function that the system needs to learn, the learning algorithm to be used, and provides the training corpus. Goal Driven Learning (GDL) [Ram and Leake, 1995] focuses on how artificial intelligence (AI) systems decide on what is to be learnt and the learning strategies to be used. This kind of reasoning requires *meta-reasoning* (also called *reflexion* or *introspection*) mechanisms. The central idea underlying GDL is that the value of learning depends on how well learning contributes to the learner's goals, and thus the learning process should be guided by those goals.

GDL systems traditionally consist of a *reasoner* or base system that has some *performance goals* to achieve in some domain. When the reasoner executes a performance task, its reasoning process is captured in a *reasoning trace*. When the reasoning results in a failure (where "failure" is defined according to the performance goals of the reasoner), a *meta-reasoner* module analyzes the reasoning trace looking for *reasoning failures*, and generates *learning goals*. For each learning goal, a *learning strategy* is devised (typically using some form of planning over a set of primitive learning or information gathering operators) and then executed, resulting in the solution of learning goals, hopefully improving the satisfaction of the performance goals of the reasoner in the future.

Classic GDL systems (e.g. [Cox and Ram, 1999]) are tailored to specific reasoners. There is a central planner that processes the learning goals and decides "how to learn" in a centralized way, having access to the set of learning operators that the reasoner is able to execute. However, this procedure is problematic when dealing with complex reasoning architectures composed of heterogeneous reasoning components, using potentially different representations and learning strategies. If a central planner was to plan for the achievement of learning goals of each component in a system, then the internals of these modules (at least its knowledge representation and learning capabilities) have to be exposed to the meta reasoner, thus increasing the coupling among components.

In this paper we propose a new GDL approach for complex and heterogeneous AI architectures where the meta-reasoner is decoupled from the base reasoners. Meta-reasoning is a service provided to the base modules, capable of analyzing reasoning traces and generating learning goals. Learning goals are then handed to the base reasoning modules, and each module is responsible for deciding whether to address each learning goal, and how to address it. Thus, planning how to address learning goals is performed in a decentralized fashion by each one of the reasoners.

We have implemented our approach in the GILA (Generalized Integrated Learning Architecture) system. GILA was developed under the Integrated Learning DARPA project by a team composed of 8 universities and 5 private research labs coordinated by Lockheed Martin. In the remainder of this paper we will provide some background in GDL. Then, we present an overview of the GILA system. After that we present our GDL approach focusing on how learning goals are generated and how individual reasoners devise their own learning strategies individually. Finally we present related work and an empirical evaluation of the behavior of GILA with and without our GDL module.

2 Goal Driven Learning

Understanding the processes of learning and providing computers with learning capabilities is one of the main goals of artificial intelligence. However, most of the research in machine learning focuses on the reconstruction of some unknown target function from a collection of examples [Mitchell, 1997]. This is too simple a formulation of the learning problem to model the surprising learning skills ex-

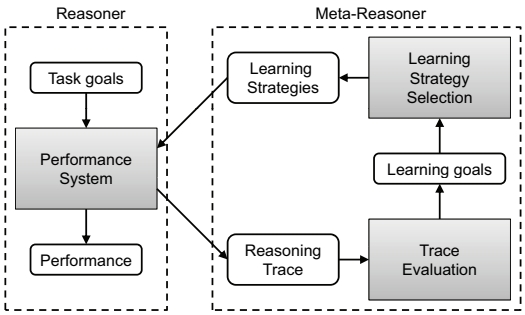

Figure 1: Main components of a GDL system.

hibited by humans. One of the simplifications is that classical machine learning omits any reference to the learner's goals. Humans tend to focus their learning towards those directions relevant to their goals, thus being able to focus and opportunistically exploit the available sources of knowledge.

There are four main questions that a GDL system has to address: whether to learn , what to learn, when to learn, and how to learn. Typically, the meta-reasoner takes charge of all of them. In the approach presented in this paper, the meta-reasoner is in charge of what's to be learnt, and delegates the rest of questions to the individual base reasoners.

Figure 1 shows a typical architecture of a GDL system, composed of both a *reasoner* and a *meta-reasoner*. The reasoner, or base system, has certain tasks or performance goals to achieve in some particular domain. The results of the reasoning process of the reasoner are captured in *reasoning traces*, handed to the meta-reasoner. The meta-reasoner will perform two basic steps: *trace evaluation* and *learning strategy selection*. Trace evaluation consists on identifying *reasoning failures* in the trace, and formulating *learning goals* that address such failures. After a set of learning goals have been identified, the next step involves constructing learning strategies for them. Once found, those strategies are typically handed back to the reasoner, which will execute them in order to address its performance failures. Additionally, some systems (e.g. [Fox, 2000]) allow the meta-reasoner to reflect on itself by also representing the reasoning processes of the meta-reasoner as reasoning traces.

3 GILA

GILA (Generalized Integrated Learning Architecture) is a complex heterogeneous learning system that tries to address the problem of learning a complex task from a very small set of examples (in fact, from a single example) and some limited amount of domain knowledge. In order to achieve this task, GILA contains several *Integrated Learners / Reasoners* (ILRs), each one of them based on a different machine learning paradigm. The ILRs learn as much of knowledge as they can from the small training set, and during performance, an overall *Meta-Reasoning Executive* (MRE) coordinates all of them to exploit their individual strengths.

The domain that GILA operates is in the airspace orders deconfliction domain: given a plan to execute some operations (represented as a set of airspaces associated with missions and air units over space and time), and a set of modification requests, GILA's task is to accommodate the modifications into the plan while taking care of all the conflicts that might arise. For example, if two airplanes have a conflict (they occupy the same space and time in the current plan), GILA has to identify which one has to be modified, and come up with a plan to modify the airspace assigned to that airplane. Moreover, if an airspace is changed, other related airspaces might have to be changed too, since they might be related. Thus, GILA has to make sure that the goals of the mission are still satisfied after the deconfliction process. One important feature of the domain is that there is no annotation provided on the association dependencies between airspaces. GILA has to learn that on its own (also from a single example and some domain knowledge).

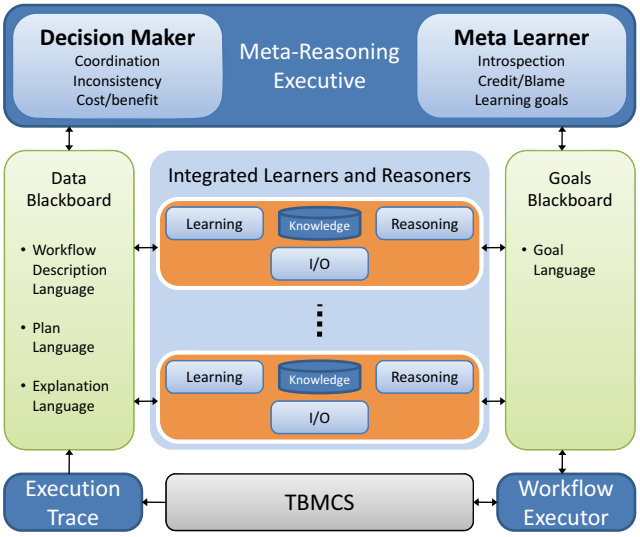

Figure 2: The GILA architecture.

The architecture of GILA can be seen in Figure 2. The ILRs are shown in the center of the architecture. Each ILR is composed of several internal modules: a learner, that can extract knowledge from examples and store it in a knowledge base, a reasoner that can solve problems using that knowledge, and an I/O module that can translate from the internal representation used by the ILR to the common language used in the GILA architecture. On the top of Figure 2 we see the MRE, divided in two modules: the *Decision Maker* (MRE-DM) and the *Meta Learner* (MRE-ML). The MRE-DM is in charge of coordinating the different ILRs during performance (i.e. during problem solving), while the MRE-ML coordinates the learning processes. The MRE and the ILRs communicate through two blackboards: The *Goals Blackboard*, where the MRE posts goals (both performance goals and learning goals) and the *Data Blackboard*, where the ILRs post the solutions found to the goals. Finally, in the lower part of the figure we can see TBMCS (Theater Battle Management Control System), which is the interface to the domain where GILA operates.

A typical execution of GILA works as follows. A single demonstration showing the steps involved to deconflict a par-

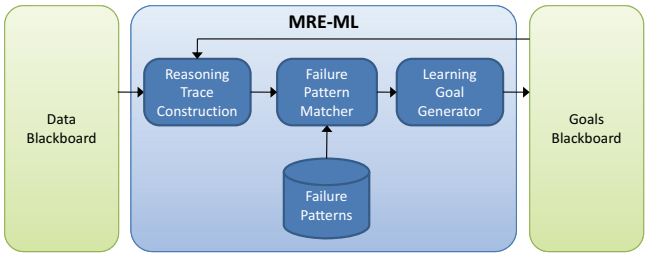

Figure 3: MRE-ML Architecture

ticular problem is provided to GILA (the problem might be composed of multiple airspaces and thus may have multiple conflicts). Each ILR individually learns from the trace. When a new problem arrives, the MRE-DM requests for conflicts to be identified in the current problem. GILA has a specific ILR that knows how to solve this goal, which returns the results to the MRE. Those conflicts define the next set of subgoals that the ILRs will have to solve. But first, the MRE-DM asks the ILRs to provide an ordering on these subgoals, some ILRs respond, and the MRE-DM composes the responses in a final ordering of subgoals. Then, each one of these conflicts is handed one by one to the ILRs. Each ILR that is capable of solving conflicts proposes solutions based on their learned knowledge. The MRE-DM composes each of the individual solutions into a global one, opening a search tree. Some ILRs know how to evaluate the similarity between a solution and a solution obtained from the expert demonstration. The MRE-DM uses this as the search heuristic. Moreover, other ILRs can learn whether some airspace modifications are dangerous (the *Safety Checker* ILR can do this), and thus the MRE-DM also uses feedback from these ILRs as a guide. The MRE-DM keeps exploring the search space until a valid solution is found. This solution is then considered the final solution. In the next section we see how GILA's performance is enhanced thanks to the MRE-ML.

4 Meta Learning in GILA

Figure 3 illustrates the architecture of the MRE-ML, as composed of four modules: a collection of failure patterns, a reasoning trace construction module, a failure pattern matcher, and a learning goal generator. The MRE-ML is constantly monitoring the blackboards to construct the GILA reasoning trace, based on which it searches for *failure patterns* and generates *learning goals*. Let us briefly explain how the MRE-ML is integrated with the GILA system.

GILA's execution has 3 distinguished phases: *learning*, *practice*, and *performance*. During learning, GILA is handed a demonstration and the ILRs learn. During practice, GILA is handed a practice problems (for which the solution is not known), and GILA attempts to solve it, allowing both the ILRs and the MRE to learn during the process. Finally, during performance, learning is disabled, and GILA just attempts to solve a target problem. It is thanks to the practice phase, that the MRE-ML can help GILA.

During practice, the MRE-ML records GILA's reasoning trace. Once the MRE-DM determines practice is over (either after finding a valid solution or after reaching a timeout), the MRE-ML kicks into action and analyzes the trace searching for failures by using *failure patterns*. If it identifies any failure, it generates *learning goals* targeted at preventing the problem that gave rise to the detected failure. The other GILA modules would then try to satisfy these *learning goals* so as to perform better next time.

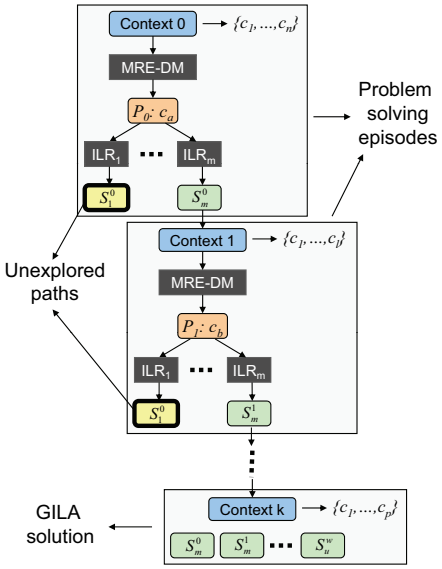

Figure 4: GILA search tree

4.1 Reasoning trace of GILA

The *reasoning trace* is a data structure that is comprised of all the steps the system considers to reach a solution. This involves the information flow amongst the individual GILA components (ILRs and MRE), but not the internals of any of the components. Since GILA solves problems by creating a search tree, the reasoning trace of GILA consists of the process by which such a tree is constructed.

Figure 4 is an illustration of the GILA search tree. The trace is made up of *problem solving episodes*. In each episode we see that there is: a *context* (the current state of the problem state), a *problem* (in the case of GILA this is always a performance goal sent by the MRE-DM to the ILRs), a set of *proposed solutions* by the ILRs, and a *selected solution* (the solution that the MRE-DM decided to expand). When analyzed at a higher level, the search tree of GILA can be seen as the result of a set of modules that produce and consume information. This is the level at which the MRE-ML analyzes data.

Such a search tree is actually never explicitly present anywhere in GILA. The MRE-ML constantly listens to the two blackboards for important events (performance goals, solutions, and selected solutions) and constructs an abstract version of this search tree, which is considered to be the *reasoning trace*. For the MRE-ML, the reasoning trace is a graph consisting of *problem nodes* and *solution nodes*. The *solution nodes* correspond to *proposed solutions* and *selected so-*

lutions. Each node in this graph contains information corresponding to the module that generated the corresponding solution or problem, and the context for the same. In GILA, the problems can be of several kinds: find conflicts, resolve conflicts, prioritize conflicts, assess safety, assess cost, etc. Solutions can also have several types. The failure patterns in the MRE-ML analyze all of this information in the reasoning trace to identify potential problems to be fixed or to identify further scope for learning.

4.2 Failure Patterns

Failure patterns in the MRE-ML consist of small graphs. In the graph that represents a failure pattern, each node can be annotated with some restrictions (e.g. that "a node represents a solution", or that "two solutions nodes must refer to the same solution", etc.). These patterns are matched against the full reasoning trace graph, and if there is any subgraph of the reasoning trace that matches with this pattern, it means that the reasoning trace contains a failure. It is important to note that such a matching has a high computational cost. However, it is performed off-line, and thus it does not affect GILA's performance. The high computational cost is due to the generality of the failure pattern definition language. However, specialized efficient failure detection routines for each pattern could be implemented.

Failure patterns are targeted at identifying potentially faulty areas or modules. For example, one of the defined failure patterns recognizes if an ILR produced an unsafe situation. The way it is detected is as follows: there is an ILR in GILA called the "Safety Checker" that is able to learn constrains and then check wether other ILRs satisfy them. Each time an ILR proposes a deconfliction strategy, the safety checker is asked to produce a safety violation score. If this score is above a threshold, the solution is considered unsafe. This pattern can be defined by defining a graph pattern with two solution nodes, one for a deconfliction strategy and another one for a safety validation score, and specifying the two restrictions: the safety validation score has to correspond to the deconfliction strategy solution and that the safety violation score has to be above a threshold.

In our experiments, the MRE-ML contained several failure patterns, some of them were domain independent, and some of them were specific to GILA:

- *Expert mismatch*: Given a problem, if the solution provided by an ILR does not exactly match the solution provided by the expert for the same problem. The ILR is at fault as it failed to generate the solution that the expert had provided it with, for the same problem.
- *Introduced conflict*: A conflict introduced by an ILR at some earlier point in the graph is also present in the final solution. This happens when an ILR proposes a solution that contains a conflict, and this solution was one of the solutions selected by the MRE-DM, and eventually the final solution also contains the same conflict. In this case, both the ILR and the MRE-DM are at fault.
- *Horizon push*: When provided with a proposed set of solutions for a given problem, the MRE-DM selects a solution whose *confidence level* is lower than a pre-defined threshold. The MRE-DM could have done better in this case in selecting a solution that had a better confidence value, and thus pushing the Horizon.
- *Weakest link*: When provided with a proposed set of solutions for a given problem, the MRE-DM selects a solution whose *confidence value* is lower than a pre-defined threshold. The MRE-DM could have done better in this case in selecting a solution that had a better confidence value. This selected solution, thus forms the weakest link in the solution tree.
- *Backtrack solution*: The MRE-DM selects a solution from the proposed set of solutions that eventually doesn't yield an acceptable Final solution forcing the MRE-DM to retrace its steps and backtrack up the tree searching for another brach to expand.
- *Safety violation*: This failure pattern is triggered when a safety violation constraint was violated by an ILR at some point in the graph.

When any of these failure patterns is triggered, learning goals are generated. These learning goals are to be incorporated by the ILRs and provide a means for the ILRs to learn better. It is not in the scope of the MRE-ML to learn for the ILRs and hence the action must come from the ILRs themselves. Thus, learning goals are addressed within the ILRs. For each match of a failure pattern in the reasoning trace, a learning goal is generated, as explained in next section.

4.3 Learning Goals

We consider three different learning goal types:

- *Solved Problem*: If the MRE-ML has enough information about a failure, it can generate a solved problem example to be sent to the ILRs. This would include information about the failure along with the correct solution for the problem. This could be a learning goal informing about a conflict and the solution to resolve the same, but is not restricted to cases concerning conflicts alone. It could include cases wherein there was a failure detected in the priority of the solutions, or even cases wherein the solution selected works but is not the same as provided by the expert. ILRs should remember these kinds of learning goals and learn from them.
- *Negative Feedback*: This learning goal is generated when the MRE-ML does not have enough information about the failure so as to generate a solved problem example, but knows that something is wrong. In order to address this learning goal, an ILR has to try to avoid generating the same solution in the future.
- *Constrained Problem*: When the MRE-ML does not have information about which is the correct solution for a problem, but at least has some information about it, it can generate a constrained problem. For instance, it can generate a learning goal saying that a proposed solution is incorrect, and that the correct solution should not violate a particular constraint. In order to solve this learning goal, an ILR has to try to find an alternative solution for the problem specified in the learning goal that does not violate the specified constraint.

Each failure pattern has a specific routine to be able to generate one or more of these three kinds of learning goals for each match generated by the failure patterns. Once learning goals are generated, they are sent to the goals blackboard so that any ILR can pick them up. Notice that other GDL systems generate more refined learning goals. However, in the context of GILA, learning goals have to be kept at this level of generality, since each one of the ILRs uses a different learning paradigm, and thus the MRE-ML cannot make any assumption about the kind of concepts they understand.

4.4 Addressing Learning Goals

To better understand how learning goals are addressed, we look at how one of the ILRs handles them, namely the Case-Based Learner/Reasoner (CBLR). The CBLR uses case-based reasoning [Aamodt and Plaza, 1994] to solve the different performance goals that the MRE-DM posts in the goal blackboard. In particular, the CBLR knows how to solve deconfliction problems (given two airspaces that overlap in space or time, it finds a sequence of steps to deconflict them), and prioritization problems (given a list of conflicts, it can sort them by relevance).

To solve those two sets of problems, the CBLR has two knowledge bases: a case-base of deconfliction cases learnt from an expert trace, and a set of constraints also learnt from the expert trace. Deconfliction cases associate particular conflicts with the particular sequence of steps that the expert used to solve them, and also with the order in which the expert solved them (to assess priority). Constraints represent ranges of the different variables in the domain. For instance, if the CBLR observes that the expert never moves, say, a helicopter higher than 50000 feet, and never lower than 100 feet, it will create a constraint for the altitude of helicopters to that range. To solve a deconfliction problem, the CBLR retrieves a set of cases from the case-base and then adapts them using a set of adaptation rules making sure that as many constraints as possible are satisfied.

The CBLR only handles two kinds of learning goals (the rest are ignored): *solved problem* ones and *negative feedback* ones. When the CBLR receives a solved problem learning goal, such a learning goal actually contains all the information required to create a new case. So, the CBLR creates a new case and stores it in its case-base for future reference.

When the CBLR receives a negative feedback learning goal g, it first checks whether the negative feedback applies to itself. The learning goal g specifies that a particular solution S is incorrect for a particular problem P. So, the CBLR tries to solve P to see if the solution obtained would be S. If it is, then the CBLR needs to modify its knowledge so that the solution generated for P is not S. In order to do that, it creates a *negative case*. The CBLR determines the case c from the case-base that has been retrieved to solve P and that led to the solution S. It then creates a negative case saying that for the problem P contained in the learning goal, case c is not the right case to retrieve. Negative cases are used during retrieval, and locally distort the similarity calculations in order to satisfy the learning goals.

Notice that the way the CBLR handles learning goals involves CBR concepts such as "cases", that the MRE-ML does not have to have knowledge about. Other ILRs address learning goals using their own learning formalisms. They might for instance learn rules or modify weights instead of learning new cases.

Table 1: Number of times the CBLR was selected with and without the MRE-ML.

	Without MRE-ML			With MRE-ML		
	CBLR	SPLR	DTLR	CBLR	SPLR	DTLR
ABC	1	7	1	3	4	12
ACB	1	7	2	4	6	0
BAC	2	9	1	2	8	2
BCA	4	5	3	4	7	2
CAB	1	7	2	3	5	2
CBA	0	8	1	3	8	0
	CBLR: 14.51%			CBLR: 26.98%		

5 Empirical Evaluation

Evaluating the quality of a solution provided by GILA is a complex process, that is done by hand by an expert in the domain (since simple comparison with a solution provided by an expert does not suffice). For that reason, in this evaluation we didn't evaluate the overall improvement of GILA due to the MRE-ML (which is still the subject of our future work). In order to evaluate the effect of the MRE-ML, we set up GILA in such a way that only one component (the CBLR) handled learning goals. We evaluated the performance of the CBLR with and without the MRE-ML.

In particular, we measure how many times a deconfliction solution provided by the CBLR is chosen by the MRE-DM. Solutions are chosen by the MRE-DM based on their quality (two ILRs learn respectively how to measure quality of solutions and how safe the solution is). Therefore, if a component's solutions are selected more often, it means that the particular component is providing better solutions. The number of times deconfliction solutions of the CBLR are chosen has to be compared with those of the SPLR (Symbolic-Planner Learner/Reasoner) and the DTLR (Decision-Theoretic Learner/Reasoner). Moreover, in order to perform our experiments we had three different demonstrations (A, B and C) (each demonstration is expensive to obtain), so we tested every possible combination, by using them for learning, practice and performance.

Table 1 shows the obtained results. Each row corresponds to a different experiment, e.g. ABC means that the system learnt in A, practiced in B and performed in C. For each experiment, we ran GILA twice: first without the MRE-ML and then with the MRE-ML. The number of times deconfliction solutions by each of the ILRs were selected was logged. The first thing we see is that when the MRE-ML is present the CBLR's solutions are selected more often, from a 14.51% of the times to a 26.98% of the times. This shows that the CBLR significantly improves the quality of its solutions and thus, they are selected more often.

A closer analysis of the learning goals generated and the improvement of the CBLR reveals that the CBLR's solutions were mostly not considered since their safety violation score

was too high. Thus, most of the learning goals are targeted at this effect. After the CBLR addressed the learning goals, the safety violation score of its solutions reduced drastically, and thus its solutions were selected more often. Moreover, the most often generated learning goal is the "negative example" kind.

6 Related Work

Our work is related to goal-driven learning and introspection. Cox and Ram [Cox and Ram, 1999] argue that in traditional AI systems, the system designer is the one who sequences the available learning algorithms for particular situations. They propose to use introspection to automatically detect reasoning failures using failure patterns. The main difference between Cox and Ram's MetaAQUA system and the technique presented in this paper is that their meta-reasoner is tailored to a particular system, and thus it can plan the sequence of learning operators required to satisfy each of the learning goals. However, in our system, the meta-reasoner only produces learning goals to a collection of individual base reasoners, and they are responsible for planning on how to satisfy the learning goals.

Zang et al. [Zang *et al.*, 2007] propose a similar architecture to MetaAQUA where instead of learning goals, failure patterns cause direct modification of the system's behavior. Their system works on the domain of believable characters, where each character has a certain personality defined. When characters do not behave according to their personality, their stress level increases. When stress goes beyond some threshold, the cause of the failure is determined by trying to match a collection of failure patterns against the execution trace. Each failure pattern has associated with it a collection of modification operators that can modify the behaviors of the characters. The system attempts different modification operators until the stress level of the characters goes down.

RILS [Fox, 2000] is an introspective system for route planning. Each module in the architecture has a set of defined "assertions" that define the expected behavior of the module, when one fails, introspection kicks in. Assertions in RILS can be compared to failure patterns in our technique. RILS contains repair cases, and for each assertion violation, it retrieves a repair case and applies the corresponding repair strategy. In that sense, RILS is more similar to the system presented by Zang et al. [Zang *et al.*, 2007]. Moreover, RILS can also introspect over the introspective reasoning layer.

Introspection has also been used for other tasks than learning, such as explanation generation [Goel and Murdock, 1996], and also for a variety of base reasoners, such as planners [Kuokka, 1991], rule-based systems [Puyol-Gruart, 1995], case-based reasoning systems [Fox, 2000] or even reinforcement learning [Ulam *et al.*, 2008].

7 Conclusions and Future Work

In this paper we have presented an approach for goal driven learning in complex AI architectures, and presented a particular implementation in the GILA architecture. Our approach is based on several key ideas: 1) in heterogeneous architectures the meta-reasoner should just generate learning goals and not construct learning strategies, 2) the planning process needed to generate learning strategies is performed in a distributed way inside of each of the base reasoners, 3) learning goals have to be kept at a high level of description, since the meta-reasoner cannot make any assumptions about the representation formalism of the base reasoners, and 4) failure patterns offer us a uniform case-based method to perform blame assignment.

Currently the MRE-ML module in GILA can only analyze the global reasoning trace of GILA. However, in the future we plan to expand this so that if any component (including the MRE-ML itself) can represent its own reasoning processes as a reasoning trace, this trace can be sent to the MRE-ML, which will offer a generic service of failure detection and learning-goal generation. Also part of our future work is to evaluate which components of GILA (that represent different learning paradigms) benefit more from meta-learning. A final line of future research is to see if failure patterns can be learnt from experience, since currently they are provided by the author to the system.

References

[Aamodt and Plaza, 1994] Agnar Aamodt and Enric Plaza. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *Artificial Intelligence Communications*, 7(1):39–59, 1994.

[Cox and Ram, 1999] Michael T. Cox and Ashwin Ram. Introspective learning: On the construction of learning strategies. *Artificial intelligence*, 1–55(112), 1999.

[Fox, 2000] Susan E. Fox. Reflective intrispective reasoning through CBR. In *Twenty-Second annual meeting of the Cognitive Science Society*, 2000.

[Goel and Murdock, 1996] Ashok K. Goel and J. William Murdock. Meta-cases: Explaining case-based reasoning. In *Third European Workshop on Case-Based Reasoning EWCBR-96*, Lecture Notes in Artificial Intelligence. Springer Verlag, 1996.

[Kuokka, 1991] Daniel R. Kuokka. Max: a meta-reasoning architecture for "x". *SIGART Bull.*, 2(4):93–97, 1991.

[Mitchell, 1997] Tom Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[Puyol-Gruart, 1995] Josep Puyol-Gruart. *MILORD II: A Language for Knowledge–Based Systems*, volume 1 of *Monografies del IIIA*. IIIA–CSIC, 1995.

[Ram and Leake, 1995] Ashwin Ram and David B. Leake. *Goal-Driven Learning*. MIT Press, 1995.

[Ulam *et al.*, 2008] Patrick Ulam, Joshua Jones, and Ashok K. Goel. Combining model-based meta-reasoning and reinforcement learning for adapting game-playing agents. In Christian Darken and Michael Mateas, editors, *AIIDE*. The AAAI Press, 2008.

[Zang *et al.*, 2007] Peng Zang, Manish Mehta, Michael Mateas, and Ashwin Ram. Towards runtime behavior adaptation for embodied characters. In *IJCAI-2007*, pages 1557–1562, 2007.